(old) htmldiff from- (new)

Proprietary Notice

This document is protected by copyright and other related rights and the practice or implementation of the information contained in this document may be protected by one or more patents or pending patent applications. No part of this document may be reproduced in any form by any means without the express prior written permission of Arm. No license, express or implied, by estoppel or otherwise to any intellectual property rights is granted by this document unless specifically stated.

Your access to the information in this document is conditional upon your acceptance that you will not use or permit others to use the information for the purposes of determining whether implementations infringe any third party patents.

THIS DOCUMENT IS PROVIDED “AS IS”. ARM PROVIDES NO REPRESENTATIONS AND NO WARRANTIES, EXPRESS, IMPLIED OR STATUTORY, INCLUDING, WITHOUT LIMITATION, THE IMPLIED WARRANTIES OF MERCHANTABILITY, SATISFACTORY QUALITY, NON-INFRINGEMENT OR FITNESS FOR A PARTICULAR PURPOSE WITH RESPECT TO THE DOCUMENT. For the avoidance of doubt, Arm makes no representation with respect to, and has undertaken no analysis to identify or understand the scope and content of, patents, copyrights, trade secrets, or other rights.

This document may include technical inaccuracies or typographical errors.

TO THE EXTENT NOT PROHIBITED BY LAW, IN NO EVENT WILL ARM BE LIABLE FOR ANY DAMAGES, INCLUDING WITHOUT LIMITATION ANY DIRECT, INDIRECT, SPECIAL, INCIDENTAL, PUNITIVE, OR CONSEQUENTIAL DAMAGES, HOWEVER CAUSED AND REGARDLESS OF THE THEORY OF LIABILITY, ARISING OUT OF ANY USE OF THIS DOCUMENT, EVEN IF ARM HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

This document consists solely of commercial items. You shall be responsible for ensuring that any use, duplication or disclosure of this document complies fully with any relevant export laws and regulations to assure that this document or any portion thereof is not exported, directly or indirectly, in violation of such export laws. Use of the word “partner” in reference to Arm’s customers is not intended to create or refer to any partnership relationship with any other company. Arm may make changes to this document at any time and without notice.

If any of the provisions contained in these terms conflict with any of the provisions of any click through or signed written agreement covering this document with Arm, then the click through or signed written agreement prevails over and supersedes the conflicting provisions of these terms. This document may be translated into other languages for convenience, and you agree that if there is any conflict between the English version of this document and any translation, the terms of the English version of the Agreement shall prevail.

The Arm corporate logo and words marked with ® or ™ are registered trademarks or trademarks of Arm Limited (or its subsidiaries) in the US and/or elsewhere. All rights reserved. Other brands and names mentioned in this document may be the trademarks of their respective owners. Please follow Arm’s trademark usage guidelines at http://www.arm.com/company/policies/trademarks.

Arm Limited. Company 02557590 registered in England.

110 Fulbourn Road, Cambridge, England CB1 9NJ.

LES-PRE-20349

(old) htmldiff from- (new)

AArch64 System Registers

ACTLR_EL1: Auxiliary Control Register (EL1)

ACTLR_EL2: Auxiliary Control Register (EL2)

ACTLR_EL3: Auxiliary Control Register (EL3)

AFSR0_EL1: Auxiliary Fault Status Register 0 (EL1)

AFSR0_EL2: Auxiliary Fault Status Register 0 (EL2)

AFSR0_EL3: Auxiliary Fault Status Register 0 (EL3)

AFSR1_EL1: Auxiliary Fault Status Register 1 (EL1)

AFSR1_EL2: Auxiliary Fault Status Register 1 (EL2)

AFSR1_EL3: Auxiliary Fault Status Register 1 (EL3)

AIDR_EL1: Auxiliary ID Register

AMAIR_EL1: Auxiliary Memory Attribute Indirection Register (EL1)

AMAIR_EL2: Auxiliary Memory Attribute Indirection Register (EL2)

AMAIR_EL3: Auxiliary Memory Attribute Indirection Register (EL3)

AMCFGR_EL0: Activity Monitors Configuration Register

AMCGCR_EL0: Activity Monitors Counter Group Configuration Register

AMCNTENCLR0_EL0: Activity Monitors Count Enable Clear Register 0

AMCNTENCLR1_EL0: Activity Monitors Count Enable Clear Register 1

AMCNTENSET0_EL0: Activity Monitors Count Enable Set Register 0

AMCNTENSET1_EL0: Activity Monitors Count Enable Set Register 1

AMCR_EL0: Activity Monitors Control Register

AMEVCNTR0<n>_EL0: Activity Monitors Event Counter Registers 0

AMEVCNTR1<n>_EL0: Activity Monitors Event Counter Registers 1

AMEVTYPER0<n>_EL0: Activity Monitors Event Type Registers 0

AMEVTYPER1<n>_EL0: Activity Monitors Event Type Registers 1

AMUSERENR_EL0: Activity Monitors User Enable Register

APDAKeyHi_EL1: Pointer Authentication Key A for Data (bits[127:64])

APDAKeyLo_EL1: Pointer Authentication Key A for Data (bits[63:0])

APDBKeyHi_EL1: Pointer Authentication Key B for Data (bits[127:64])

APDBKeyLo_EL1: Pointer Authentication Key B for Data (bits[63:0])

APGAKeyHi_EL1: Pointer Authentication Key A for Code (bits[127:64])

APGAKeyLo_EL1: Pointer Authentication Key A for Code (bits[63:0])

APIAKeyHi_EL1: Pointer Authentication Key A for Instruction (bits[127:64])

APIAKeyLo_EL1: Pointer Authentication Key A for Instruction (bits[63:0])

APIBKeyHi_EL1: Pointer Authentication Key B for Instruction (bits[127:64])

APIBKeyLo_EL1: Pointer Authentication Key B for Instruction (bits[63:0])

CCSIDR2_EL1: Current Cache Size ID Register 2

CCSIDR_EL1: Current Cache Size ID Register

CLIDR_EL1: Cache Level ID Register

CNTFRQ_EL0: Counter-timer Frequency register

CNTHCTL_EL2: Counter-timer Hypervisor Control register

CNTHPS_CTL_EL2: Counter-timer Secure Physical Timer Control register (EL2)

CNTHPS_CVAL_EL2: Counter-timer Secure Physical Timer CompareValue register (EL2)

CNTHPS_TVAL_EL2: Counter-timer Secure Physical Timer TimerValue register (EL2)

CNTHP_CTL_EL2: Counter-timer Hypervisor Physical Timer Control register

CNTHP_CVAL_EL2: Counter-timer Physical Timer CompareValue register (EL2)

CNTHP_TVAL_EL2: Counter-timer Physical Timer TimerValue register (EL2)

CNTHVS_CTL_EL2: Counter-timer Secure Virtual Timer Control register (EL2)

CNTHVS_CVAL_EL2: Counter-timer Secure Virtual Timer CompareValue register (EL2)

CNTHVS_TVAL_EL2: Counter-timer Secure Virtual Timer TimerValue register (EL2)

CNTHV_CTL_EL2: Counter-timer Virtual Timer Control register (EL2)

CNTHV_CVAL_EL2: Counter-timer Virtual Timer CompareValue register (EL2)

CNTHV_TVAL_EL2: Counter-timer Virtual Timer TimerValue Register (EL2)

CNTKCTL_EL1: Counter-timer Kernel Control register

CNTPCT_EL0: Counter-timer Physical Count register

CNTPS_CTL_EL1: Counter-timer Physical Secure Timer Control register

CNTPS_CVAL_EL1: Counter-timer Physical Secure Timer CompareValue register

CNTPS_TVAL_EL1: Counter-timer Physical Secure Timer TimerValue register

CNTP_CTL_EL0: Counter-timer Physical Timer Control register

CNTP_CVAL_EL0: Counter-timer Physical Timer CompareValue register

CNTP_TVAL_EL0: Counter-timer Physical Timer TimerValue register

CNTVCT_EL0: Counter-timer Virtual Count register

CNTVOFF_EL2: Counter-timer Virtual Offset register

CNTV_CTL_EL0: Counter-timer Virtual Timer Control register

CNTV_CVAL_EL0: Counter-timer Virtual Timer CompareValue register

CNTV_TVAL_EL0: Counter-timer Virtual Timer TimerValue register

CONTEXTIDR_EL1: Context ID Register (EL1)

CONTEXTIDR_EL2: Context ID Register (EL2)

CPACR_EL1: Architectural Feature Access Control Register

CPTR_EL2: Architectural Feature Trap Register (EL2)

CPTR_EL3: Architectural Feature Trap Register (EL3)

CSSELR_EL1: Cache Size Selection Register

CTR_EL0: Cache Type Register

CurrentEL: Current Exception Level

DACR32_EL2: Domain Access Control Register

DAIF: Interrupt Mask Bits

DBGAUTHSTATUS_EL1: Debug Authentication Status register

DBGBCR<n>_EL1: Debug Breakpoint Control Registers

DBGBVR<n>_EL1: Debug Breakpoint Value Registers

DBGCLAIMCLR_EL1: Debug Claim Tag Clear register

DBGCLAIMSET_EL1: Debug Claim Tag Set register

DBGDTRRX_EL0: Debug Data Transfer Register, Receive

DBGDTRTX_EL0: Debug Data Transfer Register, Transmit

DBGDTR_EL0: Debug Data Transfer Register, half-duplex

DBGPRCR_EL1: Debug Power Control Register

DBGVCR32_EL2: Debug Vector Catch Register

DBGWCR<n>_EL1: Debug Watchpoint Control Registers

DBGWVR<n>_EL1: Debug Watchpoint Value Registers

DCZID_EL0: Data Cache Zero ID register

DISR_EL1: Deferred Interrupt Status Register

DIT: Data Independent Timing

DLR_EL0: Debug Link Register

DSPSR_EL0: Debug Saved Program Status Register

ELR_EL1: Exception Link Register (EL1)

ELR_EL2: Exception Link Register (EL2)

ELR_EL3: Exception Link Register (EL3)

ERRIDR_EL1: Error Record ID Register

ERRSELR_EL1: Error Record Select Register

ERXADDR_EL1: Selected Error Record Address Register

ERXCTLR_EL1: Selected Error Record Control Register

ERXFR_EL1: Selected Error Record Feature Register

ERXMISC0_EL1: Selected Error Record Miscellaneous Register 0

ERXMISC1_EL1: Selected Error Record Miscellaneous Register 1

ERXMISC2_EL1: Selected Error Record Miscellaneous Register 2

ERXMISC3_EL1: Selected Error Record Miscellaneous Register 3

ERXPFGCDN_EL1: Selected Pseudo-fault Generation Countdown Register

ERXPFGCTL_EL1: Selected Pseudo-fault Generation Control Register

ERXPFGF_EL1: Selected Pseudo-fault Generation Feature Register

ERXSTATUS_EL1: Selected Error Record Primary Status Register

ESR_EL1: Exception Syndrome Register (EL1)

ESR_EL2: Exception Syndrome Register (EL2)

ESR_EL3: Exception Syndrome Register (EL3)

FAR_EL1: Fault Address Register (EL1)

FAR_EL2: Fault Address Register (EL2)

FAR_EL3: Fault Address Register (EL3)

FPCR: Floating-point Control Register

FPEXC32_EL2: Floating-Point Exception Control register

FPSR: Floating-point Status Register

GCR_EL1: Tag Control Register.

GMID_EL1: Multiple tag transfer ID register

HACR_EL2: Hypervisor Auxiliary Control Register

HCR_EL2: Hypervisor Configuration Register

HPFAR_EL2: Hypervisor IPA Fault Address Register

HSTR_EL2: Hypervisor System Trap Register

ICC_AP0R<n>_EL1: Interrupt Controller Active Priorities Group 0 Registers

ICC_AP1R<n>_EL1: Interrupt Controller Active Priorities Group 1 Registers

ICC_ASGI1R_EL1: Interrupt Controller Alias Software Generated Interrupt Group 1 Register

ICC_BPR0_EL1: Interrupt Controller Binary Point Register 0

ICC_BPR1_EL1: Interrupt Controller Binary Point Register 1

ICC_CTLR_EL1: Interrupt Controller Control Register (EL1)

ICC_CTLR_EL3: Interrupt Controller Control Register (EL3)

ICC_DIR_EL1: Interrupt Controller Deactivate Interrupt Register

ICC_EOIR0_EL1: Interrupt Controller End Of Interrupt Register 0

ICC_EOIR1_EL1: Interrupt Controller End Of Interrupt Register 1

ICC_HPPIR0_EL1: Interrupt Controller Highest Priority Pending Interrupt Register 0

ICC_HPPIR1_EL1: Interrupt Controller Highest Priority Pending Interrupt Register 1

ICC_IAR0_EL1: Interrupt Controller Interrupt Acknowledge Register 0

ICC_IAR1_EL1: Interrupt Controller Interrupt Acknowledge Register 1

ICC_IGRPEN0_EL1: Interrupt Controller Interrupt Group 0 Enable register

ICC_IGRPEN1_EL1: Interrupt Controller Interrupt Group 1 Enable register

ICC_IGRPEN1_EL3: Interrupt Controller Interrupt Group 1 Enable register (EL3)

ICC_PMR_EL1: Interrupt Controller Interrupt Priority Mask Register

ICC_RPR_EL1: Interrupt Controller Running Priority Register

ICC_SGI0R_EL1: Interrupt Controller Software Generated Interrupt Group 0 Register

ICC_SGI1R_EL1: Interrupt Controller Software Generated Interrupt Group 1 Register

ICC_SRE_EL1: Interrupt Controller System Register Enable register (EL1)

ICC_SRE_EL2: Interrupt Controller System Register Enable register (EL2)

ICC_SRE_EL3: Interrupt Controller System Register Enable register (EL3)

ICH_AP0R<n>_EL2: Interrupt Controller Hyp Active Priorities Group 0 Registers

ICH_AP1R<n>_EL2: Interrupt Controller Hyp Active Priorities Group 1 Registers

ICH_EISR_EL2: Interrupt Controller End of Interrupt Status Register

ICH_ELRSR_EL2: Interrupt Controller Empty List Register Status Register

ICH_HCR_EL2: Interrupt Controller Hyp Control Register

ICH_LR<n>_EL2: Interrupt Controller List Registers

ICH_MISR_EL2: Interrupt Controller Maintenance Interrupt State Register

ICH_VMCR_EL2: Interrupt Controller Virtual Machine Control Register

ICH_VTR_EL2: Interrupt Controller VGIC Type Register

ICV_AP0R<n>_EL1: Interrupt Controller Virtual Active Priorities Group 0 Registers

ICV_AP1R<n>_EL1: Interrupt Controller Virtual Active Priorities Group 1 Registers

ICV_BPR0_EL1: Interrupt Controller Virtual Binary Point Register 0

ICV_BPR1_EL1: Interrupt Controller Virtual Binary Point Register 1

ICV_CTLR_EL1: Interrupt Controller Virtual Control Register

ICV_DIR_EL1: Interrupt Controller Deactivate Virtual Interrupt Register

ICV_EOIR0_EL1: Interrupt Controller Virtual End Of Interrupt Register 0

ICV_EOIR1_EL1: Interrupt Controller Virtual End Of Interrupt Register 1

ICV_HPPIR0_EL1: Interrupt Controller Virtual Highest Priority Pending Interrupt Register 0

ICV_HPPIR1_EL1: Interrupt Controller Virtual Highest Priority Pending Interrupt Register 1

ICV_IAR0_EL1: Interrupt Controller Virtual Interrupt Acknowledge Register 0

ICV_IAR1_EL1: Interrupt Controller Virtual Interrupt Acknowledge Register 1

ICV_IGRPEN0_EL1: Interrupt Controller Virtual Interrupt Group 0 Enable register

ICV_IGRPEN1_EL1: Interrupt Controller Virtual Interrupt Group 1 Enable register

ICV_PMR_EL1: Interrupt Controller Virtual Interrupt Priority Mask Register

ICV_RPR_EL1: Interrupt Controller Virtual Running Priority Register

ID_AA64AFR0_EL1: AArch64 Auxiliary Feature Register 0

ID_AA64AFR1_EL1: AArch64 Auxiliary Feature Register 1

ID_AA64DFR0_EL1: AArch64 Debug Feature Register 0

ID_AA64DFR1_EL1: AArch64 Debug Feature Register 1

ID_AA64ISAR0_EL1: AArch64 Instruction Set Attribute Register 0

ID_AA64ISAR1_EL1: AArch64 Instruction Set Attribute Register 1

ID_AA64MMFR0_EL1: AArch64 Memory Model Feature Register 0

ID_AA64MMFR1_EL1: AArch64 Memory Model Feature Register 1

ID_AA64MMFR2_EL1: AArch64 Memory Model Feature Register 2

ID_AA64PFR0_EL1: AArch64 Processor Feature Register 0

ID_AA64PFR1_EL1: AArch64 Processor Feature Register 1

ID_AA64ZFR0_EL1: SVE Feature ID register 0

ID_AFR0_EL1: AArch32 Auxiliary Feature Register 0

ID_DFR0_EL1: AArch32 Debug Feature Register 0

ID_ISAR0_EL1: AArch32 Instruction Set Attribute Register 0

ID_ISAR1_EL1: AArch32 Instruction Set Attribute Register 1

ID_ISAR2_EL1: AArch32 Instruction Set Attribute Register 2

ID_ISAR3_EL1: AArch32 Instruction Set Attribute Register 3

ID_ISAR4_EL1: AArch32 Instruction Set Attribute Register 4

ID_ISAR5_EL1: AArch32 Instruction Set Attribute Register 5

ID_ISAR6_EL1: AArch32 Instruction Set Attribute Register 6

ID_MMFR0_EL1: AArch32 Memory Model Feature Register 0

ID_MMFR1_EL1: AArch32 Memory Model Feature Register 1

ID_MMFR2_EL1: AArch32 Memory Model Feature Register 2

ID_MMFR3_EL1: AArch32 Memory Model Feature Register 3

ID_MMFR4_EL1: AArch32 Memory Model Feature Register 4

ID_PFR0_EL1: AArch32 Processor Feature Register 0

ID_PFR1_EL1: AArch32 Processor Feature Register 1

ID_PFR2_EL1: AArch32 Processor Feature Register 2

IFSR32_EL2: Instruction Fault Status Register (EL2)

ISR_EL1: Interrupt Status Register

LORC_EL1: LORegion Control (EL1)

LOREA_EL1: LORegion End Address (EL1)

LORID_EL1: LORegionID (EL1)

LORN_EL1: LORegion Number (EL1)

LORSA_EL1: LORegion Start Address (EL1)

MAIR_EL1: Memory Attribute Indirection Register (EL1)

MAIR_EL2: Memory Attribute Indirection Register (EL2)

MAIR_EL3: Memory Attribute Indirection Register (EL3)

MDCCINT_EL1: Monitor DCC Interrupt Enable Register

MDCCSR_EL0: Monitor DCC Status Register

MDCR_EL2: Monitor Debug Configuration Register (EL2)

MDCR_EL3: Monitor Debug Configuration Register (EL3)

MDRAR_EL1: Monitor Debug ROM Address Register

MDSCR_EL1: Monitor Debug System Control Register

MIDR_EL1: Main ID Register

MPAM0_EL1: MPAM0 Register (EL1)

MPAM1_EL1: MPAM1 Register (EL1)

MPAM2_EL2: MPAM2 Register (EL2)

MPAM3_EL3: MPAM3 Register (EL3)

MPAMHCR_EL2: MPAM Hypervisor Control Register (EL2)

MPAMIDR_EL1: MPAM ID Register (EL1)

MPAMVPM0_EL2: MPAM Virtual PARTID Mapping Register 0

MPAMVPM1_EL2: MPAM Virtual PARTID Mapping Register 1

MPAMVPM2_EL2: MPAM Virtual PARTID Mapping Register 2

MPAMVPM3_EL2: MPAM Virtual PARTID Mapping Register 3

MPAMVPM4_EL2: MPAM Virtual PARTID Mapping Register 4

MPAMVPM5_EL2: MPAM Virtual PARTID Mapping Register 5

MPAMVPM6_EL2: MPAM Virtual PARTID Mapping Register 6

MPAMVPM7_EL2: MPAM Virtual PARTID Mapping Register 7

MPAMVPMV_EL2: MPAM Virtual Partition Mapping Valid Register

MPIDR_EL1: Multiprocessor Affinity Register

MVFR0_EL1: AArch32 Media and VFP Feature Register 0

MVFR1_EL1: AArch32 Media and VFP Feature Register 1

MVFR2_EL1: AArch32 Media and VFP Feature Register 2

NZCV: Condition Flags

OSDLR_EL1: OS Double Lock Register

OSDTRRX_EL1: OS Lock Data Transfer Register, Receive

OSDTRTX_EL1: OS Lock Data Transfer Register, Transmit

OSECCR_EL1: OS Lock Exception Catch Control Register

OSLAR_EL1: OS Lock Access Register

OSLSR_EL1: OS Lock Status Register

PAN: Privileged Access Never

PAR_EL1: Physical Address Register

PMBIDR_EL1: Profiling Buffer ID Register

PMBLIMITR_EL1: Profiling Buffer Limit Address Register

PMBPTR_EL1: Profiling Buffer Write Pointer Register

PMBSR_EL1: Profiling Buffer Status/syndrome Register

PMCCFILTR_EL0: Performance Monitors Cycle Count Filter Register

PMCCNTR_EL0: Performance Monitors Cycle Count Register

PMCEID0_EL0: Performance Monitors Common Event Identification register 0

PMCEID1_EL0: Performance Monitors Common Event Identification register 1

PMCNTENCLR_EL0: Performance Monitors Count Enable Clear register

PMCNTENSET_EL0: Performance Monitors Count Enable Set register

PMCR_EL0: Performance Monitors Control Register

PMEVCNTR<n>_EL0: Performance Monitors Event Count Registers

PMEVTYPER<n>_EL0: Performance Monitors Event Type Registers

PMINTENCLR_EL1: Performance Monitors Interrupt Enable Clear register

PMINTENSET_EL1: Performance Monitors Interrupt Enable Set register

PMMIR_EL1: Performance Monitors Machine Identification Register

PMOVSCLR_EL0: Performance Monitors Overflow Flag Status Clear Register

PMOVSSET_EL0: Performance Monitors Overflow Flag Status Set register

PMSCR_EL1: Statistical Profiling Control Register (EL1)

PMSCR_EL2: Statistical Profiling Control Register (EL2)

PMSELR_EL0: Performance Monitors Event Counter Selection Register

PMSEVFR_EL1: Sampling Event Filter Register

PMSFCR_EL1: Sampling Filter Control Register

PMSICR_EL1: Sampling Interval Counter Register

PMSIDR_EL1: Sampling Profiling ID Register

PMSIRR_EL1: Sampling Interval Reload Register

PMSLATFR_EL1: Sampling Latency Filter Register

PMSWINC_EL0: Performance Monitors Software Increment register

PMUSERENR_EL0: Performance Monitors User Enable Register

PMXEVCNTR_EL0: Performance Monitors Selected Event Count Register

PMXEVTYPER_EL0: Performance Monitors Selected Event Type Register

REVIDR_EL1: Revision ID Register

RGSR_EL1: Random Allocation Tag Seed Register.

RMR_EL1: Reset Management Register (EL1)

RMR_EL2: Reset Management Register (EL2)

RMR_EL3: Reset Management Register (EL3)

RNDR: Random Number

RNDRRS: Reseeded Random Number

RVBAR_EL1: Reset Vector Base Address Register (if EL2 and EL3 not implemented)

RVBAR_EL2: Reset Vector Base Address Register (if EL3 not implemented)

RVBAR_EL3: Reset Vector Base Address Register (if EL3 implemented)

S3_<op1>_<Cn>_<Cm>_<op2>: IMPLEMENTATION DEFINED registers

SCR_EL3: Secure Configuration Register

SCTLR_EL1: System Control Register (EL1)

SCTLR_EL2: System Control Register (EL2)

SCTLR_EL3: System Control Register (EL3)

SCXTNUM_EL0: EL0 Read/Write Software Context Number

SCXTNUM_EL1: EL1 Read/Write Software Context Number

SCXTNUM_EL2: EL2 Read/Write Software Context Number

SCXTNUM_EL3: EL3 Read/Write Software Context Number

SDER32_EL2: AArch32 Secure Debug Enable Register

SDER32_EL3: AArch32 Secure Debug Enable Register

SPSR_EL1: Saved Program Status Register (EL1)

SPSR_EL2: Saved Program Status Register (EL2)

SPSR_EL3: Saved Program Status Register (EL3)

SPSR_abt: Saved Program Status Register (Abort mode)

SPSR_fiq: Saved Program Status Register (FIQ mode)

SPSR_irq: Saved Program Status Register (IRQ mode)

SPSR_und: Saved Program Status Register (Undefined mode)

SPSel: Stack Pointer Select

SP_EL0: Stack Pointer (EL0)

SP_EL1: Stack Pointer (EL1)

SP_EL2: Stack Pointer (EL2)

SP_EL3: Stack Pointer (EL3)

SSBS: Speculative Store Bypass Safe

TCO: Tag Check Override

TCR_EL1: Translation Control Register (EL1)

TCR_EL2: Translation Control Register (EL2)

TCR_EL3: Translation Control Register (EL3)

TFSRE0_EL1: Tag Fail Status Register (EL0).

TFSR_EL1: Tag Fail Status Register (EL1).

TFSR_EL2: Tag Fail Status Register (EL2).

TFSR_EL3: Tag Fail Status Register (EL3).

TPIDRRO_EL0: EL0 Read-Only Software Thread ID Register

TPIDR_EL0: EL0 Read/Write Software Thread ID Register

TPIDR_EL1: EL1 Software Thread ID Register

TPIDR_EL2: EL2 Software Thread ID Register

TPIDR_EL3: EL3 Software Thread ID Register

TRFCR_EL1: Trace Filter Control Register (EL1)

TRFCR_EL2: Trace Filter Control Register (EL2)

TTBR0_EL1: Translation Table Base Register 0 (EL1)

TTBR0_EL2: Translation Table Base Register 0 (EL2)

TTBR0_EL3: Translation Table Base Register 0 (EL3)

TTBR1_EL1: Translation Table Base Register 1 (EL1)

TTBR1_EL2: Translation Table Base Register 1 (EL2)

UAO: User Access Override

VBAR_EL1: Vector Base Address Register (EL1)

VBAR_EL2: Vector Base Address Register (EL2)

VBAR_EL3: Vector Base Address Register (EL3)

VDISR_EL2: Virtual Deferred Interrupt Status Register

VMPIDR_EL2: Virtualization Multiprocessor ID Register

VNCR_EL2: Virtual Nested Control Register

VPIDR_EL2: Virtualization Processor ID Register

VSESR_EL2: Virtual SErrorDeferred ExceptionInterrupt SyndromeStatus Register

VSTCR_EL2: Virtualization Secure Translation Control Register

VSTTBR_EL2: Virtualization Secure Translation Table Base Register

VTCR_EL2: Virtualization Translation Control Register

VTTBR_EL2: Virtualization Translation Table Base Register

ZCR_EL1: SVE Control Register for EL1

ZCR_EL2: SVE Control Register for EL2

ZCR_EL3: SVE Control Register for EL3

2713/0312/20192018 2116:5943

AArch64 System Instructions

AT S12E0R: Address Translate Stages 1 and 2 EL0 Read

AT S12E0W: Address Translate Stages 1 and 2 EL0 Write

AT S12E1R: Address Translate Stages 1 and 2 EL1 Read

AT S12E1W: Address Translate Stages 1 and 2 EL1 Write

AT S1E0R: Address Translate Stage 1 EL0 Read

AT S1E0W: Address Translate Stage 1 EL0 Write

AT S1E1R: Address Translate Stage 1 EL1 Read

AT S1E1RP: Address Translate Stage 1 EL1 Read PAN

AT S1E1W: Address Translate Stage 1 EL1 Write

AT S1E1WP: Address Translate Stage 1 EL1 Write PAN

AT S1E2R: Address Translate Stage 1 EL2 Read

AT S1E2W: Address Translate Stage 1 EL2 Write

AT S1E3R: Address Translate Stage 1 EL3 Read

AT S1E3W: Address Translate Stage 1 EL3 Write

CFP RCTX: Control Flow Prediction Restriction by Context

CPP RCTX: Cache Prefetch Prediction Restriction by Context

DC CGDSW: Data, Allocation Tag or unified Cache line Clean of Data and Allocation Tags by Set/Way

DC CGDVAC: Data, Allocation Tag or unified Cache line Clean of Allocation Tags by VA to PoC

DC CGDVADP: Data, Allocation Tag or unified Cache line Clean of Allocation Tags by VA to PoDP

DC CGDVAP: Data, Allocation Tag or unified Cache line Clean of Data and Allocation Tags by VA to PoP

DC CGSW: Data, Allocation Tag or unified Cache line Clean of Allocation Tags by Set/Way

DC CGVAC: Data, Allocation Tag or unified Cache line Clean of Allocation Tags by VA to PoC

DC CGVADP: Data, Allocation Tag or unified Cache line Clean of Data and Allocation Tags by VA to PoDP

DC CGVAP: Data, Allocation Tag or unified Cache line Clean of Allocation Tags by VA to PoP

DC CIGDSW: Data, Allocation Tag or unified Cache line Clean and Invalidate of Data and Allocation Tags by Set/Way

DC CIGDVAC: Data, Allocation Tag or unified Cache line Clean and Invalidate of Data and Allocation Tags by VA to PoC

DC CIGSW: Data, Allocation Tag or unified Cache line Clean and Invalidate of Allocation Tags by Set/Way

DC CIGVAC: Data, Allocation Tag or unified Cache line Clean and Invalidate of Allocation Tags by VA to PoC

DC CISW: Data or unified Cache line Clean and Invalidate by Set/Way

DC CIVAC: Data or unified Cache line Clean and Invalidate by VA to PoC

DC CSW: Data or unified Cache line Clean by Set/Way

DC CVAC: Data or unified Cache line Clean by VA to PoC

DC CVADP: Data or unified Cache line Clean by VA to PoDP

DC CVAP: Data or unified Cache line Clean by VA to PoP

DC CVAU: Data or unified Cache line Clean by VA to PoU

DC GVA: Data Cache set Allocation Tag by VA

DC GZVA: Data Cache set Allocation Tags and Zero by VA

DC IGDSW: Data, Allocation Tag or unified Cache line Invalidate of Data and Allocation Tags by Set/Way

DC IGDVAC: Data, Allocation Tag or unified Cache line Invalidate of Allocation Tags by VA to PoC

DC IGSW: Data, Allocation Tag or unified Cache line Invalidate of Allocation Tags by Set/Way

DC IGVAC: Data, Allocation Tag or unified Cache line Invalidate of Allocation Tags by VA to PoC

DC ISW: Data or unified Cache line Invalidate by Set/Way

DC IVAC: Data or unified Cache line Invalidate by VA to PoC

DC ZVA: Data Cache Zero by VA

DVP RCTX: Data Value Prediction Restriction by Context

IC IALLU: Instruction Cache Invalidate All to PoU

IC IALLUIS: Instruction Cache Invalidate All to PoU, Inner Shareable

IC IVAU: Instruction Cache line Invalidate by VA to PoU

S1_<op1>_<Cn>_<Cm>_<op2>: IMPLEMENTATION DEFINED maintenance instructions

TLBI ALLE1: TLB Invalidate All, EL1

TLBI ALLE1IS: TLB Invalidate All, EL1, Inner Shareable

TLBI ALLE1OS: TLB Invalidate All, EL1, Outer Shareable

TLBI ALLE2: TLB Invalidate All, EL2

TLBI ALLE2IS: TLB Invalidate All, EL2, Inner Shareable

TLBI ALLE2OS: TLB Invalidate All, EL2, Outer Shareable

TLBI ALLE3: TLB Invalidate All, EL3

TLBI ALLE3IS: TLB Invalidate All, EL3, Inner Shareable

TLBI ALLE3OS: TLB Invalidate All, EL3, Outer Shareable

TLBI ASIDE1: TLB Invalidate by ASID, EL1

TLBI ASIDE1IS: TLB Invalidate by ASID, EL1, Inner Shareable

TLBI ASIDE1OS: TLB Invalidate by ASID, EL1, Outer Shareable

TLBI IPAS2E1: TLB Invalidate by Intermediate Physical Address, Stage 2, EL1

TLBI IPAS2E1IS: TLB Invalidate by Intermediate Physical Address, Stage 2, EL1, Inner Shareable

TLBI IPAS2E1OS: TLB Invalidate by Intermediate Physical Address, Stage 2, EL1, Outer Shareable

TLBI IPAS2LE1: TLB Invalidate by Intermediate Physical Address, Stage 2, Last level, EL1

TLBI IPAS2LE1IS: TLB Invalidate by Intermediate Physical Address, Stage 2, Last level, EL1, Inner Shareable

TLBI IPAS2LE1OS: TLB Invalidate by Intermediate Physical Address, Stage 2, Last level, EL1, Outer Shareable

TLBI RIPAS2E1: TLB Range Invalidate by Intermediate Physical Address, Stage 2, EL1

TLBI RIPAS2E1IS: TLB Range Invalidate by Intermediate Physical Address, Stage 2, EL1, Inner Shareable

TLBI RIPAS2E1OS: TLB Range Invalidate by Intermediate Physical Address, Stage 2, EL1, Outer Shareable

TLBI RIPAS2LE1: TLB Range Invalidate by Intermediate Physical Address, Stage 2, Last level, EL1

TLBI RIPAS2LE1IS: TLB Range Invalidate by Intermediate Physical Address, Stage 2, Last level, EL1, Inner Shareable

TLBI RIPAS2LE1OS: TLB Range Invalidate by Intermediate Physical Address, Stage 2, Last level, EL1, Outer Shareable

TLBI RVAAE1: TLB Range Invalidate by VA, All ASID, EL1

TLBI RVAAE1IS: TLB Range Invalidate by VA, All ASID, EL1, Inner Shareable

TLBI RVAAE1OS: TLB Range Invalidate by VA, All ASID, EL1, Outer Shareable

TLBI RVAALE1: TLB Range Invalidate by VA, All ASID, Last level, EL1

TLBI RVAALE1IS: TLB Range Invalidate by VA, All ASID, Last Level, EL1, Inner Shareable

TLBI RVAALE1OS: TLB Range Invalidate by VA, All ASID, Last Level, EL1, Outer Shareable

TLBI RVAE1: TLB Range Invalidate by VA, EL1

TLBI RVAE1IS: TLB Range Invalidate by VA, EL1, Inner Shareable

TLBI RVAE1OS: TLB Range Invalidate by VA, EL1, Outer Shareable

TLBI RVAE2: TLB Range Invalidate by VA, EL2

TLBI RVAE2IS: TLB Range Invalidate by VA, EL2, Inner Shareable

TLBI RVAE2OS: TLB Range Invalidate by VA, EL2, Outer Shareable

TLBI RVAE3: TLB Range Invalidate by VA, EL3

TLBI RVAE3IS: TLB Range Invalidate by VA, EL3, Inner Shareable

TLBI RVAE3OS: TLB Range Invalidate by VA, EL3, Outer Shareable

TLBI RVALE1: TLB Range Invalidate by VA, Last level, EL1

TLBI RVALE1IS: TLB Range Invalidate by VA, Last level, EL1, Inner Shareable

TLBI RVALE1OS: TLB Range Invalidate by VA, Last level, EL1, Outer Shareable

TLBI RVALE2: TLB Range Invalidate by VA, Last level, EL2

TLBI RVALE2IS: TLB Range Invalidate by VA, Last level, EL2, Inner Shareable

TLBI RVALE2OS: TLB Range Invalidate by VA, Last level, EL2, Outer Shareable

TLBI RVALE3: TLB Range Invalidate by VA, Last level, EL3

TLBI RVALE3IS: TLB Range Invalidate by VA, Last level, EL3, Inner Shareable

TLBI RVALE3OS: TLB Range Invalidate by VA, Last level, EL3, Outer Shareable

TLBI VAAE1: TLB Invalidate by VA, All ASID, EL1

TLBI VAAE1IS: TLB Invalidate by VA, All ASID, EL1, Inner Shareable

TLBI VAAE1OS: TLB Invalidate by VA, All ASID, EL1, Outer Shareable

TLBI VAALE1: TLB Invalidate by VA, All ASID, Last level, EL1

TLBI VAALE1IS: TLB Invalidate by VA, All ASID, Last Level, EL1, Inner Shareable

TLBI VAALE1OS: TLB Invalidate by VA, All ASID, Last Level, EL1, Outer Shareable

TLBI VAE1: TLB Invalidate by VA, EL1

TLBI VAE1IS: TLB Invalidate by VA, EL1, Inner Shareable

TLBI VAE1OS: TLB Invalidate by VA, EL1, Outer Shareable

TLBI VAE2: TLB Invalidate by VA, EL2

TLBI VAE2IS: TLB Invalidate by VA, EL2, Inner Shareable

TLBI VAE2OS: TLB Invalidate by VA, EL2, Outer Shareable

TLBI VAE3: TLB Invalidate by VA, EL3

TLBI VAE3IS: TLB Invalidate by VA, EL3, Inner Shareable

TLBI VAE3OS: TLB Invalidate by VA, EL3, Outer Shareable

TLBI VALE1: TLB Invalidate by VA, Last level, EL1

TLBI VALE1IS: TLB Invalidate by VA, Last level, EL1, Inner Shareable

TLBI VALE1OS: TLB Invalidate by VA, Last level, EL1, Outer Shareable

TLBI VALE2: TLB Invalidate by VA, Last level, EL2

TLBI VALE2IS: TLB Invalidate by VA, Last level, EL2, Inner Shareable

TLBI VALE2OS: TLB Invalidate by VA, Last level, EL2, Outer Shareable

TLBI VALE3: TLB Invalidate by VA, Last level, EL3

TLBI VALE3IS: TLB Invalidate by VA, Last level, EL3, Inner Shareable

TLBI VALE3OS: TLB Invalidate by VA, Last level, EL3, Outer Shareable

TLBI VMALLE1: TLB Invalidate by VMID, All at stage 1, EL1

TLBI VMALLE1IS: TLB Invalidate by VMID, All at stage 1, EL1, Inner Shareable

TLBI VMALLE1OS: TLB Invalidate by VMID, All at stage 1, EL1, Outer Shareable

TLBI VMALLS12E1: TLB Invalidate by VMID, All at Stage 1 and 2, EL1

TLBI VMALLS12E1IS: TLB Invalidate by VMID, All at Stage 1 and 2, EL1, Inner Shareable

TLBI VMALLS12E1OS: TLB Invalidate by VMID, All at Stage 1 and 2, EL1, Outer Shareable

27~~13~~/03~~12~~/2019~~2018~~ 21~~16~~:59~~43~~

(old) htmldiff from- (new)

AFSR0_EL1, Auxiliary Fault Status Register 0 (EL1)

The AFSR0_EL1 characteristics are:

Purpose

Provides additional IMPLEMENTATION DEFINED fault status information for exceptions taken to EL1.

Configuration

AArch64 System register AFSR0_EL1 bits [31:0] are architecturally mapped to AArch32 System register ADFSR[31:0] .

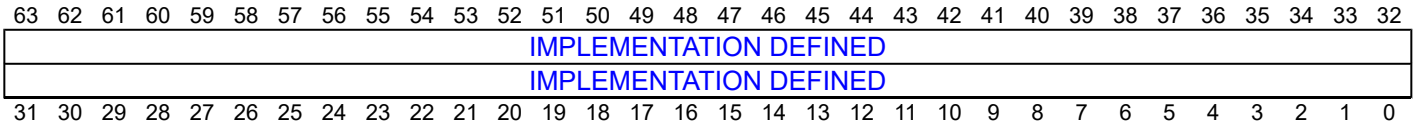
RW fields in this register reset to architecturally UNKNOWN values.

Attributes

AFSR0_EL1 is a 64-bit register.

Field descriptions

The AFSR0_EL1 bit assignments are:



IMPLEMENTATION DEFINED, bits [63:0]

IMPLEMENTATION DEFINED.

This field resets to an architecturally UNKNOWN value.

Accessing the AFSR0_EL1

When HCR_EL2.E2H is 1, without explicit synchronization, access from EL3 using the mnemonic AFSR0_EL1 or AFSR0_EL12 are not guaranteed to be ordered with respect to accesses using the other mnemonic.

Accesses to this register use the following encodings:

```
MRS <Xt>, AFSR0_EL1
```

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b11	0b000	0b0101	0b0001	0b000
~~0b11~~	~~0b0101~~	~~0b000~~	~~0b000~~	~~0b0001~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TRVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<NV2,NV1,NV> == '111' then
        return NVMem[0x128];
    else
        return AFSR0_EL1;
elsif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' then
        return AFSR0_EL2;
    else
        return AFSR0_EL1;
elsif PSTATE.EL == EL3 then
    return AFSR0_EL1;
```

MSR AFSR0_EL1, <Xt>

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b11	0b000	0b0101	0b0001	0b000
~~0b11~~	~~0b0101~~	~~0b000~~	~~0b000~~	~~0b0001~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<NV2,NV1,NV> == '111' then
        NVMem[0x128] = X[t];
    else
        AFSR0_EL1 = X[t];
elsif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' then
        AFSR0_EL2 = X[t];
    else
        AFSR0_EL1 = X[t];
elsif PSTATE.EL == EL3 then
    AFSR0_EL1 = X[t];
```

MRS <Xt>, AFSR0_EL12

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b11	0b101	0b0101	0b0001	0b000
~~0b11~~	~~0b0101~~	~~0b101~~	~~0b000~~	~~0b0001~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '101' then
        return NVMem[0x128];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if EL2Enabled() && HCR_EL2.E2H == '1' then
    if PSTATE.EL == EL2 then
        return AFSR0_EL1;
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if EL2Enabled() && HCR_EL2.E2H == '1' then
        return AFSR0_EL1;
else
    else
        UNDEFINED;
```

```
MSR AFSR0_EL12, <Xt>
```

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b11	0b101	0b0101	0b0001	0b000
~~0b11~~	~~0b0101~~	~~0b101~~	~~0b000~~	~~0b0001~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '101' then
        NVMem[0x128] = X[t];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if EL2Enabled() && HCR_EL2.E2H == '1' then
    if PSTATE.EL == EL2 then
        AFSR0_EL1 = X[t];
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if EL2Enabled() && HCR_EL2.E2H == '1' then
        AFSR0_EL1 = X[t];
else
    else
        UNDEFINED;
```

27~~13~~/03~~12~~/2019~~2018~~ 21~~16~~:59~~42~~; e5e4db499bf9867a4b93324c4dbac985d3da93766~~379d01e197f1d40720d32d0f84e419e9187e009~~

(old) htmldiff from- (new)

AFSR1_EL1, Auxiliary Fault Status Register 1 (EL1)

The AFSR1_EL1 characteristics are:

Purpose

Provides additional IMPLEMENTATION DEFINED fault status information for exceptions taken to EL1.

Configuration

AArch64 System register AFSR1_EL1 bits [31:0] are architecturally mapped to AArch32 System register AIFSR[31:0] .

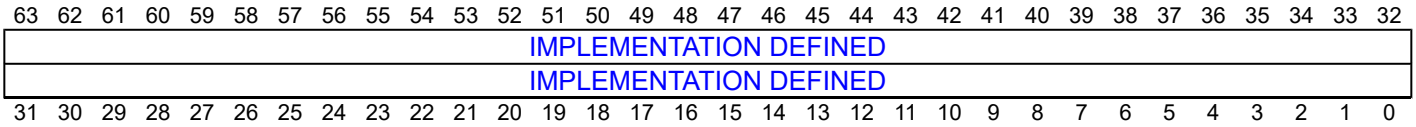
RW fields in this register reset to architecturally UNKNOWN values.

Attributes

AFSR1_EL1 is a 64-bit register.

Field descriptions

The AFSR1_EL1 bit assignments are:



IMPLEMENTATION DEFINED, bits [63:0]

IMPLEMENTATION DEFINED.

This field resets to an architecturally UNKNOWN value.

Accessing the AFSR1_EL1

When HCR_EL2.E2H is 1, without explicit synchronization, access from EL3 using the mnemonic AFSR1_EL1 or AFSR1_EL12 are not guaranteed to be ordered with respect to accesses using the other mnemonic.

Accesses to this register use the following encodings:

```
MRS <Xt>, AFSR1_EL1
```

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b11	0b000	0b0101	0b0001	0b001
~~0b11~~	~~0b0101~~	~~0b000~~	~~0b001~~	~~0b0001~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TRVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<NV2,NV1,NV> == '111' then
        return NVMem[0x130];
    else
        return AFSR1_EL1;
elsif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' then
        return AFSR1_EL2;
    else
        return AFSR1_EL1;
elsif PSTATE.EL == EL3 then
    return AFSR1_EL1;
```

MSR AFSR1_EL1, <Xt>

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b11	0b000	0b0101	0b0001	0b001
~~0b11~~	~~0b0101~~	~~0b000~~	~~0b001~~	~~0b0001~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<NV2,NV1,NV> == '111' then
        NVMem[0x130] = X[t];
    else
        AFSR1_EL1 = X[t];
elsif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' then
        AFSR1_EL2 = X[t];
    else
        AFSR1_EL1 = X[t];
elsif PSTATE.EL == EL3 then
    AFSR1_EL1 = X[t];
```

MRS <Xt>, AFSR1_EL12

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b11	0b101	0b0101	0b0001	0b001
~~0b11~~	~~0b0101~~	~~0b101~~	~~0b001~~	~~0b0001~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '101' then
        return NVMem[0x130];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if EL2Enabled() && HCR_EL2.E2H == '1' then
    if PSTATE.EL == EL2 then
        return AFSR1_EL1;
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if EL2Enabled() && HCR_EL2.E2H == '1' then
        return AFSR1_EL1;
else
    else
        UNDEFINED;
```

```
MSR AFSR1_EL12, <Xt>
```

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b11	0b101	0b0101	0b0001	0b001
~~0b11~~	~~0b0101~~	~~0b101~~	~~0b001~~	~~0b0001~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '101' then
        NVMem[0x130] = X[t];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if EL2Enabled() && HCR_EL2.E2H == '1' then
    if PSTATE.EL == EL2 then
        AFSR1_EL1 = X[t];
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if EL2Enabled() && HCR_EL2.E2H == '1' then
        AFSR1_EL1 = X[t];
else
    else
        UNDEFINED;
```

27~~13~~/03~~12~~/2019~~2018~~ 21~~16~~:59~~42~~; e5e4db499bf9867a4b93324c4dbac985d3da93766~~379d01e197f1d40720d32d0f84e419e9187e009~~

(old) htmldiff from- (new)

AMAIR_EL1, Auxiliary Memory Attribute Indirection Register (EL1)

The AMAIR_EL1 characteristics are:

Purpose

Provides IMPLEMENTATION DEFINED memory attributes for the memory regions specified by MAIR_EL1.

Configuration

AArch64 System register AMAIR_EL1 bits [31:0] are architecturally mapped to AArch32 System register AMAIR0[31:0] .

AArch64 System register AMAIR_EL1 bits [63:32] are architecturally mapped to AArch32 System register AMAIR1[31:0] .

RW fields in this register reset to architecturally UNKNOWN values.

Attributes

AMAIR_EL1 is a 64-bit register.

Field descriptions

The AMAIR_EL1 bit assignments are:

63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
IMPLEMENTATION DEFINED																															
IMPLEMENTATION DEFINED																															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

AMAIR_EL1 is permitted to be cached in a TLB.

IMPLEMENTATION DEFINED, bits [63:0]

IMPLEMENTATION DEFINED.

This field resets to an architecturally UNKNOWN value.

Accessing the AMAIR_EL1

AMAIR_EL1 is permitted to be cached in a TLB.

When HCR_EL2.E2H is 1, without explicit synchronization, access from EL3 using the mnemonic AMAIR_EL1 or AMAIR_EL12 are not guaranteed to be ordered with respect to accesses using the other mnemonic.

Accesses to this register use the following encodings:

```
MRS <Xt>, AMAIR_EL1
```

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b11	0b000	0b1010	0b0011	0b000
~~0b11~~	~~0b1010~~	~~0b000~~	~~0b000~~	~~0b0011~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TRVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<NV2,NV1,NV> == '111' then
        return NVMem[0x148];
    else
        return AMAIR_EL1;
elsif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' then
        return AMAIR_EL2;
    else
        return AMAIR_EL1;
elsif PSTATE.EL == EL3 then
    return AMAIR_EL1;
```

```
MSR AMAIR_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b11	0b000	0b1010	0b0011	0b000
~~0b11~~	~~0b1010~~	~~0b000~~	~~0b000~~	~~0b0011~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<NV2,NV1,NV> == '111' then
        NVMem[0x148] = X[t];
    else
        AMAIR_EL1 = X[t];
elsif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' then
        AMAIR_EL2 = X[t];
    else
        AMAIR_EL1 = X[t];
elsif PSTATE.EL == EL3 then
    AMAIR_EL1 = X[t];
```

```
MRS <Xt>, AMAIR_EL12
```

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b11	0b101	0b1010	0b0011	0b000
~~0b11~~	~~0b1010~~	~~0b101~~	~~0b000~~	~~0b0011~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '101' then
        return NVMem[0x148];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if EL2Enabled() && HCR_EL2.E2H == '1' then
    if PSTATE.EL == EL2 then
        return AMAIR_EL1;
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if EL2Enabled() && HCR_EL2.E2H == '1' then
        return AMAIR_EL1;
else
    else
        UNDEFINED;
```

```
MSR AMAIR_EL12, <Xt>
```

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b11	0b101	0b1010	0b0011	0b000
~~0b11~~	~~0b1010~~	~~0b101~~	~~0b000~~	~~0b0011~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '101' then
        NVMem[0x148] = X[t];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if EL2Enabled() && HCR_EL2.E2H == '1' then
    if PSTATE.EL == EL2 then
        AMAIR_EL1 = X[t];
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if EL2Enabled() && HCR_EL2.E2H == '1' then
        AMAIR_EL1 = X[t];
else
    else
        UNDEFINED;
```

27~~13~~/03~~12~~/2019~~2018~~ 21~~16~~:59~~42~~; e5e4db499bf9867a4b93324c4dbac985d3da93766~~379d01e197f1d40720d32d0f84e419e9187e009~~

(old) htmldiff from- (new)

AMAIR_EL2, Auxiliary Memory Attribute Indirection Register (EL2)

The AMAIR_EL2 characteristics are:

Purpose

Provides IMPLEMENTATION DEFINED memory attributes for the memory regions specified by MAIR_EL2.

Configuration

AArch64 System register AMAIR_EL2 bits [31:0] are architecturally mapped to AArch32 System register HAMAIR0[31:0] .

AArch64 System register AMAIR_EL2 bits [63:32] are architecturally mapped to AArch32 System register HAMAIR1[31:0] .

If EL2 is not implemented, this register is RES0 from EL3.

This register has no effect if EL2 is not enabled in the current Security state.

RW fields in this register reset to architecturally UNKNOWN values.

Attributes

AMAIR_EL2 is a 64-bit register.

Field descriptions

The AMAIR_EL2 bit assignments are:

63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
IMPLEMENTATION DEFINED																															
IMPLEMENTATION DEFINED																															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

AMAIR_EL2 is permitted to be cached in a TLB.

IMPLEMENTATION DEFINED, bits [63:0]

IMPLEMENTATION DEFINED.

This field resets to an architecturally UNKNOWN value.

Accessing the AMAIR_EL2

AMAIR_EL2 is permitted to be cached in a TLB.

When HCR_EL2.E2H is 1, without explicit synchronization, access from EL2 using the mnemonic AMAIR_EL2 or AMAIR_EL1 are not guaranteed to be ordered with respect to accesses using the other mnemonic.

Accesses to this register use the following encodings:

```
MRS <Xt>, AMAIR_EL2
```

op0	op1	CRn	CRm	op2

~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b11	0b100	0b1010	0b0011	0b000
~~0b11~~	~~0b1010~~	~~0b100~~	~~0b000~~	~~0b0011~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    return AMAIR_EL2;
elsif PSTATE.EL == EL3 then
    return AMAIR_EL2;
```

MSR AMAIR_EL2, <Xt>

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b11	0b100	0b1010	0b0011	0b000
~~0b11~~	~~0b1010~~	~~0b100~~	~~0b000~~	~~0b0011~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    AMAIR_EL2 = X[t];
elsif PSTATE.EL == EL3 then
    AMAIR_EL2 = X[t];
```

MRS <Xt>, AMAIR_EL1

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b11	0b000	0b1010	0b0011	0b000
~~0b11~~	~~0b1010~~	~~0b000~~	~~0b000~~	~~0b0011~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TRVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<NV2,NV1,NV> == '111' then
        return NVMem[0x148];
    else
        return AMAIR_EL1;
elsif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' then
        return AMAIR_EL2;
    else
        return AMAIR_EL1;
elsif PSTATE.EL == EL3 then
    return AMAIR_EL1;
```

```
MSR AMAIR_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b11	0b000	0b1010	0b0011	0b000
~~0b11~~	~~0b1010~~	~~0b000~~	~~0b000~~	~~0b0011~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<NV2,NV1,NV> == '111' then
        NVMem[0x148] = X[t];
    else
        AMAIR_EL1 = X[t];
elsif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' then
        AMAIR_EL2 = X[t];
    else
        AMAIR_EL1 = X[t];
elsif PSTATE.EL == EL3 then
    AMAIR_EL1 = X[t];
```

27~~13~~/03~~12~~/2019~~2018~~ 21~~16~~:59~~42~~; e5e4db499bf9867a4b93324c4dbac985d3da93766~~379d01e197f1d40720d32d0f84e419e9187e009~~

(old) htmldiff from- (new)

AMAIR_EL3, Auxiliary Memory Attribute Indirection Register (EL3)

The AMAIR_EL3 characteristics are:

Purpose

Provides IMPLEMENTATION DEFINED memory attributes for the memory regions specified by MAIR_EL3.

Configuration

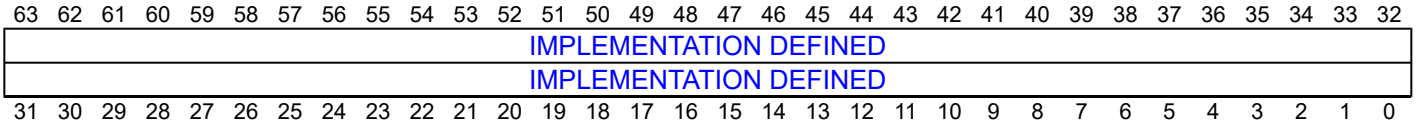
RW fields in this register reset to architecturally UNKNOWN values.

Attributes

AMAIR_EL3 is a 64-bit register.

Field descriptions

The AMAIR_EL3 bit assignments are:



AMAIR_EL3 is permitted to be cached in a TLB.

IMPLEMENTATION DEFINED, bits [63:0]

IMPLEMENTATION DEFINED.

This field resets to an architecturally UNKNOWN value.

Accessing the AMAIR_EL3

AMAIR_EL3 is permitted to be cached in a TLB.

Accesses to this register use the following encodings:

```
MRS <Xt>, AMAIR_EL3
```

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b11	0b110	0b1010	0b0011	0b000
~~0b11~~	~~0b1010~~	~~0b110~~	~~0b000~~	~~0b0011~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    return AMAIR_EL3;
```

```
MSR AMAIR_EL3, <Xt>
```

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b11	0b110	0b1010	0b0011	0b000
~~0b11~~	~~0b1010~~	~~0b110~~	~~0b000~~	~~0b0011~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    AMAIR_EL3 = X[t];
```

27~~13~~/03~~12~~/2019~~2018~~ 21~~16~~:59~~42~~; e5e4db499bf9867a4b93324c4dbac985d3da93766379d01e197f1d40720d32d0f84c419e9187e009

(old) htmldiff from- (new)

AMEVCNTR0<n>_EL0, Activity Monitors Event Counter Registers 0, n = 0 - 15

The AMEVCNTR0<n>_EL0 characteristics are:

Purpose

Provides access to the architected activity monitor event counters.

Configuration

AArch64 System register AMEVCNTR0<n>_EL0 bits [63:0] are architecturally mapped to AArch32 System register AMEVCNTR0<n>[63:0] .

AArch64 System register AMEVCNTR0<n>_EL0 bits [63:0] are architecturally mapped to External register AMEVCNTR0<n>[63:0] .

This register is present only when AMUv1 is implemented. Otherwise, direct accesses to AMEVCNTR0<n>_EL0 are UNDEFINED.

Some or all RW fields of this register have defined reset values. These apply only if the PE resets into an Exception level that is using AArch64. Otherwise, RW fields in this register reset to architecturally UNKNOWN values.

Attributes

AMEVCNTR0<n>_EL0 is a 64-bit register.

Field descriptions

The AMEVCNTR0<n>_EL0 bit assignments are:

63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
ACNT																															
ACNT																															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

ACNT, bits [63:0]

Architected activity monitor event counter n.

Value of architected activity monitor event counter n, where n is the number of this register and is a number from 0 to 15.

If the counter is enabled, writes to this register have UNPREDICTABLE results.

On a Cold reset, this field resets to 0.

Accessing the AMEVCNTR0<n>_EL0

If <n> is greater than or equal to the number of architected activity monitor event counters, reads and writes of AMEVCNTR0<n>_EL0 are CONSTRAINED UNPREDICTABLE, and the following behaviors are permitted:

- Accesses to the register are UNDEFINED.
- Accesses to the register behave as RAZ/WI.
- Accesses to the register execute as a NOP.

Note

AMCGCR_EL0.CG0NC identifies the number of architected activity monitor event counters.

Accesses to this register use the following encodings:

MRS <Xt>, AMEVCNTR0<n>_EL0

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b11	0b011	0b1101	0b010[n:3]	0b[n:2:0]
~~0b11~~	~~0b1101~~	~~0b011~~	~~0b[n:2:0]~~	~~0b010[n:3]~~

```
if PSTATE.EL == EL0 then
    if !ELUsingAArch32(EL1) && AMUSERENR_EL0.EN == '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && CPTR_EL2.TAM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TAM == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return AMEVCNTR0_EL0[UInt(CRm<0>:op2<2:0>)];
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && CPTR_EL2.TAM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TAM == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return AMEVCNTR0_EL0[UInt(CRm<0>:op2<2:0>)];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TAM == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return AMEVCNTR0_EL0[UInt(CRm<0>:op2<2:0>)];
elsif PSTATE.EL == EL3 then
    return AMEVCNTR0_EL0[UInt(CRm<0>:op2<2:0>)];
```

MSR AMEVCNTR0<n>_EL0, <Xt>

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b11	0b011	0b1101	0b010[n:3]	0b[n:2:0]
~~0b11~~	~~0b1101~~	~~0b011~~	~~0b[n:2:0]~~	~~0b010[n:3]~~

```
if IsHighestEL(PSTATE.EL) then
    AMEVCNTR0_EL0[UInt(CRm<0>:op2<2:0>)] = X[t];
else
    UNDEFINED;
```

2713/0312/20192018 2116:5942; e5e4db499bf9867a4b93324c4dbac985d3da93766379d01e197f1d40720d32d0f84c419e9187c009

(old) htmldiff from- (new)

AMEVCNTR1<n>_EL0, Activity Monitors Event Counter Registers 1, n = 0 - 15

The AMEVCNTR1<n>_EL0 characteristics are:

Purpose

Provides access to the auxiliary activity monitor event counters.

Configuration

AArch64 System register AMEVCNTR1<n>_EL0 bits [63:0] are architecturally mapped to AArch32 System register AMEVCNTR1<n>[63:0] .

AArch64 System register AMEVCNTR1<n>_EL0 bits [63:0] are architecturally mapped to External register AMEVCNTR1<n>[63:0] .

This register is present only when AMUv1 is implemented. Otherwise, direct accesses to AMEVCNTR1<n>_EL0 are UNDEFINED.

Some or all RW fields of this register have defined reset values. These apply only if the PE resets into an Exception level that is using AArch64. Otherwise, RW fields in this register reset to architecturally UNKNOWN values.

Attributes

AMEVCNTR1<n>_EL0 is a 64-bit register.

Field descriptions

The AMEVCNTR1<n>_EL0 bit assignments are:

63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
ACNT																															
ACNT																															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

ACNT, bits [63:0]

Auxiliary activity monitor event counter n.

Value of auxiliary activity monitor event counter n, where n is the number of this register and is a number from 0 to 15.

If the counter is enabled, writes to this register have UNPREDICTABLE results.

On a Cold reset, this field resets to 0.

Accessing the AMEVCNTR1<n>_EL0

If <n> is greater than or equal to the number of auxiliary activity monitor event counters, reads and writes of AMEVCNTR1<n>_EL0 are CONSTRAINED UNPREDICTABLE, and the following behaviors are permitted:

- Accesses to the register are UNDEFINED.
- Accesses to the register behave as RAZ/WI.
- Accesses to the register execute as a NOP.

Note

AMCGCR_EL0.CG1NC identifies the number of auxiliary activity monitor event counters.

Accesses to this register use the following encodings:

MRS <Xt>, AMEVCNTR1<n>_EL0

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b11	0b011	0b1101	0b110[n:3]	0b[n:2:0]
~~0b11~~	~~0b1101~~	~~0b011~~	~~0b[n:2:0]~~	~~0b110[n:3]~~

```
if PSTATE.EL == EL0 then
    if !ELUsingAArch32(EL1) && AMUSERENR_EL0.EN == '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && CPTR_EL2.TAM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TAM == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return AMEVCNTR1_EL0[UInt(CRm<0>:op2<2:0>)];
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && CPTR_EL2.TAM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TAM == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return AMEVCNTR1_EL0[UInt(CRm<0>:op2<2:0>)];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TAM == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return AMEVCNTR1_EL0[UInt(CRm<0>:op2<2:0>)];
elsif PSTATE.EL == EL3 then
    return AMEVCNTR1_EL0[UInt(CRm<0>:op2<2:0>)];
```

MSR AMEVCNTR1<n>_EL0, <Xt>

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b11	0b011	0b1101	0b110[n:3]	0b[n:2:0]
~~0b11~~	~~0b1101~~	~~0b011~~	~~0b[n:2:0]~~	~~0b110[n:3]~~

```
if IsHighestEL(PSTATE.EL) then
    AMEVCNTR1_EL0[UInt(CRm<0>:op2<2:0>)] = X[t];
else
    UNDEFINED;
```

2713/0312/20192018 2116:5942; e5e4db499bf9867a4b93324c4dbac985d3da93766379d01e197f1d40720d32d0f84c419e9187e009

(old) htmldiff from- (new)

AMEVTYPER0<n>_EL0, Activity Monitors Event Type Registers 0, n = 0 - 15

The AMEVTYPER0<n>_EL0 characteristics are:

Purpose

Provides information on the events that an architected activity monitor event counter AMEVCNTR0<n>_EL0 counts.

Configuration

AArch64 System register AMEVTYPER0<n>_EL0 bits [31:0] are architecturally mapped to AArch32 System register AMEVTYPER0<n>[31:0] .

AArch64 System register AMEVTYPER0<n>_EL0 bits [31:0] are architecturally mapped to External register AMEVTYPER0<n>[31:0] .

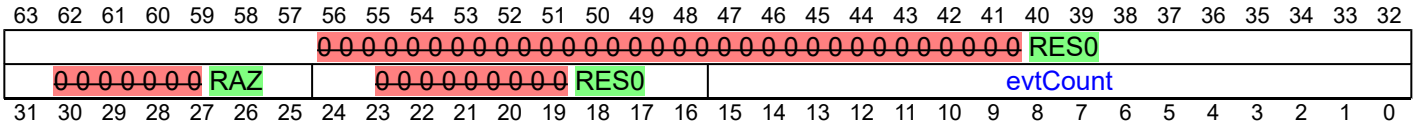
This register is present only when AMUv1 is implemented. Otherwise, direct accesses to AMEVTYPER0<n>_EL0 are UNDEFINED.

Attributes

AMEVTYPER0<n>_EL0 is a 64-bit register.

Field descriptions

The AMEVTYPER0<n>_EL0 bit assignments are:



Bits [63:32]

Reserved, RES0.

Bits [31:25]

Reserved, RAZ.

Bits [24:16]

Reserved, RES0.

evtCount, bits [15:0]

Event to count. The event number of the event that is counted by the architected activity monitor event counter AMEVCNTR0<n>_EL0. The value of this field is architecturally mandated for each architected counter.

The following table shows the mapping between required event numbers and the corresponding counters:

evtCount	Meaning	Applies when
0x0011	Processor frequency cycles	When n == 0
0x4004	Constant frequency cycles	When n == 1
0x0008	Instructions retired	When n == 2
0x4005	Memory stall cycles	When n == 3

Accessing the AMEVTYPER0<n>_EL0

If <n> is greater than or equal to the number of architected activity monitor event counters, reads and writes of AMEVTYPER0<n>_EL0 are CONSTRAINED UNPREDICTABLE, and the following behaviors are permitted:

- Accesses to the register are UNDEFINED.
- Accesses to the register behave as RAZ/WI.
- Accesses to the register execute as a NOP.

Note

AMCGCR_EL0.CG0NC identifies the number of architected activity monitor event counters.

Accesses to this register use the following encodings:

```
MRS <Xt>, AMEVTYPER0<n>_EL0
```

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b11	0b011	0b1101	0b011[n:3]	0b[n:2:0]
~~0b11~~	~~0b1101~~	~~0b011~~	~~0b[n:2:0]~~	~~0b011[n:3]~~

```
if PSTATE.EL == EL0 then
    if !ELUsingAArch32(EL1) && AMUSERENR_EL0.EN == '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && CPTR_EL2.TAM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TAM == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return AMEVTYPER0_EL0AMEVCNTR1_EL0[UInt(CRm<0>:op2<2:0>)];
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && CPTR_EL2.TAM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TAM == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return AMEVTYPER0_EL0AMEVCNTR1_EL0[UInt(CRm<0>:op2<2:0>)];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TAM == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return AMEVTYPER0_EL0AMEVCNTR1_EL0[UInt(CRm<0>:op2<2:0>)];
elsif PSTATE.EL == EL3 then
    return AMEVTYPER0_EL0AMEVCNTR1_EL0[UInt(CRm<0>:op2<2:0>)];
```

27~~13~~/03~~12~~/2019~~2018~~ 21~~16~~:59~~42~~; e5e4db499bf9867a4b93324c4dbac985d3da93766~~379d01e197f1d40720d32d0f84e419e9187e009~~

(old) htmldiff from- (new)

AMEVTYPER1<n>_EL0, Activity Monitors Event Type Registers 1, n = 0 - 15

The AMEVTYPER1<n>_EL0 characteristics are:

Purpose

Provides information on the events that an auxiliary activity monitor event counter AMEVCNTR1<n>_EL0 counts.

Configuration

AArch64 System register AMEVTYPER1<n>_EL0 bits [31:0] are architecturally mapped to AArch32 System register AMEVTYPER1<n>[31:0] .

AArch64 System register AMEVTYPER1<n>_EL0 bits [31:0] are architecturally mapped to External register AMEVTYPER1<n>[31:0] .

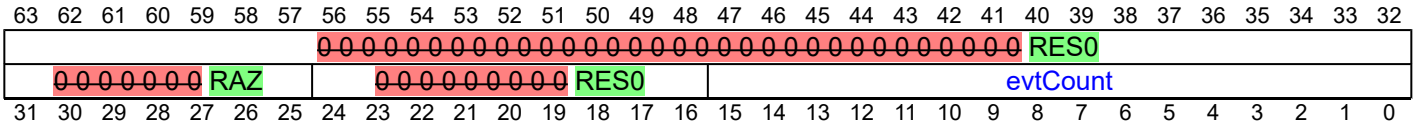
This register is present only when AMUv1 is implemented. Otherwise, direct accesses to AMEVTYPER1<n>_EL0 are UNDEFINED.

Attributes

AMEVTYPER1<n>_EL0 is a 64-bit register.

Field descriptions

The AMEVTYPER1<n>_EL0 bit assignments are:



Bits [63:32]

Reserved, RES0.

Bits [31:25]

Reserved, RAZ.

Bits [24:16]

Reserved, RES0.

evtCount, bits [15:0]

Event to count. The event number of the event that is counted by the auxiliary activity monitor event counter AMEVCNTR1<n>_EL0.

It is IMPLEMENTATION DEFINED what values are supported by each counter.

If software writes a value to this field which is not supported by the corresponding counter AMEVCNTR1<n>_EL0, then:

- It is UNPREDICTABLE which event will be counted.
- The value read back is UNKNOWN.

The event counted by AMEVCNTR1<n>_EL0 might be fixed at implementation. In this case, the field is read-only and writes are UNDEFINED.

If the corresponding counter AMEVCNTR1<n>_EL0 is enabled, writes to this register have UNPREDICTABLE results.

Accessing the AMEVTYPER1<n>_EL0

If <n> is greater than or equal to the number of auxiliary activity monitor event counters, reads and writes of AMEVTYPER1<n>_EL0 are CONSTRAINED UNPREDICTABLE, and the following behaviors are permitted:

- Accesses to the register are UNDEFINED.
- Accesses to the register behave as RAZ/WI.
- Accesses to the register execute as a NOP.

Note

AMCGCR_EL0.CG1NC identifies the number of auxiliary activity monitor event counters.

Accesses to this register use the following encodings:

MRS <Xt>, AMEVTYPER1<n>_EL0

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b11	0b011	0b1101	0b111[n:3]	0b[n:2:0]
~~0b11~~	~~0b1101~~	~~0b011~~	~~0b[n:2:0]~~	~~0b111[n:3]~~

```
if PSTATE.EL == EL0 then
    if !ELUsingAArch32(EL1) && AMUSERENR_EL0.EN == '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && CPTR_EL2.TAM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TAM == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return AMEVTYPER1_EL0[UInt(CRm<0>:op2<2:0>)];
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && CPTR_EL2.TAM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TAM == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return AMEVTYPER1_EL0[UInt(CRm<0>:op2<2:0>)];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TAM == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return AMEVTYPER1_EL0[UInt(CRm<0>:op2<2:0>)];
elsif PSTATE.EL == EL3 then
    return AMEVTYPER1_EL0[UInt(CRm<0>:op2<2:0>)];
```

MSR AMEVTYPER1<n>_EL0, <Xt>

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b11	0b011	0b1101	0b111[n:3]	0b[n:2:0]
~~0b11~~	~~0b1101~~	~~0b011~~	~~0b[n:2:0]~~	~~0b111[n:3]~~

```
if IsHighestEL(PSTATE.EL) then
    AMEVTYPER1_EL0[UInt(CRm<0>:op2<2:0>)] = X[t];
else
    UNDEFINED;
```


(old) htmldiff from- (new)

(old) htmldiff from- (new)

AT S12E0R, Address Translate Stages 1 and 2 EL0 Read

The AT S12E0R characteristics are:

Purpose

Performs stage 1 and 2 address translations from EL0, with permissions as if reading from the given virtual address from EL0, using the following translation regime:

- When EL2 is implemented and enabled in the ~~current~~ Security state described by the current value of~~:~~ SCR_EL3.NS:
  - If HCR_EL2.{E2H, TGE} is not {1, 1}, the EL1&0 translation regime.
  - If HCR_EL2.{E2H, TGE} is {1, 1}, the EL2&0 translation regime.
- Otherwise, the EL1&0 translation regime.

Configuration

Attributes

AT S12E0R is a 64-bit System instruction.

Field descriptions

The AT S12E0R input value bit assignments are:

63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
Input address for translation																															
Input address for translation																															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Bits [63:0]

Input address for translation. The resulting address can be read from the PAR_EL1.

If the address translation instructions are targeting a translation regime that is using AArch32, and so has a VA of only 32 bits, then VA[63:32] is RES0.

Executing the AT S12E0R instruction

Accesses to this instruction use the following encodings:

```
AT S12E0R, <Xt>
```

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b01	0b100	0b0111	0b1000	0b110
~~0b01~~	~~0b0111~~	~~0b100~~	~~0b110~~	~~0b1000~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HCR_EL2.<E2H,TGE> == '11' || HCR_EL2.<DC,VM> == '00' then
        AT_S1E0R(X[t]);
    else
        AT_S12E0R(X[t]);
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        AT_S1E0R(X[t]);
    elsif EL2Enabled() && (HCR_EL2.<E2H,TGE> == '11' || HCR_EL2.<DC,VM> == '00') then
        AT_S1E0R(X[t]);
    else
        AT_S12E0R(X[t]);
```

27~~13~~/03~~12~~/2019~~2018~~ 21~~16~~:59~~42~~; e5e4db499bf9867a4b93324c4dbac985d3da93766~~379d01e197f1d40720d32d0f84c419c9187c009~~

(old) htmldiff from- (new)

(old) htmldiff from- (new)

AT S12E0W, Address Translate Stages 1 and 2 EL0 Write

The AT S12E0W characteristics are:

Purpose

Performs stage 1 and 2 address translations from EL0, with permissions as if writing to the given virtual address from EL0, using the following translation regime:

- When EL2 is implemented and enabled in the ~~current~~ Security state described by the current value of~~:~~ SCR_EL3.NS:
  - If HCR_EL2.{E2H, TGE} is not {1, 1}, the EL1&0 translation regime.
  - If HCR_EL2.{E2H, TGE} is {1, 1}, the EL2&0 translation regime.
- Otherwise, the EL1&0 translation regime.

Configuration

Attributes

AT S12E0W is a 64-bit System instruction.

Field descriptions

The AT S12E0W input value bit assignments are:

63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
Input address for translation																															
Input address for translation																															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Bits [63:0]

Input address for translation. The resulting address can be read from the PAR_EL1.

If the address translation instructions are targeting a translation regime that is using AArch32, and so has a VA of only 32 bits, then VA[63:32] is RES0.

Executing the AT S12E0W instruction

Accesses to this instruction use the following encodings:

```
AT S12E0W, <Xt>
```

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b01	0b100	0b0111	0b1000	0b111
~~0b01~~	~~0b0111~~	~~0b100~~	~~0b111~~	~~0b1000~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HCR_EL2.<E2H,TGE> == '11' || HCR_EL2.<DC,VM> == '00' then
        AT_S1E0W(X[t]);
    else
        AT_S12E0W(X[t]);
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        AT_S1E0W(X[t]);
    elsif EL2Enabled() && (HCR_EL2.<E2H,TGE> == '11' || HCR_EL2.<DC,VM> == '00') then
        AT_S1E0W(X[t]);
    else
        AT_S12E0W(X[t]);
```

27~~13~~/03~~12~~/2019~~2018~~ 21~~16~~:59~~42~~; e5e4db499bf9867a4b93324c4dbac985d3da93766~~379d01e197f1d40720d32d0f84c419c9187c009~~

(old) htmldiff from- (new)

(old) htmldiff from- (new)

AT S12E1R, Address Translate Stages 1 and 2 EL1 Read

The AT S12E1R characteristics are:

Purpose

Performs stage 1 and 2 address translation, with permissions as if reading from the given virtual address from EL1, or from EL2 if the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}, using the following translation regime:

- When EL2 is implemented and enabled in the ~~current~~ Security state described by the current value of~~:~~ SCR_EL3.NS:
  - If HCR_EL2.{E2H, TGE} is not {1, 1}, the EL1&0 translation regime, accessed from EL1.
  - If HCR_EL2.{E2H, TGE} is {1, 1}, the EL2&0 translation regime, accessed from EL2.
- Otherwise, the EL1&0 translation regime, accessed from EL1.

Configuration

Attributes

AT S12E1R is a 64-bit System instruction.

Field descriptions

The AT S12E1R input value bit assignments are:

63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
Input address for translation																															
Input address for translation																															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Bits [63:0]

Input address for translation. The resulting address can be read from the PAR_EL1.

If the address translation instructions are targeting a translation regime that is using AArch32, and so has a VA of only 32 bits, then VA[63:32] is RES0.

Executing the AT S12E1R instruction

Accesses to this instruction use the following encodings:

```
AT S12E1R, <Xt>
```

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b01	0b100	0b0111	0b1000	0b100
~~0b01~~	~~0b0111~~	~~0b100~~	~~0b100~~	~~0b1000~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HCR_EL2.<E2H,TGE> == '11' || HCR_EL2.<DC,VM> == '00' then
        AT_S1E1R(X[t]);
    else
        AT_S12E1R(X[t]);
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        AT_S1E1R(X[t]);
    elsif EL2Enabled() && (HCR_EL2.<E2H,TGE> == '11' || HCR_EL2.<DC,VM> == '00') then
        AT_S1E1R(X[t]);
    else
        AT_S12E1R(X[t]);
```

2713/0312/20192018 2116:5942; e5e4db499bf9867a4b93324c4dbac985d3da93766379d01e197f1d40720d32d0f84c419c9187c009

(old) htmldiff from- (new)

(old) htmldiff from- (new)

AT S12E1W, Address Translate Stages 1 and 2 EL1 Write

The AT S12E1W characteristics are:

Purpose

Performs stage 1 and 2 address translation, with permissions as if writing to the given virtual address from EL1, or from EL2 if the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}, using the following translation regime:

- When EL2 is implemented and enabled in the ~~current~~ Security state described by the current value of~~:~~ SCR_EL3.NS:
  - If HCR_EL2.{E2H, TGE} is not {1, 1}, the EL1&0 translation regime, accessed from EL1.
  - If HCR_EL2.{E2H, TGE} is {1, 1}, the EL2&0 translation regime, accessed from EL2.
- Otherwise, the EL1&0 translation regime, accessed from EL1.

Configuration

Attributes

AT S12E1W is a 64-bit System instruction.

Field descriptions

The AT S12E1W input value bit assignments are:

63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
Input address for translation																															
Input address for translation																															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Bits [63:0]

Input address for translation. The resulting address can be read from the PAR_EL1.

If the address translation instructions are targeting a translation regime that is using AArch32, and so has a VA of only 32 bits, then VA[63:32] is RES0.

Executing the AT S12E1W instruction

Accesses to this instruction use the following encodings:

```
AT S12E1W, <Xt>
```

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b01	0b100	0b0111	0b1000	0b101
~~0b01~~	~~0b0111~~	~~0b100~~	~~0b101~~	~~0b1000~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HCR_EL2.<E2H,TGE> == '11' || HCR_EL2.<DC,VM> == '00' then
        AT_S1E1W(X[t]);
    else
        AT_S12E1W(X[t]);
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        AT_S1E1W(X[t]);
    elsif EL2Enabled() && (HCR_EL2.<E2H,TGE> == '11' || HCR_EL2.<DC,VM> == '00') then
        AT_S1E1W(X[t]);
    else
        AT_S12E1W(X[t]);
```

27~~13~~/03~~12~~/2019~~2018~~ 21~~16~~:59~~42~~; e5e4db499bf9867a4b93324c4dbac985d3da93766~~379d01e197f1d40720d32d0f84c419c9187c009~~

(old) htmldiff from- (new)

(old) htmldiff from- (new)

AT S1E0R, Address Translate Stage 1 EL0 Read

The AT S1E0R characteristics are:

Purpose

Performs stage 1 address translation from EL0, with permissions as if reading from the given virtual address from EL0, using the following translation regime:

- When EL2 is implemented and enabled in the ~~current~~ Security state described by the current value of~~:~~ SCR_EL3.NS:
  - If HCR_EL2.{E2H, TGE} is not {1, 1}, the EL1&0 translation regime.
  - If HCR_EL2.{E2H, TGE} is {1, 1}, the EL2&0 translation regime.
- Otherwise, the EL1&0 translation regime.

Configuration

Attributes

AT S1E0R is a 64-bit System instruction.

Field descriptions

The AT S1E0R input value bit assignments are:

63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
Input address for translation																															
Input address for translation																															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Bits [63:0]

Input address for translation. The resulting address can be read from the PAR_EL1.

If the address translation instructions are targeting a translation regime that is using AArch32, and so has a VA of only 32 bits, then VA[63:32] is RES0.

Executing the AT S1E0R instruction

Accesses to this instruction use the following encodings:

```
AT S1E0R, <Xt>
```

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b01	0b000	0b0111	0b1000	0b010
~~0b01~~	~~0b0111~~	~~0b000~~	~~0b010~~	~~0b1000~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.AT == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        AT_S1E0R(X[t]);
elsif PSTATE.EL == EL2 then
    AT_S1E0R(X[t]);
elsif PSTATE.EL == EL3 then
    AT_S1E0R(X[t]);
```

27~~13~~/03~~12~~/2019~~2018~~ 21~~16~~:59~~42~~; e5e4db499bf9867a4b93324c4dbac985d3da9376~~6379d01e197f1d40720d32d0f84e419e9187e009~~

(old) htmldiff from- (new)

(old) htmldiff from- (new)

AT S1E0W, Address Translate Stage 1 EL0 Write

The AT S1E0W characteristics are:

Purpose

Performs stage 1 address translation from EL0, with permissions as if writing to the given virtual address from EL0, using the following translation regime:

- When EL2 is implemented and enabled in the ~~current~~ Security state described by the current value of~~:~~ SCR_EL3.NS:
  - If HCR_EL2.{E2H, TGE} is not {1, 1}, the EL1&0 translation regime.
  - If HCR_EL2.{E2H, TGE} is {1, 1}, the EL2&0 translation regime.
- Otherwise, the EL1&0 translation regime.

Configuration

Attributes

AT S1E0W is a 64-bit System instruction.

Field descriptions

The AT S1E0W input value bit assignments are:

63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
Input address for translation																															
Input address for translation																															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Bits [63:0]

Input address for translation. The resulting address can be read from the PAR_EL1.

If the address translation instructions are targeting a translation regime that is using AArch32, and so has a VA of only 32 bits, then VA[63:32] is RES0.

Executing the AT S1E0W instruction

Accesses to this instruction use the following encodings:

```
AT S1E0W, <Xt>
```

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b01	0b000	0b0111	0b1000	0b011
~~0b01~~	~~0b0111~~	~~0b000~~	~~0b011~~	~~0b1000~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.AT == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        AT_S1E0W(X[t]);
elsif PSTATE.EL == EL2 then
    AT_S1E0W(X[t]);
elsif PSTATE.EL == EL3 then
    AT_S1E0W(X[t]);
```

27~~13~~/03~~12~~/2019~~2018~~ 21~~16~~:59~~42~~; e5e4db499bf9867a4b93324c4dbac985d3da93766~~379d01e197f1d40720d32d0f84e419e9187e009~~

(old) htmldiff from- (new)

(old) htmldiff from- (new)

AT S1E1R, Address Translate Stage 1 EL1 Read

The AT S1E1R characteristics are:

Purpose

Performs stage 1 address translation, with permissions as if reading from the given virtual address from EL1, or from EL2 if the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}, using the following translation regime:

- When EL2 is implemented and enabled in the ~~current~~ Security state described by the current value of~~:~~ SCR_EL3.NS:
  - If HCR_EL2.{E2H, TGE} is not {1, 1}, the EL1&0 translation regime, accessed from EL1.
  - If HCR_EL2.{E2H, TGE} is {1, 1}, the EL2&0 translation regime, accessed from EL2.
- Otherwise, the EL1&0 translation regime, accessed from EL1.

Configuration

Attributes

AT S1E1R is a 64-bit System instruction.

Field descriptions

The AT S1E1R input value bit assignments are:

63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
Input address for translation																															
Input address for translation																															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Bits [63:0]

Input address for translation. The resulting address can be read from the PAR_EL1.

If the address translation instructions are targeting a translation regime that is using AArch32, and so has a VA of only 32 bits, then VA[63:32] is RES0.

Executing the AT S1E1R instruction

Accesses to this instruction use the following encodings:

```
AT S1E1R, <Xt>
```

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b01	0b000	0b0111	0b1000	0b000
~~0b01~~	~~0b0111~~	~~0b000~~	~~0b000~~	~~0b1000~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.AT == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        AT_S1E1R(X[t]);
elsif PSTATE.EL == EL2 then
    AT_S1E1R(X[t]);
elsif PSTATE.EL == EL3 then
    AT_S1E1R(X[t]);
```

27~~13~~/03~~12~~/2019~~2018~~ 21~~16~~:59~~42~~; e5e4db499bf9867a4b93324c4dbac985d3da9376~~6379d01e197f1d40720d32d0f84e419e9187e009~~

(old) htmldiff from- (new)

(old) htmldiff from- (new)

AT S1E1RP, Address Translate Stage 1 EL1 Read PAN

The AT S1E1RP characteristics are:

Purpose

Performs a stage 1 address translation, where the value of PSTATE.PAN determines if a read from a location will generate a permission fault for a privileged access, using the following translation regime:

- When EL2 is implemented and enabled in the ~~current~~ Security state described by the current value of~~:~~ SCR_EL3.NS:
  - If HCR_EL2.{E2H, TGE} is not {1, 1}, the EL1&0 translation regime, accessed from EL1.
  - If HCR_EL2.{E2H, TGE} is {1, 1}, the EL2&0 translation regime, accessed from EL2.
- Otherwise, the EL1&0 translation regime, accessed from EL1.

Configuration

This instruction is present only when ARMv8.2-ATS1E1 is implemented. Otherwise, direct accesses to AT S1E1RP are UNDEFINED.

Attributes

AT S1E1RP is a 64-bit System instruction.

Field descriptions

The AT S1E1RP input value bit assignments are:

63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
Input address for translation																															
Input address for translation																															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Bits [63:0]

Input address for translation. The resulting address can be read from the PAR_EL1.

If the address translation instructions are targeting a translation regime that is using AArch32, and so has a VA of only 32 bits, then VA[63:32] is RES0.

Executing the AT S1E1RP instruction

Accesses to this instruction use the following encodings:

```
AT S1E1RP, <Xt>
```

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b01	0b000	0b0111	0b1001	0b000
~~0b01~~	~~0b0111~~	~~0b000~~	~~0b000~~	~~0b1001~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.AT == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        AT_S1E1RP(X[t]);
elsif PSTATE.EL == EL2 then
    AT_S1E1RP(X[t]);
elsif PSTATE.EL == EL3 then
    AT_S1E1RP(X[t]);
```

2713/0312/20192018 2116:5942: e5e4db499bf9867a4b93324c4dbac985d3da93766379d01e197f1d40720d32d0f84e419e9187e009

(old) htmldiff from- (new)

(old) htmldiff from- (new)

AT S1E1W, Address Translate Stage 1 EL1 Write

The AT S1E1W characteristics are:

Purpose

Performs stage 1 address translation, with permissions as if writing to the given virtual address from EL1, or from EL2 if the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}, using the following translation regime:

- When EL2 is implemented and enabled in the ~~current~~ Security state described by the current value of~~:~~ SCR_EL3.NS:
  - If HCR_EL2.{E2H, TGE} is not {1, 1}, the EL1&0 translation regime, accessed from EL1.
  - If HCR_EL2.{E2H, TGE} is {1, 1}, the EL2&0 translation regime, accessed from EL2.
- Otherwise, the EL1&0 translation regime, accessed from EL1.

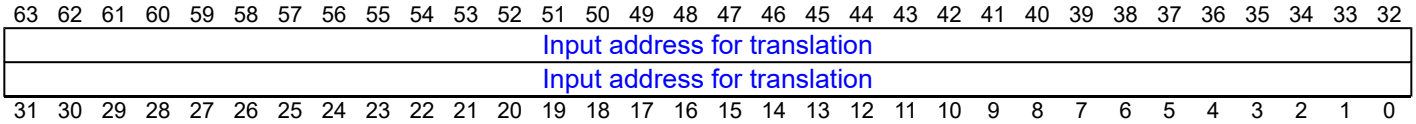
Configuration

Attributes

AT S1E1W is a 64-bit System instruction.

Field descriptions

The AT S1E1W input value bit assignments are:



Bits [63:0]

Input address for translation. The resulting address can be read from the PAR_EL1.

If the address translation instructions are targeting a translation regime that is using AArch32, and so has a VA of only 32 bits, then VA[63:32] is RES0.

Executing the AT S1E1W instruction

Accesses to this instruction use the following encodings:

```
AT S1E1W, <Xt>
```

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b01	0b000	0b0111	0b1000	0b001
~~0b01~~	~~0b0111~~	~~0b000~~	~~0b001~~	~~0b1000~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.AT == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        AT_S1E1W(X[t]);
elsif PSTATE.EL == EL2 then
    AT_S1E1W(X[t]);
elsif PSTATE.EL == EL3 then
    AT_S1E1W(X[t]);
```

27~~13~~/03~~12~~/2019~~2018~~ 21~~16~~:59~~42~~; e5e4db499bf9867a4b93324c4dbac985d3da93766~~379d01e197f1d40720d32d0f84e419e9187e009~~

(old) htmldiff from- (new)

(old) htmldiff from- (new)

AT S1E1WP, Address Translate Stage 1 EL1 Write PAN

The AT S1E1WP characteristics are:

Purpose

Performs a stage 1 address translation, where the value of PSTATE.PAN determines if a write to a location will generate a permission fault for a privileged access, using the following translation regime:

- When EL2 is implemented and enabled in the ~~current~~ Security state described by the current value of~~:~~ SCR_EL3.NS:
  - If HCR_EL2.{E2H, TGE} is not {1, 1}, the EL1&0 translation regime, accessed from EL1.
  - If HCR_EL2.{E2H, TGE} is {1, 1}, the EL2&0 translation regime, accessed from EL2.
- Otherwise, the EL1&0 translation regime, accessed from EL1.

Configuration

This instruction is present only when ARMv8.2-ATS1E1 is implemented. Otherwise, direct accesses to AT S1E1WP are UNDEFINED.

Attributes

AT S1E1WP is a 64-bit System instruction.

Field descriptions

The AT S1E1WP input value bit assignments are:

63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
Input address for translation																															
Input address for translation																															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Bits [63:0]

Input address for translation. The resulting address can be read from the PAR_EL1.

If the address translation instructions are targeting a translation regime that is using AArch32, and so has a VA of only 32 bits, then VA[63:32] is RES0.

Executing the AT S1E1WP instruction

Accesses to this instruction use the following encodings:

```
AT S1E1WP, <Xt>
```

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b01	0b000	0b0111	0b1001	0b001
~~0b01~~	~~0b0111~~	~~0b000~~	~~0b001~~	~~0b1001~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.AT == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        AT_S1E1WP(X[t]);
elsif PSTATE.EL == EL2 then
    AT_S1E1WP(X[t]);
elsif PSTATE.EL == EL3 then
    AT_S1E1WP(X[t]);
```

27~~13~~/03~~12~~/2019~~2018~~ 21~~16~~:59~~42~~; e5e4db499bf9867a4b93324c4dbac985d3da9376~~6379d01e197f1d40720d32d0f84e419e9187e009~~

(old) htmldiff from- (new)

CCSIDR_EL1, Current Cache Size ID Register

The CCSIDR_EL1 characteristics are:

Purpose

Provides information about the architecture of the currently selected cache.

Configuration

AArch64 System register CCSIDR_EL1 bits [31:0] are architecturally mapped to AArch32 System register CCSIDR[31:0] .

AArch64 System register CCSIDR_EL1 bits [63:32] are architecturally mapped to AArch32 System register CCSIDR2.

The implementation includes one CCSIDR_EL1 for each cache that it can access. CSSELR_EL1 selects which Cache Size ID Register is accessible.

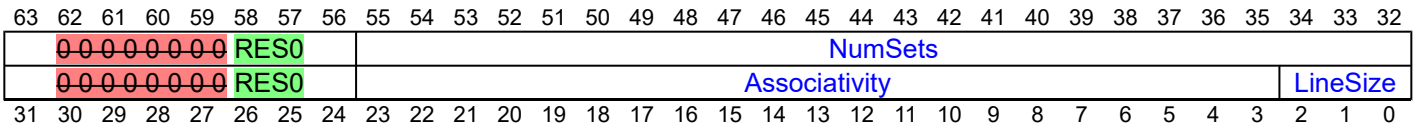
Attributes

CCSIDR_EL1 is a 64-bit register.

Field descriptions

The CCSIDR_EL1 bit assignments are:

When ARMv8.3-CCIDX is implemented:



Note

The parameters NumSets, Associativity, and LineSize in these registers define the architecturally visible parameters that are required for the cache maintenance by Set/Way instructions. They are not guaranteed to represent the actual microarchitectural features of a design. You cannot make any inference about the actual sizes of caches based on these parameters.

Bits [63:56]

Reserved, RES0.

NumSets, bits [55:32]

(Number of sets in cache) - 1, therefore a value of 0 indicates 1 set in the cache. The number of sets does not have to be a power of 2.

Bits [31:24]

Reserved, RES0.

Associativity, bits [23:3]

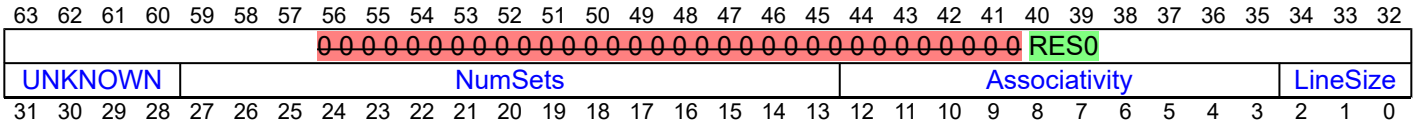
(Associativity of cache) - 1, therefore a value of 0 indicates an associativity of 1. The associativity does not have to be a power of 2.

LineSize, bits [2:0]

(Log_2(Number of bytes in cache line)) - 4. For example:

- For a line length of 16 bytes: $Log_2(16) = 4$, LineSize entry = 0. This is the minimum line length.
- For a line length of 32 bytes: $Log_2(32) = 5$, LineSize entry = 1.

Otherwise:



Note

The parameters NumSets, Associativity, and LineSize in these registers define the architecturally visible parameters that are required for the cache maintenance by Set/Way instructions. They are not guaranteed to represent the actual microarchitectural features of a design. You cannot make any inference about the actual sizes of caches based on these parameters.

Bits [63:32]

Reserved, RES0.

Bits~~UNKNOWN, bits~~ [31:28]

Reserved, UNKNOWN.

NumSets, bits [27:13]

(Number of sets in cache) - 1, therefore a value of 0 indicates 1 set in the cache. The number of sets does not have to be a power of 2.

Associativity, bits [12:3]

(Associativity of cache) - 1, therefore a value of 0 indicates an associativity of 1. The associativity does not have to be a power of 2.

LineSize, bits [2:0]

(Log_2(Number of bytes in cache line)) - 4. For example:

- For a line length of 16 bytes: $Log_2(16) = 4$, LineSize entry = 0. This is the minimum line length.
- For a line length of 32 bytes: $Log_2(32) = 5$, LineSize entry = 1.

Accessing the CCSIDR_EL1

If CSSELR_EL1.Level is programmed to a cache level that is not implemented, then on a read of the CCSIDR_EL1 the behavior is CONSTRAINED UNPREDICTABLE, and can be one of the following:

- The CCSIDR_EL1 read is treated as NOP.
- The CCSIDR_EL1 read is UNDEFINED.
- The CCSIDR_EL1 read returns an UNKNOWN value.

Accesses to this register use the following encodings:

```
MRS <Xt>, CCSIDR_EL1
```

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~

0b11	0b001	0b0000	0b0000	0b000
~~0b11~~	~~0b0000~~	~~0b001~~	~~0b000~~	~~0b0000~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TID2 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TID4 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        return CCSIDR_EL1;
elsif PSTATE.EL == EL2 then
    return CCSIDR_EL1;
elsif PSTATE.EL == EL3 then
    return CCSIDR_EL1;
```

27~~13~~/03~~12~~/2019~~2018~~ 21~~16~~:59~~42~~: e5e4db499bf9867a4b93324c4dbac985d3da93766~~379d01e197f1d40720d32d0f84c419e9187e009~~

(old) htmldiff from- (new)

(old) htmldiff from- (new)

CFP RCTX, Control Flow Prediction Restriction by Context

The CFP RCTX characteristics are:

Purpose

Control Flow Prediction Restriction by Context applies to all Control Flow Prediction Resources that predict execution based on information gathered within the target execution context or contexts.

When this instruction is complete and synchronized, control flow prediction does not permit later speculative execution within the target execution context to be observable through side channels.

This instruction is guaranteed to be complete following a DSB that covers both read and write behavior on the same PE as executed the original restriction instruction, and a subsequent context synchronization event is required to ensure that the effect of the completion of the instructions is synchronized to the current execution.

Note

This instruction does not require the invalidation of prediction structures so long as the behavior described for completion of this instruction is met by the implementation.

On some implementations the instruction is likely to take a significant number of cycles to execute. This instruction is expected to be used very rarely, such as on the roll-over of an ASID or VMID, but should not be used on every context switch.

Configuration

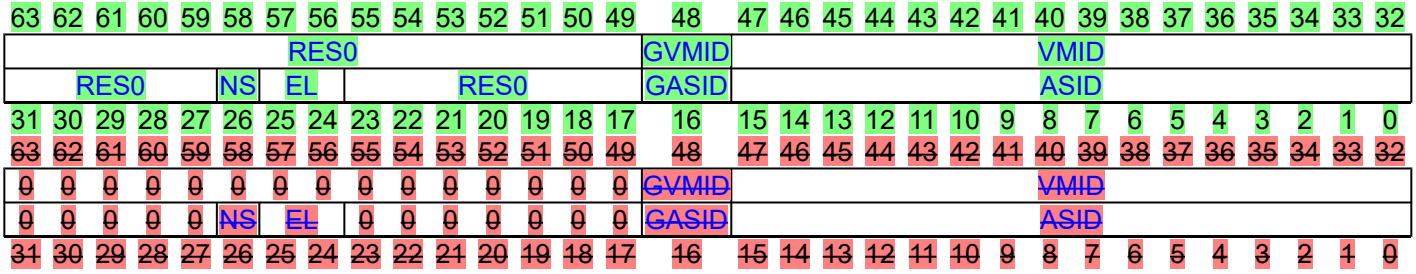
This instruction is present only when ARMv8.0-PredInv is implemented. Otherwise, direct accesses to CFP RCTX are UNDEFINED.

Attributes

CFP RCTX is a 64-bit System instruction.

Field descriptions

The CFP RCTX input value bit assignments are:



Bits [63:49]

Reserved, RES0.

GVMID, bit [48]

Execution of this instruction applies to all VMIDs or a specified VMID.

GVMID	Meaning
0b0	Applies to specified VMID for an EL0 or EL1 context. For all other contexts this field is RES0.
0b1	Applies to all VMIDs for an EL0 or EL1 context. For all other contexts this field is RES0.

If the instruction is executed at EL0 or EL1, then this field has an Effective value of 0.

VMID, bits [47:32]

Only applies when bit[48] is 0 and either:

- an EL1 context.
- an EL0 context when (HCR_EL2.E2H==0 or HCR_EL2.TGE==0).

Otherwise this field is RES0.

When the instruction is executed at EL1 then this field is treated as the current VMID.

When the instruction is executed at EL0 and (HCR_EL2.E2H==0 or HCR_EL2.TGE==0) then this field is treated as the current VMID.

When the instruction is executed at EL0 and (HCR_EL2.E2H==1 and HCR_EL2.TGE==1) then this field is ignored.

Bits [31:27]

Reserved, RES0.

NS, bit [26]

Security State

NS	Meaning
0b0	Secure state
0b1	Non-secure state

If the instruction is executed in Non-secure state, this field has an Effective value of 1.

EL, bits [25:24]

Exception Level

EL	Meaning
0b00	EL0.
0b01	EL1.
0b10	EL2.
0b11	EL3.

If the instruction is executed at an exception level lower than the specified level, this instruction is treated as a NOP.

Bits [23:17]

Reserved, RES0.

GASID, bit [16]

Execution of this instruction applies to all ASIDs or a specified ASID.

GASID	Meaning
0b0	Applies to specified ASID for an EL0 context. For all other contexts this field is RES0.
0b1	Applies to all ASID for an EL0 context. For all other contexts this field is RES0.

If the instruction is executed at EL0, then this field has an Effective value of 0.

ASID, bits [15:0]

Only applies for an EL0 context and when bit[16] is 0.

Otherwise this field is RES0.

When the instruction is executed at EL0 then this field is treated as the current ASID.

Executing the CFP RCTX instruction

Accesses to this instruction use the following encodings:

```
CFP RCTX, <Xt>
```

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b01	0b011	0b0111	0b0011	0b100
~~0b01~~	~~0b0111~~	~~0b011~~	~~0b100~~	~~0b0011~~

```
if PSTATE.EL == EL0 then
    if !ELUsingAArch32(EL1) && !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') && SCTLR_EL1.EnRCTX ==
'0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCTLR_EL2.EnRCTX ==
'0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        CFP_RCTX(X[t]);
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        CFP_RCTX(X[t]);
elsif PSTATE.EL == EL2 then
    CFP_RCTX(X[t]);
elsif PSTATE.EL == EL3 then
    CFP_RCTX(X[t]);
```

27~~13~~/03~~12~~/20192~~018~~ 21~~16~~:59~~42~~: e5e4db499bf9867a4b93324c4dbac985d3da93766379d01e197f1d40720d32d0f84e419e9187e009

(old) htmldiff from- (new)

CNTKCTL_EL1, Counter-timer Kernel Control register

The CNTKCTL_EL1 characteristics are:

Purpose

When ARMv8.1-VHE is not implemented, or when HCR_EL2.{E2H, TGE} is not {1, 1}, this register controls the generation of an event stream from the virtual counter, and access from EL0 to the physical counter, virtual counter, EL1 physical timers, and the virtual timer.

When ARMv8.1-VHE is implemented and HCR_EL2.{E2H, TGE} is {1, 1}, this register does not cause any event stream from the virtual counter to be generated, and does not control access to the counters and timers. The access to counters and timers at EL0 is controlled by CNTHCTL_EL2.

Configuration

AArch64 System register CNTKCTL_EL1 bits [31:0] are architecturally mapped to AArch32 System register CNTKCTL[31:0] .

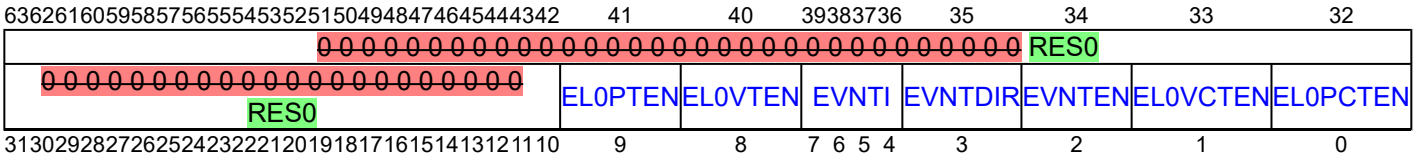
Some or all RW fields of this register have defined reset values. These apply only if the PE resets into an Exception level that is using AArch64. Otherwise, RW fields in this register reset to architecturally UNKNOWN values.

Attributes

CNTKCTL_EL1 is a 64-bit register.

Field descriptions

The CNTKCTL_EL1 bit assignments are:



Bits [63:10]

Reserved, RES0.

EL0PTEN, bit [9]

When ARMv8.1-VHE is not implemented, or when HCR_EL2.{E2H, TGE} is not {1, 1}, traps EL0 accesses to the physical timer registers to EL1.

EL0PTEN	Meaning
0b0	EL0 using AArch64: EL0 accesses to CNTP_CTL_EL0, CNTP_CVAL_EL0, and CNTP_TVAL_EL0 are trapped to EL1. EL0 using AArch32: EL0 accesses to CNTP_CTL, CNTP_CVAL, and CNTP_TVAL are trapped to EL1. When HCR_EL2.TGE is 1, this trap is routed to EL2.
0b1	This control does not cause any instructions to be trapped.

When ARMv8.1-VHE is implemented and HCR_EL2.{E2H, TGE} is {1, 1}, this control does not cause any instructions to be trapped.

This field resets to an architecturally UNKNOWN value.

EL0VTEN, bit [8]

When ARMv8.1-VHE is not implemented, or when HCR_EL2.{E2H, TGE} is not {1, 1}, traps EL0 accesses to the virtual timer registers to EL1.

EL0VTEN	Meaning
0b0	EL0 using AArch64: EL0 accesses to CNTV_CTL_EL0, CNTV_CVAL_EL0, and CNTV_TVAL_EL0 are trapped to EL1. EL0 using AArch32: EL0 accesses to CNTV_CTL, CNTV_CVAL, and CNTV_TVAL are trapped to EL1. When HCR_EL2.TGE is 1, this trap is routed to EL2.
0b1	This control does not cause any instructions to be trapped.

When ARMv8.1-VHE is implemented and HCR_EL2.{E2H, TGE} is {1, 1}, this control does not cause any instructions to be trapped.

This field resets to an architecturally UNKNOWN value.

EVNTI, bits [7:4]

Selects which bit (0 to 15) of the counter register CNTVCT_EL0 is the trigger for the event stream generated from that counter, when that stream is enabled.

This field resets to an architecturally UNKNOWN value.

EVNTDIR, bit [3]

Controls which transition of the counter register CNTVCT_EL0 trigger bit, defined by EVNTI, generates an event when the event stream is enabled:

EVNTDIR	Meaning
0b0	A 0 to 1 transition of the trigger bit triggers an event.
0b1	A 1 to 0 transition of the trigger bit triggers an event.

This field resets to an architecturally UNKNOWN value.

EVNTEN, bit [2]

When ARMv8.1-VHE is not implemented, or when HCR_EL2.{E2H, TGE} is not {1, 1}, enables the generation of an event stream from the counter register CNTVCT_EL0:

EVNTEN	Meaning
0b0	Disables the event stream.
0b1	Enables the event stream.

When ARMv8.1-VHE is implemented and HCR_EL2.{E2H, TGE} is {1, 1}, this control does not enable the event stream.

This field resets to 0.

EL0VCTEN, bit [1]

When ARMv8.1-VHE is not implemented, or when HCR_EL2.{E2H, TGE} is not {1, 1}, traps EL0 accesses to the frequency register and virtual counter register to EL1.

EL0VCTEN	Meaning
0b0	EL0 using AArch64: EL0 accesses to CNTVCT_EL0 are trapped to EL1. EL0 using AArch64: EL0 accesses to CNTFRQ_EL0 are trapped to EL1, if CNTKCTL_EL1.EL0PCTEN is also 0. EL0 using AArch32: EL0 accesses to CNTVCT are trapped to EL1. EL0 using AArch32: EL0 accesses to CNTFRQ are trapped to EL1, if CNTKCTL_EL1.EL0PCTEN is also 0. When HCR_EL2.TGE is 1, this trap is routed to EL2.
0b1	This control does not cause any instructions to be trapped.

When ARMv8.1-VHE is implemented and HCR_EL2.{E2H, TGE} is {1, 1}, this control does not cause any instructions to be trapped.

This field resets to an architecturally UNKNOWN value.

EL0PCTEN, bit [0]

When ARMv8.1-VHE is not implemented, or when HCR_EL2.{E2H, TGE} is not {1, 1}, traps EL0 accesses to the frequency register and physical counter register to EL1.

EL0PCTEN	Meaning
0b0	EL0 using AArch64: EL0 accesses to CNTPCT_EL0 are trapped to EL1. EL0 using AArch64: EL0 accesses to CNTFRQ_EL0 are trapped to EL1, if CNTKCTL_EL1.EL0VCTEN is also 0. EL0 using AArch32: EL0 accesses to CNTPCT are trapped to EL1. EL0 using AArch32: EL0 accesses to CNTFRQ are trapped to EL1, if CNTKCTL_EL1.EL0VCTEN is also 0. When HCR_EL2.TGE is 1, this trap is routed to EL2.
0b1	This control does not cause any instructions to be trapped.

When ARMv8.1-VHE is implemented and HCR_EL2.{E2H, TGE} is {1, 1}, this control does not cause any instructions to be trapped.

This field resets to an architecturally UNKNOWN value.

Accessing the CNTKCTL_EL1

When HCR_EL2.E2H is 1, without explicit synchronization, access from EL3 using the mnemonic CNTKCTL_EL1 or CNTKCTL_EL12 are not guaranteed to be ordered with respect to accesses using the other mnemonic.

Accesses to this register use the following encodings:

```
MRS <Xt>, CNTKCTL_EL1
```

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b11	0b000	0b1110	0b0001	0b000
~~0b11~~	~~0b1110~~	~~0b000~~	~~0b000~~	~~0b0001~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    return CNTKCTL_EL1;
elsif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' then
        return CNTHCTL_EL2;
    else
        return CNTKCTL_EL1;
elsif PSTATE.EL == EL3 then
    return CNTKCTL_EL1;
```

```
MSR CNTKCTL_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b11	0b000	0b1110	0b0001	0b000
~~0b11~~	~~0b1110~~	~~0b000~~	~~0b000~~	~~0b0001~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    CNTKCTL_EL1 = X[t];
elsif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' then
        CNTHCTL_EL2 = X[t];
    else
        CNTKCTL_EL1 = X[t];
elsif PSTATE.EL == EL3 then
    CNTKCTL_EL1 = X[t];
```

MRS <Xt>, CNTKCTL_EL12

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b11	0b101	0b1110	0b0001	0b000
~~0b11~~	~~0b1110~~	~~0b101~~	~~0b000~~	~~0b0001~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if EL2Enabled() && HCR_EL2.E2H == '1' then
    if PSTATE.EL == EL2 then
        return CNTKCTL_EL1;
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if EL2Enabled() && HCR_EL2.E2H == '1' then
        return CNTKCTL_EL1;
else
    else
        UNDEFINED;
```

MSR CNTKCTL_EL12, <Xt>

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b11	0b101	0b1110	0b0001	0b000
~~0b11~~	~~0b1110~~	~~0b101~~	~~0b000~~	~~0b0001~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if EL2Enabled() && HCR_EL2.E2H == '1' then
    if PSTATE.EL == EL2 then
        CNTKCTL_EL1 = X[t];
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if EL2Enabled() && HCR_EL2.E2H == '1' then
        CNTKCTL_EL1 = X[t];
else
    else
        UNDEFINED;
```

27~~13~~/03~~12~~/2019~~2018~~ 21~~16~~:59~~42~~; e5e4db499bf9867a4b93324c4dbac985d3da93766379d01e197f1d40720d32d0f84c419e9187e009

(old) htmldiff from- (new)

(old) htmldiff from- (new)

CNTP_CTL_EL0, Counter-timer Physical Timer Control register

The CNTP_CTL_EL0 characteristics are:

Purpose

Control register for the EL1 physical timer.

Configuration

AArch64 System register CNTP_CTL_EL0 bits [31:0] are architecturally mapped to AArch32 System register CNTP_CTL[31:0] .

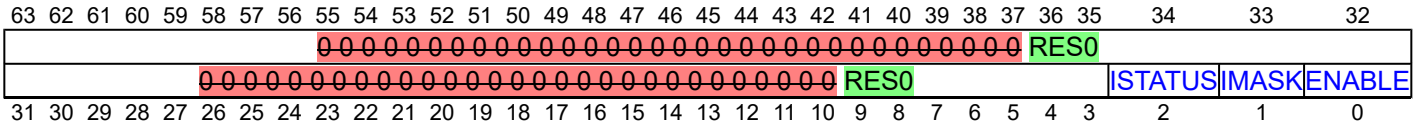
RW fields in this register reset to architecturally UNKNOWN values.

Attributes

CNTP_CTL_EL0 is a 64-bit register.

Field descriptions

The CNTP_CTL_EL0 bit assignments are:



Bits [63:3]

Reserved, RES0.

ISTATUS, bit [2]

The status of the timer. This bit indicates whether the timer condition is met:

ISTATUS	Meaning
0b0	Timer condition is not met.
0b1	Timer condition is met.

When the value of the ENABLE bit is 1, ISTATUS indicates whether the timer condition is met. ISTATUS takes no account of the value of the IMASK bit. If the value of ISTATUS is 1 and the value of IMASK is 0 then the timer interrupt is asserted.

When the value of the ENABLE bit is 0, the ISTATUS field is UNKNOWN.

For more information see 'Operation of the CompareValue views of the timers' and 'Operation of the TimerValue views of the timers' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, chapter D6.

This bit is read-only.

IMASK, bit [1]

Timer interrupt mask bit. Permitted values are:

IMASK	Meaning
0b0	Timer interrupt is not masked by the IMASK bit.
0b1	Timer interrupt is masked by the IMASK bit.

For more information, see the description of the ISTATUS bit.

This field resets to an architecturally UNKNOWN value.

ENABLE, bit [0]

Enables the timer. Permitted values are:

ENABLE	Meaning
0b0	Timer disabled.
0b1	Timer enabled.

Setting this bit to 0 disables the timer output signal, but the timer value accessible from CNTP_TVAL_EL0 continues to count down.

Note

Disabling the output signal might be a power-saving option.

This field resets to an architecturally UNKNOWN value.

Accessing the CNTP_CTL_EL0

When HCR_EL2.E2H is 1, without explicit synchronization, access from EL3 using the mnemonic CNTP_CTL_EL0 or CNTP_CTL_EL02 are not guaranteed to be ordered with respect to accesses using the other mnemonic.

Accesses to this register use the following encodings:

```
MRS <Xt>, CNTP_CTL_EL0
```

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b11	0b011	0b1110	0b0010	0b001
~~0b11~~	~~0b1110~~	~~0b011~~	~~0b001~~	~~0b0010~~

```
if PSTATE.EL == EL0 then
    if !ELUsingAArch32(EL1) && !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') && CNTKCTL_EL1.EL0PTEN
== '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '0' && CNTHCTL_EL2.EL1PCEN == '0'
then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '10' && CNTHCTL_EL2.EL1PTEN
== '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && CNTHCTL_EL2.EL0PTEN
== '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '0'
then
        return CNTHPS_CTL_EL2;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '1'
then
        return CNTHP_CTL_EL2;
    else
        return CNTP_CTL_EL0;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '0' && CNTHCTL_EL2.EL1PCEN == '0'
then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '1' && CNTHCTL_EL2.EL1PTEN == '0'
then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<NV2,NV1,NV> == '111' then
        return NVMem[0x180];
    else
        return CNTP_CTL_EL0;
elsif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' && SCR_EL3.NS == '0' then
        return CNTHPS_CTL_EL2;
    elsif HCR_EL2.E2H == '1' && SCR_EL3.NS == '1' then
        return CNTHP_CTL_EL2;
    else
        return CNTP_CTL_EL0;
elsif PSTATE.EL == EL3 then
    return CNTP_CTL_EL0;
```

```
MSR CNTP_CTL_EL0, <Xt>
```

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b11	0b011	0b1110	0b0010	0b001
~~0b11~~	~~0b1110~~	~~0b011~~	~~0b001~~	~~0b0010~~

```
if PSTATE.EL == EL0 then
    if !ELUsingAArch32(EL1) && !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') && CNTKCTL_EL1.EL0PTEN
== '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '0' && CNTHCTL_EL2.EL1PCEN == '0'
then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '10' && CNTHCTL_EL2.EL1PTEN
== '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && CNTHCTL_EL2.EL0PTEN
== '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '0'
then
        CNTHPS_CTL_EL2 = X[t];
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '1'
then
        CNTHP_CTL_EL2 = X[t];
    else
        CNTP_CTL_EL0 = X[t];
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '0' && CNTHCTL_EL2.EL1PCEN == '0'
then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '1' && CNTHCTL_EL2.EL1PTEN == '0'
then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<NV2,NV1,NV> == '111' then
        NVMem[0x180] = X[t];
    else
        CNTP_CTL_EL0 = X[t];
elsif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' && SCR_EL3.NS == '0' then
        CNTHPS_CTL_EL2 = X[t];
    elsif HCR_EL2.E2H == '1' && SCR_EL3.NS == '1' then
        CNTHP_CTL_EL2 = X[t];
    else
        CNTP_CTL_EL0 = X[t];
elsif PSTATE.EL == EL3 then
    CNTP_CTL_EL0 = X[t];
```

MRS <Xt>, CNTP_CTL_EL02

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b11	0b101	0b1110	0b0010	0b001
~~0b11~~	~~0b1110~~	~~0b101~~	~~0b001~~	~~0b0010~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '101' then
        return NVMem[0x180];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if EL2Enabled() && HCR_EL2.E2H == '1' then
    if PSTATE.EL == EL2 then
        return CNTP_CTL_EL0;
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if EL2Enabled() && HCR_EL2.E2H == '1' then
        return CNTP_CTL_EL0;
else
    else
        UNDEFINED;
```

MSR CNTP_CTL_EL02, <Xt>

op0	op1	CRn	CRm	op2
op0	CRn	op1	op2	CRm
0b11	0b101	0b1110	0b0010	0b001
0b11	0b1110	0b101	0b001	0b0010

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '101' then
        NVMem[0x180] = X[t];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if EL2Enabled() && HCR_EL2.E2H == '1' then
    if PSTATE.EL == EL2 then
        CNTP_CTL_EL0 = X[t];
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if EL2Enabled() && HCR_EL2.E2H == '1' then
        CNTP_CTL_EL0 = X[t];
else
    else
        UNDEFINED;
```

27 13/03 12/2019 2018 21 16:59 42; e5e4db499bf9867a4b93324c4dbac985d3da93766379d01e197f1d40720d32d0f84e419e9187e009

(old) htmldiff from- (new)

CNTP_CVAL_EL0, Counter-timer Physical Timer CompareValue register

The CNTP_CVAL_EL0 characteristics are:

Purpose

Holds the compare value for the EL1 physical timer.

Configuration

AArch64 System register CNTP_CVAL_EL0 bits [63:0] are architecturally mapped to AArch32 System register CNTP_CVAL[63:0] .

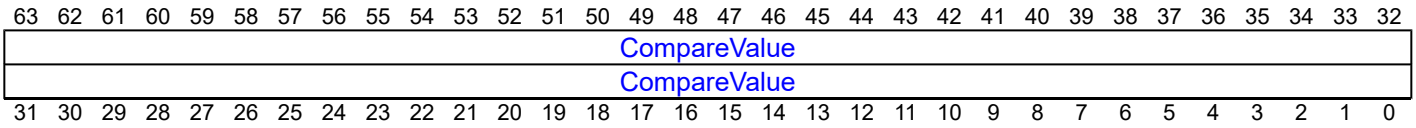
RW fields in this register reset to architecturally UNKNOWN values.

Attributes

CNTP_CVAL_EL0 is a 64-bit register.

Field descriptions

The CNTP_CVAL_EL0 bit assignments are:



CompareValue, bits [63:0]

Holds the EL1 physical timer CompareValue.

When CNTP_CTL_EL0.ENABLE is 1, the timer condition is met when (CNTPCT_EL0 - CompareValue) is greater than or equal to zero. This means that CompareValue acts like a 64-bit upcounter timer. When the timer condition is met:

- CNTP_CTL_EL0.ISTATUS is set to 1.
- If CNTP_CTL_EL0.IMASK is 0, an interrupt is generated.

When CNTP_CTL_EL0.ENABLE is 0, the timer condition is not met, but CNTPCT_EL0 continues to count.

This field resets to an architecturally UNKNOWN value.

Accessing the CNTP_CVAL_EL0

When HCR_EL2.E2H is 1, without explicit synchronization, access from EL3 using the mnemonic CNTP_CVAL_EL0 or CNTP_CVAL_EL02 are not guaranteed to be ordered with respect to accesses using the other mnemonic.

Accesses to this register use the following encodings:

```
MRS <Xt>, CNTP_CVAL_EL0
```

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b11	0b011	0b1110	0b0010	0b010

~~0b11~~	~~0b1110~~	~~0b011~~	~~0b010~~	~~0b0010~~

```
if PSTATE.EL == EL0 then
    if !ELUsingAArch32(EL1) && !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') && CNTKCTL_EL1.EL0PTEN
== '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '0' && CNTHCTL_EL2.EL1PCEN == '0'
then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '10' && CNTHCTL_EL2.EL1PTEN
== '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && CNTHCTL_EL2.EL0PTEN
== '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '0'
then
        return CNTHPS_CVAL_EL2;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '1'
then
        return CNTHP_CVAL_EL2;
    else
        return CNTP_CVAL_EL0;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '0' && CNTHCTL_EL2.EL1PCEN == '0'
then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '1' && CNTHCTL_EL2.EL1PTEN == '0'
then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<NV2,NV1,NV> == '111' then
        return NVMem[0x178];
    else
        return CNTP_CVAL_EL0;
elsif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' && SCR_EL3.NS == '0' then
        return CNTHPS_CVAL_EL2;
    elsif HCR_EL2.E2H == '1' && SCR_EL3.NS == '1' then
        return CNTHP_CVAL_EL2;
    else
        return CNTP_CVAL_EL0;
elsif PSTATE.EL == EL3 then
    return CNTP_CVAL_EL0;
```

```
MSR CNTP_CVAL_EL0, <Xt>
```

op0	**op1**	**CRn**	**CRm**	**op2**
~~**op0**~~	~~**CRn**~~	~~**op1**~~	~~**op2**~~	~~**CRm**~~
0b11	0b011	0b1110	0b0010	0b010
~~0b11~~	~~0b1110~~	~~0b011~~	~~0b010~~	~~0b0010~~

```
if PSTATE.EL == EL0 then
    if !ELUsingAArch32(EL1) && !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') && CNTKCTL_EL1.EL0PTEN
== '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '0' && CNTHCTL_EL2.EL1PCEN == '0'
then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '10' && CNTHCTL_EL2.EL1PTEN
== '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && CNTHCTL_EL2.EL0PTEN
== '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '0'
then
        CNTHPS_CVAL_EL2 = X[t];
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '1'
then
        CNTHP_CVAL_EL2 = X[t];
    else
        CNTP_CVAL_EL0 = X[t];
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '0' && CNTHCTL_EL2.EL1PCEN == '0'
then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '1' && CNTHCTL_EL2.EL1PTEN == '0'
then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<NV2,NV1,NV> == '111' then
        NVMem[0x178] = X[t];
    else
        CNTP_CVAL_EL0 = X[t];
elsif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' && SCR_EL3.NS == '0' then
        CNTHPS_CVAL_EL2 = X[t];
    elsif HCR_EL2.E2H == '1' && SCR_EL3.NS == '1' then
        CNTHP_CVAL_EL2 = X[t];
    else
        CNTP_CVAL_EL0 = X[t];
elsif PSTATE.EL == EL3 then
    CNTP_CVAL_EL0 = X[t];
```

MRS <Xt>, CNTP_CVAL_EL02

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b11	0b101	0b1110	0b0010	0b010
~~0b11~~	~~0b1110~~	~~0b101~~	~~0b010~~	~~0b0010~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '101' then
        return NVMem[0x178];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if EL2Enabled() && HCR_EL2.E2H == '1' then
    if PSTATE.EL == EL2 then
        return CNTP_CVAL_EL0;
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if EL2Enabled() && HCR_EL2.E2H == '1' then
        return CNTP_CVAL_EL0;
else
    else
        UNDEFINED;
```

MSR CNTP_CVAL_EL02, <Xt>

op0	op1	CRn	CRm	op2
op0	CRn	op1	op2	CRm
0b11	0b101	0b1110	0b0010	0b010
0b11	0b1110	0b101	0b010	0b0010

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '101' then
        NVMem[0x178] = X[t];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if EL2Enabled() && HCR_EL2.E2H == '1' then
    if PSTATE.EL == EL2 then
        CNTP_CVAL_EL0 = X[t];
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if EL2Enabled() && HCR_EL2.E2H == '1' then
        CNTP_CVAL_EL0 = X[t];
else
    else
        UNDEFINED;
```

27 13/03 12/2019 2018 21 16:59 42; e5e4db499bf9867a4b93324c4dbac985d3da93766379d01e197f1d40720d32d0f84e419e9187e009

CNTP_TVAL_EL0, Counter-timer Physical Timer TimerValue register

The CNTP_TVAL_EL0 characteristics are:

Purpose

Holds the timer value for the EL1 physical timer.

Configuration

AArch64 System register CNTP_TVAL_EL0 bits [31:0] are architecturally mapped to AArch32 System register CNTP_TVAL[31:0] .

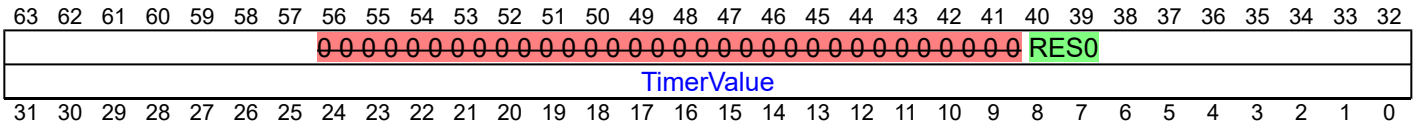
RW fields in this register reset to architecturally UNKNOWN values.

Attributes

CNTP_TVAL_EL0 is a 64-bit register.

Field descriptions

The CNTP_TVAL_EL0 bit assignments are:



Bits [63:32]

Reserved, RES0.

TimerValue, bits [31:0]

The TimerValue view of the EL1 physical timer.

On a read of this register:

- If CNTP_CTL_EL0.ENABLE is 0, the value returned is UNKNOWN.
- If CNTP_CTL_EL0.ENABLE is 1, the value returned is (CNTP_CVAL_EL0 - CNTPCT_EL0).

On a write of this register, CNTP_CVAL_EL0 is set to (CNTPCT_EL0 + TimerValue), where TimerValue is treated as a signed 32-bit integer.

When CNTP_CTL_EL0.ENABLE is 1, the timer condition is met when (CNTPCT_EL0 - CNTP_CVAL_EL0) is greater than or equal to zero. This means that TimerValue acts like a 32-bit downcounter timer. When the timer condition is met:

- CNTP_CTL_EL0.ISTATUS is set to 1.
- If CNTP_CTL_EL0.IMASK is 0, an interrupt is generated.

When CNTP_CTL_EL0.ENABLE is 0, the timer condition is not met, but CNTPCT_EL0 continues to count, so the TimerValue view appears to continue to count down.

This field resets to an architecturally UNKNOWN value.

Accessing the CNTP_TVAL_EL0

When HCR_EL2.E2H is 1, without explicit synchronization, access from EL3 using the mnemonic CNTP_TVAL_EL0 or CNTP_TVAL_EL02 are not guaranteed to be ordered with respect to accesses using the other mnemonic.

Accesses to this register use the following encodings:

```
MRS <Xt>, CNTP_TVAL_EL0
```

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b11	0b011	0b1110	0b0010	0b000
~~0b11~~	~~0b1110~~	~~0b011~~	~~0b000~~	~~0b0010~~

```
if PSTATE.EL == EL0 then
    if !ELUsingAArch32(EL1) && !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') && CNTKCTL_EL1.EL0PTEN
== '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '0' && CNTHCTL_EL2.EL1PCEN == '0'
then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '10' && CNTHCTL_EL2.EL1PTEN
== '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && CNTHCTL_EL2.EL0PTEN
== '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '0'
then
        return CNTHPS_TVAL_EL2;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '1'
then
        return CNTHP_TVAL_EL2;
    else
        return CNTP_TVAL_EL0;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '0' && CNTHCTL_EL2.EL1PCEN == '0'
then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '1' && CNTHCTL_EL2.EL1PTEN == '0'
then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        return CNTP_TVAL_EL0;
elsif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' && SCR_EL3.NS == '0' then
        return CNTHPS_TVAL_EL2;
    elsif HCR_EL2.E2H == '1' && SCR_EL3.NS == '1' then
        return CNTHP_TVAL_EL2;
    else
        return CNTP_TVAL_EL0;
elsif PSTATE.EL == EL3 then
    return CNTP_TVAL_EL0;
```

```
MSR CNTP_TVAL_EL0, <Xt>
```

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b11	0b011	0b1110	0b0010	0b000
~~0b11~~	~~0b1110~~	~~0b011~~	~~0b000~~	~~0b0010~~

```
if PSTATE.EL == EL0 then
    if !ELUsingAArch32(EL1) && !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') && CNTKCTL_EL1.EL0PTEN
== '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '0' && CNTHCTL_EL2.EL1PCEN == '0'
then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '10' && CNTHCTL_EL2.EL1PTEN
== '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && CNTHCTL_EL2.EL0PTEN
== '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '0'
then
        CNTHPS_TVAL_EL2 = X[t];
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '1'
then
        CNTHP_TVAL_EL2 = X[t];
    else
        CNTP_TVAL_EL0 = X[t];
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '0' && CNTHCTL_EL2.EL1PCEN == '0'
then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '1' && CNTHCTL_EL2.EL1PTEN == '0'
then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        CNTP_TVAL_EL0 = X[t];
elsif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' && SCR_EL3.NS == '0' then
        CNTHPS_TVAL_EL2 = X[t];
    elsif HCR_EL2.E2H == '1' && SCR_EL3.NS == '1' then
        CNTHP_TVAL_EL2 = X[t];
    else
        CNTP_TVAL_EL0 = X[t];
elsif PSTATE.EL == EL3 then
    CNTP_TVAL_EL0 = X[t];
```

MRS <Xt>, CNTP_TVAL_EL02

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b11	0b101	0b1110	0b0010	0b000
~~0b11~~	~~0b1110~~	~~0b101~~	~~0b000~~	~~0b0010~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if EL2Enabled() && HCR_EL2.E2H == '1' then
    if PSTATE.EL == EL2 then
        return CNTP_TVAL_EL0;
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if EL2Enabled() && HCR_EL2.E2H == '1' then
        return CNTP_TVAL_EL0;
else
    else
        UNDEFINED;
```

MSR CNTP_TVAL_EL02, <Xt>

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b11	0b101	0b1110	0b0010	0b000
~~0b11~~	~~0b1110~~	~~0b101~~	~~0b000~~	~~0b0010~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if EL2Enabled() && HCR_EL2.E2H == '1' then
    if PSTATE.EL == EL2 then
        CNTP_TVAL_EL0 = X[t];
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if EL2Enabled() && HCR_EL2.E2H == '1' then
        CNTP_TVAL_EL0 = X[t];
else
    else
        UNDEFINED;
```

27~~13~~/03~~12~~/2019~~2018~~ 21~~16~~:59~~42~~: e5e4db499bf9867a4b93324c4dbac985d3da93766379d01e197f1d40720d32d0f84e419e9187e009

(old) htmldiff from- (new)

CNTV_CTL_EL0, Counter-timer Virtual Timer Control register

The CNTV_CTL_EL0 characteristics are:

Purpose

Control register for the virtual timer.

Configuration

AArch64 System register CNTV_CTL_EL0 bits [31:0] are architecturally mapped to AArch32 System register CNTV_CTL[31:0] .

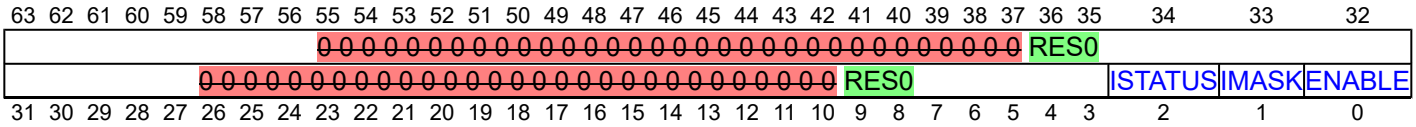
RW fields in this register reset to architecturally UNKNOWN values.

Attributes

CNTV_CTL_EL0 is a 64-bit register.

Field descriptions

The CNTV_CTL_EL0 bit assignments are:



Bits [63:3]

Reserved, RES0.

ISTATUS, bit [2]

The status of the timer. This bit indicates whether the timer condition is met:

ISTATUS	Meaning
0b0	Timer condition is not met.
0b1	Timer condition is met.

When the value of the ENABLE bit is 1, ISTATUS indicates whether the timer condition is met. ISTATUS takes no account of the value of the IMASK bit. If the value of ISTATUS is 1 and the value of IMASK is 0 then the timer interrupt is asserted.

When the value of the ENABLE bit is 0, the ISTATUS field is UNKNOWN.

For more information see 'Operation of the CompareValue views of the timers' and 'Operation of the TimerValue views of the timers' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, chapter D6.

This bit is read-only.

IMASK, bit [1]

Timer interrupt mask bit. Permitted values are:

IMASK	Meaning
0b0	Timer interrupt is not masked by the IMASK bit.
0b1	Timer interrupt is masked by the IMASK bit.

For more information, see the description of the ISTATUS bit.

This field resets to an architecturally UNKNOWN value.

ENABLE, bit [0]

Enables the timer. Permitted values are:

ENABLE	Meaning
0b0	Timer disabled.
0b1	Timer enabled.

Setting this bit to 0 disables the timer output signal, but the timer value accessible from CNTV_TVAL_EL0 continues to count down.

Note

Disabling the output signal might be a power-saving option.

This field resets to an architecturally UNKNOWN value.

Accessing the CNTV_CTL_EL0

When HCR_EL2.E2H is 1, without explicit synchronization, access from EL3 using the mnemonic CNTV_CTL_EL0 or CNTV_CTL_EL02 are not guaranteed to be ordered with respect to accesses using the other mnemonic.

Accesses to this register use the following encodings:

```
MRS <Xt>, CNTV_CTL_EL0
```

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b11	0b011	0b1110	0b0011	0b001
~~0b11~~	~~0b1110~~	~~0b011~~	~~0b001~~	~~0b0011~~

```
if PSTATE.EL == EL0 then
    if !ELUsingAArch32(EL1) && !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') && CNTKCTL_EL1.EL0VTEN
== '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && CNTHCTL_EL2.EL0VTEN
== '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '0'
then
        return CNTHVS_CTL_EL2;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '1'
then
        return CNTHV_CTL_EL2;
    else
        return CNTV_CTL_EL0;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<NV2,NV1,NV> == '111' then
        return NVMem[0x170];
    else
        return CNTV_CTL_EL0;
elsif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' && SCR_EL3.NS == '0' then
        return CNTHVS_CTL_EL2;
    elsif HCR_EL2.E2H == '1' && SCR_EL3.NS == '1' then
        return CNTHV_CTL_EL2;
    else
        return CNTV_CTL_EL0;
elsif PSTATE.EL == EL3 then
    return CNTV_CTL_EL0;
```

```
MSR CNTV_CTL_EL0, <Xt>
```

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b11	0b011	0b1110	0b0011	0b001
~~0b11~~	~~0b1110~~	~~0b011~~	~~0b001~~	~~0b0011~~

```
if PSTATE.EL == EL0 then
    if !ELUsingAArch32(EL1) && !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') && CNTKCTL_EL1.EL0VTEN
== '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && CNTHCTL_EL2.EL0VTEN
== '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '0'
then
        CNTHVS_CTL_EL2 = X[t];
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '1'
then
        CNTHV_CTL_EL2 = X[t];
    else
        CNTV_CTL_EL0 = X[t];
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<NV2,NV1,NV> == '111' then
        NVMem[0x170] = X[t];
    else
        CNTV_CTL_EL0 = X[t];
elsif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' && SCR_EL3.NS == '0' then
        CNTHVS_CTL_EL2 = X[t];
    elsif HCR_EL2.E2H == '1' && SCR_EL3.NS == '1' then
        CNTHV_CTL_EL2 = X[t];
    else
        CNTV_CTL_EL0 = X[t];
elsif PSTATE.EL == EL3 then
    CNTV_CTL_EL0 = X[t];
```

MRS <Xt>, CNTV_CTL_EL02

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b11	0b101	0b1110	0b0011	0b001
~~0b11~~	~~0b1110~~	~~0b101~~	~~0b001~~	~~0b0011~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '101' then
        return NVMem[0x170];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if EL2Enabled() && HCR_EL2.E2H == '1' then
    if PSTATE.EL == EL2 then
        return CNTV_CTL_EL0;
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if EL2Enabled() && HCR_EL2.E2H == '1' then
        return CNTV_CTL_EL0;
else
    else
        UNDEFINED;
```

```
MSR CNTV_CTL_EL02, <Xt>
```

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b11	0b101	0b1110	0b0011	0b001
~~0b11~~	~~0b1110~~	~~0b101~~	~~0b001~~	~~0b0011~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '101' then
        NVMem[0x170] = X[t];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if EL2Enabled() && HCR_EL2.E2H == '1' then
    if PSTATE.EL == EL2 then
        CNTV_CTL_EL0 = X[t];
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if EL2Enabled() && HCR_EL2.E2H == '1' then
        CNTV_CTL_EL0 = X[t];
else
    else
        UNDEFINED;
```


(old) htmldiff from- (new)

(old) htmldiff from- (new)

CNTV_CVAL_EL0, Counter-timer Virtual Timer CompareValue register

The CNTV_CVAL_EL0 characteristics are:

Purpose

Holds the compare value for the virtual timer.

Configuration

AArch64 System register CNTV_CVAL_EL0 bits [63:0] are architecturally mapped to AArch32 System register CNTV_CVAL[63:0] .

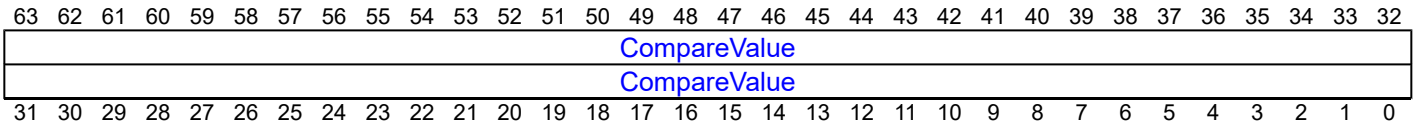
RW fields in this register reset to architecturally UNKNOWN values.

Attributes

CNTV_CVAL_EL0 is a 64-bit register.

Field descriptions

The CNTV_CVAL_EL0 bit assignments are:



CompareValue, bits [63:0]

Holds the EL1 virtual timer CompareValue.

When CNTV_CTL_EL0.ENABLE is 1, the timer condition is met when (CNTVCT_EL0 - CompareValue) is greater than or equal to zero. This means that CompareValue acts like a 64-bit upcounter timer. When the timer condition is met:

- CNTV_CTL_EL0.ISTATUS is set to 1.
- If CNTV_CTL_EL0.IMASK is 0, an interrupt is generated.

When CNTV_CTL_EL0.ENABLE is 0, the timer condition is not met, but CNTVCT_EL0 continues to count.

This field resets to an architecturally UNKNOWN value.

Accessing the CNTV_CVAL_EL0

When HCR_EL2.E2H is 1, without explicit synchronization, access from EL3 using the mnemonic CNTV_CVAL_EL0 or CNTV_CVAL_EL02 are not guaranteed to be ordered with respect to accesses using the other mnemonic.

Accesses to this register use the following encodings:

```
MRS <Xt>, CNTV_CVAL_EL0
```

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b11	0b011	0b1110	0b0011	0b010

0b11	0b1110	0b011	0b010	0b0011

```
if PSTATE.EL == EL0 then
    if !ELUsingAArch32(EL1) && !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') && CNTKCTL_EL1.EL0VTEN
== '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && CNTHCTL_EL2.EL0VTEN
== '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '0'
then
        return CNTHVS_CVAL_EL2;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '1'
then
        return CNTHV_CVAL_EL2;
    else
        return CNTV_CVAL_EL0;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<NV2,NV1,NV> == '111' then
        return NVMem[0x168];
    else
        return CNTV_CVAL_EL0;
elsif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' && SCR_EL3.NS == '0' then
        return CNTHVS_CVAL_EL2;
    elsif HCR_EL2.E2H == '1' && SCR_EL3.NS == '1' then
        return CNTHV_CVAL_EL2;
    else
        return CNTV_CVAL_EL0;
elsif PSTATE.EL == EL3 then
    return CNTV_CVAL_EL0;
```

```
MSR CNTV_CVAL_EL0, <Xt>
```

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b11	0b011	0b1110	0b0011	0b010
~~0b11~~	~~0b1110~~	~~0b011~~	~~0b010~~	~~0b0011~~

```
if PSTATE.EL == EL0 then
    if !ELUsingAArch32(EL1) && !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') && CNTKCTL_EL1.EL0VTEN
== '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && CNTHCTL_EL2.EL0VTEN
== '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '0'
then
        CNTHVS_CVAL_EL2 = X[t];
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '1'
then
        CNTHV_CVAL_EL2 = X[t];
    else
        CNTV_CVAL_EL0 = X[t];
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<NV2,NV1,NV> == '111' then
        NVMem[0x168] = X[t];
    else
        CNTV_CVAL_EL0 = X[t];
elsif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' && SCR_EL3.NS == '0' then
        CNTHVS_CVAL_EL2 = X[t];
    elsif HCR_EL2.E2H == '1' && SCR_EL3.NS == '1' then
        CNTHV_CVAL_EL2 = X[t];
    else
        CNTV_CVAL_EL0 = X[t];
elsif PSTATE.EL == EL3 then
    CNTV_CVAL_EL0 = X[t];
```

MRS <Xt>, CNTV_CVAL_EL02

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b11	0b101	0b1110	0b0011	0b010
~~0b11~~	~~0b1110~~	~~0b101~~	~~0b010~~	~~0b0011~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '101' then
        return NVMem[0x168];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if EL2Enabled() && HCR_EL2.E2H == '1' then
    if PSTATE.EL == EL2 then
        return CNTV_CVAL_EL0;
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if EL2Enabled() && HCR_EL2.E2H == '1' then
        return CNTV_CVAL_EL0;
else
    else
        UNDEFINED;
```

```
MSR CNTV_CVAL_EL02, <Xt>
```

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b11	0b101	0b1110	0b0011	0b010
~~0b11~~	~~0b1110~~	~~0b101~~	~~0b010~~	~~0b0011~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '101' then
        NVMem[0x168] = X[t];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if EL2Enabled() && HCR_EL2.E2H == '1' then
    if PSTATE.EL == EL2 then
        CNTV_CVAL_EL0 = X[t];
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if EL2Enabled() && HCR_EL2.E2H == '1' then
        CNTV_CVAL_EL0 = X[t];
else
    else
        UNDEFINED;
```

27~~13~~/03~~12~~/2019~~2018~~ 21~~16~~:59~~42~~; e5e4db499bf9867a4b93324c4dbac985d3da93766~~379d01e197f1d40720d32d0f84e419e9187e009~~

CNTV_TVAL_EL0, Counter-timer Virtual Timer TimerValue register

The CNTV_TVAL_EL0 characteristics are:

Purpose

Holds the timer value for the EL1 virtual timer.

Configuration

AArch64 System register CNTV_TVAL_EL0 bits [31:0] are architecturally mapped to AArch32 System register CNTV_TVAL[31:0] .

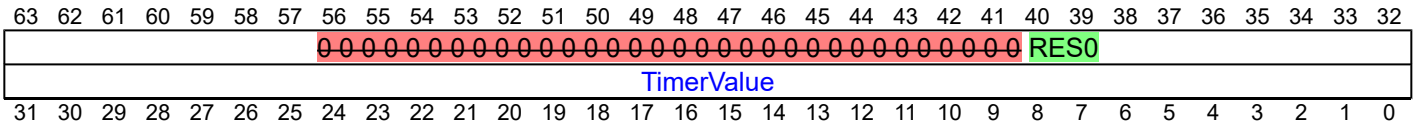
RW fields in this register reset to architecturally UNKNOWN values.

Attributes

CNTV_TVAL_EL0 is a 64-bit register.

Field descriptions

The CNTV_TVAL_EL0 bit assignments are:



Bits [63:32]

Reserved, RES0.

TimerValue, bits [31:0]

The TimerValue view of the EL1 virtual timer.

On a read of this register:

- If CNTV_CTL_EL0.ENABLE is 0, the value returned is UNKNOWN.
- If CNTV_CTL_EL0.ENABLE is 1, the value returned is (CNTV_CVAL_EL0 - CNTVCT_EL0).

On a write of this register, CNTV_CVAL_EL0 is set to (CNTVCT_EL0 + TimerValue), where TimerValue is treated as a signed 32-bit integer.

When CNTV_CTL_EL0.ENABLE is 1, the timer condition is met when (CNTVCT_EL0 - CNTV_CVAL_EL0) is greater than or equal to zero. This means that TimerValue acts like a 32-bit downcounter timer. When the timer condition is met:

- CNTV_CTL_EL0.ISTATUS is set to 1.
- If CNTV_CTL_EL0.IMASK is 0, an interrupt is generated.

When CNTV_CTL_EL0.ENABLE is 0, the timer condition is not met, but CNTVCT_EL0 continues to count, so the TimerValue view appears to continue to count down.

This field resets to an architecturally UNKNOWN value.

Accessing the CNTV_TVAL_EL0

When HCR_EL2.E2H is 1, without explicit synchronization, access from EL3 using the mnemonic CNTV_TVAL_EL0 or CNTV_TVAL_EL02 are not guaranteed to be ordered with respect to accesses using the other mnemonic.

Accesses to this register use the following encodings:

```
MRS <Xt>, CNTV_TVAL_EL0
```

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b11	0b011	0b1110	0b0011	0b000
~~0b11~~	~~0b1110~~	~~0b011~~	~~0b000~~	~~0b0011~~

```
if PSTATE.EL == EL0 then
    if !ELUsingAArch32(EL1) && !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') && CNTKCTL_EL1.EL0VTEN
== '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && CNTHCTL_EL2.EL0VTEN
== '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '0'
then
        return CNTHVS_TVAL_EL2;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '1'
then
        return CNTHV_TVAL_EL2;
    else
        return CNTV_TVAL_EL0;
elsif PSTATE.EL == EL1 then
    return CNTV_TVAL_EL0;
elsif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' && SCR_EL3.NS == '0' then
        return CNTHVS_TVAL_EL2;
    elsif HCR_EL2.E2H == '1' && SCR_EL3.NS == '1' then
        return CNTHV_TVAL_EL2;
    else
        return CNTV_TVAL_EL0;
elsif PSTATE.EL == EL3 then
    return CNTV_TVAL_EL0;
```

```
MSR CNTV_TVAL_EL0, <Xt>
```

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b11	0b011	0b1110	0b0011	0b000
~~0b11~~	~~0b1110~~	~~0b011~~	~~0b000~~	~~0b0011~~

```
if PSTATE.EL == EL0 then
    if !ELUsingAArch32(EL1) && !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') && CNTKCTL_EL1.EL0VTEN
== '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && CNTHCTL_EL2.EL0VTEN
== '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '0'
then
        CNTHVS_TVAL_EL2 = X[t];
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '1'
then
        CNTHV_TVAL_EL2 = X[t];
    else
        CNTV_TVAL_EL0 = X[t];
elsif PSTATE.EL == EL1 then
    CNTV_TVAL_EL0 = X[t];
elsif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' && SCR_EL3.NS == '0' then
        CNTHVS_TVAL_EL2 = X[t];
    elsif HCR_EL2.E2H == '1' && SCR_EL3.NS == '1' then
        CNTHV_TVAL_EL2 = X[t];
    else
        CNTV_TVAL_EL0 = X[t];
elsif PSTATE.EL == EL3 then
    CNTV_TVAL_EL0 = X[t];
```

MRS <Xt>, CNTV_TVAL_EL02

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b11	0b101	0b1110	0b0011	0b000
~~0b11~~	~~0b1110~~	~~0b101~~	~~0b000~~	~~0b0011~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if EL2Enabled() && HCR_EL2.E2H == '1' then
    if PSTATE.EL == EL2 then
        return CNTV_TVAL_EL0;
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if EL2Enabled() && HCR_EL2.E2H == '1' then
        return CNTV_TVAL_EL0;
else
    else
        UNDEFINED;
```

MSR CNTV_TVAL_EL02, <Xt>

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b11	0b101	0b1110	0b0011	0b000

0b11	0b1110	0b101	0b000	0b0011

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if EL2Enabled() && HCR_EL2.E2H == '1' then
    if PSTATE.EL == EL2 then
        CNTV_TVAL_EL0 = X[t];
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if EL2Enabled() && HCR_EL2.E2H == '1' then
        CNTV_TVAL_EL0 = X[t];
else
    else
        UNDEFINED;
```

27 13/03 12/2019 2018 21 16:59 42; e5e4db499bf9867a4b93324c4dbac985d3da93766379d01e197f1d40720d32d0f84e419e9187e009

(old) htmldiff from- (new)

CNTVOFF_EL2, Counter-timer Virtual Offset register

The CNTVOFF_EL2 characteristics are:

Purpose

Holds the 64-bit virtual offset. This is the offset between the physical count value visible in CNTPCT_EL0 and the virtual count value visible in CNTVCT_EL0.

Configuration

AArch64 System register CNTVOFF_EL2 bits [63:0] are architecturally mapped to AArch32 System register CNTVOFF[63:0] .

If EL2 is not implemented, this register is RES0 from EL3 and the virtual counter uses a fixed virtual offset of zero.

~~This register has no effect if EL2 is not enabled in the current Security state.~~

Note

When EL2 is implemented and enabled in the current Security state, and is using AArch64, the virtual counter uses a fixed virtual offset of zero in the following situations:

- HCR_EL2.E2H is 1, and CNTVCT_EL0 is read from EL2.
- HCR_EL2.{E2H, TGE} is {1, 1}, and either:
  - CNTVCT_EL0 is read from EL0 or EL2.
  - CNTVCT is read from EL0.

RW fields in this register reset to architecturally UNKNOWN values.

Attributes

CNTVOFF_EL2 is a 64-bit register.

Field descriptions

The CNTVOFF_EL2 bit assignments are:

63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
Virtual offset																															
Virtual offset																															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Bits [63:0]

Virtual offset.

This field resets to an architecturally UNKNOWN value.

Accessing the CNTVOFF_EL2

Accesses to this register use the following encodings:

```
MRS <Xt>, CNTVOFF_EL2
```

op0	op1	CRn	CRm	op2

op0	CRn	op1	op2	CRm
0b11	0b100	0b1110	0b0000	0b011
0b11	0b1110	0b100	0b011	0b0000

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
        return NVMem[0x060];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    return CNTVOFF_EL2;
elsif PSTATE.EL == EL3 then
    return CNTVOFF_EL2;
```

```
MSR CNTVOFF_EL2, <Xt>
```

op0	op1	CRn	CRm	op2
op0	CRn	op1	op2	CRm
0b11	0b100	0b1110	0b0000	0b011
0b11	0b1110	0b100	0b011	0b0000

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
        NVMem[0x060] = X[t];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    CNTVOFF_EL2 = X[t];
elsif PSTATE.EL == EL3 then
    CNTVOFF_EL2 = X[t];
```

2713/0312/20192018 2116:5942: e5e4db499bf9867a4b93324c4dbac985d3da93766379d01e197f1d40720d32d0f84e419e9187e009

CONTEXTIDR_EL1, Context ID Register (EL1)

The CONTEXTIDR_EL1 characteristics are:

Purpose

Identifies the current Process Identifier.

The value of the whole of this register is called the Context ID and is used by:

- The debug logic, for Linked and Unlinked Context ID matching.
- The trace logic, to identify the current process.

The significance of this register is for debug and trace use only.

This register is used:

- In Armv8.0.
- When ARMv8.1-VHE is implemented and HCR_EL2.E2H is 0.

Note

When ARMv8.1-VHE is implemented and HCR_EL2.E2H is set to 1, CONTEXTIDR_EL2 is used.

Configuration

AArch64 System register CONTEXTIDR_EL1 bits [31:0] are architecturally mapped to AArch32 System register CONTEXTIDR[31:0] .

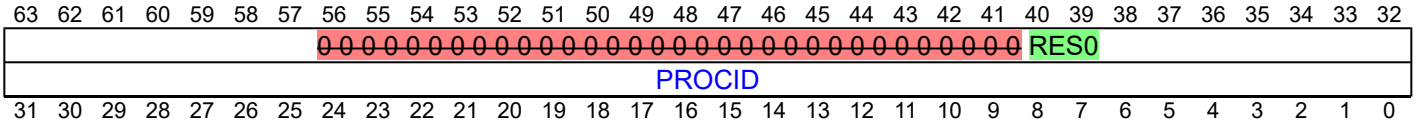
RW fields in this register reset to architecturally UNKNOWN values.

Attributes

CONTEXTIDR_EL1 is a 64-bit register.

Field descriptions

The CONTEXTIDR_EL1 bit assignments are:



Bits [63:32]

Reserved, RES0.

PROCID, bits [31:0]

Process Identifier. This field must be programmed with a unique value that identifies the current process.

Note

In AArch32 state, when TTBCR.EAE is set to 0, CONTEXTIDR.ASID holds the ASID.

In AArch64 state, CONTEXTIDR_EL1 is independent of the ASID, and for the EL1&0 translation regime either TTBR0_EL1 or TTBR1_EL1 holds the ASID.

This field resets to an architecturally UNKNOWN value.

Accessing the CONTEXTIDR_EL1

When HCR_EL2.E2H is 1, without explicit synchronization, access from EL3 using the mnemonic CONTEXTIDR_EL1 or CONTEXTIDR_EL12 are not guaranteed to be ordered with respect to accesses using the other mnemonic.

Accesses to this register use the following encodings:

```
MRS <Xt>, CONTEXTIDR_EL1
```

op0	op1	CRn	CRm	op2
~~CRn~~	~~op0~~	~~op1~~	~~op2~~	~~CRm~~
0b11	0b000	0b1101	0b0000	0b001
~~0b1101~~	~~0b11~~	~~0b000~~	~~0b001~~	~~0b0000~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TRVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<NV2,NV1,NV> == '111' then
        return NVMem[0x108];
    else
        return CONTEXTIDR_EL1;
elsif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' then
        return CONTEXTIDR_EL2;
    else
        return CONTEXTIDR_EL1;
elsif PSTATE.EL == EL3 then
    return CONTEXTIDR_EL1;
```

```
MSR CONTEXTIDR_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
~~CRn~~	~~op0~~	~~op1~~	~~op2~~	~~CRm~~
0b11	0b000	0b1101	0b0000	0b001
~~0b1101~~	~~0b11~~	~~0b000~~	~~0b001~~	~~0b0000~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<NV2,NV1,NV> == '111' then
        NVMem[0x108] = X[t];
    else
        CONTEXTIDR_EL1 = X[t];
elsif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' then
        CONTEXTIDR_EL2 = X[t];
    else
        CONTEXTIDR_EL1 = X[t];
elsif PSTATE.EL == EL3 then
    CONTEXTIDR_EL1 = X[t];
```

```
MRS <Xt>, CONTEXTIDR_EL12
```

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b11	0b101	0b1101	0b0000	0b001
~~0b11~~	~~0b1101~~	~~0b101~~	~~0b001~~	~~0b0000~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '101' then
        return NVMem[0x108];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if EL2Enabled() && HCR_EL2.E2H == '1' then
    if PSTATE.EL == EL2 then
        return CONTEXTIDR_EL1;
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if EL2Enabled() && HCR_EL2.E2H == '1' then
        return CONTEXTIDR_EL1;
else
    else
        UNDEFINED;
```

```
MSR CONTEXTIDR_EL12, <Xt>
```

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b11	0b101	0b1101	0b0000	0b001
~~0b11~~	~~0b1101~~	~~0b101~~	~~0b001~~	~~0b0000~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '101' then
        NVMem[0x108] = X[t];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if EL2Enabled() && HCR_EL2.E2H == '1' then
    if PSTATE.EL == EL2 then
        CONTEXTIDR_EL1 = X[t];
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if EL2Enabled() && HCR_EL2.E2H == '1' then
        CONTEXTIDR_EL1 = X[t];
else
    else
        UNDEFINED;
```

(old) htmldiff from- (new)

(old) htmldiff from- (new)

CPACR_EL1, Architectural Feature Access Control Register

The CPACR_EL1 characteristics are:

Purpose

Controls access to trace, SVE, Advanced SIMD and floating-point functionality.

Configuration

AArch64 System register CPACR_EL1 bits [31:0] are architecturally mapped to AArch32 System register CPACR[31:0] .

When HCR_EL2.{E2H, TGE} == {1, 1}, the fields in this register have no effect on execution at EL0 and EL1. In this case, the controls provided by CPTR_EL2 are used.

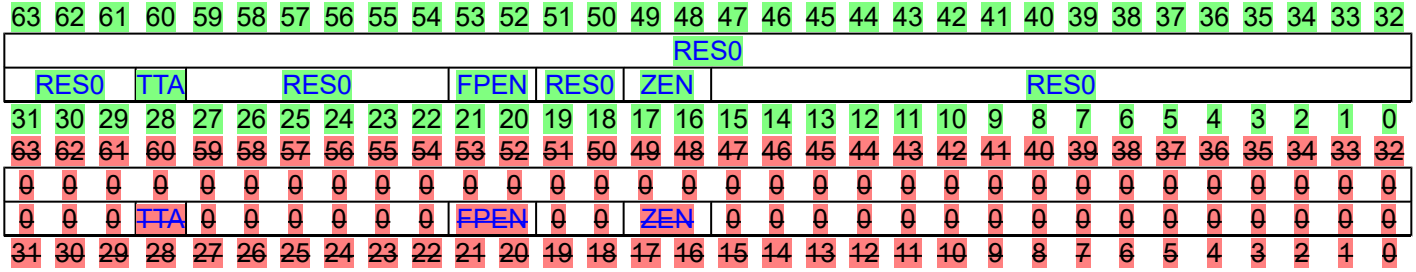
RW fields in this register reset to architecturally UNKNOWN values.

Attributes

CPACR_EL1 is a 64-bit register.

Field descriptions

The CPACR_EL1 bit assignments are:



Bits [63:29]

Reserved, RES0.

TTA, bit [28]

Traps EL0 and EL1 System register accesses to all implemented trace registers to EL1, or to EL2 when it is implemented and enabled for the current Security state and HCR_EL2.TGE is 1, from both Execution states.

TTA	Meaning
0b0	This control does not cause any instructions to be trapped.
0b1	This control causes EL0 and EL1 System register accesses to all implemented trace registers to be trapped.

Note

- The ETMv4 architecture does not permit EL0 to access the trace registers. If the Armv8-A architecture is implemented with an ETMv4 implementation, EL0 accesses to the trace registers are UNDEFINED, and any resulting exception is higher priority than an exception that would be generated because the value of CPACR_EL1.TTA is 1.
- The Armv8-A architecture does not provide traps on trace register accesses through the optional memory-mapped interface.

System register accesses to the trace registers can have side-effects. When a System register access is trapped, any side-effects that are normally associated with the access do not occur before the exception is taken.

If System register access to the trace functionality is not implemented, this bit is RES0.

This field resets to an architecturally UNKNOWN value.

Bits [27:22]

Reserved, RES0.

FPEN, bits [21:20]

Traps EL0 and EL1 accesses to the SVE, Advanced SIMD, and floating-point registers to EL1, or to EL2 when it is implemented and enabled for the current Security state and HCR_EL2.TGE is 1, from both Execution states.

FPEN	Meaning
0b00	This control causes any instructions at EL0 or EL1 that use the registers associated with SVE, Advanced SIMD and floating-point execution to be trapped, unless they are trapped by CPACR_EL1.ZEN.
0b01	This control causes any instructions at EL0 that use the registers associated with SVE, Advanced SIMD and floating-point execution to be trapped, unless they are trapped by CPACR_EL1.ZEN, but does not cause any instruction at EL1 to be trapped.
0b10	This control causes any instructions at EL0 or EL1 that use the registers associated with SVE, Advanced SIMD and floating-point execution to be trapped, unless they are trapped by CPACR_EL1.ZEN.
0b11	This control does not cause any instructions to be trapped.

Writes to MVFR0, MVFR1 and MVFR2 from EL1 or higher are CONSTRAINED UNPREDICTABLE and whether these accesses can be trapped by this control depends on implemented CONSTRAINED UNPREDICTABLE behavior.

Note

- Attempts to write to the FPSID count as use of the registers for accesses from EL1 or higher.
- Accesses from EL0 to FPSID, MVFR0, MVFR1, MVFR2, and FPEXC are UNDEFINED, and any resulting exception is higher priority than an exception that would be generated because the value of CPACR_EL1.FPEN is not 0b11.

This field resets to an architecturally UNKNOWN value.

Bits [19:18]

Reserved, RES0.

ZEN, bits [17:16]

When SVE is implemented:

Traps SVE instructions and instructions that access SVE System registers at EL0 and EL1 to EL1, or to EL2 when it is implemented and enabled for the current Security state and HCR_EL2.TGE is 1.

ZEN	Meaning
0b00	This control causes these instructions executed at EL0 or EL1 to be trapped.
0b01	This control causes these instructions executed at EL0 to be trapped, but does not cause any instruction at EL1 to be trapped.
0b10	This control causes these instructions executed at EL0 or EL1 to be trapped.
0b11	This control does not cause any instruction to be trapped.

If SVE is not implemented, this field is RES0.

This field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [15:0]

Reserved, RES0.

Accessing the CPACR_EL1

When HCR_EL2.E2H is 1, without explicit synchronization, access from EL3 using the mnemonic CPACR_EL1 or CPACR_EL12 are not guaranteed to be ordered with respect to accesses using the other mnemonic.

Accesses to this register use the following encodings:

```
MRS <Xt>, CPACR_EL1
```

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b11	0b000	0b0001	0b0000	0b010
~~0b11~~	~~0b0001~~	~~0b000~~	~~0b010~~	~~0b0000~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && CPTR_EL2.TCPAC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TCPAC == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<NV2,NV1,NV> == '111' then
        return NVMem[0x100];
    else
        return CPACR_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TCPAC == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HCR_EL2.E2H == '1' then
        return CPTR_EL2;
    else
        return CPACR_EL1;
elsif PSTATE.EL == EL3 then
    return CPACR_EL1;
```

```
MSR CPACR_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b11	0b000	0b0001	0b0000	0b010
~~0b11~~	~~0b0001~~	~~0b000~~	~~0b010~~	~~0b0000~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && CPTR_EL2.TCPAC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TCPAC == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<NV2,NV1,NV> == '111' then
        NVMem[0x100] = X[t];
    else
        CPACR_EL1 = X[t];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TCPAC == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HCR_EL2.E2H == '1' then
        CPTR_EL2 = X[t];
    else
        CPACR_EL1 = X[t];
elsif PSTATE.EL == EL3 then
    CPACR_EL1 = X[t];
```

MRS <Xt>, CPACR_EL12

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b11	0b101	0b0001	0b0000	0b010
~~0b11~~	~~0b0001~~	~~0b101~~	~~0b010~~	~~0b0000~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '101' then
        return NVMem[0x100];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if EL2Enabled() && HCR_EL2.E2H == '1' then
    if PSTATE.EL == EL2 then
        if HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TCPAC == '1' then
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return CPACR_EL1;
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if EL2Enabled() && HCR_EL2.E2H == '1' then
        return CPACR_EL1;
else
    else
        UNDEFINED;
```

MSR CPACR_EL12, <Xt>

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b11	0b101	0b0001	0b0000	0b010
~~0b11~~	~~0b0001~~	~~0b101~~	~~0b010~~	~~0b0000~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '101' then
        NVMem[0x100] = X[t];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if EL2Enabled() && HCR_EL2.E2H == '1' then
    if PSTATE.EL == EL2 then
        if HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TCPAC == '1' then
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            CPACR_EL1 = X[t];
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if EL2Enabled() && HCR_EL2.E2H == '1' then
        CPACR_EL1 = X[t];
else
    else
        UNDEFINED;
```

27~~13~~/03~~12~~/2019~~2018~~ 21~~16~~:59~~42~~; e5e4db499bf9867a4b93324c4dbac985d3da93766~~379d01e197f1d40720d32d0f84e419e9187e009~~

(old) htmldiff from- (new)

(old) htmldiff from- (new)

CPP RCTX, Cache Prefetch Prediction Restriction by Context

The CPP RCTX characteristics are:

Purpose

Cache Prefetch Prediction Restriction by Context applies to all Cache Allocation Resources that predict cache allocations based on information gathered within the target execution context or contexts.

When this instruction is complete and synchronized, cache prefetch prediction does not permit later speculative execution within the target execution context to be observable through side channels.

This instruction applies to all:

- Instruction caches.
- Data caches.
- TLB prefetching hardware used by the executing PE that applies to the supplied context or contexts.

This instruction is guaranteed to be complete following a DSB that covers both read and write behavior on the same PE as executed the original restriction instruction, and a subsequent context synchronization event is required to ensure that the effect of the completion of the instructions is synchronized to the current execution.

Note

This instruction does not require the invalidation of Cache Allocation Resources so long as the behavior described for completion of this instruction is met by the implementation.

On some implementations the instruction is likely to take a significant number of cycles to execute. This instruction is expected to be used very rarely, such as on the roll-over of an ASID or VMID, but should not be used on every context switch.

Configuration

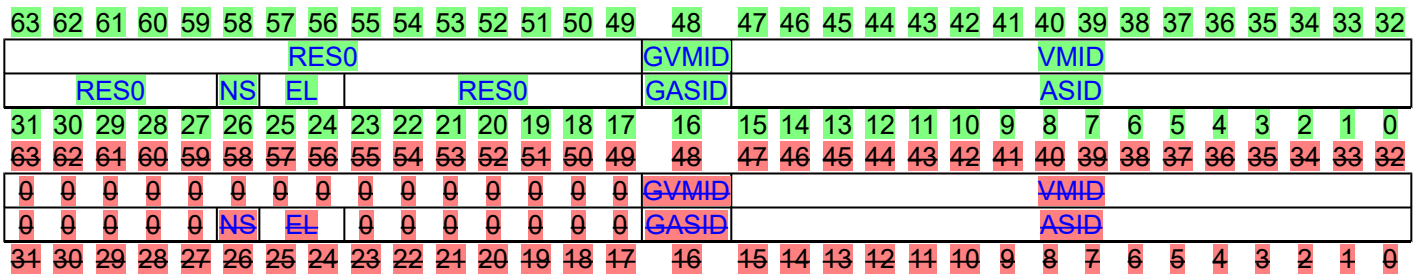
This instruction is present only when ARMv8.0-PredInv is implemented. Otherwise, direct accesses to CPP RCTX are UNDEFINED.

Attributes

CPP RCTX is a 64-bit System instruction.

Field descriptions

The CPP RCTX input value bit assignments are:



Bits [63:49]

Reserved, RES0.

GVMID, bit [48]

Execution of this instruction applies to all VMIDs or a specified VMID.

GVMID	Meaning
0b0	Applies to specified VMID for an EL0 or EL1 context. For all other contexts this field is RES0.
0b1	Applies to all VMIDs for an EL0 or EL1 context. For all other contexts this field is RES0.

If the instruction is executed at EL0 or EL1, then this field has an Effective value of 0.

VMID, bits [47:32]

Only applies when bit[48] is 0 and one of:

- an EL1 context.
- an EL0 context when (HCR_EL2.E2H==0 or HCR_EL2.TGE==0).

Otherwise this field is RES0.

When the instruction is executed at EL1 then this field is treated as the current VMID.

When the instruction is executed at EL0 and (HCR_EL2.E2H==0 or HCR_EL2.TGE==0) then this field is treated as the current VMID.

When the instruction is executed at EL0 and (HCR_EL2.E2H==1 and HCR_EL2.TGE==1) then this field is ignored.

Bits [31:27]

Reserved, RES0.

NS, bit [26]

Security State.

NS	Meaning
0b0	Secure state.
0b1	Non-secure state.

If the instruction is executed in Non-secure state, this field has an Effective value of 1.

EL, bits [25:24]

Exception Level.

EL	Meaning
0b00	EL0.
0b01	EL1.
0b10	EL2.
0b11	EL3.

If the instruction is executed at an exception level lower than the specified level, this instruction is treated as a NOP.

Bits [23:17]

Reserved, RES0.

GASID, bit [16]

Execution of this instruction applies to all ASIDs or a specified ASID.

GASID	Meaning
0b0	Applies to specified ASID for an EL0 context. For all other contexts this field is RES0.
0b1	Applies to all ASID for an EL0 context. For all other contexts this field is RES0.

If the instruction is executed at EL0, then this field has an Effective value of 0.

ASID, bits [15:0]

Only applies for an EL0 context and when bit[16] is 0.

Otherwise this field is RES0.

When the instruction is executed at EL0 then this field is treated as the current ASID.

Executing the CPP RCTX instruction

Accesses to this instruction use the following encodings:

```
CPP RCTX, <Xt>
```

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b01	0b011	0b0111	0b0011	0b111
~~0b01~~	~~0b0111~~	~~0b011~~	~~0b111~~	~~0b0011~~

```
if PSTATE.EL == EL0 then
    if !ELUsingAArch32(EL1) && !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') && SCTLR_EL1.EnRCTX ==
'0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCTLR_EL2.EnRCTX ==
'0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        CPP_RCTX(X[t]);
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        CPP_RCTX(X[t]);
elsif PSTATE.EL == EL2 then
    CPP_RCTX(X[t]);
elsif PSTATE.EL == EL3 then
    CPP_RCTX(X[t]);
```

27~~13~~/03~~12~~/2019~~2018~~ 21~~16~~:59~~42~~; e5e4db499bf9867a4b93324c4dbac985d3da93766~~379d01e197f1d40720d32d0f84e419e9187e009~~

(old) htmldiff from- (new)

CPTR_EL2, Architectural Feature Trap Register (EL2)

The CPTR_EL2 characteristics are:

Purpose

Controls:

- Trapping to EL2 of access to CPACR, CPACR_EL1, trace functionality, and to SVE, Advanced SIMD and floating-point functionality.
- EL2 access to trace functionality, and to SVE, Advanced SIMD and floating-point functionality.

Configuration

AArch64 System register CPTR_EL2 bits [31:0] are architecturally mapped to AArch32 System register HCPTR[31:0] .

If EL2 is not implemented, this register is RES0 from EL3.

This register has no effect if EL2 is not enabled in the current Security state.

RW fields in this register reset to architecturally UNKNOWN values.

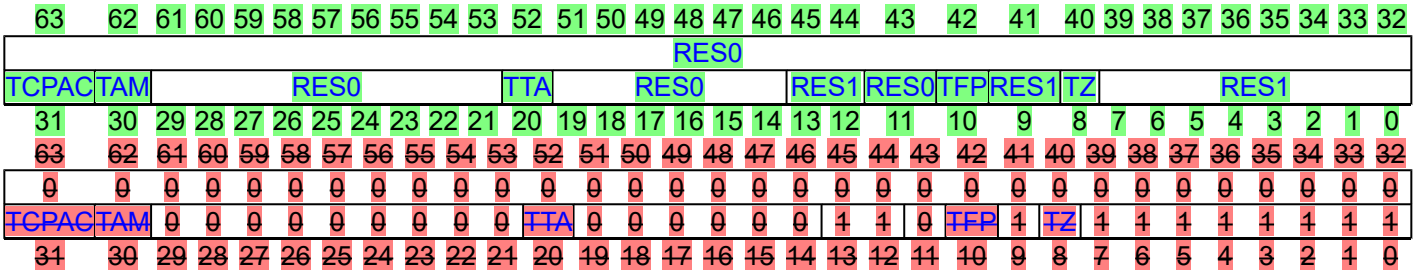
Attributes

CPTR_EL2 is a 64-bit register.

Field descriptions

The CPTR_EL2 bit assignments are:

When HCR_EL2.E2H == 0:



This format applies in all Armv8.0 implementations.

Bits [63:32]

Reserved, RES0.

TCPAC, bit [31]

Traps EL1 accesses to CPACR_EL1 or CPACR to EL2 when EL2 is enabled in the current Security state.

TCPAC	Meaning
0b0	This control does not cause any instructions to be trapped.
0b1	EL1 accesses to CPACR_EL1 and CPACR are trapped to EL2 when EL2 is enabled in the current Security state.

Note

CPACR_EL1 and CPACR are not accessible at EL0.

This field resets to an architecturally UNKNOWN value.

TAM, bit [30]

When AMUv1 is implemented:

Trap Activity Monitor access. ~~Traps EL1 and EL0 accesses to all Activity Monitor registers to EL2, when EL2 is enabled in the current Security state.~~

TAM	Meaning
0b0	Accesses from EL1 and EL0 to Activity Monitor registers are not trapped.
0b1	Accesses from EL1 and EL0 to Activity Monitor registers are trapped to EL2, when EL2 is enabled in the current Security state.~~EL2.~~

This field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [29:21]

Reserved, RES0.

TTA, bit [20]

From Armv8.1:

Traps System register accesses to all implemented trace registers to EL2 when EL2 is enabled in the current Security state, from both Execution states.

TTA	Meaning
0b0	This control does not cause any instructions to be trapped.
0b1	Any attempt at EL0, EL1, or EL2, to execute a System register access to an implemented trace register is trapped to EL2 when EL2 is enabled in the current Security state, unless it is trapped by CPACR.TRCDIS or CPACR_EL1.TTA.

Note

- The ETMv4 architecture does not permit EL0 to access the trace registers. If the Armv8-A architecture is implemented with an ETMv4 implementation, EL0 accesses to the trace registers are UNDEFINED, and any resulting exception is higher priority than an exception that would be generated because the value of CPTR_EL2.TTA is 1.
- EL2 does not provide traps on trace register accesses through the optional memory-mapped interface.

System register accesses to the trace registers can have side-effects. When a System register access is trapped, any side-effects that are normally associated with the access do not occur before the exception is taken.

If System register access to the trace functionality is not supported, this bit is RES0.

This field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [19:14]

Reserved, RES0.

Bits [13:12]

Reserved, RES1.

Bit [11]

Reserved, RES0.

TFP, bit [10]

Traps accesses to SVE, Advanced SIMD and floating-point functionality to EL2 when EL2 is enabled in the current Security state, from both Execution states.

TFP	Meaning
0b0	This control does not cause any instructions to be trapped.
0b1	Any attempt at EL0, EL1 or EL2, to execute an instruction that uses the registers associated with SVE, Advanced SIMD and floating-point execution is trapped to EL2 when EL2 is enabled in the current Security state, subject to the exception prioritization rules, unless it is trapped by CPTR_EL2.TZ.

This field resets to an architecturally UNKNOWN value.

Bit [9]

Reserved, RES1.

TZ, bit [8]

When SVE is implemented:

Traps execution at EL2, EL1, or EL0 of SVE instructions and instructions that access SVE System registers to EL2 when EL2 is enabled in the current Security state.

TZ	Meaning
0b0	This control does not cause any instruction to be trapped.
0b1	This control causes these instructions to be trapped, subject to the exception prioritization rules.

This field resets to an architecturally UNKNOWN value.

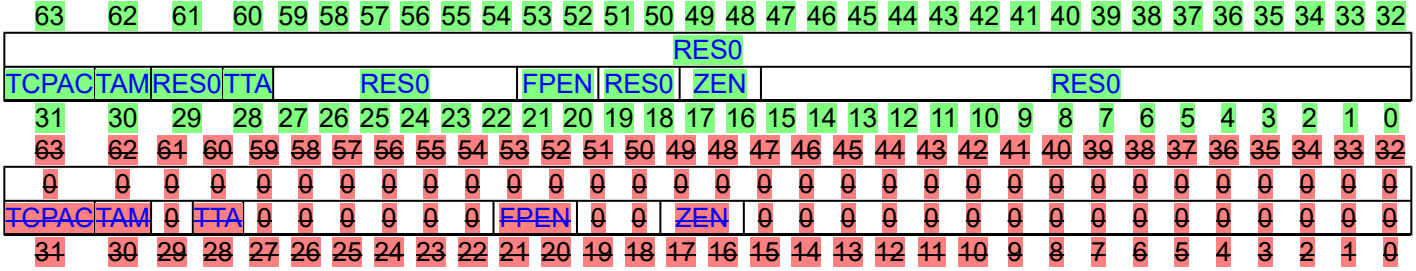
Otherwise:

Reserved, RES1.

Bits [7:0]

Reserved, RES1.

When HCR_EL2.E2H == 1:



Bits [63:32]

Reserved, RES0.

TCPAC, bit [31]

From Armv8.1:

When HCR_EL2.TGE is 0, traps EL1 accesses to CPACR_EL1 and CPACR to EL2 when EL2 is enabled in the current Security state.

TCPAC	Meaning
0b0	This control does not cause any instructions to be trapped.
0b1	EL1 accesses to CPACR_EL1 and CPACR are trapped to EL2 when EL2 is enabled in the current Security state.

When HCR_EL2.TGE is 1, this control does not cause any instructions to be trapped.

Note

CPACR_EL1 and CPACR are not accessible at EL0.

This field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TAM, bit [30]

When AMUv1 is implemented:

Trap Activity Monitor access. Traps EL1 and EL0 accesses to all Activity Monitor registers to EL2.

TAM	Meaning
0b0	Accesses from EL1 and EL0 to Activity Monitor registers are not trapped.
0b1	Accesses from EL1 and EL0 to Activity Monitor registers are trapped to EL2, when EL2 is enabled in the current Security state.

This field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bit [29]

Reserved, RES0.

TTA, bit [28]

From Armv8.1:

Traps System register accesses to all implemented trace registers to EL2 when EL2 is enabled in the current Security state, from both Execution states.

TTA	Meaning
0b0	This control does not cause any instructions to be trapped.
0b1	Any attempt at EL0, EL1 or EL2, to execute a System register access to an implemented trace register is trapped to EL2 when EL2 is enabled in the current Security state, unless HCR_EL2.TGE is 0 and it is trapped by CPACR.NSTRCDIS or CPACR_EL1.TTA. When HCR_EL2.TGE is 1, any attempt at EL0 or EL2 to execute a System register access to an implemented trace register is trapped to EL2 when EL2 is enabled in the current Security state.

Note

- The ETMv4 architecture does not permit EL0 to access the trace registers. If the Armv8-A architecture is implemented with an ETMv4 implementation, EL0 accesses to the trace registers are UNDEFINED, and any resulting exception is higher priority than an exception that would be generated because the value of CPTR_EL2.TTA is 1.
- EL2 does not provide traps on trace register accesses through the optional memory-mapped interface.

System register accesses to the trace registers can have side-effects. When a System register access is trapped, any side-effects that are normally associated with the access do not occur before the exception is taken.

If System register access to the trace functionality is not supported, this bit is RES0.

This field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [27:22]

Reserved, RES0.

FPEN, bits [21:20]

From Armv8.1:

Traps EL0, EL2 and, when HCR_EL2.TGE is 0, EL1 accesses to the SVE, Advanced SIMD and floating-point registers to EL2 when EL2 is enabled in the current Security state, from both Execution states.

FPEN	Meaning
0b00	This control causes any instructions at EL0, EL1, or EL2 that use the registers associated with SVE, Advanced SIMD and floating-point execution to be trapped, subject to the exception prioritization rules, unless they are trapped by CPTR_EL2.ZEN.
0b01	When HCR_EL2.TGE is 0, this control does not cause any instructions to be trapped. When HCR_EL2.TGE is 1, this control causes instructions at EL0 that use the registers associated with SVE, Advanced SIMD and floating-point execution to be trapped, unless they are trapped by CPTR_EL2.ZEN, but does not cause any instruction at EL2 to be trapped.
0b10	This control causes any instructions at EL0, EL1, or EL2 that use the registers associated with SVE, Advanced SIMD and floating-point execution to be trapped, subject to the exception prioritization rules, unless they are trapped by CPTR_EL2.ZEN.
0b11	This control does not cause any instructions to be trapped.

Writes to MVFR0, MVFR1, and MVFR2 from EL1 or higher are CONSTRAINED UNPREDICTABLE and whether these accesses can be trapped by this control depends on implemented CONSTRAINED UNPREDICTABLE behavior.

Note

- Attempts to write to the FPSID count as use of the registers for accesses from EL1 or higher.
- Accesses from EL0 to FPSID, MVFR0, MVFR1, MVFR2, and FPEXC are UNDEFINED, and any resulting exception is higher priority than an exception that would be generated because the value of CPTR_EL2.FPEN is not 0b11.

This field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [19:18]

Reserved, RES0.

ZEN, bits [17:16]

When SVE is implemented:

Traps execution at EL2, EL1, and EL0 of SVE instructions or instructions that access SVE System registers to EL2 when EL2 is enabled in the current Security state.

ZEN	Meaning
0b00	This control causes execution at EL2, EL1, and EL0 of these instructions to be trapped, subject to the exception prioritization rules.
0b01	When HCR_EL2.TGE is 0, this control does not cause any instruction to be trapped. When HCR_EL2.TGE is 1, this control causes these instructions executed at EL0 to be trapped, but does not cause any instruction at EL2 to be trapped.
0b10	This control causes execution at EL2, EL1, and EL0 of these instructions to be trapped, subject to the exception prioritization rules.
0b11	This control does not cause any instruction to be trapped.

This field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [15:0]

Reserved, RES0.

Accessing the CPTR_EL2

Accesses to this register use the following encodings:

```
MRS <Xt>, CPTR_EL2
```

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b11	0b100	0b0001	0b0001	0b010
~~0b11~~	~~0b0001~~	~~0b100~~	~~0b010~~	~~0b0001~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TCPAC == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return CPTR_EL2;
elsif PSTATE.EL == EL3 then
    return CPTR_EL2;
```

MSR CPTR_EL2, <Xt>

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b11	0b100	0b0001	0b0001	0b010
~~0b11~~	~~0b0001~~	~~0b100~~	~~0b010~~	~~0b0001~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TCPAC == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        CPTR_EL2 = X[t];
elsif PSTATE.EL == EL3 then
    CPTR_EL2 = X[t];
```

MRS <Xt>, CPACR_EL1

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b11	0b000	0b0001	0b0000	0b010
~~0b11~~	~~0b0001~~	~~0b000~~	~~0b010~~	~~0b0000~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && CPTR_EL2.TCPAC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TCPAC == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<NV2,NV1,NV> == '111' then
        return NVMem[0x100];
    else
        return CPACR_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TCPAC == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HCR_EL2.E2H == '1' then
        return CPTR_EL2;
    else
        return CPACR_EL1;
elsif PSTATE.EL == EL3 then
    return CPACR_EL1;
```

MSR CPACR_EL1, <Xt>

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b11	0b000	0b0001	0b0000	0b010
~~0b11~~	~~0b0001~~	~~0b000~~	~~0b010~~	~~0b0000~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && CPTR_EL2.TCPAC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TCPAC == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<NV2,NV1,NV> == '111' then
        NVMem[0x100] = X[t];
    else
        CPACR_EL1 = X[t];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TCPAC == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HCR_EL2.E2H == '1' then
        CPTR_EL2 = X[t];
    else
        CPACR_EL1 = X[t];
elsif PSTATE.EL == EL3 then
    CPACR_EL1 = X[t];
```

27~~13~~/03~~12~~/2019~~2018~~ 21~~16~~:59~~42~~: e5e4db499bf9867a4b93324c4dbac985d3da93766~~379d01e197f1d40720d32d0f84e419e9187e009~~

(old) htmldiff from- (new)

CSSELR_EL1, Cache Size Selection Register

The CSSELR_EL1 characteristics are:

Purpose

Selects the current Cache Size ID Register, CCSIDR_EL1, by specifying the required cache level and the cache type (either instruction or data cache).

Configuration

AArch64 System register CSSELR_EL1 bits [31:0] are architecturally mapped to AArch32 System register CSSELR[31:0] .

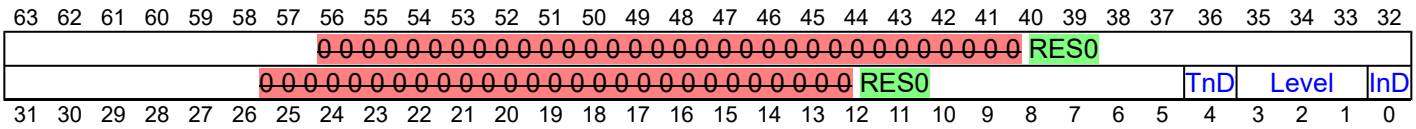
RW fields in this register reset to architecturally UNKNOWN values.

Attributes

CSSELR_EL1 is a 64-bit register.

Field descriptions

The CSSELR_EL1 bit assignments are:



Bits [63:5]

Reserved, RES0.

TnD, bit [4]

When ARMv8.5-MemTag is implemented:

Allocation Tag not Data bit.

TnD	Meaning
0b0	Data or unified cache.
0b1	Allocation Tag cache.

When CCSELR_EL1.InD == 1, this bit is RES0.

If CSSELR_EL1.Level is programmed to a cache level that is not implemented, then the value for this field on a read of CSSELR is UNKNOWN.

This field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Level, bits [3:1]

Cache level of required cache.

Level	Meaning
0b000	Level 1 cache.
0b001	Level 2 cache.
0b010	Level 3 cache.
0b011	Level 4 cache.
0b100	Level 5 cache.
0b101	Level 6 cache.
0b110	Level 7 cache.

All other values are reserved.

If CSSELR_EL1.Level is programmed to a cache level that is not implemented, then the value for this field on a read of CSSELR is UNKNOWN.

This field resets to an architecturally UNKNOWN value.

InD, bit [0]

Instruction not Data bit.

InD	Meaning
0b0	Data or unified cache.
0b1	Instruction cache.

If CSSELR_EL1.Level is programmed to a cache level that is not implemented, then the value for this field on a read of CSSELR is UNKNOWN.

This field resets to an architecturally UNKNOWN value.

Accessing the CSSELR_EL1

If CSSELR_EL1.Level is programmed to a cache level that is not implemented, then a read of CSSELR_EL1 is CONSTRAINED UNPREDICTABLE, and returns UNKNOWN values for CSSELR_EL1.{Level, InD}.

Accesses to this register use the following encodings:

```
MRS <Xt>, CSSELR_EL1
```

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b11	0b010	0b0000	0b0000	0b000
~~0b11~~	~~0b0000~~	~~0b010~~	~~0b000~~	~~0b0000~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TID2 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TID4 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        return CSSELR_EL1;
elsif PSTATE.EL == EL2 then
    return CSSELR_EL1;
elsif PSTATE.EL == EL3 then
    return CSSELR_EL1;
```

```
MSR CSSELR_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b11	0b010	0b0000	0b0000	0b000
~~0b11~~	~~0b0000~~	~~0b010~~	~~0b000~~	~~0b0000~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TID2 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TID4 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        CSSELR_EL1 = X[t];
elsif PSTATE.EL == EL2 then
    CSSELR_EL1 = X[t];
elsif PSTATE.EL == EL3 then
    CSSELR_EL1 = X[t];
```

27~~13~~/03~~12~~/2019~~2018~~ 21~~16~~:59~~42~~; e5e4db499bf9867a4b93324c4dbac985d3da93766~~379d01e197f1d40720d32d0f84e419e9187e009~~

(old) htmldiff from- (new)

(old) htmldiff from- (new)

DACR32_EL2, Domain Access Control Register

The DACR32_EL2 characteristics are:

Purpose

Allows access to the AArch32 DACR register from AArch64 state only. Its value has no effect on execution in AArch64 state.

Configuration

AArch64 System register DACR32_EL2 bits [31:0] are architecturally mapped to AArch32 System register DACR[31:0] .

If EL1 does not support AArch32, this register is UNDEFINED.

If EL2 is not implemented but EL3 is implemented, and EL1 is capable of using AArch32, then this register is not RES0.

~~This register has no effect if EL2 is not enabled in the current Security state.~~

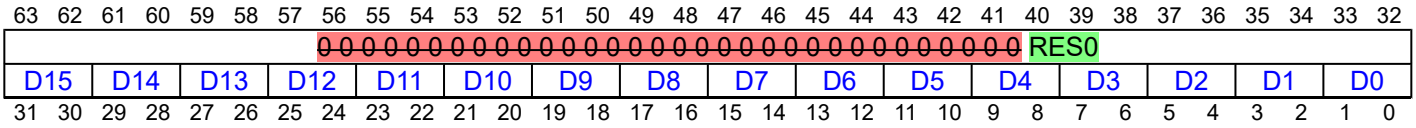
RW fields in this register reset to architecturally UNKNOWN values.

Attributes

DACR32_EL2 is a 64-bit register.

Field descriptions

The DACR32_EL2 bit assignments are:



Bits [63:32]

Reserved, RES0.

D<n>, bits [2n+1:2n], for n = 0 to 15

Domain n access permission, where n = 0 to 15. Permitted values are:

D<n>	Meaning
0b00	No access. Any access to the domain generates a Domain fault.
0b01	Client. Accesses are checked against the permission bits in the translation tables.
0b11	Manager. Accesses are not checked against the permission bits in the translation tables.

The value 0b10 is reserved.

This field resets to an architecturally UNKNOWN value.

Accessing the DACR32_EL2

Accesses to this register use the following encodings:

MRS <Xt>, DACR32_EL2

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b11	0b100	0b0011	0b0000	0b000
~~0b11~~	~~0b0011~~	~~0b100~~	~~0b000~~	~~0b0000~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    return DACR32_EL2;
elsif PSTATE.EL == EL3 then
    return DACR32_EL2;
```

MSR DACR32_EL2, <Xt>

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b11	0b100	0b0011	0b0000	0b000
~~0b11~~	~~0b0011~~	~~0b100~~	~~0b000~~	~~0b0000~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    DACR32_EL2 = X[t];
elsif PSTATE.EL == EL3 then
    DACR32_EL2 = X[t];
```

27~~13~~/03~~12~~/2019~~2018~~ 21~~16~~:59~~42~~; e5e4db499bf9867a4b93324c4dbac985d3da93766379d01e197f1d40720d32d0f84e419e9187e009

(old) htmldiff from- (new)

DC IGVAC, Data, Allocation Tag or unified Cache line Invalidate of Allocation Tags by VA to PoC

The DC IGVAC characteristics are:

Purpose

Invalidate Allocation Tags in data cache by address to Point of Coherency.

Configuration

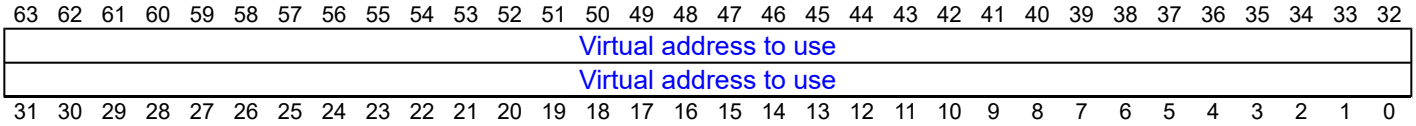
This instruction is present only when ARMv8.5-MemTag is implemented. Otherwise, direct accesses to DC IGVAC are UNDEFINED.

Attributes

DC IGVAC is a 64-bit System instruction.

Field descriptions

The DC IGVAC input value bit assignments are:



Bits [63:0]

Virtual address to use. No alignment restrictions apply to this VA.

Executing the DC IGVAC instruction

When the instruction is executed, it can generate a watchpoint, which is prioritized in the same way as other watchpoints. If a watchpoint is generated, the CM bit in the ESR_ELx.ISS field is set to 1.

This instruction requires write access permission to the VA, otherwise it generates a Permission Fault, subject to the constraints described in 'Permission fault' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

Execution of this instruction might require an address translation from VA to PA, and that translation might fault. For more information, see 'The data cache maintenance instruction (DC)' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

Accesses to this instruction use the following encodings:

```
DC IGVAC, <Xt>
```

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b01	0b000	0b0111	0b0110	0b011
~~0b01~~	~~0b0111~~	~~0b000~~	~~0b011~~	~~0b0110~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TPCPHCR_EL2.TSW == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.SWIO == '1' then
        DC_CIGVAC(X[t]);
    else
        DC_IGVAC(X[t]);
elsif PSTATE.EL == EL2 then
    DC_IGVAC(X[t]);
elsif PSTATE.EL == EL3 then
    DC_IGVAC(X[t]);
```

27~~13~~/03~~12~~/2019~~2018~~ 21~~16~~:59~~42~~; e5e4db499bf9867a4b93324c4dbac985d3da93766~~379d01e197f1d40720d32d0f84e419e9187e009~~

(old) htmldiff from- (new)

(old) htmldiff from- (new)

DVP RCTX, Data Value Prediction Restriction by Context

The DVP RCTX characteristics are:

Purpose

Data Value Prediction Restriction by Context applies to all Data Value Prediction Resources that predict execution based on information gathered within the target execution context or contexts.

When this instruction is complete and synchronized, data value prediction does not permit later speculative execution within the target execution context to be observable through side channels.

This instruction is guaranteed to be complete following a DSB that covers both read and write behavior on the same PE as executed the original restriction instruction, and a subsequent context synchronization event is required to ensure that the effect of the completion of the instructions is synchronized to the current execution.

Note

This instruction does not require the invalidation of prediction structures so long as the behavior described for completion of this instruction is met by the implementation.

On some implementations the instruction is likely to take a significant number of cycles to execute. This instruction is expected to be used very rarely, such as on the roll-over of an ASID or VMID, but should not be used on every context switch.

Configuration

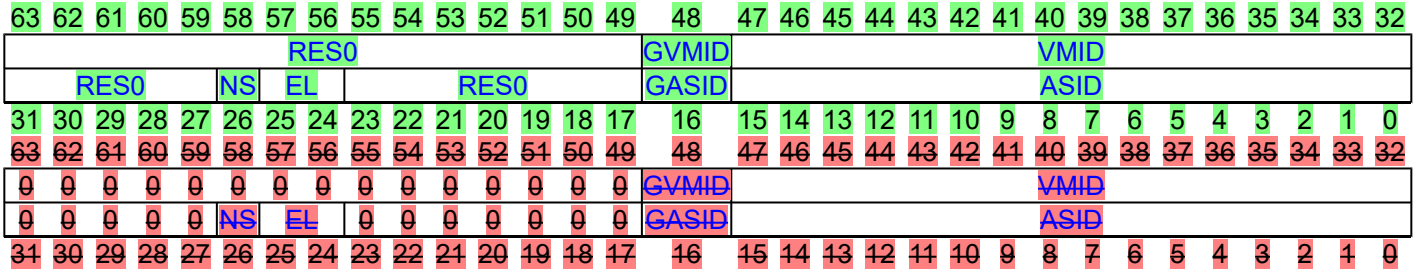
This instruction is present only when ARMv8.0-PredInv is implemented. Otherwise, direct accesses to DVP RCTX are UNDEFINED.

Attributes

DVP RCTX is a 64-bit System instruction.

Field descriptions

The DVP RCTX input value bit assignments are:



Bits [63:49]

Reserved, RES0.

GVMID, bit [48]

Execution of this instruction applies to all VMIDs or a specified VMID.

GVMID	Meaning
0b0	Applies to specified VMID for an EL0 or EL1 context. For all other contexts this field is RES0.
0b1	Applies to all VMIDs for an EL0 or EL1 context. For all other contexts this field is RES0.

If the instruction is executed at EL0 or EL1, then this field has an Effective value of 0.

VMID, bits [47:32]

Only applies when bit[48] is 0 and one of:

- an EL1 context.
- an EL0 context when (HCR_EL2.E2H==0 or HCR_EL2.TGE==0).

Otherwise this field is RES0.

When the instruction is executed at EL1 then this field is treated as the current VMID.

When the instruction is executed at EL0 and (HCR_EL2.E2H==0 or HCR_EL2.TGE==0) then this field is treated as the current VMID.

When the instruction is executed at EL0 and (HCR_EL2.E2H==1 and HCR_EL2.TGE==1) then this field is ignored.

Bits [31:27]

Reserved, RES0.

NS, bit [26]

Security State.

NS	Meaning
0b0	Secure state.
0b1	Non-secure state.

If the instruction is executed in Non-secure state, this field has an Effective value of 1.

EL, bits [25:24]

Exception Level

EL	Meaning
0b00	EL0.
0b01	EL1.
0b10	EL2.
0b11	EL3.

If the instruction is executed at an exception level lower than the specified level, this instruction is treated as a NOP.

Bits [23:17]

Reserved, RES0.

GASID, bit [16]

Execution of this instruction applies to all ASIDs or a specified ASID.

GASID	Meaning
0b0	Applies to specified ASID for an EL0 context. For all other contexts this field is RES0.
0b1	Applies to all ASID for an EL0 context. For all other contexts this field is RES0.

If the instruction is executed at EL0, then this field has an Effective value of 0.

ASID, bits [15:0]

Only applies for an EL0 context and when bit[16] is 0.

Otherwise this field is RES0.

When the instruction is executed at EL0 then this field is treated as the current ASID.

Executing the DVP RCTX instruction

Accesses to this instruction use the following encodings:

```
DVP RCTX, <Xt>
```

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b01	0b011	0b0111	0b0011	0b101
~~0b01~~	~~0b0111~~	~~0b011~~	~~0b101~~	~~0b0011~~

```
if PSTATE.EL == EL0 then
    if !ELUsingAArch32(EL1) && !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') && SCTLR_EL1.EnRCTX ==
'0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCTLR_EL2.EnRCTX ==
'0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        DVP_RCTX(X[t]);
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        DVP_RCTX(X[t]);
elsif PSTATE.EL == EL2 then
    DVP_RCTX(X[t]);
elsif PSTATE.EL == EL3 then
    DVP_RCTX(X[t]);
```

27~~13~~/03~~12~~/2019~~2018~~ 21~~16~~:59~~42~~: e5e4db499bf9867a4b93324c4dbac985d3da93766379d01e197f1d40720d32d0f84e419e9187e009

(old) htmldiff from- (new)

ELR_EL1, Exception Link Register (EL1)

The ELR_EL1 characteristics are:

Purpose

When taking an exception to EL1, holds the address to return to.

Configuration

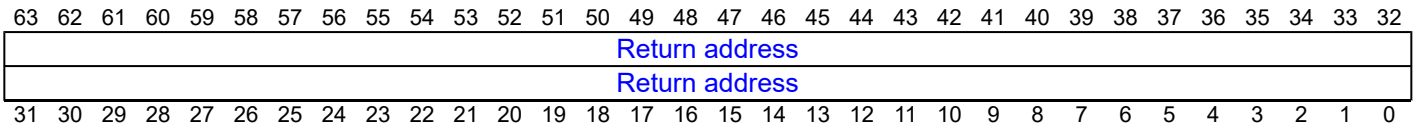
RW fields in this register reset to architecturally UNKNOWN values.

Attributes

ELR_EL1 is a 64-bit register.

Field descriptions

The ELR_EL1 bit assignments are:



Bits [63:0]

Return address.

An exception return from EL1 using AArch64 makes ELR_EL1 become UNKNOWN.

This field resets to an architecturally UNKNOWN value.

Accessing the ELR_EL1

An exception return from EL1 using AArch64 makes ELR_EL1 become UNKNOWN.

When HCR_EL2.E2H is 1, without explicit synchronization, access from EL3 using the mnemonic ELR_EL1 or ELR_EL12 are not guaranteed to be ordered with respect to accesses using the other mnemonic.

Accesses to this register use the following encodings:

```
MRS <Xt>, ELR_EL1
```

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b11	0b000	0b0100	0b0000	0b001
~~0b11~~	~~0b0100~~	~~0b000~~	~~0b001~~	~~0b0000~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV1> == '01' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<NV2,NV1,NV> == '111' then
        return NVMem[0x230];
    else
        return ELR_EL1;
elsif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' then
        return ELR_EL2;
    else
        return ELR_EL1;
elsif PSTATE.EL == EL3 then
    return ELR_EL1;
```

MSR ELR_EL1, <Xt>

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b11	0b000	0b0100	0b0000	0b001
~~0b11~~	~~0b0100~~	~~0b000~~	~~0b001~~	~~0b0000~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV1> == '01' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<NV2,NV1,NV> == '111' then
        NVMem[0x230] = X[t];
    else
        ELR_EL1 = X[t];
elsif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' then
        ELR_EL2 = X[t];
    else
        ELR_EL1 = X[t];
elsif PSTATE.EL == EL3 then
    ELR_EL1 = X[t];
```

MRS <Xt>, ELR_EL12

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b11	0b101	0b0100	0b0000	0b001
~~0b11~~	~~0b0100~~	~~0b101~~	~~0b001~~	~~0b0000~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '101' then
        return NVMem[0x230];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if EL2Enabled() && HCR_EL2.E2H == '1' then
    if PSTATE.EL == EL2 then
        return ELR_EL1;
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if EL2Enabled() && HCR_EL2.E2H == '1' then
        return ELR_EL1;
else
    else
        UNDEFINED;
```

MSR ELR_EL12, <Xt>

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b11	0b101	0b0100	0b0000	0b001
~~0b11~~	~~0b0100~~	~~0b101~~	~~0b001~~	~~0b0000~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '101' then
        NVMem[0x230] = X[t];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if EL2Enabled() && HCR_EL2.E2H == '1' then
    if PSTATE.EL == EL2 then
        ELR_EL1 = X[t];
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if EL2Enabled() && HCR_EL2.E2H == '1' then
        ELR_EL1 = X[t];
else
    else
        UNDEFINED;
```

MRS <Xt>, ELR_EL2

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b11	0b100	0b0100	0b0000	0b001
~~0b11~~	~~0b0100~~	~~0b100~~	~~0b001~~	~~0b0000~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
        return ELR_EL1;
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    return ELR_EL2;
elsif PSTATE.EL == EL3 then
    return ELR_EL2;
```

```
MSR ELR_EL2, <Xt>
```

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b11	0b100	0b0100	0b0000	0b001
~~0b11~~	~~0b0100~~	~~0b100~~	~~0b001~~	~~0b0000~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
        ELR_EL1 = X[t];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    ELR_EL2 = X[t];
elsif PSTATE.EL == EL3 then
    ELR_EL2 = X[t];
```

27~~13~~/03~~12~~/2019~~2018~~ 21~~16~~:59~~42~~; e5e4db499bf9867a4b93324c4dbac985d3da93766~~379d01e197f1d40720d32d0f84e419e9187e009~~

(old) htmldiff from- (new)

(old) htmldiff from- (new)

ELR_EL2, Exception Link Register (EL2)

The ELR_EL2 characteristics are:

Purpose

When taking an exception to EL2, holds the address to return to.

Configuration

AArch64 System register ELR_EL2 bits [31:0] are architecturally mapped to AArch32 System register ELR_hyp[31:0] .

This register has no effect if EL2 is not enabled in the current Security state.

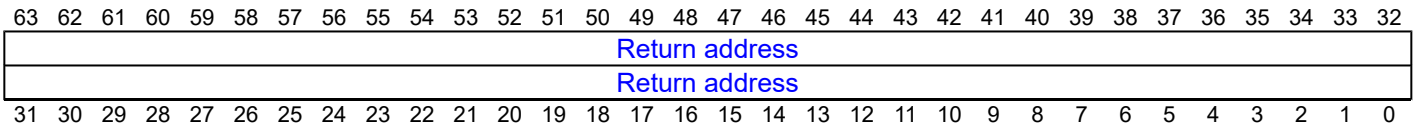
RW fields in this register reset to architecturally UNKNOWN values.

Attributes

ELR_EL2 is a 64-bit register.

Field descriptions

The ELR_EL2 bit assignments are:



Bits [63:0]

Return address.

An exception return from EL2 using AArch64 makes ELR_EL2 become UNKNOWN.

When EL2 is in AArch32 Execution state and an exception is taken from EL0, EL1, or EL2 to EL3 and AArch64 execution, the upper 32-bits of ELR_EL2 are either set to 0 or hold the same value that they did before AArch32 execution. Which option is adopted is determined by an implementation, and might vary dynamically within an implementation. Correspondingly software must regard the value as being an UNKNOWN choice between the two values.

This field resets to an architecturally UNKNOWN value.

Accessing the ELR_EL2

An exception return from EL2 using AArch64 makes ELR_EL2 become UNKNOWN.

When EL2 is in AArch32 Execution state and an exception is taken from EL0, EL1, or EL2 to EL3 and AArch64 execution, the upper 32-bits of ELR_EL2 are either set to 0 or hold the same value that they did before AArch32 execution. Which option is adopted is determined by an implementation, and might vary dynamically within an implementation. Correspondingly software must regard the value as being an UNKNOWN choice between the two values.

When HCR_EL2.E2H is 1, without explicit synchronization, access from EL2 using the mnemonic ELR_EL2 or ELR_EL1 are not guaranteed to be ordered with respect to accesses using the other mnemonic.

Accesses to this register use the following encodings:

MRS <Xt>, ELR_EL2

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b11	0b100	0b0100	0b0000	0b001
~~0b11~~	~~0b0100~~	~~0b100~~	~~0b001~~	~~0b0000~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
        return ELR_EL1;
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    return ELR_EL2;
elsif PSTATE.EL == EL3 then
    return ELR_EL2;
```

MSR ELR_EL2, <Xt>

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b11	0b100	0b0100	0b0000	0b001
~~0b11~~	~~0b0100~~	~~0b100~~	~~0b001~~	~~0b0000~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
        ELR_EL1 = X[t];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    ELR_EL2 = X[t];
elsif PSTATE.EL == EL3 then
    ELR_EL2 = X[t];
```

MRS <Xt>, ELR_EL1

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b11	0b000	0b0100	0b0000	0b001
~~0b11~~	~~0b0100~~	~~0b000~~	~~0b001~~	~~0b0000~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV1> == '01' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<NV2,NV1,NV> == '111' then
        return NVMem[0x230];
    else
        return ELR_EL1;
elsif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' then
        return ELR_EL2;
    else
        return ELR_EL1;
elsif PSTATE.EL == EL3 then
    return ELR_EL1;
```

MSR ELR_EL1, <Xt>

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b11	0b000	0b0100	0b0000	0b001
~~0b11~~	~~0b0100~~	~~0b000~~	~~0b001~~	~~0b0000~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV1> == '01' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<NV2,NV1,NV> == '111' then
        NVMem[0x230] = X[t];
    else
        ELR_EL1 = X[t];
elsif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' then
        ELR_EL2 = X[t];
    else
        ELR_EL1 = X[t];
elsif PSTATE.EL == EL3 then
    ELR_EL1 = X[t];
```

27~~13~~/03~~12~~/2019~~2018~~ 21~~16~~:59~~42~~; e5e4db499bf9867a4b93324c4dbac985d3da93766~~379d01e197f1d40720d32d0f84e419e9187e009~~

(old) htmldiff from- (new)

ELR_EL3, Exception Link Register (EL3)

The ELR_EL3 characteristics are:

Purpose

When taking an exception to EL3, holds the address to return to.

Configuration

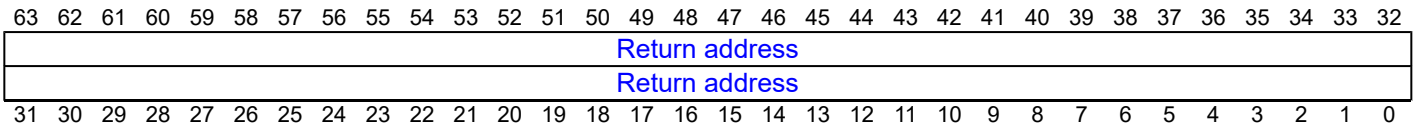
RW fields in this register reset to architecturally UNKNOWN values.

Attributes

ELR_EL3 is a 64-bit register.

Field descriptions

The ELR_EL3 bit assignments are:



Bits [63:0]

Return address.

An exception return from EL3 using AArch64 makes ELR_EL3 become UNKNOWN.

This field resets to an architecturally UNKNOWN value.

Accessing the ELR_EL3

An exception return from EL3 using AArch64 makes ELR_EL3 become UNKNOWN.

Accesses to this register use the following encodings:

```
MRS <Xt>, ELR_EL3
```

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b11	0b110	0b0100	0b0000	0b001
~~0b11~~	~~0b0100~~	~~0b110~~	~~0b001~~	~~0b0000~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    return ELR_EL3;
```

```
MSR ELR_EL3, <Xt>
```

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b11	0b110	0b0100	0b0000	0b001
~~0b11~~	~~0b0100~~	~~0b110~~	~~0b001~~	~~0b0000~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    ELR_EL3 = X[t];
```

27~~13~~/03~~12~~/2019~~2018~~ 21~~16~~:59~~42~~; e5e4db499bf9867a4b93324c4dbac985d3da93766~~379d01e197f1d40720d32d0f84c419e9187e009~~

(old) htmldiff from- (new)

(old) htmldiff from- (new)

ESR_EL1, Exception Syndrome Register (EL1)

The ESR_EL1 characteristics are:

Purpose

Holds syndrome information for an exception taken to EL1.

Configuration

AArch64 System register ESR_EL1 bits [31:0] are architecturally mapped to AArch32 System register DFSR[31:0] .

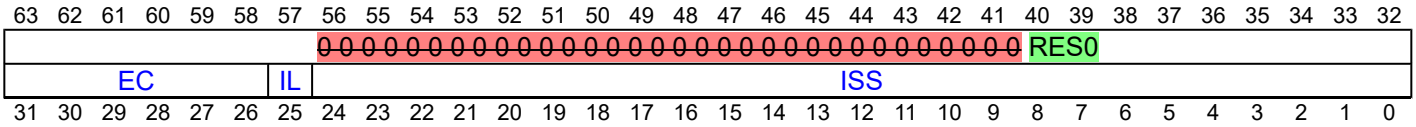
RW fields in this register reset to architecturally UNKNOWN values.

Attributes

ESR_EL1 is a 64-bit register.

Field descriptions

The ESR_EL1 bit assignments are:



ESR_EL1 is made UNKNOWN as a result of an exception return from EL1.

When an UNPREDICTABLE instruction is treated as UNDEFINED, and the exception is taken to EL1, the value of ESR_EL1 is UNKNOWN. The value written to ESR_EL1 must be consistent with a value that could be created as a result of an exception from the same Exception level that generated the exception as a result of a situation that is not UNPREDICTABLE at that Exception level, in order to avoid the possibility of a privilege violation.

Bits [63:32]

Reserved, RES0.

EC, bits [31:26]

Exception Class. Indicates the reason for the exception that this register holds information about.

For each EC value, the table references a subsection that gives information about:

- The cause of the exception, for example the configuration required to enable the trap.
- The encoding of the associated ISS.

Possible values of the EC field are:

EC	Meaning	ISS	Applies when
0b000000	Unknown reason.	ISS encoding for exceptions with an unknown reason.	
0b000001	Trapped WFI or WFE instruction execution. Conditional WFE and WFI instructions that fail their condition code check do not cause an exception.	ISS encoding for an exception from a WFI or WFE instruction.	
0b000011	Trapped MCR or MRC access with (coproc==0b1111) that is not reported using EC 0b000000.	ISS encoding for an exception from an MCR or MRC access.	
0b000100	Trapped MCRR or MRRC access with (coproc==0b1111) that is not reported using EC 0b000000.	ISS encoding for an exception from an MCRR or MRRC access.	
0b000101	Trapped MCR or MRC access with (coproc==0b1110).	ISS encoding for an exception from an MCR or MRC access.	
0b000110	Trapped LDC or STC access. The only architected uses of these instruction are: • An STC to write data to memory from DBGDTRRXint. • An LDC to read data from memory to DBGDTRTXint.	ISS encoding for an exception from an LDC or STC instruction.	
0b000111	Access to SVE, Advanced SIMD, or floating-point functionality trapped by CPACR_EL1.FPEN, CPTR_EL2.FPEN, CPTR_EL2.TFP, or CPTR_EL3.TFP control. Excludes exceptions resulting from CPACR_EL1 when the value of HCR_EL2.TGE is 1, or because SVE or Advanced SIMD and floating-point are not implemented. These are reported with EC value 0b000000 as described in 'EC encodings when routing exceptions to EL2' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section D1.10.4.	ISS encoding for an exception from an access to SVE, Advanced SIMD or floating-point functionality, resulting from CPACR_EL1.FPEN, CPTR_EL2.FPEN or CPTR_ELx.TFP.	
0b001100	Trapped MRRC access with (coproc==0b1110).	ISS encoding for an exception from an MCRR or MRRC access.	
0b001101	Branch Target Exception.	ISS encoding for an exception from Branch Target Identification instruction.	When ARMv8.5-BTI is implemented
0b001110	Illegal Execution state.	ISS encoding for an exception from an Illegal Execution state, or a PC or SP alignment fault.	
0b010001	SVC instruction execution in AArch32 state. This is reported in ESR_EL2 only when the exception is	ISS encoding for an exception from HVC or SVC instruction execution.	

	generated because the value of HCR_EL2.TGE is 1.	
0b010101	SVC instruction execution in AArch64 state.	ISS encoding for an exception from HVC or SVC instruction execution.
0b011000	Trapped MSR, MRS or System instruction execution in AArch64 state, that is not reported using EC 0b000000, 0b000001 or 0b000111. If ARMv8.4-IDST is implemented, also exceptions generated on a read of an ID register. If ARMv8.0-CSV2 is implemented, also Cache Speculation Variant exceptions. This includes all instructions that cause exceptions that are part of the encoding space defined in 'System instruction class encoding overview' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section C5.2.2, except for those exceptions reported using EC values 0b000000, 0b000001, or 0b000111.	ISS encoding for an exception from MSR, MRS, or System instruction execution in AArch64 state.
0b011001	Access to SVE functionality trapped as a result of CPACR_EL1.ZEN, CPTR_EL2.ZEN, CPTR_EL2.TZ, or CPTR_EL3.EZ, that is not reported using EC 0b000000. This EC is defined only if SVE is implemented.	ISS encoding for an exception from an access to SVE functionality, resulting from CPACR_EL1.ZEN, CPTR_EL2.ZEN, CPTR_EL2.TZ, or CPTR_EL3.EZ.
0b100000	Instruction Abort from a lower Exception level, that might be using AArch32 or AArch64. Used for MMU faults generated by instruction accesses and synchronous External aborts, including synchronous parity or ECC errors. Not used for debug related exceptions.	ISS encoding for an exception from an Instruction Abort.
0b100001	Instruction Abort taken without a change in Exception level. Used for MMU faults generated by instruction accesses and synchronous External aborts, including synchronous parity or ECC errors. Not used for debug related exceptions.	ISS encoding for an exception from an Instruction Abort.
0b100010	PC alignment fault exception.	ISS encoding for an exception from an Illegal Execution state, or a PC or SP alignment fault.
0b100100	Data Abort from a lower Exception level, that might be using AArch32 or AArch64. Used for MMU faults generated by data accesses, alignment faults other than those caused by Stack Pointer misalignment, and synchronous External	ISS encoding for an exception from a Data Abort.

	aborts, including synchronous parity or ECC errors. Not used for debug related exceptions.	
0b100101	Data Abort taken without a change in Exception level. Used for MMU faults generated by data accesses, alignment faults other than those caused by Stack Pointer misalignment, and synchronous External aborts, including synchronous parity or ECC errors. Not used for debug related exceptions.	ISS encoding for an exception from a Data Abort.
0b100110	SP alignment fault exception.	ISS encoding for an exception from an Illegal Execution state, or a PC or SP alignment fault.
0b101000	Trapped floating-point exception taken from AArch32 state. This EC value is valid if the implementation supports trapping of floating-point exceptions, otherwise it is reserved. Whether a floating-point implementation supports trapping of floating-point exceptions is IMPLEMENTATION DEFINED.	ISS encoding for an exception from a trapped floating-point exception.
0b101100	Trapped floating-point exception taken from AArch64 state. This EC value is valid if the implementation supports trapping of floating-point exceptions, otherwise it is reserved. Whether a floating-point implementation supports trapping of floating-point exceptions is IMPLEMENTATION DEFINED.	ISS encoding for an exception from a trapped floating-point exception.
0b101111	SError interrupt.	ISS encoding for a SError interrupt.
0b110000	Breakpoint exception from a lower Exception level, that might be using AArch32 or AArch64.	ISS encoding for an exception from a Breakpoint or Vector Catch debug exception.
0b110001	Breakpoint exception taken without a change in Exception level.	ISS encoding for an exception from a Breakpoint or Vector Catch debug exception.
0b110010	Software Step exception from a lower Exception level, that might be using AArch32 or AArch64.	ISS encoding for an exception from a Software Step exception.
0b110011	Software Step exception taken without a change in Exception level.	ISS encoding for an exception from a Software Step exception.
0b110100	Watchpoint exception from a lower Exception level, that might be using AArch32 or AArch64.	ISS encoding for an exception from a Watchpoint exception.
0b110101	Watchpoint exception taken without a change in Exception level.	ISS encoding for an exception from a Watchpoint exception.

0b111000	BKPT instruction execution in AArch32 state.	ISS encoding for an exception from execution of a Breakpoint instruction.
0b111100	BRK instruction execution in AArch64 state. This is reported in ESR_EL3 only if a BRK instruction is executed.	ISS encoding for an exception from execution of a Breakpoint instruction.

All other EC values are reserved by Arm, and:

- Unused values in the range 0b000000 - 0b101100 (0x00 - 0x2C) are reserved for future use for synchronous exceptions.
- Unused values in the range 0b101101 - 0b111111 (0x2D - 0x3F) are reserved for future use, and might be used for synchronous or asynchronous exceptions.

The effect of programming this field to a reserved value is that behavior is CONSTRAINED UNPREDICTABLE, as described in 'Reserved values in System and memory-mapped registers and translation table entries' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section K1.1.11.

This field resets to an architecturally UNKNOWN value.

IL, bit [25]

Instruction Length for synchronous exceptions. Possible values of this bit are:

IL	Meaning
0b0	16-bit instruction trapped.
0b1	• An SError interrupt. • An Instruction Abort exception. • A PC alignment fault exception. • An SP alignment fault exception. • A Data Abort exception for which the value of the ISV bit is 0. • An Illegal Execution state exception. • Any debug exception except for Breakpoint instruction exceptions. For Breakpoint instruction exceptions, this bit has its standard meaning: ◦ 0b0: 16-bit T32 BKPT instruction. ◦ 0b1: 32-bit A32 BKPT instruction or A64 BRK instruction. • An exception reported using EC value 0b000000.

This field resets to an architecturally UNKNOWN value.

ISS, bits [24:0]

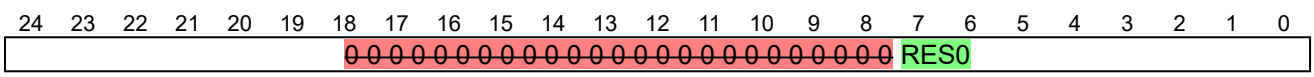
Instruction Specific Syndrome. Architecturally, this field can be defined independently for each defined Exception class. However, in practice, some ISS encodings are used for more than one Exception class.

Typically, an ISS encoding has a number of subfields. When an ISS subfield holds a register number, the value returned in that field is the AArch64 view of the register number. For an exception taken from AArch32 state, 'Mapping of the general-purpose registers between the Execution states' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section D1.20.1, defines this view of the specified AArch32 register. If the AArch32 register descriptor is 0b1111, then:

- If the instruction that generated the exception was not UNPREDICTABLE, the field takes the value 0b11111.
- If the instruction that generated the exception was UNPREDICTABLE, the field takes an UNKNOWN value that must be either:
  - The AArch64 view of the register number of a register that might have been used at the Exception level from which the exception was taken.
  - The value 0b11111.

When the EC field is 0b000000, indicating an exception with an unknown reason, the ISS field is not valid, RES0.

ISS encoding for exceptions with an unknown reason.



Bits [24:0]

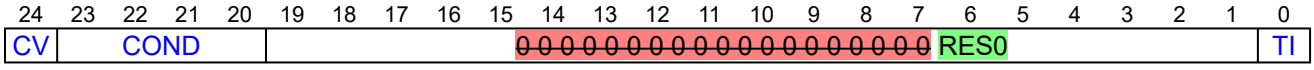
Reserved, RES0.

When an exception is reported using this EC code the IL field is set to 1.

This EC code is used for all exceptions that are not covered by any other EC value. This includes exceptions that are generated in the following situations:

- The attempted execution of an instruction bit pattern that has no allocated instruction at the current Exception level and Security state, including:
  - A read access using a System register pattern that is not allocated for reads at the current Exception level and Security state.
  - A write access using a System register pattern that is not allocated for writes at the current Exception level and Security state.
  - Instruction encodings for instructions not implemented in the implementation.
- In Debug state, the attempted execution of an instruction bit pattern that is unallocated in Debug state.
- In Non-debug state, the attempted execution of an instruction bit pattern that is unallocated in Non-debug state.
- In AArch32 state, attempted execution of a short vector floating-point instruction.
- In an implementation that does not include Advanced SIMD and floating-point functionality, an attempted access to Advanced SIMD or floating-point functionality under conditions where that access would be permitted if that functionality was present. This includes the attempted execution of an Advanced SIMD or floating-point instruction, and attempted accesses to Advanced SIMD and floating-point System registers.
- An exception generated because of the value of one of the SCTLR_EL1.{ITD, SED, CP15BEN} control bits.
- Attempted execution of:
  - An HVC instruction when disabled by HCR_EL2.HCD or SCR_EL3.HCE.
  - An SMC instruction when disabled by SCR_EL3.SMD.
  - An HLT instruction when disabled by EDSCR.HDE.
- Attempted execution of an MSR or MRS instruction to access SP_EL0 when the value of SPSel.SP is 0.
- Attempted execution, in Debug state, of:
  - A DCPS1 instruction when the value of HCR_EL2.TGE is 1 and EL2 is disabled or not implemented in the current Security state.
  - A DCPS2 instruction from EL1 or EL0 when EL2 is disabled or not implemented in the current Security state.
  - A DCPS3 instruction when the value of EDSCR.SDD is 1, or when EL3 is not implemented.
- When EL3 is using AArch64, attempted execution from Secure EL1 of an SRS instruction using R13_mon. See 'Traps to EL3 of monitor functionality from Secure EL1 using AArch32' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section D1 (The AArch64 System Level Programmers' Model).
- In Debug state when the value of EDSCR.SDD is 1, the attempted execution at EL2, EL1, or EL0 of an instruction that is configured to trap to EL3.
- In AArch32 state, the attempted execution of an MRS (banked register) or an MSR (banked register) instruction to SPSR_mon, SP_mon, or LR_mon.
- An exception that is taken to EL2 because the value of HCR_EL2.TGE is 1 that, if the value of HCR_EL2.TGE was 0 would have been reported with an ESR_ELx.EC value of `0b000111`.
- When SVE is not implemented, attempted execution of:
  - An SVE instruction.
  - An MSR or MRS instruction to access ZCR_EL1, ZCR_EL2, or ZCR_EL3.

ISS encoding for an exception from a WFI or WFE instruction.



CV, bit [24]

Condition code valid. Possible values of this bit are:

CV	Meaning
`0b0`	The COND field is not valid.
`0b1`	The COND field is valid.

For exceptions taken from AArch64, CV is set to 1.

For exceptions taken from AArch32:

- When an A32 instruction is trapped, CV is set to 1.
- When a T32 instruction is trapped, it is IMPLEMENTATION DEFINED whether CV is set to 1 or set to 0. See the description of the COND field for more information.

This field resets to an architecturally UNKNOWN value.

COND, bits [23:20]

The condition code for the trapped instruction. This field is valid only for exceptions taken from AArch32, and only when the value of CV is 1.

For exceptions taken from AArch64, this field is set to `0b1110`.

For exceptions taken from AArch32:

- When an A32 instruction is trapped, CV is set to 1 and:
  - If the instruction is conditional, COND is set to the condition code field value from the instruction.
  - If the instruction is unconditional, COND is set to `0b1110`.
- A conditional A32 instruction that is known to pass its condition code check can be presented either:
  - With COND set to `0b1110`, the value for unconditional.
  - With the COND value held in the instruction.
- When a T32 instruction is trapped, it is IMPLEMENTATION DEFINED whether:
  - CV is set to 0 and COND is set to an UNKNOWN value. Software must examine the SPSR.IT field to determine the condition, if any, of the T32 instruction.
  - CV is set to 1 and COND is set to the condition code for the condition that applied to the instruction.
- For an implementation that, for both A32 and T32 instructions, takes an exception on a trapped conditional instruction only if the instruction passes its condition code check, these definitions mean that when CV is set to 1 it is IMPLEMENTATION DEFINED whether the COND field is set to `0b1110`, or to the value of any condition that applied to the instruction.

This field resets to an architecturally UNKNOWN value.

Bits [19:1]

Reserved, RES0.

TI, bit [0]

Trapped instruction. Possible values of this bit are:

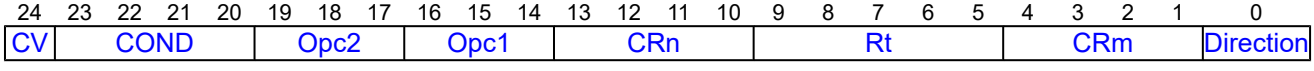
TI	Meaning
`0b0`	WFI trapped.
`0b1`	WFE trapped.

This field resets to an architecturally UNKNOWN value.

The following sections describe configuration settings for generating this exception:

- 'Controls for exceptions taken to EL1 using AArch64' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section D1 (The AArch64 System Level Programmers' Model).
- 'Traps to EL2 of Non-secure EL1 and EL0 execution of WFE and WFI instructions' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section D1.
- 'Traps to EL3 of EL2, EL1, and EL0 execution of WFE and WFI instructions' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section D1.

ISS encoding for an exception from an MCR or MRC access.



CV, bit [24]

Condition code valid. Possible values of this bit are:

CV	Meaning
`0b0`	The COND field is not valid.
`0b1`	The COND field is valid.

For exceptions taken from AArch64, CV is set to 1.

For exceptions taken from AArch32:

- When an A32 instruction is trapped, CV is set to 1.
- When a T32 instruction is trapped, it is IMPLEMENTATION DEFINED whether CV is set to 1 or set to 0. See the description of the COND field for more information.

This field resets to an architecturally UNKNOWN value.

COND, bits [23:20]

The condition code for the trapped instruction. This field is valid only for exceptions taken from AArch32, and only when the value of CV is 1.

For exceptions taken from AArch64, this field is set to `0b1110`.

For exceptions taken from AArch32:

- When an A32 instruction is trapped, CV is set to 1 and:
  - If the instruction is conditional, COND is set to the condition code field value from the instruction.
  - If the instruction is unconditional, COND is set to `0b1110`.
- A conditional A32 instruction that is known to pass its condition code check can be presented either:
  - With COND set to `0b1110`, the value for unconditional.
  - With the COND value held in the instruction.
- When a T32 instruction is trapped, it is IMPLEMENTATION DEFINED whether:
  - CV is set to 0 and COND is set to an UNKNOWN value. Software must examine the SPSR.IT field to determine the condition, if any, of the T32 instruction.
  - CV is set to 1 and COND is set to the condition code for the condition that applied to the instruction.
- For an implementation that, for both A32 and T32 instructions, takes an exception on a trapped conditional instruction only if the instruction passes its condition code check, these definitions mean that when CV is set to 1 it is IMPLEMENTATION DEFINED whether the COND field is set to `0b1110`, or to the value of any condition that applied to the instruction.

This field resets to an architecturally UNKNOWN value.

Opc2, bits [19:17]

The Opc2 value from the issued instruction.

For a trapped VMRS access, holds the value `0b000`.

This field resets to an architecturally UNKNOWN value.

Opc1, bits [16:14]

The Opc1 value from the issued instruction.

For a trapped VMRS access, holds the value `0b111`.

This field resets to an architecturally UNKNOWN value.

CRn, bits [13:10]

The CRn value from the issued instruction.

For a trapped VMRS access, holds the reg field from the VMRS instruction encoding.

This field resets to an architecturally UNKNOWN value.

Rt, bits [9:5]

The Rt value from the issued instruction, the general-purpose register used for the transfer. The reported value gives the AArch64 view of the register. See 'Mapping of the general-purpose registers between the Execution states' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section D1.20.1.

This field resets to an architecturally UNKNOWN value.

CRm, bits [4:1]

The CRm value from the issued instruction.

For a trapped VMRS access, holds the value `0b0000`.

This field resets to an architecturally UNKNOWN value.

Direction, bit [0]

Indicates the direction of the trapped instruction. The possible values of this bit are:

Direction	Meaning
`0b0`	Write to System register space. MCR instruction.
`0b1`	Read from System register space. MRC or VMRS instruction.

This field resets to an architecturally UNKNOWN value.

The following sections describe configuration settings for generating exceptions that are reported using EC value `0b000011`:

- 'Traps to EL1 of EL0 accesses to the Generic Timer registers' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section D1 (The AArch64 System Level Programmers' Model).
- 'Traps to EL1 of EL0 accesses to Performance Monitors registers' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section D1 (The AArch64 System Level Programmers' Model).
- 'Traps to EL1 of EL0 accesses to Activity Monitors registers' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section D1 (The AArch64 System Level Programmers' Model).
- 'Traps to EL2 of Non-secure EL1 accesses to virtual memory control registers' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section D1.
- 'Traps to EL2 of Non-secure EL1 execution of TLB maintenance instructions' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section D1.
- 'Traps to EL2 of Non-secure EL1 and EL0 execution of cache maintenance instructions' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section D1.
- 'Traps to EL2 of Non-secure EL1 and EL0 accesses to the Auxiliary Control Register' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section D1.
- 'Traps to EL2 of Non-secure EL1 and EL0 accesses to lockdown, DMA, and TCM operations' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section D1.
- 'Traps to EL2 of Non-secure EL1 and EL0 accesses to the ID registers' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section D1 (The AArch64 System Level Programmers' Model).
- 'Trapping to EL2 of Non-secure EL1 accesses to the CPACR_EL1 or CPACR' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section D1.
- 'Generic trapping to EL2 of Non-secure EL1 and EL0 accesses to System registers, from AArch32 state only' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section D1
- 'Traps to EL2 of Non-secure EL0 and EL1 accesses to the Generic Timer registers' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section D1
- 'Traps to EL2 of Non-secure EL1 and EL0 accesses to Performance Monitors registers' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section D1.
- 'Traps to EL2 of EL1 and EL0 accesses to Activity Monitors registers' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section D1 (The AArch64 System Level Programmers' Model).
- 'Traps to EL3 of Secure monitor functionality from Secure EL1 using AArch32' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section D1 (The AArch64 System Level Programmers' Model).
- 'Trapping to EL3 of EL2 accesses to the CPTR_EL2 or HCPTR, and EL2 and EL1 accesses to the CPACR_EL1 or CPACR' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section D1 (The AArch64 System Level Programmers' Model).
- 'Traps to EL3 of EL2, EL1, and EL0 accesses to Performance Monitors registers' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section D1 (The AArch64 System Level Programmers' Model).
- 'Traps to EL3 of EL2, EL1, and EL0 accesses to Activity Monitors registers' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section D1 (The AArch64 System Level Programmers' Model).

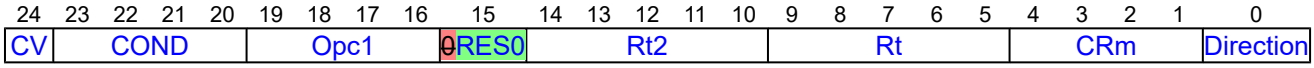
The following sections describe configuration settings for generating exceptions that are reported using EC value `0b000101`:

- 'Traps to EL1 of EL0 and EL1 System register accesses to the trace registers' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section D1.
- 'Traps to EL1 of EL0 accesses to the Debug Communications Channel (DCC) registers' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section D1.
- 'Traps to EL2 of Non-secure EL1 and EL0 accesses to the ID registers' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section D1, for trapped accesses to the JIDR.
- 'Traps to EL2 of Non-secure system register accesses to the trace registers' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section D1.

- 'Trapping System register accesses to Debug ROM registers to EL2' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section D1.
- 'Trapping System register accesses to OS-related debug registers to EL2' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section D1.
- 'Trapping general System register accesses to debug registers to EL2' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section D1.
- 'Traps to EL3 of all System register accesses to the trace registers' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section D1.
- 'Trapping System register accesses to OS-related debug registers to EL3' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section D1.
- 'Trapping general System register accesses to debug registers to EL3' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section D1.

'Traps to EL2 of Non-secure EL1 and EL0 accesses to the ID registers' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section D1, describes configuration settings for generating exceptions that are reported using EC value `0b001000`.

ISS encoding for an exception from an MCRR or MRRC access.



CV, bit [24]

Condition code valid. Possible values of this bit are:

CV	Meaning
`0b0`	The COND field is not valid.
`0b1`	The COND field is valid.

For exceptions taken from AArch64, CV is set to 1.

For exceptions taken from AArch32:

- When an A32 instruction is trapped, CV is set to 1.
- When a T32 instruction is trapped, it is IMPLEMENTATION DEFINED whether CV is set to 1 or set to 0. See the description of the COND field for more information.

This field resets to an architecturally UNKNOWN value.

COND, bits [23:20]

The condition code for the trapped instruction. This field is valid only for exceptions taken from AArch32, and only when the value of CV is 1.

For exceptions taken from AArch64, this field is set to `0b1110`.

For exceptions taken from AArch32:

- When an A32 instruction is trapped, CV is set to 1 and:
  - If the instruction is conditional, COND is set to the condition code field value from the instruction.
  - If the instruction is unconditional, COND is set to `0b1110`.
- A conditional A32 instruction that is known to pass its condition code check can be presented either:
  - With COND set to `0b1110`, the value for unconditional.
  - With the COND value held in the instruction.
- When a T32 instruction is trapped, it is IMPLEMENTATION DEFINED whether:
  - CV is set to 0 and COND is set to an UNKNOWN value. Software must examine the SPSR.IT field to determine the condition, if any, of the T32 instruction.
  - CV is set to 1 and COND is set to the condition code for the condition that applied to the instruction.
- For an implementation that, for both A32 and T32 instructions, takes an exception on a trapped conditional instruction only if the instruction passes its condition code check, these definitions mean that when CV is set to 1 it is IMPLEMENTATION DEFINED whether the COND field is set to `0b1110`, or to the value of any condition that applied to the instruction.

This field resets to an architecturally UNKNOWN value.

Opc1, bits [19:16]

The Opc1 value from the issued instruction.

This field resets to an architecturally UNKNOWN value.

Bit [15]

Reserved, RES0.

Rt2, bits [14:10]

The Rt2 value from the issued instruction, the second general-purpose register used for the transfer. The reported value gives the AArch64 view of the register. See 'Mapping of the general-purpose registers between the Execution states' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section D1.20.1.

This field resets to an architecturally UNKNOWN value.

Rt, bits [9:5]

The Rt value from the issued instruction, the first general-purpose register used for the transfer. The reported value gives the AArch64 view of the register. See 'Mapping of the general-purpose registers between the Execution states' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section D1.20.1.

This field resets to an architecturally UNKNOWN value.

CRm, bits [4:1]

The CRm value from the issued instruction.

This field resets to an architecturally UNKNOWN value.

Direction, bit [0]

Indicates the direction of the trapped instruction. The possible values of this bit are:

Direction	Meaning
0b0	Write to System register space. MCRR instruction.
0b1	Read from System register space. MRRC instruction.

This field resets to an architecturally UNKNOWN value.

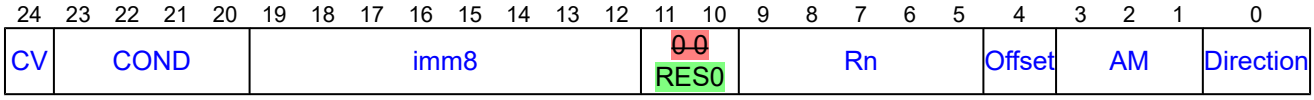
The following sections describe configuration settings for generating exceptions that are reported using EC value 0b000100:

- 'Traps to EL1 of EL0 accesses to the Generic Timer registers' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section D1 (The AArch64 System Level Programmers' Model).
- 'Traps to EL1 of EL0 accesses to Performance Monitors registers' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section D1 (The AArch64 System Level Programmers' Model).
- 'Traps to EL1 of EL0 accesses to Activity Monitors registers' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section D1 (The AArch64 System Level Programmers' Model).
- 'Traps to EL2 of Non-secure EL1 accesses to virtual memory control registers' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section D1.
- 'General trapping to EL2 of Non-secure EL0 and EL1 accesses to System registers, from AArch32 state only' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section D1 (The AArch64 System Level Programmers' Model).
- 'Traps to EL2 of Non-secure EL0 and EL1 accesses to the Generic Timer registers' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section D1.
- 'Traps to EL2 of Non-secure EL0 and EL1 accesses to Performance Monitors registers' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section D1.
- 'Traps to EL2 of EL1 and EL0 accesses to Activity Monitors registers' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section D1 (The AArch64 System Level Programmers' Model).
- 'Traps to EL3 of EL2, EL1, and EL0 accesses to Performance Monitors registers' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section D1 (The AArch64 System Level Programmers' Model).
- 'Traps to EL3 of EL2, EL1, and EL0 accesses to Activity Monitors registers' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section D1 (The AArch64 System Level Programmers' Model).

The following sections describe configuration settings for generating exceptions that are reported using EC value `0b001100`:

- 'Traps to EL1 of EL0 and EL1 System register accesses to the trace registers' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section D1.
- 'Traps to EL1 of EL0 accesses to the Debug Communications Channel (DCC) registers' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section D1.
- 'Traps to EL2 of Non-secure system register accesses to the trace registers' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section D1.
- 'Trapping System register accesses to Debug ROM registers' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section D1.
- 'Traps to EL3 of all System register accesses to the trace registers' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section D1.
- 'Trapping System register accesses to powerdown debug registers to EL3' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section D1.
- 'Trapping general System register accesses to debug registers to EL3' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section D1.

ISS encoding for an exception from an LDC or STC instruction.



CV, bit [24]

Condition code valid. Possible values of this bit are:

CV	Meaning
`0b0`	The COND field is not valid.
`0b1`	The COND field is valid.

For exceptions taken from AArch64, CV is set to 1.

For exceptions taken from AArch32:

- When an A32 instruction is trapped, CV is set to 1.
- When a T32 instruction is trapped, it is IMPLEMENTATION DEFINED whether CV is set to 1 or set to 0. See the description of the COND field for more information.

This field resets to an architecturally UNKNOWN value.

COND, bits [23:20]

The condition code for the trapped instruction. This field is valid only for exceptions taken from AArch32, and only when the value of CV is 1.

For exceptions taken from AArch64, this field is set to `0b1110`.

For exceptions taken from AArch32:

- When an A32 instruction is trapped, CV is set to 1 and:
  - If the instruction is conditional, COND is set to the condition code field value from the instruction.
  - If the instruction is unconditional, COND is set to `0b1110`.
- A conditional A32 instruction that is known to pass its condition code check can be presented either:
  - With COND set to `0b1110`, the value for unconditional.
  - With the COND value held in the instruction.
- When a T32 instruction is trapped, it is IMPLEMENTATION DEFINED whether:
  - CV is set to 0 and COND is set to an UNKNOWN value. Software must examine the SPSR.IT field to determine the condition, if any, of the T32 instruction.
  - CV is set to 1 and COND is set to the condition code for the condition that applied to the instruction.
- For an implementation that, for both A32 and T32 instructions, takes an exception on a trapped conditional instruction only if the instruction passes its condition code check, these definitions mean that when CV is set to 1 it is IMPLEMENTATION DEFINED whether the COND field is set to `0b1110`, or to the value of any condition that applied to the instruction.

This field resets to an architecturally UNKNOWN value.

imm8, bits [19:12]

The immediate value from the issued instruction.

This field resets to an architecturally UNKNOWN value.

Bits [11:10]

Reserved, RES0.

Rn, bits [9:5]

The Rn value from the issued instruction, the general-purpose register used for the transfer. The reported value gives the AArch64 view of the register. See 'Mapping of the general-purpose registers between the Execution states' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section D1.20.1.

This field is valid only when AM[2] is 0, indicating an immediate form of the LDC or STC instruction. When AM[2] is 1, indicating a literal form of the LDC or STC instruction, this field is UNKNOWN.

This field resets to an architecturally UNKNOWN value.

Offset, bit [4]

Indicates whether the offset is added or subtracted:

Offset	Meaning
0b0	Subtract offset.
0b1	Add offset.

This bit corresponds to the U bit in the instruction encoding.

This field resets to an architecturally UNKNOWN value.

AM, bits [3:1]

Addressing mode. The permitted values of this field are:

AM	Meaning
0b000	Immediate unindexed.
0b001	Immediate post-indexed.
0b010	Immediate offset.
0b011	Immediate pre-indexed.
0b100	For a trapped STC instruction or a trapped T32 LDC instruction this encoding is reserved.
0b110	For a trapped STC instruction, this encoding is reserved.

The values 0b101 and 0b111 are reserved. The effect of programming this field to a reserved value is that behavior is CONSTRAINED UNPREDICTABLE, as described in 'Reserved values in AArch64 System registers and translation table entries' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section K1.2.2.

Bit [2] in this subfield indicates the instruction form, immediate or literal.

Bits [1:0] in this subfield correspond to the bits {P, W} in the instruction encoding.

This field resets to an architecturally UNKNOWN value.

Direction, bit [0]

Indicates the direction of the trapped instruction. The possible values of this bit are:

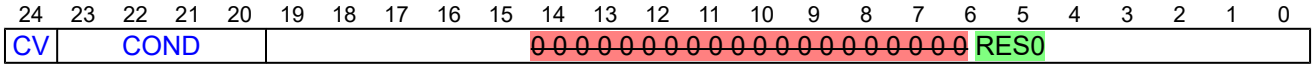
Direction	Meaning
0b0	Write to memory. STC instruction.
0b1	Read from memory. LDC instruction.

This field resets to an architecturally UNKNOWN value.

The following sections describe the configuration settings for the traps that are reported using EC value 0b000110:

- 'Traps to EL1 of EL0 accesses to the Debug Communications Channel (DCC) registers' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section D1.
- 'Trapping general System register accesses to debug registers to EL2' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section D1.
- 'Trapping general System register accesses to debug registers to EL3' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section D1.

ISS encoding for an exception from an access to SVE, Advanced SIMD or floating-point functionality, resulting from CPACR_EL1.FPEN, CPTR_EL2.FPEN or CPTR_ELx.TFP.



The accesses covered by this trap include:

- Execution of SVE or Advanced SIMD and floating-point instructions.
- Accesses to the Advanced SIMD and floating-point System registers.

For an implementation that does not include either SVE or support for floating-point and Advanced SIMD, the exception is reported using the EC value 0b000000.

CV, bit [24]

Condition code valid. Possible values of this bit are:

CV	Meaning
0b0	The COND field is not valid.
0b1	The COND field is valid.

For exceptions taken from AArch64, CV is set to 1.

For exceptions taken from AArch32:

- When an A32 instruction is trapped, CV is set to 1.
- When a T32 instruction is trapped, it is IMPLEMENTATION DEFINED whether CV is set to 1 or set to 0. See the description of the COND field for more information.

This field resets to an architecturally UNKNOWN value.

COND, bits [23:20]

The condition code for the trapped instruction. This field is valid only for exceptions taken from AArch32, and only when the value of CV is 1.

For exceptions taken from AArch64, this field is set to 0b1110.

For exceptions taken from AArch32:

- When an A32 instruction is trapped, CV is set to 1 and:
  - If the instruction is conditional, COND is set to the condition code field value from the instruction.
  - If the instruction is unconditional, COND is set to 0b1110.
- A conditional A32 instruction that is known to pass its condition code check can be presented either:
  - With COND set to 0b1110, the value for unconditional.
  - With the COND value held in the instruction.
- When a T32 instruction is trapped, it is IMPLEMENTATION DEFINED whether:
  - CV is set to 0 and COND is set to an UNKNOWN value. Software must examine the SPSR.IT field to determine the condition, if any, of the T32 instruction.
  - CV is set to 1 and COND is set to the condition code for the condition that applied to the instruction.
- For an implementation that, for both A32 and T32 instructions, takes an exception on a trapped conditional instruction only if the instruction passes its condition code check, these definitions mean that when CV is set to 1 it is IMPLEMENTATION DEFINED whether the COND field is set to 0b1110, or to the value of any condition that applied to the instruction.

This field resets to an architecturally UNKNOWN value.

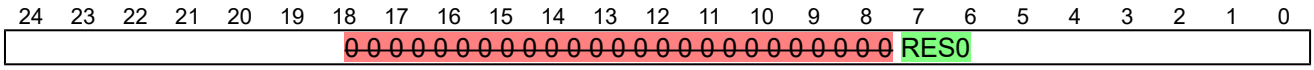
Bits [19:0]

Reserved, RES0.

The following sections describe the configuration settings for the traps that are reported using EC value `0b000111`:

- 'Traps to EL1 of EL0 and EL1 accesses to SIMD and floating-point functionality' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section D1.
- 'General trapping to EL2 of Non-secure accesses to the SIMD and floating-point registers' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section D1.
- 'Traps to EL3 of all accesses to the SIMD and floating-point registers' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section D1

ISS encoding for an exception from an access to SVE functionality, resulting from CPACR_EL1.ZEN, CPTR_EL2.ZEN, CPTR_EL2.TZ, or CPTR_EL3.EZ.



Bits [24:0]

When SVE is implemented:

Reserved, RES0.

Otherwise:

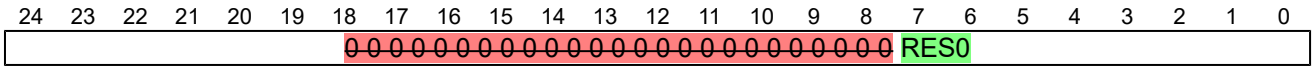
Reserved, RES0.

The accesses covered by this trap include:

- Execution of SVE instructions.
- Accesses to the SVE system registers, ZCR_ELx and ID_AA64ZFR0_EL1.

For an implementation that does not include SVE, the exception is reported using the EC value `0b000000`.

ISS encoding for an exception from an Illegal Execution state, or a PC or SP alignment fault.



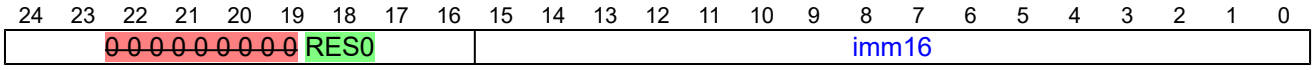
Bits [24:0]

Reserved, RES0.

There are no configuration settings for generating Illegal Execution state exceptions and PC alignment fault exceptions. For more information about these exceptions see 'The Illegal Execution state exception' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section D1 and 'PC alignment checking' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section D1.

'Stack pointer alignment checking' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section D1 describes the configuration settings for generating SP alignment fault exceptions.

ISS encoding for an exception from HVC or SVC instruction execution.



Bits [24:16]

Reserved, RES0.

imm16, bits [15:0]

The value of the immediate field from the HVC or SVC instruction.

For an HVC instruction, and for an A64 SVC instruction, this is the value of the imm16 field of the issued instruction.

For an A32 or T32 SVC instruction:

- If the instruction is unconditional, then:
  - For the T32 instruction, this field is zero-extended from the imm8 field of the instruction.
  - For the A32 instruction, this field is the bottom 16 bits of the imm24 field of the instruction.
- If the instruction is conditional, this field is UNKNOWN.

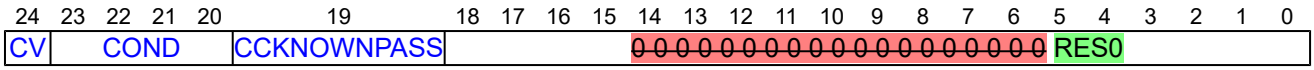
This field resets to an architecturally UNKNOWN value.

In AArch32 state, the HVC instruction is unconditional, and a conditional SVC instruction generates an exception only if it passes its condition code check. Therefore, the syndrome information for these exceptions does not require conditionality information.

For T32 and A32 instructions, see 'SVC' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section F7 (T32 and A32 Base Instruction Set Instruction Descriptions) and 'HVC' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section F7.

For A64 instructions, see 'SVC' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section C5 (A64 Base Instruction Descriptions), and 'HVC' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section C5.

ISS encoding for an exception from SMC instruction execution in AArch32 state.



For an SMC instruction that completes normally and generates an exception that is taken to EL3, the ISS encoding is RES0.

For an SMC instruction that is trapped to EL2 from EL1 because HCR_EL2.TSC is 1, the ISS encoding is as shown in the diagram.

CV, bit [24]

Condition code valid. Possible values of this bit are:

CV	Meaning
0b0	The COND field is not valid.
0b1	The COND field is valid.

For exceptions taken from AArch64, CV is set to 1.

For exceptions taken from AArch32:

- When an A32 instruction is trapped, CV is set to 1.
- When a T32 instruction is trapped, it is IMPLEMENTATION DEFINED whether CV is set to 1 or set to 0. See the description of the COND field for more information.

This field is only valid if CCKNOWNPASS is 1, otherwise it is RES0.

This field resets to an architecturally UNKNOWN value.

COND, bits [23:20]

The condition code for the trapped instruction. This field is valid only for exceptions taken from AArch32, and only when the value of CV is 1.

For exceptions taken from AArch64, this field is set to 0b1110.

For exceptions taken from AArch32:

- When an A32 instruction is trapped, CV is set to 1 and:
  - If the instruction is conditional, COND is set to the condition code field value from the instruction.
  - If the instruction is unconditional, COND is set to `0b1110`.
- A conditional A32 instruction that is known to pass its condition code check can be presented either:
  - With COND set to `0b1110`, the value for unconditional.
  - With the COND value held in the instruction.
- When a T32 instruction is trapped, it is IMPLEMENTATION DEFINED whether:
  - CV is set to 0 and COND is set to an UNKNOWN value. Software must examine the SPSR.IT field to determine the condition, if any, of the T32 instruction.
  - CV is set to 1 and COND is set to the condition code for the condition that applied to the instruction.
- For an implementation that, for both A32 and T32 instructions, takes an exception on a trapped conditional instruction only if the instruction passes its condition code check, these definitions mean that when CV is set to 1 it is IMPLEMENTATION DEFINED whether the COND field is set to `0b1110`, or to the value of any condition that applied to the instruction.

This field is only valid if CCKNOWNPASS is 1, otherwise it is RES0.

This field resets to an architecturally UNKNOWN value.

CCKNOWNPASS, bit [19]

Indicates whether the instruction might have failed its condition code check.

CCKNOWNPASS	Meaning
`0b0`	The instruction was unconditional, or was conditional and passed its condition code check.
`0b1`	The instruction was conditional, and might have failed its condition code check.

Note

In an implementation in which an SMC instruction that fails it code check is not trapped, this field can always return the value 0.

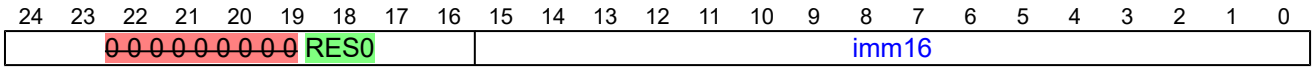
This field resets to an architecturally UNKNOWN value.

Bits [18:0]

Reserved, RES0.

'Traps to EL2 of Non-secure EL1 execution of SMC instructions' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section D1 (The AArch64 System Level Programmers' Model), describes the configuration settings for trapping SMC instructions from EL1 modes, and 'System calls' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section D1.16, describes the case where these exceptions are trapped to EL3.

ISS encoding for an exception from SMC instruction execution in AArch64 state.



Bits [24:16]

Reserved, RES0.

imm16, bits [15:0]

The value of the immediate field from the issued SMC instruction.

This field resets to an architecturally UNKNOWN value.

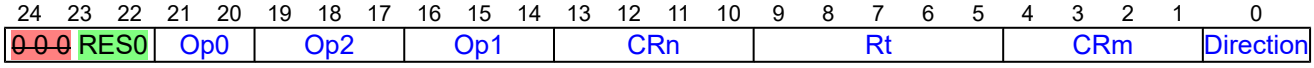
The value of ISS[24:0] described here is used both:

- When an SMC instruction is trapped from EL1 modes.

- When an SMC instruction is not trapped, so completes normally and generates an exception that is taken to EL3.

'Traps to EL2 of Non-secure EL1 execution of SMC instructions' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section D1 (The AArch64 System Level Programmers' Model), describes the configuration settings for trapping SMC instructions from Non-secure EL1 modes, and 'System calls' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section D1.16, describes the case where these exceptions are trapped to EL3.

ISS encoding for an exception from MSR, MRS, or System instruction execution in AArch64 state.



Bits [24:22]

Reserved, RES0.

Op0, bits [21:20]

The Op0 value from the issued instruction.

This field resets to an architecturally UNKNOWN value.

Op2, bits [19:17]

The Op2 value from the issued instruction.

This field resets to an architecturally UNKNOWN value.

Op1, bits [16:14]

The Op1 value from the issued instruction.

This field resets to an architecturally UNKNOWN value.

CRn, bits [13:10]

The CRn value from the issued instruction.

This field resets to an architecturally UNKNOWN value.

Rt, bits [9:5]

The Rt value from the issued instruction, the general-purpose register used for the transfer.

This field resets to an architecturally UNKNOWN value.

CRm, bits [4:1]

The CRm value from the issued instruction.

This field resets to an architecturally UNKNOWN value.

Direction, bit [0]

Indicates the direction of the trapped instruction. The possible values of this bit are:

Direction	Meaning
0b0	Write access, including MSR instructions.
0b1	Read access, including MRS instructions.

This field resets to an architecturally UNKNOWN value.

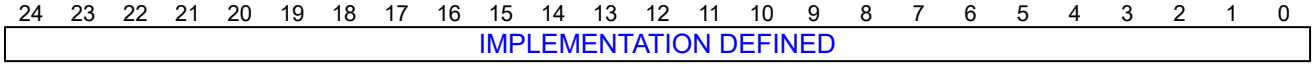
For exceptions caused by System instructions, see the 'System' subsection of 'Branches, exception generating and System instructions' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section C3 (A64 Instruction Set Encoding), for the encoding values returned by an instruction.

The following sections describe configuration settings for generating the exception that is reported using EC value `0b011000`:

- In 'EL1 configurable controls' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section D1.
  - 'Traps to EL1 of EL0 execution of cache maintenance instructions' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section D1.
  - 'Traps to EL1 of EL0 accesses to the CTR_EL0' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section D1.
  - 'Traps to EL1 of EL0 execution of DC ZVA instructions' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section D1.
  - 'Traps to EL1 of EL0 accesses to the PSTATE.{D, A, I, F} interrupt masks' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section D1(The AArch64 System Level Programmers' Model).
  - 'Traps to EL1 of EL0 and EL1 System register accesses to the trace registers' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section D1.
  - 'Traps to EL1 of EL0 accesses to the Debug Communications Channel (DCC) registers' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section D1.
  - 'Traps to EL1 of EL0 accesses to the Generic Timer registers' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section D1 (The AArch64 System Level Programmers' Model).
  - 'Traps to EL1 of EL0 accesses to Performance Monitors registers' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section D1.
  - 'Traps to EL1 of EL0 accesses to Activity Monitors registers' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section D1 (The AArch64 System Level Programmers' Model).
- In 'EL2 configurable controls' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section D1.
  - 'Traps to EL2 of Non-secure EL1 accesses to virtual memory control registers' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section D1.
  - 'Traps to EL2 of Non-secure EL0 and EL1 execution of DC ZVA instructions' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section D1.
  - 'Traps to EL2 of Non-secure EL1 execution of TLB maintenance instructions' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section D1.
  - 'Traps to EL2 of Non-secure EL0 and EL1 execution of cache maintenance instructions' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section D1.
  - 'Traps to EL2 of Non-secure EL1 accesses to the Auxiliary Control Register' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section D1.
  - 'Traps to EL2 of Non-secure EL0 and EL1 accesses to lockdown, DMA, and TCM operations' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section D1.
  - 'Traps to EL2 of Non-secure EL0 and EL1 accesses to the ID registers' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section D1.
  - 'Trapping to EL2 of Non-secure EL1 accesses to the CPACR_EL1 or CPACR' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section D1.
  - 'Traps to EL2 of Non-secure system register accesses to the trace registers' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section D1.
  - 'Trapping System register accesses to Debug ROM registers to EL2' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section D1.
  - 'Trapping System register accesses to OS-related debug registers to EL2' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section D1.
  - 'Traps to EL2 of Non-secure EL0 and EL1 accesses to the Generic Timer registers' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section D1.
  - 'Trapping general System register accesses to debug registers to EL2' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section D1.
  - 'Traps to EL2 of Non-secure EL0 and EL1 accesses to Performance Monitors registers' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section D1.
  - 'Traps to EL2 of EL1 and EL0 accesses to Activity Monitors registers' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section D1 (The AArch64 System Level Programmers' Model).
  - 'Trap to EL2 Non-secure EL1 accesses to Pointer authentication key registers' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section D1.
  - 'Traps to EL2 for Nested virtualization' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section D1.
  - 'Trap to EL2 Non-secure EL1 accesses to AT S1E* instructions' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section D1.
  - 'Trap to EL3 accesses to Pointer authentication key registers' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section D1.
- In 'EL3 configurable controls' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section D1.
  - 'Traps to EL3 of Secure EL1 accesses to the Counter-timer Physical Secure timer registers' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section D1.

- 'Trapping to EL3 of EL2 accesses to the CPTR_EL2 or HCPTR, and EL2 and EL1 accesses to the CPACR_EL1 or CPACR' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section D1.
- 'Traps to EL3 of all System register accesses to the trace registers' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section D1.
- 'Trapping System register accesses to OS-related debug registers to EL3' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section D1.
- 'Trapping general System register accesses to debug registers to EL3' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section D1.
- 'Traps to EL3 of EL2, EL1, and EL0 accesses to Performance Monitors registers' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section D1.
- 'Traps to EL3 of EL2, EL1, and EL0 accesses to Activity Monitors registers' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section D1 (The AArch64 System Level Programmers' Model).

ISS encoding for a IMPLEMENTATION DEFINED exception to EL3.

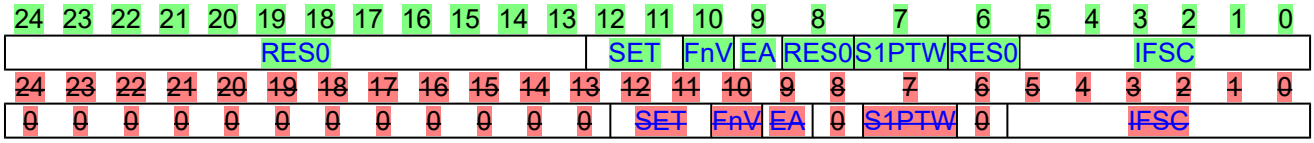


IMPLEMENTATION DEFINED, bits [24:0]

IMPLEMENTATION DEFINED.

This field resets to an architecturally UNKNOWN value.

ISS encoding for an exception from an Instruction Abort.



Bits [24:13]

Reserved, RES0.

SET, bits [12:11]

Synchronous Error Type. When the RAS Extension is implemented and IFSC is `0b010000`, describes the state of the PE after taking the Instruction Abort exception. The possible values of this field are:

SET	Meaning
`0b00`	Recoverable error (UER).
`0b10`	Uncontainable error (UC).
`0b11`	Restartable error (UEO) or Corrected error (CE).

All other values are reserved.

Note

Software can use this information to determine what recovery might be possible. Taking a synchronous External Abort exception might result in an unrecoverable PE state.

This field is RES0 if either:

- The RAS Extension is not implemented.
- The value returned in the IFSC field is not `0b010000`.

This field resets to an architecturally UNKNOWN value.

FnV, bit [10]

FAR not Valid, for a synchronous External abort other than a synchronous External abort on a translation table walk.

FnV	Meaning
0b0	FAR is valid.
0b1	FAR is not valid, and holds an UNKNOWN value.

This field is only valid if the IFSC code is 0b010000. It is RES0 for all other aborts.

This field resets to an architecturally UNKNOWN value.

EA, bit [9]

External abort type. This bit can provide an IMPLEMENTATION DEFINED classification of External aborts.

For any abort other than an External abort this bit returns a value of 0.

This field resets to an architecturally UNKNOWN value.

Bit [8]

Reserved, RES0.

S1PTW, bit [7]

For a stage 2 fault, indicates whether the fault was a stage 2 fault on an access made for a stage 1 translation table walk:

S1PTW	Meaning
0b0	Fault not on a stage 2 translation for a stage 1 translation table walk.
0b1	Fault on the stage 2 translation of an access for a stage 1 translation table walk.

For any abort other than a stage 2 fault this bit is RES0.

This field resets to an architecturally UNKNOWN value.

Bit [6]

Reserved, RES0.

IFSC, bits [5:0]

Instruction Fault Status Code. Possible values of this field are:

IFSC	Meaning
0b000000	Address size fault, level 0 of translation or translation table base register
0b000001	Address size fault, level 1
0b000010	Address size fault, level 2
0b000011	Address size fault, level 3
0b000100	Translation fault, level 0
0b000101	Translation fault, level 1
0b000110	Translation fault, level 2
0b000111	Translation fault, level 3
0b001001	Access flag fault, level 1
0b001010	Access flag fault, level 2
0b001011	Access flag fault, level 3
0b001101	Permission fault, level 1
0b001110	Permission fault, level 2
0b001111	Permission fault, level 3
0b010000	Synchronous External abort, not on translation table walk
0b010100	Synchronous External abort, on translation table walk, level 0
0b010101	Synchronous External abort, on translation table walk, level 1
0b010110	Synchronous External abort, on translation table walk, level 2
0b010111	Synchronous External abort, on translation table walk, level 3
0b011000	Synchronous parity or ECC error on memory access, not on translation table walk
0b011100	Synchronous parity or ECC error on memory access on translation table walk, level 0
0b011101	Synchronous parity or ECC error on memory access on translation table walk, level 1
0b011110	Synchronous parity or ECC error on memory access on translation table walk, level 2
0b011111	Synchronous parity or ECC error on memory access on translation table walk, level 3
0b110000	TLB conflict abort
0b110001	Unsupported atomic hardware update fault, if the implementation includes ARMv8.1-TTHM]. Otherwise reserved.

All other values are reserved.

When the RAS Extension is implemented, 0b011000, 0b011100, 0b011101, 0b011110, and 0b011111, are reserved.

Note

Armv8.2 requires the implementation of the RAS Extension.

For more information about the lookup level associated with a fault, see 'The level associated with MMU faults' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

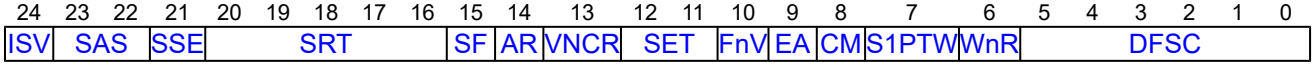
Note

Because Access flag faults and Permission faults can only result from a Block or Page translation table descriptor, they cannot occur at level 0.

If the S1PTW bit is set, then the level refers the level of the stage2 translation that is translating a stage 1 translation walk.

This field resets to an architecturally UNKNOWN value.

ISS encoding for an exception from a Data Abort.



ISV, bit [24]

Instruction syndrome valid. Indicates whether the syndrome information in ISS[23:0] is valid.

ISV	Meaning
0b0	No valid instruction syndrome. ISS[23:14] are RES0.
0b1	ISS[23:14] hold a valid instruction syndrome.

This bit is 0 for all faults reported in ESR_EL2 except the following stage 2 aborts:

- AArch64 loads and stores of a single general-purpose register (including the register specified with 0b11111, including those with Acquire/Release semantics, but excluding Load Exclusive or Store Exclusive and excluding those with writeback.
- AArch32 instructions where the instruction:
    - Is an LDR, LDA, LDRT, LDRSH, LDRSHT, LDRH, LDAH, LDRHT, LDRSB, LDRSBT, LDRB, LDAB, LDRBT, STR, STL, STRT, STRH, STLH, STRHT, STRB, STLB, or STRBT instruction.
    - Is not performing register writeback.
    - Is not using R15 as a source or destination register.

For these cases, ISV is UNKNOWN if the exception was generated in Debug state in memory access mode, and otherwise indicates whether ISS[23:14] hold a valid syndrome.

ISV is 0 for all faults reported in ESR_EL1 or ESR_EL3.

When the RAS Extension is implemented, ISV is 0 for any synchronous External abort.

For ISS reporting, a stage 2 abort on a stage 1 translation table walk does not return a valid instruction syndrome, and therefore ISV is 0 for these aborts.

When the RAS Extension is not implemented, the value of ISV on a synchronous External abort on a stage 2 translation table walk is IMPLEMENTATION DEFINED.

This field resets to an architecturally UNKNOWN value.

SAS, bits [23:22]

Syndrome Access Size. When ISV is 1, indicates the size of the access attempted by the faulting operation.

SAS	Meaning
0b00	Byte
0b01	Halfword
0b10	Word
0b11	Doubleword

This field is UNKNOWN when the value of ISV is UNKNOWN.

This field is RES0 when the value of ISV is 0.

This field resets to an architecturally UNKNOWN value.

SSE, bit [21]

Syndrome Sign Extend. When ISV is 1, for a byte, halfword, or word load operation, indicates whether the data item must be sign extended. For these cases, the possible values of this bit are:

SSE	Meaning
0b0	Sign-extension not required.
0b1	Data item must be sign-extended.

For all other operations this bit is 0.

This field is UNKNOWN when the value of ISV is UNKNOWN.

This field is RES0 when the value of ISV is 0.

This field resets to an architecturally UNKNOWN value.

SRT, bits [20:16]

Syndrome Register transfer. When ISV is 1, the register number of the Rt operand of the faulting instruction. If the exception was taken from an Exception level that is using AArch32 then this is the AArch64 view of the register. See 'Mapping of the general-purpose registers between the Execution states' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section D1.20.1.

This field is UNKNOWN when the value of ISV is UNKNOWN.

This field is RES0 when the value of ISV is 0.

This field resets to an architecturally UNKNOWN value.

SF, bit [15]

Width of the register accessed by the instruction is Sixty-Four. When ISV is 1, the possible values of this bit are:

SF	Meaning
0b0	Instruction loads/stores a 32-bit wide register.
0b1	Instruction loads/stores a 64-bit wide register.

Note

This field specifies the register width identified by the instruction, not the Execution state.

This field is UNKNOWN when the value of ISV is UNKNOWN.

This field is RES0 when the value of ISV is 0.

This field resets to an architecturally UNKNOWN value.

AR, bit [14]

Acquire/Release. When ISV is 1, the possible values of this bit are:

AR	Meaning
0b0	Instruction did not have acquire/release semantics.
0b1	Instruction did have acquire/release semantics.

This field is UNKNOWN when the value of ISV is UNKNOWN.

This field is RES0 when the value of ISV is 0.

This field resets to an architecturally UNKNOWN value.

VNCR, bit [13]

When ARMv8.4-NV is implemented:

Indicates that the fault came from use of VNCR_EL2 register by EL1 code.

VNCR	Meaning
0b0	The fault was not generated by the use of VNCR_EL2, by an MRS or MSR instruction executed at EL1.
0b1	The fault was generated by the use of VNCR_EL2, by an MRS or MSR instruction executed at EL1.

This field is 0 in ESR_EL1.

This field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

SET, bits [12:11]

Synchronous Error Type. When the RAS Extension is implemented and DFSC is 0b010000, describes the state of the PE after taking the Data Abort exception. The possible values of this field are:

SET	Meaning
0b00	Recoverable error (UER).
0b10	Uncontainable error (UC).
0b11	Restartable error (UEO) or Corrected error (CE).

All other values are reserved.

Note

Software can use this information to determine what recovery might be possible. Taking a synchronous External Abort exception might result in an unrecoverable PE state.

This field is RES0 if either:

- The RAS Extension is not implemented.
- The value returned in the DFSC field is not 0b010000.

This field resets to an architecturally UNKNOWN value.

FnV, bit [10]

FAR not Valid, for a synchronous External abort other than a synchronous External abort on a translation table walk.

FnV	Meaning
0b0	FAR is valid.
0b1	FAR is not valid, and holds an UNKNOWN value.

This field is valid only if the DFSC code is 0b010000. It is RES0 for all other aborts.

This field resets to an architecturally UNKNOWN value.

EA, bit [9]

External abort type. This bit can provide an IMPLEMENTATION DEFINED classification of External aborts.

For any abort other than an External abort this bit returns a value of 0.

This field resets to an architecturally UNKNOWN value.

CM, bit [8]

Cache maintenance. Indicates whether the Data Abort came from a cache maintenance or address translation instruction:

CM	Meaning
0b0	The Data Abort was not generated by the execution of one of the System instructions identified in the description of value 1.
0b1	The Data Abort was generated by either the execution of a cache maintenance instruction or by a synchronous fault on the execution of an address translation instruction. The DC ZVA instruction is not classified as a cache maintenance instruction, and therefore its execution cannot cause this field to be set to 1.

This field resets to an architecturally UNKNOWN value.

S1PTW, bit [7]

For a stage 2 fault, indicates whether the fault was a stage 2 fault on an access made for a stage 1 translation table walk:

S1PTW	Meaning
0b0	Fault not on a stage 2 translation for a stage 1 translation table walk.
0b1	Fault on the stage 2 translation of an access for a stage 1 translation table walk.

For any abort other than a stage 2 fault this bit is RES0.

This field resets to an architecturally UNKNOWN value.

WnR, bit [6]

Write not Read. Indicates whether a synchronous abort was caused by an instruction writing to a memory location, or by an instruction reading from a memory location. The possible values of this bit are:

WnR	Meaning
0b0	Abort caused by an instruction reading from a memory location.
0b1	Abort caused by an instruction writing to a memory location.

For faults on cache maintenance and address translation instructions, this bit always returns a value of 1.

For faults from an atomic instruction that both reads and writes from a memory location, this bit is set to 0 if a read of the address specified by the instruction would have generated the fault which is being reported, otherwise it is set to 1. The architecture permits, but does not require, a relaxation of this requirement such that for all stage 2 aborts on stage 1 translation table walks for atomic instructions, the WnR bit is always 0.

This field is UNKNOWN for:

- An External abort on an Atomic access.
- A fault reported using a DFSC value of 0b110101 or 0b110001, indicating an unsupported Exclusive or atomic access.

This field resets to an architecturally UNKNOWN value.

DFSC, bits [5:0]

Data Fault Status Code. Possible values of this field are:

DFSC	Meaning
0b000000	Address size fault, level 0 of translation or translation table base register.
0b000001	Address size fault, level 1.
0b000010	Address size fault, level 2.
0b000011	Address size fault, level 3.
0b000100	Translation fault, level 0.
0b000101	Translation fault, level 1.
0b000110	Translation fault, level 2.
0b000111	Translation fault, level 3.
0b001001	Access flag fault, level 1.
0b001010	Access flag fault, level 2.
0b001011	Access flag fault, level 3.
0b001101	Permission fault, level 1.
0b001110	Permission fault, level 2.
0b001111	Permission fault, level 3.
0b010000	Synchronous External abort, not on translation table walk.
0b010001	Synchronous Tag Check fail
0b010100	Synchronous External abort, on translation table walk, level 0.
0b010101	Synchronous External abort, on translation table walk, level 1.
0b010110	Synchronous External abort, on translation table walk, level 2.
0b010111	Synchronous External abort, on translation table walk, level 3.
0b011000	Synchronous parity or ECC error on memory access, not on translation table walk.
0b011100	Synchronous parity or ECC error on memory access on translation table walk, level 0.
0b011101	Synchronous parity or ECC error on memory access on translation table walk, level 1.
0b011110	Synchronous parity or ECC error on memory access on translation table walk, level 2.
0b011111	Synchronous parity or ECC error on memory access on translation table walk, level 3.
0b100001	Alignment fault.
0b110000	TLB conflict abort.
0b110001	Unsupported atomic hardware update fault, if the implementation includes ARMv8.1-TTHM]. Otherwise reserved.
0b110100	IMPLEMENTATION DEFINED fault (Lockdown).
0b110101	IMPLEMENTATION DEFINED fault (Unsupported Exclusive or Atomic access).
0b111101	Section Domain Fault, used only for faults reported in the PAR_EL1.
0b111110	Page Domain Fault, used only for faults reported in the PAR_EL1.

All other values are reserved.

When the RAS Extension is implemented, 0b011000, 0b011100, 0b011101, 0b011110, and 0b011111, are reserved.

For more information about the lookup level associated with a fault, see 'The level associated with MMU faults' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

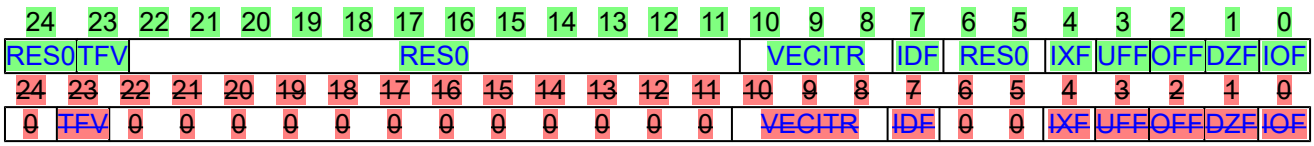
Note

Because Access flag faults and Permission faults can only result from a Block or Page translation table descriptor, they cannot occur at level 0.

If the S1PTW bit is set, then the level refers the level of the stage2 translation that is translating a stage 1 translation walk.

This field resets to an architecturally UNKNOWN value.

ISS encoding for an exception from a trapped floating-point exception.



Bit [24]

Reserved, RES0.

TFV, bit [23]

Trapped Fault Valid bit. Indicates whether the IDF, IXF, UFF, OFF, DZF, and IOF bits hold valid information about trapped floating-point exceptions. The possible values of this bit are:

TFV	Meaning
0b0	The IDF, IXF, UFF, OFF, DZF, and IOF bits do not hold valid information about trapped floating-point exceptions and are UNKNOWN.
0b1	One or more floating-point exceptions occurred during an operation performed while executing the reported instruction. The IDF, IXF, UFF, OFF, DZF, and IOF bits indicate trapped floating-point exceptions that occurred. For more information see 'Floating-point exception traps' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section D1.13.4.

It is IMPLEMENTATION DEFINED whether this field is set to 0 on an exception generated by a trapped floating point exception from a vector instruction.

Note

This is not a requirement. Implementations can set this field to 1 on a trapped floating-point exception from a vector instruction and return valid information in the {IDF, IXF, UFF, OFF, DZF, IOF} fields.

This field resets to an architecturally UNKNOWN value.

Bits [22:11]

Reserved, RES0.

VECITR, bits [10:8]

For a trapped floating-point exception from an instruction executed in AArch32 state this field is RES1.

For a trapped floating-point exception from an instruction executed in AArch64 state this field is UNKNOWN.

This field resets to an architecturally UNKNOWN value.

IDF, bit [7]

Input Denormal floating-point exception trapped bit. If the TFV field is 0, this bit is UNKNOWN. Otherwise, the possible values of this bit are:

IDF	Meaning
0b0	Input denormal floating-point exception has not occurred.
0b1	Input denormal floating-point exception occurred during execution of the reported instruction.

This field resets to an architecturally UNKNOWN value.

Bits [6:5]

Reserved, RES0.

IXF, bit [4]

Inexact floating-point exception trapped bit. If the TFV field is 0, this bit is UNKNOWN. Otherwise, the possible values of this bit are:

IXF	Meaning
0b0	Inexact floating-point exception has not occurred.
0b1	Inexact floating-point exception occurred during execution of the reported instruction.

This field resets to an architecturally UNKNOWN value.

UFF, bit [3]

Underflow floating-point exception trapped bit. If the TFV field is 0, this bit is UNKNOWN. Otherwise, the possible values of this bit are:

UFF	Meaning
0b0	Underflow floating-point exception has not occurred.
0b1	Underflow floating-point exception occurred during execution of the reported instruction.

This field resets to an architecturally UNKNOWN value.

OFF, bit [2]

Overflow floating-point exception trapped bit. If the TFV field is 0, this bit is UNKNOWN. Otherwise, the possible values of this bit are:

OFF	Meaning
0b0	Overflow floating-point exception has not occurred.
0b1	Overflow floating-point exception occurred during execution of the reported instruction.

This field resets to an architecturally UNKNOWN value.

DZF, bit [1]

Divide by Zero floating-point exception trapped bit. If the TFV field is 0, this bit is UNKNOWN. Otherwise, the possible values of this bit are:

DZF	Meaning
0b0	Divide by Zero floating-point exception has not occurred.
0b1	Divide by Zero floating-point exception occurred during execution of the reported instruction.

This field resets to an architecturally UNKNOWN value.

IOF, bit [0]

Invalid Operation floating-point exception trapped bit. If the TFV field is 0, this bit is UNKNOWN. Otherwise, the possible values of this bit are:

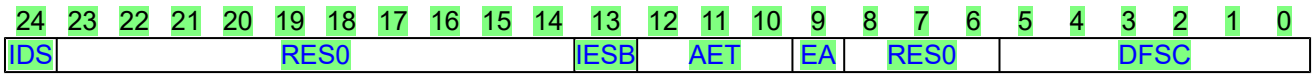
IOF	Meaning
0b0	Invalid Operation floating-point exception has not occurred.
0b1	Invalid Operation floating-point exception occurred during execution of the reported instruction.

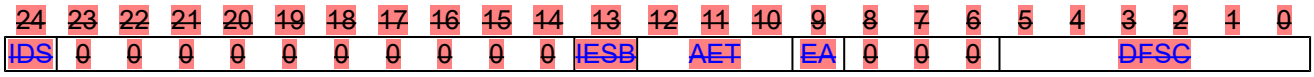
This field resets to an architecturally UNKNOWN value.

In an implementation that supports the trapping of floating-point exceptions:

- From an Exception level using AArch64, the FPCR.{IDE, IXE, UFE, OFE, DZE, IOE} bits enable each of the floating-point exception traps.
- From an Exception level using AArch32, the FPSCR.{IDE, IXE, UFE, OFE, DZE, IOE} bits enable each of the floating-point exception traps.

ISS encoding for a SError interrupt.





IDS, bit [24]

IMPLEMENTATION DEFINED syndrome. Possible values of this bit are:

IDS	Meaning
0b0	Bits[23:0] of the ISS field holds the fields described in this encoding. **Note** If the RAS Extension is not implemented, this means that bits[23:0] of the ISS field are RES0.
0b1	Bits[23:0] of the ISS field holds IMPLEMENTATION DEFINED syndrome information that can be used to provide additional information about the SError interrupt.

Note

This field was previously called ISV.

This field resets to an architecturally UNKNOWN value.

Bits [23:14]

Reserved, RES0.

IESB, bit [13]

When ARMv8.2-IESB is implemented:

Implicit error synchronization event.

IESB	Meaning
0b0	The SError interrupt was either not synchronized by the implicit error synchronization event or not taken immediately.
0b1	The SError interrupt was synchronized by the implicit error synchronization event and taken immediately.

This field is RES0 if the value returned in the DFSC field is not 0b010001.

Note

Armv8.2 requires the implementation of the RAS Extension and ARMv8.2-IESB.

This field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

AET, bits [12:10]

Asynchronous Error Type.

When the RAS Extension is implemented and DFSC is 0b010001, describes the state of the PE after taking the SError interrupt exception. The possible values of this field are:

AET	Meaning
0b000	Uncontainable error (UC).
0b001	Unrecoverable error (UEU).
0b010	Restartable error (UEO).
0b011	Recoverable error (UER).
0b110	Corrected error (CE).

All other values are reserved.

If multiple errors are taken as a single SError interrupt exception, the overall state of the PE is reported. For example, if both a Recoverable and Unrecoverable error occurred, the state is Unrecoverable.

Note

Software can use this information to determine what recovery might be possible. The recovery software must also examine any implemented fault records to determine the location and extent of the error.

This field is RES0 if either:

- The RAS Extension is not implemented.
- The value returned in the DFSC field is not 0b010001.

Note

Armv8.2 requires the implementation of the RAS Extension.

This field resets to an architecturally UNKNOWN value.

EA, bit [9]

External abort type. When the RAS Extension is implemented, this bit can provide an IMPLEMENTATION DEFINED classification of External aborts.

For any abort other than an External abort this bit returns a value of 0.

This field is RES0 if either:

- The RAS Extension is not implemented.
- The value returned in the DFSC field is not 0b010001.

Note

Armv8.2 requires the implementation of the RAS Extension.

This field resets to an architecturally UNKNOWN value.

Bits [8:6]

Reserved, RES0.

DFSC, bits [5:0]

Data Fault Status Code. When the RAS Extension is implemented, possible values of this field are:

DFSC	Meaning
0b000000	Uncategorized.
0b010001	Asynchronous SError interrupt.

All other values are reserved.

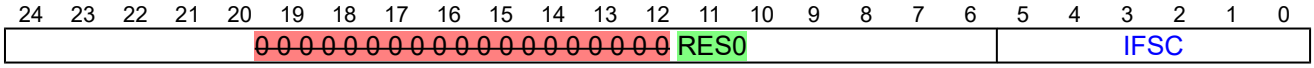
If the RAS Extension is not implemented, this field is RES0.

Note

Armv8.2 requires the implementation of the RAS Extension.

This field resets to an architecturally UNKNOWN value.

ISS encoding for an exception from a Breakpoint or Vector Catch debug exception.



Bits [24:6]

Reserved, RES0.

IFSC, bits [5:0]

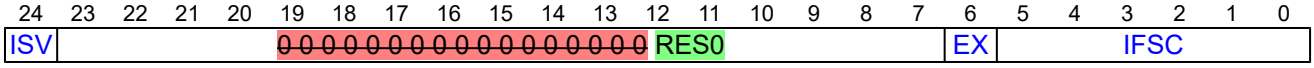
Instruction Fault Status Code. This field is set to `0b100010`, to indicate a Debug exception.

This field resets to an architecturally UNKNOWN value.

For more information about generating these exceptions:

- For exceptions from AArch64, see 'Breakpoint exceptions' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section D2 (AArch64 Self-hosted Debug).
- For exceptions from AArch32, see 'Breakpoint exceptions' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section G2 (AArch32 Self-hosted Debug) and 'Vector Catch exceptions' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section G2.

ISS encoding for an exception from a Software Step exception.



ISV, bit [24]

Instruction syndrome valid. Indicates whether the EX bit, ISS[6], is valid, as follows:

ISV	Meaning
`0b0`	EX bit is RES0.
`0b1`	EX bit is valid.

See the EX bit description for more information.

This field resets to an architecturally UNKNOWN value.

Bits [23:7]

Reserved, RES0.

EX, bit [6]

Exclusive operation. If the ISV bit is set to 1, this bit indicates whether a Load-Exclusive instruction was stepped.

EX	Meaning
`0b0`	An instruction other than a Load-Exclusive instruction was stepped.
`0b1`	A Load-Exclusive instruction was stepped.

If the ISV bit is set to 0, this bit is RES0, indicating no syndrome data is available.

This field resets to an architecturally UNKNOWN value.

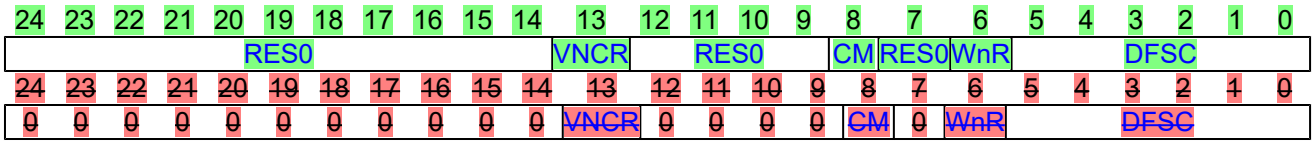
IFSC, bits [5:0]

Instruction Fault Status Code. This field is set to 0b100010, to indicate a Debug exception.

This field resets to an architecturally UNKNOWN value.

For more information about generating these exceptions, see 'Software Step exceptions' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section D2 (AArch64 Self-hosted Debug).

ISS encoding for an exception from a Watchpoint exception.



Bits [24:14]

Reserved, RES0.

VNCR, bit [13]

When ARMv8.4-NV is implemented:

Indicates that the watchpoint came from use of VNCR_EL2 register by EL1 code.

VNCR	Meaning
0b0	The watchpoint was not generated by the use of VNCR_EL2 by EL1 code.
0b1	The watchpoint was generated by the use of VNCR_EL2 by EL1 code.

This field is 0 in ESR_EL1.

This field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [12:9]

Reserved, RES0.

CM, bit [8]

Cache maintenance. Indicates whether the Watchpoint exception came from a cache maintenance or address translation instruction:

CM	Meaning
0b0	The Watchpoint exception was not generated by the execution of one of the System instructions identified in the description of value 1.
0b1	The Watchpoint exception was generated by either the execution of a cache maintenance instruction or by a synchronous Watchpoint exception on the execution of an address translation instruction. The DC ZVA instruction is not classified as a cache maintenance instruction, and therefore its execution cannot cause this field to be set to 1.

This field resets to an architecturally UNKNOWN value.

Bit [7]

Reserved, RES0.

WnR, bit [6]

Write not Read. Indicates whether the Watchpoint exception was caused by an instruction writing to a memory location, or by an instruction reading from a memory location. The possible values of this bit are:

WnR	Meaning
0b0	Watchpoint exception caused by an instruction reading from a memory location.
0b1	Watchpoint exception caused by an instruction writing to a memory location.

For Watchpoint exceptions on cache maintenance and address translation instructions, this bit always returns a value of 1.

For Watchpoint exceptions from an atomic instruction, this field is set to 0 if a read of the location would have generated the Watchpoint exception, otherwise it is set to 1.

If multiple watchpoints match on the same access, it is UNPREDICTABLE which watchpoint generates the Watchpoint exception.

This field resets to an architecturally UNKNOWN value.

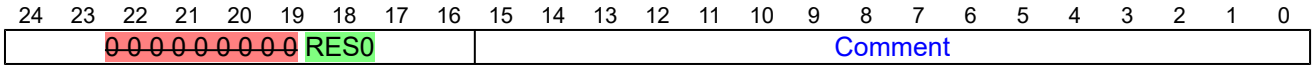
DFSC, bits [5:0]

Data Fault Status Code. This field is set to 0b100010, to indicate a Debug exception.

This field resets to an architecturally UNKNOWN value.

For more information about generating these exceptions, see 'Watchpoint exceptions' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section D2 (AArch64 Self-hosted Debug).

ISS encoding for an exception from execution of a Breakpoint instruction.



Bits [24:16]

Reserved, RES0.

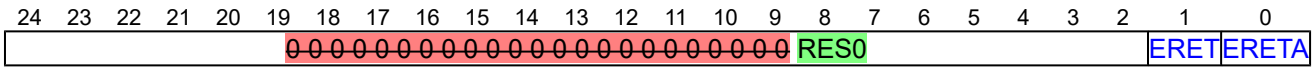
Comment, bits [15:0]

Set to the instruction comment field value, zero extended as necessary. For the AArch32 BKPT instructions, the comment field is described as the immediate field.

This field resets to an architecturally UNKNOWN value.

For more information about generating these exceptions, see 'Breakpoint instruction exceptions' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section D2 (AArch64 Self-hosted Debug).

ISS encoding for an exception from ERET, ERETAA or ERETAB instruction.



This EC value only applies when HCR_EL2.NV is 1.

Bits [24:2]

Reserved, RES0.

ERET, bit [1]

Indicates whether an ERET or ERETA* instruction was trapped to EL2. Possible values are:

ERET	Meaning
0b0	ERET instruction trapped to EL2.
0b1	ERETAA or ERETAB instruction trapped to EL2.

If this bit is 0, the ERETA field is RES0.

This field resets to an architecturally UNKNOWN value.

ERETA, bit [0]

Indicates whether an ERETAA or ERETAB instruction was trapped to EL2. Possible values are:

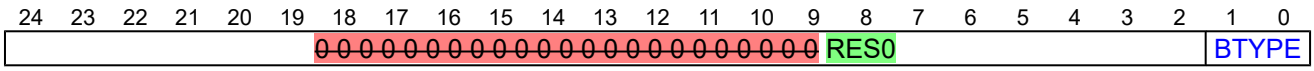
ERETA	Meaning
0b0	ERETAA instruction trapped to EL2.
0b1	ERETAB instruction trapped to EL2.

When the ERET field is 0, this bit is RES0.

This field resets to an architecturally UNKNOWN value.

For more information about generating these exceptions, see 'Traps to EL2 for Nested virtualization' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section D1.

ISS encoding for an exception from Branch Target Identification instruction.



Bits [24:2]

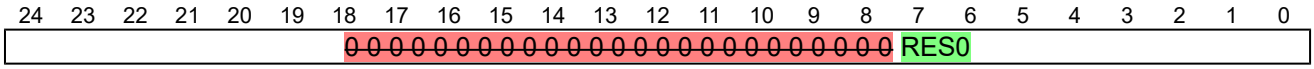
Reserved, RES0.

BTYPE, bits [1:0]

This field is set to the PSTATE.BTYPE value that generated the Branch Target Exception.

For more information about generating these exceptions, see The AArch64 application level programmers' model' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section B1.

ISS encoding for an exception from a Pointer Authentication instruction when HCR_EL2.API == 0 || SCR_EL3.API == 0.



Bits [24:0]

Reserved, RES0.

For more information about generating these exceptions, see:

- 'Trap to EL2 Non-secure EL0 accesses to Pointer authentication key registers' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section D1.
- 'Trap to EL3 accesses to Pointer authentication instructions' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section D1.

Accessing the ESR_EL1

When HCR_EL2.E2H is 1, without explicit synchronization, access from EL3 using the mnemonic ESR_EL1 or ESR_EL12 are not guaranteed to be ordered with respect to accesses using the other mnemonic.

Accesses to this register use the following encodings:

MRS <Xt>, ESR_EL1

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b11	0b000	0b0101	0b0010	0b000
~~0b11~~	~~0b0101~~	~~0b000~~	~~0b000~~	~~0b0010~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TRVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<NV2,NV1,NV> == '111' then
        return NVMem[0x138];
    else
        return ESR_EL1;
elsif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' then
        return ESR_EL2;
    else
        return ESR_EL1;
elsif PSTATE.EL == EL3 then
    return ESR_EL1;
```

MSR ESR_EL1, <Xt>

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b11	0b000	0b0101	0b0010	0b000
~~0b11~~	~~0b0101~~	~~0b000~~	~~0b000~~	~~0b0010~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<NV2,NV1,NV> == '111' then
        NVMem[0x138] = X[t];
    else
        ESR_EL1 = X[t];
elsif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' then
        ESR_EL2 = X[t];
    else
        ESR_EL1 = X[t];
elsif PSTATE.EL == EL3 then
    ESR_EL1 = X[t];
```

MRS <Xt>, ESR_EL12

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b11	0b101	0b0101	0b0010	0b000
~~0b11~~	~~0b0101~~	~~0b101~~	~~0b000~~	~~0b0010~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '101' then
        return NVMem[0x138];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if EL2Enabled() && HCR_EL2.E2H == '1' then
    if PSTATE.EL == EL2 then
        return ESR_EL1;
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if EL2Enabled() && HCR_EL2.E2H == '1' then
        return ESR_EL1;
else
    else
        UNDEFINED;
```

MSR ESR_EL12, <Xt>

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b11	0b101	0b0101	0b0010	0b000
~~0b11~~	~~0b0101~~	~~0b101~~	~~0b000~~	~~0b0010~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '101' then
        NVMem[0x138] = X[t];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if EL2Enabled() && HCR_EL2.E2H == '1' then
    if PSTATE.EL == EL2 then
        ESR_EL1 = X[t];
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if EL2Enabled() && HCR_EL2.E2H == '1' then
        ESR_EL1 = X[t];
else
    else
        UNDEFINED;
```

MRS <Xt>, ESR_EL2

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b11	0b100	0b0101	0b0010	0b000
~~0b11~~	~~0b0101~~	~~0b100~~	~~0b000~~	~~0b0010~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
        return ESR_EL1;
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    return ESR_EL2;
elsif PSTATE.EL == EL3 then
    return ESR_EL2;
```

```
MSR ESR_EL2, <Xt>
```

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b11	0b100	0b0101	0b0010	0b000
~~0b11~~	~~0b0101~~	~~0b100~~	~~0b000~~	~~0b0010~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
        ESR_EL1 = X[t];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    ESR_EL2 = X[t];
elsif PSTATE.EL == EL3 then
    ESR_EL2 = X[t];
```

27~~13~~/03~~12~~/2019~~2018~~ 21~~16~~:59~~42~~; e5e4db499bf9867a4b93324c4dbac985d3da93766~~379d01e197f1d40720d32d0f84e419e9187e009~~

(old) htmldiff from- (new)

FAR_EL1, Fault Address Register (EL1)

The FAR_EL1 characteristics are:

Purpose

Holds the faulting Virtual Address for all synchronous Instruction or Data Abort, PC alignment fault and Watchpoint exceptions that are taken to EL1.

Configuration

AArch64 System register FAR_EL1 bits [31:0] are architecturally mapped to AArch32 System register DFAR[31:0] (NS) .

AArch64 System register FAR_EL1 bits [63:32] are architecturally mapped to AArch32 System register IFAR[31:0] (NS) .

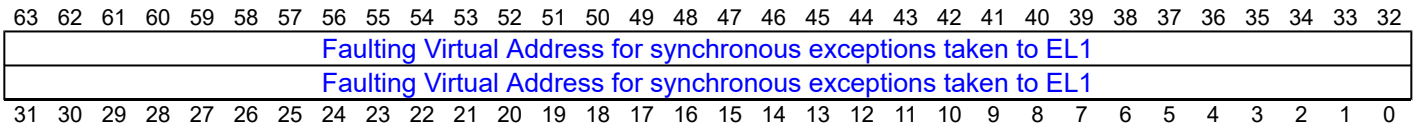
RW fields in this register reset to architecturally UNKNOWN values.

Attributes

FAR_EL1 is a 64-bit register.

Field descriptions

The FAR_EL1 bit assignments are:



Bits [63:0]

Faulting Virtual Address for synchronous exceptions taken to EL1. Exceptions that set the FAR_EL1 are Instruction Aborts (EC `0x20` or `0x21`), Data Aborts (EC `0x24` or `0x25`), PC alignment faults (EC `0x22`), and Watchpoints (EC `0x34` or `0x35`). ESR_EL1.EC holds the EC value for the exception.

For a synchronous External abort, if the VA that generated the abort was from an address range for which TCR_ELx.TBI{<0|1>} == 1 for the translation regime in use when the abort was generated, then the top eight bits of FAR_EL1 are UNKNOWN.

For a synchronous External abort other than a synchronous External abort on a translation table walk, this field is valid only if ESR_EL1.FnV is 0, and the FAR_EL1 is UNKNOWN if ESR_EL1.FnV is 1.

For all other exceptions taken to EL1, the FAR_EL1 is UNKNOWN.

If a memory fault that sets FAR_EL1 is generated from a data cache maintenance or other DC instruction, this field holds the address specified in the register argument of the instruction.

If the exception that updates FAR_EL1 is taken from an Exception level that is using AArch32, the top 32 bits are all zero, unless both of the following apply, in which case the top 32 bits of FAR_ELx are `0x00000001`:

- The faulting address was generated by a load or store instruction that sequentially incremented from address `0xFFFFFFFF`. Such a load or store is CONSTRAINED UNPREDICTABLE. See 'Out of range VA' in Appendix K1 Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.
- The implementation treats such incrementing as setting bit[32] of the virtual address to 1.

For a Data Abort or Watchpoint exception, if address tagging is enabled for the address accessed by the data access that caused the exception, then this field includes the tag. For more information about address tagging, see 'Address tagging in AArch64 state' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

Execution at EL0 makes FAR_EL1 become UNKNOWN.

Note

The address held in this field is an address accessed by the instruction fetch or data access that caused the exception that gave rise to the instruction or data abort. It is the lower address that gave rise to the fault. Where different faults from different addresses arise from the same instruction, such as for an instruction that loads or stores a mis-aligned address that crosses a page boundary, the architecture does not prioritize between those different faults.

FAR_EL1 is made UNKNOWN on an exception return from EL1.

This field resets to an architecturally UNKNOWN value.

Accessing the FAR_EL1

When HCR_EL2.E2H is 1, without explicit synchronization, access from EL3 using the mnemonic FAR_EL1 or FAR_EL12 are not guaranteed to be ordered with respect to accesses using the other mnemonic.

Accesses to this register use the following encodings:

```
MRS <Xt>, FAR_EL1
```

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b11	0b000	0b0110	0b0000	0b000
~~0b11~~	~~0b0110~~	~~0b000~~	~~0b000~~	~~0b0000~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TRVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<NV2,NV1,NV> == '111' then
        return NVMem[0x220];
    else
        return FAR_EL1;
elsif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' then
        return FAR_EL2;
    else
        return FAR_EL1;
elsif PSTATE.EL == EL3 then
    return FAR_EL1;
```

```
MSR FAR_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b11	0b000	0b0110	0b0000	0b000
~~0b11~~	~~0b0110~~	~~0b000~~	~~0b000~~	~~0b0000~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<NV2,NV1,NV> == '111' then
        NVMem[0x220] = X[t];
    else
        FAR_EL1 = X[t];
elsif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' then
        FAR_EL2 = X[t];
    else
        FAR_EL1 = X[t];
elsif PSTATE.EL == EL3 then
    FAR_EL1 = X[t];
```

MRS <Xt>, FAR_EL12

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b11	0b101	0b0110	0b0000	0b000
~~0b11~~	~~0b0110~~	~~0b101~~	~~0b000~~	~~0b0000~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '101' then
        return NVMem[0x220];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if EL2Enabled() && HCR_EL2.E2H == '1' then
    if PSTATE.EL == EL2 then
        return FAR_EL1;
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if EL2Enabled() && HCR_EL2.E2H == '1' then
        return FAR_EL1;
else
    else
        UNDEFINED;
```

MSR FAR_EL12, <Xt>

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b11	0b101	0b0110	0b0000	0b000
~~0b11~~	~~0b0110~~	~~0b101~~	~~0b000~~	~~0b0000~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '101' then
        NVMem[0x220] = X[t];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if EL2Enabled() && HCR_EL2.E2H == '1' then
    if PSTATE.EL == EL2 then
        FAR_EL1 = X[t];
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if EL2Enabled() && HCR_EL2.E2H == '1' then
        FAR_EL1 = X[t];
else
    else
        UNDEFINED;
```

MRS <Xt>, FAR_EL2

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b11	0b100	0b0110	0b0000	0b000
~~0b11~~	~~0b0110~~	~~0b100~~	~~0b000~~	~~0b0000~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
        return FAR_EL1;
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    return FAR_EL2;
elsif PSTATE.EL == EL3 then
    return FAR_EL2;
```

MSR FAR_EL2, <Xt>

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b11	0b100	0b0110	0b0000	0b000
~~0b11~~	~~0b0110~~	~~0b100~~	~~0b000~~	~~0b0000~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
        FAR_EL1 = X[t];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    FAR_EL2 = X[t];
elsif PSTATE.EL == EL3 then
    FAR_EL2 = X[t];
```

27~~13~~/03~~12~~/2019~~2018~~ 21~~16~~:59~~42~~; e5e4db499bf9867a4b93324c4dbac985d3da93766~~6379d01e197f1d40720d32d0f84e419e9187e009~~

FAR_EL2, Fault Address Register (EL2)

The FAR_EL2 characteristics are:

Purpose

Holds the faulting Virtual Address for all synchronous Instruction or Data Abort, PC alignment fault and Watchpoint exceptions that are taken to EL2.

Configuration

AArch64 System register FAR_EL2 bits [31:0] are architecturally mapped to AArch32 System register HDFAR[31:0] .

AArch64 System register FAR_EL2 bits [63:32] are architecturally mapped to AArch32 System register HIFAR[31:0] .

AArch64 System register FAR_EL2 bits [31:0] are architecturally mapped to AArch32 System register DFAR[31:0] (S) when HaveEL(EL2).

AArch64 System register FAR_EL2 bits [63:32] are architecturally mapped to AArch32 System register IFAR[31:0] (S) when HaveEL(EL2).

If EL2 is not implemented, this register is RES0 from EL3.

This register has no effect if EL2 is not enabled in the current Security state.

RW fields in this register reset to architecturally UNKNOWN values.

Attributes

FAR_EL2 is a 64-bit register.

Field descriptions

The FAR_EL2 bit assignments are:

63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
Faulting Virtual Address for synchronous exceptions taken to EL2																															
Faulting Virtual Address for synchronous exceptions taken to EL2																															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Bits [63:0]

Faulting Virtual Address for synchronous exceptions taken to EL2. Exceptions that set the FAR_EL2 are Instruction Aborts (EC `0x20` or `0x21`), Data Aborts (EC `0x24` or `0x25`), PC alignment faults (EC `0x22`), and Watchpoints (EC `0x34` or `0x35`). ESR_EL2.EC holds the EC value for the exception.

For a synchronous External abort, if the VA that generated the abort was from an address range for which TCR_ELx.TBI{<0|1>} == 1 for the translation regime in use when the abort was generated, then the top eight bits of FAR_EL2 are UNKNOWN.

For a synchronous External abort other than a synchronous External abort on a translation table walk, this field is valid only if ESR_EL2.FnV is 0, and the FAR_EL2 is UNKNOWN if ESR_EL2.FnV is 1.

For all other exceptions taken to EL2, the FAR_EL2 is UNKNOWN.

If a memory fault that sets FAR_EL2 is generated from a data cache maintenance or other DC instruction, this field holds the address specified in the register argument of the instruction.

If the exception that updates FAR_EL2 is taken from an Exception level that is using AArch32, the top 32 bits are all zero, unless both of the following apply, in which case the top 32 bits of FAR_ELx are `0x00000001`:

- The faulting address was generated by a load or store instruction that sequentially incremented from address `0xFFFFFFFF`. Such a load or store instruction is CONSTRAINED UNPREDICTABLE. See 'Out of range VA' in Appendix K1 Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.
- The implementation treats such incrementing as setting bit[32] of the virtual address to 1.

For a Data Abort or Watchpoint exception, if address tagging is enabled for the address accessed by the data access that caused the exception, then this field includes the tag. For more information about address tagging, see 'Address tagging in AArch64 state' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

Execution at EL1 or EL0 makes FAR_EL2 become UNKNOWN.

Note

The address held in this field is an address accessed by the instruction fetch or data access that caused the exception that gave rise to the instruction or data abort. It is the lower address that gave rise to the fault. Where different faults from different addresses arise from the same instruction, such as for an instruction that loads or stores a mis-aligned address that crosses a page boundary, the architecture does not prioritize between those different faults.

FAR_EL2 is made UNKNOWN on an exception return from EL2.

This field resets to an architecturally UNKNOWN value.

Accessing the FAR_EL2

When HCR_EL2.E2H is 1, without explicit synchronization, access from EL2 using the mnemonic FAR_EL2 or FAR_EL1 are not guaranteed to be ordered with respect to accesses using the other mnemonic.

Accesses to this register use the following encodings:

```
MRS <Xt>, FAR_EL2
```

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b11	0b100	0b0110	0b0000	0b000
~~0b11~~	~~0b0110~~	~~0b100~~	~~0b000~~	~~0b0000~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
        return FAR_EL1;
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    return FAR_EL2;
elsif PSTATE.EL == EL3 then
    return FAR_EL2;
```

```
MSR FAR_EL2, <Xt>
```

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b11	0b100	0b0110	0b0000	0b000
~~0b11~~	~~0b0110~~	~~0b100~~	~~0b000~~	~~0b0000~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
        FAR_EL1 = X[t];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    FAR_EL2 = X[t];
elsif PSTATE.EL == EL3 then
    FAR_EL2 = X[t];
```

MRS <Xt>, FAR_EL1

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b11	0b000	0b0110	0b0000	0b000
~~0b11~~	~~0b0110~~	~~0b000~~	~~0b000~~	~~0b0000~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TRVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<NV2,NV1,NV> == '111' then
        return NVMem[0x220];
    else
        return FAR_EL1;
elsif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' then
        return FAR_EL2;
    else
        return FAR_EL1;
elsif PSTATE.EL == EL3 then
    return FAR_EL1;
```

MSR FAR_EL1, <Xt>

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b11	0b000	0b0110	0b0000	0b000
~~0b11~~	~~0b0110~~	~~0b000~~	~~0b000~~	~~0b0000~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<NV2,NV1,NV> == '111' then
        NVMem[0x220] = X[t];
    else
        FAR_EL1 = X[t];
elsif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' then
        FAR_EL2 = X[t];
    else
        FAR_EL1 = X[t];
elsif PSTATE.EL == EL3 then
    FAR_EL1 = X[t];
```

2713/0312/20192018 2116:5942; e5e4db499bf9867a4b93324c4dbac985d3da93766379d01e197f1d40720d32d0f84e419e9187e009

(old) htmldiff from- (new)

(old) htmldiff from- (new)

FAR_EL3, Fault Address Register (EL3)

The FAR_EL3 characteristics are:

Purpose

Holds the faulting Virtual Address for all synchronous Instruction or Data Abort and PC alignment fault exceptions that are taken to EL3.

Configuration

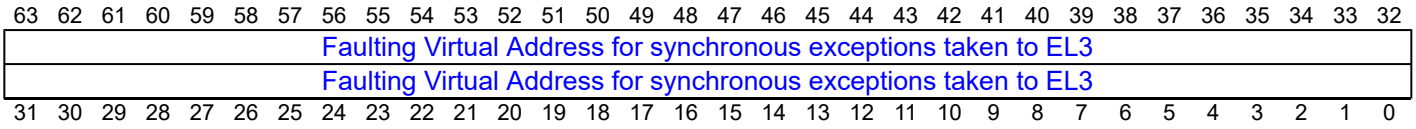
RW fields in this register reset to architecturally UNKNOWN values.

Attributes

FAR_EL3 is a 64-bit register.

Field descriptions

The FAR_EL3 bit assignments are:



Bits [63:0]

Faulting Virtual Address for synchronous exceptions taken to EL3. Exceptions that set the FAR_EL3 are Instruction Aborts (EC `0x20` or `0x21`), Data Aborts (EC `0x24` or `0x25`), and PC alignment faults (EC `0x22`). ESR_EL3.EC holds the EC value for the exception.

For a synchronous External abort, if the VA that generated the abort was from an address range for which TCR_ELx.TBI{<0|1>} == 1 for the translation regime in use when the abort was generated, then the top eight bits of FAR_EL3 are UNKNOWN.

For a synchronous External abort other than a synchronous External abort on a translation table walk, this field is valid only if ESR_EL3.FnV is 0, and the FAR_EL3 is UNKNOWN if ESR_EL3.FnV is 1.

For all other exceptions taken to EL3, the FAR_EL3 is UNKNOWN.

If a memory fault that sets FAR_EL3 is generated from a data cache maintenance or other DC instruction, this field holds the address specified in the register argument of the instruction.

If the exception that updates FAR_EL3 is taken from an Exception Level using AArch32, the top 32 bits are all zero, unless both of the following apply, in which case the top 32 bits of FAR_ELx are `0x00000001`:

- The faulting address was generated by a load or store instruction that sequentially incremented from address `0xFFFFFFFF`. Such a load or store instruction is CONSTRAINED UNPREDICTABLE. See 'Out of range VA' in Appendix K1 Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.
- The implementation treats such incrementing as setting bit[32] of the virtual address to 1.

For a Data Abort or Watchpoint exception, if address tagging is enabled for the address accessed by the data access that caused the exception, then this field includes the tag. For more information about address tagging, see 'Address tagging in AArch64 state' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

Execution at EL2, EL1 or EL0 makes FAR_EL3 become UNKNOWN.

Note

The address held in this register is an address accessed by the instruction fetch or data access that caused the exception that actually gave rise to the instruction or data abort. It is the lowest address that gave rise to the fault. Where different faults from different addresses arise from the same instruction, such as for an instruction that loads or stores a mis-aligned address that crosses a page boundary, the architecture does not prioritize between those different faults.

FAR_EL3 is made UNKNOWN on an exception return from EL3.

This field resets to an architecturally UNKNOWN value.

Accessing the FAR_EL3

Accesses to this register use the following encodings:

```
MRS <Xt>, FAR_EL3
```

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b11	0b110	0b0110	0b0000	0b000
~~0b11~~	~~0b0110~~	~~0b110~~	~~0b000~~	~~0b0000~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    return FAR_EL3;
```

```
MSR FAR_EL3, <Xt>
```

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b11	0b110	0b0110	0b0000	0b000
~~0b11~~	~~0b0110~~	~~0b110~~	~~0b000~~	~~0b0000~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    FAR_EL3 = X[t];
```

27~~13~~/03~~12~~/2019~~2018~~ 21~~16~~:59~~42~~; e5e4db499bf9867a4b93324c4dbac985d3da9376~~6379d01e197f1d40720d32d0f84e419e9187e009~~

(old) htmldiff from- (new)

FPEXC32_EL2, Floating-Point Exception Control register

The FPEXC32_EL2 characteristics are:

Purpose

Allows access to the AArch32 register FPEXC from AArch64 state only. Its value has no effect on execution in AArch64 state.

Configuration

AArch64 System register FPEXC32_EL2 bits [31:0] are architecturally mapped to AArch32 System register FPEXC[31:0] .

If EL1 cannot use AArch32, this register is UNDEFINED.

If EL2 is not implemented but EL3 is implemented, and EL1 is capable of using AArch32, then this register is not RES0.

Implemented only if the implementation includes the Advanced SIMD and floating-point functionality.

This register has no effect if EL2 is not enabled in the current Security state.

RW fields in this register reset to architecturally UNKNOWN values.

Attributes

FPEXC32_EL2 is a 64-bit register.

Field descriptions

The FPEXC32_EL2 bit assignments are:

Bits [63:32]

Reserved, RES0.

EX, bit [31]

Exception bit. In Armv8, this bit is RAZ/WI.

This field resets to an architecturally UNKNOWN value.

EN, bit [30]

Enables access to the Advanced SIMD and floating-point functionality from all Exception levels, except that setting this field to 0 does not disable the following:

- VMSR accesses to the FPEXC or FPSID.
- VMRS accesses from the FPEXC, FPSID, MVFR0, MVFR1, or MVFR2.

EN	Meaning
0b0	Accesses to the FPSCR, and any of the SIMD and floating-point registers Q0-Q15, including their views as D0-D31 registers or S0-S31 registers, are UNDEFINED at all Exception levels.
0b1	This control permits access to the Advanced SIMD and floating-point functionality at all Exception levels.

Execution of floating-point and Advanced SIMD instructions in AArch32 state can be disabled or trapped by the following controls:

- CPACR.cp10, or, if executing at EL0, CPACR_EL1.FPEN.
- FPEXC.EN.
- If executing in Non-secure state:
  - HCPTR.TCP10, or if EL2 is using AArch64, CPTR_EL2.TFP.
  - NSACR.cp10, or if EL3 is using AArch64, CPTR_EL3.TFP.
- For Advanced SIMD instructions only:
  - CPACR.ASEDIS.
  - If executing in Non-secure state, HCPTR.TASE and NSACR.NSTRCDIS.

See the descriptions of the controls for more information.

Note

When executing at EL0 using AArch32:

- If EL1 is using AArch64 then behavior is as if the value of FPEXC.EN is 1.
- If EL2 is using AArch64 and enabled in the current Security state, and the value of HCR_EL2.{RW, TGE} is {1, 1} then behavior is as if the value of FPEXC.EN is 1.
- If EL2 is using AArch64 and enabled in the current Security state, and the value of HCR_EL2.{RW, TGE} is {0, 1} then it is IMPLEMENTATION DEFINED whether the behavior is:
  - As if the value of FPEXC.EN is 1.
  - Determined by the value of FPEXC32_EL2.EN, as described in this field description. However, Arm deprecates using the value of FPEXC32_EL2.EN to determine behavior.

This field resets to an architecturally UNKNOWN value.

DEX, bit [29]

Defined synchronous exception on floating-point execution.

This field identifies whether a synchronous exception generated by the attempted execution of an instruction was generated by an unallocated encoding. The instruction must be in the encoding space that is identified by the pseudocode function ExecutingCP10or11Instr() returning TRUE. This field also indicates whether the FPEXC32_EL2.TFV field is valid.

The meaning of this bit is:

DEX	Meaning
0b0	The exception was generated by the attempted execution of an unallocated instruction in the encoding space that is identified by the pseudocode function ExecutingCP10or11Instr(). If FPEXC32_EL2.TFV is RW then it is invalid and UNKNOWN. If FPEXC32_EL2.{IDF, IXF, UFF, OFF, DZF, IOF} are RW then they are invalid and UNKNOWN.
0b1	The exception was generated during the execution of an unallocated encoding. FPEXC32_EL2.TFV is valid and indicates the cause of the exception.

On an exception that sets this bit to 1 the exception-handling routine must clear this bit to 0.

On an implementation that both does not support trapping of floating-point exceptions and implements the AArch32 FPSCR.{Stride, Len} fields as RAZ, this bit is RES0.

This field resets to an architecturally UNKNOWN value.

FP2V, bit [28]

FPINST2 instruction valid bit. In Armv8, this bit is RES0.

This field resets to an architecturally UNKNOWN value.

VV, bit [27]

VECITR valid bit. In Armv8, this bit is RES0.

This field resets to an architecturally UNKNOWN value.

TFV, bit [26]

Trapped Fault Valid bit. Valid only when the value of FPEXC.DEX is 1. When valid, it indicates the cause of the exception and therefore whether the FPEXC.{IDF, IXF, UFF, OFF, DZF, IOF} bits are valid.

TFV	Meaning
0b0	The exception was caused by the execution of a floating-point VABS, VADD, VDIV, VFMA, VFMS, VFNMA, VFNMS, VMLA, VMLS, VMOV, VMUL, VNEG, VNMLA, VNMLS, VNMUL, VSQRT, or VSUB instruction when one or both of FPSCR.{Stride, Len} was non-zero. If the FPEXC.{IDF, IXF, UFF, OFF, DZF, IOF} bits are RW then they are invalid and UNKNOWN.
0b1	FPEXC.{IDF, IXF, UFF, OFF, DZF, IOF} indicate the presence of trapped floating-point exceptions that had occurred at the time of the exception. Bits are set for all trapped exceptions that had occurred at the time of the exception.

This bit returns a status value and ignores writes.

When the value of FPEXC.DEX is 0 and this bit is RW, this bit is invalid and UNKNOWN.

On an implementation that does not support the trapping of floating-point exceptions this bit is RAZ/WI.

On an implementation that supports the trapping of floating-point exceptions and implements FPSCR.{Stride, Len} as RAZ, this bit is RAO/WI.

This field resets to an architecturally UNKNOWN value.

Bits [25:11]

Reserved, RES0.

VECITR, bits [10:8]

Vector iteration count. In Armv8, this field is RES1.

This field resets to an architecturally UNKNOWN value.

IDF, bit [7]

Input Denormal trapped exception bit. Valid only when the value of FPEXC.TFV is 1. When valid, it indicates whether an Input Denormal exception occurred while FPSCR.IDE was 1:

IDF	Meaning
0b0	Input Denormal exception has not occurred.
0b1	Input Denormal exception has occurred.

Input Denormal exceptions can occur only when FPSCR.FZ is 1.

Note

A half-precision floating-point value that is flushed to zero because the value of FPSCR.FZ16 is 1 does not generate an Input Denormal exception.

This bit must be cleared to 0 by the exception-handling routine.

When the value of FPEXC32_EL2.TFV is 0 and this bit is RW, this bit is invalid and UNKNOWN.

On an implementation that does not support the trapping of floating-point exceptions this bit is RAZ/WI.

This field resets to an architecturally UNKNOWN value.

Bits [6:5]

Reserved, RES0.

IXF, bit [4]

Inexact trapped exception bit. Valid only when the value of FPEXC.TFV is 1. When valid, it indicates whether an Inexact exception occurred while FPSCR.IXE was 1:

IXF	Meaning
0b0	Inexact exception has not occurred.
0b1	Inexact exception has occurred.

This bit must be cleared to 0 by the exception-handling routine.

When the value of FPEXC.TFV is 0 and this bit is RW, this bit is invalid and UNKNOWN.

On an implementation that does not support the trapping of floating-point exceptions this bit is RAZ/WI.

This field resets to an architecturally UNKNOWN value.

UFF, bit [3]

Underflow trapped exception bit. Valid only when the value of FPEXC.TFV is 1. When valid, it indicates whether an Underflow exception occurred while FPSCR.UFE was 1:

UFF	Meaning
0b0	Underflow exception has not occurred.
0b1	Underflow exception has occurred.

Underflow trapped exceptions can occur:

- On half-precision data-processing instructions only when FPSCR.FZ16 is 0.
- Otherwise only when FPSCR.FZ is 0.

This bit must be cleared to 0 by the exception-handling routine.

When the value of FPEXC32_EL2.TFV is 0 and this bit is RW, this bit is invalid and UNKNOWN.

On an implementation that does not support the trapping of floating-point exceptions this bit is RAZ/WI.

This field resets to an architecturally UNKNOWN value.

OFF, bit [2]

Overflow trapped exception bit. Valid only when the value of FPEXC.TFV is 1. When valid, it indicates whether an Overflow exception occurred while FPSCR.OFE was 1:

OFF	Meaning
0b0	Overflow exception has not occurred.
0b1	Overflow exception has occurred.

This bit must be cleared to 0 by the exception-handling routine.

When the value of FPEXC.TFV is 0 and this bit is RW, this bit is invalid and UNKNOWN.

On an implementation that does not support the trapping of floating-point exceptions this bit is RAZ/WI.

This field resets to an architecturally UNKNOWN value.

DZF, bit [1]

Divide by Zero trapped exception bit. Valid only when the value of FPEXC.TFV is 1. When valid, it indicates whether a Divide by Zero exception occurred while FPSCR.DZE was 1:

DZF	Meaning
0b0	Divide by Zero exception has not occurred.
0b1	Divide by Zero exception has occurred.

This bit must be cleared to 0 by the exception-handling routine.

When the value of FPEXC.TFV is 0 and this bit is RW, this bit is invalid and UNKNOWN.

On an implementation that does not support the trapping of floating-point exceptions this bit is RAZ/WI.

This field resets to an architecturally UNKNOWN value.

IOF, bit [0]

Invalid Operation trapped exception bit. Valid only when the value of FPEXC.TFV is 1. When valid, it indicates whether an Invalid Operation exception occurred while FPSCR.IOE was 1:

IOF	Meaning
0b0	Invalid Operation exception has not occurred.
0b1	Invalid Operation exception has occurred.

This bit must be cleared to 0 by the exception-handling routine.

When the value of FPEXC.TFV is 0 and this bit is RW, this bit is invalid and UNKNOWN.

On an implementation that does not support the trapping of floating-point exceptions this bit is RAZ/WI.

This field resets to an architecturally UNKNOWN value.

Accessing the FPEXC32_EL2

Accesses to this register use the following encodings:

```
MRS <Xt>, FPEXC32_EL2
```

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b11	0b100	0b0101	0b0011	0b000
~~0b11~~	~~0b0101~~	~~0b100~~	~~0b000~~	~~0b0011~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '0' && CPTR_EL2.TFP == '1' then
        AArch64.SystemAccessTrap(EL2, 0x07);
    elsif HCR_EL2.E2H == '1' && CPTR_EL2.FPEN == 'x0' then
        AArch64.SystemAccessTrap(EL2, 0x07);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TFP == '1' then
        AArch64.SystemAccessTrap(EL3, 0x07);
    else
        return FPEXC32_EL2;
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TFP == '1' then
        AArch64.SystemAccessTrap(EL3, 0x07);
    else
        return FPEXC32_EL2;
```

```
MSR FPEXC32_EL2, <Xt>
```

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b11	0b100	0b0101	0b0011	0b000
~~0b11~~	~~0b0101~~	~~0b100~~	~~0b000~~	~~0b0011~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '0' && CPTR_EL2.TFP == '1' then
        AArch64.SystemAccessTrap(EL2, 0x07);
    elsif HCR_EL2.E2H == '1' && CPTR_EL2.FPEN == 'x0' then
        AArch64.SystemAccessTrap(EL2, 0x07);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TFP == '1' then
        AArch64.SystemAccessTrap(EL3, 0x07);
    else
        FPEXC32_EL2 = X[t];
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TFP == '1' then
        AArch64.SystemAccessTrap(EL3, 0x07);
    else
        FPEXC32_EL2 = X[t];
```

27~~13~~/03~~12~~/2019~~2018~~ 21~~16~~:59~~42~~; e5e4db499bf9867a4b93324c4dbac985d3da93766~~379d01e197f1d40720d32d0f84e419e9187e009~~

(old) htmldiff from- (new)

GCR_EL1, Tag Control Register.

The GCR_EL1 characteristics are:

Purpose

Tag Control Register.

Configuration

This register is present only when ARMv8.5-MemTag is implemented. Otherwise, direct accesses to GCR_EL1 are UNDEFINED.

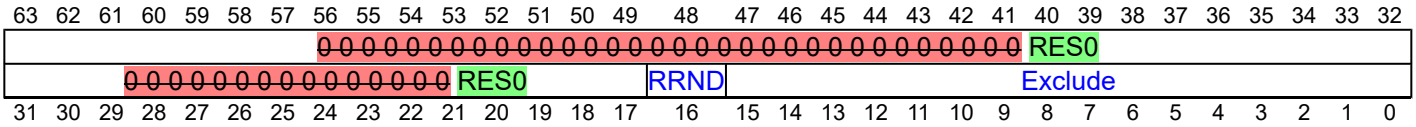
RW fields in this register reset to architecturally UNKNOWN values.

Attributes

GCR_EL1 is a 64-bit register.

Field descriptions

The GCR_EL1 bit assignments are:



Bits [63:17]

Reserved, RES0.

RRND, bit [16]

Controls whether RandomTag() generates a deterministic value solely based on the contents of RGSR_EL1, or a non-deterministic value.

RRND	Meaning
0b0	Generate a deterministic value based on RGSR_EL1.
0b1	Generate an IMPLEMENTATION DEFINED non-deterministic value.

Note

When the value of GCR_EL1.RRND is 1, the value generated does not need to be cryptographically random.

A similar algorithm to that used when RRND=0 but with free running clock is sufficient.

This field resets to an architecturally UNKNOWN value.

Exclude, bits [15:0]

Allocation Tag values excluded from selection by ChooseNonExcludedTag().

This field resets to an architecturally UNKNOWN value.

Accessing the GCR_EL1

Accesses to this register use the following encodings:

```
MRS <Xt>, GCR_EL1
```

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b11	0b000	0b0001	0b0000	0b110
~~0b11~~	~~0b0001~~	~~0b000~~	~~0b110~~	~~0b0000~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.ATA == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.ATA == '0' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return GCR_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.ATA == '0' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return GCR_EL1;
elsif PSTATE.EL == EL3 then
    return GCR_EL1;
```

```
MSR GCR_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b11	0b000	0b0001	0b0000	0b110
~~0b11~~	~~0b0001~~	~~0b000~~	~~0b110~~	~~0b0000~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.ATA == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.ATA == '0' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        GCR_EL1 = X[t];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.ATA == '0' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        GCR_EL1 = X[t];
elsif PSTATE.EL == EL3 then
    GCR_EL1 = X[t];
```

27~~13~~/03~~12~~/2019~~2018~~ 21~~16~~:59~~42~~; e5e4db499bf9867a4b93324c4dbac985d3da93766~~379d01e197f1d40720d32d0f84e419e9187e009~~

GMID_EL1, Multiple tag transfer ID register

The GMID_EL1 characteristics are:

Purpose

Indicates the block size that is accessed by the LDGM and STGM System instructions.

Configuration

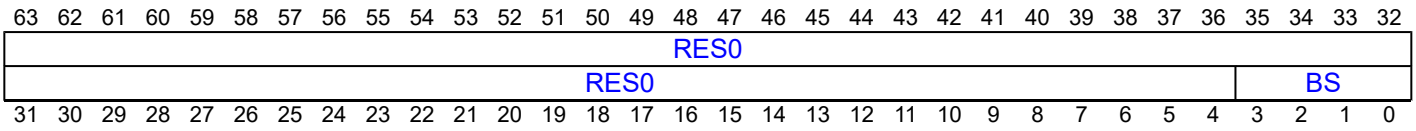
This register is present only when ARMv8.5-MemTag is implemented. Otherwise, direct accesses to GMID_EL1 are UNDEFINED.

Attributes

GMID_EL1 is a 64-bit register.

Field descriptions

The GMID_EL1 bit assignments are:



Bits [63:4]

Reserved, RES0.

BS, bits [3:0]

Log_2 of the block size in words. The minimum supported size is 16B (value == 2) and the maximum is 256B (value == 6).

Accessing the GMID_EL1

Accesses to this register use the following encodings:

```
MRS <Xt>, GMID_EL1
```

CRn	op0	op1	op2	CRm
0b0000	0b11	0b001	0b100	0b0000

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TID5 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        return GMID_EL1;
elsif PSTATE.EL == EL2 then
    return GMID_EL1;
elsif PSTATE.EL == EL3 then
    return GMID_EL1;
```

27/03/2019 21:59; e5e4db499bf9867a4b93324c4dbac985d3da9376

no old file htmldiff from- (new)

(old) htmldiff from- (new)

HCR_EL2, Hypervisor Configuration Register

The HCR_EL2 characteristics are:

Purpose

Provides configuration controls for virtualization, including defining whether various operations are trapped to EL2.

Configuration

AArch64 System register HCR_EL2 bits [31:0] are architecturally mapped to AArch32 System register HCR[31:0] .

AArch64 System register HCR_EL2 bits [63:32] are architecturally mapped to AArch32 System register HCR2[31:0] .

If EL2 is not implemented, this register is RES0 from EL3.

This register has no effect if EL2 is not enabled in the current Security state.

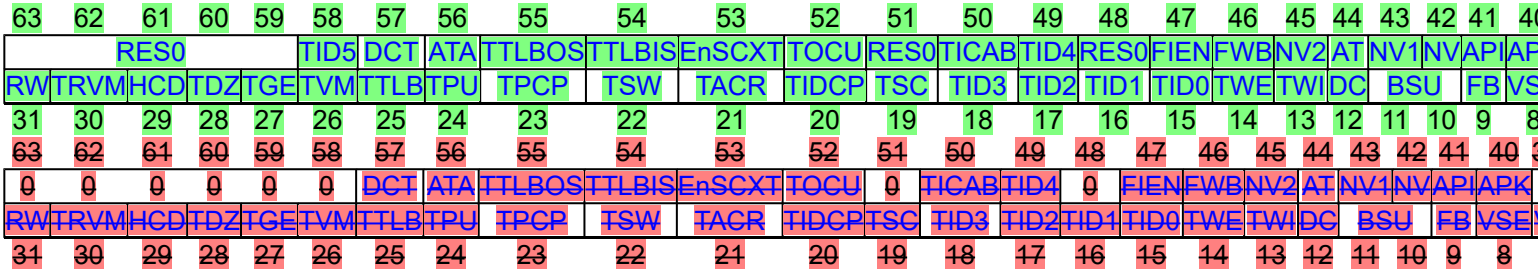
RW fields in this register reset to architecturally UNKNOWN values.

Attributes

HCR_EL2 is a 64-bit register.

Field descriptions

The HCR_EL2 bit assignments are:



Bits [63:59~~58~~]

Reserved, RES0.

TID5, bit [58]

When ARMv8.5-MemTag is implemented:

Trap ID group 5. Traps the following register accesses to EL2, when EL2 is enabled in the current Security state:

AArch64:

- GMID_EL1.

TID5	Meaning
0b0	This control does not cause any instructions to be trapped.
0b1	The specified EL1 and EL0 accesses to ID group 5 registers are trapped to EL2.

When the value of HCR_EL2.{E2H, TGE} is {1, 1}, this field has an Effective value of 0 for all purposes other than a direct read of the value of this bit.

This field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

DCT, bit [57]

When ARMv8.5-MemTag is implemented:

Default Cacheability Tagging. When HCR_EL2.DC is in effect, controls whether addresses are Tagged or Untagged.

DCT	Meaning
0b0	Addresses are Untagged.
0b1	Addresses are Tagged.

This field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

ATA, bit [56]

When ARMv8.5-MemTag is implemented:

Allocation Tag Access. When SCR_EL3.ATA=1 and HCR_EL2.{E2H,TGE} != {1,1}, controls EL1 and EL0 access to Allocation Tags.

When access ~~to Allocation Tags~~ is prevented:

- Instructions which Load or Store data are Unchecked.
- Instructions which Load or Store Allocation Tags treat the Allocation Tag as RAZ/WI.
- Instructions which insert Logical Address Tags into addresses treat the Allocation Tag used to generate the Logical Address Tag as 0.
- Cache maintenance instructions which invalidate Allocation Tags from caches behave as the equivalent Clean and Invalidate operation on Allocation Tags.
- MRS and MSR instructions at EL1 using GCR_EL1, RGSR_EL1, TFSR_EL1, TFSR_EL2, or TFSRE0_EL1 that are not UNDEFINED are trapped to EL2.

ATA	Meaning
0b0	Access ~~to Allocation Tags~~ is prevented.
0b1	Access ~~to Allocation Tags~~ is not prevented.

This field is permitted to be cached in a TLB.

This field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TTLBOS, bit [55]

When ARMv8.2-EVT is implemented:

Trap TLB maintenance instructions that operate on the Outer Shareable domain. Traps execution of those TLB maintenance instructions at EL1 using AArch64 to EL2 when EL2 is enabled in the current Security state. This applies to the following instructions:

TLBI VMALLE1OS, TLBI VAE1OS, TLBI ASIDE1OS,TLBI VAAE1OS, TLBI VALE1OS, TLBI VAALE1OS,TLBI RVAE1OS, TLBI RVAAE1OS,TLBI RVALE1OS, and TLBI RVAALE1OS.

TTLBOS	Meaning
0b0	This control does not cause any instructions to be trapped.
0b1	Execution of the specified instructions are trapped to EL2.

If ARMv8.2-EVT is not implemented, this field is RES0.

When ARMv8.1-VHE is implemented, and the value of HCR_EL2.{E2H, TGE} is {1, 1}, this field behaves as 0 for all purposes other than a direct read of the value of this bit.

This field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TTLBIS, bit [54]

When ARMv8.2-EVT is implemented:

Trap TLB maintenance instructions that operate on the Inner Shareable domain. Traps execution of those TLB maintenance instructions at EL1 using AArch64, and at EL1 using AArch32, to EL2 when EL2 is enabled in the current Security state. This applies to the following instructions:

- When EL1 is using AArch64, TLBI VMALLE1IS, TLBI VAE1IS, TLBI ASIDE1IS,TLBI VAAE1IS, TLBI VALE1IS, TLBI VAALE1IS,TLBI RVAE1IS, TLBI RVAAE1IS,TLBI RVALE1IS, and TLBI RVAALE1IS.
- When EL1 is using AArch32, TLBIALLIS, TLBIMVAIS, TLBIASIDIS, TLBIMVAAIS, TLBIMVALIS, and TLBIMVAALIS.

TTLBIS	Meaning
0b0	This control does not cause any instructions to be trapped.
0b1	Execution of the specified instructions are trapped to EL2.

If ARMv8.2-EVT is not implemented, this field is RES0.

When ARMv8.1-VHE is implemented, and the value of HCR_EL2.{E2H, TGE} is {1, 1}, this field behaves as 0 for all purposes other than a direct read of the value of this bit.

This field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EnSCXT, bit [53]

When ARMv8.0-CSV3 is implemented:

Enable Access to the SCXTNUM_EL1 and SCXTNUM_EL0 registers. The defined values are:

EnSCXT	Meaning
0b0	When (HCR_EL2.TGE==0 or HCR_EL2.E2H==0) and EL2 is enabled in the current Security state , EL1 and EL0 access to SCXTNUM_EL0 and EL1 access to SCXTNUM_EL1 is disabled by this mechanism, causing an exception to EL2, and the values of these registers to be treated as 0. When ((HCR_EL2.TGE==1 and HCR_EL2.E2H==1) and EL2 is enabled in the current Security state , EL0 access to SCXTNUM_EL0 is disabled by this mechanism, causing an exception to EL2, and the value of this register to be treated as 0.
0b1	This control does not cause accesses to SCXTNUM_EL0 or SCXTNUM_EL1 to be trapped.

When ARMv8.1-VHE is implemented, and the value of HCR_EL2.{E2H, TGE} is {1,1}, this bit has no effect on execution at EL0.

This field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TOCU, bit [52]

When ARMv8.2-EVT is implemented:

Trap cache maintenance instructions that operate to the Point of Unification. Traps execution of those cache maintenance instructions at EL1 or EL0 using AArch64, and at EL1 using AArch32, to EL2 when EL2 is enabled in the current Security state. This applies to the following instructions:

- When EL0 is using AArch64, IC IVAU, DC CVAU. However, if the value of SCTLR_EL1.UCI is 0 these instructions are UNDEFINED at EL0 and any resulting exception is higher priority than this trap to EL2.
- When EL1 is using AArch64, IC IVAU, IC IALLU, DC CVAU.
- When EL1 is using AArch32, ICIMVAU, ICIALLU, DCCMVAU.

Note

An exception generated because an instruction is UNDEFINED at EL0 is higher priority than this trap to EL2. In addition:

- IC IALLUIS and IC IALLU are always UNDEFINED at EL0 using AArch64.
- ICIMVAU, ICIALLU, ICIALLUIS, and DCCMVAU are always UNDEFINED at EL0 using AArch32.

TOCU	Meaning
0b0	This control does not cause any instructions to be trapped.
0b1	Execution of the specified instructions are trapped to EL2.

If ARMv8.2-EVT is not implemented, this field is RES0.

If the Point of Unification is before any level of data cache, it is IMPLEMENTATION DEFINED whether the execution of any data or unified cache clean by VA to the Point of Unification instruction can be trapped when the value of this control is 1.

If the Point of Unification is before any level of instruction cache, it is IMPLEMENTATION DEFINED whether the execution of any instruction cache invalidate to the Point of Unification instruction can be trapped when the value of this control is 1.

When ARMv8.1-VHE is implemented, and the value of HCR_EL2.{E2H, TGE} is {1, 1}, this field behaves as 0 for all purposes other than a direct read of the value of this bit.

This field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bit [51]

Reserved, RES0.

TICAB, bit [50]

When ARMv8.2-EVT is implemented:

Trap ICIALLUIS/IC IALLUIS cache maintenance instructions. Traps execution of those cache maintenance instructions at EL1 using AArch64, and at EL1 using AArch32, to EL2 when EL2 is enabled in the current Security state. This applies to the following instructions:

- When EL1 is using AArch64, IC IALLUIS.
- When EL1 is using AArch32, ICIALLUIS.

TICAB	Meaning
0b0	This control does not cause any instructions to be trapped.
0b1	EL1 execution of the specified instructions is trapped to EL2.

If ARMv8.2-EVT is not implemented, this field is RES0.

If the Point of Unification is before any level of instruction cache, it is IMPLEMENTATION DEFINED whether the execution of any instruction cache invalidate to the Point of Unification instruction can be trapped when the value of this control is 1.

When ARMv8.1-VHE is implemented, and the value of HCR_EL2.{E2H, TGE} is {1, 1}, this field behaves as 0 for all purposes other than a direct read of the value of this bit.

This field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TID4, bit [49]

When ARMv8.2-EVT is implemented:

Trap ID group 4. Traps the following register accesses to EL2, when EL2 is enabled in the current Security state:

AArch64:

- CCSIDR_EL1, CCSIDR2_EL1, CLIDR_EL1, and CSSELR_EL1.
- EL1 writes to CSSELR_EL1.

AArch32:

- EL1 reads of the CCSIDR, CCSIDR2, CLIDR, and CSSELR.
- EL1 writes to the CSSELR.

TID4	Meaning
0b0	This control does not cause any instructions to be trapped.
0b1	The specified EL1 and EL0 accesses to ID group 4 registers are trapped to EL2.

If ARMv8.2-EVT is not implemented, this field is RES0.

When ARMv8.1-VHE is implemented, and the value of HCR_EL2.{E2H, TGE} is {1, 1}, this field behaves as 0 for all purposes other than a direct read of the value of this bit.

This field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bit [48]

Reserved, RES0.

FIEN, bit [47]

When~~From~~ ARMv8.4-RAS is implemented~~Armv8.4~~:

Fault Injection Enable. Unless this bit is set to 1, accesses to the ERXPFGCDN_EL1, ERXPFGCTL_EL1, and ERXPFGF_EL1 registers from EL1 generate a Trap exception to EL2, when EL2 is enabled in the current Security state.

FIEN	Meaning
0b0	Accesses to the specified registers from EL1 are trapped to EL2, when EL2 is enabled in the current Security state.
0b1	This control does not cause any instructions to be trapped.

If EL2 is disabled in the current Security state, the Effective value of HCR_EL2.FIEN is 0b1.

If ~~the RAS Common Fault Injection Model Extension is not implemented, this field is~~ ERRIDR_EL1.NUM is zero, meaning no error records are implemented, or no error record accessible using System registers is owned by a node that implements the RAS Common Fault Injection Model Extension, then this bit might be RES0.

This field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

FWB, bit [46]

When ARMv8.4-S2FWB is implemented:

Defines the combined cacheability attributes in a 2 stage translation regime.

FWB	Meaning
0b0	When this bit is 0, then: • The combination of stage 1 and stage 2 translations on memory type and cacheability attributes are as described in the Armv8.0 architecture. For more information see D4.5.4 Combining the stage 1 and stage 2 attributes • The encoding of the stage 2 memory type and cacheability attributes in bits[5:2] of the stage 2 page or block descriptors are as described in the Armv8.0 architecture.
0b1	When this bit is 1, then: • Bit[5] of stage 2 page or block descriptor is RES0. • When bit[4] of stage 2 page or block descriptor is 1 and when: ◦ Bits[3:2] of stage 2 page or block descriptor are 0b11, the resultant memory type and inner or outer cachebility attribute is the same as the stage 1 memory type and inner or outer cacheability attribute. ◦ Bits[3:2] of stage 2 page or block descriptor are 0b10, the resultant memory type and attribute is Normal Write-Back. ◦ Bits[3:2] of stage 2 page or block descriptor are 0b0x, the resultant memory type will be Normal Non-cacheable except where the stage 1 memory type was Device->attr< the resultant memory type will be Device->attr< • When bit[4] of stage 2 page or block descriptor is 0 the memory type is Device, and when: ◦ Bits[3:2] of stage 2 page or block descriptor are 0b00, the resultant memory type is Device-nGnRnE. ◦ Bits[3:2] of stage 2 page or block descriptor are 0b01, the resultant memory type is Device-nGnRE. ◦ Bits[3:2] of stage 2 page or block descriptor are 0b10, the resultant memory type is Device-nGRE. ◦ Bits[3:2] of stage 2 page or block descriptor are 0b11, the resultant memory type is Device-GRE. • If the stage 1 translation specifies a cacheable memory type, then the stage 1 cache allocation hint is applied to the final cache allocation hint where the final memory type is cacheable. • If the stage 1 translation does not specify a cacheable memory type, then if the final memory type is cacheable, it is treated as read allocate, write allocate.

In Secure state, this bit applies to both the Secure stage 2 translation and the Non-secure stage 2 translation.

This bit is permitted to be cached in a TLB.

This field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

NV2, bit [45]

When ARMv8.4-NV is implemented:

Nested Virtualization. Changes the behaviors of HCR_EL2.{NV, NV1} to provide a mechanism for hardware to transform reads and writes from System registers into reads and writes from memory.

NV2	Meaning
0b0	This bit has no effect on the behavior of HCR_EL2.{NV, NV1}.
0b1	Redefines behavior of HCR_EL2{NV, NV1} to enable: • Transformation of read/writes to registers into read/writes to memory. • Redirection of EL2 registers to EL1 registers.

When this bit is 0, the behavior of HCR_EL2.{NV, NV1} is as defined for ARMv8.3-NV.

When this is bit is 1, then any exception taken from EL1 and taken to EL1 causes SPSR_EL1.M[3:2] to be set to 0b10 and not 0b01

This field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

AT, bit [44]

When ARMv8.3-NV is implemented:

Address Translation. EL1 execution of the following address translation instructions is trapped to EL2, when EL2 is enabled in the current Security state:

AT S1E0R, AT S1E0W, AT S1E1R, AT S1E1W, AT S1E1RP, AT S1E1WP

AT	Meaning
0b0	This control does not cause any instructions to be trapped.
0b1	EL1 execution of the specified instructions is trapped to EL2.

This field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

NV1, bit [43]

When ARMv8.4-NV is implemented:

Nested Virtualization.

NV1	Meaning
0b0	If HCR_EL2.{NV, NV2} are both 1, accesses executed from EL1 to implemented EL12, EL02, or EL2 registers are transformed to loads and stores. If HCR_EL2.NV2 is 0 or HCR_EL2.{NV, NV2} == {0, 1}, this control does not cause any instructions to be trapped.
0b1	If HCR_EL2.NV2 is 1, accesses executed from EL1 to implemented EL2 registers are transformed to loads and stores. If HCR_EL2.NV2 is 0, EL1 accesses to VBAR_EL1, ELR_EL1, and SPSR_EL1, are trapped to EL2, when EL2 is enabled in the current Security state.

If HCR_EL2.NV2 is 1, the value of HCR_EL2.NV1 defines which EL1 register accesses are transformed to loads and stores. These transformed accesses have priority over the trapping of registers.

The trapping of EL1 registers caused by other control bits has priority over the transformation of these accesses.

If a register is specified that is not implemented by an implementation, then access to that register is treated as unallocated.

For the list of registers affected, see Enhanced support for nested virtualization.

If HCR_EL2.{NV, NV1, NV2} are {1, 0, 0},any exception taken from EL1, and taken to EL1, causes the SPSR_EL1.M[3:2] to be set to `0b10`, and not `0b01`.

If HCR_EL2.{NV, NV1, NV2} are {1, 1, 0}, then:

- The EL1 translation table Block and Page descriptors:
  - Bit[54] holds the PXN instead of the UXN.
  - Bit[53] is RES0.
  - Bit[6] is treated as 0 regardless of the actual value.
- If Hierarchical Permissions are enabled, the EL1 translation table Table descriptors are as follows:
  - Bit[61] is treated as 0 regardless of the actual value.
  - Bit[60] holds the PXNTable instead of the UXNTable.
  - Bit[59] is RES0.
- When executing at EL1, the PSTATE.PAN bit is treated as zero for all purposes except reading the value of the bit.
- When executing at EL1, the LDTR* instructions are treated as the equivalent LDR* instructions, and the STTR* instructions are treated as the equivalent STR* instructions.

If HCR_EL2.{NV, NV1, NV2} are {0, 1, 0}, then the behavior is a CONSTRAINED UNPREDICTABLE choice of:

- Behaving as if HCR_EL2.NV is 1 and HCR_EL2.NV1 is 1 for all purposes other than reading than reading back the value of the HCR_EL2.NV bit.
- Behaving as if HCR_EL2.NV is 0 and HCR_EL2.NV1 is 0 for all purposes other than reading than reading back the value of the HCR_EL2.NV1 bit.
- Behaving with regard to the HCR_EL2.NV and HCR_EL2.NV1 bits behavior as defined in the rest of this description.

This bit is permitted to be cached in a TLB.

This field resets to an architecturally UNKNOWN value.

When ARMv8.3-NV is implemented:

Nested Virtualization. EL1 accesses to certain registers are trapped to EL2, when EL2 is enabled in the current Security state.

NV1	Meaning
`0b0`	This control does not cause any instructions to be trapped.
`0b1`	EL1 accesses to VBAR_EL1, ELR_EL1, SPSR_EL1 are trapped to EL2, when EL2 is enabled in the current Security state.

If HCR_EL2.NV is 1 and HCR_EL2.NV1 is 0 then the following effects also apply:

- Any exception taken from EL1, and taken to EL1, causes the SPSR_EL1.M[3:2] to be set to `0b10`, and not `0b01`.

If the bits HCR_EL2.NV and HCR_EL2.NV1 are both set to 1 then following effects also apply:

- The EL1 translation table Block and Page descriptors:
  - Bit[54] holds the PXN instead of the UXN.
  - Bit[53] is RES0.
  - Bit[6] is treated as 0 regardless of the actual value.
- If Hierarchical Permissions are enabled, the EL1 translation table Table descriptors are as follows:
  - Bit[61] is treated as 0 regardless of the actual value.
  - Bit[60] holds the PXNTable instead of the UXNTable.
  - Bit[59] is RES0.
- When executing at EL1, the PSTATE.PAN bit is treated as zero for all purposes except reading the value of the bit.
- When executing at EL1, the LDTR* instructions are treated as the equivalent LDR* instructions, and the STTR* instructions are treated as the equivalent STR* instructions.

If HCR_EL2.NV is 0 and HCR_EL2.NV1 is 1 then the behavior is a CONSTRAINED UNPREDICTABLE choice of:

- Behaving as if HCR_EL2.NV is 1 and HCR_EL2.NV1 is 1 for all purposes other than reading than reading back the value of the HCR_EL2.NV bit.
- Behaving as if HCR_EL2.NV is 0 and HCR_EL2.NV1 is 0 for all purposes other than reading than reading back the value of the HCR_EL2.NV1 bit.
- Behaving with regard to the HCR_EL2.NV and HCR_EL2.NV1 bits behavior as defined in the rest of this description.

This bit is permitted to be cached in a TLB.

This field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

NV, bit [42]

When ARMv8.4-NV is implemented:

Nested Virtualization.

When HCR_EL2.NV2 is 1, redefines register accesses so that:

- Instructions accessing the Special purpose registers SPSR_EL2 and ELR_EL2 instead access SPSR_EL1 and ELR_EL1 respectively.
- Instructions accessing the System registers ESR_EL2 and FAR_EL2 instead access ESR_EL1 and FAR_EL1.

When HCR_EL2.NV2 is 0, or if ARMv8.4-NV is not implemented, traps functionality that is permitted at EL2 and would be UNDEFINED at EL1 if this field was 0, when EL2 is enabled in the current Security state. This applies to the following operations:

- EL1 accesses to Special-purpose registers that are not UNDEFINED at EL2.
- EL1 accesses to System registers that are not UNDEFINED at EL2.
- Execution of EL1 or EL2 translation regime address translation and TLB maintenance instructions for EL2 and above.

NV	Meaning
0b0	When this bit is set to 0, HCR_EL2.NV2 == 0 for all purposes other than reading this register. This control does not cause any instructions to be trapped. When HCR_EL2.NV2 is 1, no ARMv8.4-NV functionality is implemented.
0b1	When HCR_EL2.NV2 is 0, or if ARMv8.4-NV is not implemented, EL1 accesses to the specified registers or the execution of the specified instructions are trapped to EL2, when EL2 is enabled in the current Security state. EL1 read accesses to the CurrentEL register return a value of 0x2. When HCR_EL2.NV2 is 1, this control redefines EL1 register accesses so that instructions accessing SPSR_EL2, ELR_EL2, ESR_EL2, and FAR_EL2 instead access SPSR_EL1, ELR_EL1, ESR_EL1, and FAR_EL1 respectively.

When HCR_EL2.NV2 is 0, or if ARMv8.4-NV is not implemented, then:

- The System or Special-purpose registers for which accesses are trapped are as follows:
  - Registers accessed using MRS or MSR with a name ending in _EL2.
  - Registers accessed using MRS or MSR with a name ending in _EL12.
  - Registers accessed using MRS or MSR with a name ending in _EL02.
  - Special-purpose registers SPSR_irq, SPSR_abt, SPSR_und and SPSR_fiq, accessed using MRS or MSR.
  - Special-purpose register SP_EL1 accessed using the dedicated MRS or MSR instruction.
- The instructions for which the execution is trapped are as follows:
  - EL2 translation regime Address Translation instructions and TLB maintenance instructions.
  - EL1 translation regime Address Translation instructions and TLB maintenance instructions that are only accessible from EL2 and EL3.
  - SMC in an implementation that does not include EL3 and when HCR_EL2.TSC is 1. HCR_EL2.TSC bit is not RES0 in this case.
  - The ERET, ERETAA, and ERETAB instructions.

Note

The priority of this trap is higher than the priority of the HCR_EL2.API trap. If both of these bits are set so that EL1 execution of an ERETAA or ERETAB instruction is trapped to EL2, then the syndrome reported is 0x1A.

This field resets to an architecturally UNKNOWN value.

When ARMv8.3-NV is implemented:

Nested Virtualization. Traps functionality that is permitted at EL2 and would be UNDEFINED at EL1 if this field was 0, when EL2 is enabled in the current Security state. This applies to the following operations:

- EL1 accesses to Special-purpose registers that are not UNDEFINED at EL2.
- EL1 accesses to System registers that are not UNDEFINED at EL2.
- Execution of EL1 or EL2 translation regime address translation and TLB maintenance instructions for EL2 and above.

The possible values are:

NV	Meaning
0b0	This control does not cause any instructions to be trapped.
0b1	EL1 accesses to the specified registers or the execution of the specified instructions are trapped to EL2, when EL2 is enabled in the current Security state. EL1 read accesses to the CurrentEL register return a value of 0x2.

The System or Special-purpose registers for which accesses are trapped are as follows:

- Registers accessed using MRS or MSR with a name ending in _EL2.
- Registers accessed using MRS or MSR with a name ending in _EL12.
- Registers accessed using MRS or MSR with a name ending in _EL02.
- Special-purpose registers SPSR_irq, SPSR_abt, SPSR_und and SPSR_fiq, accessed using MRS or MSR.
- Special-purpose register SP_EL1 accessed using the dedicated MRS or MSR instruction.

The instructions for which the execution is trapped are as follows:

- EL2 translation regime Address Translation instructions and TLB maintenance instructions.
- EL1 translation regime Address Translation instructions and TLB maintenance instructions that are only accessible from EL2 and EL3.
- The ERET, ERETAA, and ERETAB instructions.

Note

The priority of this trap is higher than the priority of the HCR_EL2.API trap. If both of these bits are set so that EL1 execution of an ERETAA or ERETAB instruction is trapped to EL2, then the syndrome reported is 0x1A.

- SMC in an implementation that does not include EL3 and when HCR_EL2.TSC is 1. HCR_EL2.TSC bit is not RES0 in this case.

This bit is permitted to be cached in a TLB.

This field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

API, bit [41]

When ARMv8.3-PAuth is implemented:

Controls the use of instructions related to Pointer Authentication:

- PACGA.
- AUTDA, AUTDB, AUTDZA, AUTDZB, AUTIA, AUTIA1716, AUTIASP, AUTIAZ, AUTIB, AUTIB1716, AUTIBSP, AUTIBZ, AUTIZA, AUTIZB, PACDA, PACDB, PACDZA, PACDZB, PACIA, PACIA1716, PACIASP, PACIAZ, PACIB, PACIB1716, PACIBSP, PACIBZ, PACIZA, PACIZ, RETAA, RETAB, BRAA, BRAB, BLRAA, BLRAB, BRAAZ, BRABZ, BLRAAZ, BLRABZ, ERETAA, ERETAB, LDRAA and LDRAB when:
  - In EL0, when HCR_EL2.TGE==0 or HCR_EL2.E2H==0, and the associated SCTLR_EL1.En<N><M>==1.
  - In EL1, the associated SCTLR_EL1.En<N><M>==1.

API	Meaning
0b0	The instructions related to Pointer Authentication are trapped to EL2, when EL2 is enabled in the current Security state and the instructions are enabled for the EL1&0 translation regime, from: • EL0 when HCR_EL2.TGE==0 or HCR_EL2.E2H==0. • EL1. If HCR_EL2.NV is 1, the HCR_EL2.NV trap takes precedence over the HCR_EL2.API trap for the ERETAA and ERETAB instructions.
0b1	This control does not cause any instructions to be trapped.

~~**Note**~~

~~If ARMv8.3-PAuth is implemented but EL2 is not implemented or disabled in the current Security state, the system behaves as if this bit is 1.~~

If ARMv8.3-PAuth is implemented but EL2 is not implemented or disabled in the current Security state, the system behaves as if this bit is 1.

This field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

APK, bit [40]

When ARMv8.3-PAuth is implemented:

Trap registers holding "key" values for Pointer Authentication. Traps accesses to the following registers from EL1 to EL2, when EL2 is enabled in the current Security state:

- APIAKeyLo_EL1, APIAKeyHi_EL1, APIBKeyLo_EL1, APIBKeyHi_EL1, APDAKeyLo_EL1, APDAKeyHi_EL1, APDBKeyLo_EL1, APDBKeyHi_EL1, APGAKeyLo_EL1, and APGAKeyHi_EL1.

APK	Meaning
0b0	Access to the registers holding "key" values for pointer authentication from EL1 are trapped to EL2, when EL2 is enabled in the current Security state.
0b1	This control does not cause any instructions to be trapped.

Note

If ARMv8.3-PAuth is implemented but EL2 is not implemented or is disabled in the current Security state, the system behaves as if this bit is 1.

This field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bit [39]

Reserved, RES0.

MIOCNCE, bit [38]

Mismatched Inner/Outer Cacheable Non-Coherency Enable, for the EL1&0 translation regimes.

MIOCNCE	Meaning
0b0	For the EL1&0 translation regimes, for permitted accesses to a memory location that use a common definition of the Shareability and Cacheability of the location, there must be no loss of coherency if the Inner Cacheability attribute for those accesses differs from the Outer Cacheability attribute.
0b1	For the EL1&0 translation regimes, for permitted accesses to a memory location that use a common definition of the Shareability and Cacheability of the location, there might be a loss of coherency if the Inner Cacheability attribute for those accesses differs from the Outer Cacheability attribute.

For more information see 'Mismatched memory attributes' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section B2 (The AArch64 Application Level Memory Model).

This field can be implemented as RAZ/WI.

When ARMv8.1-VHE is implemented, and the value of HCR_EL2.{E2H, TGE} is {1, 1}, the PE ignores the value of this field for all purposes other than a direct read of this field.

This field resets to an architecturally UNKNOWN value.

TEA, bit [37]

Route synchronous External abort exceptions to EL2. If the RAS Extension is implemented, the possible values of this bit are:

TEA	Meaning
0b0	This control does not cause exceptions to be routed from EL0 and EL1 to EL2.
0b1	Route synchronous External abort exceptions from EL0 and EL1 to EL2, when EL2 is enabled in the current Security state, if not routed to EL3.

When the RAS Extension is not implemented, this field is RES0.

This field resets to an architecturally UNKNOWN value.

TERR, bit [36]

When RAS is implemented:

Trap Error record accesses. Trap accesses to the following registers from EL1 to EL2:

EL1 using AArch64: ERRIDR_EL1, ERRSELR_EL1, ERXADDR_EL1, ERXCTLR_EL1, ERXFR_EL1, ERXMISC0_EL1, ERXMISC1_EL1, and ERXSTATUS_EL1. When ARMv8.4-RAS is implemented, ERXMISC2_EL1, and ERXMISC3_EL1.

EL1 using AArch32: ERRIDR, ERRSELR, ERXADDR, ERXADDR2, ERXCTLR, ERXCTLR2, ERXFR, ERXFR2, ERXMISC0, ERXMISC1, ERXMISC2, ERXMISC3, and ERXSTATUS. When ARMv8.4-RAS is implemented, ERXMISC4, ERXMISC5, ERXMISC6, and ERXMISC7.

TERR	Meaning
0b0	This control does not cause any instructions to be trapped.
0b1	Accesses to the specified registers from EL1 generate a Trap exception to EL2, when EL2 is enabled in the current Security state.

This field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TLOR, bit [35]

From Armv8.1:

Trap LOR registers. Traps accesses to the LORSA_EL1, LOREA_EL1, LORN_EL1, LORC_EL1, and LORID_EL1 registers from EL1 to EL2, when EL2 is enabled in the current Security state.

TLOR	Meaning
0b0	This control does not cause any instructions to be trapped.
0b1	EL1 accesses to the LOR registers are trapped to EL2, when EL2 is enabled in the current Security state.

When HCR_EL2.TGE is 1, the PE ignores the value of this field for all purposes other than a direct read of this field.

This field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

E2H, bit [34]

From Armv8.1:

EL2 Host. Enables a configuration where a Host Operating System is running in EL2, and the Host Operating System's applications are running in EL0.

E2H	Meaning
0b0	The facilities to support a Host Operating System at EL2 are disabled.
0b1	The facilities to support a Host Operating System at EL2 are enabled.

For information on the behavior of this bit see Behavior of HCR_EL2.E2H.

This bit is permitted to be cached in a TLB.

This field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

ID, bit [33]

Stage 2 Instruction access cacheability disable. For the EL1&0 translation regime, when EL2 is enabled in the current Security state and HCR_EL2.VM==1, this control forces all stage 2 translations for instruction accesses to Normal memory to be Non-cacheable.

ID	Meaning
0b0	This control has no effect on stage 2 of the EL1&0 translation regime.
0b1	Forces all stage 2 translations for instruction accesses to Normal memory to be Non-cacheable.

This bit has no effect on the EL2, EL2&0, or EL3 translation regimes.

When ARMv8.1-VHE is implemented, and the value of HCR_EL2.{E2H, TGE} is {1, 1}, the PE ignores the value of this field for all purposes other than a direct read of this field.

This field resets to an architecturally UNKNOWN value.

CD, bit [32]

Stage 2 Data access cacheability disable. For the EL1&0 translation regime, when EL2 is enabled in the current Security state and HCR_EL2.VM==1, this control forces all stage 2 translations for data accesses and translation table walks to Normal memory to be Non-cacheable.

CD	Meaning
0b0	This control has no effect on stage 2 of the EL1&0 translation regime for data accesses and translation table walks.
0b1	Forces all stage 2 translations for data accesses and translation table walks to Normal memory to be Non-cacheable.

This bit has no effect on the EL2, EL2&0, or EL3 translation regimes.

When ARMv8.1-VHE is implemented, and the value of HCR_EL2.{E2H, TGE} is {1, 1}, the PE ignores the value of this field for all purposes other than a direct read of this field.

This field resets to an architecturally UNKNOWN value.

RW, bit [31]

Execution state control for lower Exception levels:

RW	Meaning
0b0	Lower levels are all AArch32.
0b1	The Execution state for EL1 is AArch64. The Execution state for EL0 is determined by the current value of PSTATE.nRW when executing at EL0.

If AArch32 state is not supported by the implementation at EL1, then this bit is RAO/WI.

In an implementation that includes EL3, when EL2 is not enabled in Secure state, the PE behaves as if this bit has the same value as the SCR_EL3.RW bit for all purposes other than a direct read or write access of HCR_EL2.

The RW bit is permitted to be cached in a TLB.

When ARMv8.1-VHE is implemented, and the value of HCR_EL2.{E2H, TGE} is {1, 1}, this field behaves as 1 for all purposes other than a direct read of the value of this bit.

This field resets to an architecturally UNKNOWN value.

TRVM, bit [30]

Trap Reads of Virtual Memory controls. Traps EL1 reads of the virtual memory control registers to EL2, when EL2 is enabled in the current Security state, from both Execution states. The registers for which read accesses are trapped are as follows:

EL1 using AArch64: SCTLR_EL1, TTBR0_EL1, TTBR1_EL1, TCR_EL1, ESR_EL1, FAR_EL1, AFSR0_EL1, AFSR1_EL1, MAIR_EL1, AMAIR_EL1, CONTEXTIDR_EL1.

EL1 using AArch32: SCTLR, TTBR0, TTBR1, TTBCR, TTBCR2, DACR, DFSR, IFSR, DFAR, IFAR, ADFSR, AIFSR, PRRR, NMRR, MAIR0, MAIR1, AMAIR0, AMAIR1, CONTEXTIDR.

TRVM	Meaning
0b0	This control does not cause any instructions to be trapped.
0b1	EL1 read accesses to the specified Virtual Memory controls are trapped to EL2, when EL2 is enabled in the current Security state.

When HCR_EL2.TGE is 1, the PE ignores the value of this field for all purposes other than a direct read of this field.

This field resets to an architecturally UNKNOWN value.

HCD, bit [29]

HVC instruction disable. Disables EL1 execution of HVC instructions, from both Execution states, when EL2 is enabled in the current Security state.

HCD	Meaning
0b0	HVC instruction execution is enabled at EL2 and EL1.
0b1	HVC instructions are UNDEFINED at EL2 and EL1. Any resulting exception is taken to the Exception level at which the HVC instruction is executed.

Note

HVC instructions are always UNDEFINED at EL0.

This bit is only implemented if EL3 is not implemented. Otherwise, it is RES0.

When HCR_EL2.TGE is 1, the PE ignores the value of this field for all purposes other than a direct read of this field.

This field resets to an architecturally UNKNOWN value.

TDZ, bit [28]

Trap DC ZVA instructions. Traps EL0 and EL1 execution of DC ZVA instructions to EL2, when EL2 is enabled in the current Security state, from AArch64 state only.

If ARMv8.5-MemTag is implemented, this trap also applies to DC GVA and DC GZVA.

TDZ	Meaning
0b0	This control does not cause any instructions to be trapped.
0b1	In AArch64 state, any attempt to execute an instruction this trap applies to at EL1, or at EL0 when the instruction is not UNDEFINED at EL0, is trapped to EL2 when EL2 is enabled in the current Security state. Reading the DCZID_EL0 returns a value that indicates that the instructions this trap applies to are not supported.

When ARMv8.1-VHE is implemented, and the value of HCR_EL2.{E2H, TGE} is {1, 1}, this field behaves as 0 for all purposes other than a direct read of the value of this bit.

This field resets to an architecturally UNKNOWN value.

TGE, bit [27]

Trap General Exceptions, from EL0.

TGE	Meaning
0b0	This control has no effect on execution at EL0.
0b1	When EL2 is not enabled in the current Security state, this control has no effect on execution at EL0. When EL2 is enabled in the current Security state, in all cases: • All exceptions that would be routed to EL1 are routed to EL2. • The SCTLR_EL1.M field, or the SCTLR.M field if EL1 is using AArch32, is treated as being 0 for all purposes other than returning the result of a direct read of SCTLR_EL1 or SCTLR. • All virtual interrupts are disabled. • Any IMPLEMENTATION DEFINED mechanisms for signaling virtual interrupts are disabled. • An exception return to EL1 is treated as an illegal exception return. • The MDCR_EL2.{TDRA, TDOSA, TDA, TDE} fields are treated as being 1 for all purposes other than returning the result of a direct read of MDCR_EL2. In addition, when EL2 is enabled in the current Security state, if: • HCR_EL2.E2H is 0, the Effective values of the HCR_EL2.{FMO, IMO, AMO} fields are 1. • HCR_EL2.E2H is 1, the Effective values of the HCR_EL2.{FMO, IMO, AMO} fields are 0. For further information on the behavior of this bit when E2H is 1, see Behavior of HCR_EL2.E2H.

HCR_EL2.TGE must not be cached in a TLB.

This field resets to an architecturally UNKNOWN value.

TVM, bit [26]

Trap Virtual Memory controls. Traps EL1 writes to the virtual memory control registers to EL2, when EL2 is enabled in the current Security state, from both Execution states. The registers for which write accesses are trapped are as follows:

EL1 using AArch64: SCTLR_EL1, TTBR0_EL1, TTBR1_EL1, TCR_EL1, ESR_EL1, FAR_EL1, AFSR0_EL1, AFSR1_EL1, MAIR_EL1, AMAIR_EL1, CONTEXTIDR_EL1.

EL1 using AArch32: SCTLR, TTBR0, TTBR1, TTBCR, TTBCR2, DACR, DFSR, IFSR, DFAR, IFAR, ADFSR, AIFSR, PRRR, NMRR, MAIR0, MAIR1, AMAIR0, AMAIR1, CONTEXTIDR.

TVM	Meaning
0b0	This control does not cause any instructions to be trapped.
0b1	EL1 write accesses to the specified EL1 virtual memory control registers are trapped to EL2, when EL2 is enabled in the current Security state.

When HCR_EL2.TGE is 1, the PE ignores the value of this field for all purposes other than a direct read of this field.

This field resets to an architecturally UNKNOWN value.

TTLB, bit [25]

When ARMv8.4-TLBI is implemented:

Trap TLB maintenance instructions. Traps EL1 execution of TLB maintenance instructions to EL2, when EL2 is enabled in the current Security state, from both Execution states. This applies to the following instructions:

- When EL1 is using AArch64, TLBI VMALLE1IS, TLBI VAE1IS, TLBI ASIDE1IS, TLBI VAAE1IS, TLBI VALE1IS, TLBI VAALE1IS, TLBI VMALLE1, TLBI VAE1, TLBI ASIDE1, TLBI VAAE1, TLBI VALE1, TLBI VAALE1, TLBI VMALLE1OS, TLBI VAE1OS, TLBI ASIDE1OS, TLBI VAAE1OS, TLBI VALE1OS, TLBI VAALE1OS, TLBI RVAE1, TLBI RVAE1IS, TLBI RVAE1OS, TLBI RVAAE1, TLBI RVAAE1IS, TLBI RVAAE1OS, TLBI RVALE1, TLBI RVALE1IS, TLBI RVALE1OS, TLBI RVAALE1, TLBI RVAALE1IS, TLBI RVAALE1OS.
- When EL1 is using AArch32, TLBIALLIS, TLBIMVAIS, TLBIASIDIS, TLBIMVAAIS, TLBIMVALIS, TLBIMVAALIS, ITLBIALL, ITLBIMVA, ITLBIASID, DTLBIALL, DTLBIMVA, DTLBIASID, TLBIALL, TLBIMVA, TLBIASID, TLBIMVAA, TLBIMVAL, TLBIMVAAL

TTLB	Meaning
0b0	This control does not cause any instructions to be trapped.
0b1	EL1 execution of the specified TLB maintenance instructions are trapped to EL2, when EL2 is enabled in the current Security state.

When HCR_EL2.TGE is 1, the PE ignores the value of this field for all purposes other than a direct read of this field.

This field resets to an architecturally UNKNOWN value.

From Armv8.0:

Trap TLB maintenance instructions. Traps EL1 execution of TLB maintenance instructions to EL2, when EL2 is enabled in the current Security state, from both Execution states. This applies to the following instructions:

- When EL1 is using AArch64, TLBI VMALLE1IS, TLBI VAE1IS, TLBI ASIDE1IS, TLBI VAAE1IS, TLBI VALE1IS, TLBI VAALE1IS, TLBI VMALLE1, TLBI VAE1, TLBI ASIDE1, TLBI VAAE1, TLBI VALE1, TLBI VAALE1.
- When EL1 is using AArch32, TLBIALLIS, TLBIMVAIS, TLBIASIDIS, TLBIMVAAIS, TLBIMVALIS, TLBIMVAALIS, ITLBIALL, ITLBIMVA, ITLBIASID, DTLBIALL, DTLBIMVA, DTLBIASID, TLBIALL, TLBIMVA, TLBIASID, TLBIMVAA, TLBIMVAL, TLBIMVAAL

TTLB	Meaning
0b0	This control does not cause any instructions to be trapped.
0b1	EL1 execution of the specified TLB maintenance instructions are trapped to EL2, when EL2 is enabled in the current Security state.

When HCR_EL2.TGE is 1, the PE ignores the value of this field for all purposes other than a direct read of this field.

This field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TPU, bit [24]

Trap cache maintenance instructions that operate to the Point of Unification. Traps execution of those cache maintenance instructions at EL1 or EL0 using AArch64, and at EL1 using AArch32, to EL2 when EL2 is enabled in the current Security state. This applies to the following instructions:

- When EL0 is using AArch64, IC IVAU, DC CVAU. However, if the value of SCTLR_EL1.UCI is 0 these instructions are UNDEFINED at EL0 and any resulting exception is higher priority than this trap to EL2.
- When EL1 is using AArch64, IC IVAU, IC IALLU, IC IALLUIS, DC CVAU.
- When EL1 is using AArch32, ICIMVAU, ICIALLU, ICIALLUIS, DCCMVAU.

Note

An exception generated because an instruction is UNDEFINED at EL0 is higher priority than this trap to EL2. In addition:

- IC IALLUIS and IC IALLU are always UNDEFINED at EL0 using AArch64.
- ICIMVAU, ICIALLU, ICIALLUIS, and DCCMVAU are always UNDEFINED at EL0 using AArch32.

TPU	Meaning
0b0	This control does not cause any instructions to be trapped.
0b1	Execution of the specified instructions is trapped to EL2, when EL2 is enabled in the current Security state.

If the Point of Unification is before any level of data cache, it is IMPLEMENTATION DEFINED whether the execution of any data or unified cache clean by VA to the Point of Unification instruction can be trapped when the value of this control is 1.

If the Point of Unification is before any level of instruction cache, it is IMPLEMENTATION DEFINED whether the execution of any instruction cache invalidate to the Point of Unification instruction can be trapped when the value of this control is 1.

When ARMv8.1-VHE is implemented, and the value of HCR_EL2.{E2H, TGE} is {1, 1}, this field behaves as 0 for all purposes other than a direct read of the value of this bit.

This field resets to an architecturally UNKNOWN value.

TPCP, bit [23]

When ARMv8.2-DCPoP is implemented:

Trap data or unified cache maintenance instructions that operate to the Point of Coherency or Persistence. Traps execution of those cache maintenance instructions at EL1 or EL0 using AArch64, and at EL1 using AArch32, to EL2 when EL2 is enabled in the current Security state. This applies to the following instructions:

- When EL0 is using AArch64, DC CIVAC, DC CVAC, DC CVAP. However, if the value of SCTLR_EL1.UCI is 0 these instructions are UNDEFINED at EL0 and any resulting exception is higher priority than this trap to EL2.
- When EL1 is using AArch64, DC IVAC, DC CIVAC, DC CVAC, DC CVAP.
- When EL1 is using AArch32, DCIMVAC, DCCIMVAC, DCCMVAC.

If ARMv8.2-DCCVADP is implemented, this trap also applies to DC CVADP.

If ARMv8.5-MemTag is implemented, this trap also applies to DC CIGVAC, DC CIGDVAC, DC IGVAC, DC IGDVAC, DC CGVAC, DC CGDVAC, DC CGVAP and DC CGDVAP.

If ARMv8.2-DCCVADP and ARMv8.5-MemTag are implemented, this trap also applies to DC CGVADP and DC CGDVADP.

Note

- An exception generated because an instruction is UNDEFINED at EL0 is higher priority than this trap to EL2. In addition:
  - AArch64 instructions which invalidate by VA to the Point of Coherency are always UNDEFINED at EL0 using AArch64.
  - DCIMVAC, DCCIMVAC, and DCCMVAC are always UNDEFINED at EL0 using AArch32.
- In Armv8.0 this field is named TPC. From Armv8.2 it is named TPCP.

TPCP	Meaning
0b0	This control does not cause any instructions to be trapped.
0b1	Execution of the specified instructions is trapped to EL2, when EL2 is enabled in the current Security state.

If the Point of Coherency is before any level of data cache, it is IMPLEMENTATION DEFINED whether the execution of any data or unified cache clean, invalidate, or clean and invalidate instruction that operates by VA to the point of coherency can be trapped when the value of this control is 1.

If HCR_EL2.{E2H, TGE} is set to {1, 1}, this field behaves as 0 for all purposes other than a direct read of the value of this bit.

This field resets to an architecturally UNKNOWN value.

Otherwise:

Trap data or unified cache maintenance instructions that operate to the Point of Coherency. Traps execution of those cache maintenance instructions at EL1 or EL0 using AArch64, and at EL1 using AArch32, to EL2 when EL2 is enabled in the current Security state. This applies to the following instructions:

- When EL0 is using AArch64, DC CIVAC, DC CVAC. However, if the value of SCTLR_EL1.UCI is 0 these instructions are UNDEFINED at EL0 and any resulting exception is higher priority than this trap to EL2.
- When EL1 is using AArch64, DC IVAC, DC CIVAC, DC CVAC.
- When EL1 is using AArch32, DCIMVAC, DCCIMVAC, DCCMVAC.

Note

- An exception generated because an instruction is UNDEFINED at EL0 is higher priority than this trap to EL2. In addition:
  - AArch64 instructions which invalidate by VA to the Point of Coherency are always UNDEFINED at EL0 using AArch64.
  - DCIMVAC, DCCIMVAC, and DCCMVAC are always UNDEFINED at EL0 using AArch32.
- In Armv8.0 this field is named TPC. From Armv8.2 it is named TPCP.

TPC	Meaning
0b0	This control does not cause any instructions to be trapped.
0b1	Execution of the specified instructions is trapped to EL2, when EL2 is enabled in the current Security state.

If the Point of Coherency is before any level of data cache, it is IMPLEMENTATION DEFINED whether the execution of any data or unified cache clean, invalidate, or clean and invalidate instruction that operates by VA to the point of coherency can be trapped when the value of this control is 1.

When ARMv8.1-VHE is implemented, and the value of HCR_EL2.{E2H, TGE} is {1, 1}, this field behaves as 0 for all purposes other than a direct read of the value of this bit.

This field resets to an architecturally UNKNOWN value.

TSW, bit [22]

Trap data or unified cache maintenance instructions that operate by Set/Way. Traps execution of those cache maintenance instructions at EL1 using AArch64, and at EL1 using AArch32, to EL2 when EL2 is enabled in the current Security state. This applies to the following instructions:

- When EL1 is using AArch64, DC ISW, DC CSW, DC CISW.
- When EL1 is using AArch32, DCISW, DCCSW, DCCISW.

If ARMv8.5-MemTag is implemented, this trap also applies to DC IGSW, DC IGDSW, DC CGSW, DC CGDW, DC CIGSW, and DC CIGDSW.

Note

An exception generated because an instruction is UNDEFINED at EL0 is higher priority than this trap to EL2, and these instructions are always UNDEFINED at EL0.

TSW	Meaning
0b0	This control does not cause any instructions to be trapped.
0b1	Execution of the specified instructions is trapped to EL2, when EL2 is enabled in the current Security state.

When HCR_EL2.TGE is 1, the PE ignores the value of this field for all purposes other than a direct read of this field.

This field resets to an architecturally UNKNOWN value.

TACR, bit [21]

Trap Auxiliary Control Registers. Traps EL1 accesses to the Auxiliary Control Registers to EL2, when EL2 is enabled in the current Security state, from both Execution states. This applies to the following register accesses:

- EL1 using AArch64: ACTLR_EL1.
- EL1 using AArch32: ACTLR and, if implemented, ACTLR2.

TACR	Meaning
0b0	This control does not cause any instructions to be trapped.
0b1	EL1 accesses to the specified registers are trapped to EL2, when EL2 is enabled in the current Security state.

When HCR_EL2.TGE is 1, the PE ignores the value of this field for all purposes other than a direct read of this field.

This field resets to an architecturally UNKNOWN value.

TIDCP, bit [20]

Trap IMPLEMENTATION DEFINED functionality. Traps EL1 accesses to the encodings reserved for IMPLEMENTATION DEFINED functionality to EL2, when EL2 is enabled in the current Security state. This applies to the following register accesses:

AArch64: The following reserved encoding spaces:

- IMPLEMENTATION DEFINED System instructions, which are accessed using SYS and SYSL, with CRn == {11, 15}.
- IMPLEMENTATION DEFINED System registers, which are accessed using MRS and MSR with the S3_<op1>_<Cn>_<Cm>_<op2> register name.

AArch32: MCR and MRC instructions accessing the following encodings:

- All coproc==p15, CRn==c9, opc1 == {0-7}, CRm == {c0-c2, c5-c8}, opc2 == {0-7}.
- All coproc==p15, CRn==c10, opc1 =={0-7}, CRm == {c0, c1, c4, c8}, opc2 == {0-7}.
- All coproc==p15, CRn==c11, opc1=={0-7}, CRm == {c0-c8, c15}, opc2 == {0-7}.

When the value of HCR_EL2.TIDCP is 1, it is IMPLEMENTATION DEFINED whether any of this functionality accessed from EL0 is trapped to EL2. If it is not, then it is UNDEFINED, and any attempt to access it from EL0 generates an exception that is taken to EL1.

TIDCP	Meaning
0b0	This control does not cause any instructions to be trapped.
0b1	EL1 accesses to or execution of the specified encodings reserved for IMPLEMENTATION DEFINED functionality are trapped to EL2, when EL2 is enabled in the current Security state.

This field resets to an architecturally UNKNOWN value.

TSC, bit [19]

Trap SMC instructions. Traps EL1 execution of SMC instructions to EL2, when EL2 is enabled in the current Security state, from both Execution states.

TSC	Meaning
0b0	This control does not cause any instructions to be trapped.
0b1	If EL3 is implemented, then any attempt to execute an SMC instruction at EL1 using AArch64 or EL1 using AArch32 is trapped to EL2, when EL2 is enabled in the current Security state, regardless of the value of SCR_EL3.SMD. If EL3 is not implemented, ARMv8.3-NV is implemented, and HCR_EL2.NV is 1, then any attempt to execute an SMC instruction at EL1 using AArch64 is trapped to EL2, when EL2 is enabled in the current Security state.

In AArch32 state, the Armv8-A architecture permits, but does not require, this trap to apply to conditional SMC instructions that fail their condition code check, in the same way as with traps on other conditional instructions.

If EL3 is not implemented, and HCR_EL2.NV is 0, this bit is RES0.

When HCR_EL2.TGE is 1, the PE ignores the value of this field for all purposes other than a direct read of this field.

This field resets to an architecturally UNKNOWN value.

TID3, bit [18]

Trap ID group 3. Traps EL1 reads of the following registers to EL2, when EL2 is enabled in the current Security state:

AArch64: ID_PFR0_EL1, ID_PFR1_EL1, ID_DFR0_EL1, ID_AFR0_EL1, ID_MMFR0_EL1, ID_MMFR1_EL1, ID_MMFR2_EL1, ID_MMFR3_EL1, ID_ISAR0_EL1, ID_ISAR1_EL1, ID_ISAR2_EL1, ID_ISAR3_EL1, ID_ISAR4_EL1, ID_ISAR5_EL1, ID_ISAR6_EL1, MVFR0_EL1, MVFR1_EL1, MVFR2_EL1, ID_AA64PFR0_EL1, ID_AA64PFR1_EL1, ID_AA64DFR0_EL1, ID_AA64DFR1_EL1, ID_AA64ISAR0_EL1, ID_AA64ISAR1_EL1, ID_AA64MMFR0_EL1, ID_AA64MMFR1_EL1, ID_AA64MMFR2_EL1, ID_AA64AFR0_EL1, ID_AA64AFR1_EL1, ID_AA64ZFR0_EL1 (where SVE is implemented), and ID_MMFR4_EL1, except that if ID_MMFR4_EL1 is implemented as RAZ/WI then it is IMPLEMENTATION DEFINED whether accesses to ID_MMFR4_EL1 are trapped.

It is IMPLEMENTATION DEFINED whether this field traps MRS accesses to encodings in the following range that are not already mentioned in this field description:

- Op0 == 3, op1 == 0, CRn == c0, CRm == {c2-c7}, op2 == {0-7}.

AArch32: ID_PFR0, ID_PFR1, ID_DFR0, ID_AFR0, ID_MMFR0, ID_MMFR1, ID_MMFR2, ID_MMFR3, ID_ISAR0, ID_ISAR1, ID_ISAR2, ID_ISAR3, ID_ISAR4, ID_ISAR5, MVFR0, MVFR1, MVFR2, and ID_MMFR4, except that if ID_MMFR4 is implemented as RAZ/WI then it is IMPLEMENTATION DEFINED whether accesses to ID_MMFR4 are trapped.

MRC access to any of the following encodings are also trapped:

- coproc==p15, opc1 == 0, CRn == c0, CRm == {c3-c7}, opc2 == {0,1}.
- coproc==p15, opc1 == 0, CRn == c0, CRm == c3, opc2 == 2.
- coproc==p15, opc1 == 0, CRn == c0, CRm == c5, opc2 == {4,5}.

It is IMPLEMENTATION DEFINED whether this bit traps MRC accesses to the following encodings:

- coproc==p15, opc1 == 0, CRn == c0, CRm == c2, opc2 == 7.
- coproc==p15, opc1 == 0, CRn == c0, CRm == c3, opc2 == {3-7}.
- coproc==p15, opc1 == 0, CRn == c0, CRm == {c4, c6, c7}, opc2 == {2-7}.
- coproc==p15, opc1 == 0, CRn == c0, CRm == c5, opc2 == {2, 3, 6, 7}.

TID3	Meaning
0b0	This control does not cause any instructions to be trapped.
0b1	The specified EL1 read accesses to ID group 3 registers are trapped to EL2, when EL2 is enabled in the current Security state.

When HCR_EL2.TGE is 1, the PE ignores the value of this field for all purposes other than a direct read of this field.

This field resets to an architecturally UNKNOWN value.

TID2, bit [17]

Trap ID group 2. Traps the following register accesses to EL2, when EL2 is enabled in the current Security state:

AArch64:

- EL1 reads of CTR_EL0, CCSIDR_EL1, CCSIDR2_EL1, CLIDR_EL1, and CSSELR_EL1.
- EL0 reads of CTR_EL0, except that if the value of SCTLR_EL1.UCT is 0 then EL0 reads of CTR_EL0 are UNDEFINED and any resulting exception takes precedence over this trap.
- EL1 writes to CSSELR_EL1.

AArch32:

- EL1 reads of the CTR, CCSIDR, CCSIDR2, CLIDR, and CSSELR.
- EL1 writes to the CSSELR.

TID2	Meaning
0b0	This control does not cause any instructions to be trapped.
0b1	The specified EL1 and EL0 accesses to ID group 2 registers are trapped to EL2, when EL2 is enabled in the current Security state.

When ARMv8.1-VHE is implemented, and the value of HCR_EL2.{E2H, TGE} is {1, 1}, this field behaves as 0 for all purposes other than a direct read of the value of this bit.

This field resets to an architecturally UNKNOWN value.

TID1, bit [16]

Trap ID group 1. Traps EL1 reads of the following registers to EL2, when EL2 is enabled in the current Security state:

AArch64: REVIDR_EL1, AIDR_EL1.

AArch32: TCMTR, TLBTR, REVIDR, AIDR.

TID1	Meaning
0b0	This control does not cause any instructions to be trapped.
0b1	The specified EL1 read accesses to ID group 1 registers are trapped to EL2, when EL2 is enabled in the current Security state.

When HCR_EL2.TGE is 1, the PE ignores the value of this field for all purposes other than a direct read of this field.

This field resets to an architecturally UNKNOWN value.

TID0, bit [15]

Trap ID group 0. Traps the following register accesses to EL2:

AArch64: None.

AArch32:

- EL1 reads of the JIDR.
- If the JIDR is RAZ from EL0, EL0 reads of the JIDR.
- EL1 reads of the FPSID.

Note

- It is IMPLEMENTATION DEFINED whether the JIDR is RAZ or UNDEFINED at EL0. If it is UNDEFINED at EL0 then any resulting exception takes precedence over this trap.
- The FPSID is not accessible at EL0 using AArch32.
- Writes to the FPSID are ignored, and not trapped by this control.

TID0	Meaning
0b0	This control does not cause any instructions to be trapped.
0b1	The specified EL1 read accesses to ID group 0 registers are trapped to EL2, when EL2 is enabled in the current Security state.

In an AArch64 only implementation, this bit is RES0.

When ARMv8.1-VHE is implemented, and the value of HCR_EL2.{E2H, TGE} is {1, 1}, this field behaves as 0 for all purposes other than a direct read of the value of this bit.

This field resets to an architecturally UNKNOWN value.

TWE, bit [14]

Traps EL0 and EL1 execution of WFE instructions to EL2, when EL2 is enabled in the current Security state, from both Execution states.

TWE	Meaning
0b0	This control does not cause any instructions to be trapped.
0b1	Any attempt to execute a WFE instruction at EL0 or EL1 is trapped to EL2, when EL2 is enabled in the current Security state, if the instruction would otherwise have caused the PE to enter a low-power state and it is not trapped by SCTLR.nTWE or SCTLR_EL1.nTWE.

In AArch32 state, the attempted execution of a conditional WFE instruction is only trapped if the instruction passes its condition code check.

Note

Since a WFE can complete at any time, even without a Wakeup event, the traps on WFE are not guaranteed to be taken, even if the WFE is executed when there is no Wakeup event. The only guarantee is that if the instruction does not complete in finite time in the absence of a Wakeup event, the trap will be taken.

When ARMv8.1-VHE is implemented, and the value of HCR_EL2.{E2H, TGE} is {1, 1}, this field behaves as 0 for all purposes other than a direct read of the value of this bit.

This field resets to an architecturally UNKNOWN value.

TWI, bit [13]

Traps EL0 and EL1 execution of WFI instructions to EL2, when EL2 is enabled in the current Security state, from both Execution states.

TWI	Meaning
0b0	This control does not cause any instructions to be trapped.
0b1	Any attempt to execute a WFI instruction at EL0 or EL1 is trapped to EL2, when EL2 is enabled in the current Security state, if the instruction would otherwise have caused the PE to enter a low-power state and it is not trapped by SCTLR.nTWI or SCTLR_EL1.nTWI.

In AArch32 state, the attempted execution of a conditional WFI instruction is only trapped if the instruction passes its condition code check.

Note

Since a WFI can complete at any time, even without a Wakeup event, the traps on WFI are not guaranteed to be taken, even if the WFI is executed when there is no Wakeup event. The only guarantee is that if the instruction does not complete in finite time in the absence of a Wakeup event, the trap will be taken.

When ARMv8.1-VHE is implemented, and the value of HCR_EL2.{E2H, TGE} is {1, 1}, this field behaves as 0 for all purposes other than a direct read of the value of this bit.

This field resets to an architecturally UNKNOWN value.

DC, bit [12]

Default Cacheability.

DC	Meaning
0b0	This control has no effect on the EL1&0 translation regime.
0b1	In both Security states: • When EL1 is using AArch64, the PE behaves as if the value of the SCTLR_EL1.M field is 0 for all purposes other than returning the value of a direct read of SCTLR_EL1. • When EL1 is using AArch32, the PE behaves as if the value of the SCTLR.M field is 0 for all purposes other than returning the value of a direct read of SCTLR. • The PE behaves as if the value of the HCR_EL2.VM field is 1 for all purposes other than returning the value of a direct read of HCR_EL2. • The memory type produced by stage 1 of the EL1&0 translation regime is Normal Non-Shareable, Inner Write-Back Read-Allocate Write-Allocate, Outer Write-Back Read-Allocate Write-Allocate.

This field has no effect on the EL2, EL2&0, and EL3 translation regimes.

This field is permitted to be cached in a TLB.

When ARMv8.1-VHE is implemented, and the value of HCR_EL2.{E2H, TGE} is {1, 1}, this field behaves as 0 for all purposes other than a direct read of the value of this field.

This field resets to an architecturally UNKNOWN value.

BSU, bits [11:10]

Barrier Shareability upgrade. This field determines the minimum shareability domain that is applied to any barrier instruction executed from EL1 or EL0:

BSU	Meaning
0b00	No effect.
0b01	Inner Shareable.
0b10	Outer Shareable.
0b11	Full system.

This value is combined with the specified level of the barrier held in its instruction, using the same principles as combining the shareability attributes from two stages of address translation.

When ARMv8.1-VHE is implemented, and the value of HCR_EL2.{E2H, TGE} is {1, 1}, this field behaves as 0b00 for all purposes other than a direct read of the value of this bit.

This field resets to an architecturally UNKNOWN value.

FB, bit [9]

Force broadcast. Causes the following instructions to be broadcast within the Inner Shareable domain when executed from EL1:

AArch32: BPIALL, TLBIALL, TLBIMVA, TLBIASID, DTLBIALL, DTLBIMVA, DTLBIASID, ITLBIALL, ITLBIMVA, ITLBIASID, TLBIMVAA, ICIALLU, TLBIMVAL, TLBIMVAAL.

AArch64: TLBI VMALLE1, TLBI VAE1, TLBI ASIDE1, TLBI VAAE1, TLBI VALE1, TLBI VAALE1, IC IALLU, TLBI RVAE1, TLBI RVAAE1, TLBI RVALE1, TLBI RVAALE1.

FB	Meaning
0b0	This field has no effect on the operation of the specified instructions.
0b1	When one of the specified instruction is executed at EL1, the instruction is broadcast within the Inner Shareable shareability domain.

When HCR_EL2.TGE is 1, the PE ignores the value of this field for all purposes other than a direct read of this field.

This field resets to an architecturally UNKNOWN value.

VSE, bit [8]

Virtual SError interrupt.

VSE	Meaning
0b0	This mechanism is not making a virtual SError interrupt pending.
0b1	A virtual SError interrupt is pending because of this mechanism.

The virtual SError interrupt is only enabled when the value of HCR_EL2.{TGE, AMO} is {0, 1}.

This field resets to an architecturally UNKNOWN value.

VI, bit [7]

Virtual IRQ Interrupt.

VI	Meaning
0b0	This mechanism is not making a virtual IRQ pending.
0b1	A virtual IRQ is pending because of this mechanism.

The virtual IRQ is enabled only when the value of HCR_EL2.{TGE, IMO} is {0, 1}.

This field resets to an architecturally UNKNOWN value.

VF, bit [6]

Virtual FIQ Interrupt.

VF	Meaning
0b0	This mechanism is not making a virtual FIQ pending.
0b1	A virtual FIQ is pending because of this mechanism.

The virtual FIQ is enabled only when the value of HCR_EL2.{TGE, FMO} is {0, 1}.

This field resets to an architecturally UNKNOWN value.

AMO, bit [5]

Physical SError interrupt routing.

AMO	Meaning
0b0	When executing at Exception levels below EL2, and EL2 is enabled in the current Security state: • Physical SError interrupts are not taken to EL2. • When the value of HCR_EL2.TGE is 0, if the PE is executing at EL2 using AArch64, physical SError interrupts are not taken unless they are routed to EL3 by the SCR_EL3.EA bit. • Virtual SError interrupts are disabled.
0b1	When executing at any Exception level, and EL2 is enabled in the current Security state: • Physical SError interrupts are taken to EL2, unless they are routed to EL3. • When the value of HCR_EL2.TGE is 0, then virtual SError interrupts are enabled.

If EL2 is enabled in the current Security state and the value of HCR_EL2.TGE is 1:

- Regardless of the value of the AMO bit physical asynchronous External aborts and SError interrupts target EL2 unless they are routed to EL3.
- When ARMv8.1-VHE is not implemented, or if HCR_EL2.E2H is 0, this field behaves as 1 for all purposes other than a direct read of the value of this bit.
- When ARMv8.1-VHE is implemented and HCR_EL2.E2H is 1, this field behaves as 0 for all purposes other than a direct read of the value of this bit.

For more information, see 'Asynchronous exception routing' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section D1 (The AArch64 System Level Programmers' Model).

This field resets to an architecturally UNKNOWN value.

IMO, bit [4]

Physical IRQ Routing.

IMO	Meaning
0b0	When executing at Exception levels below EL2, and EL2 is enabled in the current Security state: • Physical IRQ interrupts are not taken to EL2. • When the value of HCR_EL2.TGE is 0, if the PE is executing at EL2 using AArch64, physical IRQ interrupts are not taken unless they are routed to EL3 by the SCR_EL3.IRQ bit. • Virtual IRQ interrupts are disabled.
0b1	When executing at any Exception level, and EL2 is enabled in the current Security state: • Physical IRQ interrupts are taken to EL2, unless they are routed to EL3. • When the value of HCR_EL2.TGE is 0, then Virtual IRQ interrupts are enabled.

If EL2 is enabled in the current Security state, and the value of HCR_EL2.TGE is 1:

- Regardless of the value of the IMO bit, physical IRQ Interrupts target EL2 unless they are routed to EL3.
- When ARMv8.1-VHE is not implemented, or if HCR_EL2.E2H is 0, this field behaves as 1 for all purposes other than a direct read of the value of this bit.
- When ARMv8.1-VHE is implemented and HCR_EL2.E2H is 1, this field behaves as 0 for all purposes other than a direct read of the value of this bit.

For more information, see 'Asynchronous exception routing' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section D1.

This field resets to an architecturally UNKNOWN value.

FMO, bit [3]

Physical FIQ Routing.

FMO	Meaning
0b0	When executing at Exception levels below EL2, and EL2 is enabled in the current Security state: • Physical FIQ interrupts are not taken to EL2. • When the value of HCR_EL2.TGE is 0, if the PE is executing at EL2 using AArch64, physical FIQ interrupts are not taken unless they are routed to EL3 by the SCR_EL3.FIQ bit. • Virtual FIQ interrupts are disabled.
0b1	When executing at any Exception level, and EL2 is enabled in the current Security state: • Physical FIQ interrupts are taken to EL2, unless they are routed to EL3. • When HCR_EL2.TGE is 0, then Virtual FIQ interrupts are enabled.

If EL2 is enabled in the current Security state and the value of HCR_EL2.TGE is 1:

- Regardless of the value of the FMO bit, physical FIQ Interrupts target EL2 unless they are routed to EL3.
- When ARMv8.1-VHE is not implemented, or if HCR_EL2.E2H is 0, this field behaves as 1 for all purposes other than a direct read of the value of this bit.
- When ARMv8.1-VHE is implemented and HCR_EL2.E2H is 1, this field behaves as 0 for all purposes other than a direct read of the value of this bit.

For more information, see 'Asynchronous exception routing' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section D1.

This field resets to an architecturally UNKNOWN value.

PTW, bit [2]

Protected Table Walk. In the EL1&0 translation regime, a translation table access made as part of a stage 1 translation table walk is subject to a stage 2 translation. The combining of the memory type attributes from the two stages of translation means the access might be made to a type of Device memory. If this occurs, then the value of this bit determines the behavior:

PTW	Meaning
0b0	The translation table walk occurs as if it is to Normal Non-cacheable memory. This means it can be made speculatively.
0b1	The memory access generates a stage 2 Permission fault.

This field is permitted to be cached in a TLB.

When HCR_EL2.TGE is 1, the PE ignores the value of this field for all purposes other than a direct read of this field.

This field resets to an architecturally UNKNOWN value.

SWIO, bit [1]

Set/Way Invalidation Override. Causes EL1 execution of the data cache invalidate by set/way instructions to perform a data cache clean and invalidate by set/way:

SWIO	Meaning
0b0	This control has no effect on the operation of data cache invalidate by set/way instructions.
0b1	Data cache invalidate by set/way instructions perform a data cache clean and invalidate by set/way.

When the value of this bit is 1:

AArch32: DCISW performs the same invalidation as a DCCISW instruction.

AArch64: DC ISW performs the same invalidation as a DC CISW instruction.

This bit can be implemented as RES1.

When HCR_EL2.TGE is 1, the PE ignores the value of this field for all purposes other than a direct read of this field.

This field resets to an architecturally UNKNOWN value.

VM, bit [0]

Virtualization enable. Enables stage 2 address translation for the EL1&0 translation regime, when EL2 is enabled in the current Security state.

VM	Meaning
0b0	EL1&0 stage 2 address translation disabled.
0b1	EL1&0 stage 2 address translation enabled.

When the value of this bit is 1, data cache invalidate instructions executed at EL1 perform a data cache clean and invalidate. For the invalidate by set/way instruction this behavior applies regardless of the value of the HCR_EL2.SWIO bit.

This bit is permitted to be cached in a TLB.

When ARMv8.1-VHE is implemented, and the value of HCR_EL2.{E2H, TGE} is {1, 1}, this field behaves as 0 for all purposes other than a direct read of the value of this bit.

This field resets to an architecturally UNKNOWN value.

Accessing the HCR_EL2

Accesses to this register use the following encodings:

```
MRS <Xt>, HCR_EL2
```

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b11	0b100	0b0001	0b0001	0b000
~~0b11~~	~~0b0001~~	~~0b100~~	~~0b000~~	~~0b0001~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
        return NVMem[0x078];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    return HCR_EL2;
elsif PSTATE.EL == EL3 then
    return HCR_EL2;
```

```
MSR HCR_EL2, <Xt>
```

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b11	0b100	0b0001	0b0001	0b000
~~0b11~~	~~0b0001~~	~~0b100~~	~~0b000~~	~~0b0001~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
        NVMem[0x078] = X[t];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    HCR_EL2 = X[t];
elsif PSTATE.EL == EL3 then
    HCR_EL2 = X[t];
```

27~~13~~/03~~12~~/2019~~2018~~ 21~~16~~:59~~42~~; e5e4db499bf9867a4b93324c4dbac985d3da93766~~379d01e197f1d40720d32d0f84e419e9187e009~~

(old) htmldiff from- (new)

(old) htmldiff from- (new)

ICH_AP0R<n>_EL2, Interrupt Controller Hyp Active Priorities Group 0 Registers, n = 0 - 3

The ICH_AP0R<n>_EL2 characteristics are:

Purpose

Provides information about Group 0 virtual active priorities for EL2.

Configuration

AArch64 System register ICH_AP0R<n>_EL2 bits [31:0] are architecturally mapped to AArch32 System register ICH_AP0R<n>[31:0] .

If EL2 is not implemented, this register is RES0 from EL3.

This register has no effect if EL2 is not enabled in the current Security state.

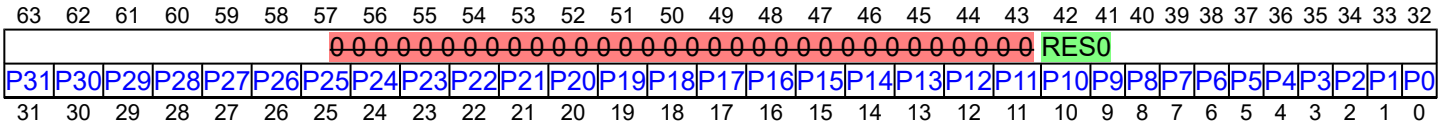
Some or all RW fields of this register have defined reset values. These apply only if the PE resets into an Exception level that is using AArch64. Otherwise, RW fields in this register reset to architecturally UNKNOWN values.

Attributes

ICH_AP0R<n>_EL2 is a 64-bit register.

Field descriptions

The ICH_AP0R<n>_EL2 bit assignments are:



Bits [63:32]

Reserved, RES0.

P<x>, bit [x], for x = 0 to 31

Provides the access to the virtual active priorities for Group 0 interrupts. Possible values of each bit are:

P<x>	Meaning
0b0	There is no Group 0 interrupt active with this priority level, or all active Group 0 interrupts with this priority level have undergone priority-drop.
0b1	There is a Group 0 interrupt active with this priority level which has not undergone priority drop.

The correspondence between priority levels and bits depends on the number of bits of priority that are implemented.

If 5 bits of preemption are implemented (bits [7:3] of priority), then there are 32 preemption levels, and the active state of these preemption levels are held in ICH_AP0R0_EL2 in the bits corresponding to Priority[7:3].

If 6 bits of preemption are implemented (bits [7:2] of priority), then there are 64 preemption levels, and:

- The active state of preemption levels 0 - 124 are held in ICH_AP0R0_EL2 in the bits corresponding to 0:Priority[6:2].
- The active state of preemption levels 128 - 252 are held in ICH_AP0R1_EL2 in the bits corresponding to 1:Priority[6:2].

If 7 bits of preemption are implemented (bits [7:1] of priority), then there are 128 preemption levels, and:

- The active state of preemption levels 0 - 62 are held in ICH_AP0R0_EL2 in the bits corresponding to 00:Priority[5:1].
- The active state of preemption levels 64 - 126 are held in ICH_AP0R1_EL2 in the bits corresponding to 01:Priority[5:1].
- The active state of preemption levels 128 - 190 are held in ICH_AP0R2_EL2 in the bits corresponding to 10:Priority[5:1].
- The active state of preemption levels 192 - 254 are held in ICH_AP0R3_EL2 in the bits corresponding to 11:Priority[5:1].

Note

Having the bit corresponding to a priority set to 1 in both ICH_AP0R<n>_EL2 and ICH_AP1R<n>_EL2 might result in UNPREDICTABLE behavior of the interrupt prioritization system for virtual interrupts.

This field resets to `0`.

Software must ensure that ICH_AP0R<n>_EL2 is 0 for legacy VMs otherwise behaviour is UNPREDICTABLE. For more information about support for legacy VMs, see Support for legacy operation of VMs.

The active priorities for Group 0 and Group 1 interrupts for legacy VMs are held in ICH_AP1R<n>_EL2 and reads and writes to GICV_APR access ICH_AP1R<n>_EL2. This means that ICH_AP0R<n>_EL2 is inaccessible to legacy VMs.

Accessing the ICH_AP0R<n>_EL2

ICH_AP0R1_EL2 is only implemented in implementations that support 6 or more bits of preemption. ICH_AP0R2_EL2 and ICH_AP0R3_EL2 are only implemented in implementations that support 7 bits of preemption. Unimplemented registers are UNDEFINED.

Note

The number of preemption bits is reported by ICH_VTR_EL2.PREbits.

Writing to these registers with any value other than the last read value of the register (or `0x00000000` for a newly set up virtual machine) can result in UNPREDICTABLE behavior of the virtual interrupt prioritization system allowing either:

- Virtual interrupts that should preempt execution to not preempt execution.
- Interrupts that should not preempt execution to preempt execution at EL1 or EL0.

Writing to the active priority registers in any order other than the following order will result in UNPREDICTABLE behavior:

- ICH_AP0R<n>_EL2.
- ICH_AP1R<n>_EL2.

Having the bit corresponding to a priority set in both ICH_AP0R<n>_EL2 and ICH_AP1R<n>_EL2 can result in UNPREDICTABLE behavior of the interrupt prioritization system for virtual interrupts.

ICH_AP0R1_EL2 is only implemented in implementations that support 6 or more bits of preemption. ICH_AP0R2_EL2 and ICH_AP0R3_EL2 are only implemented in implementations that support 7 bits of preemption. Unimplemented registers are UNDEFINED.

Note

The number of bits of preemption is indicated by ICH_VTR_EL2.PREbits

Writing to these registers with any value other than the last read value of the register (or `0x00000000` for a newly set up virtual machine) can result in UNPREDICTABLE behavior of the virtual interrupt prioritization system allowing either:

Writing to the active priority registers in any order other than the following order will result in UNPREDICTABLE behavior:

Having the bit corresponding to a priority set in both ICH_AP0R<n>_EL2 and ICH_AP1R<n>_EL2 can result in UNPREDICTABLE behavior of the interrupt prioritization system for virtual interrupts.

Accesses to this register use the following encodings:

```
MRS <Xt>, ICH_AP0R<n>_EL2
```

op0	op1	CRn	CRm	op2

~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b11	0b100	0b1100	0b1000	0b0[n:1:0]
~~0b11~~	~~0b1100~~	~~0b100~~	~~0b0[n:1:0]~~	~~0b1000~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
        return NVMem[0x480+8*UInt(op2<1:0>)];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ICC_SRE_EL2.SRE == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        return ICH_AP0R_EL2[UInt(op2<1:0>)];
elsif PSTATE.EL == EL3 then
    if ICC_SRE_EL3.SRE == '0' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return ICH_AP0R_EL2[UInt(op2<1:0>)];
```

```
MSR ICH_AP0R<n>_EL2, <Xt>
```

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b11	0b100	0b1100	0b1000	0b0[n:1:0]
~~0b11~~	~~0b1100~~	~~0b100~~	~~0b0[n:1:0]~~	~~0b1000~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
        NVMem[0x480+8*UInt(op2<1:0>)] = X[t];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ICC_SRE_EL2.SRE == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        ICH_AP0R_EL2[UInt(op2<1:0>)] = X[t];
elsif PSTATE.EL == EL3 then
    if ICC_SRE_EL3.SRE == '0' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        ICH_AP0R_EL2[UInt(op2<1:0>)] = X[t];
```

27~~13~~/03~~12~~/2019~~2018~~ 21~~16~~:59~~42~~; e5e4db499bf9867a4b93324c4dbac985d3da93766~~379d01e197f1d40720d32d0f84c419e9187e009~~

(old) htmldiff from- (new)

ICH_LR<n>_EL2, Interrupt Controller List Registers, n = 0 - 15

The ICH_LR<n>_EL2 characteristics are:

Purpose

Provides interrupt context information for the virtual CPU interface.

Configuration

AArch64 System register ICH_LR<n>_EL2 bits [31:0] are architecturally mapped to AArch32 System register ICH_LR<n>[31:0] .

AArch64 System register ICH_LR<n>_EL2 bits [63:32] are architecturally mapped to AArch32 System register ICH_LRC<n>[31:0] .

If EL2 is not implemented, this register is RES0 from EL3.

This register has no effect if EL2 is not enabled in the current Security state.

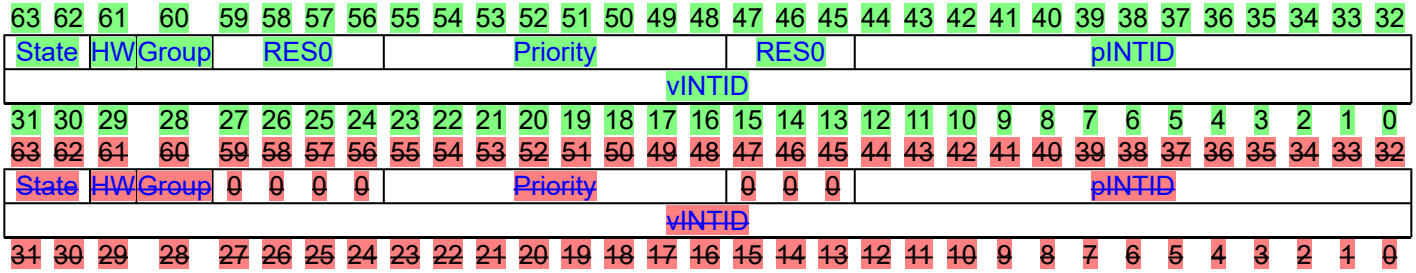
RW fields in this register reset to architecturally UNKNOWN values.

Attributes

ICH_LR<n>_EL2 is a 64-bit register.

Field descriptions

The ICH_LR<n>_EL2 bit assignments are:



State, bits [63:62]

The state of the interrupt:

State	Meaning
0b00	Invalid (Inactive).
0b01	Pending.
0b10	Active.
0b11	Pending and active.

The GIC updates these state bits as virtual interrupts proceed through the interrupt life cycle. Entries in the invalid state are ignored, except for the purpose of generating virtual maintenance interrupts.

For hardware interrupts, the pending and active state is held in the physical Distributor rather than the virtual CPU interface. A hypervisor must only use the pending and active state for software originated interrupts, which are typically associated with virtual devices, or SGIs.

This field resets to an architecturally UNKNOWN value.

HW, bit [61]

Indicates whether this virtual interrupt maps directly to a hardware interrupt, meaning that it corresponds to a physical interrupt. Deactivation of the virtual interrupt also causes the deactivation of the physical interrupt with the ID that the pINTID field indicates.

HW	Meaning
0b0	The interrupt is triggered entirely by software. No notification is sent to the Distributor when the virtual interrupt is deactivated.
0b1	The interrupt maps directly to a hardware interrupt. A deactivate interrupt request is sent to the Distributor when the virtual interrupt is deactivated, using the pINTID field from this register to indicate the physical interrupt ID. If ICH_VMCR_EL2.VEOIM is 0, this request corresponds to a write to ICC_EOIR0_EL1 or ICC_EOIR1_EL1. Otherwise, it corresponds to a write to ICC_DIR_EL1.

This field resets to an architecturally UNKNOWN value.

Group, bit [60]

Indicates the group for this virtual interrupt.

Group	Meaning
0b0	This is a Group 0 virtual interrupt. ICH_VMCR_EL2.VFIQEn determines whether it is signaled as a virtual IRQ or as a virtual FIQ, and ICH_VMCR_EL2.VENG0 enables signaling of this interrupt to the virtual machine.
0b1	This is a Group 1 virtual interrupt, signaled as a virtual IRQ. ICH_VMCR_EL2.VENG1 enables the signalling of this interrupt to the virtual machine. If ICH_VMCR_EL2.VCBPR is 0, then ICC_BPR1_EL1 determines if a pending Group 1 interrupt has sufficient priority to preempt current execution. Otherwise, ICH_LR<n>_EL2 determines preemption.

This field resets to an architecturally UNKNOWN value.

Bits [59:56]

Reserved, RES0.

Priority, bits [55:48]

The priority of this interrupt.

It is IMPLEMENTATION DEFINED how many bits of priority are implemented, though at least five bits must be implemented. Unimplemented bits are RES0 and start from bit[48] up to bit[50]. The number of implemented bits can be discovered from ICH_VTR_EL2.PRIbits.

This field resets to an architecturally UNKNOWN value.

Bits [47:45]

Reserved, RES0.

pINTID, bits [44:32]

Physical INTID, for hardware interrupts.

When ICH_LR<n>_EL2.HW is 0 (there is no corresponding physical interrupt), this field has the following meaning:

- Bits[44:42] : RES0.
- Bit[41] : EOI. If this bit is 1, then when the interrupt identified by vINTID is deactivated, a maintenance interrupt is asserted.
- Bits[40:32] : RES0.

When ICH_LR<n>_EL2.HW is 1 (there is a corresponding physical interrupt):

- This field indicates the physical INTID. This field is only required to implement enough bits to hold a valid value for the implemented INTID size. Any unused higher order bits are RES0.

- When ICC_CTLR_EL1.ExtRange is 0, then bits[44:42] of this field are RES0.
- If the value of pINTID is not a vald INTID, behavior is UNPREDICTABLE. If the value of pINTID indicates a PPI, this field applies to the PPI associated with this same physical PE ID as the virtual CPU interface requesting the deactivation.

A hardware physical identifier is only required in List Registers for interrupts that require deactivation. This means only 13 bits of Physical INTID are required, regardless of the number specified by ICC_CTLR_EL1.IDbits.

This field resets to an architecturally UNKNOWN value.

vINTID, bits [31:0]

Virtual INTID of the interrupt.

If the value of vINTID is 1020-1023 and ICH_LR<n>_EL2.State!=0b00 (Inactive), behavior is UNPREDICTABLE.

Behavior is UNPREDICTABLE if two or more List Registers specify the same vINTID when:

- ICH_LR<n>_EL2.State == 0b01.
- ICH_LR<n>_EL2.State == 0b10.
- ICH_LR<n>_EL2.State == 0b11.

It is IMPLEMENTATION DEFINED how many bits are implemented, though at least 16 bits must be implemented. Unimplemented bits are RES0. The number of implemented bits can be discovered from ICH_VTR_EL2.IDbits.

When ICC_SRE_EL1.SRE == 0, specifying a vINTID in the LPI range is UNPREDICTABLE

Note

When a VM is using memory-mapped access to the GIC, software must ensure that the correct source PE ID is provided in bits[12:10].

This field resets to an architecturally UNKNOWN value.

Accessing the ICH_LR<n>_EL2

Accesses to this register use the following encodings:

```
MRS <Xt>, ICH_LR<n>_EL2
```

op0	op1	CRn	CRm	op2
~~CRn~~	~~op0~~	~~op1~~	~~op2~~	~~CRm~~
0b11	0b100	0b1100	0b110[n:3]	0b[n:2:0]
~~0b1100~~	~~0b11~~	~~0b100~~	~~0b[n:2:0]~~	~~0b110[n:3]~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
        return NVMem[0x400+8*UInt(CRm<0>:op2<2:0>)];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ICC_SRE_EL2.SRE == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        return ICH_LR_EL2[UInt(CRm<0>:op2<2:0>)];
elsif PSTATE.EL == EL3 then
    if ICC_SRE_EL3.SRE == '0' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return ICH_LR_EL2[UInt(CRm<0>:op2<2:0>)];
```

```
MSR ICH_LR<n>_EL2, <Xt>
```

op0	op1	CRn	CRm	op2
~~CRn~~	~~op0~~	~~op1~~	~~op2~~	~~CRm~~
0b11	0b100	0b1100	0b110[n:3]	0b[n:2:0]
~~0b1100~~	~~0b11~~	~~0b100~~	~~0b[n:2:0]~~	~~0b110[n:3]~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
        NVMem[0x400+8*UInt(CRm<0>:op2<2:0>)] = X[t];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ICC_SRE_EL2.SRE == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        ICH_LR_EL2[UInt(CRm<0>:op2<2:0>)] = X[t];
elsif PSTATE.EL == EL3 then
    if ICC_SRE_EL3.SRE == '0' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        ICH_LR_EL2[UInt(CRm<0>:op2<2:0>)] = X[t];
```

27~~13~~/03~~12~~/2019~~2018~~ 21~~16~~:59~~42~~; e5e4db499bf9867a4b93324c4dbac985d3da93766~~379d01e197f1d40720d32d0f84e419e9187e009~~

(old) htmldiff from- (new)

ID_AA64DFR0_EL1, AArch64 Debug Feature Register 0

The ID_AA64DFR0_EL1 characteristics are:

Purpose

Provides top level information about the debug system in AArch64 state.

For general information about the interpretation of the ID registers see 'Principles of the ID scheme for fields in ID registers' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section D10.4.1.

Configuration

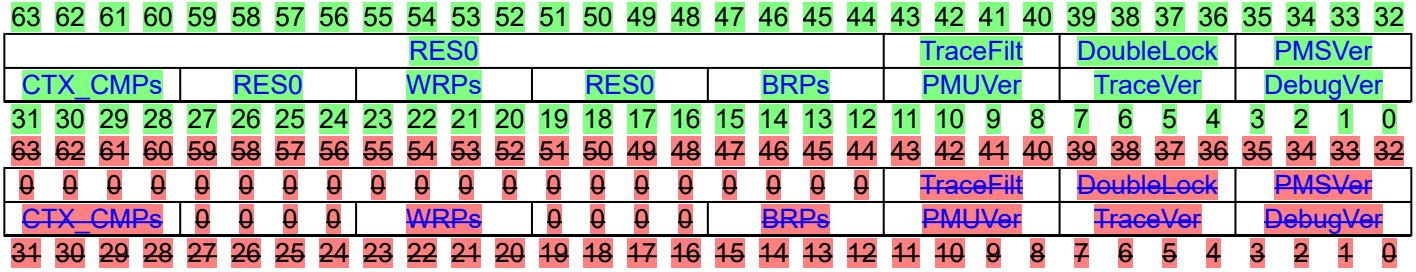
The external register EDDFR gives information from this register.

Attributes

ID_AA64DFR0_EL1 is a 64-bit register.

Field descriptions

The ID_AA64DFR0_EL1 bit assignments are:



Bits [63:44]

Reserved, RES0.

TraceFilt, bits [43:40]

From Armv8.4:

Armv8.4 Self-hosted Trace Extension version. Defined values are:

TraceFilt	Meaning
0b0000	Armv8.4 Self-hosted Trace Extension not implemented.
0b0001	Armv8.4 Self-hosted Trace Extension implemented.

All other values are reserved.

Otherwise:

Reserved, RES0.

DoubleLock, bits [39:36]

OS Double Lock implemented. Defined values are:

DoubleLock	Meaning
0b0000	OS Double Lock not implemented. OSDLR_EL1 is readRAZ/write.WI.
0b1111 0b0001	OS Double Lock not implemented. OSDLR_EL1 is RAZread/WI.write.

ARMv8.0-DoubleLock implements the functionality added by the value 0b0000.

All other values are reserved.

PMSVer, bits [35:32]

From Armv8.2:

Statistical Profiling Extension version. DefinedThe defined values of this field are:

PMSVer	Meaning
0b0000	Statistical Profiling Extension not implemented.
0b0001	Statistical Profiling Extension implemented.
0b0010	As 0b0001 and also includes support for: • TheSupport for the Event packet Alignment flag. • If SVE is implemented, support for the Scalable Vector extensions to Statistical Profiling. SVE is implemented, the Scalable Vector extensions to Statistical Profiling.

SPE SVE implements the functionality added by the value 0b0001.

ARMv8.3-SPE implements the functionality added by the value 0b0010. If ARMv8.3-SPE is implemented, then ID_AA64DFR0_EL1.PMSVer is not permitted to read as 0b0000 or 0b0001.

All other values are reserved.

Otherwise:

Reserved, RES0.

CTX_CMPs, bits [31:28]

Number of breakpoints that are context-aware, minus 1. These are the highest numbered breakpoints.

Bits [27:24]

Reserved, RES0.

WRPs, bits [23:20]

Number of watchpoints, minus 1. The value of 0b0000 is reserved.

Bits [19:16]

Reserved, RES0.

BRPs, bits [15:12]

Number of breakpoints, minus 1. The value of 0b0000 is reserved.

PMUVer, bits [11:8]

Performance Monitors Extension version.

This field does not follow the standard ID scheme, but uses the Alternative ID scheme described in 'Alternative ID scheme used for the Performance Monitors Extension version' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, ~~section D10.1.4~~.

Defined values are:

PMUVer	Meaning
0b0000	Performance Monitors Extension not implemented.
0b0001	Performance Monitors Extension implemented, PMUv3.
0b0100	PMUv3 for Armv8.1. As 0b0001, and also includes support for: • Extended 16-bit PMEVTYPER<n>_EL0.evtCount field. • If EL2 is implemented, the MDCR_EL2.HPMD control bit.
0b0101	PMUv3 for Armv8.4. As 0b0100 and also includes support for the PMMIR_EL1 register.
0b0110	PMUv3 for Armv8.5. As 0b0101 and also includes support for: • 64-bit event counters. • If EL2 is implemented, the MDCR_EL2.HCCD control bit. • If EL3 is implemented, the MDCR_EL3.SCCD control bit.
0b1111	IMPLEMENTATION DEFINED form of performance monitors supported, PMUv3 not supported. Arm does not recommend this value in new implementations.

ARMv8.1-PMU implements the functionality added by the value 0b0100.

ARMv8.4-PMU implements the functionality added by the value 0b0101.

ARMv8.5-PMU implements the functionality added by the value 0b0110.

All other values are reserved.

From Armv8.1, the value 0b0001 is not permitted.

From Armv8.4, the value 0b0100 is not permitted.

From Armv8.5, the value 0b0101 is not permitted.

TraceVer, bits [7:4]

Trace support. Indicates whether System register interface to a PE trace unit is implemented. Defined values are:

TraceVer	Meaning
0b0000	PE trace unit System registers not implemented.
0b0001	PE trace unit System registers implemented.

All other values are reserved.

A value of 0b0000 only indicates that no System register interface to a PE trace unit is implemented. A PE trace unit might nevertheless be implemented without a System register interface.

See the ETM Architecture Specification for more information.

DebugVer, bits [3:0]

Debug architecture version. Indicates presence of Armv8 debug architecture. Defined values are:

DebugVer	Meaning
0b0110	Armv8 debug architecture.
0b0111	Armv8 debug architecture with Virtualization Host Extensions.
0b1000	Armv8.2 debug architecture
0b1001	Armv8.4 debug architecture

All other values are reserved.

ARMv8.2-Debug adds the functionality indicated by the value 0b1000.

- If ARMv8.1-VHE is not implemented the only permitted value is 0b0110.
- In an Armv8.0 implementation the value 0b1000 is not permitted.

Accessing the ID_AA64DFR0_EL1

Accesses to this register use the following encodings:

```
MRS <Xt>, ID_AA64DFR0_EL1
```

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b11	0b000	0b0000	0b0101	0b000
~~0b11~~	~~0b0000~~	~~0b000~~	~~0b000~~	~~0b0101~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TID3 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        return ID_AA64DFR0_EL1;
elsif PSTATE.EL == EL2 then
    return ID_AA64DFR0_EL1;
elsif PSTATE.EL == EL3 then
    return ID_AA64DFR0_EL1;
```

27~~13~~/03~~12~~/2019~~2018~~ 21~~16~~:59~~42~~; e5e4db499bf9867a4b93324c4dbac985d3da93766~~379d01e197f1d40720d32d0f84e419e9187e009~~

(old) htmldiff from- (new)

ID_AA64ISAR1_EL1, AArch64 Instruction Set Attribute Register 1

The ID_AA64ISAR1_EL1 characteristics are:

Purpose

Provides information about the features and instructions implemented in AArch64 state.

For general information about the interpretation of the ID registers see 'Principles of the ID scheme for fields in ID registers' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section D10.4.1.

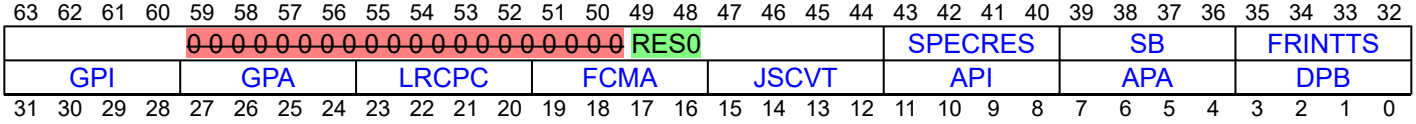
Configuration

Attributes

ID_AA64ISAR1_EL1 is a 64-bit register.

Field descriptions

The ID_AA64ISAR1_EL1 bit assignments are:



Bits [63:44]

Reserved, RES0.

SPECRES, bits [43:40]

Speculation invalidation instruction support in AArch64 state. Defined values are:

SPECRES	Meaning
0b0000	CFP RCTX, DVP RCTX, and CPP RCTX instructions are not implemented.
0b0001	CFP RCTX, DVP RCTX, and CPP RCTX instructions are implemented.

All other values are reserved.

From Armv8.5, the only permitted value is 0b0001.

SB, bits [39:36]

SB instruction support in AArch64 state. Defined values are:

SB	Meaning
0b0000	SB instruction is not implemented.
0b0001	SB instruction is implemented.

All other values are reserved.

From Armv8.5, the only permitted value is 0b0001.

FRINTTS, bits [35:32]

Indicates whether FRINT32Z, FRINT32X, FRINT64Z, and FRINT64X instructions are implemented. Defined values are:

FRINTTS	Meaning
0b0000	FRINT32Z, FRINT32X, FRINT64Z, and FRINT64X instructions are not implemented.
0b0001	FRINT32Z, FRINT32X, FRINT64Z, and FRINT64X instructions are implemented.

All other values are reserved.

From Armv8.5, the only permitted value is 0b0001.

GPI, bits [31:28]

From Armv8.3:

Indicates whether an IMPLEMENTATION DEFINED algorithm is implemented in the PE for generic code authentication, in AArch64 state. Defined values are:

GPI	Meaning
0b0000	Generic Authentication using an IMPLEMENTATION DEFINED algorithm is not implemented.
0b0001	Generic Authentication using an IMPLEMENTATION DEFINED algorithm is implemented. This involves the PACGA instruction.

All other values are reserved.

From Armv8.3, the permitted values are 0b0000 and 0b0001.

If the value of ID_AA64ISAR1_EL1.GPA is non-zero, this field must have the value 0b0000.

Otherwise:

Reserved, RES0.

GPA, bits [27:24]

From Armv8.3:

Indicates whether QARMA or Architected algorithm is implemented in the PE for generic code authentication, in AArch64 state. Defined values are:

GPA	Meaning
0b0000	Generic Authentication using an Architected algorithm is not implemented.
0b0001	Generic Authentication using the QARMA algorithm is implemented. This involves the PACGA instruction.

All other values are reserved.

From Armv8.3, the permitted values are 0b0000 and 0b0001.

If the value of ID_AA64ISAR1_EL1.GPI is non-zero, this field must have the value 0b0000.

Otherwise:

Reserved, RES0.

LRCPC, bits [23:20]

From Armv8.4:

Indicates support for weaker release consistency, RCpc based model. Defined values are:

LRCPC	Meaning
0b0000	The LDAPUR*, STLUR*, and LDAPR* instructions are not implemented.
0b0001	The LDAPR* instructions are implemented.
0b0010	The LDAPUR*, STLUR*, and LDAPR* instructions are implemented.

In Armv8.0, Armv8.1, and Armv8.2, the only permitted value is 0b0000.

In Armv8.3, the only permitted value is 0b0001. ARMv8.3-RCPC implements the functionality identified by the value 0b0001.

From Armv8.4, the only permitted value is 0b0010. ARMv8.4-RCPC implements the functionality identified by the value 0b0010.

All other values are reserved.

From Armv8.3:

Indicates support for weaker release consistency, RCpc based model. Defined values are:

LRCPC	Meaning
0b0000	The LDAPRB, LDAPRH and LDAPR instructions are not implemented.
0b0001	The LDAPRB, LDAPRH and LDAPR instructions are implemented.

All other values are reserved.

ARMv8.3-RCPC implements the functionality identified by the value 0b0001.

In Armv8.0, Armv8.1, and Armv8.2, the only permitted value is 0b0000.

In Armv8.3, the only permitted value is 0b0001.

Otherwise:

Reserved, RES0.

FCMA, bits [19:16]

From Armv8.3:

Indicates support for complex number addition and multiplication, where numbers are stored in vectors. Defined values are:

FCMA	Meaning
0b0000	The FCMLA and FCADD instructions are not implemented.
0b0001	The FCMLA and FCADD instructions are implemented.

All other values are reserved.

ARMv8.3-CompNum implements the functionality identified by the value 0b0001.

In Armv8.0, Armv8.1, and Armv8.2, the only permitted value is 0b0000.

From Armv8.3, the only permitted value is 0b0001.

Otherwise:

Reserved, RES0.

JSCVT, bits [15:12]

From Armv8.3:

Indicates support for javascript conversion from double precision floating point values to integers in AArch64 state. Defined values are:

JSCVT	Meaning
0b0000	The FJCVTZS instruction is not implemented.
0b0001	The FJCVTZS instruction is implemented.

All other values are reserved.

ARMv8.3.JSConv implements the functionality identified by 0b0001.

In Armv8.0, Armv8.1, and Armv8.2, the only permitted value is 0b0000.

From Armv8.3, the only permitted value is 0b0001.

Otherwise:

Reserved, RES0.

API, bits [11:8]

From Armv8.3:

Indicates whether an IMPLEMENTATION DEFINED algorithm is implemented in the PE for address authentication, in AArch64 state. Defined values are:

API	Meaning
0b0000	Address Authentication using an IMPLEMENTATION DEFINED algorithm is not implemented.
0b0001	Address Authentication using an IMPLEMENTATION DEFINED algorithm is implemented. This involves all Pointer Authentication instructions other than the PACGA instruction.
0b0010	Address Authentication using an IMPLEMENTATION DEFINED algorithm is implemented, with the EnhancedPAC() function returning TRUE. This applies to all Pointer Authentication instructions other than the PACGA instruction.

All other values are reserved.

From Armv8.3, the permitted values are 0b0000 and 0b0001.

If the value of ID_AA64ISAR1_EL1.APA is non-zero, this field must have the value 0b0000.

Otherwise:

Reserved, RES0.

APA, bits [7:4]

From Armv8.3:

Indicates whether QARMA or Architected algorithm is implemented in the PE for address authentication, in AArch64 state. Defined values are:

APA	Meaning
0b0000	Address Authentication using an Architected algorithm is not implemented.
0b0001	Address Authentication using the QARMA algorithm is implemented,~~implemented.~~ with the EnhancedPAC() function returning FALSE. This applies to~~involves~~ all Pointer Authentication instructions other than the PACGA instruction.
0b0010	Address Authentication using the QARMA algorithm is implemented, with the EnhancedPAC() function returning TRUE. This applies to all Pointer Authentication instructions other than the PACGA instruction.

All other values are reserved.

From Armv8.3, the permitted values are 0b0000 and 0b0001.

If the value of the ID_AA64ISAR1_EL1.API is non-zero, this field must have the value 0b0000.

Otherwise:

Reserved, RES0.

DPB, bits [3:0]

From Armv8.2:

Data Persistence writeback. Indicates support for the DC CVAP and DC CVADP instructions in AArch64 state. Defined values are:

DPB	Meaning
0b0000	DC CVAP not supported.
0b0001	DC CVAP supported.
0b0010	DC CVAP and DC CVADP supported.

All other values are reserved.

ARMv8.2-DCPoP implements the functionality identified by the value 0b0001.

ARMv8.2-DCCVADP implements the functionality identified by the value 0b0010.

From Armv8.2 to Armv8.4, the only permitted value is 0b0001.

From Armv8.5, the only permitted value is 0b0010

Otherwise:

Reserved, RES0.

If API == 0000 and APA == 0000, then:

- The TCR_EL1.{TBID,TBID0}, TCR_EL2.{TBID0,TBID1}, TCR_EL2.TBID and TCR_EL3.TBID bits are RES0.
- APIAKeyHi_EL1, APIAKeyLo_EL1, APIBKeyHi_EL1, APIBKeyLo_EL1, APDAKeyHi_EL1, APDAKeyLo_EL1, APDBKeyHi_EL1, APDBKeyLo_EL1 are not allocated.
- SCTLR_ELx.EnIA, SCTLR_ELx.EnIB, SCTLR_ELx.EnDA, SCTLR_ELx.EnDB are all RES0.

If API == 0000 and APA == 0000 and GPI == 0000 and GPA == 0000, then:

- HCR_EL2.APK and HCR_EL2.API are RES0.
- SCR_EL3.APK and SCR_EL3.API are RES0.

Accessing the ID_AA64ISAR1_EL1

Accesses to this register use the following encodings:

```
MRS <Xt>, ID_AA64ISAR1_EL1
```

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b11	0b000	0b0000	0b0110	0b001
~~0b11~~	~~0b0000~~	~~0b000~~	~~0b001~~	~~0b0110~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TID3 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        return ID_AA64ISAR1_EL1;
elsif PSTATE.EL == EL2 then
    return ID_AA64ISAR1_EL1;
elsif PSTATE.EL == EL3 then
    return ID_AA64ISAR1_EL1;
```

2713/0312/20192018 2116:5942: e5e4db499bf9867a4b93324c4dbac985d3da93766379d01e197f1d40720d32d0f84e419e9187e009

(old) htmldiff from- (new)

ID_AA64MMFR0_EL1, AArch64 Memory Model Feature Register 0

The ID_AA64MMFR0_EL1 characteristics are:

Purpose

Provides information about the implemented memory model and memory management support in AArch64 state.

For general information about the interpretation of the ID registers see 'Principles of the ID scheme for fields in ID registers' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section D10.4.1.

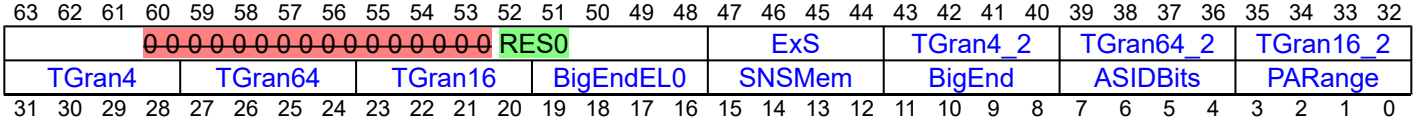
Configuration

Attributes

ID_AA64MMFR0_EL1 is a 64-bit register.

Field descriptions

The ID_AA64MMFR0_EL1 bit assignments are:



Bits [63:48]

Reserved, RES0.

ExS, bits [47:44]

From Armv8.5:

Support for disabling context synchronising exception entry and exit. Defined values are:

ExS	Meaning
0b0000	All exception entries and exits are context synchronization events.
0b0001	Non-context synchronising exception entry and exit are supported.

All other values are reserved.

Otherwise:

Reserved, RES0.

TGran4_2, bits [43:40]

From Armv8.5:

Support for 4KB memory granule size for stage 2. Defined values are:

TGran4_2	Meaning
0b0000	4KB Stage 2 granule is identified in the TGran4 field
0b0001	4KB granule not supported at stage 2
0b0010	4KB granule supported at stage 2

All other values are reserved.

The 0b0000 value is deprecated.

Otherwise:

Reserved, RES0.

TGran64_2, bits [39:36]

From Armv8.5:

Support for 64KB memory granule size for stage 2. Defined values are:

TGran64_2	Meaning
0b0000	64KB Stage 2 granule is identified in the TGran64 field
0b0001	64KB granule not supported at stage 2
0b0010	64KB granule supported at stage 2

All other values are reserved.

The 0b0000 value is deprecated.

Otherwise:

Reserved, RES0.

TGran16_2, bits [35:32]

From Armv8.5:

Support for 16KB memory granule size for stage 2. Defined values are:

TGran16_2	Meaning
0b0000	16KB Stage 2 granule is identified in the TGran16 field
0b0001	16KB granule not supported at stage 2
0b0010	16KB granule supported at stage 2

All other values are reserved.

The 0b0000 value is deprecated.

Otherwise:

Reserved, RES0.

TGran4, bits [31:28]

Support for 4KB memory translation granule size. Defined values are:

TGran4	Meaning
0b0000	4KB granule supported.
0b1111	4KB granule not supported.

All other values are reserved.

TGran64, bits [27:24]

Support for 64KB memory translation granule size. Defined values are:

TGran64	Meaning
0b0000	64KB granule supported.
0b1111	64KB granule not supported.

All other values are reserved.

TGran16, bits [23:20]

Support for 16KB memory translation granule size. Defined values are:

TGran16	Meaning
0b0000	16KB granule not supported.
0b0001	16KB granule supported.

All other values are reserved.

BigEndEL0, bits [19:16]

Mixed-endian support at EL0 only. Defined values are:

BigEndEL0	Meaning
0b0000	No mixed-endian support at EL0. The SCTLR_EL1.E0E bit has a fixed value.
0b0001	Mixed-endian support at EL0. The SCTLR_EL1.E0E bit can be configured.

All other values are reserved.

This field is invalid and is RES0 if the BigEnd field, bits [11:8], is not 0b0000.

SNSMem, bits [15:12]

Secure versus Non-secure Memory distinction. Defined values are:

SNSMem	Meaning
0b0000	Does not support a distinction between Secure and Non-secure Memory.
0b0001	Does support a distinction between Secure and Non-secure Memory.

All other values are reserved.

BigEnd, bits [11:8]

Mixed-endian configuration support. Defined values are:

BigEnd	Meaning
0b0000	No mixed-endian support. The SCTLR_ELx.EE bits have a fixed value. See the BigEndEL0 field, bits[19:16], for whether EL0 supports mixed-endian.
0b0001	Mixed-endian support. The SCTLR_ELx.EE and SCTLR_EL1.E0E bits can be configured.

All other values are reserved.

ASIDBits, bits [7:4]

Number of ASID bits. Defined values are:

ASIDBits	Meaning
0b0000	8 bits.
0b0010	16 bits.

All other values are reserved.

PARange, bits [3:0]

Physical Address range supported. Defined values are:

PARange	Meaning
0b0000	32 bits, 4GB.
0b0001	36 bits, 64GB.
0b0010	40 bits, 1TB.
0b0011	42 bits, 4TB.
0b0100	44 bits, 16TB.
0b0101	48 bits, 256TB.
0b0110	52 bits, 4PB.

All other values are reserved.

The value 0b0110 is permitted only if the implementation includes ARMv8.2-LPA, otherwise it is reserved.

Accessing the ID_AA64MMFR0_EL1

Accesses to this register use the following encodings:

```
MRS <Xt>, ID_AA64MMFR0_EL1
```

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b11	0b000	0b0000	0b0111	0b000
~~0b11~~	~~0b0000~~	~~0b000~~	~~0b000~~	~~0b0111~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TID3 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        return ID_AA64MMFR0_EL1;
elsif PSTATE.EL == EL2 then
    return ID_AA64MMFR0_EL1;
elsif PSTATE.EL == EL3 then
    return ID_AA64MMFR0_EL1;
```

27~~13~~/03~~12~~/2019~~2018~~ 21~~16~~:59~~42~~; e5e4db499bf9867a4b93324c4dbac985d3da93766~~379d01e197f1d40720d32d0f84e419e9187e009~~

(old) htmldiff from- (new)

ID_AA64MMFR2_EL1, AArch64 Memory Model Feature Register 2

The ID_AA64MMFR2_EL1 characteristics are:

Purpose

Provides information about the implemented memory model and memory management support in AArch64 state.

For general information about the interpretation of the ID registers see Principles of the ID scheme for fields in ID registers.

Configuration

This register is present only from Armv8.2. Otherwise, direct accesses to ID_AA64MMFR2_EL1 are RES0.

RW fields in this register reset to architecturally UNKNOWN values.

Attributes

ID_AA64MMFR2_EL1 is a 64-bit register.

Field descriptions

The ID_AA64MMFR2_EL1 bit assignments are:

63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
E0PD				EVT				BBM				TTL				~~0~~	~~0~~	~~0~~	~~0~~	RES0				FWB				IDS				AT			
ST				NV				CCIDX				VARange				IESB				LSM				UAO				CnP			
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

E0PD, bits [63:60]

From Armv8.5:

Indicates support for the ARMv8.5-E0PD mechanism. Defined values are:

E0PD	Meaning
0b0000	E0PDx mechanism is not implemented.
0b0001	E0PDx mechanism is implemented.

All other values are reserved.

In Armv8.4, the only permitted value is 0b0000.

From Armv8.5, the only permitted values is 0b0001.

Otherwise:

Reserved, RES0.

EVT, bits [59:56]

When~~From~~ ARMv8.2-EVT is implemented~~Armv8.5~~:

Enhanced Virtualization Traps. If EL2 is implemented, indicates support for the ~~TICAB, TOCU and TID4 traps. Defined values are:~~ HCR_EL2.{TTLBOS, TTLBIS, TOCU, TICAB, TID4} traps. Defined values are:

EVT	Meaning
0b0000	HCR_EL2.{TTLBOS, TTLBIS, TOCU, ~~,~~TICAB, TID4} traps are not supported.~~HCR_EL2.TOCU, HCR_EL2.TID4 traps are not supported.~~
0b0001	HCR_EL2.{TOCU, ~~,~~TICAB, TID4} traps are supported. HCR_EL2.{TTLBOS~~.TOCU~~, TTLBIS} traps are not supported.~~HCR_EL2.TID4 traps are supported. HCR_EL2.TTLBIS, HCR_EL2.TTLBOS traps are not supported~~
0b0010	HCR_EL2.{TTLBOS, TTLBIS, TOCU, ~~,~~TICAB, TID4} traps are supported.~~HCR_EL2.TOCU, HCR_EL2.TID4, HCR_EL2.TTLBIS, HCR_EL2.TTLBOS traps are supported~~

All other values are reserved.

In Armv8.0~~Armv8.4~~, the only permitted value is 0b0000.

From Armv8.1, the permitted values are 0b0000, 0b0001, and 0b0010.

From Armv8.5, the permitted values are:

- 0b0000 when EL2 is not implemented.
- 0b0010 when EL2 is implemented. This feature is identified as ARMv8.2-EVT.

Otherwise:

Reserved, RES0.

BBM, bits [55:52]

From Armv8.4:

Allows identification of the requirements of the hardware to have break-before-make sequences when changing block size for a translation.

BBM	Meaning
0b0000	Level 0 support for changing block size is supported.
0b0001	Level 1 support for changing block size is supported.
0b0010	Level 2 support for changing block size is supported.

All other values are reserved.

ARMv8.4-TTRem implements the functionality identified by the values 0b0000, 0b0001, and 0b0010.

From Armv8.4, the permitted values are 0b0000, 0b0001, and 0b0010.

Otherwise:

Reserved, RES0.

TTL, bits [51:48]

From Armv8.4:

Indicates support for TTL field in address operations. Defined values are:

TTL	Meaning
0b0000	TLB maintenance instructions by address have bits[47:44] as RES0.
0b0001	TLB maintenance instructions by address have bits[47:44] holding the TTL field.

All other values are reserved.

ARMv8.4-TTL implements the functionality identified by the value 0b0001.

This field affects TLBI IPAS2E1, TLBI IPAS2E1IS, TLBI IPAS2E1OS, TLBI IPAS2LE1, TLBI IPAS2LE1IS, TLBI IPAS2LE1OS, TLBI VAAE1, TLBI VAAE1IS, TLBI VAAE1OS, TLBI VAALE1, TLBI VAALE1IS, TLBI VAALE1OS, TLBI VAE1, TLBI VAE1IS, TLBI VAE1OS, TLBI VAE2, TLBI VAE2IS, TLBI VAE2OS, TLBI VAE3, TLBI VAE3IS, TLBI VAE3OS,TLBI VALE1, TLBI VALE1IS, TLBI VALE1OS, TLBI VALE2, TLBI VALE2IS, TLBI VALE2OS, TLBI VALE3, TLBI VALE3IS, TLBI VALE3OS.

From Armv8.4, the only permitted value is 0b0001.

Otherwise:

Reserved, RES0.

Bits [47:44]

Reserved, RES0.

FWB, bits [43:40]

From Armv8.4:

Indicates support for HCR_EL2.FWB

FWB	Meaning
0b0000	HCR_EL2.FWB bit is not supported and the field is RES0
0b0001	HCR_EL2.FWB is supported.

If ARMv8.4-SecEL2 is implemented the only permitted value is 0b0001

All other values reserved.

This field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

IDS, bits [39:36]

From Armv8.4:

Indicates the value of ESR_ELx.EC that reports an exception generated by a read access to the feature ID space. Defined values are:

IDS	Meaning
0b0000	An exception, othergenerated thanby a trapread causedaccess to the feature ID space is reported by ESR_ELx.EC == HCR_EL2.TID3, generated by a read access to the feature ID space is reported by ESR_ELx.EC == 0x0.
0b0001	AllAn exceptionsexception generated by ana AArch64 read access to the feature ID space areis reported by ESR_ELx.EC == 0x18.

All other values are reserved.

The Feature ID space is defined as the System register space in AArch64 with op0==3, op1=={0, 1, 3}, CRn==0, CRm=={0-7}, op2=={0-7}.

ARMv8.4-IDST implements the functionality identified by the value 0b0001.

From Armv8.4, the only permitted value is 0b0001.

Otherwise:

Reserved, RES0.

AT, bits [35:32]

From Armv8.4:

Identifies support for unaligned single-copy atomicity and atomic functions. Defined values are:

AT	Meaning
0b0000	Unaligned single-copy atomicity and atomic functions are not supported.
0b0001	Unaligned single-copy atomicity and atomic functions with a 16-byte address range aligned to 16-bytes are supported.

All other values are reserved.

ARMv8.4-LSE implements the functionality identified by the value 0b0001.

From Armv8.4, the only permitted value is 0b0001.

Otherwise:

Reserved, RES0.

ST, bits [31:28]

From Armv8.4:

Identifies support for small translation tables. Defined values are:

ST	Meaning
0b0000	The maximum value of the TCR_ELx.{T0SZ,T1SZ} and VTCR_EL2.T0SZ fields is 39.
0b0001	The maximum value of the TCR_ELx.{T0SZ,T1SZ} and VTCR_EL2.T0SZ fields is 48 for 4KB and 16KB granules, and 47 for 64KB granules.

All other values are reserved.

ARMv8.4-TTST implements the functionality identified by the value 0b0001.

If ARMv8.4-SecEL2 is implemented the only permitted value is 0b0001.

In an implementation which does not support Secure EL2, the permitted values are 0b0000 and 0b0001.

Otherwise:

Reserved, RES0.

NV, bits [27:24]

From Armv8.4:

Nested Virtualization. If EL2 is implemented, indicates support for the use of nested virtualization. Defined values are:

NV	Meaning
0b0000	Nested virtualization is not supported.
0b0001	The HCR_EL2.NV, HCR_EL2.NV1, HCR_EL2.AT bits are implemented.
0b0010	The VNCR_EL2 register and the HCR_EL2.{AT, NV, NV1, NV2} bits are implemented.

All other values are reserved.

In Armv8.2, the only permitted value is 0b0000.

In Armv8.3, the permitted values are:

- When EL2 is not implemented, 0b0000.
- When EL2 is implemented, 0b0001.

The feature ARMv8.3-NV implements the functionality identified by the value 0b0001.

In Armv8.4, the permitted values are:

- When EL2 is not implemented, 0b0000.
- When EL2 is implemented, 0b0010.

The feature ARMv8.4-NV implements the functionality identified by the value 0b0010.

From Armv8.3:

Nested Virtualization. If EL2 is implemented, indicates support for the use of nested virtualization. Defined values are:

NV	Meaning
0b0000	Nested virtualization is not supported.
0b0001	The HCR_EL2.NV, HCR_EL2.NV1, HCR_EL2.AT bits are implemented.

All other values are reserved.

In Armv8.2, the only permitted value is 0b0000.

From Armv8.3, the permitted values are:

- When EL2 is not implemented, 0b0000.
- When EL2 is implemented, 0b0001.

The feature ARMv8.3-NV implements the functionality identified by this value.

Otherwise:

Reserved, RES0.

CCIDX, bits [23:20]

From Armv8.3:

Support for the use of revised CCSIDR_EL1 register format. Defined values are:

CCIDX	Meaning
0b0000	32-bit format implemented for all levels of the CCSIDR_EL1.
0b0001	64-bit format implemented for all levels of the CCSIDR_EL1.

All other values are reserved.

This feature is identified as ARMv8.3-CCIDX.

From Armv8.3, the permitted values are 0b0000 and 0b0001.

Otherwise:

Reserved, RES0.

VARange, bits [19:16]

From Armv8.2:

Indicates support for a larger virtual address. Defined values are:

VARange	Meaning
0b0000	VMSAv8-64 supports 48-bit VAs.
0b0001	VMSAv8-64 supports 52-bit VAs when using the 64KB translation granule. The other translation granules support 48-bit VAs.

All other values are reserved.

ARMv8.2-LVA implements the functionality identified by the value 0b0001.

From Armv8.2, the permitted values are 0b0000 and 0b0001.

Otherwise:

Reserved, RES0.

IESB, bits [15:12]

From Armv8.2:

Indicates support for the IESB bit in the SCTLR_ELx registers. Defined values are:

IESB	Meaning
0b0000	IESB bit in the SCTLR_ELx registers is not supported.
0b0001	IESB bit in the SCTLR_ELx registers is supported.

All other values are reserved.

ARMv8.2-IESB implements the functionality identified by the value 0b0001.

From Armv8.2, the only permitted value is 0b0001.

Otherwise:

Reserved, RES0.

LSM, bits [11:8]

From Armv8.2:

Indicates support for LSMAOE and nTLSMD bits in SCTLR_EL1 and SCTLR_EL2. Defined values are:

LSM	Meaning
0b0000	LSMAOE and nTLSMD bits not supported.
0b0001	LSMAOE and nTLSMD bits supported.

All other values are reserved.

ARMv8.2-LSMAOC implements the functionality identified by the value 0b0001.

Otherwise:

Reserved, RES0.

UAO, bits [7:4]

From Armv8.2:

User Access Override. Defined values are:

UAO	Meaning
0b0000	UAO not supported.
0b0001	UAO supported.

All other values are reserved.

ARMv8.2-UAO implements the functionality identified by the value 0b0001.

From Armv8.2, the only permitted value is 0b0001.

Otherwise:

Reserved, RES0.

CnP, bits [3:0]

From Armv8.2:

Common not Private translations. Defined values are:

CnP	Meaning
0b0000	Common not Private translations not supported.
0b0001	Common not Private translations supported.

All other values are reserved.

ARMv8.2-TTCNP implements the functionality identified by the value 0b0001.

From Armv8.2, the only permitted value is 0b0001.

Otherwise:

Reserved, RES0.

Accessing the ID_AA64MMFR2_EL1

Accesses to this register use the following encodings:

```
MRS <Xt>, ID_AA64MMFR2_EL1
```

op0	op1	CRn	CRm	op2
~~CRn~~	~~op0~~	~~op1~~	~~op2~~	~~CRm~~
0b11	0b000	0b0000	0b0111	0b010
~~0b0000~~	~~0b11~~	~~0b000~~	~~0b010~~	~~0b0111~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TID3 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        return ID_AA64MMFR2_EL1;
elsif PSTATE.EL == EL2 then
    return ID_AA64MMFR2_EL1;
elsif PSTATE.EL == EL3 then
    return ID_AA64MMFR2_EL1;
```

27~~13~~/03~~12~~/2019~~2018~~ 21~~16~~:59~~42~~: e5e4db499bf9867a4b93324c4dbac985d3da93766~~379d01e197f1d40720d32d0f84e419e9187e009~~

(old) htmldiff from- (new)

ID_MMFR4_EL1, AArch32 Memory Model Feature Register 4

The ID_MMFR4_EL1 characteristics are:

Purpose

Provides information about the implemented memory model and memory management support in AArch32 state.

Must be interpreted with ID_MMFR0_EL1, ID_MMFR1_EL1, ID_MMFR2_EL1, and ID_MMFR3_EL1.

For general information about the interpretation of the ID registers see 'Principles of the ID scheme for fields in ID registers' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section D10.4.1.

Configuration

AArch64 System register ID_MMFR4_EL1 bits [31:0] are architecturally mapped to AArch32 System register ID_MMFR4[31:0] .

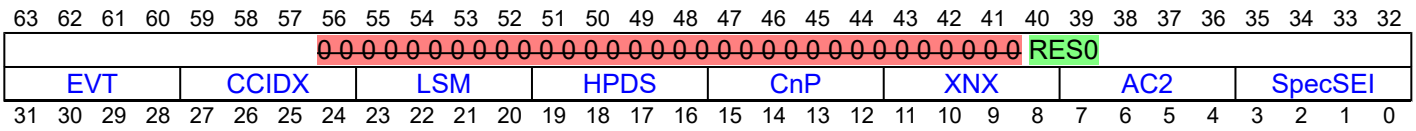
In an implementation that supports only AArch64 state, this register is UNKNOWN.

Attributes

ID_MMFR4_EL1 is a 64-bit register.

Field descriptions

The ID_MMFR4_EL1 bit assignments are:



Bits [63:32]

Reserved, RES0.

EVT, bits [31:28]

WhenFrom ARMv8.2-EVT is implementedArmv8.5:

Enhanced Virtualization Traps. If EL2 is implemented, indicates support for the TICAB, TOCU and TID4 traps. Defined values are: HCR2.{TTLBIS, TOCU, TICAB, TID4} traps. Defined values are:

EVT	Meaning
0b0000	HCR2.{TTLBISHCR2.TICAB, TOCUHCR2.TOCU, TICAB, TID4}HCR2.TID4 traps are not supported.
0b0001	HCR2.{TOCU, TICAB, TID4} traps are supported. HCR2.TTLBISHCR2.TICAB, trapHCR2.TOCU, isHCR2.TID4 nottraps supported.are supported
0b0010	HCR2.{TTLBIS, TOCU, TICAB, TID4} traps are supported.

All other values are reserved.

In Armv8.0,Armv8.4 the only permitted value is 0b0000.

From Armv8.1, the permitted values are 0b0000, 0b0001, and 0b0010.

From Armv8.5 the permitted values are:

- 0b0000 when EL2 is not implemented.
- 0b0001 when EL2 is implemented. This feature is identified as ARMv8.2-EVT.

Otherwise:

Reserved, RES0.

CCIDX, bits [27:24]

From Armv8.3:

Support for use of the revised CCSIDR format and the presence of the CCSIDR2 is indicated. Defined values are:

CCIDX	Meaning
0b0000	32-bit format implemented for all levels of the CCSIDR, and the CCSIDR2 register is not implemented.
0b0001	64-bit format implemented for all levels of the CCSIDR, and the CCSIDR2 register is implemented.

All other values are reserved.

From Armv8.3, the permitted values are 0b0000 and 0b0001. This feature is identified as ARMv8.3-CCIDX.

Otherwise:

Reserved, RAZ.

LSM, bits [23:20]

From Armv8.2:

Indicates support for LSMAOE and nTLSMD bits in HSCTLR and SCTLR. Defined values are:

LSM	Meaning
0b0000	LSMAOE and nTLSMD bits not supported.
0b0001	LSMAOE and nTLSMD bits supported.

All other values are reserved.

ARMv8.2-LSMAOC implements the functionality identified by the value 0b0001.

Otherwise:

Reserved, RAZ.

HPDS, bits [19:16]

From Armv8.2:

Hierarchical permission disables bits in translation tables. Defined values are:

HPDS	Meaning
0b0000	Disabling of hierarchical controls not supported.
0b0001	Supports disabling of hierarchical controls using the TTBCR2.HPD0, TTBCR2.HPD1, and HTCR.HPD bits.
0b0010	As for value 0b0001, and adds possible hardware allocation of bits[62:59] of the translation table descriptors from the final lookup level for IMPLEMENTATION DEFINED use.

All other values are reserved.

ARMv8.2-AA32HPD implements the functionality identified by the value 0b0001.

ARMv8.2-TTPBHA implements the functionality added by the value 0b0010.

Note

The value 0b0000 implies that the encoding for TTBCR2 is unallocated.

Otherwise:

Reserved, RAZ.

CnP, bits [15:12]

From Armv8.2:

Common not Private translations. Defined values are:

CnP	Meaning
0b0000	Common not Private translations not supported.
0b0001	Common not Private translations supported.

All other values are reserved.

ARMv8.2-TTCNP implements the functionality identified by the value 0b0001.

From Armv8.2 the only permitted value is 0b0001.

Otherwise:

Reserved, RAZ.

XNX, bits [11:8]

From Armv8.2:

Support for execute-never control distinction by Exception level at stage 2. Defined values are:

XNX	Meaning
0b0000	Distinction between EL0 and EL1 execute-never control at stage 2 not supported.
0b0001	Distinction between EL0 and EL1 execute-never control at stage 2 supported.

All other values are reserved.

ARMv8.2-TTS2UXN implements the functionality identified by the value 0b0001.

When ARMv8.2-TTS2UXN is implemented:

- If all of the following conditions are true it is IMPLEMENTATION DEFINED whether the value of ID_MMFR4_EL1.XNX is 0b0000 or 0b0001:
  - ID_AA64MMFR1_EL1.XNX ==1.
  - EL2 cannot use AArch32.
  - EL1 can use AArch32.
- If EL2 can use AArch32 then the only permitted value is 0b0001.

Otherwise:

Reserved, RAZ.

AC2, bits [7:4]

Indicates the extension of the ACTLR and HACTLR registers using ACTLR2 and HACTLR2. Defined values are:

AC2	Meaning
0b0000	ACTLR2 and HACTLR2 are not implemented.
0b0001	ACTLR2 and HACTLR2 are implemented.

All other values are reserved.

In Armv8.0 and Armv8.1 the permitted values are 0b0000 and 0b0001.

From Armv8.2, the only permitted value is 0b0001.

SpecSEI, bits [3:0]

When RAS is implemented:

Describes whether the PE can generate SError interrupt exceptions from speculative reads of memory, including speculative instruction fetches. The defined values of this field are:

SpecSEI	Meaning
0b0000	The PE never generates an SError interrupt due to an External abort on a speculative read.
0b0001	The PE might generate an SError interrupt due to an External abort on a speculative read.

All other values are reserved.

Otherwise:

Reserved, RES0. This provides no information about whether the PE generates a speculative SError interrupt.

Accessing the ID_MMFR4_EL1

Accesses to this register use the following encodings:

```
MRS <Xt>, ID_MMFR4_EL1
```

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b11	0b000	0b0000	0b0010	0b110
~~0b11~~	~~0b0000~~	~~0b000~~	~~0b110~~	~~0b0010~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && (!IsZero(ID_MMFR4_EL1) || boolean
IMPLEMENTATION_DEFINED "ID_MMFR4_EL1 trapped by HCR_EL2.TID3 and ID_MMFR4 trapped by HCR_EL2.TID3
and HCR.TID3") && HCR_EL2.TID3 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        return ID_MMFR4_EL1;
elsif PSTATE.EL == EL2 then
    return ID_MMFR4_EL1;
elsif PSTATE.EL == EL3 then
    return ID_MMFR4_EL1;
```

27~~13~~/03~~12~~/2019~~2018~~ 21~~16~~:59~~42~~; e5e4db499bf9867a4b93324c4dbac985d3da93766~~379d01e197f1d40720d32d0f84e419e9187e009~~

(old) htmldiff from- (new)

MAIR_EL1, Memory Attribute Indirection Register (EL1)

The MAIR_EL1 characteristics are:

Purpose

Provides the memory attribute encodings corresponding to the possible AttrIndx values in a Long-descriptor format translation table entry for stage 1 translations at EL1.

Configuration

AArch64 System register MAIR_EL1 bits [31:0] are architecturally mapped to AArch32 System register PRRR[31:0] when TTBCR.EAE == 0.

AArch64 System register MAIR_EL1 bits [31:0] are architecturally mapped to AArch32 System register MAIR0[31:0] when TTBCR.EAE == 1.

AArch64 System register MAIR_EL1 bits [63:32] are architecturally mapped to AArch32 System register NMRR[31:0] when TTBCR.EAE == 0.

AArch64 System register MAIR_EL1 bits [63:32] are architecturally mapped to AArch32 System register MAIR1[31:0] when TTBCR.EAE == 1.

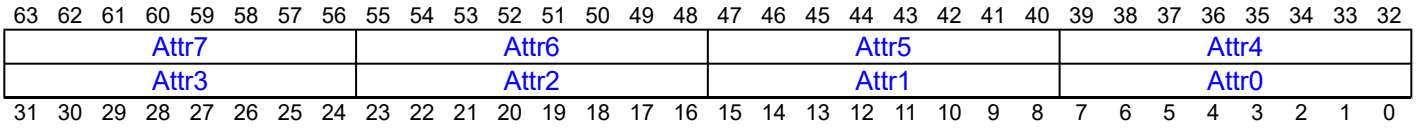
RW fields in this register reset to architecturally UNKNOWN values.

Attributes

MAIR_EL1 is a 64-bit register.

Field descriptions

The MAIR_EL1 bit assignments are:



MAIR_EL1 is permitted to be cached in a TLB.

Attr<n>, bits [8n+7:8n], for n = 0 to 7

The memory attribute encoding for an AttrIndx[2:0] entry in a Long descriptor format translation table entry, where AttrIndx[2:0] gives the value of <n> in Attr<n>.

Attr is encoded as follows:

Attr	Meaning
0b0000dd00	Device memory. See encoding of 'dd' for the type of Device memory.
0b0000ddxx, (xx != 00)	UNPREDICTABLE
0booooiiii, (oooo != 0000 and iiii != 0000)	Normal memory. See encoding of 'oooo' and 'iiii' for the type of Normal Memory.
0b11110000	Tagged Normal Memory. Inner+Outer Write-Back Non-Transient memory, Inner+Outer Read-Allocate, Inner+Outer Write-Allocate.
0bxxxx0000, (xxxx != 0000 and xxxx != 1111)	UNPREDICTABLE

'dd' is encoded as follows:

dd	Meaning
0b00	Device-nGnRnE memory
0b01	Device-nGnRE memory
0b10	Device-nGRE memory
0b11	Device-GRE memory

'oooo' is encoded as follows:

'oooo'	Meaning
0b0000	See encoding of Attr
0b00RW, RW not 0b00	Normal memory, Outer Write-Through Transient
0b0100	Normal memory, Outer Non-cacheable
0b01RW, RW not 0b00	Normal memory, Outer Write-Back Transient
0b10RW	Normal memory, Outer Write-Through Non-transient
0b11RW	Normal memory, Outer Write-Back Non-transient

R = Outer Read-Allocate policy, W = Outer Write-Allocate policy.

'iiii' is encoded as follows:

'iiii'	Meaning
0b0000	See encoding of Attr
0b00RW, RW not 0b00	Normal memory, Inner Write-Through Transient
0b0100	Normal memory, Inner Non-cacheable
0b01RW, RW not 0b00	Normal memory, Inner Write-Back Transient
0b10RW	Normal memory, Inner Write-Through Non-transient
0b11RW	Normal memory, Inner Write-Back Non-transient

R = Inner Read-Allocate policy, W = Inner Write-Allocate policy.

The R and W bits in 'oooo' and 'iiii' fields have the following meanings:

R or W	Meaning
0b0	No Allocate
0b1	Allocate

This field resets to an architecturally UNKNOWN value.

Accessing the MAIR_EL1

When HCR_EL2.E2H is 1, without explicit synchronization, access from EL3 using the mnemonic MAIR_EL1 or MAIR_EL12 are not guaranteed to be ordered with respect to accesses using the other mnemonic.

Accesses to this register use the following encodings:

```
MRS <Xt>, MAIR_EL1
```

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b11	0b000	0b1010	0b0010	0b000
~~0b11~~	~~0b1010~~	~~0b000~~	~~0b000~~	~~0b0010~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TRVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<NV2,NV1,NV> == '111' then
        return NVMem[0x140];
    else
        return MAIR_EL1;
elsif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' then
        return MAIR_EL2;
    else
        return MAIR_EL1;
elsif PSTATE.EL == EL3 then
    return MAIR_EL1;
```

```
MSR MAIR_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b11	0b000	0b1010	0b0010	0b000
~~0b11~~	~~0b1010~~	~~0b000~~	~~0b000~~	~~0b0010~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<NV2,NV1,NV> == '111' then
        NVMem[0x140] = X[t];
    else
        MAIR_EL1 = X[t];
elsif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' then
        MAIR_EL2 = X[t];
    else
        MAIR_EL1 = X[t];
elsif PSTATE.EL == EL3 then
    MAIR_EL1 = X[t];
```

```
MRS <Xt>, MAIR_EL12
```

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b11	0b101	0b1010	0b0010	0b000
~~0b11~~	~~0b1010~~	~~0b101~~	~~0b000~~	~~0b0010~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '101' then
        return NVMem[0x140];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if EL2Enabled() && HCR_EL2.E2H == '1' then
    if PSTATE.EL == EL2 then
        return MAIR_EL1;
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if EL2Enabled() && HCR_EL2.E2H == '1' then
        return MAIR_EL1;
else
    else
        UNDEFINED;
```

```
MSR MAIR_EL12, <Xt>
```

op0	op1	CRn	CRm	op2
op0	CRn	op1	op2	CRm
0b11	0b101	0b1010	0b0010	0b000
0b11	0b1010	0b101	0b000	0b0010

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '101' then
        NVMem[0x140] = X[t];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if EL2Enabled() && HCR_EL2.E2H == '1' then
    if PSTATE.EL == EL2 then
        MAIR_EL1 = X[t];
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if EL2Enabled() && HCR_EL2.E2H == '1' then
        MAIR_EL1 = X[t];
else
    else
        UNDEFINED;
```

27 13/03 12/2019 2018 21 16:59 42; e5e4db499bf9867a4b93324c4dbac985d3da93766379d01e197f1d40720d32d0f84e419e9187e009

(old) htmldiff from- (new)

MDCR_EL2, Monitor Debug Configuration Register (EL2)

The MDCR_EL2 characteristics are:

Purpose

Provides EL2 configuration options for self-hosted debug and the Performance Monitors Extension.

Configuration

AArch64 System register MDCR_EL2 bits [31:0] are architecturally mapped to AArch32 System register HDCR[31:0] .

If EL2 is not implemented, this register is RES0 from EL3.

This register has no effect if EL2 is not enabled in the current Security state.

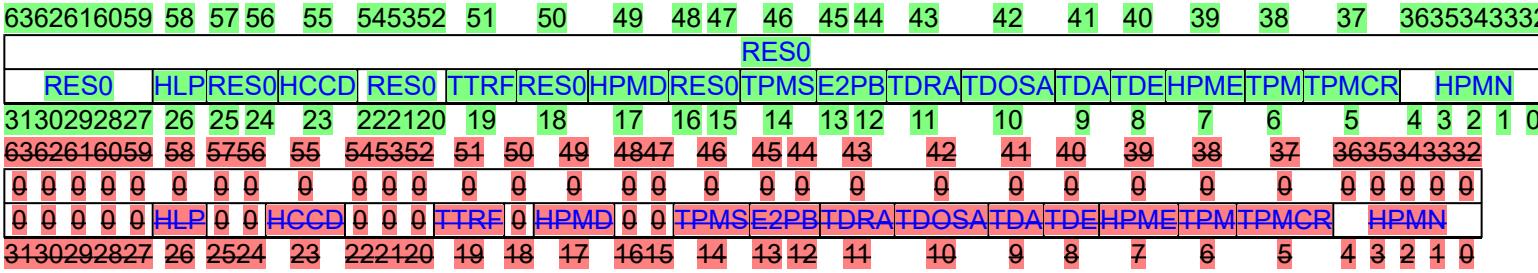
This register is in the Warm reset domain. Some or all RW fields of this register have defined reset values. On a Warm or Cold reset these apply only if the PE resets into an Exception level that is using AArch64. Otherwise, on a Warm or Cold reset RW fields in this register reset to architecturally UNKNOWN values.

Attributes

MDCR_EL2 is a 64-bit register.

Field descriptions

The MDCR_EL2 bit assignments are:



Bits [63:27]

Reserved, RES0.

HLP, bit [26]

When ARMv8.5-PMU is implemented:

Hypervisor Long event counter enable. Determines when unsigned overflow is recorded by a counter overflow bit.

HLP	Meaning
0b0	Event counter overflow on increment that causes unsigned overflow of PMEVCNTR<n>_EL0[31:0].
0b1	Event counter overflow on increment that causes unsigned overflow of PMEVCNTR<n>_EL0[63:0].

If MDCR_EL2.HPMN is less than PMCR_EL0.N or PMCR.N, this bit affects the operation of event counters in the range [MDCR_EL2.HPMN:(PMCR_EL0.N-1)] or [MDCR_EL2.HPMN:(PMCR.N-1)]. Otherwise this bit has no effect on the operation of the event counters.

Note

The effect of MDCR_EL2.HPMN on the operation of this bit applies regardless of whether EL2 is enabled in the current Security state.

For more information see the description of the MDCR_EL2.HPMN~~HPMN~~ field.

On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [25:24]

Reserved, RES0.

HCCD, bit [23]

When ARMv8.5-PMU is implemented:

Hypervisor Cycle Counter Disable. Prohibits PMCCNTR_EL0 from counting at EL2.

HCCD	Meaning
0b0	Cycle counting by PMCCNTR_EL0 is not affected by this bit.
0b1	Cycle counting by PMCCNTR_EL0 is prohibited at EL2.

This bit does not affect the CPU_CYCLES event or any other event that counts cycles.

On a Warm reset, this field resets to 0.

Otherwise:

Reserved, RES0.

Bits [22:20]

Reserved, RES0.

TTRF, bit [19]

When ARMv8.4-Trace is implemented:

Traps use of the Trace Filter Control registers at EL1 to EL2.

TTRF	Meaning
0b0	Accesses to TRFCR_EL1 and TRFCR at EL1 are not affected by this control.
0b1	Accesses to TRFCR_EL1 and TRFCR at EL1 generate a trap exception to EL2 when EL2 is enabled in the current Security state.

Otherwise:

Reserved, RES0.

Bit [18]

Reserved, RES0.

HPMD, bit [17]

When ARMv8.1-PMU is implemented:

Guest Performance Monitors Disable. This control prohibits event counting at EL2.

HPMD	Meaning
0b0	Event counting allowed at EL2.
0b1	Event counting prohibited at EL2. In an Armv8.1 implementation, event counting is prohibited unless enabled by the IMPLEMENTATION DEFINED authentication interface ExternalSecureNoninvasiveDebugEnabled().

This control applies only to:

- The event counters in the range [0:(MDCR_EL2.HPMN-1)].
- If PMCR_EL0.DP is set to 1, PMCCNTR_EL0.

The other event counters are unaffected, and when PMCR_EL0.DP is set to 0, PMCCNTR_EL0 is unaffected.

On a Warm reset, this field resets to 0.

Otherwise:

Reserved, RES0.

Bits [16:15]

Reserved, RES0.

TPMS, bit [14]

When SPE is implemented:

Trap Performance Monitor Sampling. When EL2 is enabled in the current Security state, this field controls access to Statistical Profiling control registers from EL1.

TPMS	Meaning
0b0	Do not trap Statistical Profiling controls to EL2.
0b1	Accesses to Statistical Profiling controls at EL1 generate a Trap exception to EL2 when EL2 is enabled in the current Security state.

Otherwise:

Reserved, RES0.

E2PB, bits [13:12]

When SPE is implemented:

EL2 Profiling Buffer. When EL2 is enabled in the current Security state, this field controls the owning translation regime and access to Profiling Buffer control registers from EL1.

E2PB	Meaning
0b00	Profiling Buffer uses the EL2 stage 1 translation regime. Accesses to Profiling Buffer controls at EL1 generate a Trap exception to EL2 when EL2 is enabled in the current Security state.
0b10	Profiling Buffer uses the EL1&0 stage 1 translation regime. Accesses to Profiling Buffer controls at EL1 generate a Trap exception to EL2 when EL2 is enabled in the current Security state.
0b11	Profiling Buffer uses the EL1&0 stage 1 translation regime. Accesses to Profiling Buffer controls at EL1 are not trapped.

All other values are reserved.

If EL2 is not implemented, or is disabled in the current Security State, the PE behaves as if E2PB == 0b11, other than for a direct read of the register.

On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TDRA, bit [11]

Trap Debug ROM Address register access. Traps System register accesses to the Debug ROM registers to EL2 when EL2 is enabled in the current Security state. This trap is from:

- EL0 using AArch32.
- EL1, regardless of which Execution state it is using.

TDRA	Meaning
0b0	This control does not cause any instructions to be trapped.
0b1	EL0 and EL1 System register accesses to the Debug ROM registers are trapped to EL2 when EL2 is enabled in the current Security state, unless it is trapped by DBGDSCRext.UDCCdis or MDSCR_EL1.TDCC.

The registers for which accesses are trapped are as follows:

AArch64: MDRAR_EL1.

AArch32: DBGDRAR, DBGDSAR.

This field is treated as being 1 for all purposes other than a direct read when one or more of the following are true:

- MDCR_EL2.TDE == 1.
- HCR_EL2.TGE == 1.

On a Warm reset, this field resets to an architecturally UNKNOWN value.

TDOSA, bit [10]

When ARMv8.0-DoubleLock is implemented:

Trap debug OS-related register access. Traps EL1 System register accesses to the powerdown debug registers to EL2, from both Execution states:

TDOSA	Meaning
0b0	This control does not cause any instructions to be trapped.
0b1	EL1 System register accesses to the powerdown debug registers are trapped to EL2 when EL2 is enabled in the current Security state.

The registers for which accesses are trapped are as follows:

AArch64: OSLAR_EL1, OSLSR_EL1, OSDLR_EL1, and DBGPRCR_EL1.

AArch32: DBGOSLSR, DBGOSLAR, DBGOSDLR, and DBGPRCR.

AArch64 and AArch32: Any IMPLEMENTATION DEFINED register with similar functionality that the implementation specifies as trapped by this bit.

Note

These registers are not accessible at EL0.

This field is treated as being 1 for all purposes other than a direct read when one or more of the following are true:

- MDCR_EL2.TDE == 1.
- HCR_EL2.TGE == 1.

On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Trap debug OS-related register access. Traps EL1 System register accesses to the powerdown debug registers to EL2, from both Execution states:

TDOSA	Meaning
0b0	This control does not cause any instructions to be trapped.
0b1	EL1 System register accesses to the powerdown debug registers are trapped to EL2 when EL2 is enabled in the current Security state.

The registers for which accesses are trapped are as follows:

AArch64: OSLAR_EL1, OSLSR_EL1, and DBGPRCR_EL1.

AArch32: DBGOSLSR, DBGOSLAR, and DBGPRCR.

AArch64 and AArch32: Any IMPLEMENTATION DEFINED register with similar functionality that the implementation specifies as trapped by this bit.

It is IMPLEMENTATION DEFINED whether accesses to OSDLR_EL1 and DBGOSDLR are trapped.

Note

These registers are not accessible at EL0.

This field is treated as being 1 for all purposes other than a direct read when one or more of the following are true:

- MDCR_EL2.TDE == 1.
- HCR_EL2.TGE == 1.

On a Warm reset, this field resets to an architecturally UNKNOWN value.

TDA, bit [9]

Trap Debug Access. Traps EL0 and EL1 System register accesses to those debug System registers that are not trapped by either of the following:

- MDCR_EL2.TDRA.
- MDCR_EL2.TDOSA.

TDA	Meaning
0b0	This control does not cause any instructions to be trapped.
0b1	EL0 or EL1 System register accesses to the debug registers are trapped from both Execution states to EL2 when EL2 is enabled in the current Security state, unless the access generates a higher priority exception.

Traps of AArch32 accesses to DBGDTRRXint and DBGDTRTXint are ignored in Debug state.

Traps of AArch64 accesses to DBGDTR_EL0, DBGDTRRX_EL0, and DBGDTRTX_EL0 are ignored in Debug state.

This field is treated as being 1 for all purposes other than a direct read when one or more of the following are true:

- MDCR_EL2.TDE == 1
- HCR_EL2.TGE == 1

On a Warm reset, this field resets to an architecturally UNKNOWN value.

TDE, bit [8]

Trap Debug exceptions.

TDE	Meaning
0b0	This control has no effect on the routing of debug exceptions, and has no effect on accesses to debug registers.
0b1	Debug exceptions generated at EL1 or EL0 are routed to EL2 when EL2 is enabled in the current Security state. The MDCR_EL2.{TDRA, TDOSA, TDA} fields are treated as being 1 for all purposes other than returning the result of a direct read of the register.

This field is treated as being 1 for all purposes other than a direct read when HCR_EL2.TGE == 1.

On a Warm reset, this field resets to an architecturally UNKNOWN value.

HPME, bit [7]

When PMUv3 is implemented:

[MDCR_EL2.HPMN:(N-1)] event counters enable.

HPME	Meaning
0b0	Event counters in the range [MDCR_EL2.HPMN:(PMCR_EL0.N-1)] are disabled.
0b1	Event counters in the range [MDCR_EL2.HPMN:(PMCR_EL0.N-1)] are enabled by PMCNTENSET_EL0.

If MDCR_EL2.HPMN is less than PMCR_EL0.N or PMCR.N, the event counters in the range [MDCR_EL2.HPMN:(PMCR_EL0.N-1)] or [HDCR.HPMN:(PMCR.N-1)], are enabled and disabled by this bit. Otherwise this bit has no effect on the operation of the event counters.

Note

The effect of MDCR_EL2.HPMN on the operation of this bit applies regardless of whether EL2 is enabled in the current Security state.

For more information see the description of the HPMN field.

On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TPM, bit [6]

When PMUv3 is implemented:

Trap Performance Monitors accesses. Traps EL0 and EL1 accesses to all Performance Monitors registers to EL2 when EL2 is enabled in the current Security state, from both Execution states:

TPM	Meaning
0b0	This control does not cause any instructions to be trapped.
0b1	EL0 and EL1 accesses to all Performance Monitors registers are trapped to EL2 when EL2 is enabled in the current Security state.

Note

EL2 does not provide traps on Performance Monitor register accesses through the optional memory-mapped external debug interface.

On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TPMCR, bit [5]

When PMUv3 is implemented:

Trap PMCR_EL0 or PMCR accesses. Traps EL0 and EL1 accesses to the PMCR_EL0 or PMCR to EL2 when EL2 is enabled in the current Security state.

TPMCR	Meaning
0b0	This control does not cause any instructions to be trapped.
0b1	EL0 and EL1 accesses to the PMCR_EL0 or PMCR are trapped to EL2 when EL2 is enabled in the current Security state, unless it is trapped by PMUSERENR.EN or PMUSERENR_EL0.EN.

Note

EL2 does not provide traps on Performance Monitor register accesses through the optional memory-mapped external debug interface.

On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HPMN, bits [4:0]

When PMUv3 is implemented:

Defines the number of event counters that are accessible from EL3, EL2, EL1, and from EL0 if permitted.

If HPMN is less than PMCR_EL0.N, HPMN divides the Performance Monitors into two ranges: [0:(HPMN-1)] and [HPMN:(PMCR_EL0.N-1)].

For an event counter in the range [0:(HPMN-1)]:

- The counter is accessible from EL3, EL2, and EL1, and from EL0 if permitted by PMUSERENR_EL0 or PMUSERENR.
- If~~The counter is enabled by~~ ARMv8.5-PMU is implemented, PMCR_EL0.LP~~E~~ or~~and~~ PMCR~~PMCNTENSET_EL0~~.LP determines whether the counter overflow flag is set on unsigned overflow of PMEVCNTR<n>_EL0[31:0] or PMEVCNTR<n>_EL0[63:0].~~.~~
- The counter is enabled by PMCR_EL0.E or PMCR.E and bit <n> of PMCNTENSET_EL0.

Note

If HPMN is equal to PMCR_EL0.N, this applies to all event counters.

If HPMN is less than PMCR_EL0.N, for an event counter in the range [HPMN:(PMCR_EL0.N-1)]:

- The counter is accessible from EL2 and EL3.
- If ARMv8.4-SecEL2 is disabled or is not implemented, the counter is also accessible from Secure EL1 and from Secure EL0 if permitted by PMUSERENR_EL0.
- If ARMv8.5-PMU is implemented, MDCR_EL2.HLP or HDCR.HLP determines whether the counter overflow flag is set on unsigned overflow of PMEVCNTR<n>_EL0[31:0] or PMEVCNTR<n>_EL0[63:0].
- ~~The counter is enabled by MDCR_EL2.HPME and PMCNTENSET_EL0.~~
- The counter is enabled by MDCR_EL2.HPME or HDCR.HPME and bit <n> of PMCNTENSET_EL0.

If this field is set to 0, or to a value larger than PMCR_EL0.N, then the following CONSTRAINED UNPREDICTABLE behavior applies:

- The value returned by a direct read of MDCR_EL2.HPMN is UNKNOWN.
- Either:
  - An UNKNOWN number of counters are reserved for EL2 and EL3 use. That is, the PE behaves as if MDCR_EL2.HPMN is set to an UNKNOWN non-zero value less than or equal to PMCR_EL0.N.
  - All counters are reserved for EL2 and EL3 use, meaning no counters are accessible from EL1 and EL0.

On a Warm reset, this field resets to the value in PMCR_EL0.N.

Otherwise:

Reserved, RES0.

Accessing the MDCR_EL2

Accesses to this register use the following encodings:

MRS <Xt>, MDCR_EL2

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b11	0b100	0b0001	0b0001	0b001
~~0b11~~	~~0b0001~~	~~0b100~~	~~0b001~~	~~0b0001~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return MDCR_EL2;
elsif PSTATE.EL == EL3 then
    return MDCR_EL2;
```

MSR MDCR_EL2, <Xt>

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b11	0b100	0b0001	0b0001	0b001
~~0b11~~	~~0b0001~~	~~0b100~~	~~0b001~~	~~0b0001~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        MDCR_EL2 = X[t];
elsif PSTATE.EL == EL3 then
    MDCR_EL2 = X[t];
```

27~~13~~/03~~12~~/2019~~2018~~ 21~~16~~:59~~42~~; e5e4db499bf9867a4b93324c4dbac985d3da93766~~379d01e197f1d40720d32d0f84e419e9187e009~~

(old) htmldiff from- (new)

MDCR_EL3, Monitor Debug Configuration Register (EL3)

The MDCR_EL3 characteristics are:

Purpose

Provides EL3 configuration options for self-hosted debug and the Performance Monitors Extension.

Configuration

AArch64 System register MDCR_EL3 bits [31:0] can be mapped to AArch32 System register SDCR[31:0] , but this is not architecturally mandated.

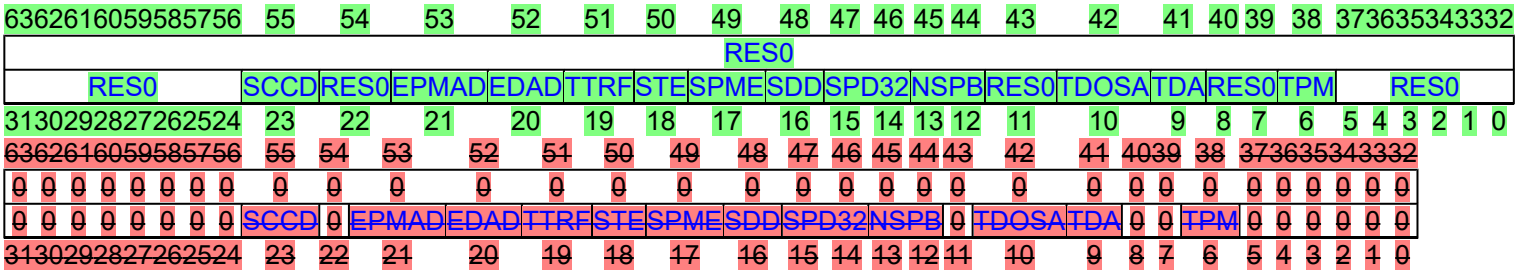
This register is in the Warm reset domain. Some or all RW fields of this register have defined reset values. On a Warm or Cold reset these apply only if the PE resets into an Exception level that is using AArch64. Otherwise, on a Warm or Cold reset RW fields in this register reset to architecturally UNKNOWN values.

Attributes

MDCR_EL3 is a 64-bit register.

Field descriptions

The MDCR_EL3 bit assignments are:



Bits [63:24]

Reserved, RES0.

SCCD, bit [23]

When ARMv8.5-PMU is implemented:

Secure Cycle Counter Disable. Prohibits PMCCNTR_EL0 from counting in Secure state.

SCCD	Meaning
0b0	Cycle counting by PMCCNTR_EL0 is not affected by this bit.
0b1	Cycle counting by PMCCNTR_EL0 is prohibited in Secure state.

This bit does not affect the CPU_CYCLES event or any other event that counts cycles.

On a Warm reset, this field resets to 0.

Otherwise:

Reserved, RES0.

Bit [22]

Reserved, RES0.

EPMAD, bit [21]

When ARMv8.4-Debug is implemented and PMUv3 is implemented:

External debug interface Performance Monitors registers disable. This disables Non-secure access to these registers by an external debugger.

EPMAD	Meaning
0b0	Non-secure access to Performance Monitors registers from external debugger is enabled.
0b1	Non-secure access to Performance Monitors registers from external debugger is disabled.

If the Performance Monitors Extension does not support external debug interface accesses this bit is RES0.

If EL3 and EL2 are not implemented and the Effective value of SCR_EL3.NS is 0b0, then the Effective value of this bit is 0b1.

On a Warm reset, this field resets to 0.

When PMUv3 is implemented:

External debug interface Performance Monitors registers disable. This disables access to these registers by an external debugger.

EPMAD	Meaning
0b0	Access to Performance Monitors registers from external debugger is enabled.
0b1	Access to Performance Monitors registers from external debugger is disabled, unless overridden by the IMPLEMENTATION DEFINED authentication interface.

If the Performance Monitors Extension ~~is not implemented, or~~ does not support external debug interface accesses this bit is RES0.

If EL3 and EL2 are not implemented and the Effective value of SCR_EL3.NS is 0b0, then the Effective value of this bit is 0b1.

On a Warm reset, this field resets to 0.

Otherwise:

Reserved, RES0.

EDAD, bit [20]

When ARMv8.4-Debug is implemented:

External debug interface breakpoint and watchpoint register access disable. This disables access to these registers by an external debugger.

EDAD	Meaning
0b0	Non-secure access to debug registers from external debugger is enabled.
0b1	Non-secure access to breakpoint and watchpoint registers, and OSLAR_EL1 from external debugger is disabled.

If EL3 and EL2 are not implemented and the Effective value of SCR_EL3.NS is 0b0, then the Effective value of this field is 0b1.

On a Warm reset, this field resets to 0.

When ARMv8.2-Debug is implemented:

External debug interface breakpoint and watchpoint register access disable. This disables access to these registers by an external debugger.

EDAD	Meaning
0b0	Access to debug registers, and to OSLAR_EL1 from external debugger is enabled.
0b1	Access to breakpoint and watchpoint registers, and to OSLAR_EL1 from external debugger is disabled, unless overridden by the IMPLEMENTATION DEFINED authentication interface.

If EL3 and EL2 are not implemented and the Effective value of SCR_EL3.NS is 0b0, then the Effective value of this field is 0b1.

On a Warm reset, this field resets to 0.

Otherwise:

External debug interface breakpoint and watchpoint register access disable. This disables access to these registers by an external debugger.

EDAD	Meaning
0b0	Access to debug registers from external debugger is enabled.
0b1	Access to breakpoint and watchpoint registers from external debugger is disabled, unless overridden by the IMPLEMENTATION DEFINED authentication interface. It is IMPLEMENTATION DEFINED whether this disable applies to the external register OSLAR_EL1.

If EL3 and EL2 are not implemented and the Effective value of SCR_EL3.NS is 0b0, then the Effective value of this field is 0b1.

On a Warm reset, this field resets to 0.

TTRF, bit [19]

When ARMv8.4-Trace is implemented:

Trap Trace Filter controls. Traps use of the Trace Filter control registers at EL2 and EL1 to EL3.

TTRF	Meaning
0b0	Accesses to TRFCR_EL2, TRFCR_EL12, TRFCR_EL1, HTRFCR and TRFCR registers at EL2 and EL1 are not affected by this control.
0b1	Accesses to TRFCR_EL2, TRFCR_EL12, TRFCR_EL1, HTRFCR and TRFCR registers at EL2 and EL1 generate a Trap exception to EL3.

Otherwise:

Reserved, RES0.

STE, bit [18]

When ARMv8.4-Trace is implemented:

Secure Trace enable. Enables tracing in Secure state.

STE	Meaning
0b0	Trace prohibited in Secure state unless overridden by the external debugger.
0b1	Trace allowed in Secure state unless prohibited by the Trace Filter control registers.

This bit also controls the level of authentication required by an external debugger to enable external tracing. If EL3 is not implemented and the PE is executing in Secure state, the Effective value of this bit is 0b1.

On a Warm reset, this field resets to 0.

Otherwise:

Reserved, RES0.

SPME, bit [17]

When ARMv8.2-Debug is implemented and PMUv3 is implemented:

Secure Performance Monitors enable. This allows event counting in Secure state.

SPME	Meaning
0b0	Event counting prohibited in Secure state.
0b1	Event counting allowed in Secure state.

If EL3 is not implemented and the PE is executing in Secure state, then the Effective value of this bit is 0b1.

On a Warm reset, this field resets to 0.

When PMUv3 is implemented:

Secure Performance Monitors enable. This allows event counting in Secure state.

SPME	Meaning
0b0	Event counting prohibited in Secure state, unless ExternalSecureNoninvasiveDebugEnabled() is TRUE.
0b1	Event counting allowed in Secure state.

If EL3 is not implemented and the PE is executing in Secure state, then the Effective value of this bit is 0b1.

On a Warm reset, this field resets to 0.

Otherwise:

Reserved, RES0.

SDD, bit [16]

AArch64 Secure self-hosted invasive debug disable. Disables Software debug exceptions in Secure state, other than Breakpoint Instruction exceptions.

SDD	Meaning
0b0	Debug exceptions from Secure EL0 are enabled, and debug exceptions from Secure EL1 are enabled if the value of MDSCR_EL1.KDE is 1 and the value of PSTATE.D is 0.
0b1	Debug exceptions, other than Breakpoint Instruction exceptions, are disabled from all Exception levels in Secure state.

The SDD bit is ignored unless both of the following are true:

- The PE is in Secure state.
- The Effective value of SCR_EL3.RW is 0b1.
- ~~The Effective value of SCR_EL3.RW is 0b0.~~

On a Warm reset, this field resets to an architecturally UNKNOWN value.

SPD32, bits [15:14]

AArch32 Secure self-hosted privileged invasive debug control. Enables or disables debug exceptions from Secure EL1 using AArch32, other than Breakpoint Instruction exceptions. ~~Valid values for this field are:~~

SPD32	Meaning
0b00	Legacy mode. Debug exceptions from Secure EL1 are enabled by the IMPLEMENTATION DEFINED authentication interface.
0b10	Secure privileged debug disabled. Debug exceptions from Secure EL1 are disabled.
0b11	Secure privileged debug enabled. Debug exceptions from Secure EL1 are enabled.

Other values are reserved, and have the CONSTRAINED UNPREDICTABLE behavior that they must have the same behavior as 0b00. Software must not rely on this property as the behavior of reserved values might change in a future revision of the architecture.

This field has no effect on Breakpoint Instruction exceptions. These are always enabled.

This field is:

- Ignored if the PE is either:
  - In Non-secure state.
  - In Secure state and the Effective value of SCR_EL3.RW is 0b1.
  - In Secure state, if the Effective value of SCR_EL3.RW is 0b1.
- RES0 if the implementation does not support EL1 using AArch32.

If Secure EL1 is using AArch32 then:

- If debug exceptions from Secure EL1 are enabled, then debug exceptions from Secure EL0 are also enabled.
- Otherwise, debug exceptions from Secure EL0 are enabled only if the value of SDER32_EL3.SUIDEN is 1.

If EL3 and EL2 are not implemented and the Effective value of SCR_EL3.NS is 0b0, then the Effective value of this field is 0b11.

On a Warm reset, this field resets to an architecturally UNKNOWN value.

NSPB, bits [13:12]

When SPE is implemented:

Non-secure Profiling Buffer. This field controls the owning translation regime and accesses to Statistical Profiling and Profiling Buffer control registers. The possible values of this field are:

NSPB	Meaning
0b00	Profiling Buffer uses Secure Virtual Addresses. Statistical Profiling enabled in Secure state and disabled in Non-secure state. Accesses to Statistical Profiling and Profiling Buffer controls at EL2 and EL1 in both security states generate Trap exceptions to EL3.
0b01	Profiling Buffer uses Secure Virtual Addresses. Statistical Profiling enabled in Secure state and disabled in Non-secure state. Accesses to Statistical Profiling and Profiling Buffer controls in Non-secure state generate Trap exceptions to EL3.
0b10	Profiling Buffer uses Non-secure Virtual Addresses. Statistical Profiling enabled in Non-secure state and disabled in Secure state. Accesses to Statistical Profiling and Profiling Buffer controls at EL2 and EL1 in both security states generate Trap exceptions to EL3.
0b11	Profiling Buffer uses Non-secure Virtual Addresses. Statistical Profiling enabled in Non-secure state and disabled in Secure state. Accesses to Statistical Profiling and Profiling Buffer controls in Secure state generate Trap exceptions to EL3.

If EL3 is not implemented and the PE is executing in Non-secure state, the Effective value of this field is 0b11.

If EL3 is not implemented and the PE is executing in Secure state, the Effective value of this field is 0b01.

Otherwise:

Reserved, RES0.

Bit [11]

Reserved, RES0.

TDOSA, bit [10]

When ARMv8.0-DoubleLock is implemented:

Trap debug OS-related register access. Traps EL2 and EL1 System register accesses to the powerdown debug registers to EL3.

TDOSA	Meaning
0b0	This control does not cause any instructions to be trapped.
0b1	EL2 and EL1 System register accesses to the powerdown debug registers are trapped to EL3, unless it is trapped by HDCR.TDOSA or MDCR_EL2.TDOSA.

The registers for which accesses are trapped are as follows:

AArch64: OSLAR_EL1, OSLSR_EL1, OSDLR_EL1, and DBGPRCR_EL1.

AArch32: DBGOSLAR, DBGOSLSR, DBGOSDLR, and DBGPRCR.

AArch64 and AArch32: Any IMPLEMENTATION DEFINED register with similar functionality that the implementation specifies as trapped by this bit.

On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Trap debug OS-related register access. Traps EL2 and EL1 System register accesses to the powerdown debug registers to EL3.

TDOSA	Meaning
0b0	This control does not cause any instructions to be trapped.
0b1	EL2 and EL1 System register accesses to the powerdown debug registers are trapped to EL3, unless it is trapped by HDCR.TDOSA or MDCR_EL2.TDOSA.

The registers for which accesses are trapped are as follows:

AArch64: OSLAR_EL1, OSLSR_EL1, and DBGPRCR_EL1.

AArch32: DBGOSLAR, DBGOSLSR, and DBGPRCR.

AArch64 and AArch32: Any IMPLEMENTATION DEFINED register with similar functionality that the implementation specifies as trapped by this bit.

It is IMPLEMENTATION DEFINED whether accesses to OSDLR_EL1 and DBGOSDLR are trapped.

On a Warm reset, this field resets to an architecturally UNKNOWN value.

TDA, bit [9]

Trap Debug Access. Traps EL2, EL1, and EL0 System register accesses to those debug System registers that cannot be trapped using the MDCR_EL3.TDOSA field. When MDCR_EL3.TDA is:

TDA	Meaning
0b0	This control does not cause any instructions to be trapped.
0b1	EL0, EL1, and EL2 accesses to the debug registers, other than the registers that can be trapped by MDCR_EL3.TDOSA, are trapped to EL3, from both Security states and both Execution states, unless it is trapped by DBGDSCRext.UDCCdis, MDSCR_EL1.TDCC, HDCR.TDA or MDCR_EL2.TDA.

Traps of AArch32 accesses to DBGDTRRXint and DBGDTRTXint are ignored in Debug state.

Traps of AArch64 accesses to DBGDTR_EL0, DBGDTRRX_EL0, and DBGDTRTX_EL0 are ignored in Debug state.

On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [8:7]

Reserved, RES0.

TPM, bit [6]

When PMUv3 is implemented:

Trap Performance Monitors accesses. Traps EL2, EL1, and EL0 accesses to all Performance Monitors registers to EL3, from both Security states and both Execution states.

TPM	Meaning
0b0	This control does not cause any instructions to be trapped.
0b1	EL2, EL1, and EL0 System register accesses to all Performance Monitors registers are trapped to EL3, unless it is trapped by HDCR.TPM or MDCR_EL2.TPM.

On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [5:0]

Reserved, RES0.

Accessing the MDCR_EL3

Accesses to this register use the following encodings:

```
MRS <Xt>, MDCR_EL3
```

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b11	0b110	0b0001	0b0011	0b001
~~0b11~~	~~0b0001~~	~~0b110~~	~~0b001~~	~~0b0011~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    return MDCR_EL3;
```

```
MSR MDCR_EL3, <Xt>
```

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b11	0b110	0b0001	0b0011	0b001
~~0b11~~	~~0b0001~~	~~0b110~~	~~0b001~~	~~0b0011~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    MDCR_EL3 = X[t];
```

27~~13~~/03~~12~~/2019~~2018~~ 21~~16~~:59~~42~~; e5e4db499bf9867a4b93324c4dbac985d3da9376~~6379d01e197f1d40720d32d0f84c419e9187e009~~

(old) htmldiff from- (new)

MDSCR_EL1, Monitor Debug System Control Register

The MDSCR_EL1 characteristics are:

Purpose

Main control register for the debug implementation.

Configuration

AArch64 System register MDSCR_EL1 bits [31:0] are architecturally mapped to AArch32 System register DBGDSCRext[31:0] .

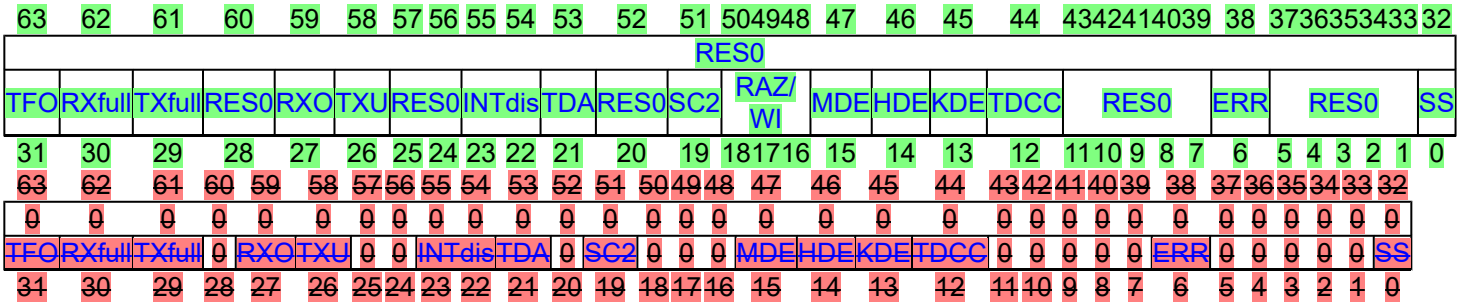
This register is in the Warm reset domain. Some or all RW fields of this register have defined reset values. On a Warm or Cold reset these apply only if the PE resets into an Exception level that is using AArch64. Otherwise, on a Warm or Cold reset RW fields in this register reset to architecturally UNKNOWN values.

Attributes

MDSCR_EL1 is a 64-bit register.

Field descriptions

The MDSCR_EL1 bit assignments are:



Bits [63:32]

Reserved, RES0.

TFO, bit [31]

When ARMv8.4-Trace is implemented:

Trace Filter override. Used for save/restore of EDSCR.TFO.

When the OS Lock is unlocked, OSLSR_EL1.OSLK == 0, this bit ignores writes, and software must treat it as UNK/SBZP.

When the OS Lock is locked, OSLSR_EL1.OSLK == 1, this bit is RW, and holds the value of EDSCR.TFO.

Reads and writes of this bit are indirect accesses to EDSCR.TFO.

Otherwise:

Reserved, RES0.

RXfull, bit [30]

Used for save/restore of EDSCR.RXfull.

When OSLSR_EL1.OSLK == 0, this bit is RO, and software must treat it as UNK/SBZP.

When OSLSR_EL1.OSLK == 1, this bit is RW and holds the value of EDSCR.RXfull.

Reads and writes of this bit are indirect accesses to EDSCR.RXfull.

The architected behavior of this field determines the value it returns after a reset.

TXfull, bit [29]

Used for save/restore of EDSCR.TXfull.

When OSLSR_EL1.OSLK == 0, this bit is RO, and software must treat it as UNK/SBZP.

When OSLSR_EL1.OSLK == 1, this bit is RW and holds the value of EDSCR.TXfull.

Reads and writes of this bit are indirect accesses to EDSCR.TXfull.

The architected behavior of this field determines the value it returns after a reset.

Bit [28]

Reserved, RES0.

RXO, bit [27]

Used for save/restore of EDSCR.RXO.

When OSLSR_EL1.OSLK == 0, this bit is RO, and software must treat it as UNK/SBZP.

When OSLSR_EL1.OSLK == 1, this bit is RW and holds the value of EDSCR.RXO.

Reads and writes of this bit are indirect accesses to EDSCR.RXO.

The architected behavior of this field determines the value it returns after a reset.

TXU, bit [26]

Used for save/restore of EDSCR.TXU.

When OSLSR_EL1.OSLK == 0, this bit is RO, and software must treat it as UNK/SBZP.

When OSLSR_EL1.OSLK == 1, this bit is RW and holds the value of EDSCR.TXU.

Reads and writes of this bit are indirect accesses to EDSCR.TXU.

The architected behavior of this field determines the value it returns after a reset.

Bits [25:24]

Reserved, RES0.

INTdis, bits [23:22]

Used for save/restore of EDSCR.INTdis.

When OSLSR_EL1.OSLK == 0, this field is RO, and software must treat it as UNK/SBZP.

When OSLSR_EL1.OSLK == 1, this field is RW and holds the value of EDSCR.INTdis.

Reads and writes of this field are indirect accesses to EDSCR.INTdis.

The architected behavior of this field determines the value it returns after a reset.

TDA, bit [21]

Used for save/restore of EDSCR.TDA.

When OSLSR_EL1.OSLK == 0, this bit is RO, and software must treat it as UNK/SBZP.

When OSLSR_EL1.OSLK == 1, this bit is RW and holds the value of EDSCR.TDA.

Reads and writes of this bit are indirect accesses to EDSCR.TDA.

The architected behavior of this field determines the value it returns after a reset.

Bit [20]

Reserved, RES0.

SC2, bit [19]

When ARMv8.0-PCSample is implemented, ARMv8.1-VHE is implemented and ARMv8.2-PCSampleARMv8.1-VHE is not implemented:

UsedIf ARMv8.2-PCSample is not implemented, used for save/restore of EDSCR.SC2.

When OSLSR_EL1.OSLK == 0, this bit is RO, and software must treat it as UNK/SBZP.

When OSLSR_EL1.OSLK == 1, this bit is RW and holds the value of EDSCR.SC2.

Reads and writes of this bit are indirect accesses to EDSCR.SC2.

Otherwise:

Reserved, RES0.

Bits [18:16]

ReservedRAZ/WI. Hardware must implement this field as RAZ/WI. Software must not rely on the register reading as zero, and must use a read-modify-write sequence to write to the register. RAZ/WI.

Hardware must implement this field as RAZ/WI. Software must not rely on the register reading as zero, and must use a read-modify-write sequence to write to the register.

MDE, bit [15]

Monitor debug events. Enable Breakpoint, Watchpoint, and Vector Catch exceptions.

MDE	Meaning
0b0	Breakpoint, Watchpoint, and Vector Catch exceptions disabled.
0b1	Breakpoint, Watchpoint, and Vector Catch exceptions enabled.

On a Warm reset, this field resets to an architecturally UNKNOWN value.

HDE, bit [14]

Used for save/restore of EDSCR.HDE.

When OSLSR_EL1.OSLK == 0, this bit is RO, and software must treat it as UNK/SBZP.

When OSLSR_EL1.OSLK == 1, this bit is RW and holds the value of EDSCR.HDE.

Reads and writes of this bit are indirect accesses to EDSCR.HDE.

The architected behavior of this field determines the value it returns after a reset.

KDE, bit [13]

Local (kernel) debug enable. If EL_D is using AArch64, enable debug exceptions within EL_D. Permitted values are:

KDE	Meaning
0b0	Debug exceptions, other than Breakpoint Instruction exceptions, disabled within EL_D.
0b1	All debug exceptions enabled within EL_D.

RES0 if EL_D is using AArch32.

On a Warm reset, this field resets to an architecturally UNKNOWN value.

TDCC, bit [12]

Traps EL0 accesses to the DCC registers to EL1, from both Execution states.

TDCC	Meaning
0b0	This control does not cause any instructions to be trapped.
0b1	EL0 using AArch64: EL0 accesses to the MDCCSR_EL0, DBGDTR_EL0, DBGDTRTX_EL0, and DBGDTRRX_EL0 registers are trapped to EL1. EL0 using AArch32: EL0 accesses to the DBGDSCRint, DBGDTRRXint, DBGDTRTXint, DBGDIDR, DBGDSAR, and DBGDRAR registers are trapped to EL1.

Note

All accesses to these AArch32 registers are trapped, including LDC and STC accesses to DBGDTRTXint and DBGDTRRXint, and MRRC accesses to DBGDSAR and DBGDRAR.

Traps of AArch32 accesses to the DBGDTRRXint and DBGDTRTXint are ignored in Debug state.

Traps of AArch64 accesses to DBGDTR_EL0, DBGDTRRX_EL0, and DBGDTRTX_EL0 are ignored in Debug state.

On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [11:7]

Reserved, RES0.

ERR, bit [6]

Used for save/restore of EDSCR.ERR.

When OSLSR_EL1.OSLK == 0, this bit is RO, and software must treat it as UNK/SBZP.

When OSLSR_EL1.OSLK == 1, this bit is RW and holds the value of EDSCR.ERR.

Reads and writes of this bit are indirect accesses to EDSCR.ERR.

The architected behavior of this field determines the value it returns after a reset.

Bits [5:1]

Reserved, RES0.

SS, bit [0]

Software step control bit. If EL_D is using AArch64, enable Software step. Permitted values are:

SS	Meaning
0b0	Software step disabled
0b1	Software step enabled.

RES0 if EL_D is using AArch32.

On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing the MDSCR_EL1

Individual fields within this register might have restricted accessibility when OSLSR_EL1.OSLK == 0 (the OS lock is unlocked). See the field descriptions for more detail.

Accesses to this register use the following encodings:

```
MRS <Xt>, MDSCR_EL1
```

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b10	0b000	0b0000	0b0010	0b010
~~0b10~~	~~0b0000~~	~~0b000~~	~~0b010~~	~~0b0010~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.<TDE,TDA> != '00' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<NV2,NV1,NV> == '1x1' then
        return NVMem[0x158];
    else
        return MDSCR_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return MDSCR_EL1;
elsif PSTATE.EL == EL3 then
    return MDSCR_EL1;
```

```
MSR MDSCR_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b10	0b000	0b0000	0b0010	0b010
~~0b10~~	~~0b0000~~	~~0b000~~	~~0b010~~	~~0b0010~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.<TDE,TDA> != '00' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<NV2,NV1,NV> == '1x1' then
        NVMem[0x158] = X[t];
    else
        MDSCR_EL1 = X[t];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        MDSCR_EL1 = X[t];
elsif PSTATE.EL == EL3 then
    MDSCR_EL1 = X[t];
```

27~~13~~/03~~12~~/2019~~2018~~ 21~~16~~:59~~42~~; e5e4db499bf9867a4b93324c4dbac985d3da93766~~379d01e197f1d40720d32d0f84e419e9187e009~~

(old) htmldiff from- (new)

(old) htmldiff from- (new)

MIDR_EL1, Main ID Register

The MIDR_EL1 characteristics are:

Purpose

Provides identification information for the PE, including an implementer code for the device and a device ID number.

Configuration

AArch64 System register MIDR_EL1 bits [31:0] are architecturally mapped to AArch32 System register MIDR[31:0] .

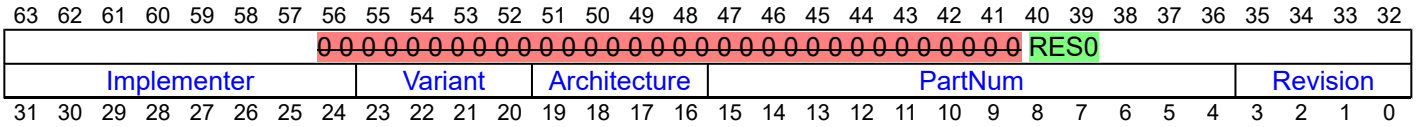
AArch64 System register MIDR_EL1 bits [31:0] are architecturally mapped to External register MIDR_EL1[31:0] .

Attributes

MIDR_EL1 is a 64-bit register.

Field descriptions

The MIDR_EL1 bit assignments are:



Bits [63:32]

Reserved, RES0.

Implementer, bits [31:24]

The Implementer code. This field must hold an implementer code that has been assigned by Arm. Assigned codes include the following:

Hex representation	Implementer
0x00~~0x0~~	Reserved for software use
0xC0	Ampere Computing
0x41	Arm Limited
0x42	Broadcom Corporation
0x43	Cavium Inc.
0x44	Digital Equipment Corporation
0x49	Infineon Technologies AG
0x4D	Motorola or Freescale Semiconductor Inc.
0x4E	NVIDIA Corporation
0x50	Applied Micro Circuits Corporation
0x51	Qualcomm Inc.
0x56	Marvell International Ltd.
0x69	Intel Corporation

Arm can assign codes that are not published in this manual. All values not assigned by Arm are reserved and must not be used.

Variant, bits [23:20]

An IMPLEMENTATION DEFINED variant number. Typically, this field is used to distinguish between different product variants, or major revisions of a product.

Architecture, bits [19:16]

The permitted values of this field are:

Architecture	Meaning
0b0001	Armv4.
0b0010	Armv4T.
0b0011	Armv5 (obsolete).
0b0100	Armv5T.
0b0101	Armv5TE.
0b0110	Armv5TEJ.
0b0111	Armv6.
0b1111	Architectural features are individually identified in the ID_* registers, see 'ID registers' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section K12.3.3.

All other values are reserved.

PartNum, bits [15:4]

An IMPLEMENTATION DEFINED primary part number for the device.

On processors implemented by Arm, if the top four bits of the primary part number are 0x0 or 0x7, the variant and architecture are encoded differently.

Revision, bits [3:0]

An IMPLEMENTATION DEFINED revision number for the device.

Accessing the MIDR_EL1

Accesses to this register use the following encodings:

```
MRS <Xt>, MIDR_EL1
```

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b11	0b000	0b0000	0b0000	0b000
~~0b11~~	~~0b0000~~	~~0b000~~	~~0b000~~	~~0b0000~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) then
        return VPIDR_EL2;
    else
        return MIDR_EL1;
elsif PSTATE.EL == EL2 then
    return MIDR_EL1;
elsif PSTATE.EL == EL3 then
    return MIDR_EL1;
```

27~~13~~/03~~12~~/2019~~2018~~ 21~~16~~:59~~42~~; e5e4db499bf9867a4b93324c4dbac985d3da9376~~6379d01e197f1d40720d32d0f84c419e9187e009~~

(old) htmldiff from- (new)

(old) htmldiff from- (new)

MPAM1_EL1, MPAM1 Register (EL1)

The MPAM1_EL1 characteristics are:

Purpose

MPAM1_EL1 holds information to generate MPAM labels for memory requests when executing at EL1.

When EL2 is present and enabled, the MPAM virtualization option is present, MPAMHCR_EL2.GSTAPP_PLK == 1 and HCR_EL2.TGE == 0, MPAM1_EL1 is used instead of MPAM0_EL1 to generate MPAM labels for memory requests when executing at EL0.

MPAM1_EL1 is an alias for MPAM2_EL2 when executing at EL2 with HCR_EL2.E2H == 1.

MPAM1_EL12 is an alias for MPAM1_EL1 when executing at EL2 or EL3 with HCR_EL2.E2H == 1.

If EL2 is is present and enabled, the MPAM virtualization option is present and MPAMHCR_EL2.EL1_VPMEN == 1, MPAM PARTIDs in MPAM1_EL1 are virtual and mapped into physical PARTIDs for the current Security state. This mapping of MPAM1_EL1 virtual PARTIDs to physical PARTIDs when EL1_VPMEN is 1 also applies when MPAM1_EL1 is used at EL0 due to MPAMHCR_EL2.GSTAPP_PLK.

Configuration

AArch64 System register MPAM1_EL1 bit [63] is architecturally mapped to AArch64 System register MPAM3_EL3[63] when HaveEL(EL3).

AArch64 System register MPAM1_EL1 bit [63] is architecturally mapped to AArch64 System register MPAM2_EL2[63] when !HaveEL(EL3) and HaveEL(EL2).

This register is present only when MPAM is implemented. Otherwise, direct accesses to MPAM1_EL1 are UNDEFINED.

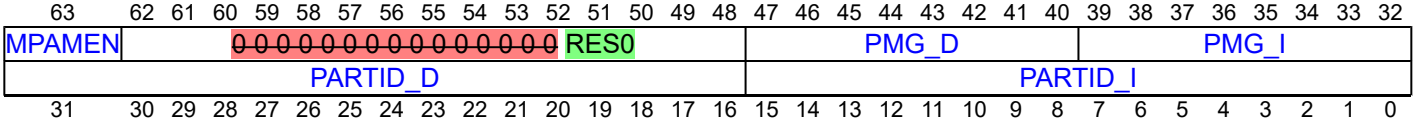
Some or all RW fields of this register have defined reset values. These apply only if the PE resets into an Exception level that is using AArch64. Otherwise, RW fields in this register reset to architecturally UNKNOWN values.

Attributes

MPAM1_EL1 is a 64-bit register.

Field descriptions

The MPAM1_EL1 bit assignments are:



MPAMEN, bit [63]

MPAM Enable: MPAM is enabled when MPAMEN == 1. When disabled, all PARTIDs and PMGs are output as their default value in the corresponding ID space.

Values of this field are:

MPAMEN	Meaning
0b0	The default PARTID and default PMG are output in MPAM information.
0b1	MPAM information is output based on the MPAMn_ELx register for ELn according the MPAM configuration.

If neither EL3 nor EL2 is implemented, this field is read/write.

If EL3 is implemented, this field is read-only and reads the current value of the read/write bit MPAM3_EL3.MPAMEN.

If EL3 is not implemented and EL2 is implemented, this field is read-only and reads the current value of the read/write bit MPAM2_EL2.MPAMEN.

This field resets to `0`.

Accessing this field has the following behavior:

- When !HaveEL(EL3) and !HaveEL(EL2), access to this field is **RW**.
- Otherwise, access to this field is **RO**.

Bits [62:48]

Reserved, RES0.

PMG_D, bits [47:40]

Performance monitoring group property for PARTID_D.

This field resets to an architecturally UNKNOWN value.

PMG_I, bits [39:32]

Performance monitoring group property for PARTID_I.

This field resets to an architecturally UNKNOWN value.

PARTID_D, bits [31:16]

Partition ID for data accesses, including load and store accesses, made from EL1.

This field resets to an architecturally UNKNOWN value.

PARTID_I, bits [15:0]

Partition ID for instruction accesses made from EL1.

This field resets to an architecturally UNKNOWN value.

Accessing the MPAM1_EL1

When HCR_EL2.E2H is 1, without explicit synchronization, accesses from EL3 using the mnemonic MPAM1_EL1 or MPAM1_EL12 are not guaranteed to be ordered with respect to accesses using the other mnemonic.

None of the fields in this register are permitted to be cached in a TLB.

Accesses to this register use the following encodings:

```
MRS <Xt>, MPAM1_EL1
```

op0	op1	CRn	CRm	op2
~~CRn~~	~~op0~~	~~op1~~	~~op2~~	~~CRm~~
0b11	0b000	0b1010	0b0101	0b000
~~0b1010~~	~~0b11~~	~~0b000~~	~~0b000~~	~~0b0101~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && !ELUsingAArch32(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && MPAM2_EL2.TRAPMPAM1EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<NV2,NV1,NV> == '111' then
        return NVMem[0x900];
    else
        return MPAM1_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HCR_EL2.E2H == '1' then
        return MPAM2_EL2;
    else
        return MPAM1_EL1;
elsif PSTATE.EL == EL3 then
    return MPAM1_EL1;
```

MSR MPAM1_EL1, <Xt>

op0	op1	CRn	CRm	op2
~~CRn~~	~~op0~~	~~op1~~	~~op2~~	~~CRm~~
0b11	0b000	0b1010	0b0101	0b000
~~0b1010~~	~~0b11~~	~~0b000~~	~~0b000~~	~~0b0101~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && !ELUsingAArch32(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && MPAM2_EL2.TRAPMPAM1EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<NV2,NV1,NV> == '111' then
        NVMem[0x900] = X[t];
    else
        MPAM1_EL1 = X[t];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HCR_EL2.E2H == '1' then
        MPAM2_EL2 = X[t];
    else
        MPAM1_EL1 = X[t];
elsif PSTATE.EL == EL3 then
    MPAM1_EL1 = X[t];
```

MRS <Xt>, MPAM1_EL12

op0	op1	CRn	CRm	op2
~~CRn~~	~~op0~~	~~op1~~	~~op2~~	~~CRm~~
0b11	0b101	0b1010	0b0101	0b000
~~0b1010~~	~~0b11~~	~~0b101~~	~~0b000~~	~~0b0101~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '101' then
        return NVMem[0x900];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        if HaveEL(EL3) && !ELUsingAArch32(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if EL2Enabled() && HCR_EL2.E2H == '1' then
    if PSTATE.EL == EL2 then
        if HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return MPAM1_EL1;
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if EL2Enabled() && HCR_EL2.E2H == '1' then
        return MPAM1_EL1;
else
    else
        UNDEFINED;
```

MSR MPAM1_EL12, <Xt>

op0	op1	CRn	CRm	op2
~~CRn~~	~~op0~~	~~op1~~	~~op2~~	~~CRm~~
0b11	0b101	0b1010	0b0101	0b000
~~0b1010~~	~~0b11~~	~~0b101~~	~~0b000~~	~~0b0101~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '101' then
        NVMem[0x900] = X[t];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        if HaveEL(EL3) && !ELUsingAArch32(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if EL2Enabled() && HCR_EL2.E2H == '1' then
    if PSTATE.EL == EL2 then
        if HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            MPAM1_EL1 = X[t];
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if EL2Enabled() && HCR_EL2.E2H == '1' then
        MPAM1_EL1 = X[t];
else
    else
        UNDEFINED;
```

(old) htmldiff from- (new)

(old) htmldiff from- (new)

OSDLR_EL1, OS Double Lock Register

The OSDLR_EL1 characteristics are:

Purpose

Used to control the OS Double Lock.

Configuration

AArch64 System register OSDLR_EL1 bits [31:0] are architecturally mapped to AArch32 System register DBGOSDLR[31:0] .

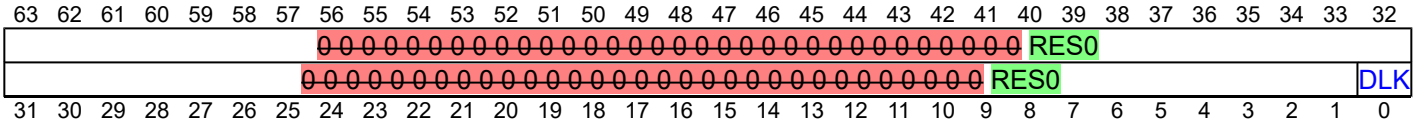
This register is in the Warm reset domain. Some or all RW fields of this register have defined reset values. On a Warm or Cold reset these apply only if the PE resets into an Exception level that is using AArch64. Otherwise, on a Warm or Cold reset RW fields in this register reset to architecturally UNKNOWN values.

Attributes

OSDLR_EL1 is a 64-bit register.

Field descriptions

The OSDLR_EL1 bit assignments are:



Bits [63:1]

Reserved, RES0.

DLK, bit [0]

When ARMv8.0-DoubleLock is implemented:

OS Double Lock control bit.

DLK	Meaning
0b0	OS Double Lock unlocked.
0b1	OS Double Lock locked, if DBGPRCR_EL1.CORENPDRQ (Core no powerdown request) bit is set to 0 and the PE is in Non-debug state.

On a Warm reset, this field resets to 0.

Otherwise:

Reserved, RAZ/WI.

Accessing the OSDLR_EL1

Accesses to this register use the following encodings:

MRS <Xt>, OSDLR_EL1

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b10	0b000	0b0001	0b0011	0b100
~~0b10~~	~~0b0001~~	~~0b000~~	~~0b100~~	~~0b0011~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.<TDE,TDOSA> != '00' &&
(IsFeatureImplemented(ARMv8.0-DoubleLock) || boolean IMPLEMENTATION_DEFINED "Trapped by
MDCR_EL2.TDOSA") then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDOSA == '1' &&
(IsFeatureImplemented(ARMv8.0-DoubleLock) || boolean IMPLEMENTATION_DEFINED "Trapped by
MDCR_EL3.TDOSA") then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return OSDLR_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDOSA == '1' &&
(IsFeatureImplemented(ARMv8.0-DoubleLock) || boolean IMPLEMENTATION_DEFINED "Trapped by
MDCR_EL3.TDOSA") then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return OSDLR_EL1;
elsif PSTATE.EL == EL3 then
    return OSDLR_EL1;
```

MSR OSDLR_EL1, <Xt>

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b10	0b000	0b0001	0b0011	0b100
~~0b10~~	~~0b0001~~	~~0b000~~	~~0b100~~	~~0b0011~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.<TDE,TDOSA> != '00' &&
(IsFeatureImplemented(ARMv8.0-DoubleLock) || boolean IMPLEMENTATION_DEFINED "Trapped by
MDCR_EL2.TDOSA") then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDOSA == '1' &&
(IsFeatureImplemented(ARMv8.0-DoubleLock) || boolean IMPLEMENTATION_DEFINED "Trapped by
MDCR_EL3.TDOSA") then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        OSDLR_EL1 = X[t];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDOSA == '1' &&
(IsFeatureImplemented(ARMv8.0-DoubleLock) || boolean IMPLEMENTATION_DEFINED "Trapped by
MDCR_EL3.TDOSA") then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        OSDLR_EL1 = X[t];
elsif PSTATE.EL == EL3 then
    OSDLR_EL1 = X[t];
```

2713/0312/20192018 2116:5942: e5e4db499bf9867a4b93324c4dbac985d3da93766379d01e197f1d40720d32d0f84c419e9187e009

(old) htmldiff from- (new)

(old) htmldiff from- (new)

PMBLIMITR_EL1, Profiling Buffer Limit Address Register

The PMBLIMITR_EL1 characteristics are:

Purpose

Defines the upper limit for the profiling buffer, and enables the profiling buffer

Configuration

This register is present only when SPE is implemented. Otherwise, direct accesses to PMBLIMITR_EL1 are UNDEFINED.

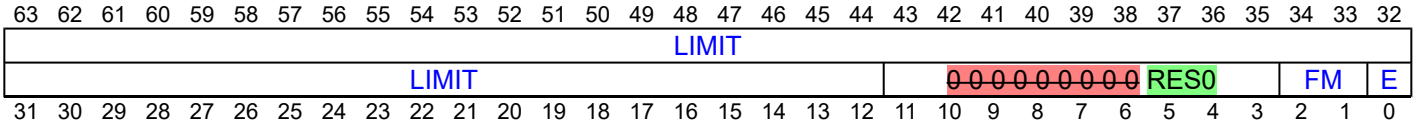
Some or all RW fields of this register have defined reset values. These apply only if the PE resets into an Exception level that is using AArch64. Otherwise, RW fields in this register reset to architecturally UNKNOWN values.

Attributes

PMBLIMITR_EL1 is a 64-bit register.

Field descriptions

The PMBLIMITR_EL1 bit assignments are:



LIMIT, bits [63:12]

Limit address. PMBLIMITR_EL1.LIMIT:Zeros(12) is the address of the first byte in memory after the last byte in the profiling buffer. If the smallest implemented translation granule is not 4KB, then bits[N-1:12] are RES0, where N is the IMPLEMENTATION DEFINED value, Log2(smallest implemented translation granule).

This field resets to an architecturally UNKNOWN value.

Bits [11:3]

Reserved, RES0.

FM, bits [2:1]

Fill mode

FM	Meaning
0b00	Stop collection and raise maintenance interrupt on buffer fill.

All other values are reserved. If this field is programmed with a reserved value, the PE behaves as if this field has a defined value, other than for a direct read of the register. Software must not rely on the behavior of reserved values, as they might change in a future version of the architecture.

This field resets to an architecturally UNKNOWN value.

E, bit [0]

Profiling Buffer enable

E	Meaning
0b0	All output is discarded.
0b1	Profiling buffer enabled.

On a Warm reset, this field resets to 0.

Accessing the PMBLIMITR_EL1

Accesses to this register use the following encodings:

MRS <Xt>, PMBLIMITR_EL1

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b11	0b000	0b1001	0b1010	0b000
~~0b11~~	~~0b1001~~	~~0b000~~	~~0b000~~	~~0b1010~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.E2PB == 'x0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.NS~~SCR_ELR.NS~~ == '0' && MDCR_EL3.NSPB !=
'01' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.NS~~SCR_ELR.NS~~ == '1' && MDCR_EL3.NSPB !=
'11' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<NV2,NV1,NV> == '1x1' then
        return NVMem[0x800];
    else
        return PMBLIMITR_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.NS~~SCR_ELR.NS~~ == '0' && MDCR_EL3.NSPB != '01'
then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.NS~~SCR_ELR.NS~~ == '1' && MDCR_EL3.NSPB !=
'11' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return PMBLIMITR_EL1;
elsif PSTATE.EL == EL3 then
    return PMBLIMITR_EL1;
```

MSR PMBLIMITR_EL1, <Xt>

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b11	0b000	0b1001	0b1010	0b000
~~0b11~~	~~0b1001~~	~~0b000~~	~~0b000~~	~~0b1010~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.E2PB == 'x0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.NSSCR_ELR.NS == '0' && MDCR_EL3.NSPB !=
'01' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.NSSCR_ELR.NS == '1' && MDCR_EL3.NSPB !=
'11' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<NV2,NV1,NV> == '1x1' then
        NVMem[0x800] = X[t];
    else
        PMBLIMITR_EL1 = X[t];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.NSSCR_ELR.NS == '0' && MDCR_EL3.NSPB != '01'
then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.NSSCR_ELR.NS == '1' && MDCR_EL3.NSPB !=
'11' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        PMBLIMITR_EL1 = X[t];
elsif PSTATE.EL == EL3 then
    PMBLIMITR_EL1 = X[t];
```

2713/0312/20192018 2116:5942: e5e4db499bf9867a4b93324c4dbac985d3da93766379d01e197f1d40720d32d0f84e419e9187e009

(old) htmldiff from- (new)

PMBPTR_EL1, Profiling Buffer Write Pointer Register

The PMBPTR_EL1 characteristics are:

Purpose

Defines the current write pointer for the profiling buffer.

Configuration

This register is present only when SPE is implemented. Otherwise, direct accesses to PMBPTR_EL1 are UNDEFINED.

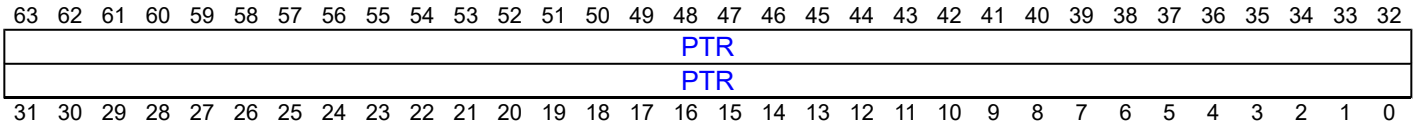
RW fields in this register reset to architecturally UNKNOWN values.

Attributes

PMBPTR_EL1 is a 64-bit register.

Field descriptions

The PMBPTR_EL1 bit assignments are:



PTR, bits [63:0]

Current write address. Defines the virtual address of the next entry to be written to the buffer.

The architecture places restrictions on the values software can write to the pointer. For more information see 'Restrictions on the current write pointer' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section D6.3.5.

Note

As a result, an implementation might treat some of bits[M:0], where M is defined by PMBIDR_EL1.Align, as RES0.

On a management interrupt, PMBPTR_EL1 is frozen.

On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing the PMBPTR_EL1

Accesses to this register use the following encodings:

```
MRS <Xt>, PMBPTR_EL1
```

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b11	0b000	0b1001	0b1010	0b001
~~0b11~~	~~0b1001~~	~~0b000~~	~~0b001~~	~~0b1010~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.E2PB == 'x0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.NSSCR_ELR.NS == '0' && MDCR_EL3.NSPB !=
'01' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.NSSCR_ELR.NS == '1' && MDCR_EL3.NSPB !=
'11' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<NV2,NV1,NV> == '1x1' then
        return NVMem[0x810];
    else
        return PMBPTR_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.NSSCR_ELR.NS == '0' && MDCR_EL3.NSPB != '01'
then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.NSSCR_ELR.NS == '1' && MDCR_EL3.NSPB !=
'11' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return PMBPTR_EL1;
elsif PSTATE.EL == EL3 then
    return PMBPTR_EL1;
```

```
MSR PMBPTR_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b11	0b000	0b1001	0b1010	0b001
~~0b11~~	~~0b1001~~	~~0b000~~	~~0b001~~	~~0b1010~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.E2PB == 'x0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.NSSCR_ELR.NS == '0' && MDCR_EL3.NSPB !=
'01' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.NSSCR_ELR.NS == '1' && MDCR_EL3.NSPB !=
'11' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<NV2,NV1,NV> == '1x1' then
        NVMem[0x810] = X[t];
    else
        PMBPTR_EL1 = X[t];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.NSSCR_ELR.NS == '0' && MDCR_EL3.NSPB != '01'
then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.NSSCR_ELR.NS == '1' && MDCR_EL3.NSPB !=
'11' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        PMBPTR_EL1 = X[t];
elsif PSTATE.EL == EL3 then
    PMBPTR_EL1 = X[t];
```

(old) htmldiff from- (new)

(old) htmldiff from- (new)

PMBSR_EL1, Profiling Buffer Status/syndrome Register

The PMBSR_EL1 characteristics are:

Purpose

Provides syndrome information to software when the buffer is disabled because the management interrupt has been raised.

Configuration

This register is present only when SPE is implemented. Otherwise, direct accesses to PMBSR_EL1 are UNDEFINED.

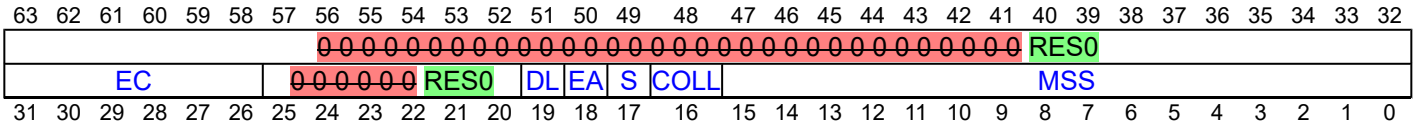
RW fields in this register reset to architecturally UNKNOWN values.

Attributes

PMBSR_EL1 is a 64-bit register.

Field descriptions

The PMBSR_EL1 bit assignments are:



Bits [63:32]

Reserved, RES0.

EC, bits [31:26]

Exception class

Top-level description of the cause of the buffer management event

EC	Meaning	MSS
0b100100	Stage 1 Data Abort on write to Profiling Buffer.	MSS encoding for a data Abort or stage 2 Data Abort on write to buffer.
0b100101	Stage 2 Data Abort on write to Profiling Buffer.	MSS encoding for a data Abort or stage 2 Data Abort on write to buffer.
0b000000	Other buffer management event. All buffer management events other than those described by other defined Exception class codes.	MSS encoding for an other buffer management event.

All other values are reserved. Reserved values might be defined in a future version of the architecture.

Writing a reserved value to this field will make the value of this field UNKNOWN. Values that are not supported act as reserved values when writing to this register.

On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [25:20]

Reserved, RES0.

DL, bit [19]

Partial record lost.

Following a buffer management event other than an asynchronous External abort, indicates whether the last record written to the Profiling Buffer is complete.

DL	Meaning
0b0	PMBPTR_EL1 points to the first byte after the last complete record written to the Profiling Buffer.
0b1	Part of a record was lost because of a buffer management event or synchronous External abort. PMBPTR_EL1 might not point to the first byte after the last complete record written to the buffer, and so restarting collection might result in a data record stream that software cannot parse. All records prior to the last record have been written to the buffer.

When the buffer management event was because of an asynchronous external abort, this bit is set to 1 and software must not assume that any valid data has been written to the Profiling Buffer.

This bit is RES0 if the PE never sets this bit as a result of a buffer management event caused by an asynchronous External abort.

On a Warm reset, this field resets to an architecturally UNKNOWN value.

EA, bit [18]

External abort.

EA	Meaning
0b0	An external abort has not been asserted.
0b1	An external abort has been asserted and detected by the Statistical Profiling Extension.

This bit is RES0 if the PE never sets this bit as the result of an External abort.

On a Warm reset, this field resets to an architecturally UNKNOWN value.

S, bit [17]

Service

S	Meaning
0b0	PMBIRQ is not asserted.
0b1	PMBIRQ is asserted. All profiling data has either been written to the buffer or discarded.

On a Warm reset, this field resets to an architecturally UNKNOWN value.

COLL, bit [16]

Collision detected.

COLL	Meaning
0b0	No collision events detected.
0b1	At least one collision event was recorded.

On a Warm reset, this field resets to an architecturally UNKNOWN value.

MSS, bits [15:0]

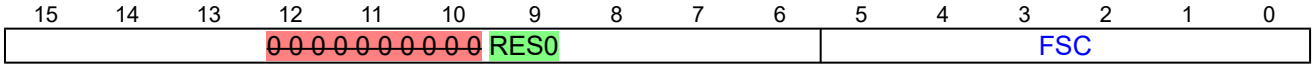
Management Event Specific Syndrome.

Contains syndrome specific to the management event.

The syndrome contents for each management event are described in the following sections.

On a Warm reset, this field resets to an architecturally UNKNOWN value.

MSS encoding for a data Abort or stage 2 Data Abort on write to buffer.



Bits [15:6]

Reserved, RES0.

FSC, bits [5:0]

Fault status code

FSC	Meaning
0b0000xx	Address Size fault. Bits [1:0] encode the level.
0b0001xx	Translation fault. Bits [1:0] encode the level.
0b0010xx	Access Flag fault. Bits [1:0] encode the level.
0b0011xx	Permission fault. Bits [1:0] encode the level.
0b010000	Synchronous External abort on write.
0b0101xx	Synchronous External abort on translation table walk or hardware update of translation table. Bits [1:0] encode the level.
0b010001	Asynchronous External abort on write.
0b100001	Alignment fault.
0b110000	TLB Conflict fault.

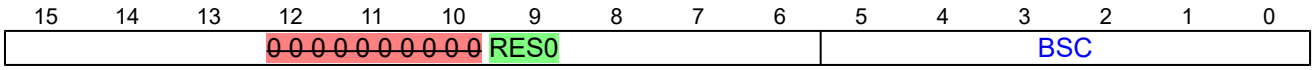
All other values are reserved. Reserved values might be defined in a future version of the architecture.

Writing a reserved value to this field will make the value of this field UNKNOWN. Values that are not supported act as reserved values when writing to this register.

It is IMPLEMENTATION DEFINED whether each of the Access Flag fault, asynchronous External abort and synchronous External abort, Alignment fault, and TLB Conflict abort values can be generated by the PE. For more information see Faults and Watchpoints.

On a Warm reset, this field resets to an architecturally UNKNOWN value.

MSS encoding for an other buffer management event.



Bits [15:6]

Reserved, RES0.

BSC, bits [5:0]

Buffer status code

BSC	Meaning
0b000000	Buffer not filled
0b000001	Buffer filled

All other values are reserved. Reserved values might be defined in a future version of the architecture.

Writing a reserved value to this field will make the value of this field UNKNOWN. Values that are not supported act as reserved values when writing to this register.

On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing the PMBSR_EL1

Accesses to this register use the following encodings:

MRS <Xt>, PMBSR_EL1

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b11	0b000	0b1001	0b1010	0b011
~~0b11~~	~~0b1001~~	~~0b000~~	~~0b011~~	~~0b1010~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.E2PB == 'x0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.NSSCR_ELR.NS == '0' && MDCR_EL3.NSPB !=
'01' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.NSSCR_ELR.NS == '1' && MDCR_EL3.NSPB !=
'11' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<NV2,NV1,NV> == '1x1' then
        return NVMem[0x820];
    else
        return PMBSR_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.NSSCR_ELR.NS == '0' && MDCR_EL3.NSPB != '01'
then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.NSSCR_ELR.NS == '1' && MDCR_EL3.NSPB !=
'11' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return PMBSR_EL1;
elsif PSTATE.EL == EL3 then
    return PMBSR_EL1;
```

MSR PMBSR_EL1, <Xt>

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b11	0b000	0b1001	0b1010	0b011
~~0b11~~	~~0b1001~~	~~0b000~~	~~0b011~~	~~0b1010~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.E2PB == 'x0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.NSSCR_ELR.NS == '0' && MDCR_EL3.NSPB !=
'01' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.NSSCR_ELR.NS == '1' && MDCR_EL3.NSPB !=
'11' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<NV2,NV1,NV> == '1x1' then
        NVMem[0x820] = X[t];
    else
        PMBSR_EL1 = X[t];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.NSSCR_ELR.NS == '0' && MDCR_EL3.NSPB != '01'
then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.NSSCR_ELR.NS == '1' && MDCR_EL3.NSPB !=
'11' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        PMBSR_EL1 = X[t];
elsif PSTATE.EL == EL3 then
    PMBSR_EL1 = X[t];
```

2713/0312/20192018 2116:5942: e5e4db499bf9867a4b93324c4dbac985d3da93766379d01e197f1d40720d32d0f84e419e9187e009

(old) htmldiff from- (new)

(old) htmldiff from- (new)

PMCCNTR_EL0, Performance Monitors Cycle Count Register

The PMCCNTR_EL0 characteristics are:

Purpose

Holds the value of the processor Cycle Counter, CCNT, that counts processor clock cycles. See 'Time as measured by the Performance Monitors cycle counter' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile~~, section D5~~ for more information.

PMCCFILTR_EL0 determines the modes and states in which the PMCCNTR_EL0 can increment.

Configuration

AArch64 System register PMCCNTR_EL0 bits [63:0] are architecturally mapped to AArch32 System register PMCCNTR[63:0] .

AArch64 System register PMCCNTR_EL0 bits [63:0] are architecturally mapped to External register PMCCNTR_EL0[63:0] .

All counters are subject to any changes in clock frequency, including clock stopping caused by the WFI and WFE instructions. This means that it is CONSTRAINED UNPREDICTABLE whether or not PMCCNTR_EL0 continues to increment when clocks are stopped by WFI and WFE instructions.

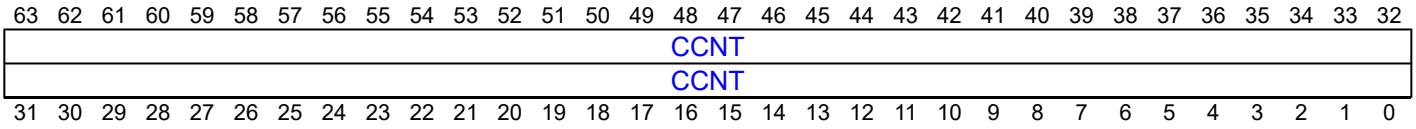
This register is in the Warm reset domain. On a Warm or Cold reset RW fields in this register reset to architecturally UNKNOWN values.

Attributes

PMCCNTR_EL0 is a 64-bit register.

Field descriptions

The PMCCNTR_EL0 bit assignments are:



CCNT, bits [63:0]

Cycle count. Depending on the values of PMCR_EL0.{LC,D}, this field increments in one of the following ways:

- Every processor clock cycle.
- Every 64th processor clock cycle.

Writing 1 to PMCR_EL0.C sets this field to 0.

On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing the PMCCNTR_EL0

Accesses to this register use the following encodings:

```
MRS <Xt>, PMCCNTR_EL0
```

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b11	0b011	0b1001	0b1101	0b000

~~0b11~~	~~0b1001~~	~~0b011~~	~~0b000~~	~~0b1101~~

```
if PSTATE.EL == EL0 then
    if !ELUsingAArch32(EL1) && PMUSERENR_EL0.<CR,EN> == '00' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return PMCCNTR_EL0;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return PMCCNTR_EL0;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return PMCCNTR_EL0;
elsif PSTATE.EL == EL3 then
    return PMCCNTR_EL0;
```

```
MSR PMCCNTR_EL0, <Xt>
```

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b11	0b011	0b1001	0b1101	0b000
~~0b11~~	~~0b1001~~	~~0b011~~	~~0b000~~	~~0b1101~~

```
if PSTATE.EL == EL0 then
    if !ELUsingAArch32(EL1) && PMUSERENR_EL0.EN == '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        PMCCNTR_EL0 = X[t];
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        PMCCNTR_EL0 = X[t];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        PMCCNTR_EL0 = X[t];
elsif PSTATE.EL == EL3 then
    PMCCNTR_EL0 = X[t];
```

2713/0312/20192018 2116:5942: e5e4db499bf9867a4b93324c4dbac985d3da93766379d01e197f1d40720d32d0f84e419e9187e009

(old) htmldiff from- (new)

(old) htmldiff from- (new)

PMCR_EL0, Performance Monitors Control Register

The PMCR_EL0 characteristics are:

Purpose

Provides details of the Performance Monitors implementation, including the number of counters implemented, and configures and controls the counters.

Configuration

AArch64 System register PMCR_EL0 bits [31:0] are architecturally mapped to AArch32 System register PMCR[31:0] .

AArch64 System register PMCR_EL0 bits [76:0] are architecturally mapped to External register PMCR_EL0[76:0] .

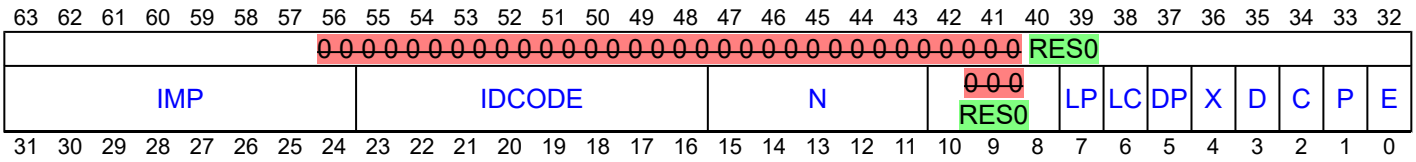
This register is in the Warm reset domain. Some or all RW fields of this register have defined reset values. On a Warm or Cold reset these apply only if the PE resets into an Exception level that is using AArch64. Otherwise, on a Warm or Cold reset RW fields in this register reset to architecturally UNKNOWN values.

Attributes

PMCR_EL0 is a 64-bit register.

Field descriptions

The PMCR_EL0 bit assignments are:



Bits [63:32]

Reserved, RES0.

IMP, bits [31:24]

Implementer code. This field is RO with an IMPLEMENTATION DEFINED value.

If this field is zero, then PMCR_EL0.IDCODE is RES0 and software must use the MIDR_EL1 to identify the PE.

Otherwise, this field and PMCR_EL0.IDCODE identifies the PMU implementation to software. The implementer codes are allocated by Arm. A non-zero value has the same interpretation as PMCR_EL0.IDCODE identify the PMU implementation to software. The implementer codes are allocated by Arm. A non-zero value has the same interpretation as MIDR_EL1.Implementer.

IDCODE, bits [23:16]

When PMCR_EL0.IMP != 0x00:

Identification code. This field is RO with an IMPLEMENTATION DEFINED value.

Each implementer must maintain a list of identification codes that areis specific to the implementer. A specific implementation is identified by the combination of the implementer code and the identification code.

Otherwise:

Reserved, RES0.

N, bits [15:11]

An RO field that indicates the number of event counters implemented. This value is in the range of `0b00000-0b111111`. If the value is `0b00000` then only PMCCNTR_EL0 is implemented. If the value is `0b111111` PMCCNTR_EL0 and 31 event counters are implemented.

When EL2 is implemented and enabled for the current Security state, reads of this field from EL1 and EL0 return the value of MDCR_EL2.HPMN.

Access to this field is **RO**.

Bits [10:8]

Reserved, RES0.

LP, bit [7]

When ARMv8.5-PMU is implemented:

Long event counter enable. Determines when unsigned overflow is recorded by a counter overflow bit.

LP	Meaning
`0b0`	Event counter overflow on increment that causes unsigned overflow of PMEVCNTR<n>_EL0[31:0].
`0b1`	Event counter overflow on increment that causes unsigned overflow of PMEVCNTR<n>_EL0[63:0].

If EL2 is implemented and MDCR_EL2.HPMN or HDCR.HPMN is less than PMCR_EL0.N, this bit does not affect the operation of event counters in the range [HDCR.HPMN:(PMCR_EL0.N-1)] or [MDCR_EL2.HPMN:(PMCR_EL0.N-1)].

Note

The effect of MDCR_EL2.HPMN or HDCR.HPMN on the operation of this bit applies if EL2 is implemented regardless of whether EL2 is enabled in the current Security state. For more information, see the description of MDCR_EL2.HPMN or HDCR.HPMN.

On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

LC, bit [6]

Long cycle counter enable. Determines when unsigned overflow is recorded by the cycle counter overflow bit.

LC	Meaning
`0b0`	Cycle counter overflow on increment that causes unsigned overflow of PMCCNTR_EL0[31:0].
`0b1`	Cycle counter overflow on increment that causes unsigned overflow of PMCCNTR_EL0[63:0].

Arm deprecates use of PMCR_EL0.LC = 0.

In an AArch64 only implementation, this field is RES1.

On a Warm reset, this field resets to an architecturally UNKNOWN value.

DP, bit [5]

Disable cycle counter when event counting is prohibited. The possible values of this bit are:

DP	Meaning
0b0	Cycle counting by PMCCNTR_EL0 is, notif affectedenabled, bycounts thiswhen bit.event counting is prohibited.
0b1	When event counting for counters in the range [0..(MDCR_EL2.HPMN-1)] is prohibited, cycle counting by PMCCNTR_EL0 does not count when event counting is disabled.prohibited.

Counting events is never prohibited in Non-secure state. However, there are some restrictions on counting events in Secure state. For more information about the interaction between the Performance Monitors and EL3, see 'EffectInteraction ofwith EL3 and EL2' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section D5.5.1.

When EL3 is not implemented, this field is RES0:

- When ARMv8.1-PMU is not implemented.
- When ARMv8.1-PMU is implemented, only if EL2 is not implemented.

Otherwise this field is RW.

On a Warm reset, this field resets to an architecturally UNKNOWN value.

X, bit [4]

Enable export of events in an IMPLEMENTATION DEFINED event stream. The possible values of this bit are:

X	Meaning
0b0	Do not export events.
0b1	Export events where not prohibited.

This field enables the exporting of events over an event bus to another device, for example to an OPTIONAL PE trace unit. If the implementation does not include such an event bus then this field is RAZ/WI, otherwise it is an RW field.

In an implementation that includes an event bus, no events are exported when counting is prohibited.

This field does not affect the generation of Performance Monitors overflow interrupt requests or signaling to a cross-trigger interface (CTI) that can be implemented as signals exported from the PE.

On a Warm reset, this field resets to an architecturally UNKNOWN value.

D, bit [3]

Clock divider. The possible values of this bit are:

D	Meaning
0b0	When enabled, PMCCNTR_EL0 counts every clock cycle.
0b1	When enabled, PMCCNTR_EL0 counts once every 64 clock cycles.

In an AArch64 only implementation this field is RES0, otherwise it is an RW field. If PMCR_EL0.LC == 1, this bit is ignored and the cycle counter counts every clock cycle.

Arm deprecates use of PMCR_EL0.D = 1.

On a Warm reset, this field resets to an architecturally UNKNOWN value.

C, bit [2]

Cycle counter reset. This bit is WO. The effects of writing to this bit are:

C	Meaning
0b0	No action.
0b1	Reset PMCCNTR_EL0 to zero.

This bit is always RAZ.

Note

Resetting PMCCNTR_EL0 does not change the cycle counter overflow bit.

P, bit [1]

Event counter reset. This bit is WO. The effects of writing to this bit are:

P	Meaning
0b0	No action.
0b1	Reset all event counters accessible in the current Exception level, not including PMCCNTR_EL0, to zero.

This bit is always RAZ.

In EL0 and EL1:

- If EL2 is implemented and enabled in the current Security state, and MDCR_EL2.HPMN is less than PMCR_EL0.N, a write of 1 to this bit does not reset event counters in the range [MDCR_EL2.HPMN:(PMCR_EL0.N-1)].
- If EL2 is not implemented, EL2 is disabled in the current Security state, or MDCR_EL2.HPMN equals PMCR_EL0.N, a write of 1 to this bit resets all the event counters.

In EL2 and EL3, a write of 1 to this bit resets all the event counters.

Note

Resetting the event counters does not change the event counter overflow bits.

If ARMv8.5-PMU is implemented, the value of MDCR_EL2.HLP, or PMCR_EL0.LP is ignored and bits [63:0] of all affected event counters are reset.

E, bit [0]

Enable.

E	Meaning
0b0	All event counters in the range [0:(PMN-1)] and PMCCNTR_EL0, are disabled.
0b1	All event counters in the range [0:(PMN-1)] and PMCCNTR_EL0, are enabled by PMCNTENSET_EL0.

This bit is RW.

If EL2 is implemented then:

- If EL2 is using AArch32, PMN is HDCR.HPMN.
- If EL2 is using AArch64, PMN is MDCR_EL2.HPMN.
- If PMN is less than PMCR_EL0.N, this bit does not affect the operation of event counters in the range [PMN:(PMCR_EL0.N-1)].

If EL2 is not implemented, PMN is PMCR_EL0.N.

Note

The effect of MDCR_EL2.HPMN or HDCR.HPMN on the operation of this bit applies if EL2 is implemented regardless of whether EL2 is enabled in the current Security state. For more information, see the description of MDCR_EL2.HPMN or HDCR.HPMN.

On a Warm reset, this field resets to 0.

Accessing the PMCR_EL0

Accesses to this register use the following encodings:

```
MRS <Xt>, PMCR_EL0
```

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b11	0b011	0b1001	0b1100	0b000
~~0b11~~	~~0b1001~~	~~0b011~~	~~0b000~~	~~0b1100~~

```
if PSTATE.EL == EL0 then
    if !ELUsingAArch32(EL1) && PMUSERENR_EL0.EN == '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPMCR == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return PMCR_EL0;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPMCR == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return PMCR_EL0;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return PMCR_EL0;
elsif PSTATE.EL == EL3 then
    return PMCR_EL0;
```

```
MSR PMCR_EL0, <Xt>
```

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b11	0b011	0b1001	0b1100	0b000
~~0b11~~	~~0b1001~~	~~0b011~~	~~0b000~~	~~0b1100~~

```
if PSTATE.EL == EL0 then
    if !ELUsingAArch32(EL1) && PMUSERENR_EL0.EN == '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPMCR == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        PMCR_EL0 = X[t];
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPMCR == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        PMCR_EL0 = X[t];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        PMCR_EL0 = X[t];
elsif PSTATE.EL == EL3 then
    PMCR_EL0 = X[t];
```

27~~13~~/03~~12~~/2019~~2018~~ 21~~16~~:59~~42~~; e5e4db499bf9867a4b93324c4dbac985d3da93766~~379d01e197f1d40720d32d0f84e419e9187e009~~

(old) htmldiff from- (new)

PMINTENCLR_EL1, Performance Monitors Interrupt Enable Clear register

The PMINTENCLR_EL1 characteristics are:

Purpose

Disables the generation of interrupt requests on overflows from the Cycle Count Register, PMCCNTR_EL0, and the event counters PMEVCNTR<n>_EL0. Reading the register shows which overflow interrupt requests are enabled.

Configuration

AArch64 System register PMINTENCLR_EL1 bits [31:0] are architecturally mapped to AArch32 System register PMINTENCLR[31:0] .

AArch64 System register PMINTENCLR_EL1 bits [31:0] are architecturally mapped to External register PMINTENCLR_EL1[31:0] .

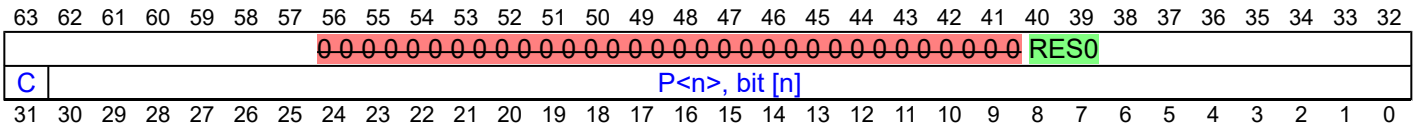
This register is in the Warm reset domain. On a Warm or Cold reset RW fields in this register reset to architecturally UNKNOWN values.

Attributes

PMINTENCLR_EL1 is a 64-bit register.

Field descriptions

The PMINTENCLR_EL1 bit assignments are:



Bits [63:32]

Reserved, RES0.

C, bit [31]

PMCCNTR_EL0 overflow interrupt request disable bit. Possible values are:

C	Meaning
0b0	When read, means the cycle counter overflow interrupt request is disabled. When written, has no effect.
0b1	When read, means the cycle counter overflow interrupt request is enabled. When written, disables the cycle count overflow interrupt request.

P<n>, bit [n], for n = 0 to 30

Event counter overflow interrupt request disable bit for PMEVCNTR<n>_EL0.

If N is less than 31, then bits [30:N] are RAZ/WI. When EL2 is implemented and enabled in the current Security state, in EL1 ~~and EL0~~, N is the value in MDCR_EL2.HPMN. Otherwise, N is the value in PMCR_EL0.N.

P<n>	Meaning
0b0	When read, means that the PMEVCNTR<n>_EL0 event counter interrupt request is disabled. When written, has no effect.
0b1	When read, means that the PMEVCNTR<n>_EL0 event counter interrupt request is enabled. When written, disables the PMEVCNTR<n>_EL0 interrupt request.

On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing the PMINTENCLR_EL1

Accesses to this register use the following encodings:

```
MRS <Xt>, PMINTENCLR_EL1
```

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b11	0b000	0b1001	0b1110	0b010
~~0b11~~	~~0b1001~~	~~0b000~~	~~0b010~~	~~0b1110~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return PMINTENCLR_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return PMINTENCLR_EL1;
elsif PSTATE.EL == EL3 then
    return PMINTENCLR_EL1;
```

```
MSR PMINTENCLR_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b11	0b000	0b1001	0b1110	0b010
~~0b11~~	~~0b1001~~	~~0b000~~	~~0b010~~	~~0b1110~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        PMINTENCLR_EL1 = X[t];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        PMINTENCLR_EL1 = X[t];
elsif PSTATE.EL == EL3 then
    PMINTENCLR_EL1 = X[t];
```

(old) htmldiff from- (new)

(old) htmldiff from- (new)

PMINTENSET_EL1, Performance Monitors Interrupt Enable Set register

The PMINTENSET_EL1 characteristics are:

Purpose

Enables the generation of interrupt requests on overflows from the Cycle Count Register, PMCCNTR_EL0, and the event counters PMEVCNTR<n>_EL0. Reading the register shows which overflow interrupt requests are enabled.

Configuration

AArch64 System register PMINTENSET_EL1 bits [31:0] are architecturally mapped to AArch32 System register PMINTENSET[31:0] .

AArch64 System register PMINTENSET_EL1 bits [31:0] are architecturally mapped to External register PMINTENSET_EL1[31:0] .

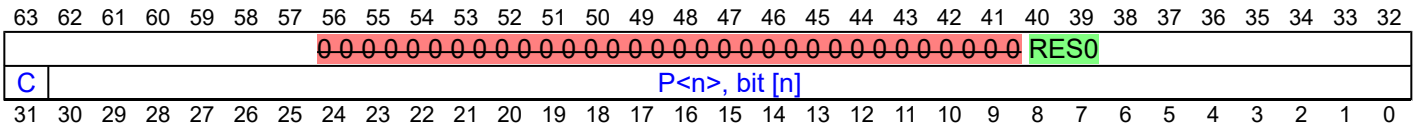
This register is in the Warm reset domain. On a Warm or Cold reset RW fields in this register reset to architecturally UNKNOWN values.

Attributes

PMINTENSET_EL1 is a 64-bit register.

Field descriptions

The PMINTENSET_EL1 bit assignments are:



Bits [63:32]

Reserved, RES0.

C, bit [31]

PMCCNTR_EL0 overflow interrupt request enable bit. Possible values are:

C	Meaning
0b0	When read, means the cycle counter overflow interrupt request is disabled. When written, has no effect.
0b1	When read, means the cycle counter overflow interrupt request is enabled. When written, enables the cycle count overflow interrupt request.

On a Warm reset, this field resets to an architecturally UNKNOWN value.

P<n>, bit [n], for n = 0 to 30

Event counter overflow interrupt request enable bit for PMEVCNTR<n>_EL0.

If N is less than 31, then bits [30:N] are RAZ/WI. When EL2 is implemented and enabled in the current Security state, in EL1 ~~and EL0~~, N is the value in MDCR_EL2.HPMN. Otherwise, N is the value in PMCR_EL0.N.

P<n>	Meaning
0b0	When read, means that the PMEVCNTR<n>_EL0 event counter interrupt request is disabled. When written, has no effect.
0b1	When read, means that the PMEVCNTR<n>_EL0 event counter interrupt request is enabled. When written, enables the PMEVCNTR<n>_EL0 interrupt request.

On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing the PMINTENSET_EL1

Accesses to this register use the following encodings:

```
MRS <Xt>, PMINTENSET_EL1
```

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b11	0b000	0b1001	0b1110	0b001
~~0b11~~	~~0b1001~~	~~0b000~~	~~0b001~~	~~0b1110~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return PMINTENSET_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return PMINTENSET_EL1;
elsif PSTATE.EL == EL3 then
    return PMINTENSET_EL1;
```

```
MSR PMINTENSET_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b11	0b000	0b1001	0b1110	0b001
~~0b11~~	~~0b1001~~	~~0b000~~	~~0b001~~	~~0b1110~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        PMINTENSET_EL1 = X[t];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        PMINTENSET_EL1 = X[t];
elsif PSTATE.EL == EL3 then
    PMINTENSET_EL1 = X[t];
```

2713/0312/20192018 2116:5942; e5e4db499bf9867a4b93324c4dbac985d3da93766379d01e197f1d40720d32d0f84e419e9187e009

(old) htmldiff from- (new)

(old) htmldiff from- (new)

PMSCR_EL1, Statistical Profiling Control Register (EL1)

The PMSCR_EL1 characteristics are:

Purpose

Provides EL1 controls for Statistical Profiling

Configuration

This register is present only when SPE is implemented. Otherwise, direct accesses to PMSCR_EL1 are UNDEFINED.

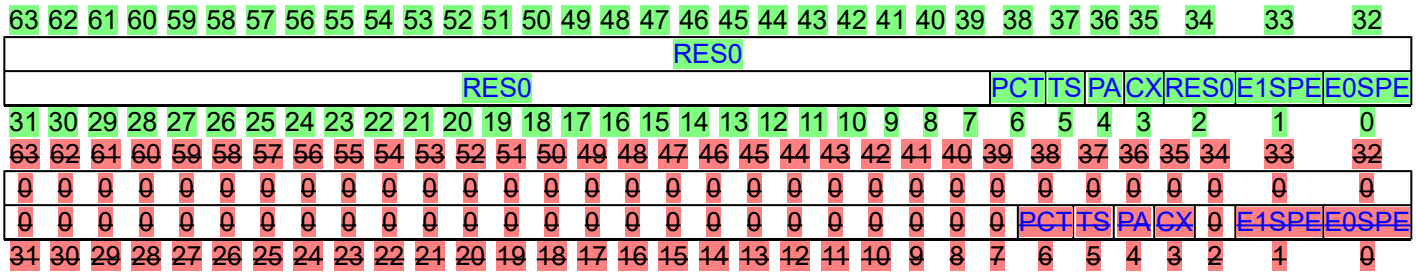
This register is in the Warm reset domain. On a Warm or Cold reset RW fields in this register reset to architecturally UNKNOWN values.

Attributes

PMSCR_EL1 is a 64-bit register.

Field descriptions

The PMSCR_EL1 bit assignments are:



Bits [63:7]

Reserved, RES0.

PCT, bit [6]

When HaveEL(EL2):

Physical Timestamp.

If timestamp sampling is enabled, determines which counter is collected.

PCT	Meaning
0b0	Virtual counter, CNTVCT_EL0, is collected.
0b1	Physical counter, CNTPCT_EL0, is collected.

If EL2 is implemented and enabled in the current Security state:

- If MDCR_EL2.E2PB != 0b00, this bit is combined with PMSCR_EL2.PCT to determine which counter is collected. For more information, see 'Controlling the data that is collected' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile
- If MDCR_EL2.E2PB == 0b00, this bit is ignored by the PE.

On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES1.

TS, bit [5]

Timestamp Enable.

TS	Meaning
0b0	Timestamp sampling disabled.
0b1	Timestamp sampling enabled.

If EL2 is implemented and enabled in the current Security state, this bit is ignored by the PE when MDCR_EL2.E2PB == 0b00.

On a Warm reset, this field resets to an architecturally UNKNOWN value.

PA, bit [4]

Physical Address Sample Enable.

PA	Meaning
0b0	Physical addresses are not collected.
0b1	Physical addresses are collected.

If EL2 is implemented and enabled in the current Security state:

- If MDCR_EL2.E2PB != 0b00, this bit is combined with PMSCR_EL2.PA to determine which address is collected. For more information, see 'Controlling the data that is collected' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.
- If MDCR_EL2.E2PB == 0b00, this bit is ignored by the PE.

On a Warm reset, this field resets to an architecturally UNKNOWN value.

CX, bit [3]

CONTEXTIDR_EL1 Sample Enable.

CX	Meaning
0b0	CONTEXTIDR_EL1 is not collected.
0b1	CONTEXTIDR_EL1 is collected.

If EL2 is implemented and enabled in the current Security state:

- If the PE is at EL2, this bit is ignored by the PE.
- If HCR_EL2.TGE == 1, this bit is ignored by the PE.

On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bit [2]

Reserved, RES0.

E1SPE, bit [1]

EL1 Statistical Profiling Enable.

E1SPE	Meaning
0b0	Sampling disabled at EL1.
0b1	Sampling enabled at EL1.

If EL2 is implemented and enabled in the current Security state, this bit is ignored by the PE when HCR_EL2.TGE == 1.

On a Warm reset, this field resets to an architecturally UNKNOWN value.

E0SPE, bit [0]

EL0 Statistical Profiling Enable. Controls sampling at EL0 when HCR_EL2.TGE == 0 or if EL2 is disabled or not implemented.

E0SPE	Meaning
0b0	Sampling disabled at EL0.
0b1	Sampling enabled at EL0.

If EL2 is implemented and enabled in the current Security state, this bit is ignored by the PE when HCR_EL2.TGE == 1.

On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing the PMSCR_EL1

Accesses to this register use the following encodings:

```
MRS <Xt>, PMSCR_EL1
```

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b11	0b000	0b1001	0b1001	0b000
~~0b11~~	~~0b1001~~	~~0b000~~	~~0b000~~	~~0b1001~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPMS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.NS~~SCR_ELR.NS~~ == '0' && MDCR_EL3.NSPB !=
'01' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.NS~~SCR_ELR.NS~~ == '1' && MDCR_EL3.NSPB !=
'11' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<NV2,NV1,NV> == '111' then
        return NVMem[0x828];
    else
        return PMSCR_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.NS~~SCR_ELR.NS~~ == '0' && MDCR_EL3.NSPB != '01'
then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.NS~~SCR_ELR.NS~~ == '1' && MDCR_EL3.NSPB !=
'11' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HCR_EL2.E2H == '1' then
        return PMSCR_EL2;
    else
        return PMSCR_EL1;
elsif PSTATE.EL == EL3 then
    return PMSCR_EL1;
```

```
MSR PMSCR_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b11	0b000	0b1001	0b1001	0b000
~~0b11~~	~~0b1001~~	~~0b000~~	~~0b000~~	~~0b1001~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPMS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.NSSCR_ELR.NS == '0' && MDCR_EL3.NSPB !=
'01' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.NSSCR_ELR.NS == '1' && MDCR_EL3.NSPB !=
'11' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<NV2,NV1,NV> == '111' then
        NVMem[0x828] = X[t];
    else
        PMSCR_EL1 = X[t];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.NSSCR_ELR.NS == '0' && MDCR_EL3.NSPB != '01'
then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.NSSCR_ELR.NS == '1' && MDCR_EL3.NSPB !=
'11' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HCR_EL2.E2H == '1' then
        PMSCR_EL2 = X[t];
    else
        PMSCR_EL1 = X[t];
elsif PSTATE.EL == EL3 then
    PMSCR_EL1 = X[t];
```

MRS <Xt>, PMSCR_EL12

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b11	0b101	0b1001	0b1001	0b000
~~0b11~~	~~0b1001~~	~~0b101~~	~~0b000~~	~~0b1001~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '101' then
        return NVMem[0x828];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if EL2Enabled() && HCR_EL2.E2H == '1' then
    if PSTATE.EL == EL2 then
        if HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.NSSCR_ELR.NS == '0' && MDCR_EL3.NSPB !=
'01' then
            AArch64.SystemAccessTrap(EL3, 0x18);
        elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.NSSCR_ELR.NS == '1' && MDCR_EL3.NSPB
!= '11' then
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return PMSCR_EL1;
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if EL2Enabled() && HCR_EL2.E2H == '1' then
        return PMSCR_EL1;
else
    else
        UNDEFINED;
```

```
MSR PMSCR_EL12, <Xt>
```

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b11	0b101	0b1001	0b1001	0b000
~~0b11~~	~~0b1001~~	~~0b101~~	~~0b000~~	~~0b1001~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '101' then
        NVMem[0x828] = X[t];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if EL2Enabled() && HCR_EL2.E2H == '1' then
    if PSTATE.EL == EL2 then
        if HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.NSSCR_ELR.NS == '0' && MDCR_EL3.NSPB !=
'01' then
            AArch64.SystemAccessTrap(EL3, 0x18);
        elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.NSSCR_ELR.NS == '1' && MDCR_EL3.NSPB
!= '11' then
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            PMSCR_EL1 = X[t];
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if EL2Enabled() && HCR_EL2.E2H == '1' then
        PMSCR_EL1 = X[t];
else
    else
        UNDEFINED;
```

27~~13~~/03~~12~~/2019~~2018~~ 21~~16~~:59~~42~~; e5e4db499bf9867a4b93324c4dbac985d3da93766379d01e197f1d40720d32d0f84e419e9187e009

(old) htmldiff from- (new)

PMSCR_EL2, Statistical Profiling Control Register (EL2)

The PMSCR_EL2 characteristics are:

Purpose

Provides EL2 controls for Statistical Profiling

Configuration

This register is present only when SPE is implemented. Otherwise, direct accesses to PMSCR_EL2 are UNDEFINED.

This register has no effect if EL2 is not enabled in the current Security state.

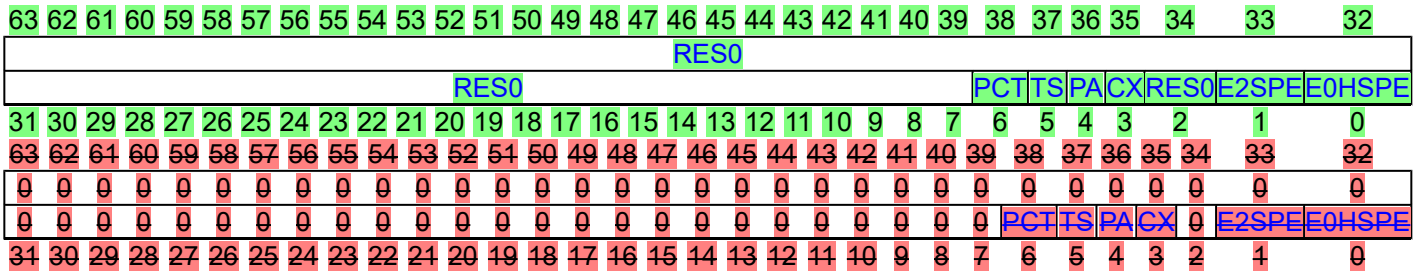
This register is in the Warm reset domain. On a Warm or Cold reset RW fields in this register reset to architecturally UNKNOWN values.

Attributes

PMSCR_EL2 is a 64-bit register.

Field descriptions

The PMSCR_EL2 bit assignments are:



Bits [63:7]

Reserved, RES0.

PCT, bit [6]

Physical Timestamp.

If timestamp sampling is enabled, determines which counter is collected.

PCT	Meaning
0b0	Virtual counter, CNTVCT_EL0, is collected.
0b1	Physical counter, CNTPCT_EL0, is collected.

If MDCR_EL2.E2PB != 0b00, this bit is combined with PMSCR_EL1.PCT to determine which counter is collected. For more information, see 'Controlling the data that is collected' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

If EL2 is not implemented in the current Security state, the PE behaves as if this bit is set to 1, other than for a direct read of the register.

If EL2 is disabled in the current Security state, this bit is ignored. If EL2 is not implemented, the PE behaves as if this bit is set to 1, other than for a direct read of the register.

If MDCR_EL2.E2PB == 0b00 and EL2 is disabled, this bit is ignored.

On a Warm reset, this field resets to an architecturally UNKNOWN value.

TS, bit [5]

Timestamp Enable.

TS	Meaning
0b0	Timestamp sampling disabled.
0b1	Timestamp sampling enabled.

If EL2 is disabled ~~or not implemented~~ in the current Security state, or if ~~the PE is in Non-secure state and~~ MDCR_EL2.E2PB != 0b00, this bit is ignored.

On a Warm reset, this field resets to an architecturally UNKNOWN value.

PA, bit [4]

Physical Address Sample Enable.

PA	Meaning
0b0	Physical addresses are not collected.
0b1	Physical addresses are collected.

If MDCR_EL2.E2PB != 0b00, this bit is combined with PMSCR_EL1.PA to determine which address is collected. For more information, see 'Controlling the data that is collected' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

IIf EL2 is disabled in the current Security state, this bit is ignored. If EL2 is not implemented, the PE behaves as if this bit is set to 1, other than for a direct read of the register.

On a Warm reset, this field resets to an architecturally UNKNOWN value.

CX, bit [3]

CONTEXTIDR_EL2 Sample Enable.

CX	Meaning
0b0	CONTEXTIDR_EL2 is not collected.
0b1	CONTEXTIDR_EL2 is collected.

If EL2 is disabled in the current Security state, this bit is ignored by the PE.

On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bit [2]

Reserved, RES0.

E2SPE, bit [1]

EL2 Statistical Profiling Enable.

E2SPE	Meaning
0b0	Sampling disabled at EL2.
0b1	Sampling enabled at EL2.

This bit is RES0 if MDCR_EL2.E2PB != 0b00.

If EL2 is disabled in the current Security state, this bit is ignored by the PE.

On a Warm reset, this field resets to an architecturally UNKNOWN value.

E0HSPE, bit [0]

EL0 Statistical Profiling Enable.

E0HSPE	Meaning
0b0	Sampling disabled at EL0.
0b1	Sampling enabled at EL0.

If MDCR_EL2.E2PB != 0b00, this bit is RES0.

If EL2 is implemented and enabled in the current Security state, this bit is ignored by the PE when HCR_EL2.TGE == 0.

On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing the PMSCR_EL2

Accesses to this register use the following encodings:

```
MRS <Xt>, PMSCR_EL2
```

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b11	0b100	0b1001	0b1001	0b000
~~0b11~~	~~0b1001~~	~~0b100~~	~~0b000~~	~~0b1001~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.NSSCR_ELR.NS == '0' && MDCR_EL3.NSPB != '01'
then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.NSSCR_ELR.NS == '1' && MDCR_EL3.NSPB !=
'11' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return PMSCR_EL2;
elsif PSTATE.EL == EL3 then
    return PMSCR_EL2;
```

```
MSR PMSCR_EL2, <Xt>
```

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b11	0b100	0b1001	0b1001	0b000
~~0b11~~	~~0b1001~~	~~0b100~~	~~0b000~~	~~0b1001~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.NSSCR_ELR.NS == '0' && MDCR_EL3.NSPB != '01'
then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.NSSCR_ELR.NS == '1' && MDCR_EL3.NSPB !=
'11' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        PMSCR_EL2 = X[t];
elsif PSTATE.EL == EL3 then
    PMSCR_EL2 = X[t];
```

MRS <Xt>, PMSCR_EL1

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b11	0b000	0b1001	0b1001	0b000
~~0b11~~	~~0b1001~~	~~0b000~~	~~0b000~~	~~0b1001~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPMS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.NSSCR_ELR.NS == '0' && MDCR_EL3.NSPB !=
'01' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.NSSCR_ELR.NS == '1' && MDCR_EL3.NSPB !=
'11' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<NV2,NV1,NV> == '111' then
        return NVMem[0x828];
    else
        return PMSCR_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.NSSCR_ELR.NS == '0' && MDCR_EL3.NSPB != '01'
then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.NSSCR_ELR.NS == '1' && MDCR_EL3.NSPB !=
'11' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HCR_EL2.E2H == '1' then
        return PMSCR_EL2;
    else
        return PMSCR_EL1;
elsif PSTATE.EL == EL3 then
    return PMSCR_EL1;
```

MSR PMSCR_EL1, <Xt>

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b11	0b000	0b1001	0b1001	0b000
~~0b11~~	~~0b1001~~	~~0b000~~	~~0b000~~	~~0b1001~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPMS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.NSSCR_ELR.NS == '0' && MDCR_EL3.NSPB !=
'01' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.NSSCR_ELR.NS == '1' && MDCR_EL3.NSPB !=
'11' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<NV2,NV1,NV> == '111' then
        NVMem[0x828] = X[t];
    else
        PMSCR_EL1 = X[t];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.NSSCR_ELR.NS == '0' && MDCR_EL3.NSPB != '01'
then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.NSSCR_ELR.NS == '1' && MDCR_EL3.NSPB !=
'11' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HCR_EL2.E2H == '1' then
        PMSCR_EL2 = X[t];
    else
        PMSCR_EL1 = X[t];
elsif PSTATE.EL == EL3 then
    PMSCR_EL1 = X[t];
```

27~~13~~/03~~12~~/2019~~2018~~ 21~~16~~:59~~42~~; e5e4db499bf9867a4b93324c4dbac985d3da93766~~379d01e197f1d40720d32d0f84e419e9187e009~~

(old) htmldiff from- (new)

(old) htmldiff from- (new)

PMSELR_EL0, Performance Monitors Event Counter Selection Register

The PMSELR_EL0 characteristics are:

Purpose

Selects the current event counter PMEVCNTR<n>_EL0~~PMEVCNTR<n>~~ or the cycle counter, CCNT.

PMSELR_EL0 is used in conjunction with PMXEVTYPER_EL0 to determine the event that increments a selected event counter, and the modes and states in which the selected counter increments.

It is also used in conjunction with PMXEVCNTR_EL0, to determine the value of a selected event counter.

Configuration

AArch64 System register PMSELR_EL0 bits [31:0] are architecturally mapped to AArch32 System register PMSELR[31:0] .

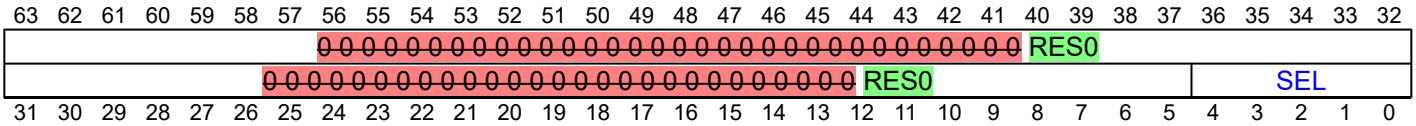
This register is in the Warm reset domain. On a Warm or Cold reset RW fields in this register reset to architecturally UNKNOWN values.

Attributes

PMSELR_EL0 is a 64-bit register.

Field descriptions

The PMSELR_EL0 bit assignments are:



Bits [63:5]

Reserved, RES0.

SEL, bits [4:0]

Selects event counter, PMEVCNTR<n>_EL0~~PMEVCNTR<n>~~, where n is the value held in this field. This value identifies which event counter is accessed when a subsequent access to PMXEVTYPER_EL0 or PMXEVCNTR_EL0 occurs.

This field can take any value from 0 (`0b00000`) to (PMCR.N)-1, or 31 (`0b11111`).

When PMSELR_EL0.SEL is `0b11111`, it selects the cycle counter and:

- A read of the PMXEVTYPER_EL0 returns the value of PMCCFILTR_EL0.
- A write of the PMXEVTYPER_EL0 writes to PMCCFILTR_EL0.
- A read or write of PMXEVCNTR_EL0 has CONSTRAINED UNPREDICTABLE effects.~~effects,~~ Sec~~that can be one of the following:~~
  - ~~Access to PMXEVCNTR_EL0 is UNDEFINED.~~
  - ~~Access to PMXEVCNTR_EL0 behaves as a NOP.~~
  - ~~Access to PMXEVCNTR_EL0 behaves as if the register is RAZ/WI.~~
  - ~~Access to PMXEVCNTR_EL0 behaves as if the PMSELR_EL0.SEL field contains an UNKNOWN value.~~

  PMXEVCNTR_EL0 for more details.

If this field is set to a value greater than or equal to the number of counters accessible at the current Exception level~~counters~~, but not equal to 31:

- Direct reads of this field return an UNKNOWN value.
- The results of access to PMXEVTYPER_EL0 or PMXEVCNTR_EL0 are CONSTRAINED UNPREDICTABLE.; Sceand can be one of the following:
  - Access to PMXEVTYPER_EL0 or PMXEVCNTR_EL0 is UNDEFINED.
  - Access to PMXEVTYPER_EL0 or PMXEVCNTR_EL0 behaves as a NOP.
  - Access to PMXEVTYPER_EL0 or PMXEVCNTR_EL0 behaves as if the register is RAZ/WI.
  - Access to PMXEVTYPER_EL0 or PMXEVCNTR_EL0 behaves as if the PMSELR_EL0.SEL field contains an UNKNOWN value.
  - Access to PMXEVTYPER_EL0 behaves as if the PMSELR_EL0.SEL field contains 0b11111.

  PMXEVTYPER_EL0 or PMXEVCNTR_EL0 for more details.

For information about the number of counters accessible at each Exception level, see MDCR_EL2.HPMN.

On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing the PMSELR_EL0

Accesses to this register use the following encodings:

```
MRS <Xt>, PMSELR_EL0
```

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b11	0b011	0b1001	0b1100	0b101
~~0b11~~	~~0b1001~~	~~0b011~~	~~0b101~~	~~0b1100~~

```
if PSTATE.EL == EL0 then
    if !ELUsingAArch32(EL1) && PMUSERENR_EL0.<ER,EN> == '00' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return PMSELR_EL0;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return PMSELR_EL0;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return PMSELR_EL0;
elsif PSTATE.EL == EL3 then
    return PMSELR_EL0;
```

```
MSR PMSELR_EL0, <Xt>
```

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b11	0b011	0b1001	0b1100	0b101
~~0b11~~	~~0b1001~~	~~0b011~~	~~0b101~~	~~0b1100~~

```
if PSTATE.EL == EL0 then
    if !ELUsingAArch32(EL1) && PMUSERENR_EL0.<ER,EN> == '00' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        PMSELR_EL0 = X[t];
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        PMSELR_EL0 = X[t];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        PMSELR_EL0 = X[t];
elsif PSTATE.EL == EL3 then
    PMSELR_EL0 = X[t];
```

2713/0312/20192018 2116:5942: e5e4db499bf9867a4b93324c4dbac985d3da93766379d01e197f1d40720d32d0f84e419e9187e009

(old) htmldiff from- (new)

PMSEVFR_EL1, Sampling Event Filter Register

The PMSEVFR_EL1 characteristics are:

Purpose

Controls sample filtering by events. The overall filter is the logical AND of these filters. For example, if E[3] and E[5] are both set to 1, only samples that have both event 3 (Level 1 unified or data cache refill) and event 5 set (TLB walk) are recorded

Configuration

This register is present only when SPE is implemented. Otherwise, direct accesses to PMSEVFR_EL1 are UNDEFINED.

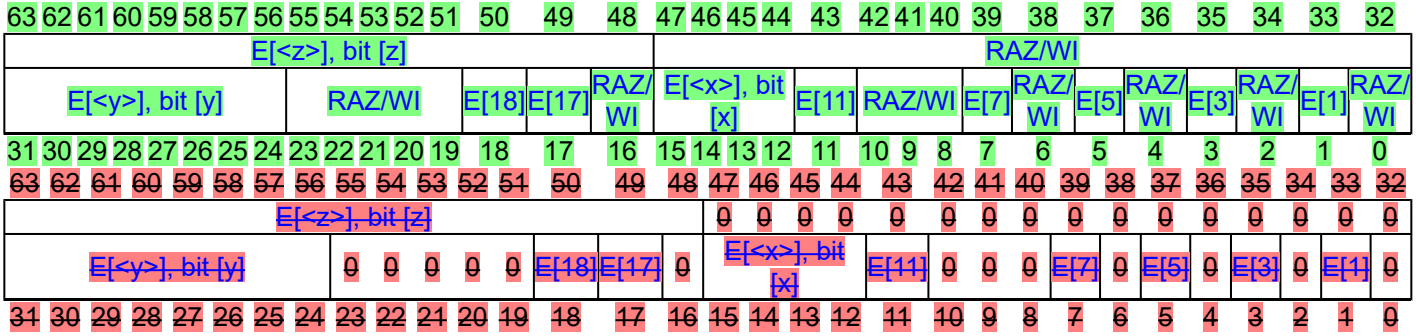
This register is in the Warm reset domain. On a Warm or Cold reset RW fields in this register reset to architecturally UNKNOWN values.

Attributes

PMSEVFR_EL1 is a 64-bit register.

Field descriptions

The PMSEVFR_EL1 bit assignments are:



E[<z>], bit [z], for z = 48 to 63

E[<z>] is the event filter for event <z>. If event <z> is not implemented, or filtering on event <z> is not supported, the corresponding bit is RAZ/WI.

E[<z>]	Meaning
0b0	Event <z> is ignored.
0b1	Do not record samples that have event <z> == 0.

An IMPLEMENTATION DEFINED event might be recorded as a multi-bit field. In this case, if the corresponding bits of PMSEVFR_EL1 define an IMPLEMENTATION DEFINED filter for the event.

This field is ignored by the PE when PMSFCR_EL1.FE == 0

On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [47:32]

Reserved, RAZ/WI.

E[<y>], bit [y], for y = 24 to 31

E[<y>] is the event filter for event <y>. If event <y> is not implemented, or filtering on event <y> is not supported, the corresponding bit is RAZ/WI.

E[<y>]	Meaning
0b0	Event <y> is ignored.
0b1	Do not record samples that have event <y> == 0.

An IMPLEMENTATION DEFINED event might be recorded as a multi-bit field. In this case, if the corresponding bits of PMSEVFR_EL1 define an IMPLEMENTATION DEFINED filter for the event.

This field is ignored by the PE when PMSFCR_EL1.FE == 0

On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [23:19]

Reserved, RAZ/WI.

E[18], bit [18]

When ARMv8.3-SPE is implemented and SVE is implemented:

Empty predicate.

E[18]	Meaning
0b0	Empty predicate event is ignored.
0b1	Do not record samples that have the Empty predicate event == 0.

This bit is ignored by the PE when PMSFCR_EL1.FE == 0.

On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RAZ/WI.

E[17], bit [17]

When ARMv8.3-SPE is implemented and SVE is implemented:

Partial predicate.

E[17]	Meaning
0b0	Partial predicate event is ignored.
0b1	Do not record samples that have the Partial predicate event == 0.

This bit is ignored by the PE when PMSFCR_EL1.FE == 0.

On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RAZ/WI.

Bit [16]

Reserved, RAZ/WI.

E[<x>], bit [x], for x = 12 to 15

E[<x>] is the event filter for event <x>. If event <x> is not implemented, or filtering on event <x> is not supported, the corresponding bit is RAZ/WI.

E[<x>]	Meaning
0b0	Event <x> is ignored.
0b1	Do not record samples that have event <x> == 0.

An IMPLEMENTATION DEFINED event might be recorded as a multi-bit field. In this case, if the corresponding bits of PMSEVFR_EL1 define an IMPLEMENTATION DEFINED filter for the event.

This field is ignored by the PE when PMSFCR_EL1.FE == 0

On a Warm reset, this field resets to an architecturally UNKNOWN value.

E[11], bit [11]

When ARMv8.3-SPE is implemented:

Alignment.

E[11]	Meaning
0b0	Alignment event is ignored.
0b1	Do not record samples that have the Alignment event == 0.

This bit is ignored by the PE when PMSFCR_EL1.FE == 0.

On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RAZ/WI.

Bits [10:8]

Reserved, RAZ/WI.

E[7], bit [7]

Mispredicted.

E[7]	Meaning
0b0	Mispredicted event is ignored.
0b1	Do not record samples that have the Mispredicted event == 0.

This bit is ignored by the PE when PMSFCR_EL1.FE == 0.

On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bit [6]

Reserved, RAZ/WI.

E[5], bit [5]

TLB walk.

E[5]	Meaning
0b0	TLB walk event is ignored.
0b1	Do not record samples that have the TLB walk event == 0.

This bit is ignored by the PE when PMSFCR_EL1.FE == 0.

On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bit [4]

Reserved, RAZ/WI.

E[3], bit [3]

Level 1 data or unified cache refill.

E[3]	Meaning
0b0	Level 1 data or unified cache refill event is ignored.
0b1	Do not record samples that have the Level 1 data or unified cache refill event == 0.

This bit is ignored by the PE when PMSFCR_EL1.FE == 0.

On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bit [2]

Reserved, RAZ/WI.

E[1], bit [1]

When the PE supports sampling of speculative instructions:

Architecturally retired.

When the PE supports sampling of speculative instructions:

E[1]	Meaning
0b0	Architecturally retired event is ignored.
0b1	Do not record samples that have the Architecturally retired event == 0.

This bit is ignored by the PE when PMSFCR_EL1.FE == 0.

If the PE does not support the sampling of speculative instructions, or always discards the sample record for speculative instructions, this bit reads as an UNKNOWN value and the PE ignores its value.

On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, UNKNOWN.

Bit [0]

Reserved, RAZ/WI.

Accessing the PMSEVFR_EL1

Accesses to this register use the following encodings:

```
MRS <Xt>, PMSEVFR_EL1
```

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b11	0b000	0b1001	0b1001	0b101
~~0b11~~	~~0b1001~~	~~0b000~~	~~0b101~~	~~0b1001~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPMS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.NSSCR_ELR.NS == '0' && MDCR_EL3.NSPB !=
'01' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.NSSCR_ELR.NS == '1' && MDCR_EL3.NSPB !=
'11' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<NV2,NV1,NV> == '1x1' then
        return NVMem[0x830];
    else
        return PMSEVFR_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.NSSCR_ELR.NS == '0' && MDCR_EL3.NSPB != '01'
then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.NSSCR_ELR.NS == '1' && MDCR_EL3.NSPB !=
'11' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return PMSEVFR_EL1;
elsif PSTATE.EL == EL3 then
    return PMSEVFR_EL1;
```

```
MSR PMSEVFR_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b11	0b000	0b1001	0b1001	0b101
~~0b11~~	~~0b1001~~	~~0b000~~	~~0b101~~	~~0b1001~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPMS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.NSSCR_ELR.NS == '0' && MDCR_EL3.NSPB !=
'01' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.NSSCR_ELR.NS == '1' && MDCR_EL3.NSPB !=
'11' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<NV2,NV1,NV> == '1x1' then
        NVMem[0x830] = X[t];
    else
        PMSEVFR_EL1 = X[t];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.NSSCR_ELR.NS == '0' && MDCR_EL3.NSPB != '01'
then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.NSSCR_ELR.NS == '1' && MDCR_EL3.NSPB !=
'11' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        PMSEVFR_EL1 = X[t];
elsif PSTATE.EL == EL3 then
    PMSEVFR_EL1 = X[t];
```

(old) htmldiff from- (new)

(old) htmldiff from- (new)

PMSFCR_EL1, Sampling Filter Control Register

The PMSFCR_EL1 characteristics are:

Purpose

Controls sample filtering. The filter is the logical AND of the FL, FT and FE bits. For example, if FE == 1 and FT == 1 only samples including the selected operation types and the selected events will be recorded

Configuration

This register is present only when SPE is implemented. Otherwise, direct accesses to PMSFCR_EL1 are UNDEFINED.

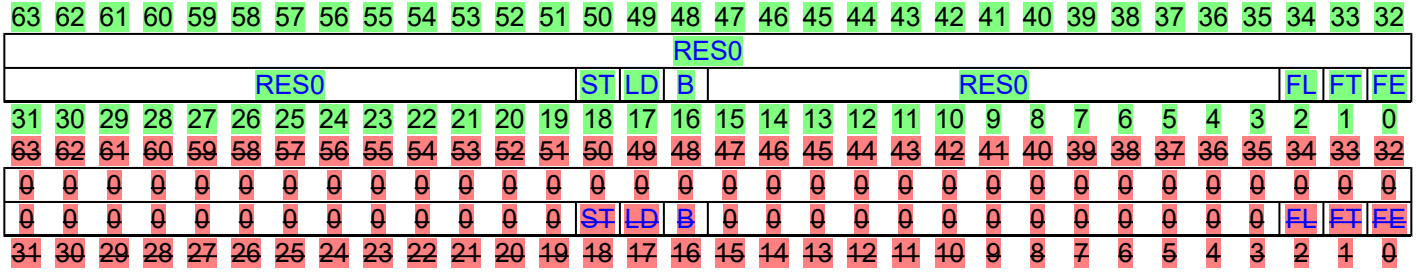
This register is in the Warm reset domain. On a Warm or Cold reset RW fields in this register reset to architecturally UNKNOWN values.

Attributes

PMSFCR_EL1 is a 64-bit register.

Field descriptions

The PMSFCR_EL1 bit assignments are:



Bits [63:19]

Reserved, RES0.

ST, bit [18]

Store filter enable

ST	Meaning
0b0	Do not record store operations
0b1	Record all store operations, including vector stores and all atomic operations

This bit is ignored by the PE when PMSFCR_EL1.FT == 0.

On a Warm reset, this field resets to an architecturally UNKNOWN value.

LD, bit [17]

Load filter enable

LD	Meaning
0b0	Do not record load operations
0b1	Record all load operations, including vector loads and atomic operations that return data

This bit is ignored by the PE when PMSFCR_EL1.FT == 0.

On a Warm reset, this field resets to an architecturally UNKNOWN value.

B, bit [16]

Branch filter enable

B	Meaning
0b0	Do not record branch and exception return operations
0b1	Record all branch and exception return operations

This bit is ignored by the PE when PMSFCR_EL1.FT == 0.

On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [15:3]

Reserved, RES0.

FL, bit [2]

Filter by latency

FL	Meaning
0b0	Latency filtering disabled
0b1	Latency filtering enabled. Samples with a total latency less than PMSLATFR_EL1.MINLAT will not be recorded

If this field is set to 1 and PMSLATFR_EL1.MINLAT is set to zero, it is CONSTRAINED UNPREDICTABLE whether no samples are recorded or the PE behaves as if PMSFCR_EL1.FL is set to 0

On a Warm reset, this field resets to an architecturally UNKNOWN value.

FT, bit [1]

Filter by operation type. The filter is the logical OR of the ST, LD and B bits. For example, if LD and ST are both set, both load and store operations are recorded

FT	Meaning
0b0	Type filtering disabled
0b1	Type filtering enabled. Samples not one of the selected operation types will not be recorded

If this field is set to 1 and the PMSFCR_EL1.{ST, LD, B} bits are all set to zero, it is CONSTRAINED UNPREDICTABLE whether no samples are recorded or the PE behaves as if PMSFCR_EL1.FT is set to 0

On a Warm reset, this field resets to an architecturally UNKNOWN value.

FE, bit [0]

Filter by event

FE	Meaning
0b0	Event filtering disabled
0b1	Event filtering enabled. Samples not including the events selected by PMSEVFR_EL1 will not be recorded

If this field is set to 1 and PMSEVFR_EL1 is set to zero, it is CONSTRAINED UNPREDICTABLE whether no samples are recorded or the PE behaves as if PMSFCR_EL1.FE is set to 0

On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing the PMSFCR_EL1

Accesses to this register use the following encodings:

MRS <Xt>, PMSFCR_EL1

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b11	0b000	0b1001	0b1001	0b100
~~0b11~~	~~0b1001~~	~~0b000~~	~~0b100~~	~~0b1001~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPMS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.NS~~SCR_ELR.NS~~ == '0' && MDCR_EL3.NSPB !=
'01' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.NS~~SCR_ELR.NS~~ == '1' && MDCR_EL3.NSPB !=
'11' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return PMSFCR_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.NS~~SCR_ELR.NS~~ == '0' && MDCR_EL3.NSPB != '01'
then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.NS~~SCR_ELR.NS~~ == '1' && MDCR_EL3.NSPB !=
'11' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return PMSFCR_EL1;
elsif PSTATE.EL == EL3 then
    return PMSFCR_EL1;
```

MSR PMSFCR_EL1, <Xt>

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b11	0b000	0b1001	0b1001	0b100
~~0b11~~	~~0b1001~~	~~0b000~~	~~0b100~~	~~0b1001~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPMS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.NSSCR_ELR.NS == '0' && MDCR_EL3.NSPB !=
'01' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.NSSCR_ELR.NS == '1' && MDCR_EL3.NSPB !=
'11' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        PMSFCR_EL1 = X[t];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.NSSCR_ELR.NS == '0' && MDCR_EL3.NSPB != '01'
then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.NSSCR_ELR.NS == '1' && MDCR_EL3.NSPB !=
'11' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        PMSFCR_EL1 = X[t];
elsif PSTATE.EL == EL3 then
    PMSFCR_EL1 = X[t];
```

2713/0312/20192018 2116:5942; e5e4db499bf9867a4b93324c4dbac985d3da93766379d01e197f1d40720d32d0f84c419c9187c009

(old) htmldiff from- (new)

PMSICR_EL1, Sampling Interval Counter Register

The PMSICR_EL1 characteristics are:

Purpose

Software must write zero to PMSICR_EL1 before enabling sample profiling for a sampling session. Software must then treat PMSICR_EL1 as an opaque, 64-bit, read/write register used for context switches only.

Configuration

This register is present only when SPE is implemented. Otherwise, direct accesses to PMSICR_EL1 are UNDEFINED.

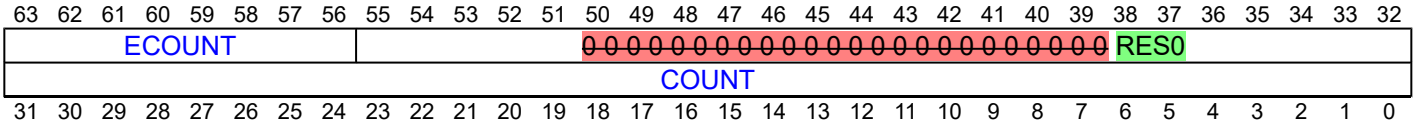
This register is in the Warm reset domain. On a Warm or Cold reset RW fields in this register reset to architecturally UNKNOWN values.

Attributes

The value of PMSICR_EL1 does not change whilst profiling is disabled.

Field descriptions

The PMSICR_EL1 bit assignments are:



ECOUNT, bits [63:56]

When PMSIDR_EL1.ERnd == 0b1:

Secondary sample interval counter.

When PMSIDR_EL1.ERnd is 1, this field provides the secondary counter used after the primary counter reaches zero. Whilst the secondary counter is nonzero and profiling is enabled, the secondary counter decrements by 1 for each member of the sample population. The primary counter also continues to decrement since it is also nonzero. When the secondary counter reaches zero, a member of the sampling population is selected for sampling.

When PMSIDR_EL1.ERnd is 0, this field is RES1.

On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [55:32]

Reserved, RES0.

COUNT, bits [31:0]

Primary sample interval counter

Provides the primary counter used for sampling.

The primary counter is reloaded when the value of this register is zero and the PE moves from a state or Exception level where profiling is disabled to a state or Exception level where profiling is enabled

Whilst the primary counter is nonzero and sampling is enabled, the primary counter decrements by 1 for each member of the sample population

When the counter reaches zero, the behavior depends on the values of PMSIDR_EL1.ERnd and PMSIRR_EL1.RND

- If PMSIRR_EL1.RND == 0 or PMSIDR_EL1.ERnd == 0:
  - A member of the sampling population is selected for sampling
  - The primary counter is reloaded
- If PMSIRR_EL1.RND == 1 and PMSIDR_EL1.ERnd == 1:
  - The secondary counter is set to a random or pseudorandom value in the range `0x00` to `0xFF`
  - The primary counter is reloaded

On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing the PMSICR_EL1

Accesses to this register use the following encodings:

```
MRS <Xt>, PMSICR_EL1
```

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b11	0b000	0b1001	0b1001	0b010
~~0b11~~	~~0b1001~~	~~0b000~~	~~0b010~~	~~0b1001~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPMS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.NS~~SCR_ELR.NS~~ == '0' && MDCR_EL3.NSPB !=
'01' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.NS~~SCR_ELR.NS~~ == '1' && MDCR_EL3.NSPB !=
'11' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<NV2,NV1,NV> == '1x1' then
        return NVMem[0x838];
    else
        return PMSICR_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.NS~~SCR_ELR.NS~~ == '0' && MDCR_EL3.NSPB != '01'
then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.NS~~SCR_ELR.NS~~ == '1' && MDCR_EL3.NSPB !=
'11' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return PMSICR_EL1;
elsif PSTATE.EL == EL3 then
    return PMSICR_EL1;
```

```
MSR PMSICR_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b11	0b000	0b1001	0b1001	0b010
~~0b11~~	~~0b1001~~	~~0b000~~	~~0b010~~	~~0b1001~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPMS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.NSSCR_ELR.NS == '0' && MDCR_EL3.NSPB !=
'01' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.NSSCR_ELR.NS == '1' && MDCR_EL3.NSPB !=
'11' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<NV2,NV1,NV> == '1x1' then
        NVMem[0x838] = X[t];
    else
        PMSICR_EL1 = X[t];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.NSSCR_ELR.NS == '0' && MDCR_EL3.NSPB != '01'
then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.NSSCR_ELR.NS == '1' && MDCR_EL3.NSPB !=
'11' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        PMSICR_EL1 = X[t];
elsif PSTATE.EL == EL3 then
    PMSICR_EL1 = X[t];
```

2713/0312/20192018 2116:5942: e5e4db499bf9867a4b93324c4dbac985d3da93766379d01e197f1d40720d32d0f84c419e9187e009

(old) htmldiff from- (new)

PMSIDR_EL1, Sampling Profiling ID Register

The PMSIDR_EL1 characteristics are:

Purpose

Describes the Statistical Profiling implementation to software

Configuration

This register is present only when SPE is implemented. Otherwise, direct accesses to PMSIDR_EL1 are UNDEFINED.

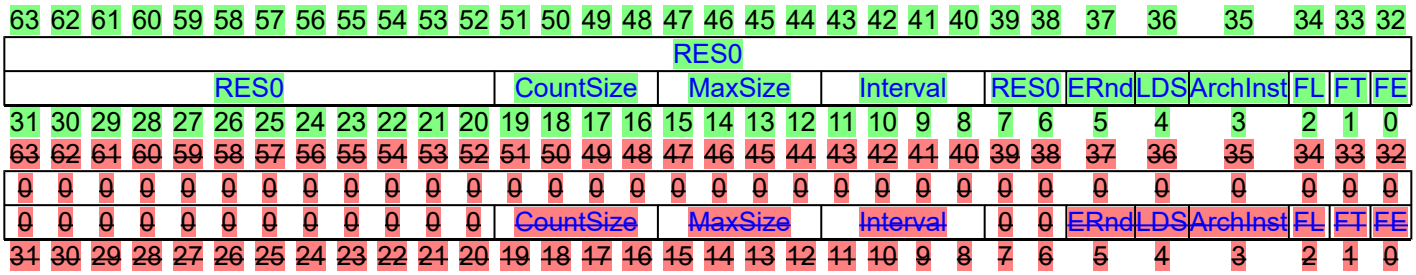
This register is in the Warm reset domain. On a Warm or Cold reset RW fields in this register reset to architecturally UNKNOWN values.

Attributes

PMSIDR_EL1 is a 64-bit register.

Field descriptions

The PMSIDR_EL1 bit assignments are:



Bits [63:20]

Reserved, RES0.

CountSize, bits [19:16]

Defines the size of the counters

CountSize	Meaning
0b0010	12-bit saturating counters

All other values are reserved. Reserved values might be defined in a future version of the architecture.

On a Warm reset, this field resets to an architecturally UNKNOWN value.

MaxSize, bits [15:12]

Defines the largest size for a single record, rounded up to a power-of-two. If this is the same as the minimum alignment (PMBIDR_EL1.Align), then each record is exactly this size

MaxSize	Meaning
0b0100	16 bytes
0b0101	32 bytes
0b0110	64 bytes
0b0111	128 bytes
0b1000	256 bytes
0b1001	512 bytes
0b1010	1024 bytes
0b1011	2KB

All other values are reserved. Reserved values might be defined in a future version of the architecture.

On a Warm reset, this field resets to an architecturally UNKNOWN value.

Interval, bits [11:8]

Recommended minimum sampling interval. This provides guidance from the implementer to the smallest minimum sampling interval, N.

Interval	Meaning
0b0000	256
0b0010	512
0b0011	768
0b0100	1,024
0b0101	1,536
0b0110	2,048
0b0111	3,072
0b1000	4,096

All other values are reserved. Reserved values might be defined in a future version of the architecture.

On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [7:6]

Reserved, RES0.

ERnd, bit [5]

Defines how the random number generator is used in determining the interval between samples, when enabled by PMSIRR_EL1.RND.

ERnd	Meaning
0b0	The random number is added at the start of the interval, and the sample is taken and a new interval started when the combined interval expires.
0b1	The random number is added and the new interval started after the interval programmed in PMSIRR_EL1.INTERVAL expires, and the sample is taken when the random interval expires.

On a Warm reset, this field resets to an architecturally UNKNOWN value.

LDS, bit [4]

Data source indicator for sampled load instructions

LDS	Meaning
0b0	Loaded data source not implemented
0b1	Loaded data source implemented

On a Warm reset, this field resets to an architecturally UNKNOWN value.

ArchInst, bit [3]

Architectural instruction profiling

ArchInst	Meaning
0b0	Micro-op sampling implemented
0b1	Architecture instruction sampling implemented

On a Warm reset, this field resets to an architecturally UNKNOWN value.

FL, bit [2]

Filtering by latency. This bit is RAO.

On a Warm reset, this field resets to an architecturally UNKNOWN value.

FT, bit [1]

Filtering by operation type. This bit is RAO.

On a Warm reset, this field resets to an architecturally UNKNOWN value.

FE, bit [0]

Filtering by events. This bit is RAO.

On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing the PMSIDR_EL1

Accesses to this register use the following encodings:

```
MRS <Xt>, PMSIDR_EL1
```

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b11	0b000	0b1001	0b1001	0b111
~~0b11~~	~~0b1001~~	~~0b000~~	~~0b111~~	~~0b1001~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPMS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.NS~~SCR_ELR.NS~~ == '0' && MDCR_EL3.NSPB !=
'01' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.NS~~SCR_ELR.NS~~ == '1' && MDCR_EL3.NSPB !=
'11' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return PMSIDR_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.NS~~SCR_ELR.NS~~ == '0' && MDCR_EL3.NSPB != '01'
then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.NS~~SCR_ELR.NS~~ == '1' && MDCR_EL3.NSPB !=
'11' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return PMSIDR_EL1;
elsif PSTATE.EL == EL3 then
    return PMSIDR_EL1;
```

27~~13~~/03~~12~~/2019~~2018~~ 21~~16~~:59~~42~~: e5e4db499bf9867a4b93324c4dbac985d3da93766~~379d01e197f1d40720d32d0f84e419e9187e009~~

(old) htmldiff from- (new)

(old) htmldiff from- (new)

PMSIRR_EL1, Sampling Interval Reload Register

The PMSIRR_EL1 characteristics are:

Purpose

Defines the interval between samples

Configuration

This register is present only when SPE is implemented. Otherwise, direct accesses to PMSIRR_EL1 are UNDEFINED.

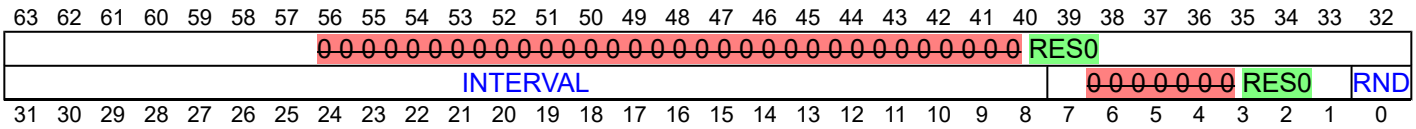
This register is in the Warm reset domain. On a Warm or Cold reset RW fields in this register reset to architecturally UNKNOWN values.

Attributes

PMSIRR_EL1 is a 64-bit register.

Field descriptions

The PMSIRR_EL1 bit assignments are:



Bits [63:32]

Reserved, RES0.

INTERVAL, bits [31:8]

Bits [31:8] of the PMSICR_EL1 interval counter reload value. Software must set this to a non-zero value. If software sets this to zero, an UNKNOWN sampling interval is used. Software should set this to a value greater than the minimum indicated by PMSIDR_EL1.Interval

On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [7:1]

Reserved, RES0.

RND, bit [0]

Controls randomization of the sampling interval

RND	Meaning
0b0	Disable randomization of sampling interval
0b1	Add (pseudo-)random jitter to sampling interval

The random number generator is not architected.

On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing the PMSIRR_EL1

Accesses to this register use the following encodings:

MRS <Xt>, PMSIRR_EL1

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b11	0b000	0b1001	0b1001	0b011
~~0b11~~	~~0b1001~~	~~0b000~~	~~0b011~~	~~0b1001~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPMS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.NS~~SCR_ELR.NS~~ == '0' && MDCR_EL3.NSPB !=
'01' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.NS~~SCR_ELR.NS~~ == '1' && MDCR_EL3.NSPB !=
'11' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<NV2,NV1,NV> == '1x1' then
        return NVMem[0x840];
    else
        return PMSIRR_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.NS~~SCR_ELR.NS~~ == '0' && MDCR_EL3.NSPB != '01'
then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.NS~~SCR_ELR.NS~~ == '1' && MDCR_EL3.NSPB !=
'11' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return PMSIRR_EL1;
elsif PSTATE.EL == EL3 then
    return PMSIRR_EL1;
```

MSR PMSIRR_EL1, <Xt>

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b11	0b000	0b1001	0b1001	0b011
~~0b11~~	~~0b1001~~	~~0b000~~	~~0b011~~	~~0b1001~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPMS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.NSSCR_ELR.NS == '0' && MDCR_EL3.NSPB !=
'01' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.NSSCR_ELR.NS == '1' && MDCR_EL3.NSPB !=
'11' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<NV2,NV1,NV> == '1x1' then
        NVMem[0x840] = X[t];
    else
        PMSIRR_EL1 = X[t];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.NSSCR_ELR.NS == '0' && MDCR_EL3.NSPB != '01'
then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.NSSCR_ELR.NS == '1' && MDCR_EL3.NSPB !=
'11' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        PMSIRR_EL1 = X[t];
elsif PSTATE.EL == EL3 then
    PMSIRR_EL1 = X[t];
```

2713/0312/20192018 2116:5942: e5e4db499bf9867a4b93324c4dbac985d3da93766379d01e197f1d40720d32d0f84e419e9187e009

(old) htmldiff from- (new)

PMSLATFR_EL1, Sampling Latency Filter Register

The PMSLATFR_EL1 characteristics are:

Purpose

Controls sample filtering by latency

Configuration

This register is present only when SPE is implemented. Otherwise, direct accesses to PMSLATFR_EL1 are UNDEFINED.

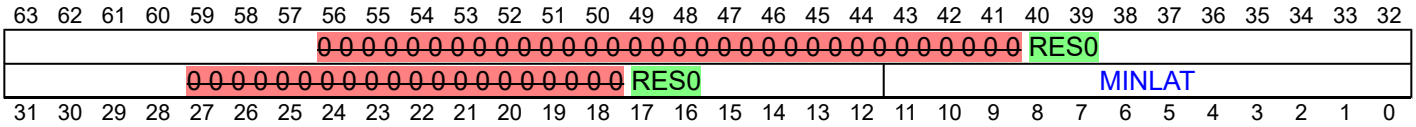
This register is in the Warm reset domain. On a Warm or Cold reset RW fields in this register reset to architecturally UNKNOWN values.

Attributes

PMSLATFR_EL1 is a 64-bit register.

Field descriptions

The PMSLATFR_EL1 bit assignments are:



Bits [63:12]

Reserved, RES0.

MINLAT, bits [11:0]

Minimum latency. When PMSFCR_EL1.FL == 1, defines the minimum total latency for filtered operations. Samples with a total latency less than MINLAT will not be recorded

This field is ignored by the PE when PMSFCR_EL1.FL == 0.

On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing the PMSLATFR_EL1

Accesses to this register use the following encodings:

```
MRS <Xt>, PMSLATFR_EL1
```

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b11	0b000	0b1001	0b1001	0b110
~~0b11~~	~~0b1001~~	~~0b000~~	~~0b110~~	~~0b1001~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPMS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.NSSCR_ELR.NS == '0' && MDCR_EL3.NSPB !=
'01' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.NSSCR_ELR.NS == '1' && MDCR_EL3.NSPB !=
'11' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<NV2,NV1,NV> == '1x1' then
        return NVMem[0x848];
    else
        return PMSLATFR_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.NSSCR_ELR.NS == '0' && MDCR_EL3.NSPB != '01'
then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.NSSCR_ELR.NS == '1' && MDCR_EL3.NSPB !=
'11' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return PMSLATFR_EL1;
elsif PSTATE.EL == EL3 then
    return PMSLATFR_EL1;
```

```
MSR PMSLATFR_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b11	0b000	0b1001	0b1001	0b110
~~0b11~~	~~0b1001~~	~~0b000~~	~~0b110~~	~~0b1001~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPMS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.NSSCR_ELR.NS == '0' && MDCR_EL3.NSPB !=
'01' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.NSSCR_ELR.NS == '1' && MDCR_EL3.NSPB !=
'11' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<NV2,NV1,NV> == '1x1' then
        NVMem[0x848] = X[t];
    else
        PMSLATFR_EL1 = X[t];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.NSSCR_ELR.NS == '0' && MDCR_EL3.NSPB != '01'
then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.NSSCR_ELR.NS == '1' && MDCR_EL3.NSPB !=
'11' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        PMSLATFR_EL1 = X[t];
elsif PSTATE.EL == EL3 then
    PMSLATFR_EL1 = X[t];
```

(old) htmldiff from- (new)

(old) htmldiff from- (new)

PMXEVCNTR_EL0, Performance Monitors Selected Event Count Register

The PMXEVCNTR_EL0 characteristics are:

Purpose

Reads or writes the value of the selected event counter, PMEVCNTR<n>_EL0. PMSELR_EL0.SEL determines which event counter is selected.

Configuration

AArch64 System register PMXEVCNTR_EL0 bits [31:0] are architecturally mapped to AArch32 System register PMXEVCNTR[31:0] .

This register is in the Warm reset domain. On a Warm or Cold reset RW fields in this register reset to architecturally UNKNOWN values.

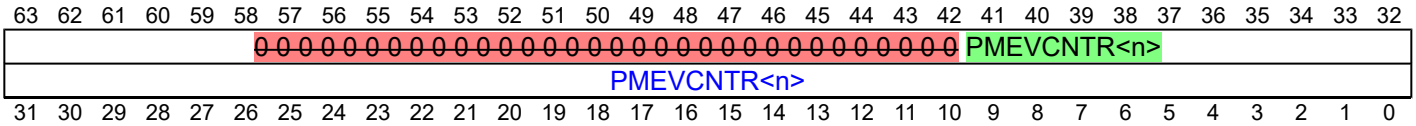
Attributes

PMXEVCNTR_EL0 is a 64-bit register.

Field descriptions

The PMXEVCNTR_EL0 bit assignments are:

When ARMv8.5-PMU is implemented:

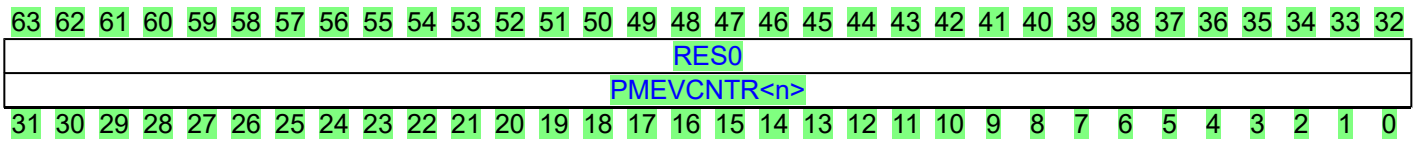


PMEVCNTR<n>, bits [63:0]

Value of the selected event counter, PMEVCNTR<n>_EL0, where n is the value stored in PMSELR_EL0.SEL.

On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:



Bits [63:32]

Reserved, RES0.

PMEVCNTR<n>, bits [31:0]

Value of the selected event counter, PMEVCNTR<n>_EL0, where n is the value stored in PMSELR_EL0.SEL.

On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing the PMXEVCNTR_EL0

If PMSELR_EL0.SEL is greater than or equal to the number of accessible counters then reads and writes of PMXEVCNTR_EL0 are CONSTRAINED UNPREDICTABLE, and the following behaviors are permitted:

- Accesses to the register are UNDEFINED.
- Accesses to the register behave as RAZ/WI.
- Accesses to the register execute as a NOP
- Accesses to the register behave as if PMSELR_EL0.SEL has an UNKNOWN value less than the number of counters accessible at the current Exception level and Security state.
- If EL2 is implemented and enabled in the current Security state, and PMSELR_EL0.SEL is less than the number of implemented counters, accesses from EL1 or permitted accesses from EL0 are trapped to EL2.

Note

In EL0, an access is permitted if it is enabled by PMUSERENR_EL0.{ER,EN}.

If EL2 is implemented and enabled in the current Security state, in EL1 and EL0, MDCR_EL2.HPMN identifies the number of accessible counters. Otherwise, the number of accessible counters is the number of implemented counters. See MDCR_EL2.HPMN for more details.

Accesses to this register use the following encodings:

```
MRS <Xt>, PMXEVCNTR_EL0
```

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b11	0b011	0b1001	0b1101	0b010
~~0b11~~	~~0b1001~~	~~0b011~~	~~0b010~~	~~0b1101~~

```
if PSTATE.EL == EL0 then
    if !ELUsingAArch32(EL1) && PMUSERENR_EL0.<ER,EN> == '00' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return PMXEVCNTR_EL0;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return PMXEVCNTR_EL0;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return PMXEVCNTR_EL0;
elsif PSTATE.EL == EL3 then
    return PMXEVCNTR_EL0;
```

```
MSR PMXEVCNTR_EL0, <Xt>
```

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~

0b11	0b011	0b1001	0b1101	0b010
~~0b11~~	~~0b1001~~	~~0b011~~	~~0b010~~	~~0b1101~~

```
if PSTATE.EL == EL0 then
    if !ELUsingAArch32(EL1) && PMUSERENR_EL0.EN == '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        PMXEVCNTR_EL0 = X[t];
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        PMXEVCNTR_EL0 = X[t];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        PMXEVCNTR_EL0 = X[t];
elsif PSTATE.EL == EL3 then
    PMXEVCNTR_EL0 = X[t];
```

27~~13~~/03~~12~~/2019~~2018~~ 21~~16~~:59~~42~~; e5e4db499bf9867a4b93324c4dbac985d3da93766~~379d01e197f1d40720d32d0f84e419e9187e009~~

(old) htmldiff from- (new)

RGSR_EL1, Random Allocation Tag Seed Register.

The RGSR_EL1 characteristics are:

Purpose

Random Allocation Tag Seed Register.

Configuration

This register is present only when ARMv8.5-MemTag is implemented. Otherwise, direct accesses to RGSR_EL1 are UNDEFINED.

When GCR_EL1.RRND==0b1, the value of RGSR_EL1 is UNKNOWN.

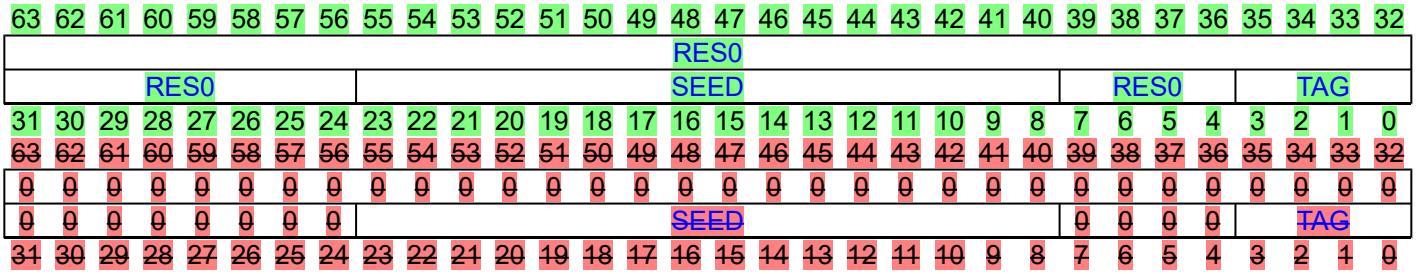
RW fields in this register reset to architecturally UNKNOWN values.

Attributes

RGSR_EL1 is a 64-bit register.

Field descriptions

The RGSR_EL1 bit assignments are:



Bits [63:24]

Reserved, RES0.

SEED, bits [23:8]

Seed register used for generating values returned by RandomAllocationTag().

This field resets to an architecturally UNKNOWN value.

Bits [7:4]

Reserved, RES0.

TAG, bits [3:0]

Tag generated by the most recent IRG instruction.

This field resets to an architecturally UNKNOWN value.

Accessing the RGSR_EL1

Accesses to this register use the following encodings:

```
MRS <Xt>, RGSR_EL1
```

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b11	0b000	0b0001	0b0000	0b101
~~0b11~~	~~0b0001~~	~~0b000~~	~~0b101~~	~~0b0000~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.ATA == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.ATA == '0' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return RGSR_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.ATA == '0' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return RGSR_EL1;
elsif PSTATE.EL == EL3 then
    return RGSR_EL1;
```

```
MSR RGSR_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b11	0b000	0b0001	0b0000	0b101
~~0b11~~	~~0b0001~~	~~0b000~~	~~0b101~~	~~0b0000~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.ATA == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.ATA == '0' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        RGSR_EL1 = X[t];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.ATA == '0' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        RGSR_EL1 = X[t];
elsif PSTATE.EL == EL3 then
    RGSR_EL1 = X[t];
```

27~~13~~/03~~12~~/2019~~2018~~ 21~~16~~:59~~42~~; e5e4db499bf9867a4b93324c4dbac985d3da93766~~379d01e197f1d40720d32d0f84e419e9187e009~~

(old) htmldiff from- (new)

SCR_EL3, Secure Configuration Register

The SCR_EL3 characteristics are:

Purpose

Defines the configuration of the current Security state. It specifies:

- The Security state of EL0, EL1, and EL2. The Security state is either Secure or Non-secure.
- The Execution state at lower Exception levels.
- Whether IRQ, FIQ, SError interrupts, and External abort exceptions are taken to EL3.
- Whether various operations are trapped to EL3.

Configuration

AArch64 System register SCR_EL3 bits [31:0] can be mapped to AArch32 System register SCR[31:0] , but this is not architecturally mandated.

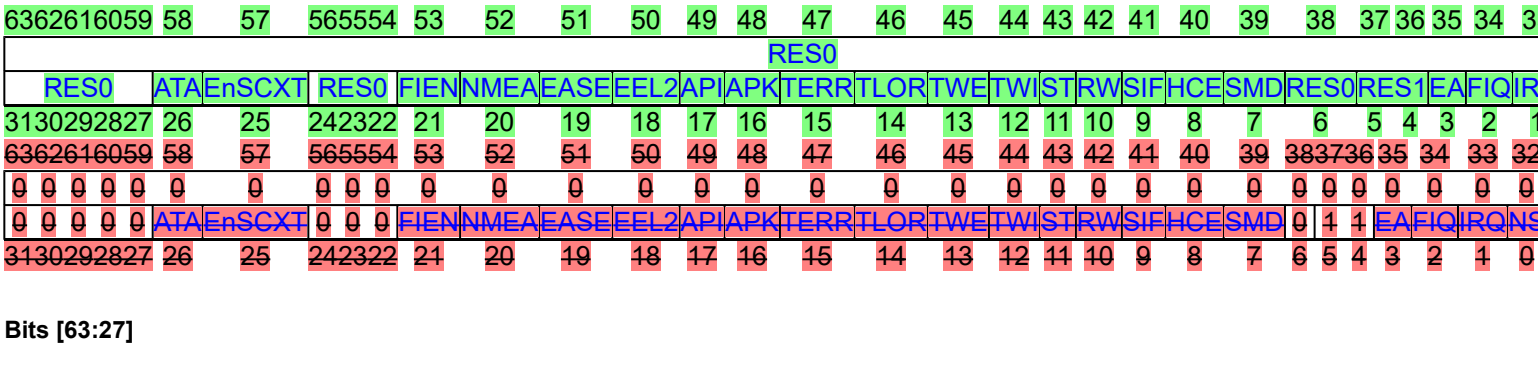
Some or all RW fields of this register have defined reset values. These apply only if the PE resets into an Exception level that is using AArch64. Otherwise, RW fields in this register reset to architecturally UNKNOWN values.

Attributes

SCR_EL3 is a 64-bit register.

Field descriptions

The SCR_EL3 bit assignments are:



Bits [63:27]

Reserved, RES0.

ATA, bit [26]

When ARMv8.5-MemTag is implemented:

Allocation Tag Access. Controls access at EL2, EL1 and EL0 to Allocation Tags.

When access to Allocation Tags is prevented:

- Instructions which Load or Store data are Unchecked.
- Instructions which Load or Store Allocation Tags treat the Allocation Tag as RAZ/WI.
- Instructions which insert Logical Address Tags into addresses treat the Allocation Tag used to generate the Logical Address Tag as 0.
- Cache maintenance instructions which invalidate Allocation Tags from caches behave as the equivalent Clean and Invalidate operation on Allocation Tags.

- MRS and MSR instructions at EL1 and EL2 using GCR_EL1, RGSR_EL1, TFSR_EL1, TFSR_EL2 or TFSRE0_EL1 that are not UNDEFINED or trapped to a lower Exception level are trapped to EL3.
- MRS and MSR instructions at EL2 using TFSR_EL12 that are not UNDEFINED are trapped to EL3.

ATA	Meaning
0b0	Access ~~to Allocation Tags~~ is prevented.
0b1	Access ~~to Allocation Tags~~ is not prevented.

This field is permitted to be cached in a TLB.

This field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EnSCXT, bit [25]

When ARMv8.0-CSV2 is implemented:

Enable access to the SCXTNUM_EL2, SCXTNUM_EL1, and SCXTNUM_EL0 registers. The defined values are:

EnSCXT	Meaning
0b0	EL2, EL1 and EL0 access to SCXTNUM_EL0, EL2 and EL1 access to SCXTNUM_EL1, EL2 access to SCXTNUM_EL2 registers are disabled by this mechanism, causing an exception to EL3, and the values of these registers to be treated as 0.
0b1	This control does not cause accesses to SCXTNUM_EL0, SCXTNUM_EL1, SCXTNUM_EL2 to be trapped.

This field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [24:22]

Reserved, RES0.

FIEN, bit [21]

When~~From~~ ARMv8.4-RAS is implemented~~Armv8.4~~:

Fault Injection enable. Trap accesses to the ERXPFGCDN_EL1, ERXPFGCTL_EL1, and ERXPFGF_EL1 registers from EL1 and EL2 to EL3.

FIEN	Meaning
0b0	Accesses to the specified registers from EL1 and EL2 generate a Trap exception to EL3.
0b1	This control does not cause any instructions to be trapped.

If EL3 is not implemented, the Effective value of SCR_EL3.FIEN is 0b1.

If ~~the RAS Common Fault Injection Model Extension is not implemented, this field is~~ ERRIDR_EL1.NUM is zero, meaning no error records are implemented, or no error record accessible using System registers is owned by a node that implements the RAS Common Fault Injection Model Extension, then this bit might be RES0.

This field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

NMEA, bit [20]

When ARMv8.4-DFE is implemented:

Non-maskable External Aborts. When SCR_EL3.EA == 1, controls whether PSTATE.A masks SError interrupts at EL3.

NMEA	Meaning
0b0	If SCR_EL3.EA == 1, asserted SError interrupts are not taken at EL3 if PSTATE.A == 1.
0b1	If SCR_EL3.EA == 1, asserted SError interrupts are taken at EL3 regardless of the value of PSTATE.A.

When SCR_EL3.EA == 0:

- Asserted SError interrupts are not taken at EL3 regardless of the value of PSTATE.A and this field.
- This field is ignored and its Effective value is 0.

This field resets to 0.

Otherwise:

Reserved, RES0.

EASE, bit [19]

When ARMv8.4-DFE is implemented:

External aborts to SError interrupt vector.

EASE	Meaning
0b0	Synchronous External abort exceptions taken to EL3 are taken to the appropriate synchronous exception vector offset from VBAR_EL3.
0b1	Synchronous External abort exceptions taken to EL3 are taken to the appropriate SError interrupt vector offset from VBAR_EL3.

This field resets to 0.

Otherwise:

Reserved, RES0.

EEL2, bit [18]

When ARMv8.4-SecEL2 is implemented:

Secure EL2 Enable.

EEL2	Meaning
0b0	All behaviors associated with Secure EL2 are disabled. All registers, including timer registers, defined by ARMv8.4-SecEL2 are UNDEFINED, and those timers are disabled.
0b1	All behaviors associated with Secure EL2 are enabled.

When the value of this bit is 1, then:

- When SCR_EL3.NS == 0, the SCR_EL3.RW bit is treated as 1 for all purposes other than reading or writing the register.
- If Secure EL1 is using AArch32, then any of the following operations, executed in Secure EL1, is trapped to Secure EL2 using the EC value of ESR_EL2.EC== 0x3 :
  - A read or write of the SCR.
  - A read or write of the NSACR.
  - A read or write of the MVBAR.
  - A read or write of the SDCR.
  - Execution of an ATS12NSO** instruction.

- If Secure EL1 is using AArch32, then any of the following operations, executed in Secure EL1, is trapped to Secure EL2 using the EC value of ESR_EL2.EC== 0x0 :
    - Execution of an SRS instruction that uses R13_mon.
    - Execution of an MRS (Banked register) or MSR (Banked register) instruction that would access SPSR_mon, R13_mon, or R14_mon.

Note

If the Effective value of SCR_EL3.EEL2 is 0, then these operations executed in Secure EL1 using AArch32 are trapped to EL3.

In a Secure only implementation that does not implement EL3 but implements EL2, behaves as if SCR_EL3.EEL2 == 1.

This bit is permitted to be cached in a TLB.

This field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

API, bit [17]

From Armv8.4, when ARMv8.3-PAuth is implemented:

Controls the use of instructions related to Pointer Authentication:

- PACGA.
- AUTDA, AUTDB, AUTDZA, AUTDZB, AUTIA, AUTIA1716, AUTIASP, AUTIAZ, AUTIB, AUTIB1716, AUTIBSP, AUTIBZ, AUTIZA, AUTIZB, PACDA, PACDB, PACDZA, PACDZB, PACIA, PACIA1716, PACIASP, PACIAZ, PACIB, PACIB1716, PACIBSP, PACIBZ, PACIZA, PACIZ, RETAA, RETAB, BRAA, BRAB, BLRAA, BLRAB, BRAAZ, BRABZ, BLRAAZ, BLRABZ, ERETAA, ERETAB, LDRAA and LDRAB when:
    - In EL0, when HCR_EL2.TGE==0 or HCR_EL2.E2H==0, and the associated SCTLR_EL1.En<N><M> == 1.
    - In EL0, when HCR_EL2.TGE==1 and HCR_EL2.E2H==1, and the associated SCTLR_EL2.En<N><M> == 1.
    - In EL1, when the associated SCTLR_EL1.En<N><M> == 1.
    - In EL2, when the associated SCTLR_EL2.En<N><M> == 1.

API	Meaning
0b0	The use of any instruction related to pointer authentication in any Exception level except EL3 when the instructions are enabled are trapped to EL3 unless they are trapped to EL2 as a result of the HCR_EL2.API bit.
0b1	This control does not cause any instructions to be trapped.

Note

If ARMv8.3-PAuth is implemented but EL3 is not implemented, the system behaves as if this bit is 1.

This field resets to an architecturally UNKNOWN value.

From Armv8.3, when ARMv8.3-PAuth is implemented:

Controls the use of instructions related to Pointer Authentication:

- PACGA.
- AUTDA, AUTDB, AUTDZA, AUTDZB, AUTIA, AUTIA1716, AUTIASP, AUTIAZ, AUTIB, AUTIB1716, AUTIBSP, AUTIBZ, AUTIZA, AUTIZB, PACDA, PACDB, PACDZA, PACDZB, PACIA, PACIA1716, PACIASP, PACIAZ, PACIB, PACIB1716, PACIBSP, PACIBZ, PACIZA, PACIZ, RETAA, RETAB, BRAA, BRAB, BLRAA, BLRAB, BRAAZ, BRABZ, BLRAAZ, BLRABZ, ERETAA, ERETAB, LDRAA and LDRAB when:
    - In Non-secure EL0, when HCR_EL2.TGE==0 or HCR_EL2.E2H==0, and the associated SCTLR_EL1.En<N><M>== 1.
    - In Non-secure EL0, when HCR_EL2.TGE==1 and HCR_EL2.E2H==1, and the associated SCTLR_EL2.En<N><M> == 1.
    - In Secure EL0, when the associated SCTLR_EL2.En<N><M> == 1.
    - In Secure or Non-secure EL1, when the associated SCTLR_EL1.En<N><M> == 1.
    - In EL2, when the associated SCTLR_EL2.En<N><M> == 1.

API	Meaning
0b0	The use of any instruction related to pointer authentication in any Exception level except EL3 when the instructions are enabled are trapped to EL3 unless they are trapped to EL2 as a result of the HCR_EL2.API bit.
0b1	This control does not cause any instructions to be trapped.

Note

If ARMv8.3-PAuth is implemented but EL3 is not implemented, the system behaves as if this bit is 1.

This field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

APK, bit [16]

When ARMv8.3-PAuth is implemented:

Trap registers holding "key" values for Pointer Authentication. Traps accesses to the following registers from EL1 or EL2 to EL3 unless they are trapped to EL2 as a result of the HCR_EL2.APK bit or other traps:

- APIAKeyLo_EL1, APIAKeyHi_EL1, APIBKeyLo_EL1, APIBKeyHi_EL1, APDAKeyLo_EL1, APDAKeyHi_EL1, APDBKeyLo_EL1, APDBKeyHi_EL1, APGAKeyLo_EL1, and APGAKeyHi_EL1.

APK	Meaning
0b0	Access to the registers holding "key" values for pointer authentication from EL1 or EL2 are trapped to EL3 unless they are trapped to EL2 as a result of the HCR_EL2.APK bit or other traps.
0b1	This control does not cause any instructions to be trapped.

Note

If ARMv8.3-PAuth is implemented but EL3 is not implemented, the system behaves as if this bit is 1.

This field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TERR, bit [15]

When RAS is implemented:

Trap Error record accesses. Trap accesses to the following registers from EL1 and EL2 to EL3:

EL1 using AArch64: ERRIDR_EL1, ERRSELR_EL1, ERXADDR_EL1, ERXCTLR_EL1, ERXFR_EL1, ERXMISC0_EL1, ERXMISC1_EL1, and ERXSTATUS_EL1. When ARMv8.4-RAS is implemented, ERXMISC2_EL1, and ERXMISC3_EL1.

EL1 using AArch32: ERRIDR, ERRSELR, ERXADDR, ERXADDR2, ERXCTLR, ERXCTLR2, ERXFR, ERXFR2, ERXMISC0, ERXMISC1, ERXMISC2, ERXMISC3, and ERXSTATUS. When ARMv8.4-RAS is implemented, ERXMISC4, ERXMISC5, ERXMISC6, and ERXMISC7.

TERR	Meaning
0b0	This control does not cause any instructions to be trapped.
0b1	Accesses to the specified registers from EL1 and EL2 generate a Trap exception to EL3.

This field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TLOR, bit [14]

When ARMv8.1-LOR is implemented:

Trap LOR registers. Traps accesses to the LORSA_EL1, LOREA_EL1, LORN_EL1, LORC_EL1, and LORID_EL1 registers from EL1 and EL2 to EL3, unless the access has been trapped to EL2.

TLOR	Meaning
0b0	This control does not cause any instructions to be trapped.
0b1	EL1 and EL2 accesses to the LOR registers that are not UNDEFINED are trapped to EL3, unless it is trapped HCR_EL2.TLOR.

This field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TWE, bit [13]

Traps EL2, EL1, and EL0 execution of WFE instructions to EL3, from both Security states and both Execution states.

TWE	Meaning
0b0	This control does not cause any instructions to be trapped.
0b1	Any attempt to execute a WFE instruction at any Exception level lower than EL3 is trapped to EL3, if the instruction would otherwise have caused the PE to enter a low-power state and it is not trapped by SCTLR.nTWE, HCR.TWE, SCTLR_EL1.nTWE, SCTLR_EL2.nTWE, or HCR_EL2.TWE.

In AArch32 state, the attempted execution of a conditional WFE instruction is only trapped if the instruction passes its condition code check.

Note

Since a WFE or WFI can complete at any time, even without a Wakeup event, the traps on WFE of WFI are not guaranteed to be taken, even if the WFE or WFI is executed when there is no Wakeup event. The only guarantee is that if the instruction does not complete in finite time in the absence of a Wakeup event, the trap will be taken.

This field resets to an architecturally UNKNOWN value.

TWI, bit [12]

Traps EL2, EL1, and EL0 execution of WFI instructions to EL3, from both Security states and both Execution states.

TWI	Meaning
0b0	This control does not cause any instructions to be trapped.
0b1	Any attempt to execute a WFI instruction at any Exception level lower than EL3 is trapped to EL3, if the instruction would otherwise have caused the PE to enter a low-power state and it is not trapped by SCTLR.nTWI, HCR.TWI, SCTLR_EL1.nTWI, SCTLR_EL2.nTWI, or HCR_EL2.TWI.

In AArch32 state, the attempted execution of a conditional WFI instruction is only trapped if the instruction passes its condition code check.

Note

Since a WFE or WFI can complete at any time, even without a Wakeup event, the traps on WFE of WFI are not guaranteed to be taken, even if the WFE or WFI is executed when there is no Wakeup

event. The only guarantee is that if the instruction does not complete in finite time in the absence of a Wakeup event, the trap will be taken.

This field resets to an architecturally UNKNOWN value.

ST, bit [11]

Traps Secure EL1 accesses to the Counter-timer Physical Secure timer registers to EL3, from AArch64 state only.

ST	Meaning
0b0	Secure EL1 using AArch64 accesses to the CNTPS_TVAL_EL1, CNTPS_CTL_EL1, and CNTPS_CVAL_EL1 are trapped to EL3 when Secure EL2 is disabled. If Secure EL2 is enabled, the behaviour is as if the value of this field was 0b1.
0b1	This control does not cause any instructions to be trapped.

This field resets to an architecturally UNKNOWN value.

RW, bit [10]

Execution state control for lower Exception levels.

RW	Meaning
0b0	Lower levels are all AArch32.
0b1	The next lower level is AArch64. If EL2 is present: • EL2 is AArch64. • EL2 controls EL1 and EL0 behaviors. If EL2 is not present: • EL1 is AArch64. • EL0 is determined by the Execution state described in the current process state when executing at EL0.

If AArch32 state is not supported by the implementation at EL2 and AArch32 state is not supported by the implementation at EL1, then this bit is RAO/WI.

If AArch32 state is supported by the implementation at EL1, SCR_EL3.NS == 1 and AArch32 state is not supported by the implementation at EL2, the Effective value of this bit is 1.

If AArch32 state is supported by the implementation at EL1, ARMv8.4-SecEL2 is implemented and SCR_EL3.{EEL2, NS} == {1, 0}, the Effective value of this bit is 1.

This bit is permitted to be cached in a TLB.

This field resets to an architecturally UNKNOWN value.

SIF, bit [9]

From Armv8.4:

Secure instruction fetch. When the PE is in Secure state, this bit disables instruction fetch from memory marked in the first stage of translation as being Non-secure. The possible values for this bit are:

SIF	Meaning
0b0	Secure state instruction fetches from memory marked in the first stage of translation as being Non-secure are permitted.
0b1	Secure state instruction fetches from memory marked in the first stage of translation as being Non-secure are not permitted.

This bit is permitted to be cached in a TLB.

This field resets to an architecturally UNKNOWN value.

Otherwise:

Secure instruction fetch. When the PE is in Secure state, this bit disables instruction fetch from Non-secure memory.

SIF	Meaning
0b0	Secure state instruction fetches from Non-secure memory are permitted.
0b1	Secure state instruction fetches from Non-secure memory are not permitted.

This bit is permitted to be cached in a TLB.

This field resets to an architecturally UNKNOWN value.

HCE, bit [8]

Hypervisor Call instruction enable. Enables HVC instructions at EL3 and, if EL2 is enabled in the current Security state, at EL2 and EL1, in both Execution states.

HCE	Meaning
0b0	HVC instructions are UNDEFINED.
0b1	HVC instructions are enabled at EL3, EL2, and EL1.

Note

HVC instructions are always UNDEFINED at EL0 and, if Secure EL2 is disabled, at Secure EL1.

If EL2 is not implemented, this bit is RES0.

This field resets to an architecturally UNKNOWN value.

SMD, bit [7]

Secure Monitor Call disable. Disables SMC instructions at EL1 and above, from both Security states and both Execution states.

SMD	Meaning
0b0	SMC instructions are enabled at EL1 and above.
0b1	SMC instructions are UNDEFINED at EL1 and above.

Note

SMC instructions are always UNDEFINED at EL0.

This field resets to an architecturally UNKNOWN value.

Bit [6]

Reserved, RES0.

Bits [5:4]

Reserved, RES1.

EA, bit [3]

External Abort and SError interrupt routing.

EA	Meaning
0b0	When executing at Exception levels below EL3, External aborts and SError interrupts are not taken to EL3. In addition, when executing at EL3: • SError interrupts are not taken. • External aborts are taken to EL3.
0b1	When executing at any Exception level, External aborts and SError interrupts are taken to EL3.

For more information, see 'Asynchronous exception routing' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section D1 (The AArch64 System Level Programmers' Model).

This field resets to an architecturally UNKNOWN value.

FIQ, bit [2]

Physical FIQ Routing.

FIQ	Meaning
0b0	When executing at Exception levels below EL3, physical FIQ interrupts are not taken to EL3. When executing at EL3, physical FIQ interrupts are not taken.
0b1	When executing at any Exception level, physical FIQ interrupts are taken to EL3.

For more information, see 'Asynchronous exception routing' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section D1.

This field resets to an architecturally UNKNOWN value.

IRQ, bit [1]

Physical IRQ Routing.

IRQ	Meaning
0b0	When executing at Exception levels below EL3, physical IRQ interrupts are not taken to EL3. When executing at EL3, physical IRQ interrupts are not taken.
0b1	When executing at any Exception level, physical IRQ interrupts are taken to EL3.

For more information, see 'Asynchronous exception routing' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section D1.

This field resets to an architecturally UNKNOWN value.

NS, bit [0]

Non-secure bit.

NS	Meaning
0b0	Indicates that EL0 and EL1 are in Secure state.
0b1	Indicates that Exception levels lower than EL3 are in Non-secure state, and so memory accesses from those Exception levels cannot access Secure memory.

When SCR_EL3.{EEL2, NS} == {1, 0}, then EL2 is using AArch64 and in Secure state.

This field resets to an architecturally UNKNOWN value.

Accessing the SCR_EL3

Accesses to this register use the following encodings:

```
MRS <Xt>, SCR_EL3
```

op0	op1	CRn	CRm	op2

~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b11	0b110	0b0001	0b0001	0b000
~~0b11~~	~~0b0001~~	~~0b110~~	~~0b000~~	~~0b0001~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    return SCR_EL3;
```

```
MSR SCR_EL3, <Xt>
```

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b11	0b110	0b0001	0b0001	0b000
~~0b11~~	~~0b0001~~	~~0b110~~	~~0b000~~	~~0b0001~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    SCR_EL3 = X[t];
```

27~~13~~/03~~12~~/2019~~2018~~ 21~~16~~:59~~42~~; e5e4db499bf9867a4b93324c4dbac985d3da93766~~379d01e197f1d40720d32d0f84e419e9187e009~~

(old) htmldiff from- (new)

SCTLR_EL1, System Control Register (EL1)

The SCTLR_EL1 characteristics are:

Purpose

Provides top level control of the system, including its memory system, at EL1 and EL0.

Configuration

AArch64 System register SCTLR_EL1 bits [31:0] are architecturally mapped to AArch32 System register SCTLR[31:0] .

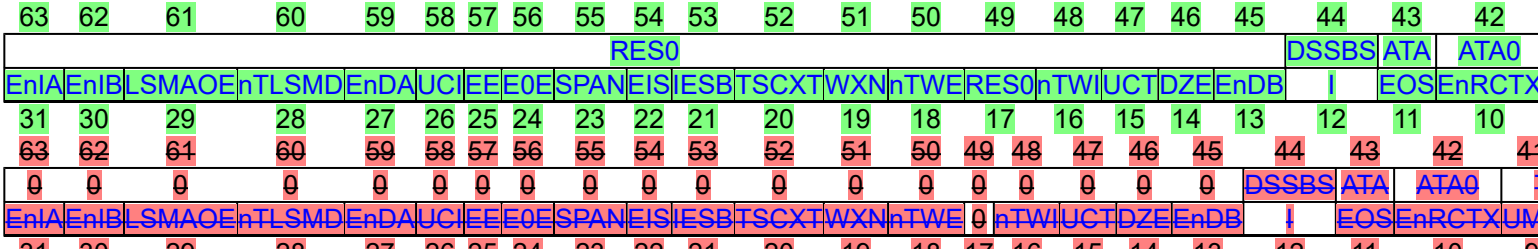
Some or all RW fields of this register have defined reset values. These apply only if the PE resets into EL1 using AArch64. Otherwise, RW fields in this register reset to architecturally UNKNOWN values.

Attributes

SCTLR_EL1 is a 64-bit register.

Field descriptions

The SCTLR_EL1 bit assignments are:



Bits [63:45]

Reserved, RES0.

DSSBS, bit [44]

From Armv8.5:

Default PSTATE.SSBS value on Exception Entry. The defined values are:

DSSBS	Meaning
0b0	PSTATE.SSBS is set to 0 on an exception to EL1
0b1	PSTATE.SSBS is set to 1 on an exception to EL1

In a system where the PE resets into EL1, this field resets to an IMPLEMENTATION DEFINED value.

Otherwise:

Reserved, RES0.

ATA, bit [43]

When ARMv8.5-MemTag is implemented:

Allocation Tag Access in EL1. When SCR_EL3.ATA=1 and HCR_EL2.ATA=1, controls EL1 access to Allocation Tags.

When access to Allocation Tags is prevented:

- Instructions which Load or Store data are Unchecked.
- Instructions which Load or Store Allocation Tags treat the Allocation Tag as RAZ/WI.
- Instructions which insert Logical Address Tags into addresses treat the Allocation Tag used to generate the Logical Address Tag as 0.
- Cache maintenance instructions which invalidate Allocation Tags from caches behave as the equivalent Clean and Invalidate operation on Allocation Tags.

ATA	Meaning
0b0	Access to Allocation Tags is prevented.
0b1	Access to Allocation Tags is not prevented.

This bit is permitted to be cached in a TLB.

In a system where the PE resets into EL1, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

ATA0, bit [42]

When ARMv8.5-MemTag is implemented:

Allocation Tag Access in EL0. When SCR_EL3.ATA=1, HCR_EL2.ATA=1, and HCR_EL2.{E2H,TGE} != {1,1}, controls EL0 access to Allocation Tags.

When access to Allocation Tags is prevented:

- Instructions which Load or Store data are Unchecked.
- Instructions which Load or Store Allocation Tags treat the Allocation Tag as RAZ/WI.
- Instructions which insert Logical Address Tags into addresses treat the Allocation Tag used to generate the Logical Address Tag as 0.
- Cache maintenance instructions which invalidate Allocation Tags from caches behave as the equivalent Clean and Invalidate operation on Allocation Tags.

ATA0	Meaning
0b0	Access to Allocation Tags is prevented.
0b1	Access to Allocation Tags is not prevented.

This field is permitted to be cached in a TLB.

Note

Software may change this control bit on a context switch.

In a system where the PE resets into EL1, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TCF, bits [41:40]

When ARMv8.5-MemTag is implemented:

Tag Check Fail in EL1. Controls the effect of tag check fails due to Loads and Stores in EL1.

TCF	Meaning
0b00	Tag check fails have no effect on the PE.
0b01	Tag check fails causes a synchronous exception.
0b10	Tag check fails are asynchronously accumulated.

The value 0b11 is reserved.

This field is permitted to be cached in a TLB.

In a system where the PE resets into EL1, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TCF0, bits [39:38]

When ARMv8.5-MemTag is implemented:

Tag Check Fail in EL0. When HCR_EL2.{E2H,TGE} != {1,1}, controls the effect of tag check fails due to Loads and Stores in EL0.

TCF0	Meaning
0b00	Tag check fails have no effect on the PE.
0b01	Tag check fails causes a synchronous exception.
0b10	Tag check fails are asynchronously accumulated.

The value 0b11 is reserved.

This field is permitted to be cached in a TLB.

Note

Software may change this control bit on a context switch.

In a system where the PE resets into EL1, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

ITFSB, bit [37]

When ARMv8.5-MemTag is implemented:

When synchronous exceptions are not being generated by Tag Check fails which are generated for Loads and Stores in EL0 or EL1, controls the auto-synchronisaton of Tag Check fails into TFSRE0_EL1 and TFSR_EL1.

ITFSB	Meaning
0b0	Tag check fails are not synchronized on entry to EL1.
0b1	Tag check fails are synchronized on entry to EL1.

In a system where the PE resets into EL1, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

BT1, bit [36]

When ARMv8.5-BTI is implemented:

PAC Branch Type compatibility at EL1.

BT1	Meaning
0b0	When the PE is executing at EL1, PACIASP and PACIBSP are compatible with PSTATE.BTYPE == 0b11.
0b1	When the PE is executing at EL1, PACIASP and PACIBSP are not compatible with PSTATE.BTYPE == 0b11.

In a system where the PE resets into EL1, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

BT0, bit [35]

When ARMv8.5-BTI is implemented:

PAC Branch Type compatibility at EL0.

BT0	Meaning
0b0	When the PE is executing at EL0, PACIASP and PACIBSP are compatible with PSTATE.BTYPE == 0b11.
0b1	When the PE is executing at EL0, PACIASP and PACIBSP are not compatible with PSTATE.BTYPE == 0b11.

When HCR_EL2.E2H == 1 && HCR_EL2.TGE == 1, the value of the SCTLR_EL1.BT0 has no effect on execution at EL0

In a system where the PE resets into EL1, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [34:32]

Reserved, RES0.

EnIA, bit [31]

From Armv8.3:

Controls enabling of pointer authentication (using the APIAKey_EL1 key) of instruction addresses in the EL1&0 translation regime.

Possible values of this bit are:

EnIA	Meaning
0b0	Pointer authentication (using the APIAKey_EL1 key) of instruction addresses is not enabled.
0b1	Pointer authentication (using the APIAKey_EL1 key) of instruction addresses is enabled.

Note

This field controls the behavior of the AddPACIA and AuthIA pseudocode functions. Specifically, when the field is 1, AddPACIA returns a copy of a pointer to which a pointer authentication code has been added, and AuthIA returns an authenticated copy of a pointer. When the field is 0, both of these functions are NOP.

In a system where the PE resets into EL1, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EnIB, bit [30]

From Armv8.3:

Controls enabling of pointer authentication (using the APIBKey_EL1 key) of instruction addresses in the EL1&0 translation regime.

Possible values of this bit are:

EnIB	Meaning
0b0	Pointer authentication (using the APIBKey_EL1 key) of instruction addresses is not enabled.
0b1	Pointer authentication (using the APIBKey_EL1 key) of instruction addresses is enabled.

Note

This field controls the behavior of the AddPACIB and AuthIB pseudocode functions. Specifically, when the field is 1, AddPACIB returns a copy of a pointer to which a pointer authentication code has been added, and AuthIB returns an authenticated copy of a pointer. When the field is 0, both of these functions are NOP.

In a system where the PE resets into EL1, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

LSMAOE, bit [29]

When ARMv8.2-LSMAOC is implemented:

Load Multiple and Store Multiple Atomicity and Ordering Enable.

LSMAOE	Meaning
0b0	For all memory accesses at EL0, A32 and T32 Load Multiple and Store Multiple can have an interrupt taken during the sequence memory accesses, and the memory accesses are not required to be ordered.
0b1	The ordering and interrupt behavior of A32 and T32 Load Multiple and Store Multiple at EL0 is as defined for Armv8.0.

This bit is permitted to be cached in a TLB.

When ARMv8.1-VHE is implemented, and the value of HCR_EL2.{E2H, TGE} is {1,1}, this bit has no effect on execution at EL0.

In a system where the PE resets into EL1, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES1.

nTLSMD, bit [28]

When ARMv8.2-LSMAOC is implemented:

No Trap Load Multiple and Store Multiple to Device-nGRE/Device-nGnRE/Device-nGnRnE memory.

nTLSMD	Meaning
0b0	All memory accesses by A32 and T32 Load Multiple and Store Multiple at EL0 that are marked at stage 1 as Device-nGRE/Device-nGnRE/Device-nGnRnE memory are trapped and generate a stage 1 Alignment fault.
0b1	All memory accesses by A32 and T32 Load Multiple and Store Multiple at EL0 that are marked at stage 1 as Device-nGRE/Device-nGnRE/Device-nGnRnE memory are not trapped.

This bit is permitted to be cached in a TLB.

When ARMv8.1-VHE is implemented, and the value of HCR_EL2.{E2H, TGE} is {1,1}, this bit has no effect on execution at EL0.

In a system where the PE resets into EL1, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES1.

EnDA, bit [27]

From Armv8.3:

Controls enabling of pointer authentication (using the APDAKey_EL1 key) of instruction addresses in the EL1&0 translation regime.

Possible values of this bit are:

EnDA	Meaning
0b0	Pointer authentication (using the APDAKey_EL1 key) of data addresses is not enabled.
0b1	Pointer authentication (using the APDAKey_EL1 key) of data addresses is enabled.

Note

This field controls the behavior of the AddPACDA and AuthDA pseudocode functions. Specifically, when the field is 1, AddPACDA returns a copy of a pointer to which a pointer authentication code has been added, and AuthDA returns an authenticated copy of a pointer. When the field is 0, both of these functions are NOP.

In a system where the PE resets into EL1, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

UCI, bit [26]

Traps EL0 execution of cache maintenance instructions to EL1, from AArch64 state only. This applies to DC CVAU, DC CIVAC, DC CVAC, DC CVAP, and IC IVAU.

If ARMv8.2-DCCVADP is implemented, this trap also applies to DC CVADP.

If ARMv8.5-MemTag is implemented, this trap also applies to DC CIGVAC, DC CIGDVAC, DC CGVAC, DC CGDVAC, DC CGVAP, and DC CGDVAP.

If ARMv8.2-DCCVADP and ARMv8.5-MemTag are implemented, this trap also applies to DC CGVADP and DC CGDVADP.

UCI	Meaning
0b0	Execution of the specified instructions at EL0 using AArch64 is trapped to EL1.
0b1	This control does not cause any instructions to be trapped.

When ARMv8.1-VHE is implemented, and the value of HCR_EL2.{E2H, TGE} is {1, 1}, this bit has no effect on execution at EL0.

If the Point of Coherency is before any level of data cache, it is IMPLEMENTATION DEFINED whether the execution of any data or unified cache clean, or clean and invalidate instruction that operates by VA to the point of coherency can be trapped when the value of this control is 1.

If the Point of Unification is before any level of data cache, it is IMPLEMENTATION DEFINED whether the execution of any data or unified cache clean by VA to the Point of Unification instruction can be trapped when the value of this control is 1.

If the Point of Unification is before any level of instruction cache, it is IMPLEMENTATION DEFINED whether the execution of any instruction cache invalidate by VA to the Point of Unification instruction can be trapped when the value of this control is 1.

In a system where the PE resets into EL1, this field resets to an architecturally UNKNOWN value.

EE, bit [25]

Endianness of data accesses at EL1, and stage 1 translation table walks in the EL1&0 translation regime.

The possible values of this bit are:

EE	Meaning
0b0	Explicit data accesses at EL1, and stage 1 translation table walks in the EL1&0 translation regime are little-endian.
0b1	Explicit data accesses at EL1, and stage 1 translation table walks in the EL1&0 translation regime are big-endian.

If an implementation does not provide Big-endian support at Exception Levels higher than EL0, this bit is RES0.

If an implementation does not provide Little-endian support at Exception Levels higher than EL0, this bit is RES1.

The EE bit is permitted to be cached in a TLB.

When ARMv8.1-VHE is implemented, and the value of HCR_EL2.{E2H, TGE} is {1, 1}, this bit has no effect on the PE.

In a system where the PE resets into EL1, this field resets to an IMPLEMENTATION DEFINED value.

E0E, bit [24]

Endianness of data accesses at EL0.

The possible values of this bit are:

E0E	Meaning
0b0	Explicit data accesses at EL0 are little-endian.
0b1	Explicit data accesses at EL0 are big-endian.

If an implementation only supports Little-endian accesses at EL0 then this bit is RES0. This option is not permitted when SCTLR_EL1.EE is RES1.

If an implementation only supports Big-endian accesses at EL0 then this bit is RES1. This option is not permitted when SCTLR_EL1.EE is RES0.

This bit has no effect on the endianness of LDTR, LDTRH, LDTRSH, LDTRSW, STTR, and STTRH instructions executed at EL1.

When ARMv8.1-VHE is implemented, and the value of HCR_EL2.{E2H, TGE} is {1, 1}, this bit has no effect on execution at EL0.

In a system where the PE resets into EL1, this field resets to an architecturally UNKNOWN value.

SPAN, bit [23]

From Armv8.1:

Set Privileged Access Never, on taking an exception to EL1.

SPAN	Meaning
0b0	PSTATE.PAN is set to 1 on taking an exception to EL1.
0b1	The value of PSTATE.PAN is left unchanged on taking an exception to EL1.

When ARMv8.1-VHE is implemented, and the value of HCR_EL2.{E2H, TGE} is {1, 1}, this bit has no effect on execution at EL0.

In a system where the PE resets into EL1, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES1.

EIS, bit [22]

From Armv8.5:

Exception Entry is Context Synchronizing. The defined values are:

EIS	Meaning
0b0	The taking of an exception to EL1 is not a context synchronizing event.
0b1	The taking of an exception to EL1 is a context synchronizing event.

When ARMv8.1-VHE is implemented, and the value of HCR_EL2.{E2H, TGE} is {1,1}, this bit has no effect on execution at EL0.

In a system where the PE resets into EL1, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES1.

IESB, bit [21]

When ARMv8.2-IESB is implemented:

Implicit Error Synchronization event enable. Possible values are:

IESB	Meaning
0b0	Disabled.
0b1	An implicit error synchronization event is added: • After each exception taken to EL1. • Before the operational pseudocode of each ERET instruction executed at EL1.

When the PE is in Debug state, the effect of this field is CONSTRAINED UNPREDICTABLE, and its Effective value might be 0 or 1 regardless of the value of the field. If the Effective value of the field is 1, then an implicit error synchronization event is added after each DCPSx instruction taken to EL1 and before each DRPS instruction executed at EL1, in addition to the other cases where it is added.

In a system where the PE resets into EL1, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TSCXT, bit [20]

From Armv8.5:

Trap EL0 Access to the SCXTNUM_EL0 register, when EL0 is using AArch64. The defined values are:

TSCXT	Meaning
0b0	EL0 access to SCXTNUM_EL0 is not disabled by this mechanism.
0b1	EL0 access to SCXTNUM_EL0 is disabled, causing an exception to EL1, and the SCXTNUM_EL0 value is treated at 0.

When ARMv8.1-VHE is implemented, and the value of HCR_EL2.{E2H, TGE} is {1,1}, this bit has no effect on execution at EL0.

In a system where the PE resets into EL1, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES1.

WXN, bit [19]

Write permission implies XN (Execute-never). For the EL1&0 translation regime, this bit can force all memory regions that are writable to be treated as XN. The possible values of this bit are:

WXN	Meaning
0b0	This control has no effect on memory access permissions.
0b1	Any region that is writable in the EL1&0 translation regime is forced to XN for accesses from software executing at EL1 or EL0.

The WXN bit is permitted to be cached in a TLB.

When ARMv8.1-VHE is implemented, and the value of HCR_EL2.{E2H, TGE} is {1, 1}, this bit has no effect on the PE.

In a system where the PE resets into EL1, this field resets to an architecturally UNKNOWN value.

nTWE, bit [18]

When HCR_EL2.TGE == 1 and (SCR_EL3.NS == 1 or SCR_EL3.EEL2 == 1):

Traps EL0 execution of WFE instructions to EL2, from both Execution states.

nTWE	Meaning
0b0	Any attempt to execute a WFE instruction at EL0 is trapped to EL2, if the instruction would otherwise have caused the PE to enter a low-power state.
0b1	This control does not cause any instructions to be trapped.

In AArch32 state, the attempted execution of a conditional WFE instruction is only trapped if the instruction passes its condition code check.

Note

Since a WFE or WFI can complete at any time, even without a Wakeup event, the traps on WFE of WFI are not guaranteed to be taken, even if the WFE or WFI is executed when there is no Wakeup event. The only guarantee is that if the instruction does not complete in finite time in the absence of a Wakeup event, the trap will be taken.

When ARMv8.1-VHE is implemented, and the value of HCR_EL2.{E2H, TGE} is {1, 1}, this bit has no effect on execution at EL0.

In a system where the PE resets into EL1, this field resets to an architecturally UNKNOWN value.

Otherwise:

Traps EL0 execution of WFE instructions to EL1, from both Execution states.

nTWE	Meaning
0b0	Any attempt to execute a WFE instruction at EL0 is trapped to EL1, if the instruction would otherwise have caused the PE to enter a low-power state.
0b1	This control does not cause any instructions to be trapped.

In AArch32 state, the attempted execution of a conditional WFE instruction is only trapped if the instruction passes its condition code check.

Note

Since a WFE or WFI can complete at any time, even without a Wakeup event, the traps on WFE of WFI are not guaranteed to be taken, even if the WFE or WFI is executed when there is no Wakeup event. The only guarantee is that if the instruction does not complete in finite time in the absence of a Wakeup event, the trap will be taken.

When ARMv8.1-VHE is implemented, and the value of HCR_EL2.{E2H, TGE} is {1, 1}, this bit has no effect on execution at EL0.

In a system where the PE resets into EL1, this field resets to an architecturally UNKNOWN value.

Bit [17]

Reserved, RES0.

nTWI, bit [16]

When HCR_EL2.TGE == 1 and (SCR_EL3.NS == 1 or SCR_EL3.EEL2 == 1):

Traps EL0 execution of WFI instructions to EL2, from both Execution states.

nTWI	Meaning
0b0	Any attempt to execute a WFI instruction at EL0 is trapped to EL2, if the instruction would otherwise have caused the PE to enter a low-power state.
0b1	This control does not cause any instructions to be trapped.

In AArch32 state, the attempted execution of a conditional WFI instruction is only trapped if the instruction passes its condition code check.

Note

Since a WFE or WFI can complete at any time, even without a Wakeup event, the traps on WFE of WFI are not guaranteed to be taken, even if the WFE or WFI is executed when there is no Wakeup event. The only guarantee is that if the instruction does not complete in finite time in the absence of a Wakeup event, the trap will be taken.

When ARMv8.1-VHE is implemented, and the value of HCR_EL2.{E2H, TGE} is {1, 1}, this bit has no effect on execution at EL0.

In a system where the PE resets into EL1, this field resets to an architecturally UNKNOWN value.

Otherwise:

Traps EL0 execution of WFI instructions to EL1, from both Execution states.

nTWI	Meaning
0b0	Any attempt to execute a WFI instruction at EL0 is trapped to EL1, if the instruction would otherwise have caused the PE to enter a low-power state.
0b1	This control does not cause any instructions to be trapped.

In AArch32 state, the attempted execution of a conditional WFI instruction is only trapped if the instruction passes its condition code check.

Note

Since a WFE or WFI can complete at any time, even without a Wakeup event, the traps on WFE of WFI are not guaranteed to be taken, even if the WFE or WFI is executed when there is no Wakeup event. The only guarantee is that if the instruction does not complete in finite time in the absence of a Wakeup event, the trap will be taken.

When ARMv8.1-VHE is implemented, and the value of HCR_EL2.{E2H, TGE} is {1, 1}, this bit has no effect on execution at EL0.

In a system where the PE resets into EL1, this field resets to an architecturally UNKNOWN value.

UCT, bit [15]

Traps EL0 accesses to the CTR_EL0 to EL1, from AArch64 state only.

UCT	Meaning
0b0	Accesses to the CTR_EL0 from EL0 using AArch64 are trapped to EL1.
0b1	This control does not cause any instructions to be trapped.

When ARMv8.1-VHE is implemented, and the value of HCR_EL2.{E2H, TGE} is {1, 1}, this bit has no effect on execution at EL0.

In a system where the PE resets into EL1, this field resets to an architecturally UNKNOWN value.

DZE, bit [14]

Traps EL0 execution of DC ZVA instructions to EL1, from AArch64 state only.

If ARMv8.5-MemTag is implemented, this trap also applies to DC GVA and DC GZVA.

DZE	Meaning
0b0	Any attempt to execute an instruction that this trap applies to at EL0 using AArch64 is trapped to EL1. Reading DCZID_EL0.DZP from EL0 returns 1, indicating that the instructions this trap applies to are not supported.
0b1	This control does not cause any instructions to be trapped.

When ARMv8.1-VHE is implemented, and the value of HCR_EL2.{E2H, TGE} is {1, 1}, this bit has no effect on execution at EL0.

In a system where the PE resets into EL1, this field resets to an architecturally UNKNOWN value.

EnDB, bit [13]

From Armv8.3:

Controls enabling of pointer authentication (using the APDBKey_EL1 key) of instruction addresses in the EL1&0 translation regime.

EnDB	Meaning
0b0	Pointer authentication (using the APDBKey_EL1 key) of data addresses is not enabled.
0b1	Pointer authentication (using the APDBKey_EL1 key) of data addresses is enabled.

Note

This field controls the behavior of the AddPACDB and AuthDB pseudocode functions. Specifically, when the field is 1, AddPACDB returns a copy of a pointer to which a pointer authentication code has been added, and AuthDB returns an authenticated copy of a pointer. When the field is 0, both of these functions are NOP.

In a system where the PE resets into EL1, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

I, bit [12]

Instruction access Cacheability control, for accesses at EL0 and EL1:

I	Meaning
0b0	All instruction access to Normal memory from EL0 and EL1 are Non-cacheable for all levels of instruction and unified cache. If the value of SCTLR_EL1.M is 0, instruction accesses from stage 1 of the EL1&0 translation regime are to Normal, Outer Shareable, Inner Non-cacheable, Outer Non-cacheable memory.
0b1	This control has no effect on the Cacheability of instruction access to Normal memory from EL0 and EL1. If the value of SCTLR_EL1.M is 0, instruction accesses from stage 1 of the EL1&0 translation regime are to Normal, Outer Shareable, Inner Write-Through, Outer Write-Through memory.

When the value of the HCR_EL2.DC bit is 1, then instruction access to Normal memory from EL0 and EL1 are Cacheable regardless of the value of the SCTLR_EL1.I bit.

When ARMv8.1-VHE is implemented, and the value of HCR_EL2.{E2H, TGE} is {1, 1}, this bit has no effect on the PE.

In a system where the PE resets into EL1, this field resets to 0.

EOS, bit [11]

From Armv8.5:

Exception Exit is Context Synchronizing. The defined values are:

EOS	Meaning
0b0	An exception return from EL1 is not a context synchronizing event
0b1	An exception return from EL1 is a context synchronizing event

When ARMv8.1-VHE is implemented, and the value of HCR_EL2.{E2H, TGE} is {1,1}, this bit has no effect on execution at EL0.

In a system where the PE resets into EL1, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES1.

EnRCTX, bit [10]

From Armv8.5:

Enable EL0 Access to the AArch32 CFPRCTX, DVPRCTX and CPPRCTX instructions, and the AArch64 CFP RCTX, DVP RCT and CPP RCTX instructions. The defined values are:

EnRCTX	Meaning
0b0	EL0 access to these instructions is disabled, and these instructions are trapped to EL1.
0b1	EL0 access to these instructions is enabled.

When ARMv8.1-VHE is implemented, and the value of HCR_EL2.{E2H, TGE} is {1,1}, this bit has no effect on execution at EL0.

In a system where the PE resets into EL1, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

UMA, bit [9]

User Mask Access. Traps EL0 execution of MSR and MRS instructions that access the PSTATE.{D, A, I, F} masks to EL1, from AArch64 state only.

UMA	Meaning
0b0	Any attempt at EL0 using AArch64 to execute an MRS, MSR(register), or MSR(immediate) instruction that accesses the DAIF is trapped to EL1.
0b1	This control does not cause any instructions to be trapped.

When ARMv8.1-VHE is implemented, and the value of HCR_EL2.{E2H, TGE} is {1, 1}, this bit has no effect on execution at EL0.

In a system where the PE resets into EL1, this field resets to an architecturally UNKNOWN value.

SED, bit [8]

SETEND instruction disable. Disables SETEND instructions at EL0 using AArch32.

SED	Meaning
0b0	SETEND instruction execution is enabled at EL0 using AArch32.
0b1	SETEND instructions are UNDEFINED at EL0 using AArch32.

If the implementation does not support mixed-endian operation at any Exception level, this bit is RES1.

If EL0 cannot use AArch32, this bit is RES1.

When ARMv8.1-VHE is implemented, and the value of HCR_EL2.{E2H, TGE} is {1, 1}, this bit has no effect on execution at EL0.

In a system where the PE resets into EL1, this field resets to an architecturally UNKNOWN value.

ITD, bit [7]

IT Disable. Disables some uses of IT instructions at EL0 using AArch32.

ITD	Meaning
0b0	All IT instruction functionality is enabled at EL0 using AArch32.
0b1	Any attempt at EL0 using AArch32 to execute any of the following is UNDEFINED: • All encodings of the IT instruction with hw1[3:0]!=1000. • All encodings of the subsequent instruction with the following values for hw1: ◦ 0b11xxxxxxxxxxxxx: All 32-bit instructions, and the 16-bit instructions B, UDF, SVC, LDM, and STM. ◦ 0b1011xxxxxxxxxxx: All instructions in 'Miscellaneous 16-bit instructions' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section F3.2.5. ◦ 0b10100xxxxxxxxxx: ADD Rd, PC, #imm ◦ 0b01001xxxxxxxxxx: LDR Rd, [PC, #imm] ◦ 0b0100x1xxx1111xxx: ADD Rdn, PC; CMP Rn, PC; MOV Rd, PC; BX PC; BLX PC. ◦ 0b010001xx1xxxx111: ADD PC, Rm; CMP PC, Rm; MOV PC, Rm. This pattern also covers unpredictable cases with BLX Rn. These instructions are always UNDEFINED, regardless of whether they would pass or fail the condition code check that applies to them as a result of being in an IT block. It is IMPLEMENTATION DEFINED whether the IT instruction is treated as: • A 16-bit instruction, that can only be followed by another 16-bit instruction. • The first half of a 32-bit instruction. This means that, for the situations that are UNDEFINED, either the second 16-bit instruction or the 32-bit instruction is UNDEFINED. An implementation might vary dynamically as to whether IT is treated as a 16-bit instruction or the first half of a 32-bit instruction.

If an instruction in an active IT block that would be disabled by this field sets this field to 1 then behavior is CONSTRAINED UNPREDICTABLE. For more information see 'Changes to an ITD control by an instruction in an IT block' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section E1.2.4

If EL0 cannot use AArch32, this bit is RES1.

ITD is optional, but if it is implemented in the SCTLR then it must also be implemented in the SCTLR_EL1. If it is not implemented then this bit is RAZ/WI.

When ARMv8.1-VHE is implemented, and the value of HCR_EL2.{E2H, TGE} is {1, 1}, this bit has no effect on execution at EL0.

In a system where the PE resets into EL1, this field resets to an architecturally UNKNOWN value.

nAA, bit [6]

When ARMv8.4-LSE is implemented:

Non-aligned access. This bit controls generation of Alignment faults at EL1 and EL0 under certain conditions.

nAA	Meaning
0b0	LDAPR, LDAPRH, LDAPUR, LDAPURH, LDAPURSH, LDAPURSW, LDAPURW, LDAR, LDARH, LDLAR, LDLARH, STLLR, STLLRH, STLR, STLRH, STLUR, and STLURH generate an Alignment fault if all bytes being accessed are not within a single 16-byte quantity, aligned to 16 bytes for accesses.
0b1	This control bit does not cause LDAPR, LDAPRH, LDAPUR, LDAPURH, LDAPURSH, LDAPURSW, LDAPURW, LDAR, LDARH, LDLAR, LDLARH, STLLR, STLLRH, STLR, STLRH, STLUR, or STLURH to generate an Alignment fault if all bytes being accessed are not within a single 16-byte quantity, aligned to 16 bytes.

When ARMv8.1-VHE is implemented, and the value of HCR_EL2.{E2H, TGE} is {1, 1}, this bit has no effect on execution at EL0.

In a system where the PE resets into EL1, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

CP15BEN, bit [5]

System instruction memory barrier enable. Enables accesses to the DMB, DSB, and ISB System instructions in the (coproc==0b1111) encoding space from EL0:

CP15BEN	Meaning
0b0	EL0 using AArch32: EL0 execution of the CP15DMB, CP15DSB, and CP15ISB instructions is UNDEFINED.
0b1	EL0 using AArch32: EL0 execution of the CP15DMB, CP15DSB, and CP15ISB instructions is enabled.

If EL0 cannot use AArch32, this bit is RES0.

CP15BEN is optional, but if it is implemented in the SCTLR then it must also be implemented in the SCTLR_EL1. If it is not implemented then this bit is RAO/WI.

When ARMv8.1-VHE is implemented, and the value of HCR_EL2.{E2H, TGE} is {1, 1}, this bit has no effect on execution at EL0.

In a system where the PE resets into EL1, this field resets to an architecturally UNKNOWN value.

SA0, bit [4]

SP Alignment check enable for EL0. When set to 1, if a load or store instruction executed at EL0 uses the SP as the base address and the SP is not aligned to a 16-byte boundary, then a SP alignment fault exception is generated. For more information, see 'SP alignment checking' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section D1 (The AArch64 System Level Programmers' Model).

When ARMv8.1-VHE is implemented, and the value of HCR_EL2.{E2H, TGE} is {1, 1}, this bit has no effect on execution at EL0.

In a system where the PE resets into EL1, this field resets to an architecturally UNKNOWN value.

SA, bit [3]

SP Alignment check enable. When set to 1, if a load or store instruction executed at EL1 uses the SP as the base address and the SP is not aligned to a 16-byte boundary, then a SP alignment fault exception is generated. For more information, see 'SP alignment checking' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section D1 (The AArch64 System Level Programmers' Model).

When ARMv8.1-VHE is implemented, and the value of HCR_EL2.{E2H, TGE} is {1, 1}, this bit has no effect on the PE.

In a system where the PE resets into EL1, this field resets to an architecturally UNKNOWN value.

C, bit [2]

Cacheability control, for data accesses.

C	Meaning
0b0	All data access to Normal memory from EL0 and EL1, and all Normal memory accesses to the EL1&0 stage 1 translation tables, are Non-cacheable for all levels of data and unified cache.
0b1	This control has no effect on the Cacheability of: • Data access to Normal memory from EL0 and EL1. • Normal memory accesses to the EL1&0 stage 1 translation tables.

When the value of the HCR_EL2.DC bit is 1, the PE ignores SCLTR.C. This means that Non-secure EL0 and Non-secure EL1 data accesses to Normal memory are Cacheable.

When ARMv8.1-VHE is implemented, and the value of HCR_EL2.{E2H, TGE} is {1, 1}, this bit has no effect on the PE.

In a system where the PE resets into EL1, this field resets to 0.

A, bit [1]

Alignment check enable. This is the enable bit for Alignment fault checking at EL1 and EL0.

A	Meaning
0b0	Alignment fault checking disabled when executing at EL1 or EL0. Instructions that load or store one or more registers, other than load/store exclusive and load-acquire/store-release, do not check that the address being accessed is aligned to the size of the data element(s) being accessed.
0b1	Alignment fault checking enabled when executing at EL1 or EL0. All instructions that load or store one or more registers have an alignment check that the address being accessed is aligned to the size of the data element(s) being accessed. If this check fails it causes an Alignment fault, which is taken as a Data Abort exception.

Load/store exclusive and load-acquire/store-release instructions have an alignment check regardless of the value of the A bit.

When ARMv8.1-VHE is implemented, and the value of HCR_EL2.{E2H, TGE} is {1, 1}, this bit has no effect on execution at EL0.

In a system where the PE resets into EL1, this field resets to an architecturally UNKNOWN value.

M, bit [0]

MMU enable for EL1 and EL0 stage 1 address translation. Possible values of this bit are:

M	Meaning
0b0	EL1 and EL0 stage 1 address translation disabled. See the SCTLR_EL1.I field for the behavior of instruction accesses to Normal memory.
0b1	EL1 and EL0 stage 1 address translation enabled.

If the value of HCR_EL2.{DC, TGE} is not {0, 0} then in Non-secure state the PE behaves as if the value of the SCTLR_EL1.M field is 0 for all purposes other than returning the value of a direct read of the field.

When ARMv8.1-VHE is implemented, and the value of HCR_EL2.{E2H, TGE} is {1, 1}, this bit has no effect on the PE.

In a system where the PE resets into EL1, this field resets to 0.

Accessing the SCTLR_EL1

When HCR_EL2.E2H is 1, without explicit synchronization, access from EL3 using the mnemonic SCTLR_EL1 or SCTLR_EL12 are not guaranteed to be ordered with respect to accesses using the other mnemonic.

Accesses to this register use the following encodings:

```
MRS <Xt>, SCTLR_EL1
```

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b11	0b000	0b0001	0b0000	0b000
~~0b11~~	~~0b0001~~	~~0b000~~	~~0b000~~	~~0b0000~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TRVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<NV2,NV1,NV> == '111' then
        return NVMem[0x110];
    else
        return SCTLR_EL1;
elsif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' then
        return SCTLR_EL2;
    else
        return SCTLR_EL1;
elsif PSTATE.EL == EL3 then
    return SCTLR_EL1;
```

```
MSR SCTLR_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b11	0b000	0b0001	0b0000	0b000
~~0b11~~	~~0b0001~~	~~0b000~~	~~0b000~~	~~0b0000~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<NV2,NV1,NV> == '111' then
        NVMem[0x110] = X[t];
    else
        SCTLR_EL1 = X[t];
elsif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' then
        SCTLR_EL2 = X[t];
    else
        SCTLR_EL1 = X[t];
elsif PSTATE.EL == EL3 then
    SCTLR_EL1 = X[t];
```

```
MRS <Xt>, SCTLR_EL12
```

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b11	0b101	0b0001	0b0000	0b000
~~0b11~~	~~0b0001~~	~~0b101~~	~~0b000~~	~~0b0000~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '101' then
        return NVMem[0x110];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if EL2Enabled() && HCR_EL2.E2H == '1' then
    if PSTATE.EL == EL2 then
        return SCTLR_EL1;
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if EL2Enabled() && HCR_EL2.E2H == '1' then
        return SCTLR_EL1;
else
    else
        UNDEFINED;
```

```
MSR SCTLR_EL12, <Xt>
```

op0	op1	CRn	CRm	op2
op0	CRn	op1	op2	CRm
0b11	0b101	0b0001	0b0000	0b000
0b11	0b0001	0b101	0b000	0b0000

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '101' then
        NVMem[0x110] = X[t];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if EL2Enabled() && HCR_EL2.E2H == '1' then
    if PSTATE.EL == EL2 then
        SCTLR_EL1 = X[t];
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if EL2Enabled() && HCR_EL2.E2H == '1' then
        SCTLR_EL1 = X[t];
else
    else
        UNDEFINED;
```

27~~13~~/03~~12~~/2019~~2018~~ 21~~16~~:59~~42~~; e5e4db499bf9867a4b93324c4dbac985d3da93766~~379d01e197f1d40720d32d0f84e419e9187e009~~

(old) htmldiff from- (new)

SCTLR_EL3, System Control Register (EL3)

The SCTLR_EL3 characteristics are:

Purpose

Provides top level control of the system, including its memory system, at EL3.

Configuration

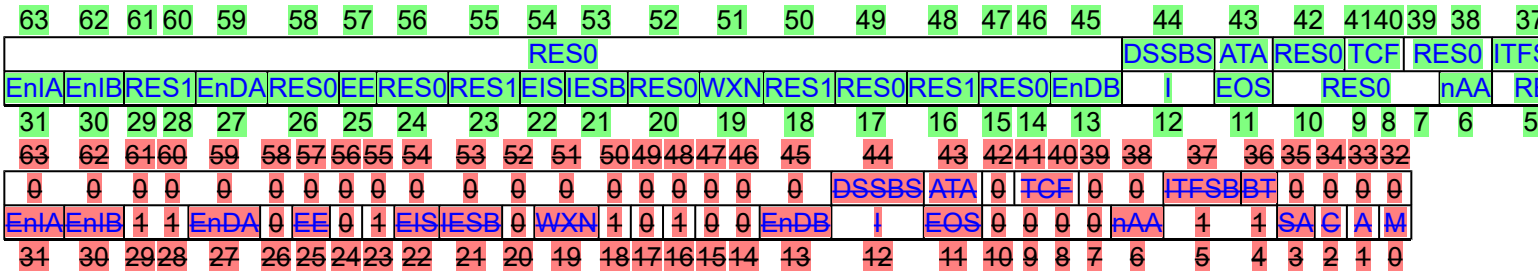
Some or all RW fields of this register have defined reset values. These apply only if the PE resets into EL3 using AArch64. Otherwise, RW fields in this register reset to architecturally UNKNOWN values.

Attributes

SCTLR_EL3 is a 64-bit register.

Field descriptions

The SCTLR_EL3 bit assignments are:



Bits [63:45]

Reserved, RES0.

DSSBS, bit [44]

From Armv8.5:

Default PSTATE.SSBS value on Exception Entry. The defined values are:

DSSBS	Meaning
0b0	PSTATE.SSBS is set to 0 on an exception to EL3
0b1	PSTATE.SSBS is set to 1 on an exception to EL3

In a system where the PE resets into EL3, this field resets to an IMPLEMENTATION DEFINED value.

Otherwise:

Reserved, RES0.

ATA, bit [43]

When ARMv8.5-MemTag is implemented:

Allocation Tag Access in EL3. Controls EL3 access to Allocation Tags.

When access to Allocation Tags is prevented:

- Instructions which Load or Store data are Unchecked.
- Instructions which Load or Store Allocation Tags treat the Allocation Tag as RAZ/WI.
- Instructions which insert Logical Address Tags into addresses treat the Allocation Tag used to generate the Logical Address Tag as 0.
- Cache maintenance instructions which invalidate Allocation Tags from caches behave as the equivalent Clean and Invalidate operation on Allocation Tags.

ATA	Meaning
0b0	Access to Allocation Tags is prevented.
0b1	Access to Allocation Tags is not prevented.

This bit is permitted to be cached in a TLB.

In a system where the PE resets into EL3, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bit [42]

Reserved, RES0.

TCF, bits [41:40]

When ARMv8.5-MemTag is implemented:

Tag Check Fail in EL3. Controls the effect of tag check fails due to Loads and Stores in EL3.

TCF	Meaning
0b00	Tag check fails have no effect on the PE.
0b01	Tag check fails causes a synchronous exception.
0b10	Tag check fails are asynchronously accumulated.

The value 0b11 is reserved.

This field is permitted to be cached in a TLB.

In a system where the PE resets into EL3, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [39:38]

Reserved, RES0.

ITFSB, bit [37]

When ARMv8.5-MemTag is implemented:

When asynchronous~~synchronous~~ exceptions are ~~not~~ being generated by Tag Check fails which are generated for Loads and Stores at any exception level, controls the auto-synchronisaton of Tag Check fails into TFSRE0_EL1 and TFSR_ELx.

ITFSB	Meaning
0b0	Tag check fails are not synchronized on entry to EL3.
0b1	Tag check fails are synchronized on entry to EL3.

In a system where the PE resets into EL3, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

BT, bit [36]

When ARMv8.5-BTI is implemented:

PAC Branch Type compatibility at EL3.

BT	Meaning
0b0	When the PE is executing at EL3, PACIASP and PACIBSP are compatible with PSTATE.BTYPE == 0b11.
0b1	When the PE is executing at EL3, PACIASP and PACIBSP are not compatible with PSTATE.BTYPE == 0b11.

In a system where the PE resets into EL3, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [35:32]

Reserved, RES0.

EnIA, bit [31]

From Armv8.3:

Controls enabling of pointer authentication (using the APIAKey_EL1 key) of instruction addresses in the EL3 translation regime.

Possible values of this bit are:

EnIA	Meaning
0b0	Pointer authentication (using the APIAKey_EL1 key) of instruction addresses is not enabled.
0b1	Pointer authentication (using the APIAKey_EL1 key) of instruction addresses is enabled.

Note

This field controls the behavior of the AddPACIA and AuthIA pseudocode functions. Specifically, when the field is 1, AddPACIA returns a copy of a pointer to which a pointer authentication code has been added, and AuthIA returns an authenticated copy of a pointer. When the field is 0, both of these functions are NOP.

In a system where the PE resets into EL3, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EnIB, bit [30]

From Armv8.3:

Controls enabling of pointer authentication (using the APIBKey_EL1 key) of instruction addresses in the EL3 translation regime.

Possible values of this bit are:

EnIB	Meaning
0b0	Pointer authentication (using the APIBKey_EL1 key) of instruction addresses is not enabled.
0b1	Pointer authentication (using the APIBKey_EL1 key) of instruction addresses is enabled.

Note

This field controls the behavior of the AddPACIB and AuthIB pseudocode functions. Specifically, when the field is 1, AddPACIB returns a copy of a pointer to which a pointer authentication code has been added, and AuthIB returns an authenticated copy of a pointer. When the field is 0, both of these functions are NOP.

In a system where the PE resets into EL3, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [29:28]

Reserved, RES1.

EnDA, bit [27]

From Armv8.3:

Controls enabling of pointer authentication (using the APDAKey_EL1 key) of instruction addresses in the EL3 translation regime.

EnDA	Meaning
0b0	Pointer authentication (using the APDAKey_EL1 key) of data addresses is not enabled.
0b1	Pointer authentication (using the APDAKey_EL1 key) of data addresses is enabled.

Note

This field controls the behavior of the AddPACDA and AuthDA pseudocode functions. Specifically, when the field is 1, AddPACDA returns a copy of a pointer to which a pointer authentication code has been added, and AuthDA returns an authenticated copy of a pointer. When the field is 0, both of these functions are NOP.

In a system where the PE resets into EL3, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bit [26]

Reserved, RES0.

EE, bit [25]

Endianness of data accesses at EL3, and stage 1 translation table walks in the EL3 translation regime.

The possible values of this bit are:

EE	Meaning
0b0	Explicit data accesses at EL3, and stage 1 translation table walks in the EL3 translation regime are little-endian.
0b1	Explicit data accesses at EL3, and stage 1 translation table walks in the EL3 translation regime are big-endian.

If an implementation does not provide Big-endian support at Exception Levels higher than EL0, this bit is RES0.

If an implementation does not provide Little-endian support at Exception Levels higher than EL0, this bit is RES1.

The EE bit is permitted to be cached in a TLB.

In a system where the PE resets into EL3, this field resets to an IMPLEMENTATION DEFINED value.

Bit [24]

Reserved, RES0.

Bit [23]

Reserved, RES1.

EIS, bit [22]

From Armv8.5:

Exception Entry is Context Synchronizing.

EIS	Meaning
0b0	The taking of an exception to EL3 is not a context synchronizing event.
0b1	The taking of an exception to EL3 is a context synchronizing event.

In a system where the PE resets into EL3, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES1.

IESB, bit [21]

When ARMv8.2-IESB is implemented:

Implicit Error Synchronization event enable.

IESB	Meaning
0b0	Disabled.
0b1	An implicit error synchronization event is added: • After each exception taken to EL3. • Before the operational pseudocode of each ERET instruction executed at EL3.

When the PE is in Debug state, the effect of this field is CONSTRAINED UNPREDICTABLE, and its Effective value might be 0 or 1 regardless of the value of the field. If the Effective value of the field is 1, then an implicit error synchronization event is added after each DCPSx instruction taken to EL3 and before each DRPS instruction executed at EL3, in addition to the other cases where it is added.

When ARMv8.4-DFE is implemented, and the Effective value of SCR_EL3.NMEA is 1, this field is ignored and its Effective value is 1.

In a system where the PE resets into EL3, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bit [20]

Reserved, RES0.

WXN, bit [19]

Write permission implies XN (Execute-never). For the EL3 translation regime, this bit can force all memory regions that are writable to be treated as XN. The possible values of this bit are:

WXN	Meaning
0b0	This control has no effect on memory access permissions.
0b1	Any region that is writable in the EL3 translation regime is forced to XN for accesses from software executing at EL3.

The WXN bit is permitted to be cached in a TLB.

In a system where the PE resets into EL3, this field resets to an architecturally UNKNOWN value.

Bit [18]

Reserved, RES1.

Bit [17]

Reserved, RES0.

Bit [16]

Reserved, RES1.

Bits [15:14]

Reserved, RES0.

EnDB, bit [13]

From Armv8.3:

Controls enabling of pointer authentication (using the APDBKey_EL1 key) of instruction addresses in the EL3 translation regime.

EnDB	Meaning
0b0	Pointer authentication (using the APDBKey_EL1 key) of data addresses is not enabled.
0b1	Pointer authentication (using the APDBKey_EL1 key) of data addresses is enabled.

Note

This field controls the behavior of the AddPACDB and AuthDB pseudocode functions. Specifically, when the field is 1, AddPACDB returns a copy of a pointer to which a pointer authentication code has been added, and AuthDB returns an authenticated copy of a pointer. When the field is 0, both of these functions are NOP.

In a system where the PE resets into EL3, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

I, bit [12]

Instruction access Cacheability control, for accesses at EL3:

I	Meaning
0b0	All instruction access to Normal memory from EL3 are Non-cacheable for all levels of instruction and unified cache. If the value of SCTLR_EL3.M is 0, instruction accesses from stage 1 of the EL3 translation regime are to Normal, Outer Shareable, Inner Non-cacheable, Outer Non-cacheable memory.
0b1	This control has no effect on the Cacheability of instruction access to Normal memory from EL3. If the value of SCTLR_EL3.M is 0, instruction accesses from stage 1 of the EL3 translation regime are to Normal, Outer Shareable, Inner Write-Through, Outer Write-Through memory.

This bit has no effect on the EL1&0, EL2, or EL2&0 translation regimes.

In a system where the PE resets into EL3, this field resets to 0.

EOS, bit [11]

From Armv8.5:

Exception Exit is Context Synchronizing.

EOS	Meaning
0b0	An exception return from EL3 is not a context synchronizing event
0b1	An exception return from EL3 is a context synchronizing event

In a system where the PE resets into EL3, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES1.

Bits [10:7]

Reserved, RES0.

nAA, bit [6]

When ARMv8.4-LSE is implemented:

Non-aligned access. This bit controls generation of Alignment faults at EL3 under certain conditions.

nAA	Meaning
0b0	LDAPR, LDAPRH, LDAPUR, LDAPURH, LDAPURSH, LDAPURSW, LDAPURW, LDAR, LDARH, LDLAR, LDLARH, STLLR, STLLRH, STLR, STLRH, STLUR, and STLURH generate an Alignment fault if all bytes being accessed are not within a single 16-byte quantity, aligned to 16 bytes for accesses.
0b1	This control bit does not cause LDAPR, LDAPRH, LDAPUR, LDAPURH, LDAPURSH, LDAPURSW, LDAPURW, LDAR, LDARH, LDLAR, LDLARH, STLLR, STLLRH, STLR, STLRH, STLUR, or STLURH to generate an Alignment fault if all bytes being accessed are not within a single 16-byte quantity, aligned to 16 bytes.

In a system where the PE resets into EL3, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [5:4]

Reserved, RES1.

SA, bit [3]

SP Alignment check enable. When set to 1, if a load or store instruction executed at EL3 uses the SP as the base address and the SP is not aligned to a 16-byte boundary, then a SP alignment fault exception is generated. For more information, see 'SP alignment checking' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section D1 (The AArch64 System Level Programmers' Model).

In a system where the PE resets into EL3, this field resets to an architecturally UNKNOWN value.

C, bit [2]

Cacheability control, for data accesses.

C	Meaning
0b0	All data access to Normal memory from EL3, and all Normal memory accesses to the EL3 translation tables, are Non-cacheable for all levels of data and unified cache.
0b1	This control has no effect on the Cacheability of: • Data access to Normal memory from EL3. • Normal memory accesses to the EL3 translation tables.

This bit has no effect on the EL1&0, EL2, or EL2&0 translation regimes.

In a system where the PE resets into EL3, this field resets to 0.

A, bit [1]

Alignment check enable. This is the enable bit for Alignment fault checking at EL3.

A	Meaning
0b0	Alignment fault checking disabled when executing at EL3. Instructions that load or store one or more registers, other than load/store exclusive and load-acquire/store-release, do not check that the address being accessed is aligned to the size of the data element(s) being accessed.
0b1	Alignment fault checking enabled when executing at EL3. All instructions that load or store one or more registers have an alignment check that the address being accessed is aligned to the size of the data element(s) being accessed. If this check fails it causes an Alignment fault, which is taken as a Data Abort exception.

Load/store exclusive and load-acquire/store-release instructions have an alignment check regardless of the value of the A bit.

In a system where the PE resets into EL3, this field resets to an architecturally UNKNOWN value.

M, bit [0]

MMU enable for EL3 stage 1 address translation. Possible values of this bit are:

M	Meaning
0b0	EL3 stage 1 address translation disabled. See the SCTLR_EL3.I field for the behavior of instruction accesses to Normal memory.
0b1	EL3 stage 1 address translation enabled.

In a system where the PE resets into EL3, this field resets to 0.

Accessing the SCTLR_EL3

Accesses to this register use the following encodings:

```
MRS <Xt>, SCTLR_EL3
```

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b11	0b110	0b0001	0b0000	0b000
~~0b11~~	~~0b0001~~	~~0b110~~	~~0b000~~	~~0b0000~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    return SCTLR_EL3;
```

```
MSR SCTLR_EL3, <Xt>
```

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b11	0b110	0b0001	0b0000	0b000
~~0b11~~	~~0b0001~~	~~0b110~~	~~0b000~~	~~0b0000~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    SCTLR_EL3 = X[t];
```

27~~13~~/03~~12~~/2019~~2018~~ 21~~16~~:59~~42~~: e5e4db499bf9867a4b93324c4dbac985d3da93766379d01e197f1d40720d32d0f84e419e9187e009

(old) htmldiff from- (new)

SCXTNUM_EL1, EL1 Read/Write Software Context Number

The SCXTNUM_EL1 characteristics are:

Purpose

Provides a number that can be used to separate out different context numbers with the EL1 exception level, for the purpose of protecting against side-channels using branch prediction and similar resources.

Configuration

This register is present only when ARMv8.0-CSV2 is implemented. Otherwise, direct accesses to SCXTNUM_EL1 are UNDEFINED.

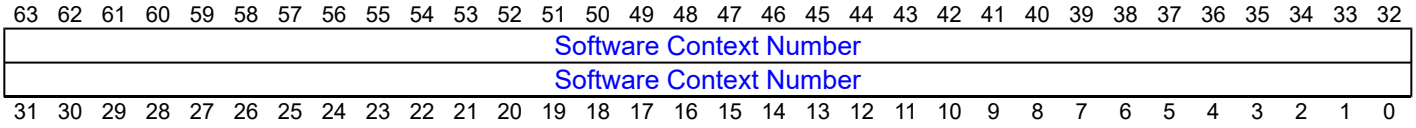
RW fields in this register reset to architecturally UNKNOWN values.

Attributes

SCXTNUM_EL1 is a 64-bit register.

Field descriptions

The SCXTNUM_EL1 bit assignments are:



Bits [63:0]

Software Context Number. A number to identify the context within the EL1 exception level.

This field resets to an architecturally UNKNOWN value.

Accessing the SCXTNUM_EL1

Accesses to this register use the following encodings:

```
MRS <Xt>, SCXTNUM_EL1
```

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b11	0b000	0b1101	0b0000	0b111
~~0b11~~	~~0b1101~~	~~0b000~~	~~0b111~~	~~0b0000~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.EnSCXT == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.EnSCXT == '0' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<NV2,NV1,NV> == '111' then
        return NVMem[0x188];
    else
        return SCXTNUM_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.EnSCXT == '0' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HCR_EL2.E2H == '1' then
        return SCXTNUM_EL2;
    else
        return SCXTNUM_EL1;
elsif PSTATE.EL == EL3 then
    return SCXTNUM_EL1;
```

MSR SCXTNUM_EL1, <Xt>

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b11	0b000	0b1101	0b0000	0b111
~~0b11~~	~~0b1101~~	~~0b000~~	~~0b111~~	~~0b0000~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.EnSCXT == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.EnSCXT == '0' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<NV2,NV1,NV> == '111' then
        NVMem[0x188] = X[t];
    else
        SCXTNUM_EL1 = X[t];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.EnSCXT == '0' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HCR_EL2.E2H == '1' then
        SCXTNUM_EL2 = X[t];
    else
        SCXTNUM_EL1 = X[t];
elsif PSTATE.EL == EL3 then
    SCXTNUM_EL1 = X[t];
```

MRS <Xt>, SCXTNUM_EL12

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b11	0b101	0b1101	0b0000	0b111
~~0b11~~	~~0b1101~~	~~0b101~~	~~0b111~~	~~0b0000~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '101' then
        return NVMem[0x188];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if EL2Enabled() && HCR_EL2.E2H == '1' then
    if PSTATE.EL == EL2 then
        if HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.EnSCXT == '0' then
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return SCXTNUM_EL1;
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if EL2Enabled() && HCR_EL2.E2H == '1' then
        return SCXTNUM_EL1;
else
    else
        UNDEFINED;
```

MSR SCXTNUM_EL12, <Xt>

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b11	0b101	0b1101	0b0000	0b111
~~0b11~~	~~0b1101~~	~~0b101~~	~~0b111~~	~~0b0000~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '101' then
        NVMem[0x188] = X[t];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if EL2Enabled() && HCR_EL2.E2H == '1' then
    if PSTATE.EL == EL2 then
        if HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.EnSCXT == '0' then
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            SCXTNUM_EL1 = X[t];
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if EL2Enabled() && HCR_EL2.E2H == '1' then
        SCXTNUM_EL1 = X[t];
else
    else
        UNDEFINED;
```

27~~13~~/03~~12~~/2019~~2018~~ 21~~16~~:59~~42~~; e5e4db499bf9867a4b93324c4dbac985d3da93766~~379d01e197f1d40720d32d0f84c419e9187e009~~

(old) htmldiff from- (new)

SP_EL2, Stack Pointer (EL2)

The SP_EL2 characteristics are:

Purpose

Holds the stack pointer associated with EL2. When executing at EL2, the value of SPSel. SP determines the current stack pointer:

SPSel.SP	Current stack pointer
0b0	SP_EL0
0b1	SP_EL2

Configuration

This register has no effect if EL2 is not enabled in the current Security state.

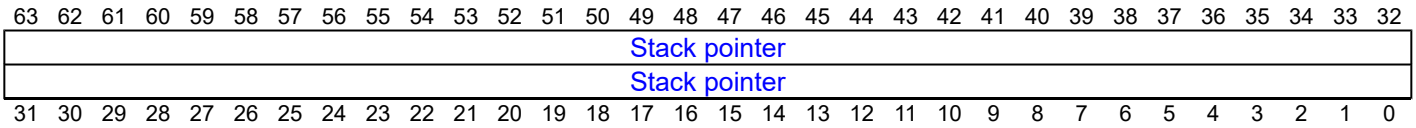
RW fields in this register reset to architecturally UNKNOWN values.

Attributes

SP_EL2 is a 64-bit register.

Field descriptions

The SP_EL2 bit assignments are:



Bits [63:0]

Stack pointer.

This field resets to an architecturally UNKNOWN value.

Accessing the SP_EL2

This accessibility information only applies to accesses using the MRS or MSR instructions.

When the value of SPSel.SP is 1, this register is also accessible at EL2 as the current stack pointer.

Note

When the value of SPSel.SP is 0, SP_EL0 is used as the current stack pointer at all Exception levels.

This accessibility information only applies to accesses using the MRS or MSR instructions.

When the value of SPSel.SP is 1, this register is also accessible at EL2 as the current stack pointer.

Note

When the value of SPSel.SP is 0, SP_EL0 is used as the current stack pointer at all Exception levels.

Accesses to this register use the following encodings:

```
MRS <Xt>, SP_EL2
```

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b11	0b110	0b0100	0b0001	0b000
~~0b11~~	~~0b0100~~	~~0b110~~	~~0b000~~	~~0b0001~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    return SP_EL2;
```

```
MSR SP_EL2, <Xt>
```

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b11	0b110	0b0100	0b0001	0b000
~~0b11~~	~~0b0100~~	~~0b110~~	~~0b000~~	~~0b0001~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    SP_EL2 = X[t];
```

27~~13~~/03~~12~~/2019~~2018~~ 21~~16~~:59~~42~~; e5e4db499bf9867a4b93324c4dbac985d3da93766~~379d01e197f1d40720d32d0f84e419e9187e009~~

(old) htmldiff from- (new)

SPSR_EL1, Saved Program Status Register (EL1)

The SPSR_EL1 characteristics are:

Purpose

Holds the saved process state when an exception is taken to EL1.

Configuration

AArch64 System register SPSR_EL1 bits [31:0] are architecturally mapped to AArch32 System register SPSR_svc[31:0] .

RW fields in this register reset to architecturally UNKNOWN values.

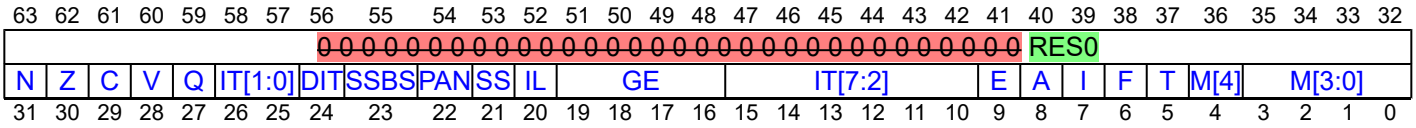
Attributes

SPSR_EL1 is a 64-bit register.

Field descriptions

The SPSR_EL1 bit assignments are:

When exception taken from AArch32 state:



An exception return from EL1 using AArch64 makes SPSR_EL1 become UNKNOWN.

Bits [63:32]

Reserved, RES0.

N, bit [31]

Negative Condition flag. Set to the value of PSTATE.N on taking an exception to EL1, and copied to PSTATE.N on executing an exception return operation in EL1.

This field resets to an architecturally UNKNOWN value.

Z, bit [30]

Zero Condition flag. Set to the value of PSTATE.Z on taking an exception to EL1, and copied to PSTATE.Z on executing an exception return operation in EL1.

This field resets to an architecturally UNKNOWN value.

C, bit [29]

Carry Condition flag. Set to the value of PSTATE.C on taking an exception to EL1, and copied to PSTATE.C on executing an exception return operation in EL1.

This field resets to an architecturally UNKNOWN value.

V, bit [28]

Overflow Condition flag. Set to the value of PSTATE.V on taking an exception to EL1, and copied to PSTATE.V on executing an exception return operation in EL1.

This field resets to an architecturally UNKNOWN value.

Q, bit [27]

Overflow or saturation flag. Set to the value of PSTATE.Q on taking an exception to EL1, and copied to PSTATE.Q on executing an exception return operation in EL1.

This field resets to an architecturally UNKNOWN value.

IT[1:0], bits [26:25]

If-Then. Set to the value of PSTATE.IT[1:0] on taking an exception to EL1, and copied to PSTATE.IT[1:0] on executing an exception return operation in EL1.

On executing an exception return operation in EL1 SPSR_EL1.IT must contain a value that is valid for the instruction being returned to.

This field resets to an architecturally UNKNOWN value.

DIT, bit [24]

When ARMv8.4-DIT is implemented:

Data Independent Timing. Set to the value of PSTATE.DIT on taking an exception to EL1, and copied to PSTATE.DIT on executing an exception return operation in EL1.

This field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

SSBS, bit [23]

When ARMv8.0-SSBS is implemented:

Speculative Store Bypass. Set to the value of PSTATE.SSBS on taking an exception to EL1, and copied to PSTATE.SSBS on executing an exception return operation in EL1.

This field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

PAN, bit [22]

When ARMv8.1-PAN is implemented:

Privileged Access Never. Set to the value of PSTATE.PAN on taking an exception to EL1, and copied to PSTATE.PAN on executing an exception return operation in EL1.

This field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

SS, bit [21]

Software Step. Set to the value of PSTATE.SS on taking an exception to EL1, and conditionally copied to PSTATE.SS on executing an exception return operation in EL1.

This field resets to an architecturally UNKNOWN value.

IL, bit [20]

Illegal Execution state. Set to the value of PSTATE.IL on taking an exception to EL1, and copied to PSTATE.IL on executing an exception return operation in EL1.

This field resets to an architecturally UNKNOWN value.

GE, bits [19:16]

Greater than or Equal flags. Set to the value of PSTATE.GE on taking an exception to EL1, and copied to PSTATE.GE on executing an exception return operation in EL1.

This field resets to an architecturally UNKNOWN value.

IT[7:2], bits [15:10]

If-Then. Set to the value of PSTATE.IT[7:2] on taking an exception to EL1, and copied to PSTATE.IT[7:2] on executing an exception return operation in EL1.

SPSR_EL1.IT must contain a value that is valid for the instruction being returned to.

This field resets to an architecturally UNKNOWN value.

E, bit [9]

Endianness. Set to the value of PSTATE.E on taking an exception to EL1, and copied to PSTATE.E on executing an exception return operation in EL1.

If the implementation does not support big-endian operation, SPSR_EL1.E is RES0. If the implementation does not support little-endian operation, SPSR_EL1.E is RES1. On executing an exception return operation in EL1, if the implementation does not support big-endian operation at the Exception level being returned to, SPSR_EL1.E is RES0, and if the implementation does not support little-endian operation at the Exception level being returned to, SPSR_EL1.E is RES1.

This field resets to an architecturally UNKNOWN value.

A, bit [8]

SError interrupt mask. Set to the value of PSTATE.A on taking an exception to EL1, and copied to PSTATE.A on executing an exception return operation in EL1.

This field resets to an architecturally UNKNOWN value.

I, bit [7]

IRQ interrupt mask. Set to the value of PSTATE.I on taking an exception to EL1, and copied to PSTATE.I on executing an exception return operation in EL1.

This field resets to an architecturally UNKNOWN value.

F, bit [6]

FIQ interrupt mask. Set to the value of PSTATE.F on taking an exception to EL1, and copied to PSTATE.F on executing an exception return operation in EL1.

This field resets to an architecturally UNKNOWN value.

T, bit [5]

T32 Instruction set state. Set to the value of PSTATE.T on taking an exception to EL1, and copied to PSTATE.T on executing an exception return operation in EL1.

This field resets to an architecturally UNKNOWN value.

M[4], bit [4]

Execution state. Set to `0b1`, the value of PSTATE.nRW, on taking an exception to EL1 from AArch32 state, and copied to PSTATE.nRW on executing an exception return operation in EL1.

M[4]	Meaning
`0b1`	AArch32 execution state.

This field resets to an architecturally UNKNOWN value.

M[3:0], bits [3:0]

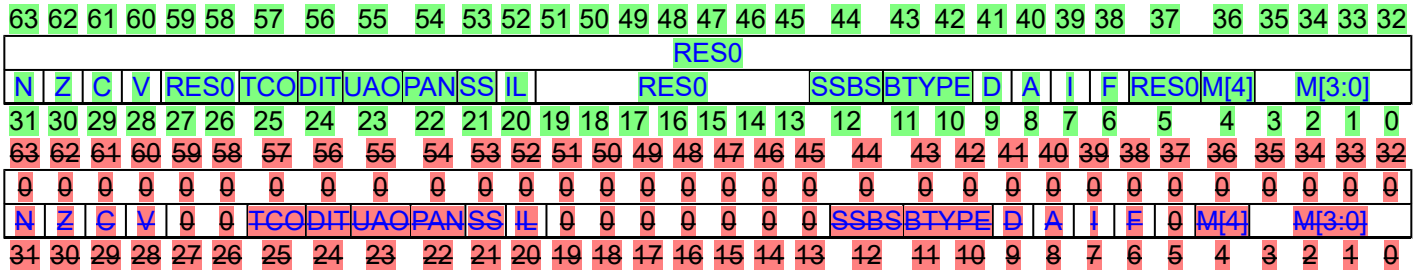
AArch32 Mode. Set to the value of PSTATE.M[3:0] on taking an exception to EL1, and copied to PSTATE.M[3:0] on executing an exception return operation in EL1.

M[3:0]	Meaning
`0b0000`	User.
`0b0001`	FIQ.
`0b0010`	IRQ.
`0b0011`	Supervisor.
`0b0111`	Abort.
`0b1011`	Undefined.
`0b1111`	System.

Other values are reserved. If SPSR_EL1.M[3:0] has a Reserved value, or a value for an unimplemented Exception level, executing an exception return operation in EL1 is an illegal return event, as described in 'Illegal return events from AArch64 state' in the Arm®Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

This field resets to an architecturally UNKNOWN value.

When exception taken from AArch64 state:



An exception return from EL1 using AArch64 makes SPSR_EL1 become UNKNOWN.

Bits [63:32]

Reserved, RES0.

N, bit [31]

Negative Condition flag. Set to the value of PSTATE.N on taking an exception to EL1, and copied to PSTATE.N on executing an exception return operation in EL1.

This field resets to an architecturally UNKNOWN value.

Z, bit [30]

Zero Condition flag. Set to the value of PSTATE.Z on taking an exception to EL1, and copied to PSTATE.Z on executing an exception return operation in EL1.

This field resets to an architecturally UNKNOWN value.

C, bit [29]

Carry Condition flag. Set to the value of PSTATE.C on taking an exception to EL1, and copied to PSTATE.C on executing an exception return operation in EL1.

This field resets to an architecturally UNKNOWN value.

V, bit [28]

Overflow Condition flag. Set to the value of PSTATE.V on taking an exception to EL1, and copied to PSTATE.V on executing an exception return operation in EL1.

This field resets to an architecturally UNKNOWN value.

Bits [27:26]

Reserved, RES0.

TCO, bit [25]

When ARMv8.5-MemTag is implemented:

Tag Check Override. Set to the value of PSTATE.TCO on taking an exception to EL1, and copied to PSTATE.TCO on executing an exception return operation in EL1.

This field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

DIT, bit [24]

When ARMv8.4-DIT is implemented:

Data Independent Timing. Set to the value of PSTATE.DIT on taking an exception to EL1, and copied to PSTATE.DIT on executing an exception return operation in EL1.

This field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

UAO, bit [23]

When ARMv8.2-UAO is implemented:

User Access Override. Set to the value of PSTATE.UAO on taking an exception to EL1, and copied to PSTATE.UAO on executing an exception return operation in EL1.

This field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

PAN, bit [22]

When ARMv8.1-PAN is implemented:

Privileged Access Never. Set to the value of PSTATE.PAN on taking an exception to EL1, and copied to PSTATE.PAN on executing an exception return operation in EL1.

This field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

SS, bit [21]

Software Step. Set to the value of PSTATE.SS on taking an exception to EL1, and conditionally copied to PSTATE.SS on executing an exception return operation in EL1.

This field resets to an architecturally UNKNOWN value.

IL, bit [20]

Illegal Execution state. Set to the value of PSTATE.IL on taking an exception to EL1, and copied to PSTATE.IL on executing an exception return operation in EL1.

This field resets to an architecturally UNKNOWN value.

Bits [19:13]

Reserved, RES0.

SSBS, bit [12]

When ARMv8.0-SSBS is implemented:

Speculative Store Bypass. Set to the value of PSTATE.SSBS on taking an exception to EL1, and copied to PSTATE.SSBS on executing an exception return operation in EL1.

This field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

BTYPE, bits [11:10]

When ARMv8.5-BTI is implemented:

Branch Type Indicator. Set to the value of PSTATE.BTYPE on taking an exception to EL1, and copied to PSTATE.BTYPE on executing an exception return operation in EL1.

This field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

D, bit [9]

Debug exception mask. Set to the value of PSTATE.D on taking an exception to EL1, and copied to PSTATE.D on executing an exception return operation in EL1.

This field resets to an architecturally UNKNOWN value.

A, bit [8]

SError interrupt mask. Set to the value of PSTATE.A on taking an exception to EL1, and copied to PSTATE.A on executing an exception return operation in EL1.

This field resets to an architecturally UNKNOWN value.

I, bit [7]

IRQ interrupt mask. Set to the value of PSTATE.I on taking an exception to EL1, and copied to PSTATE.I on executing an exception return operation in EL1.

This field resets to an architecturally UNKNOWN value.

F, bit [6]

FIQ interrupt mask. Set to the value of PSTATE.F on taking an exception to EL1, and copied to PSTATE.F on executing an exception return operation in EL1.

This field resets to an architecturally UNKNOWN value.

Bit [5]

Reserved, RES0.

M[4], bit [4]

Execution state. Set to `0b0`, the value of PSTATE.nRW, on taking an exception to EL1 from AArch64 state, and copied to PSTATE.nRW on executing an exception return operation in EL1.

M[4]	Meaning
`0b0`	AArch64 execution state.

If AArch32 is not supported at any Exception level, this bit is RES0.

This field resets to an architecturally UNKNOWN value.

M[3:0], bits [3:0]

AArch64 Exception level and selected Stack Pointer.

M[3:0]	Meaning
0b0000	EL0t.
0b0100	EL1t.
0b0101	EL1h.

Other values are reserved. If SPSR_EL1.M[3:0] has a Reserved value, or a value for an unimplemented Exception level, executing an exception return operation in EL1 is an illegal return event, as described in 'Illegal return events from AArch64 state' in the Arm®Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

The bits in this field are interpreted as follows:

- M[3:2] is set to the value of PSTATE.EL on taking an exception to EL1 and copied to PSTATE.EL on executing an exception return operation in EL1.
- M[1] is unused and is 0 for all non-reserved values.
- M[0] is set to the value of PSTATE.SP on taking an exception to EL1 and copied to PSTATE.SP on executing an exception return operation in EL1

This field resets to an architecturally UNKNOWN value.

Accessing the SPSR_EL1

When HCR_EL2.E2H is 1, without explicit synchronization, access from EL3 using the mnemonic SPSR_EL1 or SPSR_EL12 are not guaranteed to be ordered with respect to accesses using the other mnemonic.

Accesses to this register use the following encodings:

```
MRS <Xt>, SPSR_EL1
```

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b11	0b000	0b0100	0b0000	0b000
~~0b11~~	~~0b0100~~	~~0b000~~	~~0b000~~	~~0b0000~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV1> == '01' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<NV2,NV1,NV> == '111' then
        return NVMem[0x160];
    else
        return SPSR_EL1;
elsif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' then
        return SPSR_EL2;
    else
        return SPSR_EL1;
elsif PSTATE.EL == EL3 then
    return SPSR_EL1;
```

```
MSR SPSR_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b11	0b000	0b0100	0b0000	0b000
~~0b11~~	~~0b0100~~	~~0b000~~	~~0b000~~	~~0b0000~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV1> == '01' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<NV2,NV1,NV> == '111' then
        NVMem[0x160] = X[t];
    else
        SPSR_EL1 = X[t];
elsif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' then
        SPSR_EL2 = X[t];
    else
        SPSR_EL1 = X[t];
elsif PSTATE.EL == EL3 then
    SPSR_EL1 = X[t];
```

MRS <Xt>, SPSR_EL12

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b11	0b101	0b0100	0b0000	0b000
~~0b11~~	~~0b0100~~	~~0b101~~	~~0b000~~	~~0b0000~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '101' then
        return NVMem[0x160];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if EL2Enabled() && HCR_EL2.E2H == '1' then
    if PSTATE.EL == EL2 then
        return SPSR_EL1;
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if EL2Enabled() && HCR_EL2.E2H == '1' then
        return SPSR_EL1;
else
    else
        UNDEFINED;
```

MSR SPSR_EL12, <Xt>

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b11	0b101	0b0100	0b0000	0b000
~~0b11~~	~~0b0100~~	~~0b101~~	~~0b000~~	~~0b0000~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '101' then
        NVMem[0x160] = X[t];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if EL2Enabled() && HCR_EL2.E2H == '1' then
    if PSTATE.EL == EL2 then
        SPSR_EL1 = X[t];
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if EL2Enabled() && HCR_EL2.E2H == '1' then
        SPSR_EL1 = X[t];
else
    else
        UNDEFINED;
```

MRS <Xt>, SPSR_EL2

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b11	0b100	0b0100	0b0000	0b000
~~0b11~~	~~0b0100~~	~~0b100~~	~~0b000~~	~~0b0000~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
        return SPSR_EL1;
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    return SPSR_EL2;
elsif PSTATE.EL == EL3 then
    return SPSR_EL2;
```

MSR SPSR_EL2, <Xt>

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b11	0b100	0b0100	0b0000	0b000
~~0b11~~	~~0b0100~~	~~0b100~~	~~0b000~~	~~0b0000~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
        SPSR_EL1 = X[t];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    SPSR_EL2 = X[t];
elsif PSTATE.EL == EL3 then
    SPSR_EL2 = X[t];
```

27~~13~~/03~~12~~/2019~~2018~~ 21~~16~~:59~~42~~; e5e4db499bf9867a4b93324c4dbac985d3da93766~~379d01e197f1d40720d32d0f84e419e9187e009~~

(old) htmldiff from- (new)

(old) htmldiff from- (new)

TCO, Tag Check Override

The TCO characteristics are:

Purpose

When ARMv8.5-MemTag is implemented, this register allows tag checks to be disabled globally.

Configuration

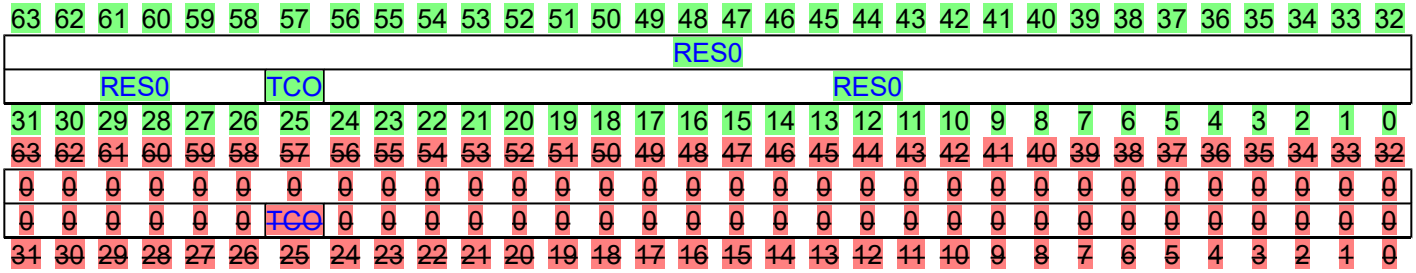
This register is present only when ARMv8.5-MemTag is implemented. Otherwise, direct accesses to TCO are UNDEFINED.

Attributes

TCO is a 64-bit register.

Field descriptions

The TCO bit assignments are:



Bits [63:26]

Reserved, RES0.

TCO, bit [25]

From Armv8.5:

Allows memory tag checks to be globally disabled.

TCO	Meaning
0b0	Loads and Stores are not affected by this control.
0b1	Loads and Stores are unchecked.

Otherwise:

Reserved, RES0.

Bits [24:0]

Reserved, RES0.

Accessing the TCO

For details on the operation of the MSR (immediate) accessor, see MSR (immediate).

Accesses to this register use the following encodings:

MRS <Xt>, TCO

op0	op1	CRn	CRm	op2
~~CRn~~	~~op0~~	~~op1~~	~~op2~~	~~CRm~~
0b11	0b011	0b0100	0b0010	0b111
~~0b0100~~	~~0b11~~	~~0b011~~	~~0b111~~	~~0b0010~~

```
if PSTATE.EL == EL0 then
    return Zeros(38):PSTATE.TCO:Zeros(25);~~UNDEFINED;~~
elsif PSTATE.EL == EL1 then
    return Zeros(38):PSTATE.TCO:Zeros(25);
elsif PSTATE.EL == EL2 then
    return Zeros(38):PSTATE.TCO:Zeros(25);
elsif PSTATE.EL == EL3 then
    return Zeros(38):PSTATE.TCO:Zeros(25);
```

MSR TCO, <Xt>

op0	op1	CRn	CRm	op2
~~CRn~~	~~op0~~	~~op1~~	~~op2~~	~~CRm~~
0b11	0b011	0b0100	0b0010	0b111
~~0b0100~~	~~0b11~~	~~0b011~~	~~0b111~~	~~0b0010~~

```
if PSTATE.EL == EL0 then
    PSTATE.TCO = X[t]<25>;~~UNDEFINED;~~
elsif PSTATE.EL == EL1 then
    PSTATE.TCO = X[t]<25>;
elsif PSTATE.EL == EL2 then
    PSTATE.TCO = X[t]<25>;
elsif PSTATE.EL == EL3 then
    PSTATE.TCO = X[t]<25>;
```

MSR TCO, #<imm>

op0	op1	CRn	op2
~~CRn~~	~~op0~~	~~op1~~	~~op2~~
0b00	0b011	0b0100	0b100
~~0b0100~~	~~0b00~~	~~0b011~~	~~0b100~~

27~~13~~/03~~12~~/2019~~2018~~ 21~~16~~:59~~42~~; e5e4db499bf9867a4b93324c4dbac985d3da93766~~379d01e197f1d40720d32d0f84e419e9187e009~~

(old) htmldiff from- (new)

TCR_EL1, Translation Control Register (EL1)

The TCR_EL1 characteristics are:

Purpose

The control register for stage 1 of the EL1&0 translation regime.

Configuration

AArch64 System register TCR_EL1 bits [31:0] are architecturally mapped to AArch32 System register TTBCR[31:0] .

AArch64 System register TCR_EL1 bits [63:32] are architecturally mapped to AArch32 System register TTBCR2[31:0] .

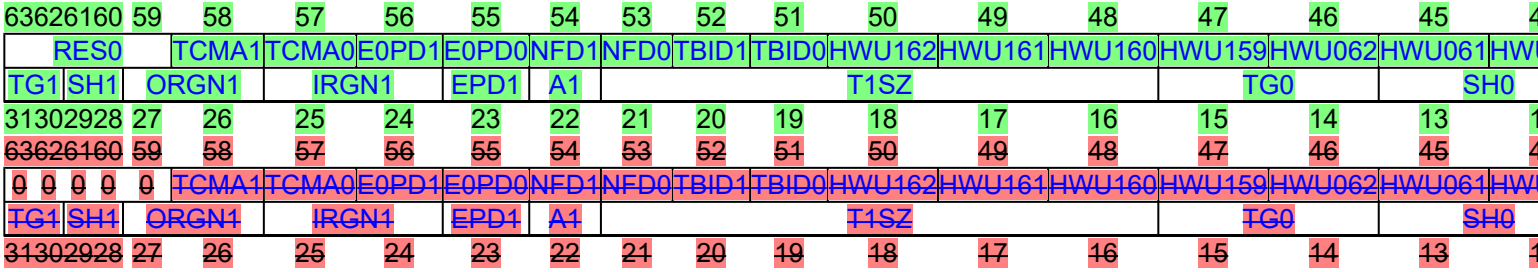
RW fields in this register reset to architecturally UNKNOWN values.

Attributes

TCR_EL1 is a 64-bit register.

Field descriptions

The TCR_EL1 bit assignments are:



Any of the bits in TCR_EL1, other than the A1 bit and the EPDx bits when they have the value 1, are permitted to be cached in a TLB.

Bits [63:59]

Reserved, RES0.

TCMA1, bit [58]

When ARMv8.5-MemTag is implemented:

Controls the generation of Unchecked accesses at EL1, and at EL0 if HCR_EL2.{E2H,TGE}!={1,1}, when address[59:55] = `0b11111`.

TCMA1	Meaning
`0b0`	This control has no effect on the generation of Unchecked accesses at EL1 or EL0.
`0b1`	All accesses at EL1 and EL0 are Unchecked.

Note

Software may change this control bit on a context switch.

This field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TCMA0, bit [57]

When ARMv8.5-MemTag is implemented:

Controls the generation of Unchecked accesses at EL1, and at EL0 if HCR_EL2.{E2H,TGE}!={1,1}, when address[59:55] = 0b00000.

TCMA0	Meaning
0b0	This control has no effect on the generation of Unchecked accesses at EL1 or EL0.
0b1	All accesses at EL1 and EL0 are Unchecked.

Note

Software may change this control bit on a context switch.

This field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

E0PD1, bit [56]

When ARMv8.5-E0PD is implemented:

Faulting control for EL0 access to any address translated by TTBR1_EL1.

E0PD1	Meaning
0b0	EL0 access to any address translated by TTBR1_EL1 will not generate a fault by this mechanism.
0b1	EL0 access to any address translated by TTBR1_EL1 will generate a level 0 translation fault

This field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

E0PD0, bit [55]

When ARMv8.5-E0PD is implemented:

Faulting control for EL0 access to any address translated by TTBR0_EL1.

E0PD0	Meaning
0b0	EL0 access to any address translated by TTBR0_EL1 will not generate a fault by this mechanism.
0b1	EL0 access to any address translated by TTBR0_EL1 will generate a level 0 translation fault

This field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

NFD1, bit [54]

When SVE is implemented:

Non-fault translation table walk disable for stage 1 translations using TTBR1_EL1.

This bit controls whether to perform a stage 1 translation table walk in response to a non-fault access from EL0 for a virtual address that is translated using TTBR1_EL1.

If SVE is implemented, the affected access types include:

- All accesses due to an SVE non-fault contiguous load instruction.
- Accesses due to an SVE first-fault gather load instruction that are not for the First active element. Accesses due to an SVE first-fault contiguous load instruction are not affected.
- Accesses due to prefetch instructions might be affected, but the effect is not architecturally visible.

See 'The Scalable Vector Extension (SVE)', in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile~~, chapter A1~~ for more information.

Defined values are:

NFD1	Meaning
0b0	Does not disable stage 1 translation table walks using TTBR1_EL1.
0b1	A TLB miss on a virtual address that is translated using TTBR1_EL1 due to the specified access types causes the access to fail without taking an exception. No stage 1 translation table walk is performed.

This field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

NFD0, bit [53]

When SVE is implemented:

Non-fault translation table walk disable for stage 1 translations using TTBR0_EL1.

This bit controls whether to perform a stage 1 translation table walk in response to a non-fault access from EL0 for a virtual address that is translated using TTBR0_EL1.

If SVE is implemented, the affected access types include:

- All accesses due to an SVE non-fault contiguous load instruction.
- Accesses due to an SVE first-fault gather load instruction that are not for the First active element. Accesses due to an SVE first-fault contiguous load instruction are not affected.
- Accesses due to prefetch instructions might be affected, but the effect is not architecturally visible.

See 'The Scalable Vector Extension (SVE)', in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile~~, chapter A1~~ for more information.

Defined values are:

NFD0	Meaning
0b0	Does not disable stage 1 translation table walks using TTBR0_EL1.
0b1	A TLB miss on a virtual address that is translated using TTBR0_EL1 due to the specified access types causes the access to fail without taking an exception. No stage 1 translation table walk is performed.

This field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TBID1, bit [52]

When ARMv8.3-PAuth is implemented:

Controls the use of the top byte of instruction addresses for address matching.

TBID1	Meaning
0b0	TCR_EL1.TBI1 applies to Instruction and Data accesses.
0b1	TCR_EL1.TBI1 applies to Data accesses only.

This affects addresses where the address would be translated by tables pointed to by TTBR1_EL1.

This field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TBID0, bit [51]

When ARMv8.3-PAuth is implemented:

Controls the use of the top byte of instruction addresses for address matching.

TBID0	Meaning
0b0	TCR_EL1.TBI0 applies to Instruction and Data accesses.
0b1	TCR_EL1.TBI0 applies to Data accesses only.

This affects addresses where the address would be translated by tables pointed to by TTBR0_EL1.

This field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HWU162, bit [50]

When ARMv8.2-TTPBHA is implemented:

Hardware Use. Indicates IMPLEMENTATION DEFINED hardware use of bit[62] of the stage 1 translation table Block or Page entry for translations using TTBR1_EL1.

HWU162	Meaning
0b0	For translations using TTBR1_EL1, bit[62] of each stage 1 translation table Block or Page entry cannot be used by hardware for an IMPLEMENTATION DEFINED purpose.
0b1	For translations using TTBR1_EL1, bit[62] of each stage 1 translation table Block or Page entry can be used by hardware for an IMPLEMENTATION DEFINED purpose if the value of TCR_EL1.HPD1 is 1.

The Effective value of this field is 0 if the value of TCR_EL1.HPD1 is 0.

This field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HWU161, bit [49]

When ARMv8.2-TTPBHA is implemented:

Hardware Use. Indicates IMPLEMENTATION DEFINED hardware use of bit[61] of the stage 1 translation table Block or Page entry for translations using TTBR1_EL1.

HWU161	Meaning
0b0	For translations using TTBR1_EL1, bit[61] of each stage 1 translation table Block or Page entry cannot be used by hardware for an IMPLEMENTATION DEFINED purpose.
0b1	For translations using TTBR1_EL1, bit[61] of each stage 1 translation table Block or Page entry can be used by hardware for an IMPLEMENTATION DEFINED purpose if the value of TCR_EL1.HPD1 is 1.

The Effective value of this field is 0 if the value of TCR_EL1.HPD1 is 0.

This field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HWU160, bit [48]

When ARMv8.2-TTPBHA is implemented:

Hardware Use. Indicates IMPLEMENTATION DEFINED hardware use of bit[60] of the stage 1 translation table Block or Page entry for translations using TTBR1_EL1.

HWU160	Meaning
0b0	For translations using TTBR1_EL1, bit[60] of each stage 1 translation table Block or Page entry cannot be used by hardware for an IMPLEMENTATION DEFINED purpose.
0b1	For translations using TTBR1_EL1, bit[60] of each stage 1 translation table Block or Page entry can be used by hardware for an IMPLEMENTATION DEFINED purpose if the value of TCR_EL1.HPD1 is 1.

The Effective value of this field is 0 if the value of TCR_EL1.HPD1 is 0.

This field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HWU159, bit [47]

When ARMv8.2-TTPBHA is implemented:

Hardware Use. Indicates IMPLEMENTATION DEFINED hardware use of bit[59] of the stage 1 translation table Block or Page entry for translations using TTBR1_EL1.

HWU159	Meaning
0b0	For translations using TTBR1_EL1, bit[59] of each stage 1 translation table Block or Page entry cannot be used by hardware for an IMPLEMENTATION DEFINED purpose.
0b1	For translations using TTBR1_EL1, bit[59] of each stage 1 translation table Block or Page entry can be used by hardware for an IMPLEMENTATION DEFINED purpose if the value of TCR_EL1.HPD1 is 1.

The Effective value of this field is 0 if the value of TCR_EL1.HPD1 is 0.

This field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HWU062, bit [46]

When ARMv8.2-TTPBHA is implemented:

Hardware Use. Indicates IMPLEMENTATION DEFINED hardware use of bit[62] of the stage 1 translation table Block or Page entry for translations using TTBR0_EL1.

HWU062	Meaning
0b0	For translations using TTBR0_EL1, bit[62] of each stage 1 translation table Block or Page entry cannot be used by hardware for an IMPLEMENTATION DEFINED purpose.
0b1	For translations using TTBR0_EL1, bit[62] of each stage 1 translation table Block or Page entry can be used by hardware for an IMPLEMENTATION DEFINED purpose if the value of TCR_EL1.HPD0 is 1.

The Effective value of this field is 0 if the value of TCR_EL1.HPD0 is 0.

This field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HWU061, bit [45]

When ARMv8.2-TTPBHA is implemented:

Hardware Use. Indicates IMPLEMENTATION DEFINED hardware use of bit[61] of the stage 1 translation table Block or Page entry for translations using TTBR0_EL1.

HWU061	Meaning
0b0	For translations using TTBR0_EL1, bit[61] of each stage 1 translation table Block or Page entry cannot be used by hardware for an IMPLEMENTATION DEFINED purpose.
0b1	For translations using TTBR0_EL1, bit[61] of each stage 1 translation table Block or Page entry can be used by hardware for an IMPLEMENTATION DEFINED purpose if the value of TCR_EL1.HPD0 is 1.

The Effective value of this field is 0 if the value of TCR_EL1.HPD0 is 0.

This field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HWU060, bit [44]

When ARMv8.2-TTPBHA is implemented:

Hardware Use. Indicates IMPLEMENTATION DEFINED hardware use of bit[60] of the stage 1 translation table Block or Page entry for translations using TTBR0_EL1.

HWU060	Meaning
0b0	For translations using TTBR0_EL1, bit[60] of each stage 1 translation table Block or Page entry cannot be used by hardware for an IMPLEMENTATION DEFINED purpose.
0b1	For translations using TTBR0_EL1, bit[60] of each stage 1 translation table Block or Page entry can be used by hardware for an IMPLEMENTATION DEFINED purpose if the value of TCR_EL1.HPD0 is 1.

The Effective value of this field is 0 if the value of TCR_EL1.HPD0 is 0.

This field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HWU059, bit [43]

When ARMv8.2-TTPBHA is implemented:

Hardware Use. Indicates IMPLEMENTATION DEFINED hardware use of bit[59] of the stage 1 translation table Block or Page entry for translations using TTBR0_EL1.

HWU059	Meaning
0b0	For translations using TTBR0_EL1, bit[59] of each stage 1 translation table Block or Page entry cannot be used by hardware for an IMPLEMENTATION DEFINED purpose.
0b1	For translations using TTBR0_EL1, bit[59] of each stage 1 translation table Block or Page entry can be used by hardware for an IMPLEMENTATION DEFINED purpose if the value of TCR_EL1.HPD0 is 1.

The Effective value of this field is 0 if the value of TCR_EL1.HPD0 is 0.

This field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HPD1, bit [42]

When ARMv8.1-HPD is implemented:

Hierarchical Permission Disables. This affects the hierarchical control bits, APTable, PXNTable, and UXNTable, except NSTable, in the translation tables pointed to by TTBR1_EL1.

HPD1	Meaning
0b0	Hierarchical permissions are enabled.
0b1	Hierarchical permissions are disabled.

When disabled, the permissions are treated as if the bits are zero.

This field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HPD0, bit [41]

When ARMv8.1-HPD is implemented:

Hierarchical Permission Disables. This affects the hierarchical control bits, APTable, PXNTable, and UXNTable, except NSTable, in the translation tables pointed to by TTBR0_EL1.

HPD0	Meaning
0b0	Hierarchical permissions are enabled.
0b1	Hierarchical permissions are disabled.

When disabled, the permissions are treated as if the bits are zero.

This field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HD, bit [40]

When ARMv8.1-TTHM is implemented:

Hardware management of dirty state in stage 1 translations from EL0 and EL1.

HD	Meaning
0b0	Stage 1 hardware management of dirty state disabled.
0b1	Stage 1 hardware management of dirty state enabled, only if the HA bit is also set to 1.

This field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HA, bit [39]

When ARMv8.1-TTHM is implemented:

Hardware Access flag update in stage 1 translations from EL0 and EL1.

HA	Meaning
0b0	Stage 1 Access flag update disabled.
0b1	Stage 1 Access flag update enabled.

This field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TBI1, bit [38]

Top Byte ignored - indicates whether the top byte of an address is used for address match for the TTBR1_EL1 region, or ignored and used for tagged addresses. Defined values are:

TBI1	Meaning
0b0	Top Byte used in the address calculation.
0b1	Top Byte ignored in the address calculation.

This affects addresses generated in EL0 and EL1 using AArch64 where the address would be translated by tables pointed to by TTBR1_EL1. It has an effect whether the EL1&0 translation regime is enabled or not.

If ARMv8.3-PAuth is implemented and TCR_EL1.TBID1 is 1, then this field only applies to Data accesses.

Otherwise, if the value of TBI1 is 1 and bit [55] of the target address to be stored to the PC is 1, then bits[63:56] of that target address are also set to 1 before the address is stored in the PC, in the following cases:

- A branch or procedure return within EL0 or EL1.
- An exception taken to EL1.
- An exception return to EL0 or EL1.

This field resets to an architecturally UNKNOWN value.

TBI0, bit [37]

Top Byte ignored - indicates whether the top byte of an address is used for address match for the TTBR0_EL1 region, or ignored and used for tagged addresses. Defined values are:

TBI0	Meaning
0b0	Top Byte used in the address calculation.
0b1	Top Byte ignored in the address calculation.

This affects addresses generated in EL0 and EL1 using AArch64 where the address would be translated by tables pointed to by TTBR0_EL1. It has an effect whether the EL1&0 translation regime is enabled or not.

If ARMv8.3-PAuth is implemented and TCR_EL1.TBID0 is 1, then this field only applies to Data accesses.

Otherwise, if the value of TBI0 is 1 and bit [55] of the target address to be stored to the PC is 0, then bits[63:56] of that target address are also set to 0 before the address is stored in the PC, in the following cases:

- A branch or procedure return within EL0 or EL1.
- An exception taken to EL1.
- An exception return to EL0 or EL1.

This field resets to an architecturally UNKNOWN value.

AS, bit [36]

ASID Size. Defined values are:

AS	Meaning
0b0	8 bit - the upper 8 bits of TTBR0_EL1 and TTBR1_EL1 are ignored by hardware for every purpose except reading back the register, and are treated as if they are all zeros for when used for allocation and matching entries in the TLB.
0b1	16 bit - the upper 16 bits of TTBR0_EL1 and TTBR1_EL1 are used for allocation and matching in the TLB.

If the implementation has only 8 bits of ASID, this field is RES0.

This field resets to an architecturally UNKNOWN value.

Bit [35]

Reserved, RES0.

IPS, bits [34:32]

Intermediate Physical Address Size.

IPS	Meaning
0b000	32 bits, 4GB.
0b001	36 bits, 64GB.
0b010	40 bits, 1TB.
0b011	42 bits, 4TB.
0b100	44 bits, 16TB.
0b101	48 bits, 256TB.
0b110	52 bits, 4PB.

Other values are reserved.

The reserved values behave in the same way as the 0b101 or 0b110 encoding, but software must not rely on this property as the behavior of the reserved values might change in a future revision of the architecture.

The value 0b110 is permitted only if ARMv8.2-LPA is implemented and the translation granule size is 64KB.

In an implementation that supports 52-bit PAs, if the value of this field is not 0b110 or a value treated as 0b110, then bits[51:48] of every translation table base address for the stage of translation controlled by TCR_EL1 are 0b0000.

This field resets to an architecturally UNKNOWN value.

TG1, bits [31:30]

Granule size for the TTBR1_EL1.

TG1	Meaning
0b01	16KB.
0b10	4KB.
0b11	64KB.

Other values are reserved.

If the value is programmed to either a reserved value, or a size that has not been implemented, then the hardware will treat the field as if it has been programmed to an IMPLEMENTATION DEFINED choice of the sizes that has been implemented for all purposes other than the value read back from this register.

It is IMPLEMENTATION DEFINED whether the value read back is the value programmed or the value that corresponds to the size chosen.

This field resets to an architecturally UNKNOWN value.

SH1, bits [29:28]

Shareability attribute for memory associated with translation table walks using TTBR1_EL1.

SH1	Meaning
0b00	Non-shareable.
0b10	Outer Shareable.
0b11	Inner Shareable.

Other values are reserved. The effect of programming this field to a Reserved value is that behavior is CONSTRAINED UNPREDICTABLE, as described in 'Reserved values in AArch64 System registers and translation table entries' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile~~, section K1.2.2~~.

This field resets to an architecturally UNKNOWN value.

ORGN1, bits [27:26]

Outer cacheability attribute for memory associated with translation table walks using TTBR1_EL1.

ORGN1	Meaning
0b00	Normal memory, Outer Non-cacheable.
0b01	Normal memory, Outer Write-Back Read-Allocate Write-Allocate Cacheable.
0b10	Normal memory, Outer Write-Through Read-Allocate No Write-Allocate Cacheable.
0b11	Normal memory, Outer Write-Back Read-Allocate No Write-Allocate Cacheable.

This field resets to an architecturally UNKNOWN value.

IRGN1, bits [25:24]

Inner cacheability attribute for memory associated with translation table walks using TTBR1_EL1.

IRGN1	Meaning
0b00	Normal memory, Inner Non-cacheable.
0b01	Normal memory, Inner Write-Back Read-Allocate Write-Allocate Cacheable.
0b10	Normal memory, Inner Write-Through Read-Allocate No Write-Allocate Cacheable.
0b11	Normal memory, Inner Write-Back Read-Allocate No Write-Allocate Cacheable.

This field resets to an architecturally UNKNOWN value.

EPD1, bit [23]

Translation table walk disable for translations using TTBR1_EL1. This bit controls whether a translation table walk is performed on a TLB miss, for an address that is translated using TTBR1_EL1. The encoding of this bit is:

EPD1	Meaning
0b0	Perform translation table walks using TTBR1_EL1.
0b1	A TLB miss on an address that is translated using TTBR1_EL1 generates a Translation fault. No translation table walk is performed.

This field resets to an architecturally UNKNOWN value.

A1, bit [22]

Selects whether TTBR0_EL1 or TTBR1_EL1 defines the ASID. The encoding of this bit is:

A1	Meaning
0b0	TTBR0_EL1.ASID defines the ASID.
0b1	TTBR1_EL1.ASID defines the ASID.

This field resets to an architecturally UNKNOWN value.

T1SZ, bits [21:16]

The size offset of the memory region addressed by TTBR1_EL1. The region size is $2^{(64-T1SZ)}$ bytes.

The maximum and minimum possible values for T1SZ depend on the level of translation table and the memory translation granule size, as described in the AArch64 Virtual Memory System Architecture chapter.

This field resets to an architecturally UNKNOWN value.

TG0, bits [15:14]

Granule size for the TTBR0_EL1.

TG0	Meaning
0b00	4KB
0b01	64KB
0b10	16KB

Other values are reserved.

If the value is programmed to either a reserved value, or a size that has not been implemented, then the hardware will treat the field as if it has been programmed to an IMPLEMENTATION DEFINED choice of the sizes that has been implemented for all purposes other than the value read back from this register.

It is IMPLEMENTATION DEFINED whether the value read back is the value programmed or the value that corresponds to the size chosen.

This field resets to an architecturally UNKNOWN value.

SH0, bits [13:12]

Shareability attribute for memory associated with translation table walks using TTBR0_EL1.

SH0	Meaning
0b00	Non-shareable
0b10	Outer Shareable
0b11	Inner Shareable

Other values are reserved. The effect of programming this field to a Reserved value is that behavior is CONSTRAINED UNPREDICTABLE, as described in 'Reserved values in AArch64 System registers and translation table entries' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile~~, section K1.2.2~~.

This field resets to an architecturally UNKNOWN value.

ORGN0, bits [11:10]

Outer cacheability attribute for memory associated with translation table walks using TTBR0_EL1.

ORGN0	Meaning
0b00	Normal memory, Outer Non-cacheable.
0b01	Normal memory, Outer Write-Back Read-Allocate Write-Allocate Cacheable.
0b10	Normal memory, Outer Write-Through Read-Allocate No Write-Allocate Cacheable.
0b11	Normal memory, Outer Write-Back Read-Allocate No Write-Allocate Cacheable.

This field resets to an architecturally UNKNOWN value.

IRGN0, bits [9:8]

Inner cacheability attribute for memory associated with translation table walks using TTBR0_EL1.

IRGN0	Meaning
0b00	Normal memory, Inner Non-cacheable.
0b01	Normal memory, Inner Write-Back Read-Allocate Write-Allocate Cacheable.
0b10	Normal memory, Inner Write-Through Read-Allocate No Write-Allocate Cacheable.
0b11	Normal memory, Inner Write-Back Read-Allocate No Write-Allocate Cacheable.

This field resets to an architecturally UNKNOWN value.

EPD0, bit [7]

Translation table walk disable for translations using TTBR0_EL1. This bit controls whether a translation table walk is performed on a TLB miss, for an address that is translated using TTBR0_EL1. The encoding of this bit is:

EPD0	Meaning
0b0	Perform translation table walks using TTBR0_EL1.
0b1	A TLB miss on an address that is translated using TTBR0_EL1 generates a Translation fault. No translation table walk is performed.

This field resets to an architecturally UNKNOWN value.

Bit [6]

Reserved, RES0.

T0SZ, bits [5:0]

The size offset of the memory region addressed by TTBR0_EL1. The region size is $2^{(64-T0SZ)}$ bytes.

The maximum and minimum possible values for T0SZ depend on the level of translation table and the memory translation granule size, as described in the AArch64 Virtual Memory System Architecture chapter.

This field resets to an architecturally UNKNOWN value.

Accessing the TCR_EL1

Any of the bits in TCR_EL1, other than the A1 bit and the EPDx bits when they have the value 1, are permitted to be cached in a TLB.

When HCR_EL2.E2H is 1, without explicit synchronization, access from EL3 using the mnemonic TCR_EL1 or TCR_EL12 are not guaranteed to be ordered with respect to accesses using the other mnemonic.

Accesses to this register use the following encodings:

```
MRS <Xt>, TCR_EL1
```

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b11	0b000	0b0010	0b0000	0b010

0b11	0b0010	0b000	0b010	0b0000

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TRVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<NV2,NV1,NV> == '111' then
        return NVMem[0x120];
    else
        return TCR_EL1;
elsif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' then
        return TCR_EL2;
    else
        return TCR_EL1;
elsif PSTATE.EL == EL3 then
    return TCR_EL1;
```

MSR TCR_EL1, <Xt>

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b11	0b000	0b0010	0b0000	0b010
~~0b11~~	~~0b0010~~	~~0b000~~	~~0b010~~	~~0b0000~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<NV2,NV1,NV> == '111' then
        NVMem[0x120] = X[t];
    else
        TCR_EL1 = X[t];
elsif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' then
        TCR_EL2 = X[t];
    else
        TCR_EL1 = X[t];
elsif PSTATE.EL == EL3 then
    TCR_EL1 = X[t];
```

MRS <Xt>, TCR_EL12

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b11	0b101	0b0010	0b0000	0b010
~~0b11~~	~~0b0010~~	~~0b101~~	~~0b010~~	~~0b0000~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '101' then
        return NVMem[0x120];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if EL2Enabled() && HCR_EL2.E2H == '1' then
    if PSTATE.EL == EL2 then
        return TCR_EL1;
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if EL2Enabled() && HCR_EL2.E2H == '1' then
        return TCR_EL1;
else
    else
        UNDEFINED;
```

```
MSR TCR_EL12, <Xt>
```

op0	op1	CRn	CRm	op2
op0	CRn	op1	op2	CRm
0b11	0b101	0b0010	0b0000	0b010
0b11	0b0010	0b101	0b010	0b0000

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '101' then
        NVMem[0x120] = X[t];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if EL2Enabled() && HCR_EL2.E2H == '1' then
    if PSTATE.EL == EL2 then
        TCR_EL1 = X[t];
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if EL2Enabled() && HCR_EL2.E2H == '1' then
        TCR_EL1 = X[t];
else
    else
        UNDEFINED;
```

27 13/03 12/2019 2018 21 16:59 42; e5e4db499bf9867a4b93324c4dbac985d3da93766379d01e197f1d40720d32d0f84e419e9187e009

(old) htmldiff from- (new)

TCR_EL2, Translation Control Register (EL2)

The TCR_EL2 characteristics are:

Purpose

The control register for stage 1 of the EL2, or EL2&0, translation regime:

- When the Effective value of HCR_EL2.E2H is 0, this register controls stage 1 of the EL2 translation regime, that supports a single VA range, translated using TTBR0_EL2.
- When the value of HCR_EL2.E2H is 1, this register controls stage 1 of the EL2&0 translation regime, that supports both:
  - A lower VA range, translated using TTBR0_EL2.
  - A higher VA range, translated using TTBR1_EL2.

Configuration

AArch64 System register TCR_EL2 bits [31:0] are architecturally mapped to AArch32 System register HTCR[31:0] .

If EL2 is not implemented, this register is RES0 from EL3.

This register has no effect if EL2 is not enabled in the current Security state.

RW fields in this register reset to architecturally UNKNOWN values.

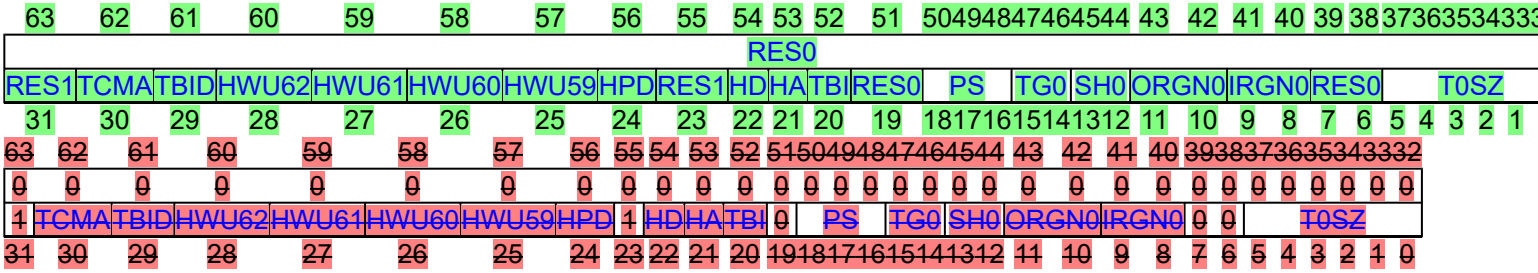
Attributes

TCR_EL2 is a 64-bit register.

Field descriptions

The TCR_EL2 bit assignments are:

When HCR_EL2.E2H == 0:



Any of the bits in TCR_EL2, other than the A1 bit and the EPDx bits when they have the value 1, are permitted to be cached in a TLB.

Bits [63:32]

Reserved, RES0.

Bit [31]

Reserved, RES1.

TCMA, bit [30]

When ARMv8.5-MemTag is implemented:

Controls the generation of Unchecked accesses at EL2 when address [59:56] = 0b0000.

TCMA	Meaning
0b0	This control has no effect on the generation of Unchecked accesses.
0b1	All accesses are Unchecked.

This field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TBID, bit [29]

When ARMv8.3-PAuth is implemented:

Controls the use of the top byte of instruction addresses for address matching.

TBID	Meaning
0b0	TCR_EL2.TBI applies to Instruction and Data accesses.
0b1	TCR_EL2.TBI applies to Data accesses only.

This affects addresses where the address would be translated by tables pointed to by TTBR0_EL2.

This field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HWU62, bit [28]

When ARMv8.2-TTPBHA is implemented:

Hardware Use. Indicates IMPLEMENTATION DEFINED hardware use of bit[62] of the stage 1 translation table Block or Page entry.

HWU62	Meaning
0b0	Bit[62] of each stage 1 translation table Block or Page entry cannot be used by hardware for an IMPLEMENTATION DEFINED purpose.
0b1	Bit[62] of each stage 1 translation table Block or Page entry can be used by hardware for an IMPLEMENTATION DEFINED purpose if the value of TCR_EL2.HPD is 1.

The Effective value of this field is 0 if the value of TCR_EL2.HPD is 0.

This field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HWU61, bit [27]

When ARMv8.2-TTPBHA is implemented:

Hardware Use. Indicates IMPLEMENTATION DEFINED hardware use of bit[61] of the stage 1 translation table Block or Page entry.

HWU61	Meaning
0b0	Bit[61] of each stage 1 translation table Block or Page entry cannot be used by hardware for an IMPLEMENTATION DEFINED purpose.
0b1	Bit[61] of each stage 1 translation table Block or Page entry can be used by hardware for an IMPLEMENTATION DEFINED purpose if the value of TCR_EL2.HPD is 1.

The Effective value of this field is 0 if the value of TCR_EL2.HPD is 0.

This field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HWU60, bit [26]

When ARMv8.2-TTPBHA is implemented:

Hardware Use. Indicates IMPLEMENTATION DEFINED hardware use of bit[60] of the stage 1 translation table Block or Page entry.

HWU60	Meaning
0b0	Bit[60] of each stage 1 translation table Block or Page entry cannot be used by hardware for an IMPLEMENTATION DEFINED purpose.
0b1	Bit[60] of each stage 1 translation table Block or Page entry can be used by hardware for an IMPLEMENTATION DEFINED purpose if the value of TCR_EL2.HPD is 1.

The Effective value of this field is 0 if the value of TCR_EL2.HPD is 0.

This field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HWU59, bit [25]

When ARMv8.2-TTPBHA is implemented:

Hardware Use. Indicates IMPLEMENTATION DEFINED hardware use of bit[59] of the stage 1 translation table Block or Page entry.

HWU59	Meaning
0b0	Bit[59] of each stage 1 translation table Block or Page entry cannot be used by hardware for an IMPLEMENTATION DEFINED purpose.
0b1	Bit[59] of each stage 1 translation table Block or Page entry can be used by hardware for an IMPLEMENTATION DEFINED purpose if the value of TCR_EL2.HPD is 1.

The Effective value of this field is 0 if the value of TCR_EL2.HPD is 0.

This field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HPD, bit [24]

When ARMv8.1-HPD is implemented:

Hierarchical Permission Disables. This affects the hierarchical control bits, APTable, PXNTable, and UXNTable, except NSTable, in the translation tables pointed to by TTBR0_EL2.

HPD	Meaning
0b0	Hierarchical permissions are enabled.
0b1	Hierarchical permissions are disabled. **Note** In this case bit[61] (APTable[0]) and bit[59] (PXNTable) of the next level descriptor attributes are required to be ignored by the PE, and are no longer reserved, allowing them to be used by software.

When disabled, the permissions are treated as if the bits are zero.

This field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bit [23]

Reserved, RES1.

HD, bit [22]

When ARMv8.1-TTHM is implemented:

Hardware management of dirty state in stage 1 translations from EL2.

HD	Meaning
0b0	Stage 1 hardware management of dirty state disabled.
0b1	Stage 1 hardware management of dirty state enabled, only if the HA bit is also set to 1.

This field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HA, bit [21]

When ARMv8.1-TTHM is implemented:

Hardware Access flag update in stage 1 translations from EL2.

HA	Meaning
0b0	Stage 1 Access flag update disabled.
0b1	Stage 1 Access flag update enabled.

This field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TBI, bit [20]

Top Byte ignored - indicates whether the top byte of an address is used for address match for the TTBR0_EL2 region, or ignored and used for tagged addresses.

TBI	Meaning
0b0	Top Byte used in the address calculation.
0b1	Top Byte ignored in the address calculation.

This affects addresses generated in EL2 using AArch64 where the address would be translated by tables pointed to by TTBR0_EL2. It has an effect whether the EL2, or EL2&0, translation regime is enabled or not.

If ARMv8.3-PAuth is implemented and TCR_EL2.TBID is 1, then this field only applies to Data accesses.

If the value of TBI is 1, then bits[63:56] of that target address are also set to 0 before the address is stored in the PC, in the following cases:

- A branch or procedure return within EL2.
- An exception taken to EL2.
- An exception return to EL2.

This field resets to an architecturally UNKNOWN value.

Bit [19]

Reserved, RES0.

PS, bits [18:16]

Physical Address Size.

PS	Meaning
0b000	32 bits, 4GB.
0b001	36 bits, 64GB.
0b010	40 bits, 1TB.
0b011	42 bits, 4TB.
0b100	44 bits, 16TB.
0b101	48 bits, 256TB.
0b110	52 bits, 4PB.

Other values are reserved.

The reserved values behave in the same way as the 0b101 or 0b110 encoding, but software must not rely on this property as the behavior of the reserved values might change in a future revision of the architecture.

The value 0b110 is permitted only if ARMv8.2-LPA is implemented and the translation granule size is 64KB.

In an implementation that supports 52-bit PAs, if the value of this field is not 0b110 or a value treated as 0b110, then bits[51:48] of every translation table base address for the stage of translation controlled by TCR_EL2 are 0b0000.

This field resets to an architecturally UNKNOWN value.

TG0, bits [15:14]

Granule size for the TTBR0_EL2.

TG0	Meaning
0b00	4KB.
0b01	64KB.
0b10	16KB.

Other values are reserved.

If the value is programmed to either a reserved value, or a size that has not been implemented, then the hardware will treat the field as if it has been programmed to an IMPLEMENTATION DEFINED choice of the sizes that has been implemented for all purposes other than the value read back from this register.

It is IMPLEMENTATION DEFINED whether the value read back is the value programmed or the value that corresponds to the size chosen.

This field resets to an architecturally UNKNOWN value.

SH0, bits [13:12]

Shareability attribute for memory associated with translation table walks using TTBR0_EL2.

SH0	Meaning
0b00	Non-shareable.
0b10	Outer Shareable.
0b11	Inner Shareable.

Other values are reserved. The effect of programming this field to a Reserved value is that behavior is CONSTRAINED UNPREDICTABLE, as described in 'Reserved values in AArch64 System registers and translation table entries' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile~~, section K1.2.2~~.

This field resets to an architecturally UNKNOWN value.

ORGN0, bits [11:10]

Outer cacheability attribute for memory associated with translation table walks using TTBR0_EL2.

ORGN0	Meaning
0b00	Normal memory, Outer Non-cacheable.
0b01	Normal memory, Outer Write-Back Read-Allocate Write-Allocate Cacheable.
0b10	Normal memory, Outer Write-Through Read-Allocate No Write-Allocate Cacheable.
0b11	Normal memory, Outer Write-Back Read-Allocate No Write-Allocate Cacheable.

This field resets to an architecturally UNKNOWN value.

IRGN0, bits [9:8]

Inner cacheability attribute for memory associated with translation table walks using TTBR0_EL2.

IRGN0	Meaning
0b00	Normal memory, Inner Non-cacheable.
0b01	Normal memory, Inner Write-Back Read-Allocate Write-Allocate Cacheable.
0b10	Normal memory, Inner Write-Through Read-Allocate No Write-Allocate Cacheable.
0b11	Normal memory, Inner Write-Back Read-Allocate No Write-Allocate Cacheable.

This field resets to an architecturally UNKNOWN value.

Bits [7:6]

Reserved, RES0.

T0SZ, bits [5:0]

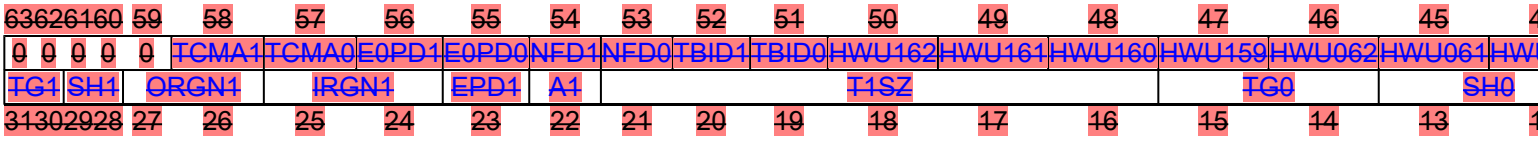
The size offset of the memory region addressed by TTBR0_EL2. The region size is $2^{(64-T0SZ)}$ bytes.

The maximum and minimum possible values for T0SZ depend on the level of translation table and the memory translation granule size, as described in the AArch64 Virtual Memory System Architecture chapter.

This field resets to an architecturally UNKNOWN value.

When HCR_EL2.E2H == 1:

63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	4
RES0				TCMA1	TCMA0	E0PD1	E0PD0	NFD1	NFD0	TBID1	TBID0	HWU162	HWU161	HWU160	HWU159	HWU062	HWU061	HW	
TG1		SH1		ORGN1		IRGN1		EPD1	A1	T1SZ						TG0		SH0	
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	1



This view of the register is only valid from Armv8.1 when HCR_EL2.E2H is 1.

Any of the bits in TCR_EL2 are permitted to be cached in a TLB.

Bits [63:59]

Reserved, RES0.

TCMA1, bit [58]

When ARMv8.5-MemTag is implemented:

Controls the generation of Unchecked accesses at EL2, and at EL0 if HCR_EL2.TGE=1, when address[59:55] = `0b11111`.

TCMA1	Meaning
`0b0`	This control has no effect on the generation of Unchecked accesses at EL2 or EL0.
`0b1`	All accesses are Unchecked.

Note

Software may change this control bit on a context switch.

This field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TCMA0, bit [57]

When ARMv8.5-MemTag is implemented:

Controls the generation of Unchecked accesses at EL2, and at EL0 if HCR_EL2.TGE=1, when address[59:55] = `0b00000`.

TCMA0	Meaning
`0b0`	This control has no effect on the generation of Unchecked accesses at EL2 or EL0.
`0b1`	All accesses are Unchecked.

Note

Software may change this control bit on a context switch.

This field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

E0PD1, bit [56]

When ARMv8.5-E0PD is implemented:

Faulting control for EL0 access to any address translated by TTBR1_EL2.

E0PD1	Meaning
0b0	EL0 access to any address translated by TTBR1_EL2 will not generate a fault by this mechanism.
0b1	EL0 access to any address translated by TTBR1_EL2 will generate a level 0 translation fault

This field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

E0PD0, bit [55]

When ARMv8.5-E0PD is implemented:

Faulting control for EL0 access to any address translated by TTBR0_EL2.

E0PD0	Meaning
0b0	EL0 access to any address translated by TTBR0_EL2 will not generate a fault by this mechanism.
0b1	EL0 access to any address translated by TTBR0_EL2 will generate a level 0 translation fault

This field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

NFD1, bit [54]

When SVE is implemented:

Non-fault translation table walk disable for stage 1 translations using TTBR1_EL2.

This bit controls whether to perform a stage 1 translation table walk in response to a non-fault access from EL0 for a virtual address that is translated using TTBR1_EL2.

If SVE is implemented, the affected access types include:

- All accesses due to an SVE non-fault contiguous load instruction.
- Accesses due to an SVE first-fault gather load instruction that are not for the First active element. Accesses due to an SVE first-fault contiguous load instruction are not affected.
- Accesses due to prefetch instructions might be affected, but the effect is not architecturally visible.

See 'The Scalable Vector Extension (SVE)', in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile~~, chapter A1~~ for more information.

Defined values are:

NFD1	Meaning
0b0	Does not disable stage 1 translation table walks using TTBR1_EL2.
0b1	A TLB miss on a virtual address that is translated using TTBR1_EL2 due to the specified access types causes the access to fail without taking an exception. No stage 1 translation table walk is performed.

This field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

NFD0, bit [53]

When SVE is implemented:

Non-fault translation table walk disable for stage 1 translations using TTBR0_EL2.

This bit controls whether to perform a stage 1 translation table walk in response to a non-fault access from EL0 for a virtual address that is translated using TTBR0_EL2.

If SVE is implemented, the affected access types include:

- All accesses due to an SVE non-fault contiguous load instruction.
- Accesses due to an SVE first-fault gather load instruction that are not for the First active element. Accesses due to an SVE first-fault contiguous load instruction are not affected.
- Accesses due to prefetch instructions might be affected, but the effect is not architecturally visible.

See 'The Scalable Vector Extension (SVE)', in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile~~, chapter A1~~ for more information.

Defined values are:

NFD0	Meaning
0b0	Does not disable stage 1 translation table walks using TTBR0_EL2.
0b1	A TLB miss on a virtual address that is translated using TTBR0_EL2 due to the specified access types causes the access to fail without taking an exception. No stage 1 translation table walk is performed.

This field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TBID1, bit [52]

When ARMv8.3-PAuth is implemented:

Controls the use of the top byte of instruction addresses for address matching.

TBID1	Meaning
0b0	TCR_EL2.TBI1 applies to Instruction and Data accesses.
0b1	TCR_EL2.TBI1 applies to Data accesses only.

This affects addresses where the address would be translated by tables pointed to by TTBR1_EL2.

This field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TBID0, bit [51]

When ARMv8.3-PAuth is implemented:

Controls the use of the top byte of instruction addresses for address matching.

TBID0	Meaning
0b0	TCR_EL2.TBI0 applies to Instruction and Data accesses.
0b1	TCR_EL2.TBI0 applies to Data accesses only.

This affects addresses where the address would be translated by tables pointed to by TTBR0_EL2.

This field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HWU162, bit [50]

When ARMv8.2-TTPBHA is implemented:

Hardware Use. Indicates IMPLEMENTATION DEFINED hardware use of bit[62] of the stage 1 translation table Block or Page entry for translations using TTBR1_EL2.

HWU162	Meaning
0b0	For translations using TTBR1_EL2, bit[62] of each stage 1 translation table Block or Page entry cannot be used by hardware for an IMPLEMENTATION DEFINED purpose.
0b1	For translations using TTBR1_EL2, bit[62] of each stage 1 translation table Block or Page entry can be used by hardware for an IMPLEMENTATION DEFINED purpose if the value of TCR_EL2.HPD1 is 1.

The Effective value of this field is 0 if the value of TCR_EL2.HPD1 is 0.

This field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HWU161, bit [49]

When ARMv8.2-TTPBHA is implemented:

Hardware Use. Indicates IMPLEMENTATION DEFINED hardware use of bit[61] of the stage 1 translation table Block or Page entry for translations using TTBR1_EL2.

HWU161	Meaning
0b0	For translations using TTBR1_EL2, bit[61] of each stage 1 translation table Block or Page entry cannot be used by hardware for an IMPLEMENTATION DEFINED purpose.
0b1	For translations using TTBR1_EL2, bit[61] of each stage 1 translation table Block or Page entry can be used by hardware for an IMPLEMENTATION DEFINED purpose if the value of TCR_EL2.HPD1 is 1.

The Effective value of this field is 0 if the value of TCR_EL2.HPD1 is 0.

This field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HWU160, bit [48]

When ARMv8.2-TTPBHA is implemented:

Hardware Use. Indicates IMPLEMENTATION DEFINED hardware use of bit[60] of the stage 1 translation table Block or Page entry for translations using TTBR1_EL2.

HWU160	Meaning
0b0	For translations using TTBR1_EL2, bit[60] of each stage 1 translation table Block or Page entry cannot be used by hardware for an IMPLEMENTATION DEFINED purpose.
0b1	For translations using TTBR1_EL2, bit[60] of each stage 1 translation table Block or Page entry can be used by hardware for an IMPLEMENTATION DEFINED purpose if the value of TCR_EL2.HPD1 is 1.

The Effective value of this field is 0 if the value of TCR_EL2.HPD1 is 0.

This field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HWU159, bit [47]

When ARMv8.2-TTPBHA is implemented:

Hardware Use. Indicates IMPLEMENTATION DEFINED hardware use of bit[59] of the stage 1 translation table Block or Page entry for translations using TTBR1_EL2.

HWU159	Meaning
0b0	For translations using TTBR1_EL2, bit[59] of each stage 1 translation table Block or Page entry cannot be used by hardware for an IMPLEMENTATION DEFINED purpose.
0b1	For translations using TTBR1_EL2, bit[59] of each stage 1 translation table Block or Page entry can be used by hardware for an IMPLEMENTATION DEFINED purpose if the value of TCR_EL2.HPD1 is 1.

The Effective value of this field is 0 if the value of TCR_EL2.HPD1 is 0.

This field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HWU062, bit [46]

When ARMv8.2-TTPBHA is implemented:

Hardware Use. Indicates IMPLEMENTATION DEFINED hardware use of bit[62] of the stage 1 translation table Block or Page entry for translations using TTBR0_EL1.

HWU062	Meaning
0b0	For translations using TTBR0_EL1, bit[62] of each stage 1 translation table Block or Page entry cannot be used by hardware for an IMPLEMENTATION DEFINED purpose.
0b1	For translations using TTBR0_EL1, bit[62] of each stage 1 translation table Block or Page entry can be used by hardware for an IMPLEMENTATION DEFINED purpose if the value of TCR_EL2.HPD0 is 1.

The Effective value of this field is 0 if the value of TCR_EL2.HPD0 is 0.

This field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HWU061, bit [45]

When ARMv8.2-TTPBHA is implemented:

Hardware Use. Indicates IMPLEMENTATION DEFINED hardware use of bit[61] of the stage 1 translation table Block or Page entry for translations using TTBR0_EL1.

HWU061	Meaning
0b0	For translations using TTBR0_EL1, bit[61] of each stage 1 translation table Block or Page entry cannot be used by hardware for an IMPLEMENTATION DEFINED purpose.
0b1	For translations using TTBR0_EL1, bit[61] of each stage 1 translation table Block or Page entry can be used by hardware for an IMPLEMENTATION DEFINED purpose if the value of TCR_EL2.HPD0 is 1.

The Effective value of this field is 0 if the value of TCR_EL2.HPD0 is 0.

This field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HWU060, bit [44]

When ARMv8.2-TTPBHA is implemented:

Hardware Use. Indicates IMPLEMENTATION DEFINED hardware use of bit[60] of the stage 1 translation table Block or Page entry for translations using TTBR0_EL1.

HWU060	Meaning
0b0	For translations using TTBR0_EL1, bit[60] of each stage 1 translation table Block or Page entry cannot be used by hardware for an IMPLEMENTATION DEFINED purpose.
0b1	For translations using TTBR0_EL1, bit[60] of each stage 1 translation table Block or Page entry can be used by hardware for an IMPLEMENTATION DEFINED purpose if the value of TCR_EL2.HPD0 is 1.

The Effective value of this field is 0 if the value of TCR_EL2.HPD0 is 0.

This field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HWU059, bit [43]

When ARMv8.2-TTPBHA is implemented:

Hardware Use. Indicates IMPLEMENTATION DEFINED hardware use of bit[59] of the stage 1 translation table Block or Page entry for translations using TTBR0_EL1.

HWU059	Meaning
0b0	For translations using TTBR0_EL1, bit[59] of each stage 1 translation table Block or Page entry cannot be used by hardware for an IMPLEMENTATION DEFINED purpose.
0b1	For translations using TTBR0_EL1, bit[59] of each stage 1 translation table Block or Page entry can be used by hardware for an IMPLEMENTATION DEFINED purpose if the value of TCR_EL2.HPD0 is 1.

The Effective value of this field is 0 if the value of TCR_EL2.HPD0 is 0.

This field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HPD1, bit [42]

When ARMv8.1-HPD is implemented:

Hierarchical Permission Disables. This affects the hierarchical control bits, APTable, PXNTable, and UXNTable, except NSTable, in the translation tables pointed to by TTBR1_EL2.

HPD1	Meaning
0b0	Hierarchical permissions are enabled.
0b1	Hierarchical permissions are disabled.

When disabled, the permissions are treated as if the bits are zero.

This field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HPD0, bit [41]

When ARMv8.1-HPD is implemented:

Hierarchical Permission Disables. This affects the hierarchical control bits, APTable, PXNTable, and UXNTable, except NSTable, in the translation tables pointed to by TTBR0_EL2.

HPD0	Meaning
0b0	Hierarchical permissions are enabled.
0b1	Hierarchical permissions are disabled.

When disabled, the permissions are treated as if the bits are zero.

This field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HD, bit [40]

When ARMv8.1-TTHM is implemented:

Hardware management of dirty state in stage 1 translations from EL2.

HD	Meaning
0b0	Stage 1 hardware management of dirty state disabled.
0b1	Stage 1 hardware management of dirty state enabled, only if the HA bit is also set to 1.

This field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HA, bit [39]

When ARMv8.1-TTHM is implemented:

Hardware Access flag update in stage 1 translations from EL2.

HA	Meaning
0b0	Stage 1 Access flag update disabled.
0b1	Stage 1 Access flag update enabled.

This field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TBI1, bit [38]

From Armv8.1:

Top Byte ignored - indicates whether the top byte of an address is used for address match for the TTBR1_EL2 region, or ignored and used for tagged addresses. Defined values are:

TBI1	Meaning
0b0	Top Byte used in the address calculation.
0b1	Top Byte ignored in the address calculation.

This affects addresses generated in EL0 and EL2 using AArch64 where the address would be translated by tables pointed to by TTBR1_EL2. It has an effect whether the EL2, or EL2&0, translation regime is enabled or not.

If ARMv8.3-PAuth is implemented and TCR_EL2.TBID1 is 1, then this field only applies to Data accesses.

If the value of TBI1 is 1 and bit [55] of the target address to be stored to the PC is 1, then bits[63:56] of that target address are also set to 1 before the address is stored in the PC, in the following cases:

- A branch or procedure return within EL0 or EL1.
- An exception taken to EL1.
- An exception return to EL0 or EL1.

This field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TBI0, bit [37]

When ARMv8.1-VHE is implemented:

Top Byte Ignored. Indicates whether the top byte of an address is used for address match for the TTBR0_EL2 region, or ignored and used for tagged addresses. Defined values are:

TBI0	Meaning
0b0	Top Byte used in the address calculation.
0b1	Top Byte ignored in the address calculation.

This affects addresses generated in EL0 and EL2 using AArch64 where the address would be translated by tables pointed to by TTBR0_EL2. It has an effect whether the EL2, or EL2&0, translation regime is enabled or not.

If ARMv8.3-PAuth is implemented and TCR_EL2.TBID0 is 1, then this field only applies to Data accesses.

If the value of TBI0 is 1 and bit [55] of the target address to be stored to the PC is 0, then bits[63:56] of that target address are also set to 0 before the address is stored in the PC, in the following cases:

- A branch or procedure return within EL0 or EL1.
- An exception taken to EL1.
- An exception return to EL0 or EL1.

This field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

AS, bit [36]

ASID Size. Defined values are:

AS	Meaning
0b0	8 bit - the upper 8 bits of TTBR0_EL2 and TTBR1_EL2 are ignored by hardware for every purpose except reading back the register, and are treated as if they are all zeros for when used for allocation and matching entries in the TLB.
0b1	16 bit - the upper 16 bits of TTBR0_EL2 and TTBR1_EL2 are used for allocation and matching in the TLB.

If the implementation has only 8 bits of ASID, this field is RES0.

This field resets to an architecturally UNKNOWN value.

Bit [35]

Reserved, RES0.

IPS, bits [34:32]

Intermediate Physical Address Size.

IPS	Meaning	Applies when
0b000	32 bits, 4GB.	
0b001	36 bits, 64GB.	
0b010	40 bits, 1TB.	
0b011	42 bits, 4TB.	
0b100	44 bits, 16TB.	
0b101	48 bits, 256TB.	
0b110	52 bits, 4PB.	When ARMv8.2-LPA is implemented

Other values are reserved.

The reserved values behave in the same way as the 0b101 or 0b110 encoding, but software must not rely on this property as the behavior of the reserved values might change in a future revision of the architecture.

The value 0b110 is permitted only if ARMv8.2-LPA is implemented and the translation granule size is 64KB.

In an implementation that supports 52-bit PAs, if the value of this field is not 0b110 or a value treated as 0b110, then bits[51:48] of every translation table base address for the stage of translation controlled by TCR_EL2 are 0b0000.

This field resets to an architecturally UNKNOWN value.

TG1, bits [31:30]

From Armv8.1:

Granule size for the TTBR1_EL2.

TG1	Meaning
0b01	16KB.
0b10	4KB.
0b11	64KB.

Other values are reserved.

If the value is programmed to either a reserved value, or a size that has not been implemented, then the hardware will treat the field as if it has been programmed to an IMPLEMENTATION DEFINED choice of the sizes that has been implemented for all purposes other than the value read back from this register.

It is IMPLEMENTATION DEFINED whether the value read back is the value programmed or the value that corresponds to the size chosen.

This field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

SH1, bits [29:28]

From Armv8.1:

Shareability attribute for memory associated with translation table walks using TTBR1_EL2. Defined values are:

SH1	Meaning
0b00	Non-shareable.
0b10	Outer Shareable.
0b11	Inner Shareable.

Other values are reserved. The effect of programming this field to a Reserved value is that behavior is CONSTRAINED UNPREDICTABLE, as described in 'Reserved values in AArch64 System registers and translation table entries' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile~~, section K1.2.2~~.

This field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

ORGN1, bits [27:26]

From Armv8.1:

Outer cacheability attribute for memory associated with translation table walks using TTBR1_EL2.

ORGN1	Meaning
0b00	Normal memory, Outer Non-cacheable.
0b01	Normal memory, Outer Write-Back Read-Allocate Write-Allocate Cacheable.
0b10	Normal memory, Outer Write-Through Read-Allocate No Write-Allocate Cacheable.
0b11	Normal memory, Outer Write-Back Read-Allocate No Write-Allocate Cacheable.

This field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

IRGN1, bits [25:24]

From Armv8.1:

Inner cacheability attribute for memory associated with translation table walks using TTBR1_EL2.

IRGN1	Meaning
0b00	Normal memory, Inner Non-cacheable.
0b01	Normal memory, Inner Write-Back Read-Allocate Write-Allocate Cacheable.
0b10	Normal memory, Inner Write-Through Read-Allocate No Write-Allocate Cacheable.
0b11	Normal memory, Inner Write-Back Read-Allocate No Write-Allocate Cacheable.

This field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EPD1, bit [23]

From Armv8.1:

Translation table walk disable for translations using TTBR1_EL2. This bit controls whether a translation table walk is performed on a TLB miss, for an address that is translated using TTBR1_EL2. The encoding of this bit is:

EPD1	Meaning
0b0	Perform translation table walks using TTBR1_EL2.
0b1	A TLB miss on an address that is translated using TTBR1_EL2 generates a Translation fault. No translation table walk is performed.

This field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

A1, bit [22]

Selects whether TTBR0_EL2 or TTBR1_EL2 defines the ASID. The encoding of this bit is:

A1	Meaning
0b0	TTBR0_EL2.ASID defines the ASID.
0b1	TTBR1_EL2.ASID defines the ASID.

This field resets to an architecturally UNKNOWN value.

T1SZ, bits [21:16]

From Armv8.1:

The size offset of the memory region addressed by TTBR1_EL2. The region size is $2^{(64-T1SZ)}$ bytes.

The maximum and minimum possible values for T1SZ depend on the level of translation table and the memory translation granule size, as described in the AArch64 Virtual Memory System Architecture chapter.

This field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TG0, bits [15:14]

From Armv8.1:

Granule size for the TTBR0_EL2.

TG0	Meaning
0b00	4KB.
0b01	64KB.
0b10	16KB.

Other values are reserved.

If the value is programmed to either a reserved value, or a size that has not been implemented, then the hardware will treat the field as if it has been programmed to an IMPLEMENTATION DEFINED choice of the sizes that has been implemented for all purposes other than the value read back from this register.

It is IMPLEMENTATION DEFINED whether the value read back is the value programmed or the value that corresponds to the size chosen.

This field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

SH0, bits [13:12]

From Armv8.1:

Shareability attribute for memory associated with translation table walks using TTBR0_EL2.

SH0	Meaning
0b00	Non-shareable.
0b10	Outer Shareable.
0b11	Inner Shareable.

Other values are reserved. The effect of programming this field to a Reserved value is that behavior is CONSTRAINED UNPREDICTABLE, as described in 'Reserved values in AArch64 System registers and translation table entries' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile~~, section K1.2.2~~.

This field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

ORGN0, bits [11:10]

From Armv8.1:

Outer cacheability attribute for memory associated with translation table walks using TTBR0_EL2.

ORGN0	Meaning
0b00	Normal memory, Outer Non-cacheable.
0b01	Normal memory, Outer Write-Back Read-Allocate Write-Allocate Cacheable.
0b10	Normal memory, Outer Write-Through Read-Allocate No Write-Allocate Cacheable.
0b11	Normal memory, Outer Write-Back Read-Allocate No Write-Allocate Cacheable.

This field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

IRGN0, bits [9:8]

From Armv8.1:

Inner cacheability attribute for memory associated with translation table walks using TTBR0_EL2.

IRGN0	Meaning
0b00	Normal memory, Inner Non-cacheable.
0b01	Normal memory, Inner Write-Back Read-Allocate Write-Allocate Cacheable.
0b10	Normal memory, Inner Write-Through Read-Allocate No Write-Allocate Cacheable.
0b11	Normal memory, Inner Write-Back Read-Allocate No Write-Allocate Cacheable.

This field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EPD0, bit [7]

From Armv8.1:

Translation table walk disable for translations using TTBR0_EL2. This bit controls whether a translation table walk is performed on a TLB miss, for an address that is translated using TTBR0_EL2. The encoding of this bit is:

EPD0	Meaning
0b0	Perform translation table walks using TTBR0_EL2.
0b1	A TLB miss on an address that is translated using TTBR0_EL2 generates a Translation fault. No translation table walk is performed.

This field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bit [6]

Reserved, RES0.

T0SZ, bits [5:0]

From Armv8.1:

The size offset of the memory region addressed by TTBR0_EL2. The region size is $2^{(64-T0SZ)}$ bytes.

The maximum and minimum possible values for T0SZ depend on the level of translation table and the memory translation granule size, as described in the AArch64 Virtual Memory System Architecture chapter.

This field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Accessing the TCR_EL2

Any of the bits in TCR_EL2, other than the A1 bit and the EPDx bits when they have the value 1, are permitted to be cached in a TLB.

When HCR_EL2.E2H is 1, without explicit synchronization, access from EL2 using the mnemonic TCR_EL2 or TCR_EL1 are not guaranteed to be ordered with respect to accesses using the other mnemonic.

Accesses to this register use the following encodings:

MRS <Xt>, TCR_EL2

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b11	0b100	0b0010	0b0000	0b010
~~0b11~~	~~0b0010~~	~~0b100~~	~~0b010~~	~~0b0000~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    return TCR_EL2;
elsif PSTATE.EL == EL3 then
    return TCR_EL2;
```

MSR TCR_EL2, <Xt>

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b11	0b100	0b0010	0b0000	0b010
~~0b11~~	~~0b0010~~	~~0b100~~	~~0b010~~	~~0b0000~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    TCR_EL2 = X[t];
elsif PSTATE.EL == EL3 then
    TCR_EL2 = X[t];
```

MRS <Xt>, TCR_EL1

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b11	0b000	0b0010	0b0000	0b010
~~0b11~~	~~0b0010~~	~~0b000~~	~~0b010~~	~~0b0000~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TRVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<NV2,NV1,NV> == '111' then
        return NVMem[0x120];
    else
        return TCR_EL1;
elsif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' then
        return TCR_EL2;
    else
        return TCR_EL1;
elsif PSTATE.EL == EL3 then
    return TCR_EL1;
```

```
MSR TCR_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b11	0b000	0b0010	0b0000	0b010
~~0b11~~	~~0b0010~~	~~0b000~~	~~0b010~~	~~0b0000~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<NV2,NV1,NV> == '111' then
        NVMem[0x120] = X[t];
    else
        TCR_EL1 = X[t];
elsif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' then
        TCR_EL2 = X[t];
    else
        TCR_EL1 = X[t];
elsif PSTATE.EL == EL3 then
    TCR_EL1 = X[t];
```

27~~13~~/03~~12~~/2019~~2018~~ 21~~16~~:59~~42~~; e5e4db499bf9867a4b93324c4dbac985d3da93766~~379d01e197f1d40720d32d0f84e419e9187e009~~

(old) htmldiff from- (new)

TCR_EL3, Translation Control Register (EL3)

The TCR_EL3 characteristics are:

Purpose

The control register for stage 1 of the EL3 translation regime.

Configuration

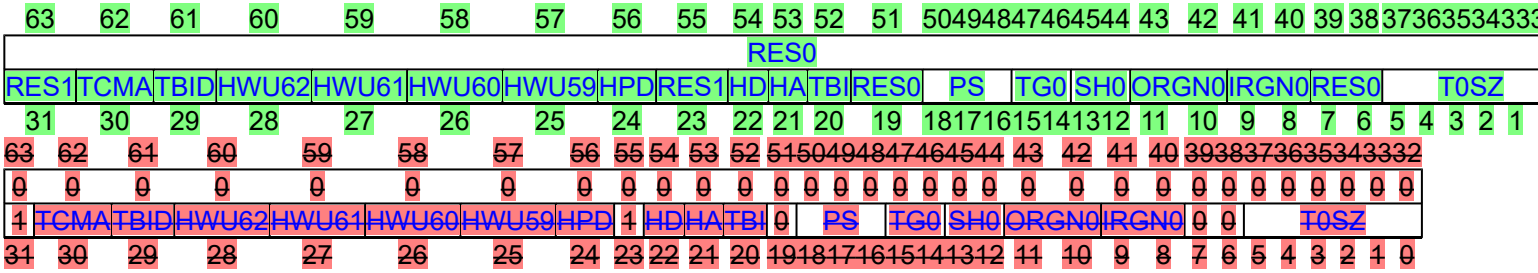
RW fields in this register reset to architecturally UNKNOWN values.

Attributes

TCR_EL3 is a 64-bit register.

Field descriptions

The TCR_EL3 bit assignments are:



Any of the bits in TCR_EL3 are permitted to be cached in a TLB.

Bits [63:32]

Reserved, RES0.

Bit [31]

Reserved, RES1.

TCMA, bit [30]

When ARMv8.5-MemTag is implemented:

Controls the generation of Unchecked accesses at EL3 when address [59:56] = `0b0000`.

TCMA	Meaning
`0b0`	This control has no effect on the generation of Unchecked accesses.
`0b1`	All accesses are Unchecked.

This field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TBID, bit [29]

When ARMv8.3-PAuth is implemented:

Controls the use of the top byte of instruction addresses for address matching.

TBID	Meaning
0b0	TCR_EL3.TBI applies to Instruction and Data accesses.
0b1	TCR_EL3.TBI applies to Data accesses only.

This affects addresses where the address would be translated by tables pointed to by TTBR0_EL3.

This field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HWU62, bit [28]

When ARMv8.2-TTPBHA is implemented:

Hardware Use. Indicates IMPLEMENTATION DEFINED hardware use of bit[62] of the stage 1 translation table Block or Page entry.

HWU62	Meaning
0b0	Bit[62] of each stage 1 translation table Block or Page entry cannot be used by hardware for an IMPLEMENTATION DEFINED purpose.
0b1	Bit[62] of each stage 1 translation table Block or Page entry can be used by hardware for an IMPLEMENTATION DEFINED purpose if the value of TCR_EL3.HPD is 1.

The Effective value of this field is 0 if the value of TCR_EL3.HPD is 0.

This field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HWU61, bit [27]

When ARMv8.2-TTPBHA is implemented:

Hardware Use. Indicates IMPLEMENTATION DEFINED hardware use of bit[61] of the stage 1 translation table Block or Page entry.

HWU61	Meaning
0b0	Bit[61] of each stage 1 translation table Block or Page entry cannot be used by hardware for an IMPLEMENTATION DEFINED purpose.
0b1	Bit[61] of each stage 1 translation table Block or Page entry can be used by hardware for an IMPLEMENTATION DEFINED purpose if the value of TCR_EL3.HPD is 1.

The Effective value of this field is 0 if the value of TCR_EL3.HPD is 0.

This field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HWU60, bit [26]

When ARMv8.2-TTPBHA is implemented:

Hardware Use. Indicates IMPLEMENTATION DEFINED hardware use of bit[60] of the stage 1 translation table Block or Page entry.

HWU60	Meaning
0b0	Bit[60] of each stage 1 translation table Block or Page entry cannot be used by hardware for an IMPLEMENTATION DEFINED purpose.
0b1	Bit[60] of each stage 1 translation table Block or Page entry can be used by hardware for an IMPLEMENTATION DEFINED purpose if the value of TCR_EL3.HPD is 1.

The Effective value of this field is 0 if the value of TCR_EL3.HPD is 0.

This field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HWU59, bit [25]

When ARMv8.2-TTPBHA is implemented:

Hardware Use. Indicates IMPLEMENTATION DEFINED hardware use of bit[59] of the stage 1 translation table Block or Page entry.

HWU59	Meaning
0b0	Bit[59] of each stage 1 translation table Block or Page entry cannot be used by hardware for an IMPLEMENTATION DEFINED purpose.
0b1	Bit[59] of each stage 1 translation table Block or Page entry can be used by hardware for an IMPLEMENTATION DEFINED purpose if the value of TCR_EL3.HPD is 1.

The Effective value of this field is 0 if the value of TCR_EL3.HPD is 0.

This field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HPD, bit [24]

When ARMv8.1-HPD is implemented:

Hierarchical Permission Disables. This affects the hierarchical control bits, APTable, PXNTable, and UXNTable, except NSTable, in the translation tables pointed to by TTBR0_EL3.

HPD	Meaning
0b0	Hierarchical permissions are enabled.
0b1	Hierarchical permissions are disabled. **Note** In this case bit[61] (APTable[0]) and bit[59] (PXNTable) of the next level descriptor attributes are required to be ignored by the PE, and are no longer reserved, allowing them to be used by software.

When disabled, the permissions are treated as if the bits are zero.

This field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bit [23]

Reserved, RES1.

HD, bit [22]

When ARMv8.1-TTHM is implemented:

Hardware management of dirty state in stage 1 translations from EL3.

HD	Meaning
0b0	Stage 1 hardware management of dirty state disabled.
0b1	Stage 1 hardware management of dirty state enabled, only if the HA bit is also set to 1.

This field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HA, bit [21]

When ARMv8.1-TTHM is implemented:

Hardware Access flag update in stage 1 translations from EL3.

HA	Meaning
0b0	Stage 1 Access flag update disabled.
0b1	Stage 1 Access flag update enabled.

This field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TBI, bit [20]

Top Byte ignored - indicates whether the top byte of an address is used for address match for the TTBR0_EL3 region, or ignored and used for tagged addresses.

TBI	Meaning
0b0	Top Byte used in the address calculation.
0b1	Top Byte ignored in the address calculation.

This affects addresses generated in EL3 using AArch64 where the address would be translated by tables pointed to by TTBR0_EL3. It has an effect whether the EL3 translation regime is enabled or not.

If ARMv8.3-PAuth is implemented and TCR_EL3.TBID is 1, then this field only applies to Data accesses.

Otherwise, if the value of TBI is 1, then bits[63:56] of that target address are also set to 0 before the address is stored in the PC, in the following cases:

- A branch or procedure return within EL3.
- A exception taken to EL3.
- An exception return to EL3.

This field resets to an architecturally UNKNOWN value.

Bit [19]

Reserved, RES0.

PS, bits [18:16]

Physical Address Size.

PS	Meaning
0b000	32 bits, 4GB.
0b001	36 bits, 64GB.
0b010	40 bits, 1TB.
0b011	42 bits, 4TB.
0b100	44 bits, 16TB.
0b101	48 bits, 256TB.
0b110	52 bits, 4PB.

Other values are reserved.

The reserved values behave in the same way as the 0b101 or 0b110 encoding, but software must not rely on this property as the behavior of the reserved values might change in a future revision of the architecture.

The value 0b110 is permitted only if ARMv8.2-LPA is implemented and the translation granule size is 64KB.

In an implementation that supports 52-bit PAs, if the value of this field is not 0b110 or a value treated as 0b110, then bits[51:48] of every translation table base address for the stage of translation controlled by TCR_EL3 are 0b0000.

This field resets to an architecturally UNKNOWN value.

TG0, bits [15:14]

Granule size for the TTBR0_EL3.

TG0	Meaning
0b00	4KB.
0b01	64KB.
0b10	16KB.

Other values are reserved.

If the value is programmed to either a reserved value, or a size that has not been implemented, then the hardware will treat the field as if it has been programmed to an IMPLEMENTATION DEFINED choice of the sizes that has been implemented for all purposes other than the value read back from this register.

It is IMPLEMENTATION DEFINED whether the value read back is the value programmed or the value that corresponds to the size chosen.

This field resets to an architecturally UNKNOWN value.

SH0, bits [13:12]

Shareability attribute for memory associated with translation table walks using TTBR0_EL3.

SH0	Meaning
0b00	Non-shareable.
0b10	Outer Shareable.
0b11	Inner Shareable.

Other values are reserved. The effect of programming this field to a Reserved value is that behavior is CONSTRAINED UNPREDICTABLE, as described in 'Reserved values in AArch64 System registers and translation table entries' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section K1.2.2.

This field resets to an architecturally UNKNOWN value.

ORGN0, bits [11:10]

Outer cacheability attribute for memory associated with translation table walks using TTBR0_EL3.

ORGN0	Meaning
0b00	Normal memory, Outer Non-cacheable.
0b01	Normal memory, Outer Write-Back Read-Allocate Write-Allocate Cacheable.
0b10	Normal memory, Outer Write-Through Read-Allocate No Write-Allocate Cacheable.
0b11	Normal memory, Outer Write-Back Read-Allocate No Write-Allocate Cacheable.

This field resets to an architecturally UNKNOWN value.

IRGN0, bits [9:8]

Inner cacheability attribute for memory associated with translation table walks using TTBR0_EL3.

IRGN0	Meaning
0b00	Normal memory, Inner Non-cacheable.
0b01	Normal memory, Inner Write-Back Read-Allocate Write-Allocate Cacheable.
0b10	Normal memory, Inner Write-Through Read-Allocate No Write-Allocate Cacheable.
0b11	Normal memory, Inner Write-Back Read-Allocate No Write-Allocate Cacheable.

This field resets to an architecturally UNKNOWN value.

Bits [7:6]

Reserved, RES0.

T0SZ, bits [5:0]

The size offset of the memory region addressed by TTBR0_EL3. The region size is $2^{(64-T0SZ)}$ bytes.

The maximum and minimum possible values for T0SZ depend on the level of translation table and the memory translation granule size, as described in the AArch64 Virtual Memory System Architecture chapter.

This field resets to an architecturally UNKNOWN value.

Accessing the TCR_EL3

Any of the bits in TCR_EL3 are permitted to be cached in a TLB.

Accesses to this register use the following encodings:

```
MRS <Xt>, TCR_EL3
```

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b11	0b110	0b0010	0b0000	0b010
~~0b11~~	~~0b0010~~	~~0b110~~	~~0b010~~	~~0b0000~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    return TCR_EL3;
```

```
MSR TCR_EL3, <Xt>
```

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b11	0b110	0b0010	0b0000	0b010
~~0b11~~	~~0b0010~~	~~0b110~~	~~0b010~~	~~0b0000~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    TCR_EL3 = X[t];
```

27~~13~~/03~~12~~/2019~~2018~~ 21~~16~~:59~~42~~; e5e4db499bf9867a4b93324c4dbac985d3da93766~~379d01e197f1d40720d32d0f84e419e9187e009~~

(old) htmldiff from- (new)

(old) htmldiff from- (new)

TFSR_EL1, Tag Fail Status Register (EL1).

The TFSR_EL1 characteristics are:

Purpose

Holds accumulated Tag Check Fails occurring in EL1 which are not taken precisely.

Configuration

This register is present only when ARMv8.5-MemTag is implemented. Otherwise, direct accesses to TFSR_EL1 are UNDEFINED.

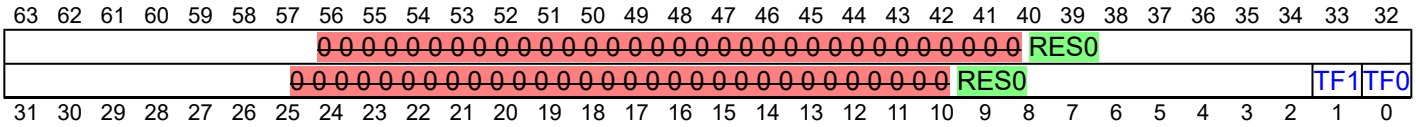
RW fields in this register reset to architecturally UNKNOWN values.

Attributes

TFSR_EL1 is a 64-bit register.

Field descriptions

The TFSR_EL1 bit assignments are:



Bits [63:2]

Reserved, RES0.

TF1, bit [1]

Tag Check Fail. Asynchronously set to 1 when a Tag Check fail using a virtual address with bit<55>==`0b1` occurs.

This field resets to an architecturally UNKNOWN value.

TF0, bit [0]

Tag Check Fail. Asynchronously set to 1 when a Tag Check fail using a virtual address with bit<55>==`0b0` occurs.

This field resets to an architecturally UNKNOWN value.

Accessing the TFSR_EL1

Accesses to this register use the following encodings:

```
MRS <Xt>, TFSR_EL1
```

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b11	0b000	0b0110	0b0101	0b000

0b11	0b0110	0b000	0b000	0b0101

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV1> == '01' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.ATA == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.ATA == '0' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<NV2,NV1,NV> == '111' then
        return NVMem[0x190];
    else
        return TFSR_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.ATA == '0' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HCR_EL2.E2H == '1' then
        return TFSR_EL2;
    else
        return TFSR_EL1;
elsif PSTATE.EL == EL3 then
    return TFSR_EL1;
```

```
MSR TFSR_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b11	0b000	0b0110	0b0101	0b000
~~0b11~~	~~0b0110~~	~~0b000~~	~~0b000~~	~~0b0101~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV1> == '01' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.ATA == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.ATA == '0' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<NV2,NV1,NV> == '111' then
        NVMem[0x190] = X[t];
    else
        TFSR_EL1 = X[t];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.ATA == '0' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HCR_EL2.E2H == '1' then
        TFSR_EL2 = X[t];
    else
        TFSR_EL1 = X[t];
elsif PSTATE.EL == EL3 then
    TFSR_EL1 = X[t];
```

```
MRS <Xt>, TFSR_EL12
```

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b11	0b101	0b0110	0b0110	0b000
~~0b11~~	~~0b0110~~	~~0b101~~	~~0b000~~	~~0b0101~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '101' then
        return NVMem[0x190];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if EL2Enabled() && HCR_EL2.E2H == '1' then
        if HaveEL(EL3) PSTATE.EL && !ELUsingAArch32(EL3) && SCR_EL3.ATA == '0' EL2 then
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return TFSR_EL1;
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if EL2Enabled() && HCR_EL2.E2H == '1' then
        return TFSR_EL1;
else
    else
        UNDEFINED;
```

MSR TFSR_EL12, <Xt>

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b11	0b101	0b0110	0b0110	0b000
~~0b11~~	~~0b0110~~	~~0b101~~	~~0b000~~	~~0b0101~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '101' then
        NVMem[0x190] = X[t];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if EL2Enabled() && HCR_EL2.E2H == '1' then
        if HaveEL(EL3) PSTATE.EL && !ELUsingAArch32(EL3) && SCR_EL3.ATA == '0' EL2 then
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            TFSR_EL1 = X[t];
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if EL2Enabled() && HCR_EL2.E2H == '1' then
        TFSR_EL1 = X[t];
else
    else
        UNDEFINED;
```

MRS <Xt>, TFSR_EL2

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b11	0b100	0b0110	0b0101	0b000
~~0b11~~	~~0b0110~~	~~0b100~~	~~0b000~~	~~0b0101~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.ATA == '0' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.ATA == '0' then
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return TFSR_EL1;
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.ATA == '0' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return TFSR_EL2;
elsif PSTATE.EL == EL3 then
    return TFSR_EL2;
```

```
MSR TFSR_EL2, <Xt>
```

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b11	0b100	0b0110	0b0101	0b000
~~0b11~~	~~0b0110~~	~~0b100~~	~~0b000~~	~~0b0101~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.ATA == '0' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.ATA == '0' then
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            TFSR_EL1 = X[t];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.ATA == '0' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        TFSR_EL2 = X[t];
elsif PSTATE.EL == EL3 then
    TFSR_EL2 = X[t];
```

27~~13~~/03~~12~~/2019~~2018~~ 21~~16~~:59~~42~~; e5e4db499bf9867a4b93324c4dbac985d3da93766~~379d01e197f1d40720d32d0f84e419e9187e009~~

(old) htmldiff from- (new)

TFSR_EL2, Tag Fail Status Register (EL2).

The TFSR_EL2 characteristics are:

Purpose

Holds accumulated Tag Check Fails occurring in EL2 which are not taken precisely.

Configuration

This register is present only when ARMv8.5-MemTag is implemented. Otherwise, direct accesses to TFSR_EL2 are UNDEFINED.

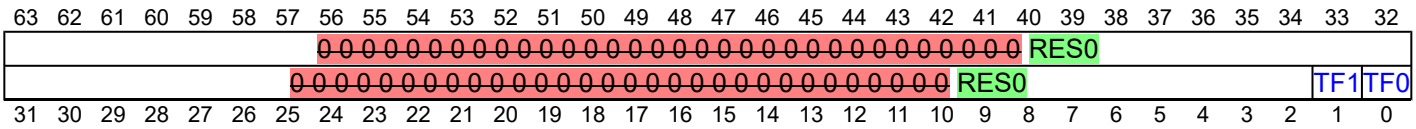
RW fields in this register reset to architecturally UNKNOWN values.

Attributes

TFSR_EL2 is a 64-bit register.

Field descriptions

The TFSR_EL2 bit assignments are:



Bits [63:2]

Reserved, RES0.

TF1, bit [1]

Tag Check Fail. Asynchronously set to 1 when a Tag Check fail using a virtual address with bit<55>==`0b1` occurs.

When HCR_EL2.E2H==`0b0`, this field is RES0.

This field resets to an architecturally UNKNOWN value.

TF0, bit [0]

Tag Check Fail. Asynchronously set to 1 when a Tag Check fail using a virtual address with bit<55>==`0b0` occurs.

This field resets to an architecturally UNKNOWN value.

Accessing the TFSR_EL2

When HCR_EL2.E2H is 1, without explicit synchronization, access from EL2 using the mnemonic TFSR_EL2 or TFSR_EL1 are not guaranteed to be ordered with respect to accesses using the other mnemonic.

Accesses to this register use the following encodings:

MRS <Xt>, TFSR_EL2

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b11	0b100	0b0110	0b0101	0b000
~~0b11~~	~~0b0110~~	~~0b100~~	~~0b000~~	~~0b0101~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.ATA == '0' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.ATA == '0' then
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return TFSR_EL1;
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.ATA == '0' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return TFSR_EL2;
elsif PSTATE.EL == EL3 then
    return TFSR_EL2;
```

MSR TFSR_EL2, <Xt>

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b11	0b100	0b0110	0b0101	0b000
~~0b11~~	~~0b0110~~	~~0b100~~	~~0b000~~	~~0b0101~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.ATA == '0' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.ATA == '0' then
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            TFSR_EL1 = X[t];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.ATA == '0' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        TFSR_EL2 = X[t];
elsif PSTATE.EL == EL3 then
    TFSR_EL2 = X[t];
```

MRS <Xt>, TFSR_EL1

op0	op1	CRn	CRm	op2

~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b11	0b000	0b0110	0b0101	0b000
~~0b11~~	~~0b0110~~	~~0b000~~	~~0b000~~	~~0b0101~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV1> == '01' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.ATA == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.ATA == '0' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<NV2,NV1,NV> == '111' then
        return NVMem[0x190];
    else
        return TFSR_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.ATA == '0' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HCR_EL2.E2H == '1' then
        return TFSR_EL2;
    else
        return TFSR_EL1;
elsif PSTATE.EL == EL3 then
    return TFSR_EL1;
```

```
MSR TFSR_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b11	0b000	0b0110	0b0101	0b000
~~0b11~~	~~0b0110~~	~~0b000~~	~~0b000~~	~~0b0101~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV1> == '01' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.ATA == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.ATA == '0' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<NV2,NV1,NV> == '111' then
        NVMem[0x190] = X[t];
    else
        TFSR_EL1 = X[t];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.ATA == '0' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HCR_EL2.E2H == '1' then
        TFSR_EL2 = X[t];
    else
        TFSR_EL1 = X[t];
elsif PSTATE.EL == EL3 then
    TFSR_EL1 = X[t];
```

27~~13~~/03~~12~~/2019~~2018~~ 21~~16~~:59~~42~~: e5e4db499bf9867a4b93324c4dbac985d3da93766379d01e197f1d40720d32d0f84e419e9187e009

(old) htmldiff from- (new)

TFSRE0_EL1, Tag Fail Status Register (EL0).

The TFSRE0_EL1 characteristics are:

Purpose

Holds accumulated Tag Check Fails occurring in EL0 which are not taken precisely.

Configuration

This register is present only when ARMv8.5-MemTag is implemented. Otherwise, direct accesses to TFSRE0_EL1 are UNDEFINED.

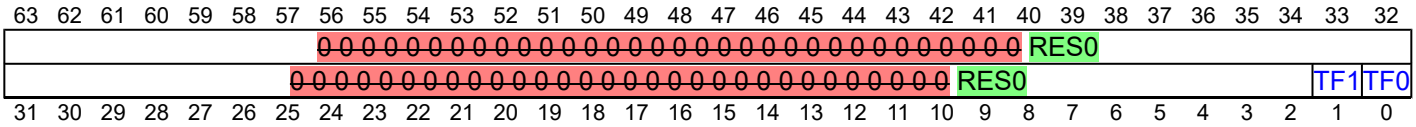
RW fields in this register reset to architecturally UNKNOWN values.

Attributes

TFSRE0_EL1 is a 64-bit register.

Field descriptions

The TFSRE0_EL1 bit assignments are:



Bits [63:2]

Reserved, RES0.

TF1, bit [1]

Tag Check Fail. Asynchronously set to 1 when a Tag Check fail using a virtual address with bit<55>==`0b1` occurs.

This field resets to an architecturally UNKNOWN value.

TF0, bit [0]

Tag Check Fail. Asynchronously set to 1 when a Tag Check fail using a virtual address with bit<55>==`0b0` occurs.

This field resets to an architecturally UNKNOWN value.

Accessing the TFSRE0_EL1

Accesses to this register use the following encodings:

```
MRS <Xt>, TFSRE0_EL1
```

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b11	0b000	0b0110	0b0110	0b001

0b11	0b0110	0b000	0b001	0b0110

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.ATA == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.ATA == '0' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return TFSRE0_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.ATA == '0' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return TFSRE0_EL1;
elsif PSTATE.EL == EL3 then
    return TFSRE0_EL1;
```

```
MSR TFSRE0_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
op0	CRn	op1	op2	CRm
0b11	0b000	0b0110	0b0110	0b001
0b11	0b0110	0b000	0b001	0b0110

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.ATA == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.ATA == '0' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        TFSRE0_EL1 = X[t];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.ATA == '0' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        TFSRE0_EL1 = X[t];
elsif PSTATE.EL == EL3 then
    TFSRE0_EL1 = X[t];
```

27~~13~~/03~~12~~/2019~~2018~~ 21~~16~~:59~~42~~; e5e4db499bf9867a4b93324c4dbac985d3da9376~~6379d01e197f1d40720d32d0f84c419e9187e009~~

(old) htmldiff from- (new)

TLBI ALLE1, TLB Invalidate All, EL1

The TLBI ALLE1 characteristics are:

Purpose

Invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 1 or stage 2 translation table entry, from any level of the translation table walk.
- If SCR_EL3.NS is 0 and the entry would be required to translate an address using the Secure EL1&0 translation regime.
- If SCR_EL3.NS is 1 and the entry would be required to translate an address using the Non-secure EL1&0 translation regime.

The invalidation applies to entries with any VMID.

The invalidation only applies to the PE that executes this System instruction.

Note

For the EL1&0 translation regimes, the invalidation applies to both global entries, and non-global entries with any ASID.

Configuration

Attributes

TLBI ALLE1 is a 64-bit System instruction.

Field descriptions

TLBI ALLE1 ignores the value in the register specified by the instruction encoding. Software does not have to write a value to the register before issuing this instruction.

Executing the TLBI ALLE1 instruction

Accesses to this instruction use the following encodings:

```
TLBI ALLE1{, <Xt>}
```

op0	op1	CRn	CRm	op2	Rt
~~Rt~~	~~op0~~	~~op1~~	~~op2~~	~~CRn~~	~~CRm~~
0b01	0b100	0b1000	0b0111	0b100	0b11111
~~0b11111~~	~~0b01~~	~~0b100~~	~~0b100~~	~~0b1000~~	~~0b0111~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    TLBI_ALLE1();
elsif PSTATE.EL == EL3 then
    TLBI_ALLE1();
```

27~~13~~/03~~12~~/2019~~2018~~ 21~~16~~:59~~42~~; e5e4db499bf9867a4b93324c4dbac985d3da93766~~379d01e197f1d40720d32d0f84e419e9187e009~~

(old) htmldiff from- (new)

(old) htmldiff from- (new)

TLBI ALLE1IS, TLB Invalidate All, EL1, Inner Shareable

The TLBI ALLE1IS characteristics are:

Purpose

Invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 1 or stage 2 translation table entry, from any level of the translation table walk.
- If SCR_EL3.NS is 0 and the entry would be required to translate an address using the Secure EL1&0 translation regime.
- If SCR_EL3.NS is 1 and the entry would be required to translate an address using the Non-secure EL1&0 translation regime.

The invalidation applies to entries with any VMID.

The invalidation applies to all PEs in the same Inner Shareable shareability domain as the PE that executes this System instruction.~~instructions.~~

Note

For the EL1&0 translation regimes, the invalidation applies to both global entries, and non-global entries with any ASID.

Configuration

Attributes

TLBI ALLE1IS is a 64-bit System instruction.

Field descriptions

TLBI ALLE1IS ignores the value in the register specified by the instruction encoding. Software does not have to write a value to the register before issuing this instruction.

Executing the TLBI ALLE1IS instruction

Accesses to this instruction use the following encodings:

```
TLBI ALLE1IS{, <Xt>}
```

op0	op1	CRn	CRm	op2	Rt
~~Rt~~	~~op0~~	~~op1~~	~~op2~~	~~CRn~~	~~CRm~~
0b01	0b100	0b1000	0b0011	0b100	0b11111
~~0b11111~~	~~0b01~~	~~0b100~~	~~0b100~~	~~0b1000~~	~~0b0011~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    TLBI_ALLE1IS();
elsif PSTATE.EL == EL3 then
    TLBI_ALLE1IS();
```

27~~13~~/03~~12~~/2019~~2018~~ 21~~16~~:59~~42~~; e5e4db499bf9867a4b93324c4dbac985d3da9376~~6379d01e197f1d40720d32d0f84e419e9187e009~~

(old) htmldiff from- (new)

(old) htmldiff from- (new)

TLBI ALLE1OS, TLB Invalidate All, EL1, Outer Shareable

The TLBI ALLE1OS characteristics are:

Purpose

Invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 1 or stage 2 translation table entry, from any level of the translation table walk.
- If SCR_EL3.NS is 0 and the entry would be required to translate an address using the Secure EL1&0 translation regime.
- If SCR_EL3.NS is 1 and the entry would be required to translate an address using the Non-secure EL1&0 translation regime.

The invalidation applies to entries with any VMID.

The invalidation applies to all PEs in the same Outer Shareable shareability domain as the PE that executes this System instruction.~~instructions.~~

Note

For the EL1&0 translation regimes, the invalidation applies to both global entries, and non-global entries with any ASID.

Configuration

This instruction is present only when ARMv8.4-TLBI is implemented. Otherwise, direct accesses to TLBI ALLE1OS are UNDEFINED.

Attributes

TLBI ALLE1OS is a 64-bit System instruction.

Field descriptions

TLBI ALLE1OS ignores the value in the register specified by the instruction encoding. Software does not have to write a value to the register before issuing this instruction.

Executing the TLBI ALLE1OS instruction

Accesses to this instruction use the following encodings:

```
TLBI ALLE1OS{, <Xt>}
```

op0	op1	CRn	CRm	op2	Rt
~~Rt~~	~~op0~~	~~op1~~	~~op2~~	~~CRn~~	~~CRm~~
0b01	0b100	0b1000	0b0001	0b100	0b11111
~~0b11111~~	~~0b01~~	~~0b100~~	~~0b100~~	~~0b1000~~	~~0b0001~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    TLBI_ALLE1OS();
elsif PSTATE.EL == EL3 then
    TLBI_ALLE1OS();
```

2713/0312/20192018 2116:5942; e5e4db499bf9867a4b93324c4dbac985d3da93766379d01e197f1d40720d32d0f84e419e9187e009

(old) htmldiff from- (new)

TLBI ALLE2, TLB Invalidate All, EL2

The TLBI ALLE2 characteristics are:

Purpose

If EL2 is implemented and enabled in the current Security state, invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 1 translation table entry, from any level of the translation table walk.
- If SCR_EL3.NS is 1 and the entry would be required to translate an address using the Non-secure EL2 or Non-secure EL2&0 translation regime.
- If SCR_EL3.NS is 0 and the entry would be required to translate an address using the Secure EL2 or Secure EL2&0 translation regime.

The invalidation only applies to the PE that executes this System instruction.

Configuration

Attributes

TLBI ALLE2 is a 64-bit System instruction.

Field descriptions

TLBI ALLE2 ignores the value in the register specified by the instruction encoding. Software does not have to write a value to the register before issuing this instruction.

Executing the TLBI ALLE2 instruction

Accesses to this instruction use the following encodings:

```
TLBI ALLE2{, <Xt>}
```

op0	op1	CRn	CRm	op2	Rt
~~Rt~~	~~op0~~	~~op1~~	~~op2~~	~~CRn~~	~~CRm~~
0b01	0b100	0b1000	0b0111	0b000	0b11111
~~0b11111~~	~~0b01~~	~~0b100~~	~~0b000~~	~~0b1000~~	~~0b0111~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    TLBI_ALLE2();
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        UNDEFINED;
    else
        TLBI_ALLE2();
```

27~~13~~/03~~12~~/2019~~2018~~ 21~~16~~:59~~42~~; e5e4db499bf9867a4b93324c4dbac985d3da93766~~379d01e197f1d40720d32d0f84e419e9187e009~~

(old) htmldiff from- (new)

(old) htmldiff from- (new)

TLBI ALLE2IS, TLB Invalidate All, EL2, Inner Shareable

The TLBI ALLE2IS characteristics are:

Purpose

If EL2 is implemented and enabled in the current Security state, invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 1 translation table entry, from any level of the translation table walk.
- If SCR_EL3.NS is 1 and the entry would be required to translate an address using the Non-secure EL2 or Non-secure EL2&0 translation regime.
- If SCR_EL3.NS is 0 and the entry would be required to translate an address using the Secure EL2 or Secure EL2&0 translation regime.

The invalidation ~~only~~ applies to all PEs in the same Inner Shareable shareability domain as the PE that executes this System instruction.

Configuration

Attributes

TLBI ALLE2IS is a 64-bit System instruction.

Field descriptions

TLBI ALLE2IS ignores the value in the register specified by the instruction encoding. Software does not have to write a value to the register before issuing this instruction.

Executing the TLBI ALLE2IS instruction

Accesses to this instruction use the following encodings:

```
TLBI ALLE2IS{, <Xt>}
```

op0	op1	CRn	CRm	op2	Rt
~~Rt~~	~~op0~~	~~op1~~	~~op2~~	~~CRn~~	~~CRm~~
0b01	0b100	0b1000	0b0011	0b000	0b11111
~~0b11111~~	~~0b01~~	~~0b100~~	~~0b000~~	~~0b1000~~	~~0b0011~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    TLBI_ALLE2IS();
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        UNDEFINED;
    else
        TLBI_ALLE2IS();
```

27~~13~~/03~~12~~/2019~~2018~~ 21~~16~~:59~~42~~; e5e4db499bf9867a4b93324c4dbac985d3da93766~~379d01e197f1d40720d32d0f84e419e9187e009~~

(old) htmldiff from- (new)

(old) htmldiff from- (new)

TLBI ALLE2OS, TLB Invalidate All, EL2, Outer Shareable

The TLBI ALLE2OS characteristics are:

Purpose

If EL2 is implemented and enabled in the current Security state, invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 1 translation table entry, from any level of the translation table walk.
- If SCR_EL3.NS is 1 and the entry would be required to translate an address using the Non-secure EL2 or Non-secure EL2&0 translation regime.
- If SCR_EL3.NS is 0 and the entry would be required to translate an address using the Secure EL2 or Secure EL2&0 translation regime.

The invalidation ~~only~~ applies to all PEs in the same Outer Shareable shareability domain as the PE that executes this System instruction.

Configuration

This instruction is present only when ARMv8.4-TLBI is implemented. Otherwise, direct accesses to TLBI ALLE2OS are UNDEFINED.

Attributes

TLBI ALLE2OS is a 64-bit System instruction.

Field descriptions

TLBI ALLE2OS ignores the value in the register specified by the instruction encoding. Software does not have to write a value to the register before issuing this instruction.

Executing the TLBI ALLE2OS instruction

Accesses to this instruction use the following encodings:

```
TLBI ALLE2OS{, <Xt>}
```

op0	op1	CRn	CRm	op2	Rt
~~Rt~~	~~op0~~	~~op1~~	~~op2~~	~~CRn~~	~~CRm~~
0b01	0b100	0b1000	0b0001	0b000	0b11111
~~0b11111~~	~~0b01~~	~~0b100~~	~~0b000~~	~~0b1000~~	~~0b0001~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    TLBI_ALLE2OS();
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        UNDEFINED;
    else
        TLBI_ALLE2OS();
```

2713/0312/20192018 2116:5942; e5e4db499bf9867a4b93324c4dbac985d3da93766379d01e197f1d40720d32d0f84e419e9187e009

(old) htmldiff from- (new)

(old) htmldiff from- (new)

TLBI ALLE3, TLB Invalidate All, EL3

The TLBI ALLE3 characteristics are:

Purpose

If EL3 is implemented, invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 1 translation table entry, from any level of the translation table walk.
- The entry would be required to translate an address using the EL3 translation regime.

The invalidation applies to ~~all PEs in~~ the ~~same Inner Shareable shareability domain as the~~ PE that executes this System instruction.~~instructions.~~

Configuration

Attributes

TLBI ALLE3 is a 64-bit System instruction.

Field descriptions

TLBI ALLE3 ignores the value in the register specified by the instruction encoding. Software does not have to write a value to the register before issuing this instruction.

Executing the TLBI ALLE3 instruction

Accesses to this instruction use the following encodings:

```
TLBI ALLE3{, <Xt>}
```

op0	op1	CRn	CRm	op2	Rt
~~Rt~~	~~op0~~	~~op1~~	~~op2~~	~~CRn~~	~~CRm~~
0b01	0b110	0b1000	0b0111	0b000	0b11111
~~0b11111~~	~~0b01~~	~~0b110~~	~~0b000~~	~~0b1000~~	~~0b0111~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    TLBI_ALLE3();
```

2713/0312/20192018 2116:5942; e5e4db499bf9867a4b93324c4dbac985d3da93766379d01e197f1d40720d32d0f84c419e9187e009

(old) htmldiff from- (new)

(old) htmldiff from- (new)

TLBI ALLE3IS, TLB Invalidate All, EL3, Inner Shareable

The TLBI ALLE3IS characteristics are:

Purpose

If EL3 is implemented, invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 1 translation table entry, from any level of the translation table walk.
- The entry would be required to translate an address using the EL3 translation regime.

The invalidation ~~only~~ applies to all PEs in the same Inner Shareable shareability domain as the PE that executes this System instruction.

Configuration

Attributes

TLBI ALLE3IS is a 64-bit System instruction.

Field descriptions

TLBI ALLE3IS ignores the value in the register specified by the instruction encoding. Software does not have to write a value to the register before issuing this instruction.

Executing the TLBI ALLE3IS instruction

Accesses to this instruction use the following encodings:

```
TLBI ALLE3IS{, <Xt>}
```

op0	op1	CRn	CRm	op2	Rt
~~Rt~~	~~op0~~	~~op1~~	~~op2~~	~~CRn~~	~~CRm~~
0b01	0b110	0b1000	0b0011	0b000	0b11111
~~0b11111~~	~~0b01~~	~~0b110~~	~~0b000~~	~~0b1000~~	~~0b0011~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    TLBI_ALLE3IS();
```

27~~13~~/03~~12~~/2019~~2018~~ 21~~16~~:59~~42~~; e5e4db499bf9867a4b93324c4dbac985d3da93766~~379d01e197f1d40720d32d0f84e419e9187e009~~

(old) htmldiff from- (new)

(old) htmldiff from- (new)

TLBI ALLE3OS, TLB Invalidate All, EL3, Outer Shareable

The TLBI ALLE3OS characteristics are:

Purpose

If EL3 is implemented, invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 1 translation table entry, from any level of the translation table walk.
- The entry would be required to translate an address using the EL3 translation regime.

The invalidation ~~only~~ applies to all PEs in the same Outer Shareable shareability domain as the PE that executes this System instruction.

Configuration

This instruction is present only when ARMv8.4-TLBI is implemented. Otherwise, direct accesses to TLBI ALLE3OS are UNDEFINED.

Attributes

TLBI ALLE3OS is a 64-bit System instruction.

Field descriptions

TLBI ALLE3OS ignores the value in the register specified by the instruction encoding. Software does not have to write a value to the register before issuing this instruction.

Executing the TLBI ALLE3OS instruction

Accesses to this instruction use the following encodings:

```
TLBI ALLE3OS{, <Xt>}
```

op0	op1	CRn	CRm	op2	Rt
~~Rt~~	~~op0~~	~~op1~~	~~op2~~	~~CRn~~	~~CRm~~
0b01	0b110	0b1000	0b0001	0b000	0b11111
~~0b11111~~	~~0b01~~	~~0b110~~	~~0b000~~	~~0b1000~~	~~0b0001~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    TLBI_ALLE3OS();
```

27~~13~~/03~~12~~/2019~~2018~~ 21~~16~~:59~~42~~: e5e4db499bf9867a4b93324c4dbac985d3da9376~~6379d01e197f1d40720d32d0f84e419e9187e009~~

(old) htmldiff from- (new)

(old) htmldiff from- (new)

TLBI ASIDE1, TLB Invalidate by ASID, EL1

The TLBI ASIDE1 characteristics are:

Purpose

Invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 1 translation table entry.
- The entry would be used for the specified ASID, and either:
  - Is from a level of lookup above the final level.
  - Is a non-global entry from the final level of lookup.
- When EL2 is implemented and enabled in the ~~current~~ Security state described by the current value of~~:~~ SCR_EL3.NS:
  - If HCR_EL2.{E2H, TGE} is not {1, 1}, the entry would be used with the current VMID, and would be required to translate an address using the EL1&0 translation regime.
  - If HCR_EL2.{E2H, TGE} is {1, 1}, the entry would be required to translate an address using the EL2&0 translation regime.
- When EL2 is not implemented or is disabled in the current Security state, the entry would be required to translate an address using the EL1&0 translation regime.

The invalidation ~~only~~ applies to the PE that executes this System instruction.

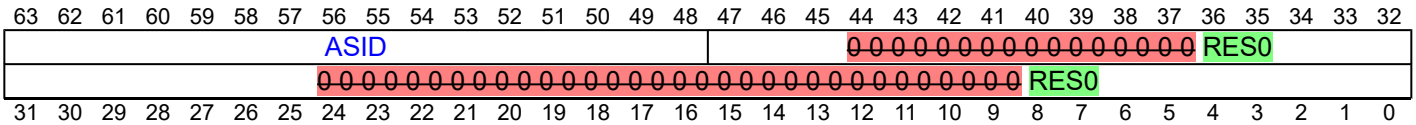
Configuration

Attributes

TLBI ASIDE1 is a 64-bit System instruction.

Field descriptions

The TLBI ASIDE1 input value bit assignments are:



ASID, bits [63:48]

ASID value to match. Any appropriate TLB entries that match the ASID values will be affected by this System instruction.~~operation.~~

If the implementation supports 16 bits of ASID, but only 8 bits are being used in the context being invalidated, the upper bits are RES0.

Bits [47:0]

Reserved, RES0.

Executing the TLBI ASIDE1 instruction

Accesses to this instruction use the following encodings:

```
TLBI ASIDE1{, <Xt>}
```

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b01	0b000	0b1000	0b0111	0b010
~~0b01~~	~~0b1000~~	~~0b000~~	~~0b010~~	~~0b0111~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TTLB == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.FB == '1' then
        TLBI_ASIDE1IS(X[t]);
    else
        TLBI_ASIDE1(X[t]);
elsif PSTATE.EL == EL2 then
    TLBI_ASIDE1(X[t]);
elsif PSTATE.EL == EL3 then
    TLBI_ASIDE1(X[t]);
```

27~~13~~/03~~12~~/2019~~2018~~ 21~~16~~:59~~42~~: e5e4db499bf9867a4b93324c4dbac985d3da93766379d01e197f1d40720d32d0f84e419e9187e009

(old) htmldiff from- (new)

TLBI ASIDE1IS, TLB Invalidate by ASID, EL1, Inner Shareable

The TLBI ASIDE1IS characteristics are:

Purpose

Invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 1 translation table entry.
- The entry would be used for the specified ASID, and either:
  - Is from a level of lookup above the final level.
  - Is a non-global entry from the final level of lookup.
- When EL2 is implemented and enabled in the ~~current~~ Security state described by the current value of~~:~~ SCR_EL3.NS:
  - If HCR_EL2.{E2H, TGE} is not {1, 1}, the entry would be used with the current VMID, and would be required to translate an address using the EL1&0 translation regime.
  - If HCR_EL2.{E2H, TGE} is {1, 1}, the entry would be required to translate an address using the EL2&0 translation regime.
- When EL2 is not implemented or is disabled in the current Security state, the entry would be required to translate an address using the EL1&0 translation regime.

The invalidation applies to all PEs in the same Inner Shareable shareability domain as the PE that executes this System instruction.~~instructions.~~

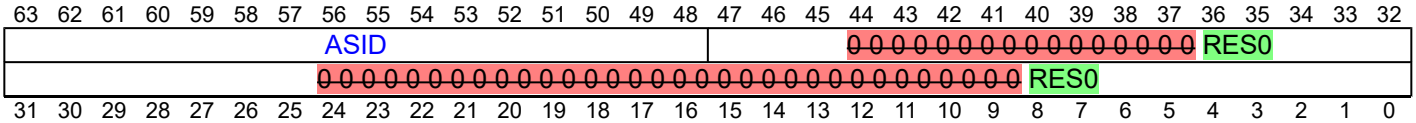
Configuration

Attributes

TLBI ASIDE1IS is a 64-bit System instruction.

Field descriptions

The TLBI ASIDE1IS input value bit assignments are:



ASID, bits [63:48]

ASID value to match. Any appropriate TLB entries that match the ASID values will be affected by this System instruction.~~operation.~~

If the implementation supports 16 bits of ASID, but only 8 bits are being used in the context being invalidated, the upper bits are RES0.

Bits [47:0]

Reserved, RES0.

Executing the TLBI ASIDE1IS instruction

Accesses to this instruction use the following encodings:

```
TLBI ASIDE1IS{, <Xt>}
```

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b01	0b000	0b1000	0b0011	0b010
~~0b01~~	~~0b1000~~	~~0b000~~	~~0b010~~	~~0b0011~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TTLB == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TTLBIS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        TLBI_ASIDE1IS(X[t]);
elsif PSTATE.EL == EL2 then
    TLBI_ASIDE1IS(X[t]);
elsif PSTATE.EL == EL3 then
    TLBI_ASIDE1IS(X[t]);
```

27~~13~~/03~~12~~/2019~~2018~~ 21~~16~~:59~~42~~: e5e4db499bf9867a4b93324c4dbac985d3da93766379d01e197f1d40720d32d0f84e419e9187e009

(old) htmldiff from- (new)

TLBI ASIDE1OS, TLB Invalidate by ASID, EL1, Outer Shareable

The TLBI ASIDE1OS characteristics are:

Purpose

Invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 1 translation table entry.
- The entry would be used for the specified ASID, and either:
  - Is from a level of lookup above the final level.
  - Is a non-global entry from the final level of lookup.
- When EL2 is implemented and enabled in the ~~current~~ Security state described by the current value of~~:~~ SCR_EL3.NS:
  - If HCR_EL2.{E2H, TGE} is not {1, 1}, the entry would be used with the current VMID, and would be required to translate an address using the EL1&0 translation regime.
  - If HCR_EL2.{E2H, TGE} is {1, 1}, the entry would be required to translate an address using the EL2&0 translation regime.
- When EL2 is not implemented or is disabled in the current Security state, the entry would be required to translate an address using the EL1&0 translation regime.

The invalidation applies to all PEs in the same Outer Shareable shareability domain as the PE that executes this System instruction.~~instructions.~~

Configuration

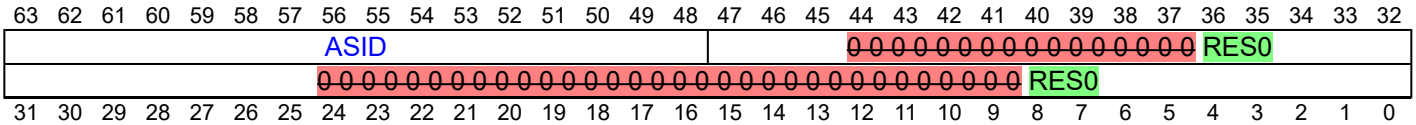
This instruction is present only when ARMv8.4-TLBI is implemented. Otherwise, direct accesses to TLBI ASIDE1OS are UNDEFINED.

Attributes

TLBI ASIDE1OS is a 64-bit System instruction.

Field descriptions

The TLBI ASIDE1OS input value bit assignments are:



ASID, bits [63:48]

ASID value to match. Any appropriate TLB entries that match the ASID values will be affected by this System instruction.~~operation.~~

If the implementation supports 16 bits of ASID, but only 8 bits are being used in the context being invalidated, the upper bits are RES0.

Bits [47:0]

Reserved, RES0.

Executing the TLBI ASIDE1OS instruction

Accesses to this instruction use the following encodings:

```
TLBI ASIDE1OS{, <Xt>}
```

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b01	0b000	0b1000	0b0001	0b010
~~0b01~~	~~0b1000~~	~~0b000~~	~~0b010~~	~~0b0001~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TTLB == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TTLBOS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        TLBI_ASIDE1OS(X[t]);
elsif PSTATE.EL == EL2 then
    TLBI_ASIDE1OS(X[t]);
elsif PSTATE.EL == EL3 then
    TLBI_ASIDE1OS(X[t]);
```

27~~13~~/03~~12~~/2019~~2018~~ 21~~16~~:59~~42~~; e5e4db499bf9867a4b93324c4dbac985d3da9376~~6379d01e197f1d40720d32d0f84c419e9187e009~~

(old) htmldiff from- (new)

TLBI IPAS2E1, TLB Invalidate by Intermediate Physical Address, Stage 2, EL1

The TLBI IPAS2E1 characteristics are:

Purpose

If EL2 is implemented and enabled in the current Security state, invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 2 only translation table entry, from any level of the translation table walk.
- One of the following applies:
  - SCR_EL3.NS==1 and the entry would be required to translate the specified IPA using the Non-secure EL1&0 translation regime.
  - SCR_EL3.NS==0 and the entry would be required to translate the specified IPA using the Secure EL1&0 translation regime.
- The entry would be used with the current VMID.

The invalidation is not required to apply to caching structures that combine stage 1 and stage 2 translation table entries.

The invalidation ~~only~~ applies to the PE that executes this System instruction.

For more information about the architectural requirements for this System instruction see 'Invalidation of TLB entries from stage 2 translations' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

Configuration

Attributes

TLBI IPAS2E1 is a 64-bit System instruction.

Field descriptions

The TLBI IPAS2E1 input value bit assignments are:

63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
NS	RES0															TTL				RES0				IPA[51:48]				IPA[47:12]			
IPA[47:12]																															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
~~63~~	~~62~~	~~61~~	~~60~~	~~59~~	~~58~~	~~57~~	~~56~~	~~55~~	~~54~~	~~53~~	~~52~~	~~51~~	~~50~~	~~49~~	~~48~~	~~47~~	~~46~~	~~45~~	~~44~~	~~43~~	~~42~~	~~41~~	~~40~~	~~39~~	~~38~~	~~37~~	~~36~~	~~35~~	~~34~~	~~33~~	~~32~~
~~NS~~	~~0~~	~~0~~	~~0~~	~~0~~	~~0~~	~~0~~	~~0~~	~~0~~	~~0~~	~~0~~	~~0~~	~~0~~	~~0~~	~~0~~	~~0~~	TTL				~~0~~	~~0~~	~~0~~	~~0~~	IPA[51:48]				IPA[47:12]			
IPA[47:12]																															
~~31~~	~~30~~	~~29~~	~~28~~	~~27~~	~~26~~	~~25~~	~~24~~	~~23~~	~~22~~	~~21~~	~~20~~	~~19~~	~~18~~	~~17~~	~~16~~	~~15~~	~~14~~	~~13~~	~~12~~	~~11~~	~~10~~	~~9~~	~~8~~	~~7~~	~~6~~	~~5~~	~~4~~	~~3~~	~~2~~	~~1~~	~~0~~

NS, bit [63]

When ARMv8.4-SecEL2 is implemented:

Not Secure. Specifies the IPA space.

NS	Meaning
0b0	IPA is in the Secure IPA space.
0b1	IPA is in the Non-secure IPA space.

When the instruction is executed in Non-secure state, this field is RES0, and the instruction applies only to the Non-secure IPA space.

When ARMv8.4-SecEL2 is not implemented or is disabled in the current Security state, this field is RES0.

Otherwise:

Reserved, RES0.

Bits [62:48]

Reserved, RES0.

TTL, bits [47:44]

When ARMv8.4-TTL is implemented:

Translation Table Level. Indicates the level of the page table walk that holds the leaf entry for the address being invalidated.

TTL	Meaning
0b00xx	No information supplied as to the translation table level. Hardware must assume that the entry can be from any level. In this case, TTL<1:0> is RES0.
0b01xx	The entry comes from a 4KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3.
0b10xx	The entry comes from a 16KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : Reserved. Treat as if TTL<3:2> is 0b00. 0b10 : Level 2. 0b11 : Level 3.
0b11xx	The entry comes from a 64KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3.

If an incorrect value of the TTL field is specified for the entry being invalidated by the instruction, then no entries are required by the architecture to be invalidated from the TLB.

Otherwise:

Reserved, RES0.

Bits [43:40]

Reserved, RES0.

IPA[51:48], bits [39:36]

When ARMv8.2-LPA is implemented:

Extension to IPA[47:12]. See IPA[47:12] for more details.

Otherwise:

Reserved, RES0.

IPA[47:12], bits [35:0]

Bits[47:12] of the intermediate physical address to match. For implementations with fewer than 48 bits, the upper bits of this field are RES0.

When ARMv8.2-LPA is implemented, and 52-bit addresses and a 64KB translation granule are in use, IPA[51:48] form the upper part of the address value. Otherwise, IPA[51:48] are RES0.

Executing the TLBI IPAS2E1 instruction

Accesses to this instruction use the following encodings:

```
TLBI IPAS2E1{, <Xt>}
```

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b01	0b100	0b1000	0b0100	0b001
~~0b01~~	~~0b1000~~	~~0b100~~	~~0b001~~	~~0b0100~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    TLBI_IPAS2E1(X[t]);
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        //no operation
    else
        TLBI_IPAS2E1(X[t]);
```

27~~13~~/03~~12~~/2019~~2018~~ 21~~16~~:59~~42~~; e5e4db499bf9867a4b93324c4dbac985d3da93766~~379d01e197f1d40720d32d0f84e419e9187e009~~

(old) htmldiff from- (new)

TLBI IPAS2E1IS, TLB Invalidate by Intermediate Physical Address, Stage 2, EL1, Inner Shareable

The TLBI IPAS2E1IS characteristics are:

Purpose

If EL2 is implemented and enabled in the current Security state, invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 2 only translation table entry, from any level of the translation table walk.
- One of the following applies:
  - SCR_EL3.NS==1 and the entry would be required to translate the specified IPA using the Non-secure EL1&0 translation regime.
  - SCR_EL3.NS==0 and the entry would be required to translate the specified IPA using the Secure EL1&0 translation regime.
- The entry would be used with the current VMID.

The invalidation is not required to apply to caching structures that combine stage 1 and stage 2 translation table entries.

The invalidation ~~only~~ applies to all PEs in the same Inner Shareable shareability domain as the PE that executes this System instruction.

For more information about the architectural requirements for this System instruction see 'Invalidation of TLB entries from stage 2 translations' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

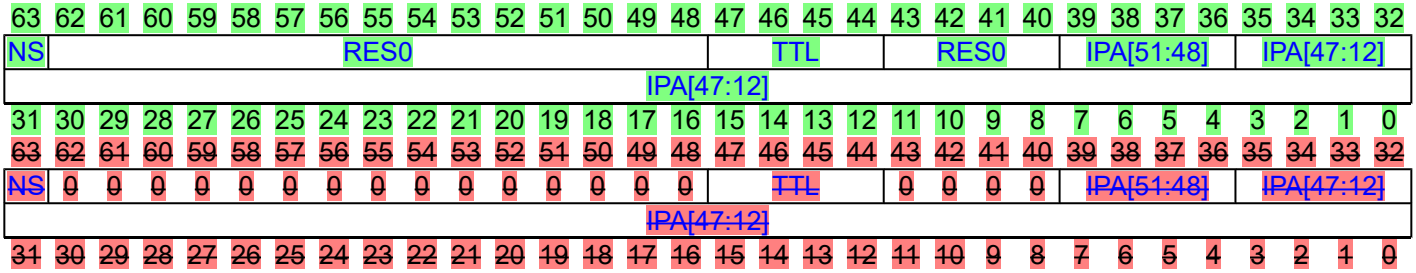
Configuration

Attributes

TLBI IPAS2E1IS is a 64-bit System instruction.

Field descriptions

The TLBI IPAS2E1IS input value bit assignments are:



NS, bit [63]

When ARMv8.4-SecEL2 is implemented:

Not Secure. Specifies the IPA space.

NS	Meaning
0b0	IPA is in the Secure IPA space.
0b1	IPA is in the Non-secure IPA space.

When the instruction is executed in Non-secure state, this field is RES0, and the instruction applies only to the Non-secure IPA space.

When ARMv8.4-SecEL2 is not implemented or is disabled in the current Security state, this field is RES0.

Otherwise:

Reserved, RES0.

Bits [62:48]

Reserved, RES0.

TTL, bits [47:44]

When ARMv8.4-TTL is implemented:

Translation Table Level. Indicates the level of the page table walk that holds the leaf entry for the address being invalidated.

TTL	Meaning
0b00xx	No information supplied as to the translation table level. Hardware must assume that the entry can be from any level. In this case, TTL<1:0> is RES0.
0b01xx	The entry comes from a 4KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3.
0b10xx	The entry comes from a 16KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : Reserved. Treat as if TTL<3:2> is 0b00. 0b10 : Level 2. 0b11 : Level 3.
0b11xx	The entry comes from a 64KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3.

If an incorrect value of the TTL field is specified for the entry being invalidated by the instruction, then no entries are required by the architecture to be invalidated from the TLB.

Otherwise:

Reserved, RES0.

Bits [43:40]

Reserved, RES0.

IPA[51:48], bits [39:36]

When ARMv8.2-LPA is implemented:

Extension to IPA[47:12]. See IPA[47:12] for more details.

Otherwise:

Reserved, RES0.

IPA[47:12], bits [35:0]

Bits[47:12] of the intermediate physical address to match. For implementations with fewer than 48 bits, the upper bits of this field are RES0.

When ARMv8.2-LPA is implemented, and 52-bit addresses and a 64KB translation granule are in use, IPA[51:48] form the upper part of the address value. Otherwise, IPA[51:48] are RES0.

Executing the TLBI IPAS2E1IS instruction

Accesses to this instruction use the following encodings:

```
TLBI IPAS2E1IS{, <Xt>}
```

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b01	0b100	0b1000	0b0000	0b001
~~0b01~~	~~0b1000~~	~~0b100~~	~~0b001~~	~~0b0000~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    TLBI_IPAS2E1IS(X[t]);
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        //no operation
    else
        TLBI_IPAS2E1IS(X[t]);
```

27~~13~~/03~~12~~/2019~~2018~~ 21~~16~~:59~~42~~; e5e4db499bf9867a4b93324c4dbac985d3da93766379d01e197f1d40720d32d0f84c419e9187e009

(old) htmldiff from- (new)

TLBI IPAS2E1OS, TLB Invalidate by Intermediate Physical Address, Stage 2, EL1, Outer Shareable

The TLBI IPAS2E1OS characteristics are:

Purpose

If EL2 is implemented and enabled in the current Security state, invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 2 only translation table entry, from any level of the translation table walk.
- One of the following applies:
  - SCR_EL3.NS==1 and the entry would be required to translate the specified IPA using the Non-secure EL1&0 translation regime.
  - SCR_EL3.NS==0 and the entry would be required to translate the specified IPA using the Secure EL1&0 translation regime.
- The entry would be used with the current VMID.

The invalidation is not required to apply to caching structures that combine stage 1 and stage 2 translation table entries.

The invalidation ~~only~~ applies to all PEs in the same Outer Shareable shareability domain as the PE that executes this System instruction.

For more information about the architectural requirements for this System instruction see 'Invalidation of TLB entries from stage 2 translations' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

Configuration

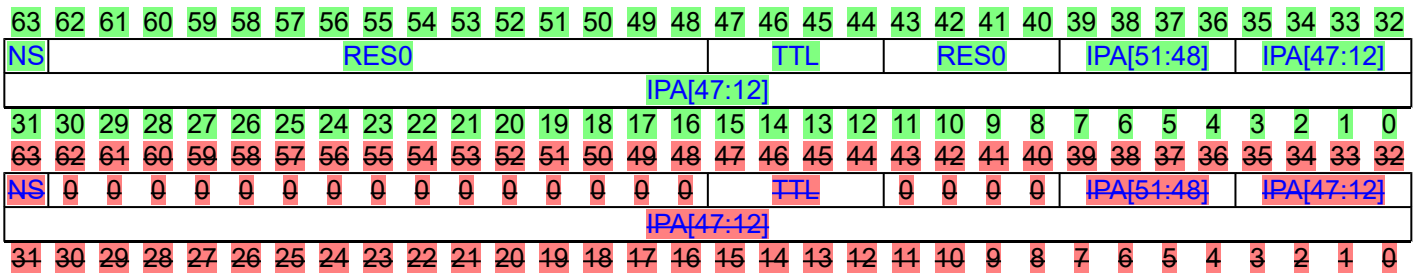
This instruction is present only when ARMv8.4-TLBI is implemented. Otherwise, direct accesses to TLBI IPAS2E1OS are UNDEFINED.

Attributes

TLBI IPAS2E1OS is a 64-bit System instruction.

Field descriptions

The TLBI IPAS2E1OS input value bit assignments are:



NS, bit [63]

Not Secure. Specifies the IPA space.

NS	Meaning
0b0	IPA is in the Secure IPA space.
0b1	IPA is in the Non-secure IPA space.

When the instruction is executed in Non-secure state, this field is RES0, and the instruction applies only to the Non-secure IPA space.

When ARMv8.4-SecEL2 is not implemented or is disabled in the current Security state, this field is RES0.

Bits [62:48]

Reserved, RES0.

TTL, bits [47:44]

When ARMv8.4-TTL is implemented:

Translation Table Level. Indicates the level of the page table walk that holds the leaf entry for the address being invalidated.

TTL	Meaning
0b00xx	No information supplied as to the translation table level. Hardware must assume that the entry can be from any level. In this case, TTL<1:0> is RES0.
0b01xx	The entry comes from a 4KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3.
0b10xx	The entry comes from a 16KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : Reserved. Treat as if TTL<3:2> is 0b00. 0b10 : Level 2. 0b11 : Level 3.
0b11xx	The entry comes from a 64KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3.

If an incorrect value of the TTL field is specified for the entry being invalidated by the instruction, then no entries are required by the architecture to be invalidated from the TLB.

Otherwise:

Reserved, RES0.

Bits [43:40]

Reserved, RES0.

IPA[51:48], bits [39:36]

Extension to IPA[47:12]. See IPA[47:12] for more details.

IPA[47:12], bits [35:0]

Bits[47:12] of the intermediate physical address to match. For implementations with fewer than 48 bits, the upper bits of this field are RES0.

When ARMv8.2-LPA is implemented, and 52-bit addresses and a 64KB translation granule are in use, IPA[51:48] form the upper part of the address value. Otherwise, IPA[51:48] are RES0.

Executing the TLBI IPAS2E1OS instruction

Accesses to this instruction use the following encodings:

```
TLBI IPAS2E1OS{, <Xt>}
```

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b01	0b100	0b1000	0b0100	0b000
~~0b01~~	~~0b1000~~	~~0b100~~	~~0b000~~	~~0b0100~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    TLBI_IPAS2E1OS(X[t]);
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        //no operation
    else
        TLBI_IPAS2E1OS(X[t]);
```

27~~13~~/03~~12~~/2019~~2018~~ 21~~16~~:59~~42~~: e5e4db499bf9867a4b93324c4dbac985d3da93766~~379d01e197f1d40720d32d0f84e419e9187e009~~

(old) htmldiff from- (new)

TLBI IPAS2LE1, TLB Invalidate by Intermediate Physical Address, Stage 2, Last level, EL1

The TLBI IPAS2LE1 characteristics are:

Purpose

If EL2 is implemented and enabled in the current Security state, invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 2 only translation table entry, from the final level of the translation table walk.
- One of the following applies:
  - SCR_EL3.NS==1 and the entry would be required to translate the specified IPA using the Non-secure EL1&0 translation regime.
  - SCR_EL3.NS==0 and the entry would be required to translate the specified IPA using the Secure EL1&0 translation regime.
- The entry would be used with the current VMID.

The invalidation is not required to apply to caching structures that combine stage 1 and stage 2 translation table entries.

The invalidation ~~only~~ applies to the PE that executes this System instruction.

For more information about the architectural requirements for this System instruction see 'Invalidation of TLB entries from stage 2 translations' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

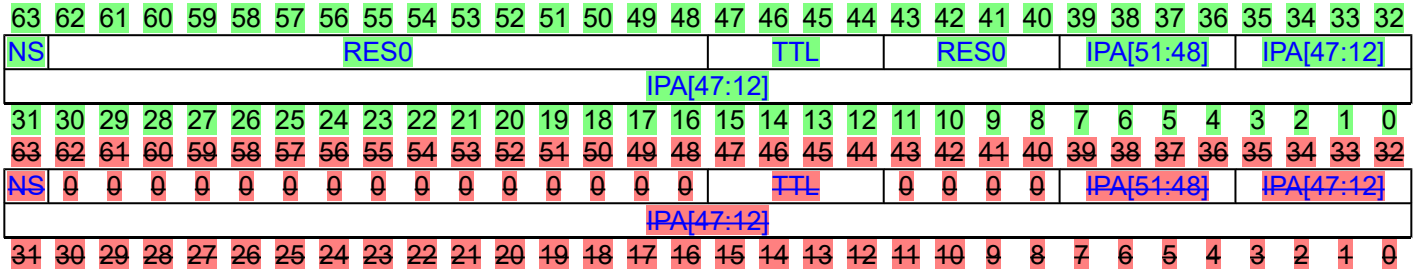
Configuration

Attributes

TLBI IPAS2LE1 is a 64-bit System instruction.

Field descriptions

The TLBI IPAS2LE1 input value bit assignments are:



NS, bit [63]

When ARMv8.4-SecEL2 is implemented:

Not Secure. Specifies the IPA space.

NS	Meaning
0b0	IPA is in the Secure IPA space.
0b1	IPA is in the Non-secure IPA space.

When the instruction is executed in Non-secure state, this field is RES0, and the instruction applies only to the Non-secure IPA space.

When ARMv8.4-SecEL2 is not implemented or is disabled in the current Security state, this field is RES0.

Otherwise:

Reserved, RES0.

Bits [62:48]

Reserved, RES0.

TTL, bits [47:44]

When ARMv8.4-TTL is implemented:

Translation Table Level. Indicates the level of the page table walk that holds the leaf entry for the address being invalidated.

TTL	Meaning
0b00xx	No information supplied as to the translation table level. Hardware must assume that the entry can be from any level. In this case, TTL<1:0> is RES0.
0b01xx	The entry comes from a 4KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3.
0b10xx	The entry comes from a 16KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : Reserved. Treat as if TTL<3:2> is 0b00. 0b10 : Level 2. 0b11 : Level 3.
0b11xx	The entry comes from a 64KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3.

If an incorrect value of the TTL field is specified for the entry being invalidated by the instruction, then no entries are required by the architecture to be invalidated from the TLB.

Otherwise:

Reserved, RES0.

Bits [43:40]

Reserved, RES0.

IPA[51:48], bits [39:36]

When ARMv8.2-LPA is implemented:

Extension to IPA[47:12]. See IPA[47:12] for more details.

Otherwise:

Reserved, RES0.

IPA[47:12], bits [35:0]

Bits[47:12] of the intermediate physical address to match. For implementations with fewer than 48 bits, the upper bits of this field are RES0.

When ARMv8.2-LPA is implemented, and 52-bit addresses and a 64KB translation granule are in use, IPA[51:48] form the upper part of the address value. Otherwise, IPA[51:48] are RES0.

Executing the TLBI IPAS2LE1 instruction

Accesses to this instruction use the following encodings:

```
TLBI IPAS2LE1{, <Xt>}
```

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b01	0b100	0b1000	0b0100	0b101
~~0b01~~	~~0b1000~~	~~0b100~~	~~0b101~~	~~0b0100~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    TLBI_IPAS2LE1(X[t]);
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        //no operation
    else
        TLBI_IPAS2LE1(X[t]);
```

27~~13~~/03~~12~~/2019~~2018~~ 21~~16~~:59~~42~~; e5e4db499bf9867a4b93324c4dbac985d3da93766~~379d01e197f1d40720d32d0f84c419e9187e009~~

(old) htmldiff from- (new)

TLBI IPAS2LE1IS, TLB Invalidate by Intermediate Physical Address, Stage 2, Last level, EL1, Inner Shareable

The TLBI IPAS2LE1IS characteristics are:

Purpose

If EL2 is implemented and enabled in the current Security state, invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 2 only translation table entry, from the final level of the translation table walk.
- One of the following applies:
  - SCR_EL3.NS==1 and the entry would be required to translate the specified IPA using the Non-secure EL1&0 translation regime.
  - SCR_EL3.NS==0 and the entry would be required to translate the specified IPA using the Secure EL1&0 translation regime.
- The entry would be used with the current VMID.

The invalidation is not required to apply to caching structures that combine stage 1 and stage 2 translation table entries.

The invalidation ~~only~~ applies to all PEs in the same Inner Shareable shareability domain as the PE that executes this System instruction.

For more information about the architectural requirements for this System instruction see 'Invalidation of TLB entries from stage 2 translations' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

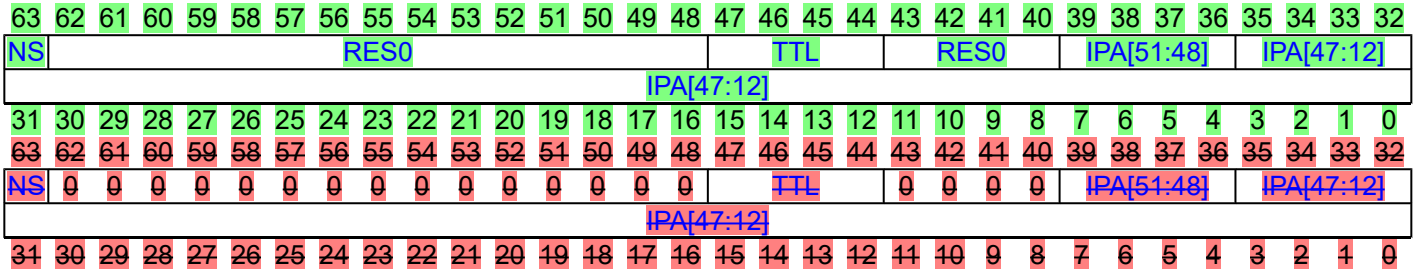
Configuration

Attributes

TLBI IPAS2LE1IS is a 64-bit System instruction.

Field descriptions

The TLBI IPAS2LE1IS input value bit assignments are:



NS, bit [63]

When ARMv8.4-SecEL2 is implemented:

Not Secure. Specifies the IPA space.

NS	Meaning
0b0	IPA is in the Secure IPA space.
0b1	IPA is in the Non-secure IPA space.

When the instruction is executed in Non-secure state, this field is RES0, and the instruction applies only to the Non-secure IPA space.

When ARMv8.4-SecEL2 is not implemented or is disabled in the current Security state, this field is RES0.

Otherwise:

Reserved, RES0.

Bits [62:48]

Reserved, RES0.

TTL, bits [47:44]

When ARMv8.4-TTL is implemented:

Translation Table Level. Indicates the level of the page table walk that holds the leaf entry for the address being invalidated.

TTL	Meaning
0b00xx	No information supplied as to the translation table level. Hardware must assume that the entry can be from any level. In this case, TTL<1:0> is RES0.
0b01xx	The entry comes from a 4KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3.
0b10xx	The entry comes from a 16KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : Reserved. Treat as if TTL<3:2> is 0b00. 0b10 : Level 2. 0b11 : Level 3.
0b11xx	The entry comes from a 64KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3.

If an incorrect value of the TTL field is specified for the entry being invalidated by the instruction, then no entries are required by the architecture to be invalidated from the TLB.

Otherwise:

Reserved, RES0.

Bits [43:40]

Reserved, RES0.

IPA[51:48], bits [39:36]

When ARMv8.2-LPA is implemented:

Extension to IPA[47:12]. See IPA[47:12] for more details.

Otherwise:

Reserved, RES0.

IPA[47:12], bits [35:0]

Bits[47:12] of the intermediate physical address to match. For implementations with fewer than 48 bits, the upper bits of this field are RES0.

When ARMv8.2-LPA is implemented, and 52-bit addresses and a 64KB translation granule are in use, IPA[51:48] form the upper part of the address value. Otherwise, IPA[51:48] are RES0.

Executing the TLBI IPAS2LE1IS instruction

Accesses to this instruction use the following encodings:

```
TLBI IPAS2LE1IS{, <Xt>}
```

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b01	0b100	0b1000	0b0000	0b101
~~0b01~~	~~0b1000~~	~~0b100~~	~~0b101~~	~~0b0000~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    TLBI_IPAS2LE1IS(X[t]);
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        //no operation
    else
        TLBI_IPAS2LE1IS(X[t]);
```

27~~13~~/03~~12~~/2019~~2018~~ 21~~16~~:59~~42~~; e5e4db499bf9867a4b93324c4dbac985d3da93766379d01e197f1d40720d32d0f84c419e9187e009

(old) htmldiff from- (new)

TLBI IPAS2LE1OS, TLB Invalidate by Intermediate Physical Address, Stage 2, Last level, EL1, Outer Shareable

The TLBI IPAS2LE1OS characteristics are:

Purpose

If EL2 is implemented and enabled in the current Security state, invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 2 only translation table entry, from the final level of the translation table walk.
- One of the following applies:
  - SCR_EL3.NS==1 and the entry would be required to translate the specified IPA using the Non-secure EL1&0 translation regime.
  - SCR_EL3.NS==0 and the entry would be required to translate the specified IPA using the Secure EL1&0 translation regime.
- The entry would be used with the current VMID.

The invalidation is not required to apply to caching structures that combine stage 1 and stage 2 translation table entries.

The invalidation ~~only~~ applies to all PEs in the same Outer Shareable shareability domain as the PE that executes this System instruction.

For more information about the architectural requirements for this System instruction see 'Invalidation of TLB entries from stage 2 translations' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

Configuration

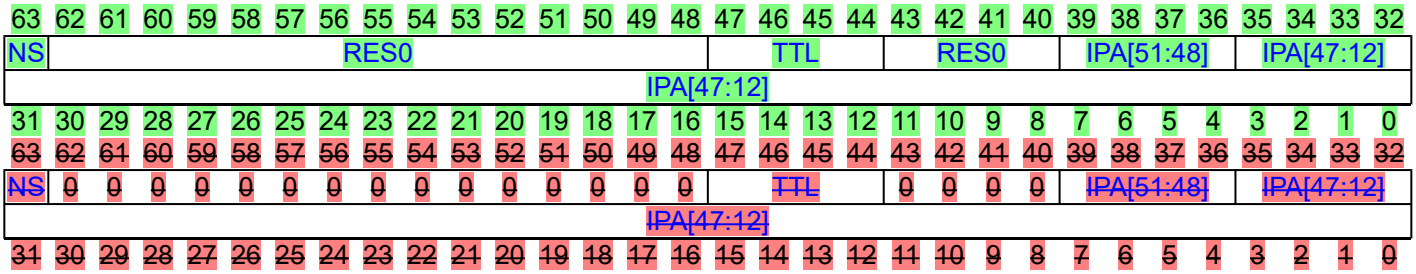
This instruction is present only when ARMv8.4-TLBI is implemented. Otherwise, direct accesses to TLBI IPAS2LE1OS are UNDEFINED.

Attributes

TLBI IPAS2LE1OS is a 64-bit System instruction.

Field descriptions

The TLBI IPAS2LE1OS input value bit assignments are:



NS, bit [63]

When ARMv8.4-SecEL2 is implemented:

Not Secure. Specifies the IPA space.

NS	Meaning
0b0	IPA is in the Secure IPA space.
0b1	IPA is in the Non-secure IPA space.

When the instruction is executed in Non-secure state, this field is RES0, and the instruction applies only to the Non-secure IPA space.

When ARMv8.4-SecEL2 is not implemented or is disabled in the current Security state, this field is RES0.

Otherwise:

Reserved, RES0.

Bits [62:48]

Reserved, RES0.

TTL, bits [47:44]

When ARMv8.4-TTL is implemented:

Translation Table Level. Indicates the level of the page table walk that holds the leaf entry for the address being invalidated.

TTL	Meaning
0b00xx	No information supplied as to the translation table level. Hardware must assume that the entry can be from any level. In this case, TTL<1:0> is RES0.
0b01xx	The entry comes from a 4KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3.
0b10xx	The entry comes from a 16KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : Reserved. Treat as if TTL<3:2> is 0b00. 0b10 : Level 2. 0b11 : Level 3.
0b11xx	The entry comes from a 64KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3.

If an incorrect value of the TTL field is specified for the entry being invalidated by the instruction then no entries are required by the architecture to be invalidated from the TLB.

Otherwise:

Reserved, RES0.

Bits [43:40]

Reserved, RES0.

IPA[51:48], bits [39:36]

Extension to IPA[47:12]. See IPA[47:12] for more details.

IPA[47:12], bits [35:0]

Bits[47:12] of the intermediate physical address to match. For implementations with fewer than 48 bits, the upper bits of this field are RES0.

When ARMv8.2-LPA is implemented, and 52-bit addresses and a 64KB translation granule are in use, IPA[51:48] form the upper part of the address value. Otherwise, IPA[51:48] are RES0.

Executing the TLBI IPAS2LE1OS instruction

Accesses to this instruction use the following encodings:

```
TLBI IPAS2LE1OS{, <Xt>}
```

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b01	0b100	0b1000	0b0100	0b100
~~0b01~~	~~0b1000~~	~~0b100~~	~~0b100~~	~~0b0100~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    TLBI_IPAS2LE1OS(X[t]);
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        //no operation
    else
        TLBI_IPAS2LE1OS(X[t]);
```

27~~13~~/03~~12~~/2019~~2018~~ 21~~16~~:59~~42~~; e5e4db499bf9867a4b93324c4dbac985d3da93766379d01e197f1d40720d32d0f84c419e9187e009

TLBI RIPAS2E1, TLB Range Invalidate by Intermediate Physical Address, Stage 2, EL1

The TLBI RIPAS2E1 characteristics are:

Purpose

If EL2 is implemented and enabled in the current Security state, invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 2 only translation table entry, from any level of the translation table walk.
- One of the following applies:
  - SCR_EL3.NS==1 and the entry would be required to translate the specified IPA using the Non-secure EL1&0 translation regime.
  - SCR_EL3.NS==0 and the entry would be required to translate the specified IPA using the Secure EL1&0 translation regime.
- The entry would be used with the current VMID.
- The entry is within the address range determined by the formula [BaseADDR <= VA < BaseADDR + ((NUM +1)*2^(5*SCALE +1) * Translation_Granule_Size)].

The invalidation is not required to apply to caching structures that combine stage 1 and stage 2 translation table entries.

The invalidation ~~only~~ applies to the PE that executes this System instruction.

The range of addresses invalidated is UNPREDICTABLE when:

- For the 4K translation granule:
  - If TTL==01 and BaseADDR[29:12] is not equal to 000000000000000000.
  - If TTL==10 and BaseADDR[20:12] is not equal to 000000000.
- For the 16K translation granule:
  - If TTL==10 and BaseADDR[24:14] is not equal to 00000000000.
- For the 64K translation granule:
  - If TTL==01 and BaseADDR[41:16] is not equal to 000000000000000000000000000.
  - If TTL==10 and BaseADDR[28:16] is not equal to 0000000000000.

For more information about the architectural requirements for this System instruction see 'Invalidation of TLB entries from stage 2 translations' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

Configuration

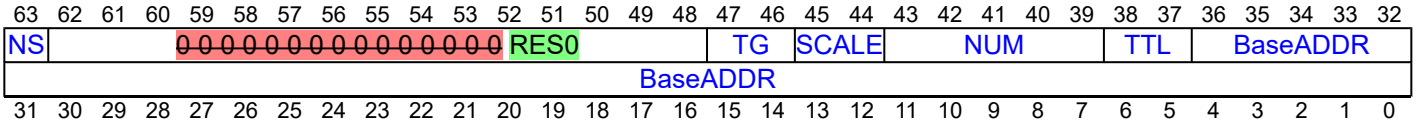
This instruction is present only when ARMv8.4-TLBI is implemented. Otherwise, direct accesses to TLBI RIPAS2E1 are UNDEFINED.

Attributes

TLBI RIPAS2E1 is a 64-bit System instruction.

Field descriptions

The TLBI RIPAS2E1 input value bit assignments are:



NS, bit [63]

When ARMv8.4-SecEL2 is implemented:

Not Secure. Specifies the IPA space.

NS	Meaning
0b0	IPA is in the Secure IPA space.
0b1	IPA is in the Non-secure IPA space.

When the instruction is executed in Non-secure state, this field is RES0, and the instruction applies only to the Non-secure IPA space.

When ARMv8.4-SecEL2 is not implemented or is disabled in the current Security state, this field is RES0.

Otherwise:

Reserved, RES0.

Bits [62:48]

Reserved, RES0.

TG, bits [47:46]

Translation granule size.

TG	Meaning
0b00	Reserved.
0b01	4K translation granule.
0b10	16K translation granule.
0b11	64K translation granule.

The instruction takes a translation granule size for the translations that are being invalidated. If the translations used a different translation granule size than the one being specified, then the architecture does not require that the instruction invalidates any entries.

SCALE, bits [45:44]

The exponent element of the calculation that is used to produce the upper range.

NUM, bits [43:39]

The base element of the calculation that is used to produce the upper range.

TTL, bits [38:37]

TTL Level hint. The TTL hint is only guaranteed to invalidate entries in the range that match the level described by the TTL hint.

TTL	Meaning
0b00	The entries in the range can be using any level for the translation table entries.
0b01	When using a 4KB or 64KB translation granule, all entries to invalidate are Level 1 translation table entries. When using a 16KB translation granule, this value is reserved. Hardware should treat this field as 0b00.
0b10	All entries to invalidate are Level 2 translation table entries.
0b11	All entries to invalidate are Level 3 translation table entries.

BaseADDR, bits [36:0]

The starting address for the range of the maintenance instruction.

When using a 4KB translation granule, this field is BaseADDR[48:12].

When using a 16KB translation granule, this field is BaseADDR[50:14].

When using a 64KB translation granule, this field is BaseADDR[52:16].

Executing the TLBI RIPAS2E1 instruction

Accesses to this instruction use the following encodings:

```
TLBI RIPAS2E1{, <Xt>}
```

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b01	0b100	0b1000	0b0100	0b010
~~0b01~~	~~0b1000~~	~~0b100~~	~~0b010~~	~~0b0100~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    TLBI_RIPAS2E1(X[t]);
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        //no operation
    else
        TLBI_RIPAS2E1(X[t]);
```

27~~13~~/03~~12~~/2019~~2018~~ 21~~16~~:59~~42~~: e5e4db499bf9867a4b93324c4dbac985d3da93766379d01e197f1d40720d32d0f84c419e9187e009

(old) htmldiff from- (new)

TLBI RIPAS2E1IS, TLB Range Invalidate by Intermediate Physical Address, Stage 2, EL1, Inner Shareable

The TLBI RIPAS2E1IS characteristics are:

Purpose

If EL2 is implemented and enabled in the current Security state, invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 2 only translation table entry, from any level of the translation table walk.
- One of the following applies:
  - SCR_EL3.NS==1 and the entry would be required to translate the specified IPA using the Non-secure EL1&0 translation regime.
  - SCR_EL3.NS==0 and the entry would be required to translate the specified IPA using the Secure EL1&0 translation regime.
- The entry would be used with the current VMID.
- The entry is within the address range determined by the formula [BaseADDR <= VA < BaseADDR + ((NUM +1)*2^(5*SCALE +1) * Translation_Granule_Size)].

The invalidation is not required to apply to caching structures that combine stage 1 and stage 2 translation table entries.

The invalidation ~~only~~ applies to all PEs in the same Inner Shareable shareability domain as the PE that executes this System instruction.

The range of addresses invalidated is UNPREDICTABLE when:

- For the 4K translation granule:
  - If TTL==01 and BaseADDR[29:12] is not equal to 000000000000000000.
  - If TTL==10 and BaseADDR[20:12] is not equal to 000000000.
- For the 16K translation granule:
  - If TTL==10 and BaseADDR[24:14] is not equal to 00000000000.
- For the 64K translation granule:
  - If TTL==01 and BaseADDR[41:16] is not equal to 000000000000000000000000000.
  - If TTL==10 and BaseADDR[28:16] is not equal to 0000000000000.

For more information about the architectural requirements for this System instruction see 'Invalidation of TLB entries from stage 2 translations' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

Configuration

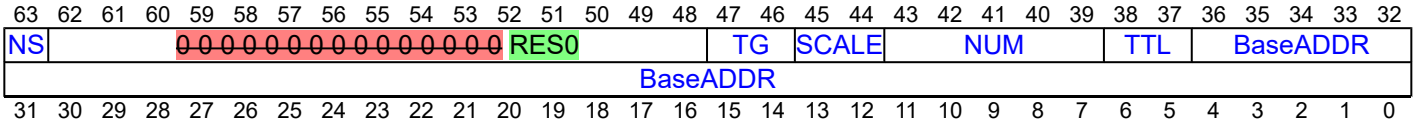
This instruction is present only when ARMv8.4-TLBI is implemented. Otherwise, direct accesses to TLBI RIPAS2E1IS are UNDEFINED.

Attributes

TLBI RIPAS2E1IS is a 64-bit System instruction.

Field descriptions

The TLBI RIPAS2E1IS input value bit assignments are:



NS, bit [63]

When ARMv8.4-SecEL2 is implemented:

Not Secure. Specifies the IPA space.

NS	Meaning
0b0	IPA is in the Secure IPA space.
0b1	IPA is in the Non-secure IPA space.

When the instruction is executed in Non-secure state, this field is RES0, and the instruction applies only to the Non-secure IPA space.

When ARMv8.4-SecEL2 is not implemented or is disabled in the current Security state, this field is RES0.

Otherwise:

Reserved, RES0.

Bits [62:48]

Reserved, RES0.

TG, bits [47:46]

Translation granule size.

TG	Meaning
0b00	Reserved.
0b01	4K translation granule.
0b10	16K translation granule.
0b11	64K translation granule.

The instruction takes a translation granule size for the translations that are being invalidated. If the translations used a different translation granule size than the one being specified, then the architecture does not require that the instruction invalidates any entries.

SCALE, bits [45:44]

The exponent element of the calculation that is used to produce the upper range.

NUM, bits [43:39]

The base element of the calculation that is used to produce the upper range.

TTL, bits [38:37]

TTL Level hint. The TTL hint is only guaranteed to invalidate entries in the range that match the level described by the TTL hint.

TTL	Meaning
0b00	The entries in the range can be using any level for the translation table entries.
0b01	When using a 4KB or 64KB translation granule, all entries to invalidate are Level 1 translation table entries. When using a 16KB translation granule, this value is reserved. Hardware should treat this field as 0b00.
0b10	All entries to invalidate are Level 2 translation table entries.
0b11	All entries to invalidate are Level 3 translation table entries.

BaseADDR, bits [36:0]

The starting address for the range of the maintenance instruction.

When using a 4KB translation granule, this field is BaseADDR[48:12].

When using a 16KB translation granule, this field is BaseADDR[50:14].

When using a 64KB translation granule, this field is BaseADDR[52:16].

Executing the TLBI RIPAS2E1IS instruction

Accesses to this instruction use the following encodings:

```
TLBI RIPAS2E1IS{, <Xt>}
```

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b01	0b100	0b1000	0b0000	0b010
~~0b01~~	~~0b1000~~	~~0b100~~	~~0b010~~	~~0b0000~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    TLBI_RIPAS2E1IS(X[t]);
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        //no operation
    else
        TLBI_RIPAS2E1IS(X[t]);
```

27~~13~~/03~~12~~/2019~~2018~~ 21~~16~~:59~~42~~: e5e4db499bf9867a4b93324c4dbac985d3da93766379d01e197f1d40720d32d0f84c419e9187e009

(old) htmldiff from- (new)

TLBI RIPAS2E1OS, TLB Range Invalidate by Intermediate Physical Address, Stage 2, EL1, Outer Shareable

The TLBI RIPAS2E1OS characteristics are:

Purpose

If EL2 is implemented and enabled in the current Security state, invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 2 only translation table entry, from any level of the translation table walk.
- One of the following applies:
  - SCR_EL3.NS==1 and the entry would be required to translate the specified IPA using the Non-secure EL1&0 translation regime.
  - SCR_EL3.NS==0 and the entry would be required to translate the specified IPA using the Secure EL1&0 translation regime.
- The entry would be used with the current VMID.
- The entry is within the address range determined by the formula [BaseADDR <= VA < BaseADDR + ((NUM +1)*2^(5*SCALE +1) * Translation_Granule_Size)].

The invalidation is not required to apply to caching structures that combine stage 1 and stage 2 translation table entries.

The invalidation ~~only~~ applies to all PEs in the same Outer Shareable shareability domain as the PE that executes this System instruction.

The range of addresses invalidated is UNPREDICTABLE when:

- For the 4K translation granule:
  - If TTL==01 and BaseADDR[29:12] is not equal to 000000000000000000.
  - If TTL==10 and BaseADDR[20:12] is not equal to 000000000.
- For the 16K translation granule:
  - If TTL==10 and BaseADDR[24:14] is not equal to 00000000000.
- For the 64K translation granule:
  - If TTL==01 and BaseADDR[41:16] is not equal to 000000000000000000000000000.
  - If TTL==10 and BaseADDR[28:16] is not equal to 0000000000000.

For more information about the architectural requirements for this System instruction see 'Invalidation of TLB entries from stage 2 translations' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

Configuration

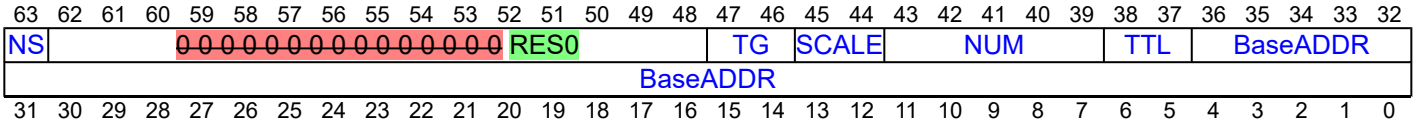
This instruction is present only when ARMv8.4-TLBI is implemented. Otherwise, direct accesses to TLBI RIPAS2E1OS are UNDEFINED.

Attributes

TLBI RIPAS2E1OS is a 64-bit System instruction.

Field descriptions

The TLBI RIPAS2E1OS input value bit assignments are:



NS, bit [63]

Not Secure. Specifies the IPA space.

NS	Meaning
0b0	IPA is in the Secure IPA space.
0b1	IPA is in the Non-secure IPA space.

When the instruction is executed in Non-secure state, this field is RES0, and the instruction applies only to the Non-secure IPA space.

When ARMv8.4-SecEL2 is not implemented or is disabled in the current Security state, this field is RES0.

Bits [62:48]

Reserved, RES0.

TG, bits [47:46]

Translation granule size.

TG	Meaning
0b00	Reserved.
0b01	4K translation granule.
0b10	16K translation granule.
0b11	64K translation granule.

The instruction takes a translation granule size for the translations that are being invalidated. If the translations used a different translation granule size than the one being specified, then the architecture does not require that the instruction invalidates any entries.

SCALE, bits [45:44]

The exponent element of the calculation that is used to produce the upper range.

NUM, bits [43:39]

The base element of the calculation that is used to produce the upper range.

TTL, bits [38:37]

TTL Level hint. The TTL hint is only guaranteed to invalidate entries in the range that match the level described by the TTL hint.

TTL	Meaning
0b00	The entries in the range can be using any level for the translation table entries.
0b01	When using a 4KB or 64KB translation granule, all entries to invalidate are Level 1 translation table entries. When using a 16KB translation granule, this value is reserved. Hardware should treat this field as 0b00.
0b10	All entries to invalidate are Level 2 translation table entries.
0b11	All entries to invalidate are Level 3 translation table entries.

BaseADDR, bits [36:0]

The starting address for the range of the maintenance instruction.

When using a 4KB translation granule, this field is BaseADDR[48:12].

When using a 16KB translation granule, this field is BaseADDR[50:14].

When using a 64KB translation granule, this field is BaseADDR[52:16].

Executing the TLBI RIPAS2E1OS instruction

Accesses to this instruction use the following encodings:

```
TLBI RIPAS2E1OS{, <Xt>}
```

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b01	0b100	0b1000	0b0100	0b011
~~0b01~~	~~0b1000~~	~~0b100~~	~~0b011~~	~~0b0100~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    TLBI_RIPAS2E1OS(X[t]);
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        //no operation
    else
        TLBI_RIPAS2E1OS(X[t]);
```

27~~13~~/03~~12~~/2019~~2018~~ 21~~16~~:59~~42~~; e5e4db499bf9867a4b93324c4dbac985d3da93766~~379d01e197f1d40720d32d0f84c419e9187c009~~

(old) htmldiff from- (new)

TLBI RIPAS2LE1, TLB Range Invalidate by Intermediate Physical Address, Stage 2, Last level, EL1

The TLBI RIPAS2LE1 characteristics are:

Purpose

If EL2 is implemented and enabled in the current Security state, invalidates~~invalidate~~ cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 2 only translation table entry, from the final level of the translation table walk.
- One of the following applies:
  - SCR_EL3.NS==1 and the entry would be required to translate the specified IPA using the Non-secure EL1&0 translation regime.
  - SCR_EL3.NS==0 and the entry would be required to translate the specified IPA using the Secure EL1&0 translation regime.
- The entry would be used with the current VMID.
- The entry is within the address range determined by the formula [BaseADDR <= VA < BaseADDR + ((NUM +1)*2^(5*SCALE +1) * Translation_Granule_Size)].

The invalidation is not required to apply to caching structures that combine stage 1 and stage 2 translation table entries.

The invalidation only applies to the PE that executes this System instruction.

The range of addresses invalidated is UNPREDICTABLE when:

- For the 4K translation granule:
  - If TTL==01 and BaseADDR[29:12] is not equal to 000000000000000000.
  - If TTL==10 and BaseADDR[20:12] is not equal to 000000000.
- For the 16K translation granule:
  - If TTL==10 and BaseADDR[24:14] is not equal to 00000000000.
- For the 64K translation granule:
  - If TTL==01 and BaseADDR[41:16] is not equal to 000000000000000000000000000.
  - If TTL==10 and BaseADDR[28:16] is not equal to 0000000000000.

For more information about the architectural requirements for this System instruction see 'Invalidation of TLB entries from stage 2 translations' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

Configuration

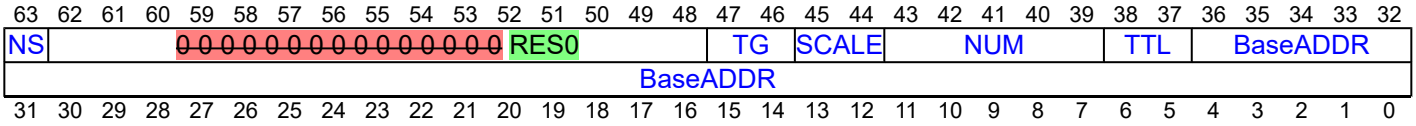
This instruction is present only when ARMv8.4-TLBI is implemented. Otherwise, direct accesses to TLBI RIPAS2LE1 are UNDEFINED.

Attributes

TLBI RIPAS2LE1 is a 64-bit System instruction.

Field descriptions

The TLBI RIPAS2LE1 input value bit assignments are:



NS, bit [63]

When ARMv8.4-SecEL2 is implemented:

Not Secure. Specifies the IPA space.

NS	Meaning
0b0	IPA is in the Secure IPA space.
0b1	IPA is in the Non-secure IPA space.

When the instruction is executed in Non-secure state, this field is RES0, and the instruction applies only to the Non-secure IPA space.

When ARMv8.4-SecEL2 is not implemented or is disabled in the current Security state, this field is RES0.

Otherwise:

Reserved, RES0.

Bits [62:48]

Reserved, RES0.

TG, bits [47:46]

Translation granule size.

TG	Meaning
0b00	Reserved.
0b01	4K translation granule.
0b10	16K translation granule.
0b11	64K translation granule.

The instruction takes a translation granule size for the translations that are being invalidated. If the translations used a different translation granule size than the one being specified, then the architecture does not require that the instruction invalidates any entries.

SCALE, bits [45:44]

The exponent element of the calculation that is used to produce the upper range.

NUM, bits [43:39]

The base element of the calculation that is used to produce the upper range.

TTL, bits [38:37]

TTL Level hint. The TTL hint is only guaranteed to invalidate entries in the range that match the level described by the TTL hint.

TTL	Meaning
0b00	The entries in the range can be using any level for the translation table entries.
0b01	When using a 4KB or 64KB translation granule, all entries to invalidate are Level 1 translation table entries. When using a 16KB translation granule, this value is reserved. Hardware should treat this field as 0b00.
0b10	All entries to invalidate are Level 2 translation table entries.
0b11	All entries to invalidate are Level 3 translation table entries.

BaseADDR, bits [36:0]

The starting address for the range of the maintenance instruction.

When using a 4KB translation granule, this field is BaseADDR[48:12].

When using a 16KB translation granule, this field is BaseADDR[50:14].

When using a 64KB translation granule, this field is BaseADDR[52:16].

Executing the TLBI RIPAS2LE1 instruction

Accesses to this instruction use the following encodings:

```
TLBI RIPAS2LE1{, <Xt>}
```

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b01	0b100	0b1000	0b0100	0b110
~~0b01~~	~~0b1000~~	~~0b100~~	~~0b110~~	~~0b0100~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    TLBI_RIPAS2LE1(X[t]);
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        //no operation
    else
        TLBI_RIPAS2LE1(X[t]);
```

27~~13~~/03~~12~~/2019~~2018~~ 21~~16~~:59~~42~~: e5e4db499bf9867a4b93324c4dbac985d3da93766~~379d01e197f1d40720d32d0f84e419e9187e009~~

(old) htmldiff from- (new)

TLBI RIPAS2LE1IS, TLB Range Invalidate by Intermediate Physical Address, Stage 2, Last level, EL1, Inner Shareable

The TLBI RIPAS2LE1IS characteristics are:

Purpose

If EL2 is implemented and enabled in the current Security state, invalidates~~invalidate~~ cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 2 only translation table entry, from the final level of the translation table walk.
- One of the following applies:
  - SCR_EL3.NS==1 and the entry would be required to translate the specified IPA using the Non-secure EL1&0 translation regime.
  - SCR_EL3.NS==0 and the entry would be required to translate the specified IPA using the Secure EL1&0 translation regime.
- The entry would be used with the current VMID.
- The entry is within the address range determined by the formula [BaseADDR <= VA < BaseADDR + ((NUM +1)*2^(5*SCALE +1) * Translation_Granule_Size)].

The invalidation is not required to apply to caching structures that combine stage 1 and stage 2 translation table entries.

The invalidation ~~only~~ applies to all PEs in the same Inner Shareable shareability domain as the PE that executes this System instruction.

The range of addresses invalidated is UNPREDICTABLE when:

- For the 4K translation granule:
  - If TTL==01 and BaseADDR[29:12] is not equal to 000000000000000000.
  - If TTL==10 and BaseADDR[20:12] is not equal to 000000000.
- For the 16K translation granule:
  - If TTL==10 and BaseADDR[24:14] is not equal to 00000000000.
- For the 64K translation granule:
  - If TTL==01 and BaseADDR[41:16] is not equal to 000000000000000000000000000.
  - If TTL==10 and BaseADDR[28:16] is not equal to 0000000000000.

For more information about the architectural requirements for this System instruction see 'Invalidation of TLB entries from stage 2 translations' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

Configuration

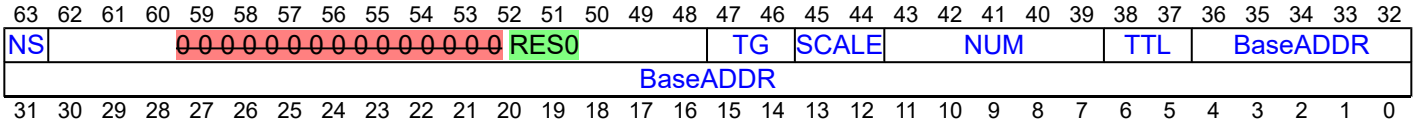
This instruction is present only when ARMv8.4-TLBI is implemented. Otherwise, direct accesses to TLBI RIPAS2LE1IS are UNDEFINED.

Attributes

TLBI RIPAS2LE1IS is a 64-bit System instruction.

Field descriptions

The TLBI RIPAS2LE1IS input value bit assignments are:



NS, bit [63]

When ARMv8.4-SecEL2 is implemented:

Not Secure. Specifies the IPA space.

NS	Meaning
0b0	IPA is in the Secure IPA space.
0b1	IPA is in the Non-secure IPA space.

When the instruction is executed in Non-secure state, this field is RES0, and the instruction applies only to the Non-secure IPA space.

When ARMv8.4-SecEL2 is not implemented or is disabled in the current Security state, this field is RES0.

Otherwise:

Reserved, RES0.

Bits [62:48]

Reserved, RES0.

TG, bits [47:46]

Translation granule size.

TG	Meaning
0b00	Reserved.
0b01	4K translation granule.
0b10	16K translation granule.
0b11	64K translation granule.

The instruction takes a translation granule size for the translations that are being invalidated. If the translations used a different translation granule size than the one being specified, then the architecture does not require that the instruction invalidates any entries.

SCALE, bits [45:44]

The exponent element of the calculation that is used to produce the upper range.

NUM, bits [43:39]

The base element of the calculation that is used to produce the upper range.

TTL, bits [38:37]

TTL Level hint. The TTL hint is only guaranteed to invalidate entries in the range that match the level described by the TTL hint.

TTL	Meaning
0b00	The entries in the range can be using any level for the translation table entries.
0b01	When using a 4KB or 64KB translation granule, all entries to invalidate are Level 1 translation table entries. When using a 16KB translation granule, this value is reserved. Hardware should treat this field as 0b00.
0b10	All entries to invalidate are Level 2 translation table entries.
0b11	All entries to invalidate are Level 3 translation table entries.

BaseADDR, bits [36:0]

The starting address for the range of the maintenance instruction.

When using a 4KB translation granule, this field is BaseADDR[48:12].

When using a 16KB translation granule, this field is BaseADDR[50:14].

When using a 64KB translation granule, this field is BaseADDR[52:16].

Executing the TLBI RIPAS2LE1IS instruction

Accesses to this instruction use the following encodings:

```
TLBI RIPAS2LE1IS{, <Xt>}
```

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b01	0b100	0b1000	0b0000	0b110
~~0b01~~	~~0b1000~~	~~0b100~~	~~0b110~~	~~0b0000~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    TLBI_RIPAS2LE1IS(X[t]);
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        //no operation
    else
        TLBI_RIPAS2LE1IS(X[t]);
```

27~~13~~/03~~12~~/2019~~2018~~ 21~~16~~:59~~42~~: e5e4db499bf9867a4b93324c4dbac985d3da93766379d01e197f1d40720d32d0f84e419e9187e009

(old) htmldiff from- (new)

TLBI RIPAS2LE1OS, TLB Range Invalidate by Intermediate Physical Address, Stage 2, Last level, EL1, Outer Shareable

The TLBI RIPAS2LE1OS characteristics are:

Purpose

If EL2 is implemented and enabled in the current Security state, invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 2 only translation table entry, from any level of the translation table walk.
- One of the following applies:
  - SCR_EL3.NS==1 and the entry would be required to translate the specified IPA using the Non-secure EL1&0 translation regime.
  - SCR_EL3.NS==0 and the entry would be required to translate the specified IPA using the Secure EL1&0 translation regime.
- The entry would be used with the current VMID.
- The entry is within the address range determined by the formula [BaseADDR <= VA < BaseADDR + ((NUM +1)*2^(5*SCALE +1) * Translation_Granule_Size)].

Note

When a TLB maintenance instruction is generated to the Secure EL1&0 translation regime and is defined to pass a VMID argument, or would be defined to pass a VMID argument if SCR_EL3.EEL2==1, then:

- A PE with SCR_EL3.EEL2==1 is not architecturally required to invalidate any entries in the Secure EL1&0 translation of a PE in the same required shareability domain with SCR_EL3.EEL2==0.
- A PE with SCR_EL3.EEL2==0 is not architecturally required to invalidate any entries in the Secure EL1&0 translation of a PE in the same required shareability domain with SCR_EL3.EEL2==1.
- A PE is architecturally required to invalidate all relevant entries in the Secure EL1&0 translation of a System MMU in the same required shareability domain with a VMID of 0.

The invalidation is not required to apply to caching structures that combine stage 1 and stage 2 translation table entries.

The invalidation ~~only~~ applies to all PEs in the same Outer Shareable shareability domain as the PE that executes this System instruction.

The range of addresses invalidated is UNPREDICTABLE when:

- For the 4K translation granule:
  - If TTL==01 and BaseADDR[29:12] is not equal to 000000000000000000.
  - If TTL==10 and BaseADDR[20:12] is not equal to 000000000.
- For the 16K translation granule:
  - If TTL==10 and BaseADDR[24:14] is not equal to 00000000000.
- For the 64K translation granule:
  - If TTL==01 and BaseADDR[41:16] is not equal to 000000000000000000000000000.
  - If TTL==10 and BaseADDR[28:16] is not equal to 0000000000000.

For more information about the architectural requirements for this System instruction see 'Invalidation of TLB entries from stage 2 translations' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

Configuration

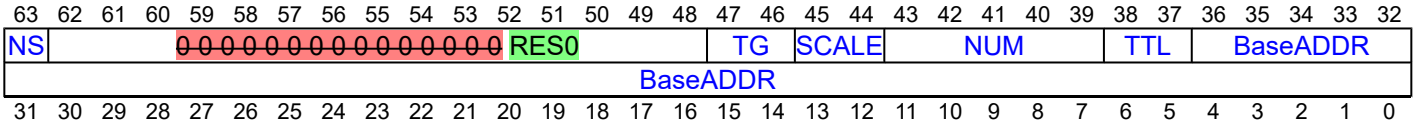
This instruction is present only when ARMv8.4-TLBI is implemented. Otherwise, direct accesses to TLBI RIPAS2LE1OS are UNDEFINED.

Attributes

TLBI RIPAS2LE1OS is a 64-bit System instruction.

Field descriptions

The TLBI RIPAS2LE1OS input value bit assignments are:



NS, bit [63]

When ARMv8.4-SecEL2 is implemented:

Not Secure. Specifies the IPA space.

NS	Meaning
0b0	IPA is in the Secure IPA space.
0b1	IPA is in the Non-secure IPA space.

When the instruction is executed in Non-secure state, this field is RES0, and the instruction applies only to the Non-secure IPA space.

When ARMv8.4-SecEL2 is not implemented or is disabled in the current Security state, this field is RES0.

Otherwise:

Reserved, RES0.

Bits [62:48]

Reserved, RES0.

TG, bits [47:46]

Translation granule size.

TG	Meaning
0b00	Reserved.
0b01	4K translation granule.
0b10	16K translation granule.
0b11	64K translation granule.

The instruction takes a translation granule size for the translations that are being invalidated. If the translations used a different translation granule size than the one being specified, then the architecture does not require that the instruction invalidates any entries.

SCALE, bits [45:44]

The exponent element of the calculation that is used to produce the upper range.

NUM, bits [43:39]

The base element of the calculation that is used to produce the upper range.

TTL, bits [38:37]

TTL Level hint. The TTL hint is only guaranteed to invalidate entries in the range that match the level described by the TTL hint.

TTL	Meaning
0b00	The entries in the range can be using any level for the translation table entries.
0b01	When using a 4KB or 64KB translation granule, all entries to invalidate are Level 1 translation table entries. When using a 16KB translation granule, this value is reserved. Hardware should treat this field as 0b00.
0b10	All entries to invalidate are Level 2 translation table entries.
0b11	All entries to invalidate are Level 3 translation table entries.

BaseADDR, bits [36:0]

The starting address for the range of the maintenance instruction.

When using a 4KB translation granule, this field is BaseADDR[48:12].

When using a 16KB translation granule, this field is BaseADDR[50:14].

When using a 64KB translation granule, this field is BaseADDR[52:16].

Executing the TLBI RIPAS2LE1OS instruction

Accesses to this instruction use the following encodings:

```
TLBI RIPAS2LE1OS{, <Xt>}
```

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b01	0b100	0b1000	0b0100	0b111
~~0b01~~	~~0b1000~~	~~0b100~~	~~0b111~~	~~0b0100~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    TLBI_RIPAS2LE1OS(X[t]);
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        //no operation
    else
        TLBI_RIPAS2LE1OS(X[t]);
```

27~~13~~/03~~12~~/2019~~2018~~ 21~~16~~:59~~42~~; e5e4db499bf9867a4b93324c4dbac985d3da937666~~379d01e197f1d40720d32d0f84e419e9187e009~~

(old) htmldiff from- (new)

TLBI RVAAE1, TLB Range Invalidate by VA, All ASID, EL1

The TLBI RVAAE1 characteristics are:

Purpose

Invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 1 translation table entry, from any level of the translation table walk.
- When EL2 is implemented and enabled in the ~~current~~ Security state described by the current value of~~:~~ SCR_EL3.NS:
  - If HCR_EL2.{E2H, TGE} is not {1, 1}, the entry would be used with the current VMID and would be required to translate the specified VA using the EL1&0 translation regime.
  - If HCR_EL2.{E2H, TGE} is {1, 1}, the entry would be required to translate the specified VA using the EL2&0 translation regime.
- When EL2 is not implemented or is disabled in the current Security state, the entry would be required to translate the specified VA using the EL1&0 translation regime.
- The entry is within the address range determined by the formula [BaseADDR <= VA < BaseADDR + ((NUM +1)*2^(5*SCALE +1) * Translation_Granule_Size)].

The invalidation ~~only~~ applies to the PE that executes this System instruction.

Note

For the EL1&0 and EL2&0 translation regimes, the invalidation applies to both:

- Global entries.
- Non-global entries with any ASID.

The range of addresses invalidated is UNPREDICTABLE when:

- For the 4K translation granule:
  - If TTL==01 and BaseADDR[29:12] is not equal to 000000000000000000.
  - If TTL==10 and BaseADDR[20:12] is not equal to 000000000.
- For the 16K translation granule:
  - If TTL==10 and BaseADDR[24:14] is not equal to 00000000000.
- For the 64K translation granule:
  - If TTL==01 and BaseADDR[41:16] is not equal to 000000000000000000000000000.
  - If TTL==10 and BaseADDR[28:16] is not equal to 0000000000000.

~~For more information about the architectural requirements for this instruction see 'Invalidation of TLB entries from stage 2 translations' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.~~

Configuration

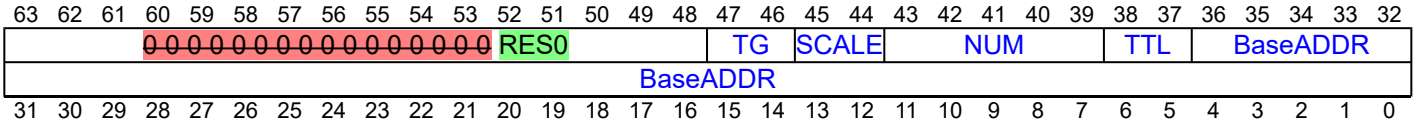
This instruction is present only when ARMv8.4-TLBI is implemented. Otherwise, direct accesses to TLBI RVAAE1 are UNDEFINED.

Attributes

TLBI RVAAE1 is a 64-bit System instruction.

Field descriptions

The TLBI RVAAE1 input value bit assignments are:



Bits [63:48]

Reserved, RES0.

TG, bits [47:46]

Translation granule size.

TG	Meaning
0b00	Reserved.
0b01	4K translation granule.
0b10	16K translation granule.
0b11	64K translation granule.

The instruction takes a translation granule size for the translations that are being invalidated. If the translations used a different translation granule size than the one being specified, then the architecture does not require that the instruction invalidates any entries.

SCALE, bits [45:44]

The exponent element of the calculation that is used to produce the upper range.

NUM, bits [43:39]

The base element of the calculation that is used to produce the upper range.

TTL, bits [38:37]

TTL Level hint. The TTL hint is only guaranteed to invalidate entries in the range that match the level described by the TTL hint.

TTL	Meaning
0b00	The entries in the range can be using any level for the translation table entries.
0b01	When using a 4KB or 64KB translation granule, all entries to invalidate are Level 1 translation table entries. When using a 16KB translation granule, this value is reserved and hardware should treat this field as 0b00.
0b10	All entries to invalidate are Level 2 translation table entries.
0b11	All entries to invalidate are Level 3 translation table entries.

BaseADDR, bits [36:0]

The starting address for the range of the maintenance instruction.

When using a 4KB translation granule, this field is BaseADDR[48:12].

When using a 16KB translation granule, this field is BaseADDR[50:14].

When using a 64KB translation granule, this field is BaseADDR[52:16].

Executing the TLBI RVAAE1 instruction

Accesses to this instruction use the following encodings:

```
TLBI RVAAE1{, <Xt>}
```

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b01	0b000	0b1000	0b0110	0b011
~~0b01~~	~~0b1000~~	~~0b000~~	~~0b011~~	~~0b0110~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TTLB == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.FB == '1' then
        TLBI_RVAAE1IS(X[t]);
    else
        TLBI_RVAAE1(X[t]);
elsif PSTATE.EL == EL2 then
    TLBI_RVAAE1(X[t]);
elsif PSTATE.EL == EL3 then
    TLBI_RVAAE1(X[t]);
```

27~~13~~/03~~12~~/2019~~2018~~ 21~~16~~:59~~42~~: e5e4db499bf9867a4b93324c4dbac985d3da93766379d01e197f1d40720d32d0f84e419e9187e009

(old) htmldiff from- (new)

TLBI RVAAE1IS, TLB Range Invalidate by VA, All ASID, EL1, Inner Shareable

The TLBI RVAAE1IS characteristics are:

Purpose

Invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 1 translation table entry, from any level of the translation table walk.
- When EL2 is implemented and enabled in the ~~current~~ Security state described by the current value of~~:~~ SCR_EL3.NS:
  - If HCR_EL2.{E2H, TGE} is not {1, 1}, the entry would be used with the current VMID and would be required to translate the specified VA using the EL1&0 translation regime.
  - If HCR_EL2.{E2H, TGE} is {1, 1}, the entry would be required to translate the specified VA using the EL2&0 translation regime.
- When EL2 is not implemented or is disabled in the current Security state, the entry would be required to translate the specified VA using the EL1&0 translation regime.
- The entry is within the address range determined by the formula [BaseADDR <= VA < BaseADDR + ((NUM +1)*2^(5*SCALE +1) * Translation_Granule_Size)].

The invalidation applies to all PEs in the same Inner Shareable shareability domain as the PE that executes this System instruction.~~instructions.~~

Note

When a TLB maintenance instruction is generated to the Secure EL1&0 translation regime and is defined to pass a VMID argument, or would be defined to pass a VMID argument if SCR_EL3.EEL2==1, then:

- A PE with SCR_EL3.EEL2==1 is not architecturally required to invalidate any entries in the Secure EL1&0 translation of a PE in the same required shareability domain with SCR_EL3.EEL2==0.
- A PE with SCR_EL3.EEL2==0 is not architecturally required to invalidate any entries in the Secure EL1&0 translation of a PE in the same required shareability domain with SCR_EL3.EEL2==1.
- A PE is architecturally required to invalidate all relevant entries in the Secure EL1&0 translation of a System MMU in the same required shareability domain with a VMID of 0.

For the EL1&0 and EL2&0 translation regimes, the invalidation applies to both:

- Global entries.
- Non-global entries with any ASID.

The range of addresses invalidated is UNPREDICTABLE when:

- For the 4K translation granule:
  - If TTL==01 and BaseADDR[29:12] is not equal to 000000000000000000.
  - If TTL==10 and BaseADDR[20:12] is not equal to 000000000.
- For the 16K translation granule:
  - If TTL==10 and BaseADDR[24:14] is not equal to 00000000000.
- For the 64K translation granule:
  - If TTL==01 and BaseADDR[41:16] is not equal to 000000000000000000000000.

- If TTL==10 and BaseADDR[28:16] is not equal to 0000000000000.

For more information about the architectural requirements for this instruction see 'Invalidation of TLB entries from stage 2 translations' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

Configuration

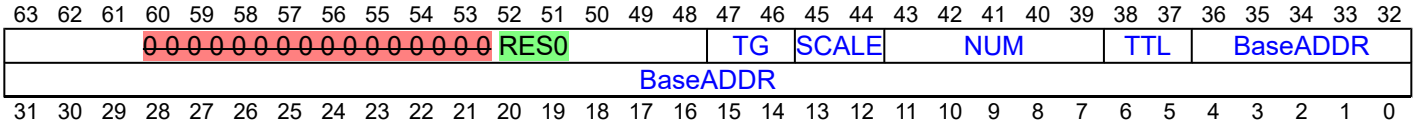
This instruction is present only when ARMv8.4-TLBI is implemented. Otherwise, direct accesses to TLBI RVAAE1IS are UNDEFINED.

Attributes

TLBI RVAAE1IS is a 64-bit System instruction.

Field descriptions

The TLBI RVAAE1IS input value bit assignments are:



Bits [63:48]

Reserved, RES0.

TG, bits [47:46]

Translation granule size.

TG	Meaning
0b00	Reserved.
0b01	4K translation granule.
0b10	16K translation granule.
0b11	64K translation granule.

The instruction takes a translation granule size for the translations that are being invalidated. If the translations used a different translation granule size than the one being specified, then the architecture does not require that the instruction invalidates any entries.

SCALE, bits [45:44]

The exponent element of the calculation that is used to produce the upper range.

NUM, bits [43:39]

The base element of the calculation that is used to produce the upper range.

TTL, bits [38:37]

TTL Level hint. The TTL hint is only guaranteed to invalidate entries in the range that match the level described by the TTL hint.

TTL	Meaning
0b00	The entries in the range can be using any level for the translation table entries.
0b01	When using a 4KB or 64KB translation granule, all entries to invalidate are Level 1 translation table entries. When using a 16KB translation granule, this value is reserved and hardware should treat this field as 0b00.
0b10	All entries to invalidate are Level 2 translation table entries.
0b11	All entries to invalidate are Level 3 translation table entries.

BaseADDR, bits [36:0]

The starting address for the range of the maintenance instruction.

When using a 4KB translation granule, this field is BaseADDR[48:12].

When using a 16KB translation granule, this field is BaseADDR[50:14].

When using a 64KB translation granule, this field is BaseADDR[52:16].

Executing the TLBI RVAAE1IS instruction

Accesses to this instruction use the following encodings:

```
TLBI RVAAE1IS{, <Xt>}
```

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b01	0b000	0b1000	0b0010	0b011
~~0b01~~	~~0b1000~~	~~0b000~~	~~0b011~~	~~0b0010~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TTLB == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TTLBIS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        TLBI_RVAAE1IS(X[t]);
elsif PSTATE.EL == EL2 then
    TLBI_RVAAE1IS(X[t]);
elsif PSTATE.EL == EL3 then
    TLBI_RVAAE1IS(X[t]);
```

27~~13~~/03~~12~~/2019~~2018~~ 21~~16~~:59~~42~~; e5e4db499bf9867a4b93324c4dbac985d3da9376~~6379d01e197f1d40720d32d0f84e419e9187e009~~

(old) htmldiff from- (new)

TLBI RVAAE1OS, TLB Range Invalidate by VA, All ASID, EL1, Outer Shareable

The TLBI RVAAE1OS characteristics are:

Purpose

Invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 1 translation table entry, from any level of the translation table walk.
- When EL2 is implemented and enabled in the ~~current~~ Security state described by the current value of~~:~~ SCR_EL3.NS:
  - If HCR_EL2.{E2H, TGE} is not {1, 1}, the entry would be used with the current VMID and would be required to translate the specified VA using the EL1&0 translation regime.
  - If HCR_EL2.{E2H, TGE} is {1, 1}, the entry would be required to translate the specified VA using the EL2&0 translation regime.
- When EL2 is not implemented or is disabled in the current Security state, the entry would be required to translate the specified VA using the EL1&0 translation regime.
- The entry is within the address range determined by the formula [BaseADDR <= VA < BaseADDR + ((NUM +1)*2^(5*SCALE +1) * Translation_Granule_Size)].

The invalidation applies to all PEs in the same Outer Shareable shareability domain as the PE that executes this System instruction.~~instructions.~~

Note

When a TLB maintenance instruction is generated to the Secure EL1&0 translation regime and is defined to pass a VMID argument, or would be defined to pass a VMID argument if SCR_EL3.EEL2==1, then:

- A PE with SCR_EL3.EEL2==1 is not architecturally required to invalidate any entries in the Secure EL1&0 translation of a PE in the same required shareability domain with SCR_EL3.EEL2==0.
- A PE with SCR_EL3.EEL2==0 is not architecturally required to invalidate any entries in the Secure EL1&0 translation of a PE in the same required shareability domain with SCR_EL3.EEL2==1.
- A PE is architecturally required to invalidate all relevant entries in the Secure EL1&0 translation of a System MMU in the same required shareability domain with a VMID of 0.

For the EL1&0 and EL2&0 translation regimes, the invalidation applies to both:

- Global entries.
- Non-global entries with any ASID.

The range of addresses invalidated is UNPREDICTABLE when:

- For the 4K translation granule:
  - If TTL==01 and BaseADDR[29:12] is not equal to 000000000000000000.
  - If TTL==10 and BaseADDR[20:12] is not equal to 000000000.
- For the 16K translation granule:
  - If TTL==10 and BaseADDR[24:14] is not equal to 00000000000.
- For the 64K translation granule:
  - If TTL==01 and BaseADDR[41:16] is not equal to 000000000000000000000000.

- If TTL==10 and BaseADDR[28:16] is not equal to 0000000000000.

For more information about the architectural requirements for this instruction see 'Invalidation of TLB entries from stage 2 translations' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

Configuration

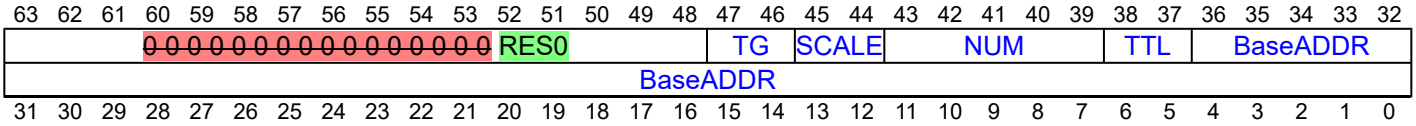
This instruction is present only when ARMv8.4-TLBI is implemented. Otherwise, direct accesses to TLBI RVAAE1OS are UNDEFINED.

Attributes

TLBI RVAAE1OS is a 64-bit System instruction.

Field descriptions

The TLBI RVAAE1OS input value bit assignments are:



Bits [63:48]

Reserved, RES0.

TG, bits [47:46]

Translation granule size.

TG	Meaning
0b00	Reserved.
0b01	4K translation granule.
0b10	16K translation granule.
0b11	64K translation granule.

The instruction takes a translation granule size for the translations that are being invalidated. If the translations used a different translation granule size than the one being specified, then the architecture does not require that the instruction invalidates any entries.

SCALE, bits [45:44]

The exponent element of the calculation that is used to produce the upper range.

NUM, bits [43:39]

The base element of the calculation that is used to produce the upper range.

TTL, bits [38:37]

TTL Level hint. The TTL hint is only guaranteed to invalidate entries in the range that match the level described by the TTL hint.

TTL	Meaning
0b00	The entries in the range can be using any level for the translation table entries.
0b01	When using a 4KB or 64KB translation granule, all entries to invalidate are Level 1 translation table entries. When using a 16KB translation granule, this value is reserved and hardware should treat this field as 0b00.
0b10	All entries to invalidate are Level 2 translation table entries.
0b11	All entries to invalidate are Level 3 translation table entries.

BaseADDR, bits [36:0]

The starting address for the range of the maintenance instruction.

When using a 4KB translation granule, this field is BaseADDR[48:12].

When using a 16KB translation granule, this field is BaseADDR[50:14].

When using a 64KB translation granule, this field is BaseADDR[52:16].

Executing the TLBI RVAAE1OS instruction

Accesses to this instruction use the following encodings:

```
TLBI RVAAE1OS{, <Xt>}
```

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b01	0b000	0b1000	0b0101	0b011
~~0b01~~	~~0b1000~~	~~0b000~~	~~0b011~~	~~0b0101~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TTLB == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TTLBOS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        TLBI_RVAAE1OS(X[t]);
elsif PSTATE.EL == EL2 then
    TLBI_RVAAE1OS(X[t]);
elsif PSTATE.EL == EL3 then
    TLBI_RVAAE1OS(X[t]);
```

27~~13~~/03~~12~~/2019~~2018~~ 21~~16~~:59~~42~~; e5e4db499bf9867a4b93324c4dbac985d3da9376~~6379d01e197f1d40720d32d0f84e419e9187e009~~

(old) htmldiff from- (new)

TLBI RVAALE1, TLB Range Invalidate by VA, All ASID, Last level, EL1

The TLBI RVAALE1 characteristics are:

Purpose

Invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 1 translation table entry, from the final level of the translation table walk.
- When EL2 is implemented and enabled in the ~~current~~ Security state described by the current value of~~:~~ SCR_EL3.NS:
  - If HCR_EL2.{E2H, TGE} is not {1, 1}, the entry would be used with the current VMID and would be required to translate the specified VA using the EL1&0 translation regime.
  - If HCR_EL2.{E2H, TGE} is {1, 1}, the entry would be required to translate the specified VA using the EL2&0 translation regime.
- When EL2 is not implemented or is disabled in the current Security state, the entry would be required to translate the specified VA using the EL1&0 translation regime.
- The entry is within the address range determined by the formula [BaseADDR <= VA < BaseADDR + ((NUM +1)*2^(5*SCALE +1) * Translation_Granule_Size)].

The invalidation ~~only~~ applies to the PE that executes this System instruction.

Note

For the EL1&0 and EL2&0 translation regimes, the invalidation applies to both:

- Global entries.
- Non-global entries with any ASID.

The range of addresses invalidated is UNPREDICTABLE when:

- For the 4K translation granule:
  - If TTL==01 and BaseADDR[29:12] is not equal to 000000000000000000.
  - If TTL==10 and BaseADDR[20:12] is not equal to 000000000.
- For the 16K translation granule:
  - If TTL==10 and BaseADDR[24:14] is not equal to 00000000000.
- For the 64K translation granule:
  - If TTL==01 and BaseADDR[41:16] is not equal to 000000000000000000000000000.
  - If TTL==10 and BaseADDR[28:16] is not equal to 0000000000000.

~~For more information about the architectural requirements for this instruction see 'Invalidation of TLB entries from stage 2 translations' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.~~

Configuration

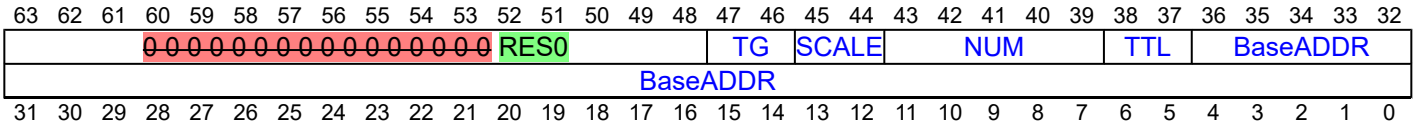
This instruction is present only when ARMv8.4-TLBI is implemented. Otherwise, direct accesses to TLBI RVAALE1 are UNDEFINED.

Attributes

TLBI RVAALE1 is a 64-bit System instruction.

Field descriptions

The TLBI RVAALE1 input value bit assignments are:



Bits [63:48]

Reserved, RES0.

TG, bits [47:46]

Translation granule size.

TG	Meaning
0b00	Reserved.
0b01	4K translation granule.
0b10	16K translation granule.
0b11	64K translation granule.

The instruction takes a translation granule size for the translations that are being invalidated. If the translations used a different translation granule size than the one being specified, then the architecture does not require that the instruction invalidates any entries.

SCALE, bits [45:44]

The exponent element of the calculation that is used to produce the upper range.

NUM, bits [43:39]

The base element of the calculation that is used to produce the upper range.

TTL, bits [38:37]

TTL Level hint. The TTL hint is only guaranteed to invalidate entries in the range that match the level described by the TTL hint.

TTL	Meaning
0b00	The entries in the range can be using any level for the translation table entries.
0b01	When using a 4KB or 64KB translation granule, all entries to invalidate are Level 1 translation table entries. When using a 16KB translation granule, this value is reserved and hardware should treat this field as 0b00.
0b10	All entries to invalidate are Level 2 translation table entries.
0b11	All entries to invalidate are Level 3 translation table entries.

BaseADDR, bits [36:0]

The starting address for the range of the maintenance instruction.

When using a 4KB translation granule, this field is BaseADDR[48:12].

When using a 16KB translation granule, this field is BaseADDR[50:14].

When using a 64KB translation granule, this field is BaseADDR[52:16].

Executing the TLBI RVAALE1 instruction

Accesses to this instruction use the following encodings:

```
TLBI RVAALE1{, <Xt>}
```

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b01	0b000	0b1000	0b0110	0b111
~~0b01~~	~~0b1000~~	~~0b000~~	~~0b111~~	~~0b0110~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TTLB == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.FB == '1' then
        TLBI_RVAAE1IS(X[t]);
    else
        TLBI_RVAALE1(X[t]);
elsif PSTATE.EL == EL2 then
    TLBI_RVAALE1(X[t]);
elsif PSTATE.EL == EL3 then
    TLBI_RVAALE1(X[t]);
```

27~~13~~/03~~12~~/2019~~2018~~ 21~~16~~:59~~42~~; e5e4db499bf9867a4b93324c4dbac985d3da9376~~6379d01e197f1d40720d32d0f84c419e9187e009~~

(old) htmldiff from- (new)

TLBI RVAALE1IS, TLB Range Invalidate by VA, All ASID, Last Level, EL1, Inner Shareable

The TLBI RVAALE1IS characteristics are:

Purpose

Invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 1 translation table entry, from the final level of the translation table walk.
- When EL2 is implemented and enabled in the ~~current~~ Security state described by the current value of~~:~~ SCR_EL3.NS:
  - If HCR_EL2.{E2H, TGE} is not {1, 1}, the entry would be used with the current VMID and would be required to translate the specified VA using the EL1&0 translation regime.
  - If HCR_EL2.{E2H, TGE} is {1, 1}, the entry would be required to translate the specified VA using the EL2&0 translation regime.
- When EL2 is not implemented or is disabled in the current Security state, the entry would be required to translate the specified VA using the EL1&0 translation regime.
- The entry is within the address range determined by the formula [BaseADDR <= VA < BaseADDR + ((NUM +1)*2^(5*SCALE +1) * Translation_Granule_Size)].

The invalidation applies to all PEs in the same Inner Shareable shareability domain as the PE that executes this System instruction.~~instructions.~~

Note

When a TLB maintenance instruction is generated to the Secure EL1&0 translation regime and is defined to pass a VMID argument, or would be defined to pass a VMID argument if SCR_EL3.EEL2==1, then:

- A PE with SCR_EL3.EEL2==1 is not architecturally required to invalidate any entries in the Secure EL1&0 translation of a PE in the same required shareability domain with SCR_EL3.EEL2==0.
- A PE with SCR_EL3.EEL2==0 is not architecturally required to invalidate any entries in the Secure EL1&0 translation of a PE in the same required shareability domain with SCR_EL3.EEL2==1.
- A PE is architecturally required to invalidate all relevant entries in the Secure EL1&0 translation of a System MMU in the same required shareability domain with a VMID of 0.

For the EL1&0 and EL2&0 translation regimes, the invalidation applies to both:

- Global entries.
- Non-global entries with any ASID.

The range of addresses invalidated is UNPREDICTABLE when:

- For the 4K translation granule:
  - If TTL==01 and BaseADDR[29:12] is not equal to 000000000000000000.
  - If TTL==10 and BaseADDR[20:12] is not equal to 000000000.
- For the 16K translation granule:
  - If TTL==10 and BaseADDR[24:14] is not equal to 00000000000.
- For the 64K translation granule:
  - If TTL==01 and BaseADDR[41:16] is not equal to 000000000000000000000000000.

- If TTL==10 and BaseADDR[28:16] is not equal to 0000000000000.

~~For more information about the architectural requirements for this instruction see 'Invalidation of TLB entries from stage 2 translations' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.~~

Configuration

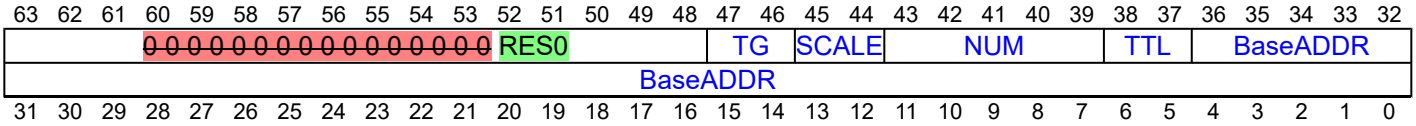
This instruction is present only when ARMv8.4-TLBI is implemented. Otherwise, direct accesses to TLBI RVAALE1IS are UNDEFINED.

Attributes

TLBI RVAALE1IS is a 64-bit System instruction.

Field descriptions

The TLBI RVAALE1IS input value bit assignments are:



Bits [63:48]

Reserved, RES0.

TG, bits [47:46]

Translation granule size.

TG	Meaning
0b00	Reserved.
0b01	4K translation granule.
0b10	16K translation granule.
0b11	64K translation granule.

The instruction takes a translation granule size for the translations that are being invalidated. If the translations used a different translation granule size than the one being specified, then the architecture does not require that the instruction invalidates any entries.

SCALE, bits [45:44]

The exponent element of the calculation that is used to produce the upper range.

NUM, bits [43:39]

The base element of the calculation that is used to produce the upper range.

TTL, bits [38:37]

TTL Level hint. The TTL hint is only guaranteed to invalidate entries in the range that match the level described by the TTL hint.

TTL	Meaning
0b00	The entries in the range can be using any level for the translation table entries.
0b01	When using a 4KB or 64KB translation granule, all entries to invalidate are Level 1 translation table entries. When using a 16KB translation granule, this value is reserved and hardware should treat this field as 0b00.
0b10	All entries to invalidate are Level 2 translation table entries.
0b11	All entries to invalidate are Level 3 translation table entries.

BaseADDR, bits [36:0]

The starting address for the range of the maintenance instruction.

When using a 4KB translation granule, this field is BaseADDR[48:12].

When using a 16KB translation granule, this field is BaseADDR[50:14].

When using a 64KB translation granule, this field is BaseADDR[52:16].

Executing the TLBI RVAALE1IS instruction

Accesses to this instruction use the following encodings:

```
TLBI RVAALE1IS{, <Xt>}
```

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b01	0b000	0b1000	0b0010	0b111
~~0b01~~	~~0b1000~~	~~0b000~~	~~0b111~~	~~0b0010~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TTLB == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TTLBIS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        TLBI_RVAALE1IS(X[t]);
elsif PSTATE.EL == EL2 then
    TLBI_RVAALE1IS(X[t]);
elsif PSTATE.EL == EL3 then
    TLBI_RVAALE1IS(X[t]);
```

27~~13~~/03~~12~~/2019~~2018~~ 21~~16~~:59~~42~~; e5e4db499bf9867a4b93324c4dbac985d3da9376~~6379d01e197f1d40720d32d0f84e419e9187e009~~

(old) htmldiff from- (new)

TLBI RVAALE1OS, TLB Range Invalidate by VA, All ASID, Last Level, EL1, Outer Shareable

The TLBI RVAALE1OS characteristics are:

Purpose

Invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 1 translation table entry, from the final level of the translation table walk.
- When EL2 is implemented and enabled in the ~~current~~ Security state described by the current value of~~:~~ SCR_EL3.NS:
  - If HCR_EL2.{E2H, TGE} is not {1, 1}, the entry would be used with the current VMID and would be required to translate the specified VA using the EL1&0 translation regime.
  - If HCR_EL2.{E2H, TGE} is {1, 1}, the entry would be required to translate the specified VA using the EL2&0 translation regime.
- When EL2 is not implemented or is disabled in the current Security state, the entry would be required to translate the specified VA using the EL1&0 translation regime.
- The entry is within the address range determined by the formula [BaseADDR <= VA < BaseADDR + ((NUM +1)*2^(5*SCALE +1) * Translation_Granule_Size)].

The invalidation applies to all PEs in the same Outer Shareable shareability domain as the PE that executes this System instruction.~~instructions.~~

Note

When a TLB maintenance instruction is generated to the Secure EL1&0 translation regime and is defined to pass a VMID argument, or would be defined to pass a VMID argument if SCR_EL3.EEL2==1, then:

- A PE with SCR_EL3.EEL2==1 is not architecturally required to invalidate any entries in the Secure EL1&0 translation of a PE in the same required shareability domain with SCR_EL3.EEL2==0.
- A PE with SCR_EL3.EEL2==0 is not architecturally required to invalidate any entries in the Secure EL1&0 translation of a PE in the same required shareability domain with SCR_EL3.EEL2==1.
- A PE is architecturally required to invalidate all relevant entries in the Secure EL1&0 translation of a System MMU in the same required shareability domain with a VMID of 0.

For the EL1&0 and EL2&0 translation regimes, the invalidation applies to both:

- Global entries.
- Non-global entries with any ASID.

The range of addresses invalidated is UNPREDICTABLE when:

- For the 4K translation granule:
  - If TTL==01 and BaseADDR[29:12] is not equal to 000000000000000000.
  - If TTL==10 and BaseADDR[20:12] is not equal to 000000000.
- For the 16K translation granule:
  - If TTL==10 and BaseADDR[24:14] is not equal to 00000000000.
- For the 64K translation granule:
  - If TTL==01 and BaseADDR[41:16] is not equal to 000000000000000000000000000.

- If TTL==10 and BaseADDR[28:16] is not equal to 0000000000000.

For more information about the architectural requirements for this instruction see 'Invalidation of TLB entries from stage 2 translations' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

Configuration

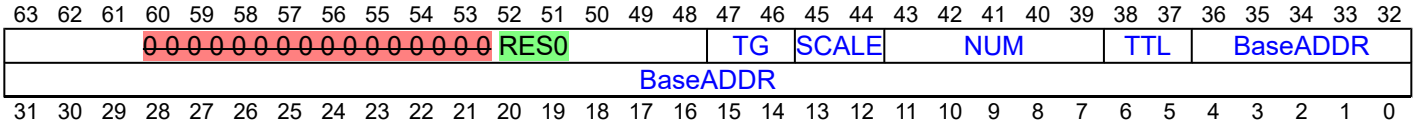
This instruction is present only when ARMv8.4-TLBI is implemented. Otherwise, direct accesses to TLBI RVAALE1OS are UNDEFINED.

Attributes

TLBI RVAALE1OS is a 64-bit System instruction.

Field descriptions

The TLBI RVAALE1OS input value bit assignments are:



Bits [63:48]

Reserved, RES0.

TG, bits [47:46]

Translation granule size.

TG	Meaning
0b00	Reserved.
0b01	4K translation granule.
0b10	16K translation granule.
0b11	64K translation granule.

The instruction takes a translation granule size for the translations that are being invalidated. If the translations used a different translation granule size than the one being specified, then the architecture does not require that the instruction invalidates any entries.

SCALE, bits [45:44]

The exponent element of the calculation that is used to produce the upper range.

NUM, bits [43:39]

The base element of the calculation that is used to produce the upper range.

TTL, bits [38:37]

TTL Level hint. The TTL hint is only guaranteed to invalidate entries in the range that match the level described by the TTL hint.

TTL	Meaning
0b00	The entries in the range can be using any level for the translation table entries.
0b01	When using a 4KB or 64KB translation granule, all entries to invalidate are Level 1 translation table entries. When using a 16KB translation granule, this value is reserved and hardware should treat this field as 0b00.
0b10	All entries to invalidate are Level 2 translation table entries.
0b11	All entries to invalidate are Level 3 translation table entries.

BaseADDR, bits [36:0]

The starting address for the range of the maintenance instruction.

When using a 4KB translation granule, this field is BaseADDR[48:12].

When using a 16KB translation granule, this field is BaseADDR[50:14].

When using a 64KB translation granule, this field is BaseADDR[52:16].

Executing the TLBI RVAALE1OS instruction

Accesses to this instruction use the following encodings:

```
TLBI RVAALE1OS{, <Xt>}
```

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b01	0b000	0b1000	0b0101	0b111
~~0b01~~	~~0b1000~~	~~0b000~~	~~0b111~~	~~0b0101~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TTLB == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TTLBOS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        TLBI_RVAALE1OS(X[t]);
elsif PSTATE.EL == EL2 then
    TLBI_RVAALE1OS(X[t]);
elsif PSTATE.EL == EL3 then
    TLBI_RVAALE1OS(X[t]);
```

27~~13~~/03~~12~~/2019~~2018~~ 21~~16~~:59~~42~~; e5e4db499bf9867a4b93324c4dbac985d3da9376~~6379d01e197f1d40720d32d0f84e419e9187e009~~

(old) htmldiff from- (new)

TLBI RVAE1, TLB Range Invalidate by VA, EL1

The TLBI RVAE1 characteristics are:

Purpose

Invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 1 translation table entry.
- The entry would be used to translate the specified VA, and one of the following applies:
  - The entry is from a level of lookup above the final level and matches the specified ASID.
  - The entry is a global entry from the final level of lookup.
  - The entry is a non-global entry from the final level of lookup that matches the specified ASID.
- When EL2 is implemented and enabled in the ~~current~~ Security state described by the current value of~~:~~ SCR_EL3.NS:
  - If HCR_EL2.{E2H, TGE} is not {1, 1}, the entry would be used with the current VMID and would be required to translate the specified VA using the EL1&0 translation regime.
  - If HCR_EL2.{E2H, TGE} is {1, 1}, the entry would be required to translate the specified VA using the EL2&0 translation regime.
- When EL2 is not implemented or is disabled in the current Security state, the entry would be required to translate the specified VA using the EL1&0 translation regime.
- The entry is within the address range determined by the formula [BaseADDR <= VA < BaseADDR + ((NUM +1)*2^(5*SCALE +1) * Translation_Granule_Size)].

The invalidation ~~only~~ applies to the PE that executes this System instruction.

The range of addresses invalidated is UNPREDICTABLE when:

- For the 4K translation granule:
  - If TTL==01 and BaseADDR[29:12] is not equal to 000000000000000000.
  - If TTL==10 and BaseADDR[20:12] is not equal to 000000000.
- For the 16K translation granule:
  - If TTL==10 and BaseADDR[24:14] is not equal to 00000000000.
- For the 64K translation granule:
  - If TTL==01 and BaseADDR[41:16] is not equal to 000000000000000000000000000.
  - If TTL==10 and BaseADDR[28:16] is not equal to 0000000000000.

~~For more information about the architectural requirements for this instruction see 'Invalidation of TLB entries from stage 2 translations' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.~~

Configuration

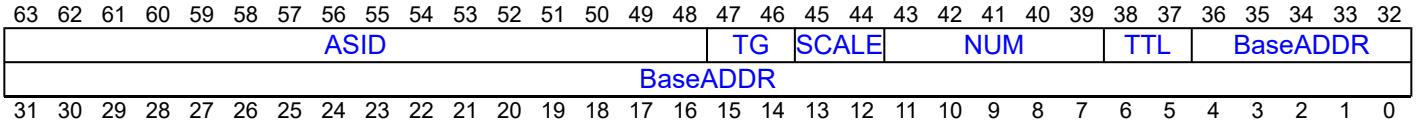
This instruction is present only when ARMv8.4-TLBI is implemented. Otherwise, direct accesses to TLBI RVAE1 are UNDEFINED.

Attributes

TLBI RVAE1 is a 64-bit System instruction.

Field descriptions

The TLBI RVAE1 input value bit assignments are:



ASID, bits [63:48]

ASID value to match. Any TLB entries that match the ASID value and VA value will be affected by this System instruction.~~operation.~~

Global TLB entries that match the VA value will be affected by this System instruction~~operation~~, regardless of the value of the ASID field.

If the implementation supports 16 bits of ASID, but only 8 bits are being used in the context being invalidated, the upper bits are RES0.

TG, bits [47:46]

Translation granule size.

TG	Meaning
0b00	Reserved.
0b01	4K translation granule.
0b10	16K translation granule.
0b11	64K translation granule.

The instruction takes a translation granule size for the translations that are being invalidated. If the translations used a different translation granule size than the one being specified, then the architecture does not require that the instruction invalidates any entries.

SCALE, bits [45:44]

The exponent element of the calculation that is used to produce the upper range.

NUM, bits [43:39]

The base element of the calculation that is used to produce the upper range.

TTL, bits [38:37]

TTL Level hint. The TTL hint is only guaranteed to invalidate entries in the range that match the level described by the TTL hint.

TTL	Meaning
0b00	The entries in the range can be using any level for the translation table entries.
0b01	When using a 4KB or 64KB translation granule, all entries to invalidate are Level 1 translation table entries. When using a 16KB translation granule, this value is reserved and hardware should treat this field as 0b00.
0b10	All entries to invalidate are Level 2 translation table entries.
0b11	All entries to invalidate are Level 3 translation table entries.

BaseADDR, bits [36:0]

The starting address for the range of the maintenance instruction.

When using a 4KB translation granule, this field is BaseADDR[48:12].

When using a 16KB translation granule, this field is BaseADDR[50:14].

When using a 64KB translation granule, this field is BaseADDR[52:16].

Executing the TLBI RVAE1 instruction

Accesses to this instruction use the following encodings:

```
TLBI RVAE1{, <Xt>}
```

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b01	0b000	0b1000	0b0110	0b001
~~0b01~~	~~0b1000~~	~~0b000~~	~~0b001~~	~~0b0110~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TTLB == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.FB == '1' then
        TLBI_RVAE1IS(X[t]);
    else
        TLBI_RVAE1(X[t]);
elsif PSTATE.EL == EL2 then
    TLBI_RVAE1(X[t]);
elsif PSTATE.EL == EL3 then
    TLBI_RVAE1(X[t]);
```

27~~13~~/03~~12~~/2019~~2018~~ 21~~16~~:59~~42~~; e5e4db499bf9867a4b93324c4dbac985d3da9376~~6379d01e197f1d40720d32d0f84c419e9187e009~~

(old) htmldiff from- (new)

TLBI RVAE1IS, TLB Range Invalidate by VA, EL1, Inner Shareable

The TLBI RVAE1IS characteristics are:

Purpose

Invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 1 translation table entry.
- The entry would be used to translate the specified VA, and one of the following applies:
  - The entry is from a level of lookup above the final level and matches the specified ASID.
  - The entry is a global entry from the final level of lookup.
  - The entry is a non-global entry from the final level of lookup that matches the specified ASID.
- When EL2 is implemented and enabled in the ~~current~~ Security state described by the current value of~~:~~ SCR_EL3.NS:
  - If HCR_EL2.{E2H, TGE} is not {1, 1}, the entry would be used with the current VMID and would be required to translate the specified VA using the EL1&0 translation regime.
  - If HCR_EL2.{E2H, TGE} is {1, 1}, the entry would be required to translate the specified VA using the EL2&0 translation regime.
- When EL2 is not implemented or is disabled in the current Security state, the entry would be required to translate the specified VA using the EL1&0 translation regime.
- The entry is within the address range determined by the formula [BaseADDR <= VA < BaseADDR + ((NUM +1)*2^(5*SCALE +1) * Translation_Granule_Size)].

The invalidation ~~only~~ applies to all PEs in the same Inner Shareable shareability domain as the PE that executes this System instruction.

Note

When a TLB maintenance instruction is generated to the Secure EL1&0 translation regime and is defined to pass a VMID argument, or would be defined to pass a VMID argument if SCR_EL3.EEL2==1, then:

- A PE with SCR_EL3.EEL2==1 is not architecturally required to invalidate any entries in the Secure EL1&0 translation of a PE in the same required shareability domain with SCR_EL3.EEL2==0.
- A PE with SCR_EL3.EEL2==0 is not architecturally required to invalidate any entries in the Secure EL1&0 translation of a PE in the same required shareability domain with SCR_EL3.EEL2==1.
- A PE is architecturally required to invalidate all relevant entries in the Secure EL1&0 translation of a System MMU in the same required shareability domain with a VMID of 0.

The range of addresses invalidated is UNPREDICTABLE when:

- For the 4K translation granule:
  - If TTL==01 and BaseADDR[29:12] is not equal to 000000000000000000.
  - If TTL==10 and BaseADDR[20:12] is not equal to 000000000.
- For the 16K translation granule:
  - If TTL==10 and BaseADDR[24:14] is not equal to 00000000000.
- For the 64K translation granule:

- If TTL==01 and BaseADDR[41:16] is not equal to 00000000000000000000000000.
- If TTL==10 and BaseADDR[28:16] is not equal to 0000000000000.

~~For more information about the architectural requirements for this instruction see 'Invalidation of TLB entries from stage 2 translations' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.~~

Configuration

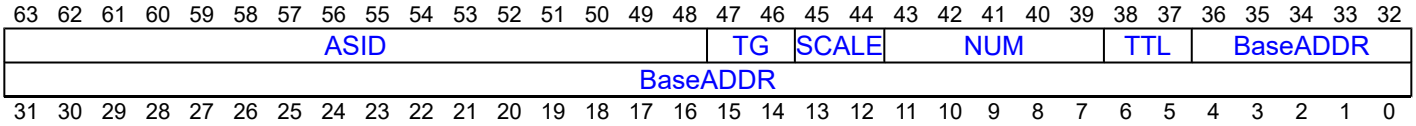
This instruction is present only when ARMv8.4-TLBI is implemented. Otherwise, direct accesses to TLBI RVAE1IS are UNDEFINED.

Attributes

TLBI RVAE1IS is a 64-bit System instruction.

Field descriptions

The TLBI RVAE1IS input value bit assignments are:



ASID, bits [63:48]

ASID value to match. Any TLB entries that match the ASID value and VA value will be affected by this System instruction.~~operation.~~

Global TLB entries that match the VA value will be affected by this System instruction~~operation~~, regardless of the value of the ASID field.

If the implementation supports 16 bits of ASID, but only 8 bits are being used in the context being invalidated, the upper bits are RES0.

TG, bits [47:46]

Translation granule size.

TG	Meaning
0b00	Reserved.
0b01	4K translation granule.
0b10	16K translation granule.
0b11	64K translation granule.

The instruction takes a translation granule size for the translations that are being invalidated. If the translations used a different translation granule size than the one being specified, then the architecture does not require that the instruction invalidates any entries.

SCALE, bits [45:44]

The exponent element of the calculation that is used to produce the upper range.

NUM, bits [43:39]

The base element of the calculation that is used to produce the upper range.

TTL, bits [38:37]

TTL Level hint. The TTL hint is only guaranteed to invalidate entries in the range that match the level described by the TTL hint.

TTL	Meaning
0b00	The entries in the range can be using any level for the translation table entries.
0b01	When using a 4KB or 64KB translation granule, all entries to invalidate are Level 1 translation table entries. When using a 16KB translation granule, this value is reserved and hardware should treat this field as 0b00.
0b10	All entries to invalidate are Level 2 translation table entries.
0b11	All entries to invalidate are Level 3 translation table entries.

BaseADDR, bits [36:0]

The starting address for the range of the maintenance instruction.

When using a 4KB translation granule, this field is BaseADDR[48:12].

When using a 16KB translation granule, this field is BaseADDR[50:14].

When using a 64KB translation granule, this field is BaseADDR[52:16].

Executing the TLBI RVAE1IS instruction

Accesses to this instruction use the following encodings:

```
TLBI RVAE1IS{, <Xt>}
```

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b01	0b000	0b1000	0b0010	0b001
~~0b01~~	~~0b1000~~	~~0b000~~	~~0b001~~	~~0b0010~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TTLB == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TTLBIS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        TLBI_RVAE1IS(X[t]);
elsif PSTATE.EL == EL2 then
    TLBI_RVAE1IS(X[t]);
elsif PSTATE.EL == EL3 then
    TLBI_RVAE1IS(X[t]);
```

27~~13~~/03~~12~~/2019~~2018~~ 21~~16~~:59~~42~~: e5e4db499bf9867a4b93324c4dbac985d3da93766379d01e197f1d40720d32d0f84e419e9187e009

(old) htmldiff from- (new)

TLBI RVAE1OS, TLB Range Invalidate by VA, EL1, Outer Shareable

The TLBI RVAE1OS characteristics are:

Purpose

Invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 1 translation table entry.
- The entry would be used to translate the specified VA, and one of the following applies:
  - The entry is from a level of lookup above the final level and matches the specified ASID.
  - The entry is a global entry from the final level of lookup.
  - The entry is a non-global entry from the final level of lookup that matches the specified ASID.
- When EL2 is implemented and enabled in the ~~current~~ Security state described by the current value of~~:~~ SCR_EL3.NS:
  - If HCR_EL2.{E2H, TGE} is not {1, 1}, the entry would be used with the current VMID and would be required to translate the specified VA using the EL1&0 translation regime.
  - If HCR_EL2.{E2H, TGE} is {1, 1}, the entry would be required to translate the specified VA using the EL2&0 translation regime.
- When EL2 is not implemented or is disabled in the current Security state, the entry would be required to translate the specified VA using the EL1&0 translation regime.
- The entry is within the address range determined by the formula [BaseADDR <= VA < BaseADDR + ((NUM +1)*2^(5*SCALE +1) * Translation_Granule_Size)].

The invalidation ~~only~~ applies to all PEs in the same Outer Shareable shareability domain as the PE that executes this System instruction.

Note

When a TLB maintenance instruction is generated to the Secure EL1&0 translation regime and is defined to pass a VMID argument, or would be defined to pass a VMID argument if SCR_EL3.EEL2==1, then:

- A PE with SCR_EL3.EEL2==1 is not architecturally required to invalidate any entries in the Secure EL1&0 translation of a PE in the same required shareability domain with SCR_EL3.EEL2==0.
- A PE with SCR_EL3.EEL2==0 is not architecturally required to invalidate any entries in the Secure EL1&0 translation of a PE in the same required shareability domain with SCR_EL3.EEL2==1.
- A PE is architecturally required to invalidate all relevant entries in the Secure EL1&0 translation of a System MMU in the same required shareability domain with a VMID of 0.

The range of addresses invalidated is UNPREDICTABLE when:

- For the 4K translation granule:
  - If TTL==01 and BaseADDR[29:12] is not equal to 000000000000000000.
  - If TTL==10 and BaseADDR[20:12] is not equal to 000000000.
- For the 16K translation granule:
  - If TTL==10 and BaseADDR[24:14] is not equal to 00000000000.
- For the 64K translation granule:

- If TTL==01 and BaseADDR[41:16] is not equal to 00000000000000000000000000.
- If TTL==10 and BaseADDR[28:16] is not equal to 0000000000000.

~~For more information about the architectural requirements for this instruction see 'Invalidation of TLB entries from stage 2 translations' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.~~

Configuration

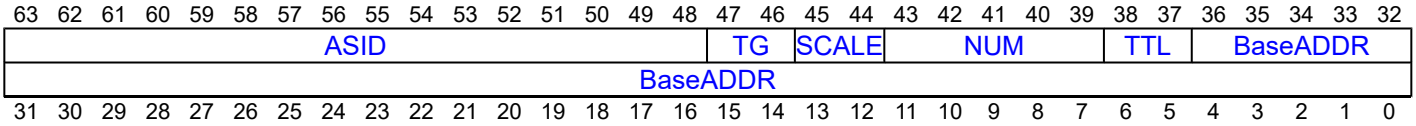
This instruction is present only when ARMv8.4-TLBI is implemented. Otherwise, direct accesses to TLBI RVAE1OS are UNDEFINED.

Attributes

TLBI RVAE1OS is a 64-bit System instruction.

Field descriptions

The TLBI RVAE1OS input value bit assignments are:



ASID, bits [63:48]

ASID value to match. Any TLB entries that match the ASID value and VA value will be affected by this System instruction.~~operation.~~

Global TLB entries that match the VA value will be affected by this System instruction~~operation~~, regardless of the value of the ASID field.

If the implementation supports 16 bits of ASID, but only 8 bits are being used in the context being invalidated, the upper bits are RES0.

TG, bits [47:46]

Translation granule size.

TG	Meaning
0b00	Reserved.
0b01	4K translation granule.
0b10	16K translation granule.
0b11	64K translation granule.

The instruction takes a translation granule size for the translations that are being invalidated. If the translations used a different translation granule size than the one being specified, then the architecture does not require that the instruction invalidates any entries.

SCALE, bits [45:44]

The exponent element of the calculation that is used to produce the upper range.

NUM, bits [43:39]

The base element of the calculation that is used to produce the upper range.

TTL, bits [38:37]

TTL Level hint. The TTL hint is only guaranteed to invalidate entries in the range that match the level described by the TTL hint.

TTL	Meaning
0b00	The entries in the range can be using any level for the translation table entries.
0b01	When using a 4KB or 64KB translation granule, all entries to invalidate are Level 1 translation table entries. When using a 16KB translation granule, this value is reserved and hardware should treat this field as 0b00.
0b10	All entries to invalidate are Level 2 translation table entries.
0b11	All entries to invalidate are Level 3 translation table entries.

BaseADDR, bits [36:0]

The starting address for the range of the maintenance instruction.

When using a 4KB translation granule, this field is BaseADDR[48:12].

When using a 16KB translation granule, this field is BaseADDR[50:14].

When using a 64KB translation granule, this field is BaseADDR[52:16].

Executing the TLBI RVAE1OS instruction

Accesses to this instruction use the following encodings:

```
TLBI RVAE1OS{, <Xt>}
```

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b01	0b000	0b1000	0b0101	0b001
~~0b01~~	~~0b1000~~	~~0b000~~	~~0b001~~	~~0b0101~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TTLB == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TTLBOS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        TLBI_RVAE1OS(X[t]);
elsif PSTATE.EL == EL2 then
    TLBI_RVAE1OS(X[t]);
elsif PSTATE.EL == EL3 then
    TLBI_RVAE1OS(X[t]);
```

27~~13~~/03~~12~~/2019~~2018~~ 21~~16~~:59~~42~~: e5e4db499bf9867a4b93324c4dbac985d3da93766379d01e197f1d40720d32d0f84e419e9187e009

(old) htmldiff from- (new)

TLBI RVAE2, TLB Range Invalidate by VA, EL2

The TLBI RVAE2 characteristics are:

Purpose

When EL2 is implemented and enabled in the current Security state, invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 1 translation table entry.entry, from any level of the translation table walk.
-
  - The entry is from a level of lookup above the final level and matches the specified ASID.
  - The entry is a global entry from the final level of lookup.
  - The entry is a non-global entry from the final level of lookup that matches the specified ASID.

  The entry would be used to translate the specified VA, inand theone specifiedof range determined by the formulafollowing [BaseADDR <= VA < BaseADDR + ((NUM +1)*2^(5*SCALE +1) * Translation_Granule_Size)], using the EL2 or EL2&0 translation regime.applies:
- If HCR_EL2..{E2H, ==TGE} 0is not {1, 1}, the entry iswould frombe anyused levelwith ofthe current VMID and would be required to translate the specified VA using the EL2 translation table walk.regime.
- If HCR_EL2..{E2H, ==TGE} is {1, one1}, ofthe entry would be required to translate the followingspecified applies:VA using the EL2&0 translation regime.
  - The entry is from a level of the translation table walk above the final level and matches the specified ASID.
  - The entry is a global entry from the final level of the translation table walk.
  - The entry is a non-global entry from the final level of the translation table walk and matches the specified ASID.
- The entry is within the address range determined by the formula [BaseADDR <= VA < BaseADDR + ((NUM +1)*2^(5*SCALE +1) * Translation_Granule_Size)].

The invalidation only applies to the PE that executes this System instruction.

The range of addresses invalidated is UNPREDICTABLE when:

- For the 4K translation granule:
  - If TTL==01 and BaseADDR[29:12] is not equal to 000000000000000000.
  - If TTL==10 and BaseADDR[20:12] is not equal to 000000000.
- For the 16K translation granule:
  - If TTL==10 and BaseADDR[24:14] is not equal to 00000000000.
- For the 64K translation granule:
  - If TTL==01 and BaseADDR[41:16] is not equal to 000000000000000000000000000.
  - If TTL==10 and BaseADDR[28:16] is not equal to 0000000000000.

For more information about the architectural requirements for this instruction see 'Invalidation of TLB entries from stage 2 translations' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

Configuration

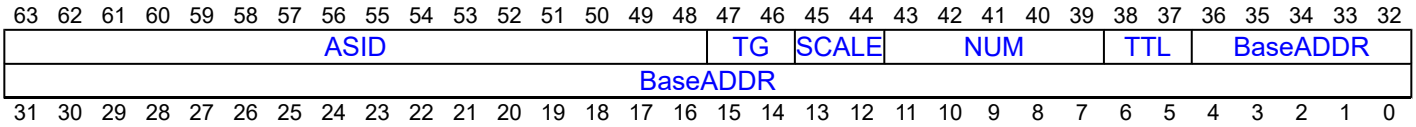
This instruction is present only when ARMv8.4-TLBI is implemented. Otherwise, direct accesses to TLBI RVAE2 are UNDEFINED.

Attributes

TLBI RVAE2 is a 64-bit System instruction.

Field descriptions

The TLBI RVAE2 input value bit assignments are:



ASID, bits [63:48]

When HCR_EL2.E2H == 1:

ASID value to match. Any TLB entries that match the ASID value and VA value will be affected by this System instruction~~.operation.~~

Global TLB entries that match the VA value will be affected by this System instruction~~operation~~, regardless of the value of the ASID field.

If the implementation supports 16 bits of ASID, but only 8 bits are being used in the context being invalidated, the upper bits are RES0.

Otherwise:

Reserved, RES0.

TG, bits [47:46]

Translation granule size.

TG	Meaning
0b00	Reserved.
0b01	4K translation granule.
0b10	16K translation granule.
0b11	64K translation granule.

The instruction takes a translation granule size for the translations that are being invalidated. If the translations used a different translation granule size than the one being specified, then the architecture does not require that the instruction invalidates any entries.

SCALE, bits [45:44]

The exponent element of the calculation that is used to produce the upper range.

NUM, bits [43:39]

The base element of the calculation that is used to produce the upper range.

TTL, bits [38:37]

TTL Level hint. The TTL hint is only guaranteed to invalidate entries in the range that match the level described by the TTL hint.

TTL	Meaning
0b00	The entries in the range can be using any level for the translation table entries.
0b01	When using a 4KB or 64KB translation granule, all entries to invalidate are Level 1 translation table entries. When using a 16KB translation granule, this value is reserved and hardware should treat this field as 0b00.
0b10	All entries to invalidate are Level 2 translation table entries.
0b11	All entries to invalidate are Level 3 translation table entries.

BaseADDR, bits [36:0]

The starting address for the range of the maintenance instruction.

When using a 4KB translation granule, this field is BaseADDR[48:12].

When using a 16KB translation granule, this field is BaseADDR[50:14].

When using a 64KB translation granule, this field is BaseADDR[52:16].

Executing the TLBI RVAE2 instruction

Accesses to this instruction use the following encodings:

```
TLBI RVAE2{, <Xt>}
```

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b01	0b100	0b1000	0b0110	0b001
~~0b01~~	~~0b1000~~	~~0b100~~	~~0b001~~	~~0b0110~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    TLBI_RVAE2(X[t]);
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        UNDEFINED;
    else
        TLBI_RVAE2(X[t]);
```

27~~13~~/03~~12~~/2019~~2018~~ 21~~16~~:59~~42~~; e5e4db499bf9867a4b93324c4dbac985d3da93766~~379d01e197f1d40720d32d0f84c419e9187c009~~

(old) htmldiff from- (new)

TLBI RVAE2IS, TLB Range Invalidate by VA, EL2, Inner Shareable

The TLBI RVAE2IS characteristics are:

Purpose

When EL2 is implemented and enabled in the current Security state, invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 1 translation table entry.entry, from any level of the translation table walk.
- ◦ The entry is from a level of lookup above the final level and matches the specified ASID.
  ◦ The entry is a global entry from the final level of lookup.
  ◦ The entry is a non-global entry from the final level of lookup that matches the specified ASID.

  The entry would be used to translate the specified VA, inand theone specifiedof range determined by the formulafollowing [BaseADDR <= VA < BaseADDR + ((NUM +1)*2^(5*SCALE +1) * Translation_Granule_Size)], using the EL2 or EL2&0 translation regime.applies:
- If HCR_EL2.{E2H, TGE} 0is not {1, 1}, the entry iswould frombe anyused levelwith ofthe current VMID and would be required to translate the specified VA using the EL2 translation table walk.regime.
- If HCR_EL2.{E2H, TGE} is {1, one1}, ofthe entry would be required to translate the followingspecified applies:VA using the EL2&0 translation regime.
  ◦ The entry is from a level of the translation table walk above the final level and matches the specified ASID.
  ◦ The entry is a global entry from the final level of the translation table walk.
  ◦ The entry is a non-global entry from the final level of the translation table walk and matches the specified ASID.
- The entry is within the address range determined by the formula [BaseADDR <= VA < BaseADDR + ((NUM +1)*2^(5*SCALE +1) * Translation_Granule_Size)].

The invalidation only applies to all PEs in the same Inner Shareable shareability domain as the PE that executes this System instruction.

The range of addresses invalidated is UNPREDICTABLE when:

- For the 4K translation granule:
  ◦ If TTL==01 and BaseADDR[29:12] is not equal to 000000000000000000.
  ◦ If TTL==10 and BaseADDR[20:12] is not equal to 000000000.
- For the 16K translation granule:
  ◦ If TTL==10 and BaseADDR[24:14] is not equal to 00000000000.
- For the 64K translation granule:
  ◦ If TTL==01 and BaseADDR[41:16] is not equal to 000000000000000000000000000.
  ◦ If TTL==10 and BaseADDR[28:16] is not equal to 0000000000000.

For more information about the architectural requirements for this instruction see 'Invalidation of TLB entries from stage 2 translations' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

Configuration

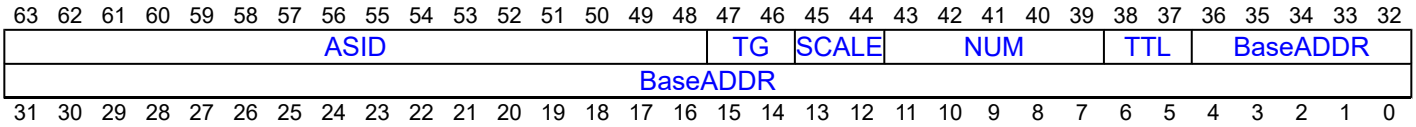
This instruction is present only when ARMv8.4-TLBI is implemented. Otherwise, direct accesses to TLBI RVAE2IS are UNDEFINED.

Attributes

TLBI RVAE2IS is a 64-bit System instruction.

Field descriptions

The TLBI RVAE2IS input value bit assignments are:



ASID, bits [63:48]

When HCR_EL2.E2H == 1:

ASID value to match. Any TLB entries that match the ASID value and VA value will be affected by this System instruction~~.operation.~~

Global TLB entries that match the VA value will be affected by this System instruction~~operation~~, regardless of the value of the ASID field.

If the implementation supports 16 bits of ASID, but only 8 bits are being used in the context being invalidated, the upper bits are RES0.

Otherwise:

Reserved, RES0.

TG, bits [47:46]

Translation granule size.

TG	Meaning
0b00	Reserved.
0b01	4K translation granule.
0b10	16K translation granule.
0b11	64K translation granule.

The instruction takes a translation granule size for the translations that are being invalidated. If the translations used a different translation granule size than the one being specified, then the architecture does not require that the instruction invalidates any entries.

SCALE, bits [45:44]

The exponent element of the calculation that is used to produce the upper range.

NUM, bits [43:39]

The base element of the calculation that is used to produce the upper range.

TTL, bits [38:37]

TTL Level hint. The TTL hint is only guaranteed to invalidate entries in the range that match the level described by the TTL hint.

TTL	Meaning
0b00	The entries in the range can be using any level for the translation table entries.
0b01	When using a 4KB or 64KB translation granule, all entries to invalidate are Level 1 translation table entries. When using a 16KB translation granule, this value is reserved and hardware should treat this field as 0b00.
0b10	All entries to invalidate are Level 2 translation table entries.
0b11	All entries to invalidate are Level 3 translation table entries.

BaseADDR, bits [36:0]

The starting address for the range of the maintenance instruction.

When using a 4KB translation granule, this field is BaseADDR[48:12].

When using a 16KB translation granule, this field is BaseADDR[50:14].

When using a 64KB translation granule, this field is BaseADDR[52:16].

Executing the TLBI RVAE2IS instruction

Accesses to this instruction use the following encodings:

```
TLBI RVAE2IS{, <Xt>}
```

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b01	0b100	0b1000	0b0010	0b001
~~0b01~~	~~0b1000~~	~~0b100~~	~~0b001~~	~~0b0010~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    TLBI_RVAE2IS(X[t]);
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        UNDEFINED;
    else
        TLBI_RVAE2IS(X[t]);
```

27~~13~~/03~~12~~/2019~~2018~~ 21~~16~~:59~~42~~; e5e4db499bf9867a4b93324c4dbac985d3da9376~~6379d01e197f1d40720d32d0f84c419e9187c009~~

(old) htmldiff from- (new)

(old) htmldiff from- (new)

TLBI RVAE2OS, TLB Range Invalidate by VA, EL2, Outer Shareable

The TLBI RVAE2OS characteristics are:

Purpose

When EL2 is implemented and enabled in the current Security state, invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 1 translation table entry.entry, from any level of the translation table walk.
-
  - The entry is from a level of lookup above the final level and matches the specified ASID.
  - The entry is a global entry from the final level of lookup.
  - The entry is a non-global entry from the final level of lookup that matches the specified ASID.

  The entry would be used to translate the specified VA, inand theone specifiedof range determined by the formulafollowing [BaseADDR <= VA < BaseADDR + ((NUM +1)*2^(5*SCALE +1) * Translation_Granule_Size)], using the EL2 or EL2&0 translation regime.applies:
- If HCR_EL2..{E2H, ==TGE} 0is not {1, 1}, the entry iswould frombe anyused levelwith ofthe current VMID and would be required to translate the specified VA using the EL2 translation table walk.regime.
- If HCR_EL2..{E2H, ==TGE} is {1, one1}, ofthe entry would be required to translate the followingspecified applies:VA using the EL2&0 translation regime.
  - The entry is from a level of the translation table walk above the final level and matches the specified ASID.
  - The entry is a global entry from the final level of the translation table walk.
  - The entry is a non-global entry from the final level of the translation table walk and matches the specified ASID.
- The entry is within the address range determined by the formula [BaseADDR <= VA < BaseADDR + ((NUM +1)*2^(5*SCALE +1) * Translation_Granule_Size)].

The invalidation only applies to all PEs in the same Outer Shareable shareability domain as the PE that executes this System instruction.

The range of addresses invalidated is UNPREDICTABLE when:

- For the 4K translation granule:
  - If TTL==01 and BaseADDR[29:12] is not equal to 000000000000000000.
  - If TTL==10 and BaseADDR[20:12] is not equal to 000000000.
- For the 16K translation granule:
  - If TTL==10 and BaseADDR[24:14] is not equal to 00000000000.
- For the 64K translation granule:
  - If TTL==01 and BaseADDR[41:16] is not equal to 000000000000000000000000000.
  - If TTL==10 and BaseADDR[28:16] is not equal to 0000000000000.

For more information about the architectural requirements for this instruction see 'Invalidation of TLB entries from stage 2 translations' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

Configuration

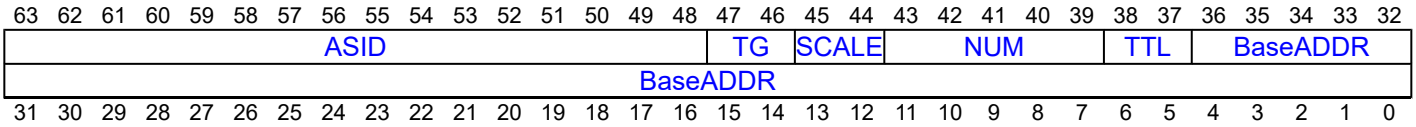
This instruction is present only when ARMv8.4-TLBI is implemented. Otherwise, direct accesses to TLBI RVAE2OS are UNDEFINED.

Attributes

TLBI RVAE2OS is a 64-bit System instruction.

Field descriptions

The TLBI RVAE2OS input value bit assignments are:



ASID, bits [63:48]

When HCR_EL2.E2H == 1:

ASID value to match. Any TLB entries that match the ASID value and VA value will be affected by this System instruction~~.operation.~~

Global TLB entries that match the VA value will be affected by this System instruction~~operation~~, regardless of the value of the ASID field.

If the implementation supports 16 bits of ASID, but only 8 bits are being used in the context being invalidated, the upper bits are RES0.

Otherwise:

Reserved, RES0.

TG, bits [47:46]

Translation granule size.

TG	Meaning
0b00	Reserved.
0b01	4K translation granule.
0b10	16K translation granule.
0b11	64K translation granule.

The instruction takes a translation granule size for the translations that are being invalidated. If the translations used a different translation granule size than the one being specified, then the architecture does not require that the instruction invalidates any entries.

SCALE, bits [45:44]

The exponent element of the calculation that is used to produce the upper range.

NUM, bits [43:39]

The base element of the calculation that is used to produce the upper range.

TTL, bits [38:37]

TTL Level hint. The TTL hint is only guaranteed to invalidate entries in the range that match the level described by the TTL hint.

TTL	Meaning
0b00	The entries in the range can be using any level for the translation table entries.
0b01	When using a 4KB or 64KB translation granule, all entries to invalidate are Level 1 translation table entries. When using a 16KB translation granule, this value is reserved and hardware should treat this field as 0b00.
0b10	All entries to invalidate are Level 2 translation table entries.
0b11	All entries to invalidate are Level 3 translation table entries.

BaseADDR, bits [36:0]

The starting address for the range of the maintenance instruction.

When using a 4KB translation granule, this field is BaseADDR[48:12].

When using a 16KB translation granule, this field is BaseADDR[50:14].

When using a 64KB translation granule, this field is BaseADDR[52:16].

Executing the TLBI RVAE2OS instruction

Accesses to this instruction use the following encodings:

```
TLBI RVAE2OS{, <Xt>}
```

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b01	0b100	0b1000	0b0101	0b001
~~0b01~~	~~0b1000~~	~~0b100~~	~~0b001~~	~~0b0101~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    TLBI_RVAE2OS(X[t]);
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        UNDEFINED;
    else
        TLBI_RVAE2OS(X[t]);
```

27~~13~~/03~~12~~/2019~~2018~~ 21~~16~~:59~~42~~; e5e4db499bf9867a4b93324c4dbac985d3da9376~~6379d01e197f1d40720d32d0f84c419e9187c009~~

(old) htmldiff from- (new)

(old) htmldiff from- (new)

TLBI RVAE3, TLB Range Invalidate by VA, EL3

The TLBI RVAE3 characteristics are:

Purpose

If EL3 is implemented, invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 1 translation table entry, from any level of the translation table walk.
- The entry would be used to translate the specified VA using the EL3 translation regime.
- The entry is within the address range determined by the formula [BaseADDR <= VA < BaseADDR + ((NUM +1)*2^(5*SCALE +1) * Translation_Granule_Size)].

The invalidation ~~only~~ applies to the PE that executes this System instruction.

The range of addresses invalidated is UNPREDICTABLE when:

- For the 4K translation granule:
  - If TTL==01 and BaseADDR[29:12] is not equal to 000000000000000000.
  - If TTL==10 and BaseADDR[20:12] is not equal to 000000000.
- For the 16K translation granule:
  - If TTL==10 and BaseADDR[24:14] is not equal to 00000000000.
- For the 64K translation granule:
  - If TTL==01 and BaseADDR[41:16] is not equal to 000000000000000000000000000000.
  - If TTL==10 and BaseADDR[28:16] is not equal to 0000000000000.

~~For more information about the architectural requirements for this instruction see 'Invalidation of TLB entries from stage 2 translations' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.~~

Configuration

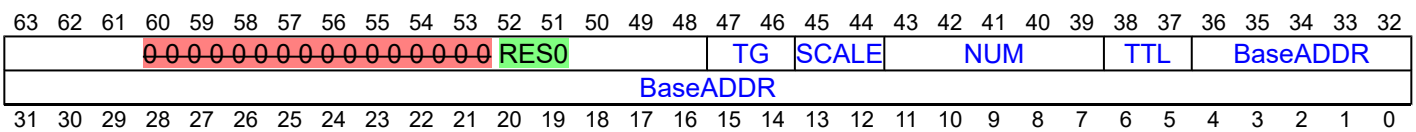
This instruction is present only when ARMv8.4-TLBI is implemented. Otherwise, direct accesses to TLBI RVAE3 are UNDEFINED.

Attributes

TLBI RVAE3 is a 64-bit System instruction.

Field descriptions

The TLBI RVAE3 input value bit assignments are:



Bits [63:48]

Reserved, RES0.

TG, bits [47:46]

Translation granule size.

TG	Meaning
0b00	Reserved.
0b01	4K translation granule.
0b10	16K translation granule.
0b11	64K translation granule.

The instruction takes a translation granule size for the translations that are being invalidated. If the translations used a different translation granule size than the one being specified, then the architecture does not require that the instruction invalidates any entries.

SCALE, bits [45:44]

The exponent element of the calculation that is used to produce the upper range.

NUM, bits [43:39]

The base element of the calculation that is used to produce the upper range.

TTL, bits [38:37]

TTL Level hint. The TTL hint is only guaranteed to invalidate entries in the range that match the level described by the TTL hint.

TTL	Meaning
0b00	The entries in the range can be using any level for the translation table entries.
0b01	When using a 4KB or 64KB translation granule, all entries to invalidate are Level 1 translation table entries. When using a 16KB translation granule, this value is reserved and hardware should treat this field as 0b00.
0b10	All entries to invalidate are Level 2 translation table entries.
0b11	All entries to invalidate are Level 3 translation table entries.

BaseADDR, bits [36:0]

The starting address for the range of the maintenance instruction.

When using a 4KB translation granule, this field is BaseADDR[48:12].

When using a 16KB translation granule, this field is BaseADDR[50:14].

When using a 64KB translation granule, this field is BaseADDR[52:16].

Executing the TLBI RVAE3 instruction

Accesses to this instruction use the following encodings:

```
TLBI RVAE3{, <Xt>}
```

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b01	0b110	0b1000	0b0110	0b001
~~0b01~~	~~0b1000~~	~~0b110~~	~~0b001~~	~~0b0110~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    TLBI_RVAE3(X[t]);
```

27~~13~~/03~~12~~/2019~~2018~~ 21~~16~~:59~~42~~: e5e4db499bf9867a4b93324c4dbac985d3da93766~~379d01e197f1d40720d32d0f84e419e9187e009~~

(old) htmldiff from- (new)

(old) htmldiff from- (new)

TLBI RVAE3IS, TLB Range Invalidate by VA, EL3, Inner Shareable

The TLBI RVAE3IS characteristics are:

Purpose

If EL3 is implemented, invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 1 translation table entry, from any level of the translation table walk.
- The entry would be used to translate the specified VA using the EL3 translation regime.
- The entry is within the address range determined by the formula [BaseADDR <= VA < BaseADDR + ((NUM +1)*2^(5*SCALE +1) * Translation_Granule_Size)].

The invalidation ~~only~~ applies to all PEs in the same Inner Shareable shareability domain as the PE that executes this System instruction.

The range of addresses invalidated is UNPREDICTABLE when:

- For the 4K translation granule:
  - If TTL==01 and BaseADDR[29:12] is not equal to 000000000000000000.
  - If TTL==10 and BaseADDR[20:12] is not equal to 000000000.
- For the 16K translation granule:
  - If TTL==10 and BaseADDR[24:14] is not equal to 00000000000.
- For the 64K translation granule:
  - If TTL==01 and BaseADDR[41:16] is not equal to 000000000000000000000000000000.
  - If TTL==10 and BaseADDR[28:16] is not equal to 0000000000000.

~~For more information about the architectural requirements for this instruction see 'Invalidation of TLB entries from stage 2 translations' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.~~

Configuration

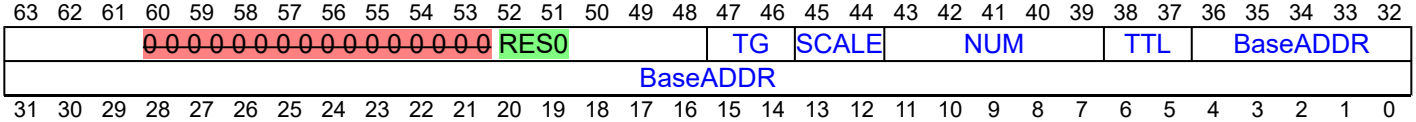
This instruction is present only when ARMv8.4-TLBI is implemented. Otherwise, direct accesses to TLBI RVAE3IS are UNDEFINED.

Attributes

TLBI RVAE3IS is a 64-bit System instruction.

Field descriptions

The TLBI RVAE3IS input value bit assignments are:



Bits [63:48]

Reserved, RES0.

TG, bits [47:46]

Translation granule size.

TG	Meaning
0b00	Reserved.
0b01	4K translation granule.
0b10	16K translation granule.
0b11	64K translation granule.

The instruction takes a translation granule size for the translations that are being invalidated. If the translations used a different translation granule size than the one being specified, then the architecture does not require that the instruction invalidates any entries.

SCALE, bits [45:44]

The exponent element of the calculation that is used to produce the upper range.

NUM, bits [43:39]

The base element of the calculation that is used to produce the upper range.

TTL, bits [38:37]

TTL Level hint. The TTL hint is only guaranteed to invalidate entries in the range that match the level described by the TTL hint.

TTL	Meaning
0b00	The entries in the range can be using any level for the translation table entries.
0b01	When using a 4KB or 64KB translation granule, all entries to invalidate are Level 1 translation table entries. When using a 16KB translation granule, this value is reserved and hardware should treat this field as 0b00.
0b10	All entries to invalidate are Level 2 translation table entries.
0b11	All entries to invalidate are Level 3 translation table entries.

BaseADDR, bits [36:0]

The starting address for the range of the maintenance instruction.

When using a 4KB translation granule, this field is BaseADDR[48:12].

When using a 16KB translation granule, this field is BaseADDR[50:14].

When using a 64KB translation granule, this field is BaseADDR[52:16].

Executing the TLBI RVAE3IS instruction

Accesses to this instruction use the following encodings:

```
TLBI RVAE3IS{, <Xt>}
```

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b01	0b110	0b1000	0b0010	0b001
~~0b01~~	~~0b1000~~	~~0b110~~	~~0b001~~	~~0b0010~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    TLBI_RVAE3IS(X[t]);
```

2713/0312/20192018 2116:5942: e5e4db499bf9867a4b93324c4dbac985d3da93766379d01e197f1d40720d32d0f84e419e9187e009

(old) htmldiff from- (new)

TLBI RVAE3OS, TLB Range Invalidate by VA, EL3, Outer Shareable

The TLBI RVAE3OS characteristics are:

Purpose

If EL3 is implemented, invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 1 translation table entry, from any level of the translation table walk.
- The entry would be used to translate the specified VA using the EL3 translation regime.
- The entry is within the address range determined by the formula [BaseADDR <= VA < BaseADDR + ((NUM +1)*2^(5*SCALE +1) * Translation_Granule_Size)].

The invalidation ~~only~~ applies to all PEs in the same Outer Shareable shareability domain as the PE that executes this System instruction.

The range of addresses invalidated is UNPREDICTABLE when:

- For the 4K translation granule:
  - If TTL==01 and BaseADDR[29:12] is not equal to 000000000000000000.
  - If TTL==10 and BaseADDR[20:12] is not equal to 000000000.
- For the 16K translation granule:
  - If TTL==10 and BaseADDR[24:14] is not equal to 00000000000.
- For the 64K translation granule:
  - If TTL==01 and BaseADDR[41:16] is not equal to 000000000000000000000000000000.
  - If TTL==10 and BaseADDR[28:16] is not equal to 0000000000000.

~~For more information about the architectural requirements for this instruction see 'Invalidation of TLB entries from stage 2 translations' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.~~

Configuration

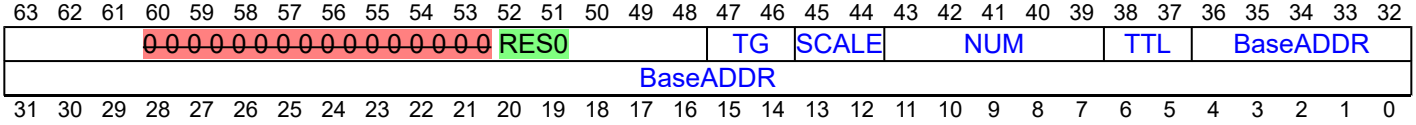
This instruction is present only when ARMv8.4-TLBI is implemented. Otherwise, direct accesses to TLBI RVAE3OS are UNDEFINED.

Attributes

TLBI RVAE3OS is a 64-bit System instruction.

Field descriptions

The TLBI RVAE3OS input value bit assignments are:



Bits [63:48]

Reserved, RES0.

TG, bits [47:46]

Translation granule size.

TG	Meaning
0b00	Reserved.
0b01	4K translation granule.
0b10	16K translation granule.
0b11	64K translation granule.

The instruction takes a translation granule size for the translations that are being invalidated. If the translations used a different translation granule size than the one being specified, then the architecture does not require that the instruction invalidates any entries.

SCALE, bits [45:44]

The exponent element of the calculation that is used to produce the upper range.

NUM, bits [43:39]

The base element of the calculation that is used to produce the upper range.

TTL, bits [38:37]

TTL Level hint. The TTL hint is only guaranteed to invalidate entries in the range that match the level described by the TTL hint.

TTL	Meaning
0b00	The entries in the range can be using any level for the translation table entries.
0b01	When using a 4KB or 64KB translation granule, all entries to invalidate are Level 1 translation table entries. When using a 16KB translation granule, this value is reserved and hardware should treat this field as 0b00.
0b10	All entries to invalidate are Level 2 translation table entries.
0b11	All entries to invalidate are Level 3 translation table entries.

BaseADDR, bits [36:0]

The starting address for the range of the maintenance instruction.

When using a 4KB translation granule, this field is BaseADDR[48:12].

When using a 16KB translation granule, this field is BaseADDR[50:14].

When using a 64KB translation granule, this field is BaseADDR[52:16].

Executing the TLBI RVAE3OS instruction

Accesses to this instruction use the following encodings:

```
TLBI RVAE3OS{, <Xt>}
```

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b01	0b110	0b1000	0b0101	0b001
~~0b01~~	~~0b1000~~	~~0b110~~	~~0b001~~	~~0b0101~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    TLBI_RVAE3OS(X[t]);
```

2713/0312/20192018 2116:5942: e5e4db499bf9867a4b93324c4dbac985d3da93766379d01e197f1d40720d32d0f84e419e9187e009

(old) htmldiff from- (new)

(old) htmldiff from- (new)

TLBI RVALE1, TLB Range Invalidate by VA, Last level, EL1

The TLBI RVALE1 characteristics are:

Purpose

Invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 1 translation table entry.
- The entry would be used to translate the specified VA, and one of the following applies:
  - The entry is a global entry from the final level of lookup.
  - The entry is a non-global entry from the final level of lookup that matches the specified ASID.
- When EL2 is implemented and enabled in the ~~current~~ Security state described by the current value of~~:~~ SCR_EL3.NS:
  - If HCR_EL2.{E2H, TGE} is not {1, 1}, the entry would be used with the current VMID and would be required to translate the specified VA using the EL1&0 translation regime.
  - If HCR_EL2.{E2H, TGE} is {1, 1}, the entry would be required to translate the specified VA using the EL2&0 translation regime.
- When EL2 is not implemented or is disabled in the current Security state, the entry would be required to translate the specified VA using the EL1&0 translation regime.
- The entry is within the address range determined by the formula [BaseADDR <= VA < BaseADDR + ((NUM +1)*2^(5*SCALE +1) * Translation_Granule_Size)].

The invalidation ~~only~~ applies to the PE that executes this System instruction.

The range of addresses invalidated is UNPREDICTABLE when:

- For the 4K translation granule:
  - If TTL==01 and BaseADDR[29:12] is not equal to 000000000000000000.
  - If TTL==10 and BaseADDR[20:12] is not equal to 000000000.
- For the 16K translation granule:
  - If TTL==10 and BaseADDR[24:14] is not equal to 00000000000.
- For the 64K translation granule:
  - If TTL==01 and BaseADDR[41:16] is not equal to 000000000000000000000000000.
  - If TTL==10 and BaseADDR[28:16] is not equal to 0000000000000.

For more information about the architectural requirements for this System instruction see 'Invalidation of TLB entries from stage 2 translations' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

Configuration

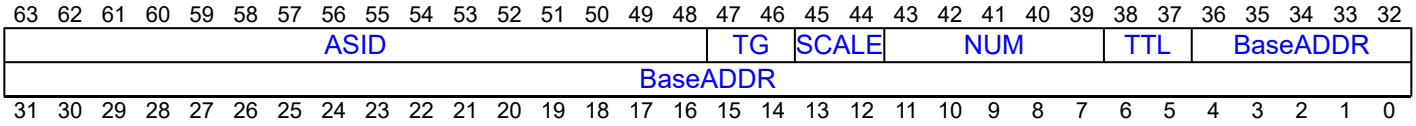
This instruction is present only when ARMv8.4-TLBI is implemented. Otherwise, direct accesses to TLBI RVALE1 are UNDEFINED.

Attributes

TLBI RVALE1 is a 64-bit System instruction.

Field descriptions

The TLBI RVALE1 input value bit assignments are:



ASID, bits [63:48]

ASID value to match. Any TLB entries that match the ASID value and VA value will be affected by this System instruction.~~operation.~~

Global TLB entries that match the VA value will be affected by this System instruction~~operation~~, regardless of the value of the ASID field.

If the implementation supports 16 bits of ASID, but only 8 bits are being used in the context being invalidated, the upper bits are RES0.

TG, bits [47:46]

Translation granule size.

TG	Meaning
0b00	Reserved.
0b01	4K translation granule.
0b10	16K translation granule.
0b11	64K translation granule.

The instruction takes a translation granule size for the translations that are being invalidated. If the translations used a different translation granule size than the one being specified, then the architecture does not require that the instruction invalidates any entries.

SCALE, bits [45:44]

The exponent element of the calculation that is used to produce the upper range.

NUM, bits [43:39]

The base element of the calculation that is used to produce the upper range.

TTL, bits [38:37]

TTL Level hint. The TTL hint is only guaranteed to invalidate entries in the range that match the level described by the TTL hint.

TTL	Meaning
0b00	The entries in the range can be using any level for the translation table entries.
0b01	When using a 4KB or 64KB translation granule, all entries to invalidate are Level 1 translation table entries. When using a 16KB translation granule, this value is reserved and hardware should treat this field as 0b00.
0b10	All entries to invalidate are Level 2 translation table entries.
0b11	All entries to invalidate are Level 3 translation table entries.

BaseADDR, bits [36:0]

The starting address for the range of the maintenance instruction.

When using a 4KB translation granule, this field is BaseADDR[48:12].

When using a 16KB translation granule, this field is BaseADDR[50:14].

When using a 64KB translation granule, this field is BaseADDR[52:16].

Executing the TLBI RVALE1 instruction

Accesses to this instruction use the following encodings:

```
TLBI RVALE1{, <Xt>}
```

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b01	0b000	0b1000	0b0110	0b101
~~0b01~~	~~0b1000~~	~~0b000~~	~~0b101~~	~~0b0110~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TTLB == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.FB == '1' then
        TLBI_RVALE1IS(X[t]);
    else
        TLBI_RVALE1(X[t]);
elsif PSTATE.EL == EL2 then
    TLBI_RVALE1(X[t]);
elsif PSTATE.EL == EL3 then
    TLBI_RVALE1(X[t]);
```

27~~13~~/03~~12~~/2019~~2018~~ 21~~16~~:59~~42~~; e5e4db499bf9867a4b93324c4dbac985d3da9376~~6379d01e197f1d40720d32d0f84c419e9187e009~~

(old) htmldiff from- (new)

(old) htmldiff from- (new)

TLBI RVALE1IS, TLB Range Invalidate by VA, Last level, EL1, Inner Shareable

The TLBI RVALE1IS characteristics are:

Purpose

Invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 1 translation table entry.
- The entry would be used to translate the specified VA, and one of the following applies:
  - The entry is a global entry from the final level of lookup.
  - The entry is a non-global entry from the final level of lookup that matches the specified ASID.
- When EL2 is implemented and enabled in the ~~current~~ Security state described by the current value of~~:~~ SCR_EL3.NS:
  - If HCR_EL2.{E2H, TGE} is not {1, 1}, the entry would be used with the current VMID and would be required to translate the specified VA using the EL1&0 translation regime.
  - If HCR_EL2.{E2H, TGE} is {1, 1}, the entry would be required to translate the specified VA using the EL2&0 translation regime.
- When EL2 is not implemented or is disabled in the current Security state, the entry would be required to translate the specified VA using the EL1&0 translation regime.
- The entry is within the address range determined by the formula [BaseADDR <= VA < BaseADDR + ((NUM +1)*2^(5*SCALE +1) * Translation_Granule_Size)].

The invalidation ~~only~~ applies to all PEs in the same Inner Shareable shareability domain as the PE that executes this System instruction.

Note

When a TLB maintenance instruction is generated to the Secure EL1&0 translation regime and is defined to pass a VMID argument, or would be defined to pass a VMID argument if SCR_EL3.EEL2==1, then:

- A PE with SCR_EL3.EEL2==1 is not architecturally required to invalidate any entries in the Secure EL1&0 translation of a PE in the same required shareability domain with SCR_EL3.EEL2==0.
- A PE with SCR_EL3.EEL2==0 is not architecturally required to invalidate any entries in the Secure EL1&0 translation of a PE in the same required shareability domain with SCR_EL3.EEL2==1.
- A PE is architecturally required to invalidate all relevant entries in the Secure EL1&0 translation of a System MMU in the same required shareability domain with a VMID of 0.

The range of addresses invalidated is UNPREDICTABLE when:

- For the 4K translation granule:
  - If TTL==01 and BaseADDR[29:12] is not equal to 000000000000000000.
  - If TTL==10 and BaseADDR[20:12] is not equal to 000000000.
- For the 16K translation granule:
  - If TTL==10 and BaseADDR[24:14] is not equal to 00000000000.
- For the 64K translation granule:
  - If TTL==01 and BaseADDR[41:16] is not equal to 000000000000000000000000000.

- If TTL==10 and BaseADDR[28:16] is not equal to 0000000000000.

~~For more information about the architectural requirements for this instruction see 'Invalidation of TLB entries from stage 2 translations' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.~~

Configuration

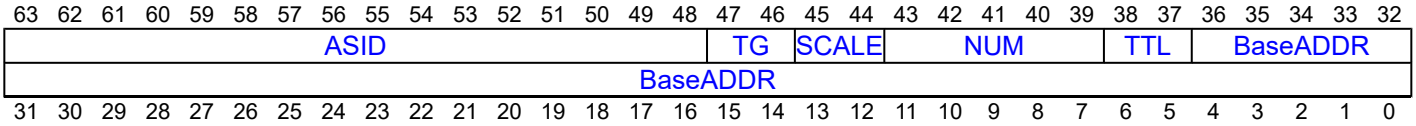
This instruction is present only when ARMv8.4-TLBI is implemented. Otherwise, direct accesses to TLBI RVALE1IS are UNDEFINED.

Attributes

TLBI RVALE1IS is a 64-bit System instruction.

Field descriptions

The TLBI RVALE1IS input value bit assignments are:



ASID, bits [63:48]

ASID value to match. Any TLB entries that match the ASID value and VA value will be affected by this System instruction.~~operation.~~

Global TLB entries that match the VA value will be affected by this System instruction~~operation~~, regardless of the value of the ASID field.

If the implementation supports 16 bits of ASID, but only 8 bits are being used in the context being invalidated, the upper bits are RES0.

TG, bits [47:46]

Translation granule size.

TG	Meaning
0b00	Reserved.
0b01	4K translation granule.
0b10	16K translation granule.
0b11	64K translation granule.

The instruction takes a translation granule size for the translations that are being invalidated. If the translations used a different translation granule size than the one being specified, then the architecture does not require that the instruction invalidates any entries.

SCALE, bits [45:44]

The exponent element of the calculation that is used to produce the upper range.

NUM, bits [43:39]

The base element of the calculation that is used to produce the upper range.

TTL, bits [38:37]

TTL Level hint. The TTL hint is only guaranteed to invalidate entries in the range that match the level described by the TTL hint.

TTL	Meaning
0b00	The entries in the range can be using any level for the translation table entries.
0b01	When using a 4KB or 64KB translation granule, all entries to invalidate are Level 1 translation table entries. When using a 16KB translation granule, this value is reserved and hardware should treat this field as 0b00.
0b10	All entries to invalidate are Level 2 translation table entries.
0b11	All entries to invalidate are Level 3 translation table entries.

BaseADDR, bits [36:0]

The starting address for the range of the maintenance instruction.

When using a 4KB translation granule, this field is BaseADDR[48:12].

When using a 16KB translation granule, this field is BaseADDR[50:14].

When using a 64KB translation granule, this field is BaseADDR[52:16].

Executing the TLBI RVALE1IS instruction

Accesses to this instruction use the following encodings:

```
TLBI RVALE1IS{, <Xt>}
```

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b01	0b000	0b1000	0b0010	0b101
~~0b01~~	~~0b1000~~	~~0b000~~	~~0b101~~	~~0b0010~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TTLB == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TTLBIS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        TLBI_RVALE1IS(X[t]);
elsif PSTATE.EL == EL2 then
    TLBI_RVALE1IS(X[t]);
elsif PSTATE.EL == EL3 then
    TLBI_RVALE1IS(X[t]);
```

27~~13~~/03~~12~~/2019~~2018~~ 21~~16~~:59~~42~~: e5e4db499bf9867a4b93324c4dbac985d3da93766379d01e197f1d40720d32d0f84e419e9187e009

(old) htmldiff from- (new)

TLBI RVALE1OS, TLB Range Invalidate by VA, Last level, EL1, Outer Shareable

The TLBI RVALE1OS characteristics are:

Purpose

Invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 1 translation table entry.
- The entry would be used to translate the specified VA, and one of the following applies:
  - The entry is a global entry from the final level of lookup.
  - The entry is a non-global entry from the final level of lookup that matches the specified ASID.
- When EL2 is implemented and enabled in the ~~current~~ Security state described by the current value of~~:~~ SCR_EL3.NS:
  - If HCR_EL2.{E2H, TGE} is not {1, 1}, the entry would be used with the current VMID and would be required to translate the specified VA using the EL1&0 translation regime.
  - If HCR_EL2.{E2H, TGE} is {1, 1}, the entry would be required to translate the specified VA using the EL2&0 translation regime.
- When EL2 is not implemented or is disabled in the current Security state, the entry would be required to translate the specified VA using the EL1&0 translation regime.
- The entry is within the address range determined by the formula [BaseADDR <= VA < BaseADDR + ((NUM +1)*2^(5*SCALE +1) * Translation_Granule_Size)].

The invalidation ~~only~~ applies to all PEs in the same Outer Shareable shareability domain as the PE that executes this System instruction.

Note

When a TLB maintenance instruction is generated to the Secure EL1&0 translation regime and is defined to pass a VMID argument, or would be defined to pass a VMID argument if SCR_EL3.EEL2==1, then:

- A PE with SCR_EL3.EEL2==1 is not architecturally required to invalidate any entries in the Secure EL1&0 translation of a PE in the same required shareability domain with SCR_EL3.EEL2==0.
- A PE with SCR_EL3.EEL2==0 is not architecturally required to invalidate any entries in the Secure EL1&0 translation of a PE in the same required shareability domain with SCR_EL3.EEL2==1.
- A PE is architecturally required to invalidate all relevant entries in the Secure EL1&0 translation of a System MMU in the same required shareability domain with a VMID of 0.

The range of addresses invalidated is UNPREDICTABLE when:

- For the 4K translation granule:
  - If TTL==01 and BaseADDR[29:12] is not equal to 000000000000000000.
  - If TTL==10 and BaseADDR[20:12] is not equal to 000000000.
- For the 16K translation granule:
  - If TTL==10 and BaseADDR[24:14] is not equal to 00000000000.
- For the 64K translation granule:
  - If TTL==01 and BaseADDR[41:16] is not equal to 000000000000000000000000000.

- If TTL==10 and BaseADDR[28:16] is not equal to 0000000000000.

~~For more information about the architectural requirements for this instruction see 'Invalidation of TLB entries from stage 2 translations' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.~~

Configuration

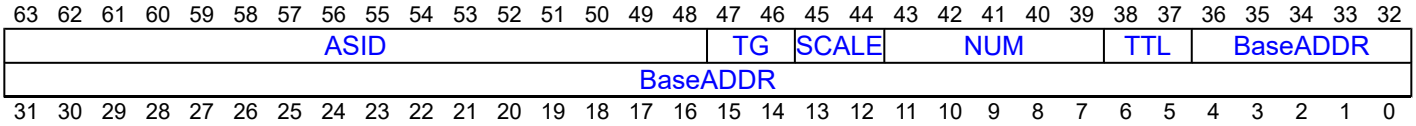
This instruction is present only from Armv8.4, or if ARMv8.4-TLBI is implemented. Otherwise, direct accesses to TLBI RVALE1OS are UNDEFINED.

Attributes

TLBI RVALE1OS is a 64-bit System instruction.

Field descriptions

The TLBI RVALE1OS input value bit assignments are:



ASID, bits [63:48]

ASID value to match. Any TLB entries that match the ASID value and VA value will be affected by this System instruction.~~operation.~~

Global TLB entries that match the VA value will be affected by this System instruction~~operation~~, regardless of the value of the ASID field.

If the implementation supports 16 bits of ASID, but only 8 bits are being used in the context being invalidated, the upper bits are RES0.

TG, bits [47:46]

Translation granule size.

TG	Meaning
0b00	Reserved.
0b01	4K translation granule.
0b10	16K translation granule.
0b11	64K translation granule.

The instruction takes a translation granule size for the translations that are being invalidated. If the translations used a different translation granule size than the one being specified, then the architecture does not require that the instruction invalidates any entries.

SCALE, bits [45:44]

The exponent element of the calculation that is used to produce the upper range.

NUM, bits [43:39]

The base element of the calculation that is used to produce the upper range.

TTL, bits [38:37]

TTL Level hint. The TTL hint is only guaranteed to invalidate entries in the range that match the level described by the TTL hint.

TTL	Meaning
0b00	The entries in the range can be using any level for the translation table entries.
0b01	When using a 4KB or 64KB translation granule, all entries to invalidate are Level 1 translation table entries. When using a 16KB translation granule, this value is reserved and hardware should treat this field as 0b00.
0b10	All entries to invalidate are Level 2 translation table entries.
0b11	All entries to invalidate are Level 3 translation table entries.

BaseADDR, bits [36:0]

The starting address for the range of the maintenance instruction.

When using a 4KB translation granule, this field is BaseADDR[48:12].

When using a 16KB translation granule, this field is BaseADDR[50:14].

When using a 64KB translation granule, this field is BaseADDR[52:16].

Executing the TLBI RVALE1OS instruction

Accesses to this instruction use the following encodings:

```
TLBI RVALE1OS{, <Xt>}
```

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b01	0b000	0b1000	0b0101	0b101
~~0b01~~	~~0b1000~~	~~0b000~~	~~0b101~~	~~0b0101~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TTLB == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TTLBOS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        TLBI_RVALE1OS(X[t]);
elsif PSTATE.EL == EL2 then
    TLBI_RVALE1OS(X[t]);
elsif PSTATE.EL == EL3 then
    TLBI_RVALE1OS(X[t]);
```

27~~13~~/03~~12~~/2019~~2018~~ 21~~16~~:59~~42~~: e5e4db499bf9867a4b93324c4dbac985d3da93766379d01e197f1d40720d32d0f84e419e9187e009

(old) htmldiff from- (new)

TLBI RVALE2, TLB Range Invalidate by VA, Last level, EL2

The TLBI RVALE2 characteristics are:

Purpose

When EL2 is implemented and enabled in the current Security state, invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 1 translation table entry.~~entry, from the final level of the translation table walk.~~
- ~~The entry is a global entry from the final level of lookup.~~
- ~~The entry is a non-global entry from the final level of lookup that matches the specified ASID.~~

  The entry would be used to translate the specified VA~~,~~ in~~and~~ the~~one~~ specified~~of~~ range determined by the formula~~following~~ [BaseADDR <= VA < BaseADDR + ((NUM +1)*2^(5*SCALE +1) * Translation_Granule_Size)] using the EL2 or EL2&0 translation regime.~~applies:~~

- If HCR_EL2.~~{~~E2H~~,~~ ~~==TGE}~~ 0~~is not {~~1~~, 1},~~ the entry is~~would~~ from~~be~~ ~~used with~~ the final~~current~~ level~~VMID~~ of~~and~~ ~~would be required to translate~~ the ~~specified VA using the EL2~~ translation table walk.~~regime.~~
- If HCR_EL2.~~{~~E2H~~,~~ ~~==TGE}~~ ~~is {~~1, one~~1},~~ of~~the~~ ~~entry would be required to translate~~ the following~~specified~~ applies:~~VA using the EL2&0 translation regime.~~
  - The entry is a global entry from the final level of translation table walk.
  - The entry is a non-global entry from the final level of the translation table walk and matches the specified ASID.
- ~~The entry is within the address range determined by the formula [BaseADDR <= VA < BaseADDR + ((NUM +1)*2^(5*SCALE +1) * Translation_Granule_Size)].~~

The invalidation ~~only~~ applies to the PE that executes this System instruction.

The range of addresses invalidated is UNPREDICTABLE when:

- For the 4K translation granule:
  - If TTL==01 and BaseADDR[29:12] is not equal to 000000000000000000.
  - If TTL==10 and BaseADDR[20:12] is not equal to 000000000.
- For the 16K translation granule:
  - If TTL==10 and BaseADDR[24:14] is not equal to 00000000000.
- For the 64K translation granule:
  - If TTL==01 and BaseADDR[41:16] is not equal to 000000000000000000000000000.
  - If TTL==10 and BaseADDR[28:16] is not equal to 0000000000000.

~~For more information about the architectural requirements for this instruction see 'Invalidation of TLB entries from stage 2 translations' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.~~

Configuration

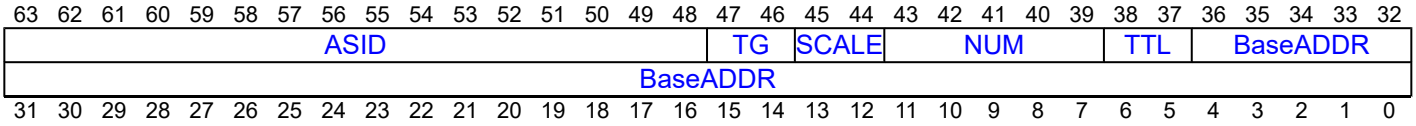
This instruction is present only when ARMv8.4-TLBI is implemented. Otherwise, direct accesses to TLBI RVALE2 are UNDEFINED.

Attributes

TLBI RVALE2 is a 64-bit System instruction.

Field descriptions

The TLBI RVALE2 input value bit assignments are:



ASID, bits [63:48]

When HCR_EL2.E2H == 1:

ASID value to match. Any TLB entries that match the ASID value and VA value will be affected by this System instruction.~~operation.~~

Global TLB entries that match the VA value will be affected by this System instruction~~operation~~, regardless of the value of the ASID field.

If the implementation supports 16 bits of ASID, but only 8 bits are being used in the context being invalidated, the upper bits are RES0.

Otherwise:

Reserved, RES0.

TG, bits [47:46]

Translation granule size.

TG	Meaning
0b00	Reserved.
0b01	4K translation granule.
0b10	16K translation granule.
0b11	64K translation granule.

The instruction takes a translation granule size for the translations that are being invalidated. If the translations used a different translation granule size than the one being specified, then the architecture does not require that the instruction invalidates any entries.

SCALE, bits [45:44]

The exponent element of the calculation that is used to produce the upper range.

NUM, bits [43:39]

The base element of the calculation that is used to produce the upper range.

TTL, bits [38:37]

TTL Level hint. The TTL hint is only guaranteed to invalidate entries in the range that match the level described by the TTL hint.

TTL	Meaning
0b00	The entries in the range can be using any level for the translation table entries.
0b01	When using a 4KB or 64KB translation granule, all entries to invalidate are Level 1 translation table entries. When using a 16KB translation granule, this value is reserved and hardware should treat this field as 0b00.
0b10	All entries to invalidate are Level 2 translation table entries.
0b11	All entries to invalidate are Level 3 translation table entries.

BaseADDR, bits [36:0]

The starting address for the range of the maintenance instruction.

When using a 4KB translation granule, this field is BaseADDR[48:12].

When using a 16KB translation granule, this field is BaseADDR[50:14].

When using a 64KB translation granule, this field is BaseADDR[52:16].

Executing the TLBI RVALE2 instruction

Accesses to this instruction use the following encodings:

```
TLBI RVALE2{, <Xt>}
```

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b01	0b100	0b1000	0b0110	0b101
~~0b01~~	~~0b1000~~	~~0b100~~	~~0b101~~	~~0b0110~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    TLBI_RVALE2(X[t]);
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        UNDEFINED;
    else
        TLBI_RVALE2(X[t]);
```

27~~13~~/03~~12~~/2019~~2018~~ 21~~16~~:59~~42~~; e5e4db499bf9867a4b93324c4dbac985d3da93766~~379d01e197f1d40720d32d0f84e419e9187e009~~

(old) htmldiff from- (new)

TLBI RVALE2IS, TLB Range Invalidate by VA, Last level, EL2, Inner Shareable

The TLBI RVALE2IS characteristics are:

Purpose

When EL2 is implemented and enabled in the current Security state, invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 1 translation table entry.~~entry, from the final level of the translation table walk.~~
- ~~The entry is a global entry from the final level of lookup.~~
- ~~The entry is a non-global entry from the final level of lookup that matches the specified ASID.~~

  The entry would be used to translate the specified VA~~,~~ in~~and~~ the~~one~~ specified~~of~~ range determined by the formula~~following~~ [BaseADDR <= VA < BaseADDR + ((NUM +1)*2^(5*SCALE +1) * Translation_Granule_Size)] using the EL2 or EL2&0 translation regime.~~applies:~~
- If~~When EL2 is implemented and enabled in the current Security state:~~ HCR_EL2.E2H == 0, the entry is from the final level of the translation table walk.
- If HCR_EL2.~~{~~E2H~~, ==TGE}~~ ~~is not {~~1, one~~1},~~ of~~the~~ ~~entry would be used with~~ the followingcurrent applies:~~VMID and would be required to translate the specified VA using the EL2 translation regime.~~
  - The entry is a global entry from the final level of translation table walk.
  - The entry is a non-global entry from the final level of the translation table walk and matches the specified ASID.
- ~~If HCR_EL2.{E2H, TGE} is {1, 1}, the entry would be required to translate the specified VA using the EL2&0 translation regime.~~
- ~~The entry is within the address range determined by the formula [BaseADDR <= VA < BaseADDR + ((NUM +1)*2^(5*SCALE +1) * Translation_Granule_Size)].~~

The invalidation ~~only~~ applies to all PEs in the same Inner Shareable shareability domain as the PE that executes this System instruction.

The range of addresses invalidated is UNPREDICTABLE when:

- For the 4K translation granule:
  - If TTL==01 and BaseADDR[29:12] is not equal to 000000000000000000.
  - If TTL==10 and BaseADDR[20:12] is not equal to 000000000.
- For the 16K translation granule:
  - If TTL==10 and BaseADDR[24:14] is not equal to 00000000000.
- For the 64K translation granule:
  - If TTL==01 and BaseADDR[41:16] is not equal to 000000000000000000000000000.
  - If TTL==10 and BaseADDR[28:16] is not equal to 0000000000000.

~~For more information about the architectural requirements for this instruction see 'Invalidation of TLB entries from stage 2 translations' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.~~

Configuration

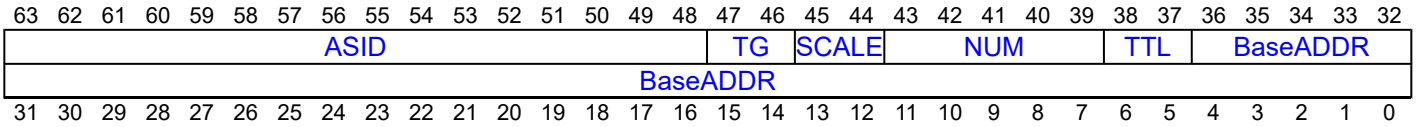
This instruction is present only when ARMv8.4-TLBI is implemented. Otherwise, direct accesses to TLBI RVALE2IS are UNDEFINED.

Attributes

TLBI RVALE2IS is a 64-bit System instruction.

Field descriptions

The TLBI RVALE2IS input value bit assignments are:



ASID, bits [63:48]

When HCR_EL2.E2H == 1:

ASID value to match. Any TLB entries that match the ASID value and VA value will be affected by this System instruction.~~operation.~~

Global TLB entries that match the VA value will be affected by this System instruction~~operation~~, regardless of the value of the ASID field.

If the implementation supports 16 bits of ASID, but only 8 bits are being used in the context being invalidated, the upper bits are RES0.

Otherwise:

Reserved, RES0.

TG, bits [47:46]

Translation granule size.

TG	Meaning
0b00	Reserved.
0b01	4K translation granule.
0b10	16K translation granule.
0b11	64K translation granule.

The instruction takes a translation granule size for the translations that are being invalidated. If the translations used a different translation granule size than the one being specified, then the architecture does not require that the instruction invalidates any entries.

SCALE, bits [45:44]

The exponent element of the calculation that is used to produce the upper range.

NUM, bits [43:39]

The base element of the calculation that is used to produce the upper range.

TTL, bits [38:37]

TTL Level hint. The TTL hint is only guaranteed to invalidate entries in the range that match the level described by the TTL hint.

TTL	Meaning
0b00	The entries in the range can be using any level for the translation table entries.
0b01	When using a 4KB or 64KB translation granule, all entries to invalidate are Level 1 translation table entries. When using a 16KB translation granule, this value is reserved and hardware should treat this field as 0b00.
0b10	All entries to invalidate are Level 2 translation table entries.
0b11	All entries to invalidate are Level 3 translation table entries.

BaseADDR, bits [36:0]

The starting address for the range of the maintenance instruction.

When using a 4KB translation granule, this field is BaseADDR[48:12].

When using a 16KB translation granule, this field is BaseADDR[50:14].

When using a 64KB translation granule, this field is BaseADDR[52:16].

Executing the TLBI RVALE2IS instruction

Accesses to this instruction use the following encodings:

```
TLBI RVALE2IS{, <Xt>}
```

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b01	0b100	0b1000	0b0010	0b101
~~0b01~~	~~0b1000~~	~~0b100~~	~~0b101~~	~~0b0010~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    TLBI_RVALE2IS(X[t]);
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        UNDEFINED;
    else
        TLBI_RVALE2IS(X[t]);
```

27~~13~~/03~~12~~/2019~~2018~~ 21~~16~~:59~~42~~; e5e4db499bf9867a4b93324c4dbac985d3da9376~~6379d01e197f1d40720d32d0f84c419e9187c009~~

(old) htmldiff from- (new)

(old) htmldiff from- (new)

TLBI RVALE2OS, TLB Range Invalidate by VA, Last level, EL2, Outer Shareable

The TLBI RVALE2OS characteristics are:

Purpose

When EL2 is implemented and enabled in the current Security state, invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 1 translation table entry.entry, from the final level of the translation table walk.
- ◦ The entry is a global entry from the final level of lookup.
  ◦ The entry is a non-global entry from the final level of lookup that matches the specified ASID.

  The entry would be used to translate the specified VA, inand theone specifiedof range determined by the formulafollowing [BaseADDR <= VA < BaseADDR + ((NUM +1)*2^(5*SCALE +1) * Translation_Granule_Size)] using the EL2 or EL2&0 translation regime.applies:
- IfWhen EL2 is implemented and enabled in the current Security state: HCR_EL2.E2H == 0, the entry is from the final level of the translation table walk.
- If HCR_EL2..{E2H, ==TGE} is not {1, one1}, ofthe entry would be used with the followingcurrent applies:VMID and would be required to translate the specified VA using the EL2 translation regime.
  ◦ The entry is a global entry from the final level of translation table walk.
  ◦ The entry is a non-global entry from the final level of the translation table walk and matches the specified ASID.
- If HCR_EL2.{E2H, TGE} is {1, 1}, the entry would be required to translate the specified VA using the EL2&0 translation regime.
- The entry is within the address range determined by the formula [BaseADDR <= VA < BaseADDR + ((NUM +1)*2^(5*SCALE +1) * Translation_Granule_Size)].

The invalidation only applies to all PEs in the same Outer Shareable shareability domain as the PE that executes this System instruction.

The range of addresses invalidated is UNPREDICTABLE when:

- For the 4K translation granule:
  ◦ If TTL==01 and BaseADDR[29:12] is not equal to 000000000000000000.
  ◦ If TTL==10 and BaseADDR[20:12] is not equal to 000000000.
- For the 16K translation granule:
  ◦ If TTL==10 and BaseADDR[24:14] is not equal to 00000000000.
- For the 64K translation granule:
  ◦ If TTL==01 and BaseADDR[41:16] is not equal to 000000000000000000000000000.
  ◦ If TTL==10 and BaseADDR[28:16] is not equal to 0000000000000.

For more information about the architectural requirements for this instruction see 'Invalidation of TLB entries from stage 2 translations' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

Configuration

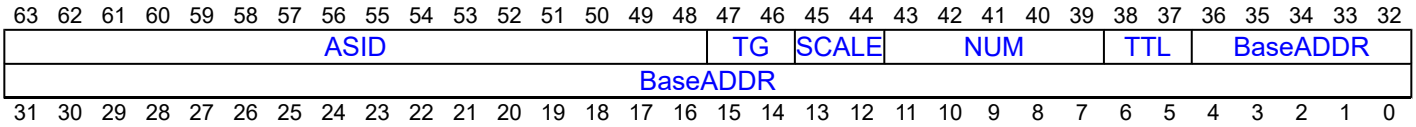
This instruction is present only when ARMv8.4-TLBI is implemented. Otherwise, direct accesses to TLBI RVALE2OS are UNDEFINED.

Attributes

TLBI RVALE2OS is a 64-bit System instruction.

Field descriptions

The TLBI RVALE2OS input value bit assignments are:



ASID, bits [63:48]

When HCR_EL2.E2H == 1:

ASID value to match. Any TLB entries that match the ASID value and VA value will be affected by this System instruction.~~operation.~~

Global TLB entries that match the VA value will be affected by this System instruction~~operation~~, regardless of the value of the ASID field.

If the implementation supports 16 bits of ASID, but only 8 bits are being used in the context being invalidated, the upper bits are RES0.

Otherwise:

Reserved, RES0.

TG, bits [47:46]

Translation granule size.

TG	Meaning
0b00	Reserved.
0b01	4K translation granule.
0b10	16K translation granule.
0b11	64K translation granule.

The instruction takes a translation granule size for the translations that are being invalidated. If the translations used a different translation granule size than the one being specified, then the architecture does not require that the instruction invalidates any entries.

SCALE, bits [45:44]

The exponent element of the calculation that is used to produce the upper range.

NUM, bits [43:39]

The base element of the calculation that is used to produce the upper range.

TTL, bits [38:37]

TTL Level hint. The TTL hint is only guaranteed to invalidate entries in the range that match the level described by the TTL hint.

TTL	Meaning
0b00	The entries in the range can be using any level for the translation table entries.
0b01	When using a 4KB or 64KB translation granule, all entries to invalidate are Level 1 translation table entries. When using a 16KB translation granule, this value is reserved and hardware should treat this field as 0b00.
0b10	All entries to invalidate are Level 2 translation table entries.
0b11	All entries to invalidate are Level 3 translation table entries.

BaseADDR, bits [36:0]

The starting address for the range of the maintenance instruction.

When using a 4KB translation granule, this field is BaseADDR[48:12].

When using a 16KB translation granule, this field is BaseADDR[50:14].

When using a 64KB translation granule, this field is BaseADDR[52:16].

Executing the TLBI RVALE2OS instruction

Accesses to this instruction use the following encodings:

```
TLBI RVALE2OS{, <Xt>}
```

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b01	0b100	0b1000	0b0101	0b101
~~0b01~~	~~0b1000~~	~~0b100~~	~~0b101~~	~~0b0101~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    TLBI_RVALE2OS(X[t]);
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        UNDEFINED;
    else
        TLBI_RVALE2OS(X[t]);
```

27~~13~~/03~~12~~/2019~~2018~~ 21~~16~~:59~~42~~; e5e4db499bf9867a4b93324c4dbac985d3da93766~~379d01e197f1d40720d32d0f84c419e9187c009~~

(old) htmldiff from- (new)

TLBI RVALE3, TLB Range Invalidate by VA, Last level, EL3

The TLBI RVALE3 characteristics are:

Purpose

If EL3 is implemented, invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 1 translation table entry, from any level of the translation table walk.
- The entry would be used to translate the specified VA using the EL3 translation regime.
- The entry is within the address range determined by the formula [BaseADDR <= VA < BaseADDR + ((NUM +1)*2^(5*SCALE +1) * Translation_Granule_Size)].

The invalidation ~~only~~ applies to the PE that executes this System instruction.

The range of addresses invalidated is UNPREDICTABLE when:

- For the 4K translation granule:
  - If TTL==01 and BaseADDR[29:12] is not equal to 000000000000000000.
  - If TTL==10 and BaseADDR[20:12] is not equal to 000000000.
- For the 16K translation granule:
  - If TTL==10 and BaseADDR[24:14] is not equal to 00000000000.
- For the 64K translation granule:
  - If TTL==01 and BaseADDR[41:16] is not equal to 000000000000000000000000000000.
  - If TTL==10 and BaseADDR[28:16] is not equal to 0000000000000.

~~For more information about the architectural requirements for this instruction see 'Invalidation of TLB entries from stage 2 translations' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.~~

Configuration

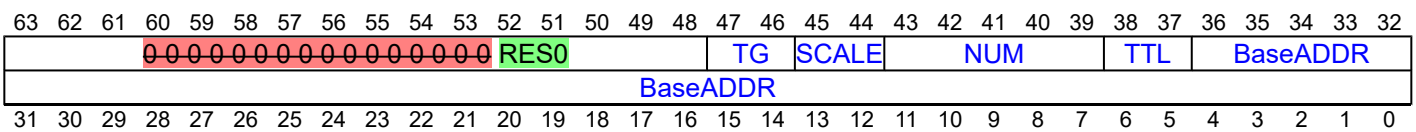
This instruction is present only when ARMv8.4-TLBI is implemented. Otherwise, direct accesses to TLBI RVALE3 are UNDEFINED.

Attributes

TLBI RVALE3 is a 64-bit System instruction.

Field descriptions

The TLBI RVALE3 input value bit assignments are:



Bits [63:48]

Reserved, RES0.

TG, bits [47:46]

Translation granule size.

TG	Meaning
0b00	Reserved.
0b01	4K translation granule.
0b10	16K translation granule.
0b11	64K translation granule.

The instruction takes a translation granule size for the translations that are being invalidated. If the translations used a different translation granule size than the one being specified, then the architecture does not require that the instruction invalidates any entries.

SCALE, bits [45:44]

The exponent element of the calculation that is used to produce the upper range.

NUM, bits [43:39]

The base element of the calculation that is used to produce the upper range.

TTL, bits [38:37]

TTL Level hint. The TTL hint is only guaranteed to invalidate entries in the range that match the level described by the TTL hint.

TTL	Meaning
0b00	The entries in the range can be using any level for the translation table entries.
0b01	When using a 4KB or 64KB translation granule, all entries to invalidate are Level 1 translation table entries. When using a 16KB translation granule, this value is reserved and hardware should treat this field as 0b00.
0b10	All entries to invalidate are Level 2 translation table entries.
0b11	All entries to invalidate are Level 3 translation table entries.

BaseADDR, bits [36:0]

The starting address for the range of the maintenance instruction.

When using a 4KB translation granule, this field is BaseADDR[48:12].

When using a 16KB translation granule, this field is BaseADDR[50:14].

When using a 64KB translation granule, this field is BaseADDR[52:16].

Executing the TLBI RVALE3 instruction

Accesses to this instruction use the following encodings:

```
TLBI RVALE3{, <Xt>}
```

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b01	0b110	0b1000	0b0110	0b101
~~0b01~~	~~0b1000~~	~~0b110~~	~~0b101~~	~~0b0110~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    TLBI_RVALE3(X[t]);
```

27~~13~~/03~~12~~/2019~~2018~~ 21~~16~~:59~~42~~: e5e4db499bf9867a4b93324c4dbac985d3da93766~~379d01e197f1d40720d32d0f84e419e9187e009~~

(old) htmldiff from- (new)

(old) htmldiff from- (new)

TLBI RVALE3IS, TLB Range Invalidate by VA, Last level, EL3, Inner Shareable

The TLBI RVALE3IS characteristics are:

Purpose

If EL3 is implemented, invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 1 translation table entry, from any level of the translation table walk.
- The entry would be used to translate the specified VA using the EL3 translation regime.
- The entry is within the address range determined by the formula [BaseADDR <= VA < BaseADDR + ((NUM +1)*2^(5*SCALE +1) * Translation_Granule_Size)].

The invalidation ~~only~~ applies to all PEs in the same Inner Shareable shareability domain as the PE that executes this System instruction.

The range of addresses invalidated is UNPREDICTABLE when:

- For the 4K translation granule:
  - If TTL==01 and BaseADDR[29:12] is not equal to 000000000000000000.
  - If TTL==10 and BaseADDR[20:12] is not equal to 000000000.
- For the 16K translation granule:
  - If TTL==10 and BaseADDR[24:14] is not equal to 00000000000.
- For the 64K translation granule:
  - If TTL==01 and BaseADDR[41:16] is not equal to 000000000000000000000000000000.
  - If TTL==10 and BaseADDR[28:16] is not equal to 0000000000000.

~~For more information about the architectural requirements for this instruction see 'Invalidation of TLB entries from stage 2 translations' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.~~

Configuration

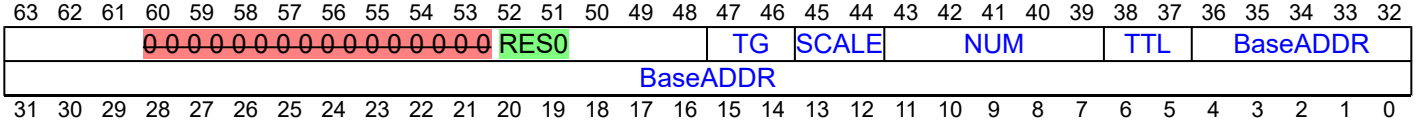
This instruction is present only when ARMv8.4-TLBI is implemented. Otherwise, direct accesses to TLBI RVALE3IS are UNDEFINED.

Attributes

TLBI RVALE3IS is a 64-bit System instruction.

Field descriptions

The TLBI RVALE3IS input value bit assignments are:



Bits [63:48]

Reserved, RES0.

TG, bits [47:46]

Translation granule size.

TG	Meaning
0b00	Reserved.
0b01	4K translation granule.
0b10	16K translation granule.
0b11	64K translation granule.

The instruction takes a translation granule size for the translations that are being invalidated. If the translations used a different translation granule size than the one being specified, then the architecture does not require that the instruction invalidates any entries.

SCALE, bits [45:44]

The exponent element of the calculation that is used to produce the upper range.

NUM, bits [43:39]

The base element of the calculation that is used to produce the upper range.

TTL, bits [38:37]

TTL Level hint. The TTL hint is only guaranteed to invalidate entries in the range that match the level described by the TTL hint.

TTL	Meaning
0b00	The entries in the range can be using any level for the translation table entries.
0b01	When using a 4KB or 64KB translation granule, all entries to invalidate are Level 1 translation table entries. When using a 16KB translation granule, this value is reserved and hardware should treat this field as 0b00.
0b10	All entries to invalidate are Level 2 translation table entries.
0b11	All entries to invalidate are Level 3 translation table entries.

BaseADDR, bits [36:0]

The starting address for the range of the maintenance instruction.

When using a 4KB translation granule, this field is BaseADDR[48:12].

When using a 16KB translation granule, this field is BaseADDR[50:14].

When using a 64KB translation granule, this field is BaseADDR[52:16].

Executing the TLBI RVALE3IS instruction

Accesses to this instruction use the following encodings:

```
TLBI RVALE3IS{, <Xt>}
```

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b01	0b110	0b1000	0b0010	0b101
~~0b01~~	~~0b1000~~	~~0b110~~	~~0b101~~	~~0b0010~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    TLBI_RVALE3IS(X[t]);
```

27~~13~~/03~~12~~/2019~~2018~~ 21~~16~~:59~~42~~: e5e4db499bf9867a4b93324c4dbac985d3da93766~~379d01e197f1d40720d32d0f84e419e9187e009~~

(old) htmldiff from- (new)

(old) htmldiff from- (new)

TLBI RVALE3OS, TLB Range Invalidate by VA, Last level, EL3, Outer Shareable

The TLBI RVALE3OS characteristics are:

Purpose

If EL3 is implemented, invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 1 translation table entry, from any level of the translation table walk.
- The entry would be used to translate the specified VA using the EL3 translation regime.
- The entry is within the address range determined by the formula [BaseADDR <= VA < BaseADDR + ((NUM +1)*2^(5*SCALE +1) * Translation_Granule_Size)].

The invalidation ~~only~~ applies to all PEs in the same Outer Shareable shareability domain as the PE that executes this System instruction.

The range of addresses invalidated is UNPREDICTABLE when:

- For the 4K translation granule:
  - If TTL==01 and BaseADDR[29:12] is not equal to 000000000000000000.
  - If TTL==10 and BaseADDR[20:12] is not equal to 000000000.
- For the 16K translation granule:
  - If TTL==10 and BaseADDR[24:14] is not equal to 00000000000.
- For the 64K translation granule:
  - If TTL==01 and BaseADDR[41:16] is not equal to 000000000000000000000000000000.
  - If TTL==10 and BaseADDR[28:16] is not equal to 0000000000000.

~~For more information about the architectural requirements for this instruction see 'Invalidation of TLB entries from stage 2 translations' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.~~

Configuration

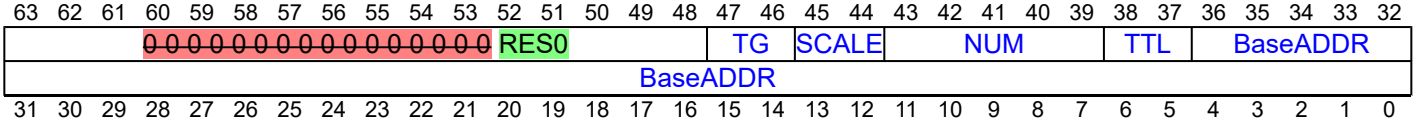
This instruction is present only from Armv8.4, or if ARMv8.4-TLBI is implemented. Otherwise, direct accesses to TLBI RVALE3OS are UNDEFINED.

Attributes

TLBI RVALE3OS is a 64-bit System instruction.

Field descriptions

The TLBI RVALE3OS input value bit assignments are:



Bits [63:48]

Reserved, RES0.

TG, bits [47:46]

Translation granule size.

TG	Meaning
0b00	Reserved.
0b01	4K translation granule.
0b10	16K translation granule.
0b11	64K translation granule.

The instruction takes a translation granule size for the translations that are being invalidated. If the translations used a different translation granule size than the one being specified, then the architecture does not require that the instruction invalidates any entries.

SCALE, bits [45:44]

The exponent element of the calculation that is used to produce the upper range.

NUM, bits [43:39]

The base element of the calculation that is used to produce the upper range.

TTL, bits [38:37]

TTL Level hint. The TTL hint is only guaranteed to invalidate entries in the range that match the level described by the TTL hint.

TTL	Meaning
0b00	The entries in the range can be using any level for the translation table entries.
0b01	When using a 4KB or 64KB translation granule, all entries to invalidate are Level 1 translation table entries. When using a 16KB translation granule, this value is reserved and hardware should treat this field as 0b00.
0b10	All entries to invalidate are Level 2 translation table entries.
0b11	All entries to invalidate are Level 3 translation table entries.

BaseADDR, bits [36:0]

The starting address for the range of the maintenance instruction.

When using a 4KB translation granule, this field is BaseADDR[48:12].

When using a 16KB translation granule, this field is BaseADDR[50:14].

When using a 64KB translation granule, this field is BaseADDR[52:16].

Executing the TLBI RVALE3OS instruction

Accesses to this instruction use the following encodings:

```
TLBI RVALE3OS{, <Xt>}
```

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b01	0b110	0b1000	0b0101	0b101
~~0b01~~	~~0b1000~~	~~0b110~~	~~0b101~~	~~0b0101~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    TLBI_RVALE3OS(X[t]);
```

2713/0312/20192018 2116:5942: e5e4db499bf9867a4b93324c4dbac985d3da93766379d01e197f1d40720d32d0f84e419e9187e009

(old) htmldiff from- (new)

(old) htmldiff from- (new)

TLBI VAAE1, TLB Invalidate by VA, All ASID, EL1

The TLBI VAAE1 characteristics are:

Purpose

Invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 1 translation table entry, from any level of the translation table walk.
- When EL2 is implemented and enabled in the ~~current~~ Security state described by the current value of~~:~~ SCR_EL3.NS:
  - If HCR_EL2.{E2H, TGE} is not {1, 1}, the entry would be used with the current VMID and would be required to translate the specified VA using the EL1&0 translation regime.
  - If HCR_EL2.{E2H, TGE} is {1, 1}, the entry would be required to translate the specified VA using the EL2&0 translation regime.
- When EL2 is not implemented or is disabled in the current Security state, the entry would be required to translate the specified VA using the EL1&0 translation regime.

The invalidation ~~only~~ applies to the PE that executes this System instruction.

Note

For the EL1&0 and EL2&0 translation regimes, the invalidation applies to both global entries, and non-global entries with any ASID.

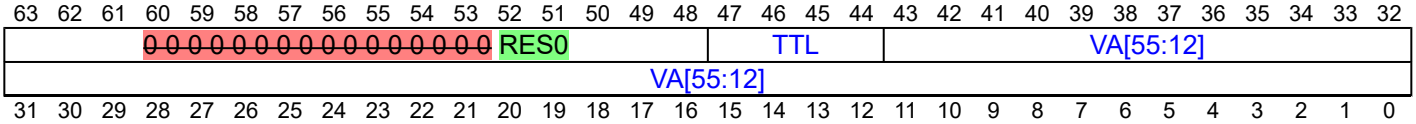
Configuration

Attributes

TLBI VAAE1 is a 64-bit System instruction.

Field descriptions

The TLBI VAAE1 input value bit assignments are:



Bits [63:48]

Reserved, RES0.

TTL, bits [47:44]

When ARMv8.4-TTL is implemented:

Translation Table Level. Indicates the level of the page table walk that holds the leaf entry for the address being invalidated.

TTL	Meaning
0b00xx	No information supplied as to the translation table level. Hardware must assume that the entry can be from any level. In this case, TTL<1:0> is RES0.
0b01xx	The entry comes from a 4KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3.
0b10xx	The entry comes from a 16KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : Reserved. Treat as if TTL<3:2> is 0b00. 0b10 : Level 2. 0b11 : Level 3.
0b11xx	The entry comes from a 64KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3.

If an incorrect value of the TTL field is specified for the entry being invalidated by the instruction, then no entries are required by the architecture to be invalidated from the TLB.

Otherwise:

Reserved, RES0.

VA[55:12], bits [43:0]

Bits[55:12] of the virtual address to match. Any appropriate TLB entries that match the VA will be affected by this System instruction~~operation~~, regardless of the ASID.

If the TLB maintenance instructions are targeting a translation regime that is using AArch32, and so has a VA of only 32 bits, then the software must treat bits[55:32] as RES0.

The treatment of the low-order bits of this field depends on the translation granule size, as follows:

- Where a 4KB translation granule is being used, all bits are valid and used for the invalidation.
- Where a 16KB translation granule is being used, bits [1:0] of this field are RES0 and ignored when the instruction is executed, because VA[13:12] have no effect on the operation of the instruction.
- Where a 64KB translation granule is being used, bits [3:0] of this field are RES0 and ignored when the instruction is executed, because VA[15:12] have no effect on the operation of the instruction.

Executing the TLBI VAAE1 instruction

Accesses to this instruction use the following encodings:

```
TLBI VAAE1{, <Xt>}
```

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b01	0b000	0b1000	0b0111	0b011
~~0b01~~	~~0b1000~~	~~0b000~~	~~0b011~~	~~0b0111~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TTLB == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.FB == '1' then
        TLBI_VAAE1IS(X[t]);
    else
        TLBI_VAAE1(X[t]);
elsif PSTATE.EL == EL2 then
    TLBI_VAAE1(X[t]);
elsif PSTATE.EL == EL3 then
    TLBI_VAAE1(X[t]);
```

27~~13~~/03~~12~~/2019~~2018~~ 21~~16~~:59~~42~~; e5e4db499bf9867a4b93324c4dbac985d3da93766~~6379d01e197f1d40720d32d0f84e419e9187e009~~

(old) htmldiff from- (new)

TLBI VAAE1IS, TLB Invalidate by VA, All ASID, EL1, Inner Shareable

The TLBI VAAE1IS characteristics are:

Purpose

Invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 1 translation table entry, from any level of the translation table walk.
- When EL2 is implemented and enabled in the ~~current~~ Security state described by the current value of~~:~~ SCR_EL3.NS:
  - If HCR_EL2.{E2H, TGE} is not {1, 1}, the entry would be used with the current VMID and would be required to translate the specified VA using the EL1&0 translation regime.
  - If HCR_EL2.{E2H, TGE} is {1, 1}, the entry would be required to translate the specified VA using the EL2&0 translation regime.
- When EL2 is not implemented or is disabled in the current Security state, the entry would be required to translate the specified VA using the EL1&0 translation regime.

The invalidation applies to all PEs in the same Inner Shareable shareability domain as the PE that executes this System instruction.~~instructions.~~

Note

From Armv8.4, when a TLB maintenance instruction is generated to the Secure EL1&0 translation regime and is defined to pass a VMID argument, or would be defined to pass a VMID argument if SCR_EL3.EEL2==1, then:

- A PE with SCR_EL3.EEL2==1 is not architecturally required to invalidate any entries in the Secure EL1&0 translation of a PE in the same required shareability domain with SCR_EL3.EEL2==0.
- A PE with SCR_EL3.EEL2==0 is not architecturally required to invalidate any entries in the Secure EL1&0 translation of a PE in the same required shareability domain with SCR_EL3.EEL2==1.
- A PE is architecturally required to invalidate all relevant entries in the Secure EL1&0 translation of a System MMU in the same required shareability domain with a VMID of 0.

For the EL1&0 and EL2&0 translation regimes, the invalidation applies to both global entries, and non-global entries with any ASID.

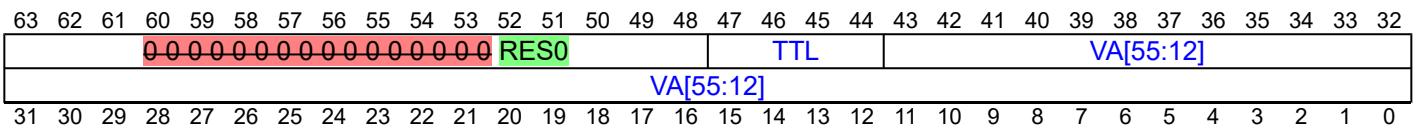
Configuration

Attributes

TLBI VAAE1IS is a 64-bit System instruction.

Field descriptions

The TLBI VAAE1IS input value bit assignments are:



Bits [63:48]

Reserved, RES0.

TTL, bits [47:44]

When ARMv8.4-TTL is implemented:

Translation Table Level. Indicates the level of the page table walk that holds the leaf entry for the address being invalidated.

TTL	Meaning
0b00xx	No information supplied as to the translation table level. Hardware must assume that the entry can be from any level. In this case, TTL<1:0> is RES0.
0b01xx	The entry comes from a 4KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3.
0b10xx	The entry comes from a 16KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : Reserved. Treat as if TTL<3:2> is 0b00. 0b10 : Level 2. 0b11 : Level 3.
0b11xx	The entry comes from a 64KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3.

If an incorrect value of the TTL field is specified for the entry being invalidated by the instruction, then no entries are required by the architecture to be invalidated from the TLB.

Otherwise:

Reserved, RES0.

VA[55:12], bits [43:0]

Bits[55:12] of the virtual address to match. Any appropriate TLB entries that match the VA will be affected by this System instruction~~operation~~, regardless of the ASID.

If the TLB maintenance instructions are targeting a translation regime that is using AArch32, and so has a VA of only 32 bits, then the software must treat bits[55:32] as RES0.

The treatment of the low-order bits of this field depends on the translation granule size, as follows:

- Where a 4KB translation granule is being used, all bits are valid and used for the invalidation.
- Where a 16KB translation granule is being used, bits [1:0] of this field are RES0 and ignored when the instruction is executed, because VA[13:12] have no effect on the operation of the instruction.
- Where a 64KB translation granule is being used, bits [3:0] of this field are RES0 and ignored when the instruction is executed, because VA[15:12] have no effect on the operation of the instruction.

Executing the TLBI VAAE1IS instruction

Accesses to this instruction use the following encodings:

```
TLBI VAAE1IS{, <Xt>}
```

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b01	0b000	0b1000	0b0011	0b011
~~0b01~~	~~0b1000~~	~~0b000~~	~~0b011~~	~~0b0011~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TTLB == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TTLBIS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        TLBI_VAAE1IS(X[t]);
elsif PSTATE.EL == EL2 then
    TLBI_VAAE1IS(X[t]);
elsif PSTATE.EL == EL3 then
    TLBI_VAAE1IS(X[t]);
```

27~~13~~/03~~12~~/2019~~2018~~ 21~~16~~:59~~42~~: e5e4db499bf9867a4b93324c4dbac985d3da93766379d01e197f1d40720d32d0f84e419e9187e009

(old) htmldiff from- (new)

TLBI VAAE1OS, TLB Invalidate by VA, All ASID, EL1, Outer Shareable

The TLBI VAAE1OS characteristics are:

Purpose

Invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 1 translation table entry, from any level of the translation table walk.
- When EL2 is implemented and enabled in the ~~current~~ Security state described by the current value of~~:~~ SCR_EL3.NS:
  - If HCR_EL2.{E2H, TGE} is not {1, 1}, the entry would be used with the current VMID and would be required to translate the specified VA using the EL1&0 translation regime.
  - If HCR_EL2.{E2H, TGE} is {1, 1}, the entry would be required to translate the specified VA using the EL2&0 translation regime.
- When EL2 is not implemented or is disabled in the current Security state, the entry would be required to translate the specified VA using the EL1&0 translation regime.

The invalidation applies to all PEs in the same Outer Shareable shareability domain as the PE that executes this System instruction.~~instructions.~~

Note

When a TLB maintenance instruction is generated to the Secure EL1&0 translation regime and is defined to pass a VMID argument, or would be defined to pass a VMID argument if SCR_EL3.EEL2==1, then:

- A PE with SCR_EL3.EEL2==1 is not architecturally required to invalidate any entries in the Secure EL1&0 translation of a PE in the same required shareability domain with SCR_EL3.EEL2==0.
- A PE with SCR_EL3.EEL2==0 is not architecturally required to invalidate any entries in the Secure EL1&0 translation of a PE in the same required shareability domain with SCR_EL3.EEL2==1.
- A PE is architecturally required to invalidate all relevant entries in the Secure EL1&0 translation of a System MMU in the same required shareability domain with a VMID of 0.

For the EL1&0 and EL2&0 translation regimes, the invalidation applies to both global entries, and non-global entries with any ASID.

Configuration

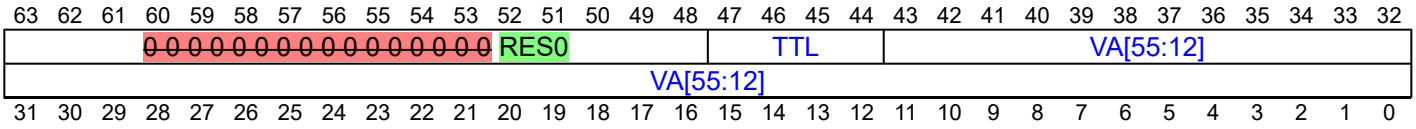
This instruction is present only from Armv8.4, or if ARMv8.4-TLBI is implemented. Otherwise, direct accesses to TLBI VAAE1OS are UNDEFINED.

Attributes

TLBI VAAE1OS is a 64-bit System instruction.

Field descriptions

The TLBI VAAE1OS input value bit assignments are:



Bits [63:48]

Reserved, RES0.

TTL, bits [47:44]

When ARMv8.4-TTL is implemented:

Translation Table Level. Indicates the level of the page table walk that holds the leaf entry for the address being invalidated.

TTL	Meaning
0b00xx	No information supplied as to the translation table level. Hardware must assume that the entry can be from any level. In this case, TTL<1:0> is RES0.
0b01xx	The entry comes from a 4KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3.
0b10xx	The entry comes from a 16KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : Reserved. Treat as if TTL<3:2> is 0b00. 0b10 : Level 2. 0b11 : Level 3.
0b11xx	The entry comes from a 64KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3.

If an incorrect value of the TTL field is specified for the entry being invalidated by the instruction, then no entries are required by the architecture to be invalidated from the TLB.

Otherwise:

Reserved, RES0.

VA[55:12], bits [43:0]

Bits[55:12] of the virtual address to match. Any appropriate TLB entries that match the VA will be affected by this System instruction~~operation~~, regardless of the ASID.

If the TLB maintenance instructions are targeting a translation regime that is using AArch32, and so has a VA of only 32 bits, then the software must treat bits[55:32] as RES0.

The treatment of the low-order bits of this field depends on the translation granule size, as follows:

- Where a 4KB translation granule is being used, all bits are valid and used for the invalidation.
- Where a 16KB translation granule is being used, bits [1:0] of this field are RES0 and ignored when the instruction is executed, because VA[13:12] have no effect on the operation of the instruction.
- Where a 64KB translation granule is being used, bits [3:0] of this field are RES0 and ignored when the instruction is executed, because VA[15:12] have no effect on the operation of the instruction.

Executing the TLBI VAAE1OS instruction

Accesses to this instruction use the following encodings:

```
TLBI VAAE1OS{, <Xt>}
```

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b01	0b000	0b1000	0b0001	0b011
~~0b01~~	~~0b1000~~	~~0b000~~	~~0b011~~	~~0b0001~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TTLB == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TTLBOS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        TLBI_VAAE1OS(X[t]);
elsif PSTATE.EL == EL2 then
    TLBI_VAAE1OS(X[t]);
elsif PSTATE.EL == EL3 then
    TLBI_VAAE1OS(X[t]);
```

27~~13~~/03~~12~~/2019~~2018~~ 21~~16~~:59~~42~~; e5e4db499bf9867a4b93324c4dbac985d3da9376~~6379d01e197f1d40720d32d0f84c419e9187e009~~

(old) htmldiff from- (new)

TLBI VAALE1, TLB Invalidate by VA, All ASID, Last level, EL1

The TLBI VAALE1 characteristics are:

Purpose

Invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 1 translation table entry, from the final level of the translation table walk.
- When EL2 is implemented and enabled in the ~~current~~ Security state described by the current value of~~:~~ SCR_EL3.NS:
  - If HCR_EL2.{E2H, TGE} is not {1, 1}, the entry would be used with the current VMID and would be required to translate the specified VA using the EL1&0 translation regime.
  - If HCR_EL2.{E2H, TGE} is {1, 1}, the entry would be required to translate the specified VA using the EL2&0 translation regime.
- When EL2 is not implemented or is disabled in the current Security state, the entry would be required to translate the specified VA using the EL1&0 translation regime.

The invalidation ~~only~~ applies to the PE that executes this System instruction.

Note

For the EL1&0 and EL2&0 translation regimes, the invalidation applies to both global entries, and non-global entries with any ASID.

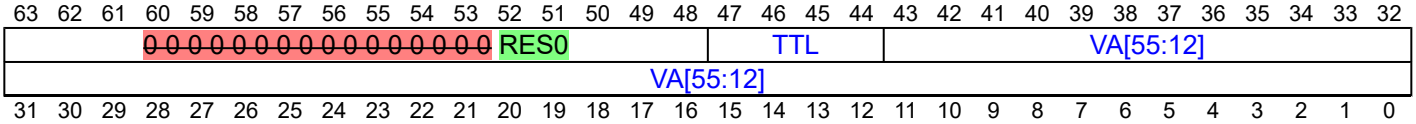
Configuration

Attributes

TLBI VAALE1 is a 64-bit System instruction.

Field descriptions

The TLBI VAALE1 input value bit assignments are:



Bits [63:48]

Reserved, RES0.

TTL, bits [47:44]

When ARMv8.4-TTL is implemented:

Translation Table Level. Indicates the level of the page table walk that holds the leaf entry for the address being invalidated.

TTL	Meaning
0b00xx	No information supplied as to the translation table level. Hardware must assume that the entry can be from any level. In this case, TTL<1:0> is RES0.
0b01xx	The entry comes from a 4KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3.
0b10xx	The entry comes from a 16KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : Reserved. Treat as if TTL<3:2> is 0b00. 0b10 : Level 2. 0b11 : Level 3.
0b11xx	The entry comes from a 64KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3.

If an incorrect value of the TTL field is specified for the entry being invalidated by the instruction, then no entries are required by the architecture to be invalidated from the TLB.

Otherwise:

Reserved, RES0.

VA[55:12], bits [43:0]

Bits[55:12] of the virtual address to match. Any appropriate TLB entries that match the VA will be affected by this System instruction~~operation~~, regardless of the ASID.

If the TLB maintenance instructions are targeting a translation regime that is using AArch32, and so has a VA of only 32 bits, then the software must treat bits[55:32] as RES0.

The treatment of the low-order bits of this field depends on the translation granule size, as follows:

- Where a 4KB translation granule is being used, all bits are valid and used for the invalidation.
- Where a 16KB translation granule is being used, bits [1:0] of this field are RES0 and ignored when the instruction is executed, because VA[13:12] have no effect on the operation of the instruction.
- Where a 64KB translation granule is being used, bits [3:0] of this field are RES0 and ignored when the instruction is executed, because VA[15:12] have no effect on the operation of the instruction.

Executing the TLBI VAALE1 instruction

Accesses to this instruction use the following encodings:

```
TLBI VAALE1{, <Xt>}
```

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b01	0b000	0b1000	0b0111	0b111
~~0b01~~	~~0b1000~~	~~0b000~~	~~0b111~~	~~0b0111~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TTLB == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.FB == '1' then
        TLBI_VAALE1IS(X[t]);
    else
        TLBI_VAALE1(X[t]);
elsif PSTATE.EL == EL2 then
    TLBI_VAALE1(X[t]);
elsif PSTATE.EL == EL3 then
    TLBI_VAALE1(X[t]);
```

27~~13~~/03~~12~~/2019~~2018~~ 21~~16~~:59~~42~~; e5e4db499bf9867a4b93324c4dbac985d3da93766~~379d01e197f1d40720d32d0f84e419e9187e009~~

(old) htmldiff from- (new)

(old) htmldiff from- (new)

TLBI VAALE1IS, TLB Invalidate by VA, All ASID, Last Level, EL1, Inner Shareable

The TLBI VAALE1IS characteristics are:

Purpose

Invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 1 translation table entry, from the final level of the translation table walk.
- When EL2 is implemented and enabled in the ~~current~~ Security state described by the current value of~~:~~ SCR_EL3.NS:
  - If HCR_EL2.{E2H, TGE} is not {1, 1}, the entry would be used with the current VMID and would be required to translate the specified VA using the EL1&0 translation regime.
  - If HCR_EL2.{E2H, TGE} is {1, 1}, the entry would be required to translate the specified VA using the EL2&0 translation regime.
- When EL2 is not implemented or is disabled in the current Security state, the entry would be required to translate the specified VA using the EL1&0 translation regime.

The invalidation applies to all PEs in the same Inner Shareable shareability domain as the PE that executes this System instruction.~~instructions.~~

Note

From Armv8.4, when a TLB maintenance instruction is generated to the Secure EL1&0 translation regime and is defined to pass a VMID argument, or would be defined to pass a VMID argument if SCR_EL3.EEL2==1, then:

- A PE with SCR_EL3.EEL2==1 is not architecturally required to invalidate any entries in the Secure EL1&0 translation of a PE in the same required shareability domain with SCR_EL3.EEL2==0.
- A PE with SCR_EL3.EEL2==0 is not architecturally required to invalidate any entries in the Secure EL1&0 translation of a PE in the same required shareability domain with SCR_EL3.EEL2==1.
- A PE is architecturally required to invalidate all relevant entries in the Secure EL1&0 translation of a System MMU in the same required shareability domain with a VMID of 0.

For the EL1&0 and EL2&0 translation regimes, the invalidation applies to both global entries, and non-global entries with any ASID.

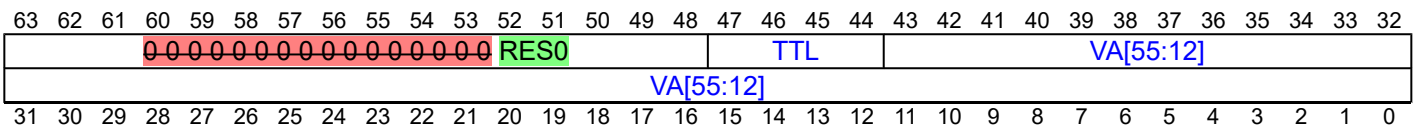
Configuration

Attributes

TLBI VAALE1IS is a 64-bit System instruction.

Field descriptions

The TLBI VAALE1IS input value bit assignments are:



Bits [63:48]

Reserved, RES0.

TTL, bits [47:44]

When ARMv8.4-TTL is implemented:

Translation Table Level. Indicates the level of the page table walk that holds the leaf entry for the address being invalidated.

TTL	Meaning
0b00xx	No information supplied as to the translation table level. Hardware must assume that the entry can be from any level. In this case, TTL<1:0> is RES0.
0b01xx	The entry comes from a 4KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3.
0b10xx	The entry comes from a 16KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : Reserved. Treat as if TTL<3:2> is 0b00. 0b10 : Level 2. 0b11 : Level 3.
0b11xx	The entry comes from a 64KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3.

If an incorrect value of the TTL field is specified for the entry being invalidated by the instruction, then no entries are required by the architecture to be invalidated from the TLB.

Otherwise:

Reserved, RES0.

VA[55:12], bits [43:0]

Bits[55:12] of the virtual address to match. Any appropriate TLB entries that match the VA will be affected by this System instruction~~operation~~, regardless of the ASID.

If the TLB maintenance instructions are targeting a translation regime that is using AArch32, and so has a VA of only 32 bits, then the software must treat bits[55:32] as RES0.

The treatment of the low-order bits of this field depends on the translation granule size, as follows:

- Where a 4KB translation granule is being used, all bits are valid and used for the invalidation.
- Where a 16KB translation granule is being used, bits [1:0] of this field are RES0 and ignored when the instruction is executed, because VA[13:12] have no effect on the operation of the instruction.
- Where a 64KB translation granule is being used, bits [3:0] of this field are RES0 and ignored when the instruction is executed, because VA[15:12] have no effect on the operation of the instruction.

Executing the TLBI VAALE1IS instruction

Accesses to this instruction use the following encodings:

```
TLBI VAALE1IS{, <Xt>}
```

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b01	0b000	0b1000	0b0011	0b111
~~0b01~~	~~0b1000~~	~~0b000~~	~~0b111~~	~~0b0011~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TTLB == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TTLBIS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        TLBI_VAALE1IS(X[t]);
elsif PSTATE.EL == EL2 then
    TLBI_VAALE1IS(X[t]);
elsif PSTATE.EL == EL3 then
    TLBI_VAALE1IS(X[t]);
```

27~~13~~/03~~12~~/2019~~2018~~ 21~~16~~:59~~42~~: e5e4db499bf9867a4b93324c4dbac985d3da93766379d01e197f1d40720d32d0f84e419e9187e009

(old) htmldiff from- (new)

TLBI VAALE1OS, TLB Invalidate by VA, All ASID, Last Level, EL1, Outer Shareable

The TLBI VAALE1OS characteristics are:

Purpose

Invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 1 translation table entry, from the final level of the translation table walk.
- When EL2 is implemented and enabled in the ~~current~~ Security state described by the current value of~~:~~ SCR_EL3.NS:
  - If HCR_EL2.{E2H, TGE} is not {1, 1}, the entry would be used with the current VMID and would be required to translate the specified VA using the EL1&0 translation regime.
  - If HCR_EL2.{E2H, TGE} is {1, 1}, the entry would be required to translate the specified VA using the EL2&0 translation regime.
- When EL2 is not implemented or is disabled in the current Security state, the entry would be required to translate the specified VA using the EL1&0 translation regime.

The invalidation applies to all PEs in the same Outer Shareable shareability domain as the PE that executes this System instruction.~~instructions.~~

Note

When a TLB maintenance instruction is generated to the Secure EL1&0 translation regime and is defined to pass a VMID argument, or would be defined to pass a VMID argument if SCR_EL3.EEL2==1, then:

- A PE with SCR_EL3.EEL2==1 is not architecturally required to invalidate any entries in the Secure EL1&0 translation of a PE in the same required shareability domain with SCR_EL3.EEL2==0.
- A PE with SCR_EL3.EEL2==0 is not architecturally required to invalidate any entries in the Secure EL1&0 translation of a PE in the same required shareability domain with SCR_EL3.EEL2==1.
- A PE is architecturally required to invalidate all relevant entries in the Secure EL1&0 translation of a System MMU in the same required shareability domain with a VMID of 0.

For the EL1&0 and EL2&0 translation regimes, the invalidation applies to both global entries, and non-global entries with any ASID.

Configuration

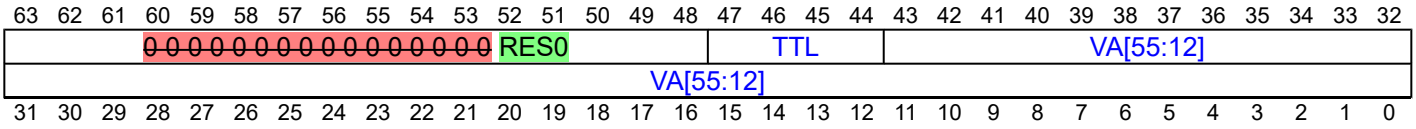
This instruction is present only from Armv8.4, or if ARMv8.4-TLBI is implemented. Otherwise, direct accesses to TLBI VAALE1OS are UNDEFINED.

Attributes

TLBI VAALE1OS is a 64-bit System instruction.

Field descriptions

The TLBI VAALE1OS input value bit assignments are:



Bits [63:48]

Reserved, RES0.

TTL, bits [47:44]

When ARMv8.4-TTL is implemented:

Translation Table Level. Indicates the level of the page table walk that holds the leaf entry for the address being invalidated.

TTL	Meaning
0b00xx	No information supplied as to the translation table level. Hardware must assume that the entry can be from any level. In this case, TTL<1:0> is RES0.
0b01xx	The entry comes from a 4KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3.
0b10xx	The entry comes from a 16KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : Reserved. Treat as if TTL<3:2> is 0b00. 0b10 : Level 2. 0b11 : Level 3.
0b11xx	The entry comes from a 64KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3.

If an incorrect value of the TTL field is specified for the entry being invalidated by the instructio, then no entries are required by the architecture to be invalidated from the TLB.

Otherwise:

Reserved, RES0.

VA[55:12], bits [43:0]

Bits[55:12] of the virtual address to match. Any appropriate TLB entries that match the VA will be affected by this System instruction~~operation~~, regardless of the ASID.

If the TLB maintenance instructions are targeting a translation regime that is using AArch32, and so has a VA of only 32 bits, then the software must treat bits[55:32] as RES0.

The treatment of the low-order bits of this field depends on the translation granule size, as follows:

- Where a 4KB translation granule is being used, all bits are valid and used for the invalidation.
- Where a 16KB translation granule is being used, bits [1:0] of this field are RES0 and ignored when the instruction is executed, because VA[13:12] have no effect on the operation of the instruction.
- Where a 64KB translation granule is being used, bits [3:0] of this field are RES0 and ignored when the instruction is executed, because VA[15:12] have no effect on the operation of the instruction.

Executing the TLBI VAALE1OS instruction

Accesses to this instruction use the following encodings:

```
TLBI VAALE1OS{, <Xt>}
```

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b01	0b000	0b1000	0b0001	0b111
~~0b01~~	~~0b1000~~	~~0b000~~	~~0b111~~	~~0b0001~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TTLB == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TTLBOS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        TLBI_VAALE1OS(X[t]);
elsif PSTATE.EL == EL2 then
    TLBI_VAALE1OS(X[t]);
elsif PSTATE.EL == EL3 then
    TLBI_VAALE1OS(X[t]);
```

27~~13~~/03~~12~~/2019~~2018~~ 21~~16~~:59~~42~~; e5e4db499bf9867a4b93324c4dbac985d3da9376~~6379d01e197f1d40720d32d0f84c419e9187e009~~

(old) htmldiff from- (new)

TLBI VAE1, TLB Invalidate by VA, EL1

The TLBI VAE1 characteristics are:

Purpose

Invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 1 translation table entry.
- The entry would be used to translate the specified VA, and one of the following applies:
  - The entry is from a level of lookup above the final level and matches the specified ASID.
  - The entry is a global entry from the final level of lookup.
  - The entry is a non-global entry from the final level of lookup that matches the specified ASID.
- When EL2 is implemented and enabled in the ~~current~~ Security state described by the current value of~~:~~ SCR_EL3.NS:
  - If HCR_EL2.{E2H, TGE} is not {1, 1}, the entry would be used with the current VMID and would be required to translate the specified VA using the EL1&0 translation regime.
  - If HCR_EL2.{E2H, TGE} is {1, 1}, the entry would be required to translate the specified VA using the EL2&0 translation regime.
- When EL2 is not implemented or is disabled in the current Security state, the entry would be required to translate the specified VA using the EL1&0 translation regime.

The invalidation ~~only~~ applies to the PE that executes this System instruction.

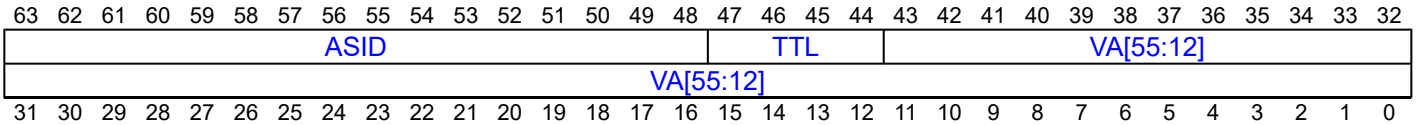
Configuration

Attributes

TLBI VAE1 is a 64-bit System instruction.

Field descriptions

The TLBI VAE1 input value bit assignments are:



ASID, bits [63:48]

ASID value to match. Any TLB entries that match the ASID value and VA value will be affected by this System instruction.~~operation.~~

Global TLB entries that match the VA value will be affected by this System instruction~~operation~~, regardless of the value of the ASID field.

If the implementation supports 16 bits of ASID, but only 8 bits are being used in the context being invalidated, the upper bits are RES0.

TTL, bits [47:44]

When ARMv8.4-TTL is implemented:

Translation Table Level. Indicates the level of the page table walk that holds the leaf entry for the address being invalidated.

TTL	Meaning
0b00xx	No information supplied as to the translation table level. Hardware must assume that the entry can be from any level. In this case, TTL<1:0> is RES0.
0b01xx	The entry comes from a 4KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3.
0b10xx	The entry comes from a 16KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : Reserved. Treat as if TTL<3:2> is 0b00. 0b10 : Level 2. 0b11 : Level 3.
0b11xx	The entry comes from a 64KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3.

If an incorrect value of the TTL field is specified for the entry being invalidated by the instruction, then no entries are required by the architecture to be invalidated from the TLB.

Otherwise:

Reserved, RES0.

VA[55:12], bits [43:0]

Bits[55:12] of the virtual address to match. Any appropriate TLB entries that match the ASID value (if appropriate) and VA will be affected by this System instruction.~~operation.~~

If the TLB maintenance instructions are targeting a translation regime that is using AArch32, and so has a VA of only 32 bits, then the software must treat bits[55:32] as RES0.

The treatment of the low-order bits of this field depends on the translation granule size, as follows:

- Where a 4KB translation granule is being used, all bits are valid and used for the invalidation.
- Where a 16KB translation granule is being used, bits [1:0] of this field are RES0 and ignored when the instruction is executed, because VA[13:12] have no effect on the operation of the instruction.
- Where a 64KB translation granule is being used, bits [3:0] of this field are RES0 and ignored when the instruction is executed, because VA[15:12] have no effect on the operation of the instruction.

Executing the TLBI VAE1 instruction

Accesses to this instruction use the following encodings:

```
TLBI VAE1{, <Xt>}
```

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b01	0b000	0b1000	0b0111	0b001
~~0b01~~	~~0b1000~~	~~0b000~~	~~0b001~~	~~0b0111~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TTLB == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.FB == '1' then
        TLBI_VAE1IS(X[t]);
    else
        TLBI_VAE1(X[t]);
elsif PSTATE.EL == EL2 then
    TLBI_VAE1(X[t]);
elsif PSTATE.EL == EL3 then
    TLBI_VAE1(X[t]);
```

27~~13~~/03~~12~~/2019~~2018~~ 21~~16~~:59~~42~~; e5e4db499bf9867a4b93324c4dbac985d3da93766~~379d01e197f1d40720d32d0f84e419e9187e009~~

(old) htmldiff from- (new)

(old) htmldiff from- (new)

TLBI VAE1IS, TLB Invalidate by VA, EL1, Inner Shareable

The TLBI VAE1IS characteristics are:

Purpose

Invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 1 translation table entry.
- The entry would be used to translate the specified VA, and one of the following applies:
  - The entry is from a level of lookup above the final level and matches the specified ASID.
  - The entry is a global entry from the final level of lookup.
  - The entry is a non-global entry from the final level of lookup that matches the specified ASID.
- When EL2 is implemented and enabled in the ~~current~~ Security state described by the current value of~~:~~ SCR_EL3.NS:
  - If HCR_EL2.{E2H, TGE} is not {1, 1}, the entry would be used with the current VMID and would be required to translate the specified VA using the EL1&0 translation regime.
  - If HCR_EL2.{E2H, TGE} is {1, 1}, the entry would be required to translate the specified VA using the EL2&0 translation regime.
- When EL2 is not implemented or is disabled in the current Security state, the entry would be required to translate the specified VA using the EL1&0 translation regime.

The invalidation ~~only~~ applies to all PEs in the same Inner Shareable shareability domain as the PE that executes this System instruction.

Note

From Armv8.4, when a TLB maintenance instruction is generated to the Secure EL1&0 translation regime and is defined to pass a VMID argument, or would be defined to pass a VMID argument if SCR_EL3.EEL2==1, then:

- A PE with SCR_EL3.EEL2==1 is not architecturally required to invalidate any entries in the Secure EL1&0 translation of a PE in the same required shareability domain with SCR_EL3.EEL2==0.
- A PE with SCR_EL3.EEL2==0 is not architecturally required to invalidate any entries in the Secure EL1&0 translation of a PE in the same required shareability domain with SCR_EL3.EEL2==1.
- A PE is architecturally required to invalidate all relevant entries in the Secure EL1&0 translation of a System MMU in the same required shareability domain with a VMID of 0.

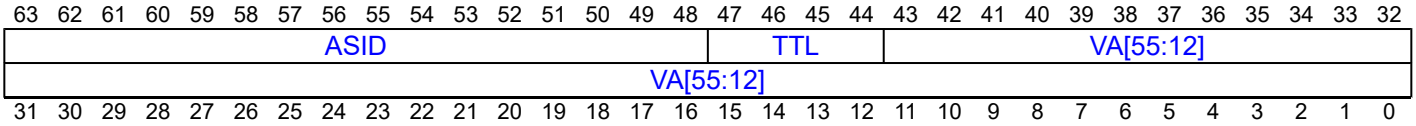
Configuration

Attributes

TLBI VAE1IS is a 64-bit System instruction.

Field descriptions

The TLBI VAE1IS input value bit assignments are:



ASID, bits [63:48]

ASID value to match. Any TLB entries that match the ASID value and VA value will be affected by this System instruction.~~operation.~~

Global TLB entries that match the VA value will be affected by this System instruction~~operation~~, regardless of the value of the ASID field.

If the implementation supports 16 bits of ASID, but only 8 bits are being used in the context being invalidated, the upper bits are RES0.

TTL, bits [47:44]

When ARMv8.4-TTL is implemented:

Translation Table Level. Indicates the level of the page table walk that holds the leaf entry for the address being invalidated.

TTL	Meaning
0b00xx	No information supplied as to the translation table level. Hardware must assume that the entry can be from any level. In this case, TTL<1:0> is RES0.
0b01xx	The entry comes from a 4KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3.
0b10xx	The entry comes from a 16KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : Reserved. Treat as if TTL<3:2> is 0b00. 0b10 : Level 2. 0b11 : Level 3.
0b11xx	The entry comes from a 64KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3.

If an incorrect value of the TTL field is specified for the entry being invalidated by the instruction, then no entries are required by the architecture to be invalidated from the TLB.

Otherwise:

Reserved, RES0.

VA[55:12], bits [43:0]

Bits[55:12] of the virtual address to match. Any appropriate TLB entries that match the ASID value (if appropriate) and VA will be affected by this System instruction.~~operation.~~

If the TLB maintenance instructions are targeting a translation regime that is using AArch32, and so has a VA of only 32 bits, then the software must treat bits[55:32] as RES0.

The treatment of the low-order bits of this field depends on the translation granule size, as follows:

- Where a 4KB translation granule is being used, all bits are valid and used for the invalidation.
- Where a 16KB translation granule is being used, bits [1:0] of this field are RES0 and ignored when the instruction is executed, because VA[13:12] have no effect on the operation of the instruction.
- Where a 64KB translation granule is being used, bits [3:0] of this field are RES0 and ignored when the instruction is executed, because VA[15:12] have no effect on the operation of the instruction.

Executing the TLBI VAE1IS instruction

Accesses to this instruction use the following encodings:

```
TLBI VAE1IS{, <Xt>}
```

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b01	0b000	0b1000	0b0011	0b001
~~0b01~~	~~0b1000~~	~~0b000~~	~~0b001~~	~~0b0011~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TTLB == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TTLBIS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        TLBI_VAE1IS(X[t]);
elsif PSTATE.EL == EL2 then
    TLBI_VAE1IS(X[t]);
elsif PSTATE.EL == EL3 then
    TLBI_VAE1IS(X[t]);
```

27~~13~~/03~~12~~/2019~~2018~~ 21~~16~~:59~~42~~: e5e4db499bf9867a4b93324c4dbac985d3da93766379d01e197f1d40720d32d0f84e419e9187e009

(old) htmldiff from- (new)

TLBI VAE1OS, TLB Invalidate by VA, EL1, Outer Shareable

The TLBI VAE1OS characteristics are:

Purpose

Invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 1 translation table entry.
- The entry would be used to translate the specified VA, and one of the following applies:
  - The entry is from a level of lookup above the final level and matches the specified ASID.
  - The entry is a global entry from the final level of lookup.
  - The entry is a non-global entry from the final level of lookup that matches the specified ASID.
- When EL2 is implemented and enabled in the ~~current~~ Security state described by the current value of~~:~~ SCR_EL3.NS:
  - If HCR_EL2.{E2H, TGE} is not {1, 1}, the entry would be used with the current VMID and would be required to translate the specified VA using the EL1&0 translation regime.
  - If HCR_EL2.{E2H, TGE} is {1, 1}, the entry would be required to translate the specified VA using the EL2&0 translation regime.
- When EL2 is not implemented or is disabled in the current Security state, the entry would be required to translate the specified VA using the EL1&0 translation regime.

The invalidation ~~only~~ applies to all PEs in the same Outer Shareable shareability domain as the PE that executes this System instruction.

Note

When a TLB maintenance instruction is generated to the Secure EL1&0 translation regime and is defined to pass a VMID argument, or would be defined to pass a VMID argument if SCR_EL3.EEL2==1, then:

- A PE with SCR_EL3.EEL2==1 is not architecturally required to invalidate any entries in the Secure EL1&0 translation of a PE in the same required shareability domain with SCR_EL3.EEL2==0.
- A PE with SCR_EL3.EEL2==0 is not architecturally required to invalidate any entries in the Secure EL1&0 translation of a PE in the same required shareability domain with SCR_EL3.EEL2==1.
- A PE is architecturally required to invalidate all relevant entries in the Secure EL1&0 translation of a System MMU in the same required shareability domain with a VMID of 0.

Configuration

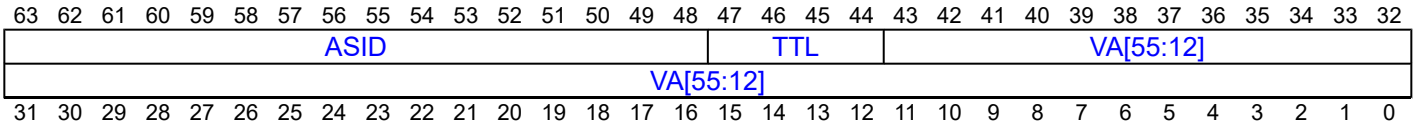
This instruction is present only from Armv8.4, or if ARMv8.4-TLBI is implemented. Otherwise, direct accesses to TLBI VAE1OS are UNDEFINED.

Attributes

TLBI VAE1OS is a 64-bit System instruction.

Field descriptions

The TLBI VAE1OS input value bit assignments are:



ASID, bits [63:48]

ASID value to match. Any TLB entries that match the ASID value and VA value will be affected by this System instruction.~~operation.~~

Global TLB entries that match the VA value will be affected by this System instruction~~operation~~, regardless of the value of the ASID field.

If the implementation supports 16 bits of ASID, but only 8 bits are being used in the context being invalidated, the upper bits are RES0.

TTL, bits [47:44]

From Armv8.4, or if ARMv8.4-TTL is implemented:

Translation Table Level. Indicates the level of the page table walk that holds the leaf entry for the address being invalidated.

TTL	Meaning
0b00xx	No information supplied as to the translation table level. Hardware must assume that the entry can be from any level. In this case, TTL<1:0> is RES0.
0b01xx	The entry comes from a 4KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3.
0b10xx	The entry comes from a 16KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : Reserved. Treat as if TTL<3:2> is 0b00. 0b10 : Level 2. 0b11 : Level 3.
0b11xx	The entry comes from a 64KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3.

If an incorrect value of the TTL field is specified for the entry being invalidated by the instruction, then no entries are required by the architecture to be invalidated from the TLB.

Otherwise:

Reserved, RES0.

VA[55:12], bits [43:0]

Bits[55:12] of the virtual address to match. Any appropriate TLB entries that match the ASID value (if appropriate) and VA will be affected by this System instruction.~~operation.~~

If the TLB maintenance instructions are targeting a translation regime that is using AArch32, and so has a VA of only 32 bits, then the software must treat bits[55:32] as RES0.

The treatment of the low-order bits of this field depends on the translation granule size, as follows:

- Where a 4KB translation granule is being used, all bits are valid and used for the invalidation.
- Where a 16KB translation granule is being used, bits [1:0] of this field are RES0 and ignored when the instruction is executed, because VA[13:12] have no effect on the operation of the instruction.
- Where a 64KB translation granule is being used, bits [3:0] of this field are RES0 and ignored when the instruction is executed, because VA[15:12] have no effect on the operation of the instruction.

Executing the TLBI VAE1OS instruction

Accesses to this instruction use the following encodings:

```
TLBI VAE1OS{, <Xt>}
```

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b01	0b000	0b1000	0b0001	0b001
~~0b01~~	~~0b1000~~	~~0b000~~	~~0b001~~	~~0b0001~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TTLB == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TTLBOS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        TLBI_VAE1OS(X[t]);
elsif PSTATE.EL == EL2 then
    TLBI_VAE1OS(X[t]);
elsif PSTATE.EL == EL3 then
    TLBI_VAE1OS(X[t]);
```

27~~13~~/03~~12~~/2019~~2018~~ 21~~16~~:59~~42~~; e5e4db499bf9867a4b93324c4dbac985d3da9376~~6379d01e197f1d40720d32d0f84c419e9187e009~~

(old) htmldiff from- (new)

(old) htmldiff from- (new)

TLBI VAE2, TLB Invalidate by VA, EL2

The TLBI VAE2 characteristics are:

Purpose

When EL2 is implemented and enabled in the current Security state, invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 1 translation table entry.entry, from any level of the translation table walk.
-
  - The entry is from a level of lookup above the final level and matches the specified ASID.
  - The entry is a global entry from the final level of lookup.
  - The entry is a non-global entry from the final level of lookup that matches the specified ASID.

  The entry would be requiredused to translate the specified VA, usingand theone EL2of or the EL2&0following translation regime.applies:
- If HCR_EL2..{E2H, ==TGE} 0is not {1, 1}, the entry iswould frombe anyused levelwith ofthe current VMID and would be required to translate the specified VA using the EL2 translation table walk.regime.
- If HCR_EL2..{E2H, ==TGE} is {1, one1}, ofthe entry would be required to translate the followingspecified applies:VA using the EL2&0 translation regime.
  - The entry is from a level of the translation table walk above the final level and matches the specified ASID.
  - The entry is a global entry from the final level of the translation table walk.
  - The entry is a non-global entry from the final level of the translation table walk and matches the specified ASID.

The invalidation only applies to the PE that executes this System instruction.

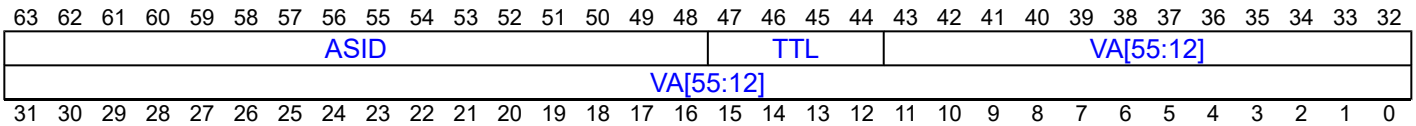
Configuration

Attributes

TLBI VAE2 is a 64-bit System instruction.

Field descriptions

The TLBI VAE2 input value bit assignments are:



ASID, bits [63:48]

ASID value to match. Any TLB entries that match the ASID value and VA value will be affected by this System instruction.operation.

Global TLB entries that match the VA value will be affected by this System instructionoperation, regardless of the value of the ASID field.

If the implementation supports 16 bits of ASID, but only 8 bits are being used in the context being invalidated, the upper bits are RES0.

TTL, bits [47:44]

When ARMv8.4-TTL is implemented:

Translation Table Level. Indicates the level of the page table walk that holds the leaf entry for the address being invalidated.

TTL	Meaning
0b00xx	No information supplied as to the translation table level. Hardware must assume that the entry can be from any level. In this case, TTL<1:0> is RES0.
0b01xx	The entry comes from a 4KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3.
0b10xx	The entry comes from a 16KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : Reserved. Treat as if TTL<3:2> is 0b00. 0b10 : Level 2. 0b11 : Level 3.
0b11xx	The entry comes from a 64KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3.

If an incorrect value of the TTL field is specified for the entry being invalidated by the instruction, then no entries are required by the architecture to be invalidated from the TLB.

Otherwise:

Reserved, RES0.

VA[55:12], bits [43:0]

Bits[55:12] of the virtual address to match. Any appropriate TLB entries that match the ASID value (if appropriate) and VA will be affected by this System instruction.~~operation.~~

If the TLB maintenance instructions are targeting a translation regime that is using AArch32, and so has a VA of only 32 bits, then the software must treat bits[55:32] as RES0.

The treatment of the low-order bits of this field depends on the translation granule size, as follows:

- Where a 4KB translation granule is being used, all bits are valid and used for the invalidation.
- Where a 16KB translation granule is being used, bits [1:0] of this field are RES0 and ignored when the instruction is executed, because VA[13:12] have no effect on the operation of the instruction.
- Where a 64KB translation granule is being used, bits [3:0] of this field are RES0 and ignored when the instruction is executed, because VA[15:12] have no effect on the operation of the instruction.

Executing the TLBI VAE2 instruction

Accesses to this instruction use the following encodings:

```
TLBI VAE2{, <Xt>}
```

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b01	0b100	0b1000	0b0111	0b001
~~0b01~~	~~0b1000~~	~~0b100~~	~~0b001~~	~~0b0111~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    TLBI_VAE2(X[t]);
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        UNDEFINED;
    else
        TLBI_VAE2(X[t]);
```

27~~13~~/03~~12~~/2019~~2018~~ 21~~16~~:59~~42~~; e5e4db499bf9867a4b93324c4dbac985d3da93766~~379d01e197f1d40720d32d0f84e419e9187e009~~

(old) htmldiff from- (new)

(old) htmldiff from- (new)

TLBI VAE2IS, TLB Invalidate by VA, EL2, Inner Shareable

The TLBI VAE2IS characteristics are:

Purpose

When EL2 is implemented and enabled in the current Security state, invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 1 translation table entry.~~entry, from any level of the translation table walk.~~
- ~~The entry is from a level of lookup above the final level and matches the specified ASID.~~
- ~~The entry is a global entry from the final level of lookup.~~
- ~~The entry is a non-global entry from the final level of lookup that matches the specified ASID.~~

  The entry would be required~~used~~ to translate the specified VA~~,~~ using~~and~~ the~~one~~ EL2~~of~~ or the EL2&0~~following~~ translation regime.~~applies:~~
- If HCR_EL2.~~{~~E2H~~,~~ ~~==TGE}~~ 0~~is not {~~1, ~~1},~~ the entry is~~would~~ from~~be~~ any~~used~~ level~~with~~ of~~the~~ ~~current VMID and would be required to translate~~ the ~~specified VA using the EL2~~ translation table walk.~~regime.~~
- If HCR_EL2.~~{~~E2H~~,~~ ~~==TGE}~~ ~~is {~~1~~,~~ one~~1},~~ of~~the~~ ~~entry would be required to translate~~ the following~~specified~~ applies:~~VA using the EL2&0 translation regime.~~
  - The entry is from a level of the translation table walk above the final level and matches the specified ASID.
  - The entry is a global entry from the final level of the translation table walk.
  - The entry is a non-global entry from the final level of the translation table walk and matches the specified ASID.

The invalidation ~~only~~ applies to all PEs in the same Inner Shareable shareability domain as the PE that executes this System instruction.

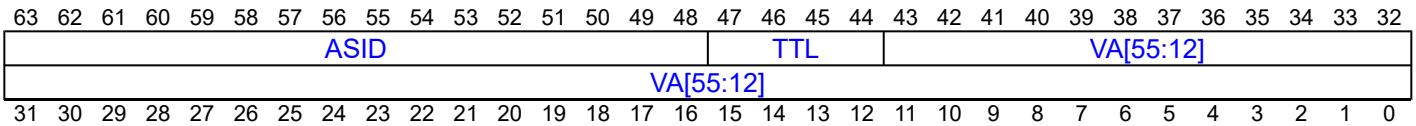
Configuration

Attributes

TLBI VAE2IS is a 64-bit System instruction.

Field descriptions

The TLBI VAE2IS input value bit assignments are:



ASID, bits [63:48]

ASID value to match. Any TLB entries that match the ASID value and VA value will be affected by this System instruction.~~operation.~~

Global TLB entries that match the VA value will be affected by this System instruction~~operation~~, regardless of the value of the ASID field.

If the implementation supports 16 bits of ASID, but only 8 bits are being used in the context being invalidated, the upper bits are RES0.

TTL, bits [47:44]

When ARMv8.4-TTL is implemented:

Translation Table Level. Indicates the level of the page table walk that holds the leaf entry for the address being invalidated.

TTL	Meaning
0b00xx	No information supplied as to the translation table level. Hardware must assume that the entry can be from any level. In this case, TTL<1:0> is RES0.
0b01xx	The entry comes from a 4KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3.
0b10xx	The entry comes from a 16KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : Reserved. Treat as if TTL<3:2> is 0b00. 0b10 : Level 2. 0b11 : Level 3.
0b11xx	The entry comes from a 64KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3.

If an incorrect value of the TTL field is specified for the entry being invalidated by the instruction, then no entries are required by the architecture to be invalidated from the TLB.

Otherwise:

Reserved, RES0.

VA[55:12], bits [43:0]

Bits[55:12] of the virtual address to match. Any appropriate TLB entries that match the ASID value (if appropriate) and VA will be affected by this System instruction.~~operation.~~

If the TLB maintenance instructions are targeting a translation regime that is using AArch32, and so has a VA of only 32 bits, then the software must treat bits[55:32] as RES0.

The treatment of the low-order bits of this field depends on the translation granule size, as follows:

- Where a 4KB translation granule is being used, all bits are valid and used for the invalidation.
- Where a 16KB translation granule is being used, bits [1:0] of this field are RES0 and ignored when the instruction is executed, because VA[13:12] have no effect on the operation of the instruction.
- Where a 64KB translation granule is being used, bits [3:0] of this field are RES0 and ignored when the instruction is executed, because VA[15:12] have no effect on the operation of the instruction.

Executing the TLBI VAE2IS instruction

Accesses to this instruction use the following encodings:

```
TLBI VAE2IS{, <Xt>}
```

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b01	0b100	0b1000	0b0011	0b001
~~0b01~~	~~0b1000~~	~~0b100~~	~~0b001~~	~~0b0011~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    TLBI_VAE2IS(X[t]);
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        UNDEFINED;
    else
        TLBI_VAE2IS(X[t]);
```

27~~13~~/03~~12~~/2019~~2018~~ 21~~16~~:59~~42~~; e5e4db499bf9867a4b93324c4dbac985d3da93766~~379d01e197f1d40720d32d0f84e419e9187e009~~

(old) htmldiff from- (new)

(old) htmldiff from- (new)

TLBI VAE2OS, TLB Invalidate by VA, EL2, Outer Shareable

The TLBI VAE2OS characteristics are:

Purpose

When EL2 is implemented and enabled in the current Security state, invalidatesinvalidate cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 1 translation table entry.entry, from any level of the translation table walk.
-
  - The entry is from a level of lookup above the final level and matches the specified ASID.
  - The entry is a global entry from the final level of lookup.
  - The entry is a non-global entry from the final level of lookup that matches the specified ASID.

  The entry would be requiredused to translate the specified VA, usingand theone EL2of or the EL2&0following translation regime.applies:
- If HCR_EL2.{E2H, ==TGE} 0is not {1, 1}, the entry iswould frombe anyused levelwith ofthe current VMID and would be required to translate the specified VA using the EL2 translation table walk.regime.
- If HCR_EL2.{E2H, ==TGE} is {1, one1}, ofthe entry would be required to translate the followingspecified applies:VA using the EL2&0 translation regime.
  - The entry is from a level of the translation table walk above the final level and matches the specified ASID.
  - The entry is a global entry from the final level of the translation table walk.
  - The entry is a non-global entry from the final level of the translation table walk and matches the specified ASID.

The invalidation only applies to all PEs in the same Outer Shareable shareability domain as the PE that executes this System instruction.

Configuration

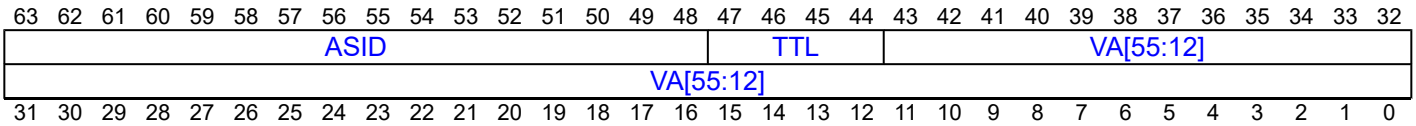
This instruction is present only when ARMv8.4-TLBI is implemented. Otherwise, direct accesses to TLBI VAE2OS are UNDEFINED.

Attributes

TLBI VAE2OS is a 64-bit System instruction.

Field descriptions

The TLBI VAE2OS input value bit assignments are:



ASID, bits [63:48]

When HCR_EL2.E2H == 1:

ASID value to match. Any TLB entries that match the ASID value and VA value will be affected by this System instruction.operation.

Global TLB entries that match the VA value will be affected by this System instructionoperation, regardless of the value of the ASID field.

If the implementation supports 16 bits of ASID, but only 8 bits are being used in the context being invalidated, the upper bits are RES0.

Otherwise:

Reserved, RES0.

TTL, bits [47:44]

When ARMv8.4-TTL is implemented:

Translation Table Level. Indicates the level of the page table walk that holds the leaf entry for the address being invalidated.

TTL	Meaning
0b00xx	No information supplied as to the translation table level. Hardware must assume that the entry can be from any level. In this case, TTL<1:0> is RES0.
0b01xx	The entry comes from a 4KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3.
0b10xx	The entry comes from a 16KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : Reserved. Treat as if TTL<3:2> is 0b00. 0b10 : Level 2. 0b11 : Level 3.
0b11xx	The entry comes from a 64KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3.

If an incorrect value of the TTL field is specified for the entry being invalidated by the instruction, then no entries are required by the architecture to be invalidated from the TLB.

Otherwise:

Reserved, RES0.

VA[55:12], bits [43:0]

Bits[55:12] of the virtual address to match. Any appropriate TLB entries that match the ASID value (if appropriate) and VA will be affected by this System instruction.~~operation.~~

If the TLB maintenance instructions are targeting a translation regime that is using AArch32, and so has a VA of only 32 bits, then the software must treat bits[55:32] as RES0.

The treatment of the low-order bits of this field depends on the translation granule size, as follows:

- Where a 4KB translation granule is being used, all bits are valid and used for the invalidation.
- Where a 16KB translation granule is being used, bits [1:0] of this field are RES0 and ignored when the instruction is executed, because VA[13:12] have no effect on the operation of the instruction.
- Where a 64KB translation granule is being used, bits [3:0] of this field are RES0 and ignored when the instruction is executed, because VA[15:12] have no effect on the operation of the instruction.

Executing the TLBI VAE2OS instruction

Accesses to this instruction use the following encodings:

```
TLBI VAE2OS{, <Xt>}
```

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b01	0b100	0b1000	0b0001	0b001
~~0b01~~	~~0b1000~~	~~0b100~~	~~0b001~~	~~0b0001~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    TLBI_VAE2OS(X[t]);
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        UNDEFINED;
    else
        TLBI_VAE2OS(X[t]);
```

27~~13~~/03~~12~~/2019~~2018~~ 21~~16~~:59~~42~~: e5e4db499bf9867a4b93324c4dbac985d3da93766~~379d01e197f1d40720d32d0f84e419e9187e009~~

(old) htmldiff from- (new)

TLBI VAE3, TLB Invalidate by VA, EL3

The TLBI VAE3 characteristics are:

Purpose

If EL3 is implemented, invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 1 translation table entry, from any level of the translation table walk.
- The entry would be used to translate the specified VA using the EL3 translation regime.

The invalidation ~~only~~ applies to the PE that executes this System instruction.

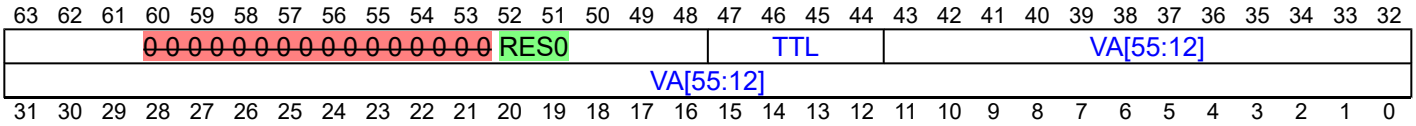
Configuration

Attributes

TLBI VAE3 is a 64-bit System instruction.

Field descriptions

The TLBI VAE3 input value bit assignments are:



Bits [63:48]

Reserved, RES0.

TTL, bits [47:44]

When ARMv8.4-TTL is implemented:

Translation Table Level. Indicates the level of the page table walk that holds the leaf entry for the address being invalidated.

TTL	Meaning
0b00xx	No information supplied as to the translation table level. Hardware must assume that the entry can be from any level. In this case, TTL<1:0> is RES0.
0b01xx	The entry comes from a 4KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3.
0b10xx	The entry comes from a 16KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : Reserved. Treat as if TTL<3:2> is 0b00. 0b10 : Level 2. 0b11 : Level 3.
0b11xx	The entry comes from a 64KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3.

If an incorrect value of the TTL field is specified for the entry being invalidated by the instruction, then no entries are required by the architecture to be invalidated from the TLB.

Otherwise:

Reserved, RES0.

VA[55:12], bits [43:0]

Bits[55:12] of the virtual address to match. Any appropriate TLB entries that match the ASID value (if appropriate) and VA will be affected by this System instruction.~~operation.~~

If the TLB maintenance instructions are targeting a translation regime that is using AArch32, and so has a VA of only 32 bits, then the software must treat bits[55:32] as RES0.

The treatment of the low-order bits of this field depends on the translation granule size, as follows:

- Where a 4KB translation granule is being used, all bits are valid and used for the invalidation.
- Where a 16KB translation granule is being used, bits [1:0] of this field are RES0 and ignored when the instruction is executed, because VA[13:12] have no effect on the operation of the instruction.
- Where a 64KB translation granule is being used, bits [3:0] of this field are RES0 and ignored when the instruction is executed, because VA[15:12] have no effect on the operation of the instruction.

Executing the TLBI VAE3 instruction

Accesses to this instruction use the following encodings:

```
TLBI VAE3{, <Xt>}
```

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b01	0b110	0b1000	0b0111	0b001
~~0b01~~	~~0b1000~~	~~0b110~~	~~0b001~~	~~0b0111~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    TLBI_VAE3(X[t]);
```

27~~13~~/03~~12~~/2019~~2018~~ 21~~16~~:59~~42~~; e5e4db499bf9867a4b93324c4dbac985d3da93766~~379d01e197f1d40720d32d0f84e419e9187e009~~

(old) htmldiff from- (new)

(old) htmldiff from- (new)

TLBI VAE3IS, TLB Invalidate by VA, EL3, Inner Shareable

The TLBI VAE3IS characteristics are:

Purpose

If EL3 is implemented, invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 1 translation table entry, from any level of the translation table walk.
- The entry would be used to translate the specified VA using the EL3 translation regime.

The invalidation ~~only~~ applies to all PEs in the same Inner Shareable shareability domain as the PE that executes this System instruction.

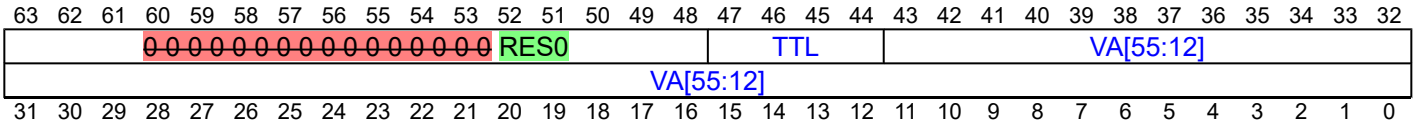
Configuration

Attributes

TLBI VAE3IS is a 64-bit System instruction.

Field descriptions

The TLBI VAE3IS input value bit assignments are:



Bits [63:48]

Reserved, RES0.

TTL, bits [47:44]

When ARMv8.4-TTL is implemented:

Translation Table Level. Indicates the level of the page table walk that holds the leaf entry for the address being invalidated.

TTL	Meaning
0b00xx	No information supplied as to the translation table level. Hardware must assume that the entry can be from any level. In this case, TTL<1:0> is RES0.
0b01xx	The entry comes from a 4KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3.
0b10xx	The entry comes from a 16KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : Reserved. Treat as if TTL<3:2> is 0b00. 0b10 : Level 2. 0b11 : Level 3.
0b11xx	The entry comes from a 64KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3.

If an incorrect value of the TTL field is specified for the entry being invalidated by the instruction, then no entries are required by the architecture to be invalidated from the TLB.

Otherwise:

Reserved, RES0.

VA[55:12], bits [43:0]

Bits[55:12] of the virtual address to match. Any appropriate TLB entries that match the ASID value (if appropriate) and VA will be affected by this System instruction.~~operation.~~

If the TLB maintenance instructions are targeting a translation regime that is using AArch32, and so has a VA of only 32 bits, then the software must treat bits[55:32] as RES0.

The treatment of the low-order bits of this field depends on the translation granule size, as follows:

- Where a 4KB translation granule is being used, all bits are valid and used for the invalidation.
- Where a 16KB translation granule is being used, bits [1:0] of this field are RES0 and ignored when the instruction is executed, because VA[13:12] have no effect on the operation of the instruction.
- Where a 64KB translation granule is being used, bits [3:0] of this field are RES0 and ignored when the instruction is executed, because VA[15:12] have no effect on the operation of the instruction.

Executing the TLBI VAE3IS instruction

Accesses to this instruction use the following encodings:

```
TLBI VAE3IS{, <Xt>}
```

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b01	0b110	0b1000	0b0011	0b001
~~0b01~~	~~0b1000~~	~~0b110~~	~~0b001~~	~~0b0011~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    TLBI_VAE3IS(X[t]);
```

2713/0312/20192018 2116:5942: e5e4db499bf9867a4b93324c4dbac985d3da93766379d01e197f1d40720d32d0f84e419e9187e009

(old) htmldiff from- (new)

(old) htmldiff from- (new)

TLBI VAE3OS, TLB Invalidate by VA, EL3, Outer Shareable

The TLBI VAE3OS characteristics are:

Purpose

If EL3 is implemented, invalidates~~invalidate~~ cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 1 translation table entry, from any level of the translation table walk.
- The entry would be used to translate the specified VA using the EL3 translation regime.

The invalidation ~~only~~ applies to all PEs in the same Outer Shareable shareability domain as the PE that executes this System instruction.

Configuration

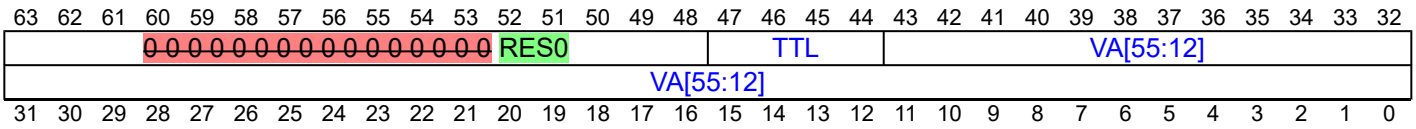
This instruction is present only from Armv8.4, or if ARMv8.4-TLBI is implemented. Otherwise, direct accesses to TLBI VAE3OS are UNDEFINED.

Attributes

TLBI VAE3OS is a 64-bit System instruction.

Field descriptions

The TLBI VAE3OS input value bit assignments are:



Bits [63:48]

Reserved, RES0.

TTL, bits [47:44]

When ARMv8.4-TTL is implemented:

Translation Table Level. Indicates the level of the page table walk that holds the leaf entry for the address being invalidated.

TTL	Meaning
0b00xx	No information supplied as to the translation table level. Hardware must assume that the entry can be from any level. In this case, TTL<1:0> is RES0.
0b01xx	The entry comes from a 4KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3.
0b10xx	The entry comes from a 16KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : Reserved. Treat as if TTL<3:2> is 0b00. 0b10 : Level 2. 0b11 : Level 3.
0b11xx	The entry comes from a 64KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3.

If an incorrect value of the TTL field is specified for the entry being invalidated by the instructio, then no entries are required by the architecture to be invalidated from the TLB.

Otherwise:

Reserved, RES0.

VA[55:12], bits [43:0]

Bits[55:12] of the virtual address to match. Any appropriate TLB entries that match the ASID value (if appropriate) and VA will be affected by this System instruction.~~operation.~~

If the TLB maintenance instructions are targeting a translation regime that is using AArch32, and so has a VA of only 32 bits, then the software must treat bits[55:32] as RES0.

The treatment of the low-order bits of this field depends on the translation granule size, as follows:

- Where a 4KB translation granule is being used, all bits are valid and used for the invalidation.
- Where a 16KB translation granule is being used, bits [1:0] of this field are RES0 and ignored when the instruction is executed, because VA[13:12] have no effect on the operation of the instruction.
- Where a 64KB translation granule is being used, bits [3:0] of this field are RES0 and ignored when the instruction is executed, because VA[15:12] have no effect on the operation of the instruction.

Executing the TLBI VAE3OS instruction

Accesses to this instruction use the following encodings:

```
TLBI VAE3OS{, <Xt>}
```

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b01	0b110	0b1000	0b0001	0b001
~~0b01~~	~~0b1000~~	~~0b110~~	~~0b001~~	~~0b0001~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    TLBI_VAE3OS(X[t]);
```

2713/0312/20192018 2116:5942: e5e4db499bf9867a4b93324c4dbac985d3da93766379d01e197f1d40720d32d0f84e419e9187e009

(old) htmldiff from- (new)

(old) htmldiff from- (new)

TLBI VALE1, TLB Invalidate by VA, Last level, EL1

The TLBI VALE1 characteristics are:

Purpose

Invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 1 translation table entry.
- The entry would be used to translate the specified VA, and one of the following applies:
  - The entry is a global entry from the final level of lookup.
  - The entry is a non-global entry from the final level of lookup that matches the specified ASID.
- When EL2 is implemented and enabled in the ~~current~~ Security state described by the current value of~~:~~ SCR_EL3.NS:
  - If HCR_EL2.{E2H, TGE} is not {1, 1}, the entry would be used with the current VMID and would be required to translate the specified VA using the EL1&0 translation regime.
  - If HCR_EL2.{E2H, TGE} is {1, 1}, the entry would be required to translate the specified VA using the EL2&0 translation regime.
- When EL2 is not implemented or is disabled in the current Security state, the entry would be required to translate the specified VA using the EL1&0 translation regime.

The invalidation ~~only~~ applies to the PE that executes this System instruction.

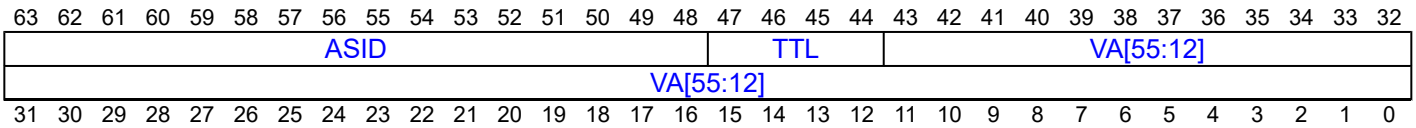
Configuration

Attributes

TLBI VALE1 is a 64-bit System instruction.

Field descriptions

The TLBI VALE1 input value bit assignments are:



ASID, bits [63:48]

ASID value to match. Any TLB entries that match the ASID value and VA value will be affected by this System instruction.~~operation.~~

Global TLB entries that match the VA value will be affected by this System instruction~~operation~~, regardless of the value of the ASID field.

If the implementation supports 16 bits of ASID, but only 8 bits are being used in the context being invalidated, the upper bits are RES0.

TTL, bits [47:44]

When ARMv8.4-TTL is implemented:

Translation Table Level. Indicates the level of the page table walk that holds the leaf entry for the address being invalidated.

TTL	Meaning
0b00xx	No information supplied as to the translation table level. Hardware must assume that the entry can be from any level. In this case, TTL<1:0> is RES0.
0b01xx	The entry comes from a 4KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3.
0b10xx	The entry comes from a 16KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : Reserved. Treat as if TTL<3:2> is 0b00. 0b10 : Level 2. 0b11 : Level 3.
0b11xx	The entry comes from a 64KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3.

If an incorrect value of the TTL field is specified for the entry being invalidated by the instruction, then no entries are required by the architecture to be invalidated from the TLB.

Otherwise:

Reserved, RES0.

VA[55:12], bits [43:0]

Bits[55:12] of the virtual address to match. Any appropriate TLB entries that match the ASID value (if appropriate) and VA will be affected by this System instruction.~~operation.~~

If the TLB maintenance instructions are targeting a translation regime that is using AArch32, and so has a VA of only 32 bits, then the software must treat bits[55:32] as RES0.

The treatment of the low-order bits of this field depends on the translation granule size, as follows:

- Where a 4KB translation granule is being used, all bits are valid and used for the invalidation.
- Where a 16KB translation granule is being used, bits [1:0] of this field are RES0 and ignored when the instruction is executed, because VA[13:12] have no effect on the operation of the instruction.
- Where a 64KB translation granule is being used, bits [3:0] of this field are RES0 and ignored when the instruction is executed, because VA[15:12] have no effect on the operation of the instruction.

Executing the TLBI VALE1 instruction

Accesses to this instruction use the following encodings:

```
TLBI VALE1{, <Xt>}
```

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b01	0b000	0b1000	0b0111	0b101
~~0b01~~	~~0b1000~~	~~0b000~~	~~0b101~~	~~0b0111~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TTLB == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.FB == '1' then
        TLBI_VALE1IS(X[t]);
    else
        TLBI_VAAE1(X[t]);
elsif PSTATE.EL == EL2 then
    TLBI_VAAE1(X[t]);
elsif PSTATE.EL == EL3 then
    TLBI_VAAE1(X[t]);
```

27~~13~~/03~~12~~/2019~~2018~~ 21~~16~~:59~~42~~; e5e4db499bf9867a4b93324c4dbac985d3da93766~~6379d01e197f1d40720d32d0f84e419e9187e009~~

(old) htmldiff from- (new)

(old) htmldiff from- (new)

TLBI VALE1IS, TLB Invalidate by VA, Last level, EL1, Inner Shareable

The TLBI VALE1IS characteristics are:

Purpose

Invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 1 translation table entry.
- The entry would be used to translate the specified VA, and one of the following applies:
  - The entry is a global entry from the final level of lookup.
  - The entry is a non-global entry from the final level of lookup that matches the specified ASID.
- When EL2 is implemented and enabled in the ~~current~~ Security state described by the current value of~~:~~ SCR_EL3.NS:
  - If HCR_EL2.{E2H, TGE} is not {1, 1}, the entry would be used with the current VMID and would be required to translate the specified VA using the EL1&0 translation regime.
  - If HCR_EL2.{E2H, TGE} is {1, 1}, the entry would be required to translate the specified VA using the EL2&0 translation regime.
- When EL2 is not implemented or is disabled in the current Security state, the entry would be required to translate the specified VA using the EL1&0 translation regime.

The invalidation ~~only~~ applies to all PEs in the same Inner Shareable shareability domain as the PE that executes this System instruction.

Note

From Armv8.4, when a TLB maintenance instruction is generated to the Secure EL1&0 translation regime and is defined to pass a VMID argument, or would be defined to pass a VMID argument if SCR_EL3.EEL2==1, then:

- A PE with SCR_EL3.EEL2==1 is not architecturally required to invalidate any entries in the Secure EL1&0 translation of a PE in the same required shareability domain with SCR_EL3.EEL2==0.
- A PE with SCR_EL3.EEL2==0 is not architecturally required to invalidate any entries in the Secure EL1&0 translation of a PE in the same required shareability domain with SCR_EL3.EEL2==1.
- A PE is architecturally required to invalidate all relevant entries in the Secure EL1&0 translation of a System MMU in the same required shareability domain with a VMID of 0.

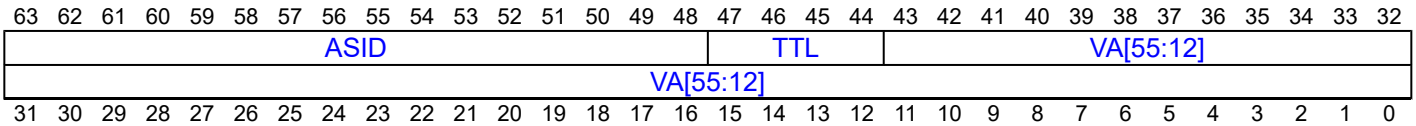
Configuration

Attributes

TLBI VALE1IS is a 64-bit System instruction.

Field descriptions

The TLBI VALE1IS input value bit assignments are:



ASID, bits [63:48]

ASID value to match. Any TLB entries that match the ASID value and VA value will be affected by this System instruction.~~operation.~~

Global TLB entries that match the VA value will be affected by this System instruction~~operation~~, regardless of the value of the ASID field.

If the implementation supports 16 bits of ASID, but only 8 bits are being used in the context being invalidated, the upper bits are RES0.

TTL, bits [47:44]

When ARMv8.4-TTL is implemented:

Translation Table Level. Indicates the level of the page table walk that holds the leaf entry for the address being invalidated.

TTL	Meaning
0b00xx	No information supplied as to the translation table level. Hardware must assume that the entry can be from any level. In this case, TTL<1:0> is RES0.
0b01xx	The entry comes from a 4KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3.
0b10xx	The entry comes from a 16KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : Reserved. Treat as if TTL<3:2> is 0b00. 0b10 : Level 2. 0b11 : Level 3.
0b11xx	The entry comes from a 64KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3.

If an incorrect value of the TTL field is specified for the entry being invalidated by the instruction, then no entries are required by the architecture to be invalidated from the TLB.

Otherwise:

Reserved, RES0.

VA[55:12], bits [43:0]

Bits[55:12] of the virtual address to match. Any appropriate TLB entries that match the ASID value (if appropriate) and VA will be affected by this System instruction.~~operation.~~

If the TLB maintenance instructions are targeting a translation regime that is using AArch32, and so has a VA of only 32 bits, then the software must treat bits[55:32] as RES0.

The treatment of the low-order bits of this field depends on the translation granule size, as follows:

- Where a 4KB translation granule is being used, all bits are valid and used for the invalidation.
- Where a 16KB translation granule is being used, bits [1:0] of this field are RES0 and ignored when the instruction is executed, because VA[13:12] have no effect on the operation of the instruction.
- Where a 64KB translation granule is being used, bits [3:0] of this field are RES0 and ignored when the instruction is executed, because VA[15:12] have no effect on the operation of the instruction.

Executing the TLBI VALE1IS instruction

Accesses to this instruction use the following encodings:

```
TLBI VALE1IS{, <Xt>}
```

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b01	0b000	0b1000	0b0011	0b101
~~0b01~~	~~0b1000~~	~~0b000~~	~~0b101~~	~~0b0011~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TTLB == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TTLBIS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        TLBI_VALE1IS(X[t]);
elsif PSTATE.EL == EL2 then
    TLBI_VALE1IS(X[t]);
elsif PSTATE.EL == EL3 then
    TLBI_VALE1IS(X[t]);
```

27~~13~~/03~~12~~/2019~~2018~~ 21~~16~~:59~~42~~: e5e4db499bf9867a4b93324c4dbac985d3da93766379d01e197f1d40720d32d0f84e419e9187e009

(old) htmldiff from- (new)

TLBI VALE1OS, TLB Invalidate by VA, Last level, EL1, Outer Shareable

The TLBI VALE1OS characteristics are:

Purpose

Invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 1 translation table entry.
- The entry would be used to translate the specified VA, and one of the following applies:
  - The entry is a global entry from the final level of lookup.
  - The entry is a non-global entry from the final level of lookup that matches the specified ASID.
- When EL2 is implemented and enabled in the ~~current~~ Security state described by the current value of~~:~~ SCR_EL3.NS:
  - If HCR_EL2.{E2H, TGE} is not {1, 1}, the entry would be used with the current VMID and would be required to translate the specified VA using the EL1&0 translation regime.
  - If HCR_EL2.{E2H, TGE} is {1, 1}, the entry would be required to translate the specified VA using the EL2&0 translation regime.
- When EL2 is not implemented or disabled in the current Security state, the entry would be required to translate the specified VA using the EL1&0 translation regime.

The invalidation ~~only~~ applies to all PEs in the same Outer Shareable shareability domain as the PE that executes this System instruction.

Note

When a TLB maintenance instruction is generated to the Secure EL1&0 translation regime and is defined to pass a VMID argument, or would be defined to pass a VMID argument if SCR_EL3.EEL2==1, then:

- A PE with SCR_EL3.EEL2==1 is not architecturally required to invalidate any entries in the Secure EL1&0 translation of a PE in the same required shareability domain with SCR_EL3.EEL2==0.
- A PE with SCR_EL3.EEL2==0 is not architecturally required to invalidate any entries in the Secure EL1&0 translation of a PE in the same required shareability domain with SCR_EL3.EEL2==1.
- A PE is architecturally required to invalidate all relevant entries in the Secure EL1&0 translation of a System MMU in the same required shareability domain with a VMID of 0.

Configuration

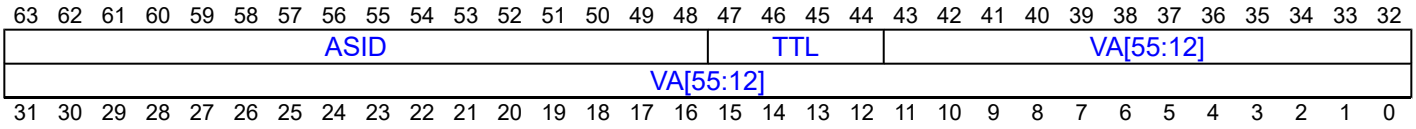
This instruction is present only from Armv8.4, or if ARMv8.4-TLBI is implemented. Otherwise, direct accesses to TLBI VALE1OS are UNDEFINED.

Attributes

TLBI VALE1OS is a 64-bit System instruction.

Field descriptions

The TLBI VALE1OS input value bit assignments are:



ASID, bits [63:48]

ASID value to match. Any TLB entries that match the ASID value and VA value will be affected by this System instruction.~~operation.~~

Global TLB entries that match the VA value will be affected by this System instruction~~operation~~, regardless of the value of the ASID field.

If the implementation supports 16 bits of ASID, but only 8 bits are being used in the context being invalidated, the upper bits are RES0.

TTL, bits [47:44]

When ARMv8.4-TTL is implemented:

Translation Table Level. Indicates the level of the page table walk that holds the leaf entry for the address being invalidated.

TTL	Meaning
0b00xx	No information supplied as to the translation table level. Hardware must assume that the entry can be from any level. In this case, TTL<1:0> is RES0.
0b01xx	The entry comes from a 4KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3.
0b10xx	The entry comes from a 16KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : Reserved. Treat as if TTL<3:2> is 0b00. 0b10 : Level 2. 0b11 : Level 3.
0b11xx	The entry comes from a 64KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3.

If an incorrect value of the TTL field is specified for the entry being invalidated by the instruction, then no entries are required by the architecture to be invalidated from the TLB.

Otherwise:

Reserved, RES0.

VA[55:12], bits [43:0]

Bits[55:12] of the virtual address to match. Any appropriate TLB entries that match the ASID value (if appropriate) and VA will be affected by this System instruction.~~operation.~~

If the TLB maintenance instructions are targeting a translation regime that is using AArch32, and so has a VA of only 32 bits, then the software must treat bits[55:32] as RES0.

The treatment of the low-order bits of this field depends on the translation granule size, as follows:

- Where a 4KB translation granule is being used, all bits are valid and used for the invalidation.
- Where a 16KB translation granule is being used, bits [1:0] of this field are RES0 and ignored when the instruction is executed, because VA[13:12] have no effect on the operation of the instruction.
- Where a 64KB translation granule is being used, bits [3:0] of this field are RES0 and ignored when the instruction is executed, because VA[15:12] have no effect on the operation of the instruction.

Executing the TLBI VALE1OS instruction

Accesses to this instruction use the following encodings:

```
TLBI VALE1OS{, <Xt>}
```

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b01	0b000	0b1000	0b0001	0b101
~~0b01~~	~~0b1000~~	~~0b000~~	~~0b101~~	~~0b0001~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TTLB == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TTLBOS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        TLBI_VALE1OS(X[t]);
elsif PSTATE.EL == EL2 then
    TLBI_VALE1OS(X[t]);
elsif PSTATE.EL == EL3 then
    TLBI_VALE1OS(X[t]);
```

27~~13~~/03~~12~~/2019~~2018~~ 21~~16~~:59~~42~~; e5e4db499bf9867a4b93324c4dbac985d3da9376~~6379d01e197f1d40720d32d0f84c419e9187e009~~

(old) htmldiff from- (new)

(old) htmldiff from- (new)

TLBI VALE2, TLB Invalidate by VA, Last level, EL2

The TLBI VALE2 characteristics are:

Purpose

When EL2 is implemented and enabled in the current Security state, invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 1 translation table entry.~~entry, from the final level of the translation table walk.~~
- ~~The entry is a global entry from the final level of lookup.~~
- ~~The entry is a non-global entry from the final level of lookup that matches the specified ASID.~~

  The entry would be used to translate the specified VA~~,~~ using~~and~~ the~~one~~ EL2~~of~~ or~~the~~ EL2&0~~following~~ translation regime.~~applies:~~
- If HCR_EL2.~~{~~E2H~~, ==TGE}~~ 0~~is not {~~1, ~~1},~~ the entry is~~would~~ from~~be~~ ~~used with~~ the final~~current~~ level~~VMID~~ of~~and~~ ~~would be required to translate~~ the ~~specified VA using the EL2~~ translation table walk.~~regime.~~
- If HCR_EL2.~~{~~E2H~~, ==TGE}~~ ~~is {~~1, one~~1},~~ of~~the~~ ~~entry would be required to translate~~ the following~~specified~~ applies:~~VA using the EL2&0 translation regime.~~
  - The entry is a global entry from the final level of the translation table walk.
  - The entry is a non-global entry from the final level of translation table walk that matches the specified ASID.

The invalidation ~~only~~ applies to the PE that executes this System instruction.

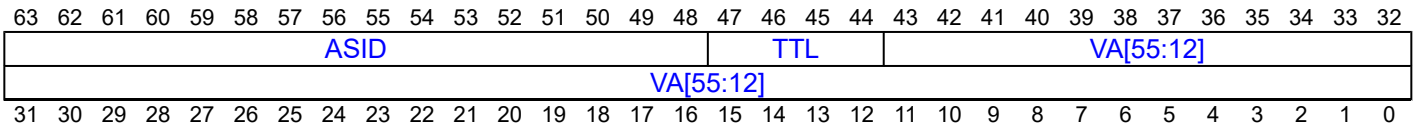
Configuration

Attributes

TLBI VALE2 is a 64-bit System instruction.

Field descriptions

The TLBI VALE2 input value bit assignments are:



ASID, bits [63:48]

When HCR_EL2.E2H == 1:

ASID value to match. Any TLB entries that match the ASID value and VA value will be affected by this System instruction.~~operation.~~

Global TLB entries that match the VA value will be affected by this System instruction~~operation~~, regardless of the value of the ASID field.

If the implementation supports 16 bits of ASID, but only 8 bits are being used in the context being invalidated, the upper bits are RES0.

Otherwise:

Reserved, RES0.

TTL, bits [47:44]

When ARMv8.4-TTL is implemented:

Translation Table Level. Indicates the level of the page table walk that holds the leaf entry for the address being invalidated.

TTL	Meaning
0b00xx	No information supplied as to the translation table level. Hardware must assume that the entry can be from any level. In this case, TTL<1:0> is RES0.
0b01xx	The entry comes from a 4KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3.
0b10xx	The entry comes from a 16KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : Reserved. Treat as if TTL<3:2> is 0b00. 0b10 : Level 2. 0b11 : Level 3.
0b11xx	The entry comes from a 64KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3.

If an incorrect value of the TTL field is specified for the entry being invalidated by the instruction, then no entries are required by the architecture to be invalidated from the TLB.

Otherwise:

Reserved, RES0.

VA[55:12], bits [43:0]

Bits[55:12] of the virtual address to match. Any appropriate TLB entries that match the ASID value (if appropriate) and VA will be affected by this System instruction.~~operation.~~

If the TLB maintenance instructions are targeting a translation regime that is using AArch32, and so has a VA of only 32 bits, then the software must treat bits[55:32] as RES0.

The treatment of the low-order bits of this field depends on the translation granule size, as follows:

- Where a 4KB translation granule is being used, all bits are valid and used for the invalidation.
- Where a 16KB translation granule is being used, bits [1:0] of this field are RES0 and ignored when the instruction is executed, because VA[13:12] have no effect on the operation of the instruction.
- Where a 64KB translation granule is being used, bits [3:0] of this field are RES0 and ignored when the instruction is executed, because VA[15:12] have no effect on the operation of the instruction.

Executing the TLBI VALE2 instruction

Accesses to this instruction use the following encodings:

```
TLBI VALE2{, <Xt>}
```

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b01	0b100	0b1000	0b0111	0b101
~~0b01~~	~~0b1000~~	~~0b100~~	~~0b101~~	~~0b0111~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    TLBI_VALE2(X[t]);
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        UNDEFINED;
    else
        TLBI_VALE2(X[t]);
```

27~~13~~/03~~12~~/2019~~2018~~ 21~~16~~:59~~42~~; e5e4db499bf9867a4b93324c4dbac985d3da93766~~379d01e197f1d40720d32d0f84e419e9187e009~~

(old) htmldiff from- (new)

(old) htmldiff from- (new)

TLBI VALE2IS, TLB Invalidate by VA, Last level, EL2, Inner Shareable

The TLBI VALE2IS characteristics are:

Purpose

When EL2 is implemented and enabled in the current Security state, invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 1 translation table entry.entry, from the final level of the translation table walk.
- ◦ The entry is a global entry from the final level of lookup.
  ◦ The entry is a non-global entry from the final level of lookup that matches the specified ASID.

  The entry would be used to translate the specified VA, usingand theone EL2of orthe EL2&0following translation regime.applies:
- If HCR_EL2..{E2H, ==TGE} 0is not {1, 1}, the entry iswould frombe used with the finalcurrent levelVMID ofand would be required to translate the specified VA using the EL2 translation table walk.regime.
- If HCR_EL2..{E2H, ==TGE} is {1, one1}, ofthe entry would be required to translate the followingspecified applies:VA using the EL2&0 translation regime.
  ◦ The entry is a global entry from the final level of the translation table walk.
  ◦ The entry is a non-global entry from the final level of translation table walk that matches the specified ASID.

The invalidation only applies to all PEs in the same Inner Shareable shareability domain as the PE that executes this System instruction.

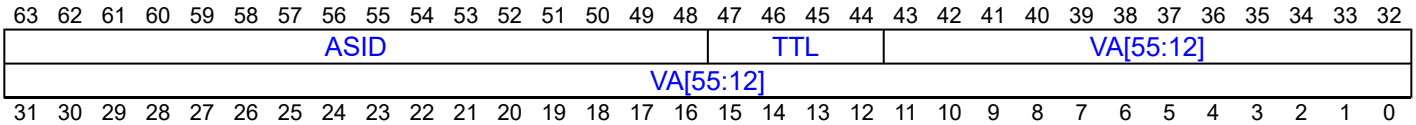
Configuration

Attributes

TLBI VALE2IS is a 64-bit System instruction.

Field descriptions

The TLBI VALE2IS input value bit assignments are:



ASID, bits [63:48]

ASID value to match. Any TLB entries that match the ASID value and VA value will be affected by this System instruction.operation.

Global TLB entries that match the VA value will be affected by this System instructionoperation, regardless of the value of the ASID field.

If the implementation supports 16 bits of ASID, but only 8 bits are being used in the context being invalidated, the upper bits are RES0.

TTL, bits [47:44]

When ARMv8.4-TTL is implemented:

Translation Table Level. Indicates the level of the page table walk that holds the leaf entry for the address being invalidated.

TTL	Meaning
0b00xx	No information supplied as to the translation table level. Hardware must assume that the entry can be from any level. In this case, TTL<1:0> is RES0.
0b01xx	The entry comes from a 4KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3.
0b10xx	The entry comes from a 16KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : Reserved. Treat as if TTL<3:2> is 0b00. 0b10 : Level 2. 0b11 : Level 3.
0b11xx	The entry comes from a 64KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3.

If an incorrect value of the TTL field is specified for the entry being invalidated by the instruction, then no entries are required by the architecture to be invalidated from the TLB.

Otherwise:

Reserved, RES0.

VA[55:12], bits [43:0]

Bits[55:12] of the virtual address to match. Any appropriate TLB entries that match the ASID value (if appropriate) and VA will be affected by this System instruction.~~operation.~~

If the TLB maintenance instructions are targeting a translation regime that is using AArch32, and so has a VA of only 32 bits, then the software must treat bits[55:32] as RES0.

The treatment of the low-order bits of this field depends on the translation granule size, as follows:

- Where a 4KB translation granule is being used, all bits are valid and used for the invalidation.
- Where a 16KB translation granule is being used, bits [1:0] of this field are RES0 and ignored when the instruction is executed, because VA[13:12] have no effect on the operation of the instruction.
- Where a 64KB translation granule is being used, bits [3:0] of this field are RES0 and ignored when the instruction is executed, because VA[15:12] have no effect on the operation of the instruction.

Executing the TLBI VALE2IS instruction

Accesses to this instruction use the following encodings:

```
TLBI VALE2IS{, <Xt>}
```

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b01	0b100	0b1000	0b0011	0b101
~~0b01~~	~~0b1000~~	~~0b100~~	~~0b101~~	~~0b0011~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    TLBI_VALE2IS(X[t]);
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        UNDEFINED;
    else
        TLBI_VALE2IS(X[t]);
```

27~~13~~/03~~12~~/2019~~2018~~ 21~~16~~:59~~42~~; e5e4db499bf9867a4b93324c4dbac985d3da93766~~379d01e197f1d40720d32d0f84e419e9187e009~~

(old) htmldiff from- (new)

(old) htmldiff from- (new)

TLBI VALE2OS, TLB Invalidate by VA, Last level, EL2, Outer Shareable

The TLBI VALE2OS characteristics are:

Purpose

When EL2 is implemented and enabled in the current Security state, invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 1 translation table entry.~~entry, from the final level of the translation table walk.~~
- ~~The entry is a global entry from the final level of lookup.~~
- ~~The entry is a non-global entry from the final level of lookup that matches the specified ASID.~~

The entry would be used to translate the specified VA~~,~~ using~~and~~ the~~one~~ EL2~~of~~ or~~the~~ EL2&0~~following~~ translation regime.~~applies:~~

- If HCR_EL2.~~{~~E2H~~, ==TGE}~~ 0~~is not {~~1, 1~~},~~ the entry is~~would~~ from~~be~~ ~~used with~~ the final~~current~~ level~~VMID~~ of~~and~~ ~~would be required to translate~~ the ~~specified VA using the EL2~~ translation table walk.~~regime.~~
- If HCR_EL2.~~{~~E2H~~, ==TGE}~~ ~~is {~~1, one~~1},~~ of~~the~~ ~~entry would be required to translate~~ the following~~specified~~ applies:~~VA using the EL2&0 translation regime.~~
  - The entry is a global entry from the final level of the translation table walk.
  - The entry is a non-global entry from the final level of translation table walk that matches the specified ASID.

The invalidation ~~only~~ applies to all PEs in the same Outer Shareable shareability domain as the PE that executes this System instruction.

Configuration

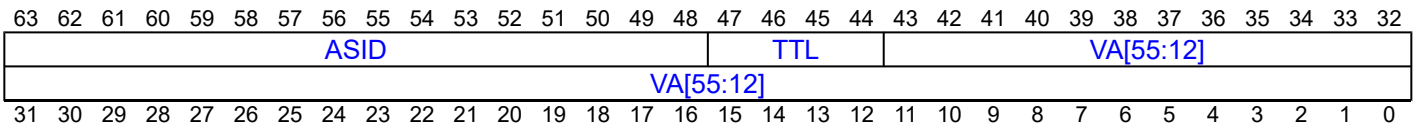
This instruction is present only when ARMv8.4-TLBI is implemented. Otherwise, direct accesses to TLBI VALE2OS are UNDEFINED.

Attributes

TLBI VALE2OS is a 64-bit System instruction.

Field descriptions

The TLBI VALE2OS input value bit assignments are:



ASID, bits [63:48]

When HCR_EL2.E2H == 1:

ASID value to match. Any TLB entries that match the ASID value and VA value will be affected by this System instruction.~~operation.~~

Global TLB entries that match the VA value will be affected by this System instruction~~operation~~, regardless of the value of the ASID field.

If the implementation supports 16 bits of ASID, but only 8 bits are being used in the context being invalidated, the upper bits are RES0.

Otherwise:

Reserved, RES0.

TTL, bits [47:44]

When ARMv8.4-TTL is implemented:

Translation Table Level. Indicates the level of the page table walk that holds the leaf entry for the address being invalidated.

TTL	Meaning
0b00xx	No information supplied as to the translation table level. Hardware must assume that the entry can be from any level. In this case, TTL<1:0> is RES0.
0b01xx	The entry comes from a 4KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3.
0b10xx	The entry comes from a 16KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : Reserved. Treat as if TTL<3:2> is 0b00. 0b10 : Level 2. 0b11 : Level 3.
0b11xx	The entry comes from a 64KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3.

If an incorrect value of the TTL field is specified for the entry being invalidated by the instruction, then no entries are required by the architecture to be invalidated from the TLB.

Otherwise:

Reserved, RES0.

VA[55:12], bits [43:0]

Bits[55:12] of the virtual address to match. Any appropriate TLB entries that match the ASID value (if appropriate) and VA will be affected by this System instruction.~~operation.~~

If the TLB maintenance instructions are targeting a translation regime that is using AArch32, and so has a VA of only 32 bits, then the software must treat bits[55:32] as RES0.

The treatment of the low-order bits of this field depends on the translation granule size, as follows:

- Where a 4KB translation granule is being used, all bits are valid and used for the invalidation.
- Where a 16KB translation granule is being used, bits [1:0] of this field are RES0 and ignored when the instruction is executed, because VA[13:12] have no effect on the operation of the instruction.
- Where a 64KB translation granule is being used, bits [3:0] of this field are RES0 and ignored when the instruction is executed, because VA[15:12] have no effect on the operation of the instruction.

Executing the TLBI VALE2OS instruction

Accesses to this instruction use the following encodings:

```
TLBI VALE2OS{, <Xt>}
```

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b01	0b100	0b1000	0b0001	0b101
~~0b01~~	~~0b1000~~	~~0b100~~	~~0b101~~	~~0b0001~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    TLBI_VALE2OS(X[t]);
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        UNDEFINED;
    else
        TLBI_VALE2OS(X[t]);
```

27~~13~~/03~~12~~/2019~~2018~~ 21~~16~~:59~~42~~: e5e4db499bf9867a4b93324c4dbac985d3da93766~~379d01e197f1d40720d32d0f84e419e9187e009~~

(old) htmldiff from- (new)

TLBI VALE3, TLB Invalidate by VA, Last level, EL3

The TLBI VALE3 characteristics are:

Purpose

If EL3 is implemented, invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 1 translation table entry, from the final level of the translation table walk.
- The entry would be used to translate the specified VA using the EL3 translation regime.

The invalidation ~~only~~ applies to the PE that executes this System instruction.

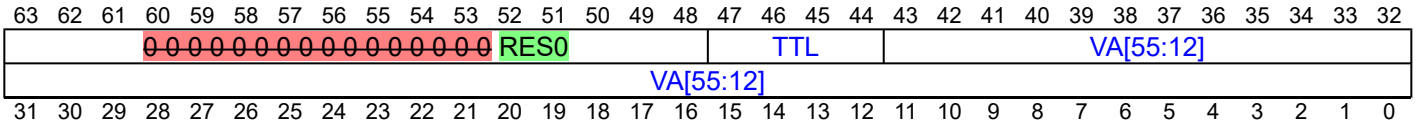
Configuration

Attributes

TLBI VALE3 is a 64-bit System instruction.

Field descriptions

The TLBI VALE3 input value bit assignments are:



Bits [63:48]

Reserved, RES0.

TTL, bits [47:44]

When ARMv8.4-TTL is implemented:

Translation Table Level. Indicates the level of the page table walk that holds the leaf entry for the address being invalidated.

TTL	Meaning
0b00xx	No information supplied as to the translation table level. Hardware must assume that the entry can be from any level. In this case, TTL<1:0> is RES0.
0b01xx	The entry comes from a 4KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3.
0b10xx	The entry comes from a 16KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : Reserved. Treat as if TTL<3:2> is 0b00. 0b10 : Level 2. 0b11 : Level 3.
0b11xx	The entry comes from a 64KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3.

If an incorrect value of the TTL field is specified for the entry being invalidated by the instruction, then no entries are required by the architecture to be invalidated from the TLB.

Otherwise:

Reserved, RES0.

VA[55:12], bits [43:0]

Bits[55:12] of the virtual address to match. Any appropriate TLB entries that match the ASID value (if appropriate) and VA will be affected by this System instruction.~~operation.~~

If the TLB maintenance instructions are targeting a translation regime that is using AArch32, and so has a VA of only 32 bits, then the software must treat bits[55:32] as RES0.

The treatment of the low-order bits of this field depends on the translation granule size, as follows:

- Where a 4KB translation granule is being used, all bits are valid and used for the invalidation.
- Where a 16KB translation granule is being used, bits [1:0] of this field are RES0 and ignored when the instruction is executed, because VA[13:12] have no effect on the operation of the instruction.
- Where a 64KB translation granule is being used, bits [3:0] of this field are RES0 and ignored when the instruction is executed, because VA[15:12] have no effect on the operation of the instruction.

Executing the TLBI VALE3 instruction

Accesses to this instruction use the following encodings:

```
TLBI VALE3{, <Xt>}
```

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b01	0b110	0b1000	0b0111	0b101
~~0b01~~	~~0b1000~~	~~0b110~~	~~0b101~~	~~0b0111~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    TLBI_VALE3(X[t]);
```

27~~13~~/03~~12~~/2019~~2018~~ 21~~16~~:59~~42~~: e5e4db499bf9867a4b93324c4dbac985d3da93766~~379d01e197f1d40720d32d0f84e419e9187e009~~

(old) htmldiff from- (new)

(old) htmldiff from- (new)

TLBI VALE3IS, TLB Invalidate by VA, Last level, EL3, Inner Shareable

The TLBI VALE3IS characteristics are:

Purpose

If EL3 is implemented, invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 1 translation table entry, from the final level of the translation table walk.
- The entry would be used to translate the specified VA using the EL3 translation regime.

The invalidation ~~only~~ applies to all PEs in the same Inner Shareable shareability domain as the PE that executes this System instruction.

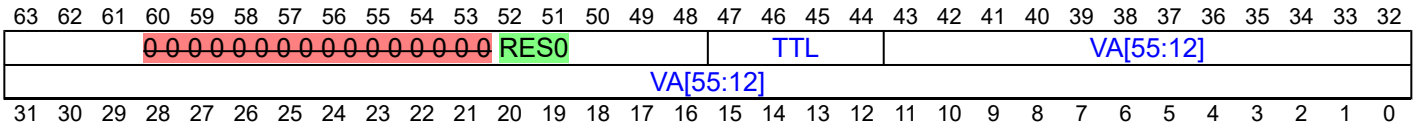
Configuration

Attributes

TLBI VALE3IS is a 64-bit System instruction.

Field descriptions

The TLBI VALE3IS input value bit assignments are:



Bits [63:48]

Reserved, RES0.

TTL, bits [47:44]

When ARMv8.4-TTL is implemented:

Translation Table Level. Indicates the level of the page table walk that holds the leaf entry for the address being invalidated.

TTL	Meaning
0b00xx	No information supplied as to the translation table level. Hardware must assume that the entry can be from any level. In this case, TTL<1:0> is RES0.
0b01xx	The entry comes from a 4KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3.
0b10xx	The entry comes from a 16KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : Reserved. Treat as if TTL<3:2> is 0b00. 0b10 : Level 2. 0b11 : Level 3.
0b11xx	The entry comes from a 64KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3.

If an incorrect value of the TTL field is specified for the entry being invalidated by the instruction, then no entries are required by the architecture to be invalidated from the TLB.

Otherwise:

Reserved, RES0.

VA[55:12], bits [43:0]

Bits[55:12] of the virtual address to match. Any appropriate TLB entries that match the ASID value (if appropriate) and VA will be affected by this System instruction.~~operation.~~

If the TLB maintenance instructions are targeting a translation regime that is using AArch32, and so has a VA of only 32 bits, then the software must treat bits[55:32] as RES0.

The treatment of the low-order bits of this field depends on the translation granule size, as follows:

- Where a 4KB translation granule is being used, all bits are valid and used for the invalidation.
- Where a 16KB translation granule is being used, bits [1:0] of this field are RES0 and ignored when the instruction is executed, because VA[13:12] have no effect on the operation of the instruction.
- Where a 64KB translation granule is being used, bits [3:0] of this field are RES0 and ignored when the instruction is executed, because VA[15:12] have no effect on the operation of the instruction.

Executing the TLBI VALE3IS instruction

Accesses to this instruction use the following encodings:

```
TLBI VALE3IS{, <Xt>}
```

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b01	0b110	0b1000	0b0011	0b101
~~0b01~~	~~0b1000~~	~~0b110~~	~~0b101~~	~~0b0011~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    TLBI_VALE3IS(X[t]);
```

27~~13~~/03~~12~~/2019~~2018~~ 21~~16~~:59~~42~~: e5e4db499bf9867a4b93324c4dbac985d3da93766~~379d01e197f1d40720d32d0f84e419e9187e009~~

(old) htmldiff from- (new)

TLBI VALE3OS, TLB Invalidate by VA, Last level, EL3, Outer Shareable

The TLBI VALE3OS characteristics are:

Purpose

If EL3 is implemented, invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 1 translation table entry, from the final level of the translation table walk.
- The entry would be used to translate the specified VA using the EL3 translation regime.

The invalidation ~~only~~ applies to all PEs in the same Outer Shareable shareability domain as the PE that executes this System instruction.

Configuration

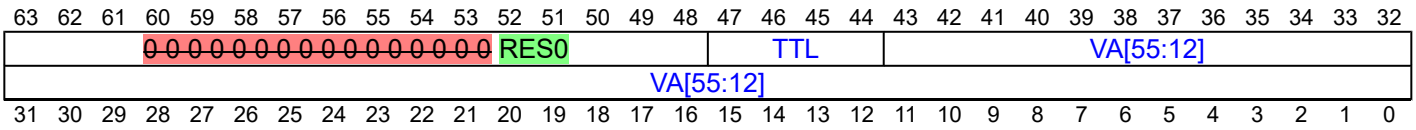
This instruction is present only when ARMv8.4-TLBI is implemented. Otherwise, direct accesses to TLBI VALE3OS are UNDEFINED.

Attributes

TLBI VALE3OS is a 64-bit System instruction.

Field descriptions

The TLBI VALE3OS input value bit assignments are:



Bits [63:48]

Reserved, RES0.

TTL, bits [47:44]

When ARMv8.4-TTL is implemented:

Translation Table Level. Indicates the level of the page table walk that holds the leaf entry for the address being invalidated.

TTL	Meaning
0b00xx	No information supplied as to the translation table level. Hardware must assume that the entry can be from any level. In this case, TTL<1:0> is RES0.
0b01xx	The entry comes from a 4KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3.
0b10xx	The entry comes from a 16KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : Reserved. Treat as if TTL<3:2> is 0b00. 0b10 : Level 2. 0b11 : Level 3.
0b11xx	The entry comes from a 64KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3.

If an incorrect value of the TTL field is specified for the entry being invalidated by the instruction, then no entries are required by the architecture to be invalidated from the TLB.

Otherwise:

Reserved, RES0.

VA[55:12], bits [43:0]

Bits[55:12] of the virtual address to match. Any appropriate TLB entries that match the ASID value (if appropriate) and VA will be affected by this System instruction.~~operation.~~

If the TLB maintenance instructions are targeting a translation regime that is using AArch32, and so has a VA of only 32 bits, then the software must treat bits[55:32] as RES0.

The treatment of the low-order bits of this field depends on the translation granule size, as follows:

- Where a 4KB translation granule is being used, all bits are valid and used for the invalidation.
- Where a 16KB translation granule is being used, bits [1:0] of this field are RES0 and ignored when the instruction is executed, because VA[13:12] have no effect on the operation of the instruction.
- Where a 64KB translation granule is being used, bits [3:0] of this field are RES0 and ignored when the instruction is executed, because VA[15:12] have no effect on the operation of the instruction.

Executing the TLBI VALE3OS instruction

Accesses to this instruction use the following encodings:

```
TLBI VALE3OS{, <Xt>}
```

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b01	0b110	0b1000	0b0001	0b101
~~0b01~~	~~0b1000~~	~~0b110~~	~~0b101~~	~~0b0001~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    TLBI_VALE3OS(X[t]);
```

2713/0312/20192018 2116:5942: e5e4db499bf9867a4b93324c4dbac985d3da93766379d01e197f1d40720d32d0f84e419e9187e009

(old) htmldiff from- (new)

(old) htmldiff from- (new)

TLBI VMALLE1, TLB Invalidate by VMID, All at stage 1, EL1

The TLBI VMALLE1 characteristics are:

Purpose

Invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 1 translation table entry, from any level of the translation table walk.
- When EL2 is implemented and enabled in the ~~current~~ Security state described by the current value of~~:~~ SCR_EL3.NS:
  - If HCR_EL2.{E2H, TGE} is not {1, 1}, the entry would be used with the current VMID and would be required to translate the specified VA using the EL1&0 translation regime.
  - If HCR_EL2.{E2H, TGE} is {1, 1}, the entry would be required to translate the specified VA using the EL2&0 translation regime.
- When EL2 is not implemented or is disabled in the current Security state, the entry would be required to translate the specified VA using the EL1&0 translation regime.

The invalidation ~~only~~ applies to the PE that executes this System instruction.

Note

For the EL1&0 translation regimes, the invalidation applies to both global entries, and non-global entries with any ASID.

Configuration

Attributes

TLBI VMALLE1 is a 64-bit System instruction.

Field descriptions

TLBI VMALLE1 ignores the value in the register specified by the instruction encoding. Software does not have to write a value to the register before issuing this instruction.

Executing the TLBI VMALLE1 instruction

Accesses to this instruction use the following encodings:

```
TLBI VMALLE1{, <Xt>}
```

op0	op1	CRn	CRm	op2	Rt
~~Rt~~	~~op0~~	~~op1~~	~~op2~~	~~CRn~~	~~CRm~~
0b01	0b000	0b1000	0b0111	0b000	0b11111
~~0b11111~~	~~0b01~~	~~0b000~~	~~0b000~~	~~0b1000~~	~~0b0111~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TTLB == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.FB == '1' then
        TLBI_VMALLE1IS();
    else
        TLBI_VMALLE1();
elsif PSTATE.EL == EL2 then
    TLBI_VMALLE1();
elsif PSTATE.EL == EL3 then
    TLBI_VMALLE1();
```

27~~13~~/03~~12~~/2019~~2018~~ 21~~16~~:59~~42~~; e5e4db499bf9867a4b93324c4dbac985d3da93766~~379d01e197f1d40720d32d0f84e419e9187e009~~

(old) htmldiff from- (new)

TLBI VMALLE1IS, TLB Invalidate by VMID, All at stage 1, EL1, Inner Shareable

The TLBI VMALLE1IS characteristics are:

Purpose

Invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 1 translation table entry, from any level of the translation table walk.
- When EL2 is implemented and enabled in the ~~current~~ Security state described by the current value of~~:~~ SCR_EL3.NS:
  - If HCR_EL2.{E2H, TGE} is not {1, 1}, the entry would be used with the current VMID and would be required to translate the specified VA using the EL1&0 translation regime.
  - If HCR_EL2.{E2H, TGE} is {1, 1}, the entry would be required to translate the specified VA using the EL2&0 translation regime.
- When EL2 is not implemented or is disabled in the current Security state, the entry would be required to translate the specified VA using the EL1&0 translation regime.

The invalidation applies to all PEs in the same Inner Shareable shareability domain as the PE that executes this System instruction.~~instructions.~~

Note

From Armv8.4, when a TLB maintenance instruction is generated to the Secure EL1&0 translation regime and is defined to pass a VMID argument, or would be defined to pass a VMID argument if SCR_EL3.EEL2==1, then:

- A PE with SCR_EL3.EEL2==1 is not architecturally required to invalidate any entries in the Secure EL1&0 translation of a PE in the same required shareability domain with SCR_EL3.EEL2==0.
- A PE with SCR_EL3.EEL2==0 is not architecturally required to invalidate any entries in the Secure EL1&0 translation of a PE in the same required shareability domain with SCR_EL3.EEL2==1.
- A PE is architecturally required to invalidate all relevant entries in the Secure EL1&0 translation of a System MMU in the same required shareability domain with a VMID of 0.

For the EL1&0 translation regimes, the invalidation applies to both global entries, and non-global entries with any ASID.

Configuration

Attributes

TLBI VMALLE1IS is a 64-bit System instruction.

Field descriptions

TLBI VMALLE1IS ignores the value in the register specified by the instruction encoding. Software does not have to write a value to the register before issuing this instruction.

Executing the TLBI VMALLE1IS instruction

Accesses to this instruction use the following encodings:

```
TLBI VMALLE1IS{, <Xt>}
```

op0	op1	CRn	CRm	op2	Rt
~~Rt~~	~~op0~~	~~op1~~	~~op2~~	~~CRn~~	~~CRm~~
0b01	0b000	0b1000	0b0011	0b000	0b11111
~~0b11111~~	~~0b01~~	~~0b000~~	~~0b000~~	~~0b1000~~	~~0b0011~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TTLB == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TTLBIS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        TLBI_VMALLE1IS();
elsif PSTATE.EL == EL2 then
    TLBI_VMALLE1IS();
elsif PSTATE.EL == EL3 then
    TLBI_VMALLE1IS();
```

27~~13~~/03~~12~~/2019~~2018~~ 21~~16~~:59~~42~~: e5e4db499bf9867a4b93324c4dbac985d3da93766379d01e197f1d40720d32d0f84e419e9187e009

(old) htmldiff from- (new)

(old) htmldiff from- (new)

TLBI VMALLE1OS, TLB Invalidate by VMID, All at stage 1, EL1, Outer Shareable

The TLBI VMALLE1OS characteristics are:

Purpose

Invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 1 translation table entry, from any level of the translation table walk.
- When EL2 is implemented and enabled in the ~~current~~ Security state described by the current value of~~:~~ SCR_EL3.NS:
  - If HCR_EL2.{E2H, TGE} is not {1, 1}, the entry would be used with the current VMID and would be required to translate the specified VA using the EL1&0 translation regime.
  - If HCR_EL2.{E2H, TGE} is {1, 1}, the entry would be required to translate the specified VA using the EL2&0 translation regime.
- When EL2 is not implemented or is disabled in the current Security state, the entry would be required to translate the specified VA using the EL1&0 translation regime.

The invalidation applies to all PEs in the same Outer Shareable shareability domain as the PE that executes this System instruction.~~instructions.~~

Note

When a TLB maintenance instruction is generated to the Secure EL1&0 translation regime and is defined to pass a VMID argument, or would be defined to pass a VMID argument if SCR_EL3.EEL2==1, then:

- A PE with SCR_EL3.EEL2==1 is not architecturally required to invalidate any entries in the Secure EL1&0 translation of a PE in the same required shareability domain with SCR_EL3.EEL2==0.
- A PE with SCR_EL3.EEL2==0 is not architecturally required to invalidate any entries in the Secure EL1&0 translation of a PE in the same required shareability domain with SCR_EL3.EEL2==1.
- A PE is architecturally required to invalidate all relevant entries in the Secure EL1&0 translation of a System MMU in the same required shareability domain with a VMID of 0.

For the EL1&0 translation regimes, the invalidation applies to both global entries, and non-global entries with any ASID.

Configuration

This instruction is present only when ARMv8.4-TLBI is implemented. Otherwise, direct accesses to TLBI VMALLE1OS are UNDEFINED.

Attributes

TLBI VMALLE1OS is a 64-bit System instruction.

Field descriptions

TLBI VMALLE1OS ignores the value in the register specified by the instruction encoding. Software does not have to write a value to the register before issuing this instruction.

Executing the TLBI VMALLE1OS instruction

Accesses to this instruction use the following encodings:

```
TLBI VMALLE1OS{, <Xt>}
```

op0	op1	CRn	CRm	op2	Rt
~~Rt~~	~~op0~~	~~op1~~	~~op2~~	~~CRn~~	~~CRm~~
0b01	0b000	0b1000	0b0001	0b000	0b11111
~~0b11111~~	~~0b01~~	~~0b000~~	~~0b000~~	~~0b1000~~	~~0b0001~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TTLB == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TTLBOS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        TLBI_VMALLE1OS();
elsif PSTATE.EL == EL2 then
    TLBI_VMALLE1OS();
elsif PSTATE.EL == EL3 then
    TLBI_VMALLE1OS();
```

27~~13~~/03~~12~~/2019~~2018~~ 21~~16~~:59~~42~~; e5e4db499bf9867a4b93324c4dbac985d3da9376~~6379d01e197f1d40720d32d0f84c419e9187e009~~

(old) htmldiff from- (new)

TLBI VMALLS12E1, TLB Invalidate by VMID, All at Stage 1 and 2, EL1

The TLBI VMALLS12E1 characteristics are:

Purpose

Invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 1 or stage 2 translation table entry, from any level of the translation table walk.
- If SCR_EL3.NS is 0, then

  ~~If SCR_EL3.NS is 0 and ARMv8.4-SecEL2 is not implemented, then the instruction invalidates any entry that would be required to translate an address using the Secure EL1&0 translation regime.~~

  - The entry would be required to translate an address using the Secure EL1&0 translation regime.
  - If ARMv8.4-SecEL2 is implemented and enabled, the entry would be used with the current VMID.
- If SCR_EL3.NS is 1, then:
  - The entry would be required to translate an address using the Non-secure EL1&0 translation regime.
  - If Non-secure EL2 is implemented, the entry would be used with the current VMID.

The invalidation ~~only~~ applies to the PE that executes this System instruction.

Note

For the EL1&0 translation regimes, the invalidation applies to both global entries, and non-global entries with any ASID.

Configuration

Attributes

TLBI VMALLS12E1 is a 64-bit System instruction.

Field descriptions

TLBI VMALLS12E1 ignores the value in the register specified by the instruction encoding. Software does not have to write a value to the register before issuing this instruction.

Executing the TLBI VMALLS12E1 instruction

Accesses to this instruction use the following encodings:

```
TLBI VMALLS12E1{, <Xt>}
```

op0	op1	CRn	CRm	op2	Rt
~~Rt~~	~~op0~~	~~op1~~	~~op2~~	~~CRn~~	~~CRm~~
0b01	0b100	0b1000	0b0111	0b110	0b11111
~~0b11111~~	~~0b01~~	~~0b100~~	~~0b110~~	~~0b1000~~	~~0b0111~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    TLBI_VMALLS12E1();
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        TLBI_VMALLE1();
    else
        TLBI_VMALLS12E1();
```

2713/0312/20192018 2116:5942; e5e4db499bf9867a4b93324c4dbac985d3da93766379d01e197f1d40720d32d0f84e419e9187e009

(old) htmldiff from- (new)

TLBI VMALLS12E1IS, TLB Invalidate by VMID, All at Stage 1 and 2, EL1, Inner Shareable

The TLBI VMALLS12E1IS characteristics are:

Purpose

Invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 1 or stage 2 translation table entry, from any level of the translation table walk.
- If SCR_EL3.NS is 0, then

  ~~If SCR_EL3.NS is 0 and ARMv8.4-SecEL2 is not implemented, then the instruction invalidates any entry that would be required to translate an address using the Secure EL1&0 translation regime.~~

  - The entry would be required to translate an address using the Secure EL1&0 translation regime.
  - If ARMv8.4-SecEL2 is implemented and enabled, the entry would be used with the current VMID.
- If SCR_EL3.NS is 1, then:
  - The entry would be required to translate an address using the Non-secure EL1&0 translation regime.
  - If Non-secure EL2 is implemented, the entry would be used with the current VMID.

The invalidation applies to all PEs in the same Inner Shareable shareability domain as the PE that executes this System instruction.~~instructions.~~

Note

From Armv8.4, when a TLB maintenance instruction is generated to the Secure EL1&0 translation regime and is defined to pass a VMID argument, or would be defined to pass a VMID argument if SCR_EL3.EEL2==1, then:

- A PE with SCR_EL3.EEL2==1 is not architecturally required to invalidate any entries in the Secure EL1&0 translation of a PE in the same required shareability domain with SCR_EL3.EEL2==0.
- A PE with SCR_EL3.EEL2==0 is not architecturally required to invalidate any entries in the Secure EL1&0 translation of a PE in the same required shareability domain with SCR_EL3.EEL2==1.
- A PE is architecturally required to invalidate all relevant entries in the Secure EL1&0 translation of a System MMU in the same required shareability domain with a VMID of 0.

For the EL1&0 translation regimes, the invalidation applies to both global entries, and non-global entries with any ASID.

Configuration

Attributes

TLBI VMALLS12E1IS is a 64-bit System instruction.

Field descriptions

TLBI VMALLS12E1IS ignores the value in the register specified by the instruction encoding. Software does not have to write a value to the register before issuing this instruction.

Executing the TLBI VMALLS12E1IS instruction

Accesses to this instruction use the following encodings:

```
TLBI VMALLS12E1IS{, <Xt>}
```

op0	op1	CRn	CRm	op2	Rt
~~Rt~~	~~op0~~	~~op1~~	~~op2~~	~~CRn~~	~~CRm~~
0b01	0b100	0b1000	0b0011	0b110	0b11111
~~0b11111~~	~~0b01~~	~~0b100~~	~~0b110~~	~~0b1000~~	~~0b0011~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    TLBI_VMALLS12E1IS();
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        TLBI_VMALLE1IS();
    else
        TLBI_VMALLS12E1IS();
```

27~~13~~/03~~12~~/2019~~2018~~ 21~~16~~:59~~42~~; e5e4db499bf9867a4b93324c4dbac985d3da93766~~379d01e197f1d40720d32d0f84e419e9187e009~~

(old) htmldiff from- (new)

TLBI VMALLS12E1OS, TLB Invalidate by VMID, All at Stage 1 and 2, EL1, Outer Shareable

The TLBI VMALLS12E1OS characteristics are:

Purpose

Invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

~~When ARMv8.4-TLBI is implemented, invalidates cached copies of translation table entries from TLBs that meet all the following requirements:~~

- The entry is a stage 1 or stage 2 translation table entry, from any level of the translation table walk.
- If SCR_EL3.NS is 0, then

  ~~If SCR_EL3.NS is 0 and ARMv8.4-SecEL2 is not implemented, then the instruction invalidates any entry that would be required to translate an address using the Secure EL1&0 translation regime.~~

  - The entry would be required to translate an address using the Secure EL1&0 translation regime.
  - If ARMv8.4-SecEL2 is implemented and enabled, the entry would be used with the current VMID.
- If SCR_EL3.NS is 1, then:
  - The entry would be required to translate an address using the Non-secure EL1&0 translation regime.
  - If Non-secure EL2 is implemented, the entry would be used with the current VMID.

The invalidation applies to all PEs in the same Outer Shareable shareability domain as the PE that executes this System instruction.~~instructions.~~

Note

When a TLB maintenance instruction is generated to the Secure EL1&0 translation regime and is defined to pass a VMID argument, or would be defined to pass a VMID argument if SCR_EL3.EEL2==1, then:

- A PE with SCR_EL3.EEL2==1 is not architecturally required to invalidate any entries in the Secure EL1&0 translation of a PE in the same required shareability domain with SCR_EL3.EEL2==0.
- A PE with SCR_EL3.EEL2==0 is not architecturally required to invalidate any entries in the Secure EL1&0 translation of a PE in the same required shareability domain with SCR_EL3.EEL2==1.
- A PE is architecturally required to invalidate all relevant entries in the Secure EL1&0 translation of a System MMU in the same required shareability domain with a VMID of 0.

For the EL1&0 translation regimes, the invalidation applies to both global entries, and non-global entries with any ASID.

Configuration

This instruction is present only when ARMv8.4-TLBI is implemented. Otherwise, direct accesses to TLBI VMALLS12E1OS are UNDEFINED.

Attributes

TLBI VMALLS12E1OS is a 64-bit System instruction.

Field descriptions

TLBI VMALLS12E1OS ignores the value in the register specified by the instruction encoding. Software does not have to write a value to the register before issuing this instruction.

Executing the TLBI VMALLS12E1OS instruction

Accesses to this instruction use the following encodings:

```
TLBI VMALLS12E1OS{, <Xt>}
```

op0	op1	CRn	CRm	op2	Rt
~~Rt~~	~~op0~~	~~op1~~	~~op2~~	~~CRn~~	~~CRm~~
0b01	0b100	0b1000	0b0001	0b110	0b11111
~~0b11111~~	~~0b01~~	~~0b100~~	~~0b110~~	~~0b1000~~	~~0b0001~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    TLBI_VMALLS12E1OS();
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        TLBI_VMALLE1OS();
    else
        TLBI_VMALLS12E1OS();
```

2713/0312/20192018 2116:5942; e5e4db499bf9867a4b93324c4dbac985d3da93766379d01e197f1d40720d32d0f84e419e9187e009

(old) htmldiff from- (new)

TRFCR_EL1, Trace Filter Control Register (EL1)

The TRFCR_EL1 characteristics are:

Purpose

Provides EL1 controls for Trace.

Configuration

AArch64 System register TRFCR_EL1 bits [31:0] are architecturally mapped to AArch32 System register TRFCR[31:0] .

This register is present only when ARMv8.4-Trace is implemented. Otherwise, direct accesses to TRFCR_EL1 are UNDEFINED.

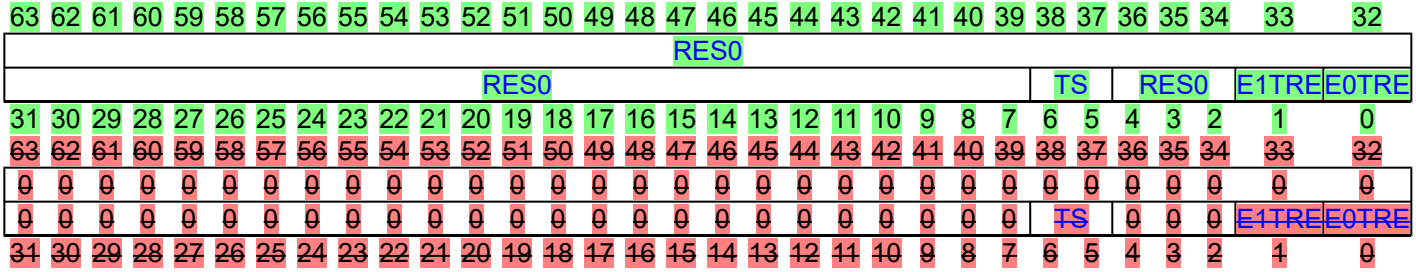
Some or all RW fields of this register have defined reset values. These apply only if the PE resets into an Exception level that is using AArch64. Otherwise, RW fields in this register reset to architecturally UNKNOWN values.

Attributes

TRFCR_EL1 is a 64-bit register.

Field descriptions

The TRFCR_EL1 bit assignments are:



Bits [63:7]

Reserved, RES0.

TS, bits [6:5]

Timestamp Control

TS	Meaning
0b01	Virtual timestamp. The traced timestamp is the physical counter value, minus the value of CNTVOFF_EL2.
0b11	Physical timestamp. The traced timestamp is the physical counter value.

All other values are reserved

This field is ignored if any of the following are true:

- SelfHostedTraceEnabled() == FALSE.
- EL2 is implemented and TRFCR_EL2.TS != 0b00.

Bits [4:2]

Reserved, RES0.

E1TRE, bit [1]

EL1 Trace Enable.

E1TRE	Meaning
0b0	Trace is prohibited at EL1.
0b1	Trace is allowed at EL1.

When SelfHostedTraceEnabled() == FALSE, this field is ignored.

On a Warm reset, this field resets to 0.

E0TRE, bit [0]

EL0 Trace Enable.

E0TRE	Meaning
0b0	Trace is prohibited at EL0.
0b1	Trace is allowed at EL0.

This field is ignored if any of the following are true:

- SelfHostedTraceEnabled() == FALSE.
- EL2 is implemented and enabled in the current Security state and HCR_EL2.TGE == 1.

On a Warm reset, this field resets to 0.

Accessing the TRFCR_EL1

Accesses to this register use the following encodings:

```
MRS <Xt>, TRFCR_EL1
```

op0	op1	CRn	CRm	op2
~~CRn~~	~~op0~~	~~op1~~	~~op2~~	~~CRm~~
0b11	0b000	0b0001	0b0010	0b001
~~0b0001~~	~~0b11~~	~~0b000~~	~~0b001~~	~~0b0010~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TTRF == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TTRF == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<NV2,NV1,NV> == '111' then
        return NVMem[0x880];
    else
        return TRFCR_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TTRF == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HCR_EL2.E2H == '1' then
        return TRFCR_EL2;
    else
        return TRFCR_EL1;
elsif PSTATE.EL == EL3 then
    return TRFCR_EL1;
```

```
MSR TRFCR_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
~~CRn~~	~~op0~~	~~op1~~	~~op2~~	~~CRm~~
0b11	0b000	0b0001	0b0010	0b001
~~0b0001~~	~~0b11~~	~~0b000~~	~~0b001~~	~~0b0010~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TTRF == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TTRF == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<NV2,NV1,NV> == '111' then
        NVMem[0x880] = X[t];
    else
        TRFCR_EL1 = X[t];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TTRF == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HCR_EL2.E2H == '1' then
        TRFCR_EL2 = X[t];
    else
        TRFCR_EL1 = X[t];
elsif PSTATE.EL == EL3 then
    TRFCR_EL1 = X[t];
```

```
MRS <Xt>, TRFCR_EL12
```

op0	op1	CRn	CRm	op2
~~CRn~~	~~op0~~	~~op1~~	~~op2~~	~~CRm~~
0b11	0b101	0b0001	0b0010	0b001
~~0b0001~~	~~0b11~~	~~0b101~~	~~0b001~~	~~0b0010~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '101' then
        return NVMem[0x880];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if EL2Enabled() && HCR_EL2.E2H == '1' then
    if PSTATE.EL == EL2 then
        if HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TTRF == '1' then
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return TRFCR_EL1;
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if EL2Enabled() && HCR_EL2.E2H == '1' then
        return TRFCR_EL1;
else
    else
        UNDEFINED;
```

```
MSR TRFCR_EL12, <Xt>
```

op0	op1	CRn	CRm	op2
~~CRn~~	~~op0~~	~~op1~~	~~op2~~	~~CRm~~
0b11	0b101	0b0001	0b0010	0b001
~~0b0001~~	~~0b11~~	~~0b101~~	~~0b001~~	~~0b0010~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '101' then
        NVMem[0x880] = X[t];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if EL2Enabled() && HCR_EL2.E2H == '1' then
    if PSTATE.EL == EL2 then
        if HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TTRF == '1' then
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            TRFCR_EL1 = X[t];
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if EL2Enabled() && HCR_EL2.E2H == '1' then
        TRFCR_EL1 = X[t];
else
    else
        UNDEFINED;
```

27~~13~~/03~~12~~/2019~~2018~~ 21~~16~~:59~~42~~; e5e4db499bf9867a4b93324c4dbac985d3da93766379d01e197f1d40720d32d0f84c419c9187c009

(old) htmldiff from- (new)

TTBR0_EL1, Translation Table Base Register 0 (EL1)

The TTBR0_EL1 characteristics are:

Purpose

Holds the base address of the translation table for the initial lookup for stage 1 of the translation of an address from the lower VA range in the EL1&0 translation regime, and other information for this translation regime.

Configuration

AArch64 System register TTBR0_EL1 bits [63:0] are architecturally mapped to AArch32 System register TTBR0[63:0] .

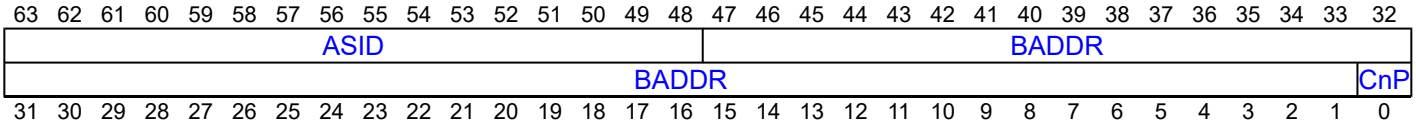
RW fields in this register reset to architecturally UNKNOWN values.

Attributes

TTBR0_EL1 is a 64-bit register.

Field descriptions

The TTBR0_EL1 bit assignments are:



ASID, bits [63:48]

An ASID for the translation table base address. The TCR_EL1.A1 field selects either TTBR0_EL1.ASID or TTBR1_EL1.ASID.

If the implementation has only 8 bits of ASID, then the upper 8 bits of this field are RES0.

This field resets to an architecturally UNKNOWN value.

BADDR, bits [47:1]

Translation table base address, A[47:x] or A[51:x], bits[47:1].

Note

- Translation table base addresses of 52 bits, A[51:x], are supported only in an implementation that includes ARMv8.2-LPA and is using the 64KB translation granule.
- A translation table must be aligned to the size of the table, except that when using a translation table base address larger than 48 bits the minimum alignment of a table containing fewer than eight entries is 64 bytes.

In an implementation that includes ARMv8.2-LPA, if the value of TCR_EL1.IPS is `0b110`, then:

- Register bits[47:z] hold bits[47:z] of the stage 1 translation table base address, where z is determined as follows:
  - If x >= 6 then z=x.
  - Otherwise, z=6.
- Register bits[5:2] hold bits[51:48] of the stage 1 translation table base address.
- When z>x register bits[(z-1):x] are RES0, and bits[(z-1):x] of the translation table base address are zero.
- When x>6 register bits[(x-1):6] are RES0.
- Register bit[1] is RES0.
- Bits[5:2] of the stage 1 translation table base address are zero.

- In an implementation that includes ARMv8.2-TTCNP bit[0] of the stage 1 translation table base address is zero.

Note

- In an implementation that includes ARMv8.2-LPA a TCR_EL1.IPS value of `0b110`, that selects an IPA size of 52 bits, is permitted only when using the 64KB translation granule.
- When the value of ID_AA64MMFR0_EL1.PARange indicates that the implementation does not support a 52 bit PA size, if a translation table lookup uses this register with the 64KB translation granule when the value of TCR_EL1.IPS is `0b110` and the value of register bits[5:2] is nonzero, an Address size fault is generated.

If the Effective value of TCR_EL1.IPS is not `0b110` then:

- Register bits[47:x] hold bits[47:x] of the stage 1 translation table base address.
- Register bits[(x-1):1] are RES0.
- If the implementation supports 52-bit PAs and IPAs, then bits[51:48] of the translation table base addresses used in this stage of translation are `0b0000`.

Note

This definition applies:

- To an implementation that includes ARMv8.2-LPA and is using a translation granule smaller than 64KB.
- To any implementation that does not include ARMv8.2-LPA.

If any TTBR0_EL1[47:0] bit that is defined as RES0 has the value 1 when a translation table walk is performed using TTBR0_EL1, then the translation table base address might be misaligned, with effects that are CONSTRAINED UNPREDICTABLE, and must be one of the following:

- Bits[x-1:0] of the translation table base address are treated as if all the bits are zero. The value read back from the corresponding register bits is either the value written to the register or zero.
- The result of the calculation of an address for a translation table walk using this register can be corrupted in those bits that are nonzero.

The AArch64 Virtual Memory System Architecture chapter describes how x is calculated based on the value of TCR_EL1.T0SZ, the stage of translation, and the translation granule size.

This field resets to an architecturally UNKNOWN value.

CnP, bit [0]

When ARMv8.2-TTCNP is implemented:

Common not Private. This bit indicates whether each entry that is pointed to by TTBR0_EL1 is a member of a common set that can be used by every PE in the Inner Shareable domain for which the value of TTBR0_EL1.CnP is 1.

CnP	Meaning
`0b0`	The translation table entries pointed to by TTBR0_EL1, for the current translation regime and ASID, are permitted to differ from corresponding entries for TTBR0_EL1 for other PEs in the Inner Shareable domain. This is not affected by: • The value of TTBR0_EL1.CnP on those other PEs. • The value of the current ASID. • If EL2 is implemented and enabled in the current Security state, the value of the current VMID.
`0b1`	The translation table entries pointed to by TTBR0_EL1 are the same as the translation table entries for every other PE in the Inner Shareable domain for which the value of TTBR0_EL1.CnP is 1 and all of the following apply: • The translation table entries are pointed to by TTBR0_EL1. • The translation tables relate to the same translation regime. • The ASID is the same as the current ASID. • If EL2 is implemented and enabled in the current Security state, the value of the current VMID.

This field is permitted to be cached in a TLB.

When a TLB combines entries from stage 1 translation and stage 2 translation into a single entry, that entry can only be shared between different PEs if the value of the CnP bit is 1 for both stage 1 and stage 2.

Note

If the value of the TTBR0_EL1.CnP bit is 1 on multiple PEs in the same Inner Shareable domain and those TTBR0_EL1s do not point to the same translation table entries when the other conditions specified for the case when the value of CnP is 1 apply, then the results of translations are CONSTRAINED UNPREDICTABLE, see 'CONSTRAINED UNPREDICTABLE behaviors due to caching of control or data values' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

This field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Accessing the TTBR0_EL1

When HCR_EL2.E2H is 1, without explicit synchronization, access from EL3 using the mnemonic TTBR0_EL1 or TTBR0_EL12 are not guaranteed to be ordered with respect to accesses using the other mnemonic.

Accesses to this register use the following encodings:

```
MRS <Xt>, TTBR0_EL1
```

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b11	0b000	0b0010	0b0000	0b000
~~0b11~~	~~0b0010~~	~~0b000~~	~~0b000~~	~~0b0000~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TRVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<NV2,NV1,NV> == '111' then
        return NVMem[0x200];
    else
        return TTBR0_EL1;
elsif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' then
        return TTBR0_EL2;
    else
        return TTBR0_EL1;
elsif PSTATE.EL == EL3 then
    return TTBR0_EL1;
```

```
MSR TTBR0_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b11	0b000	0b0010	0b0000	0b000
~~0b11~~	~~0b0010~~	~~0b000~~	~~0b000~~	~~0b0000~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<NV2,NV1,NV> == '111' then
        NVMem[0x200] = X[t];
    else
        TTBR0_EL1 = X[t];
elsif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' then
        TTBR0_EL2 = X[t];
    else
        TTBR0_EL1 = X[t];
elsif PSTATE.EL == EL3 then
    TTBR0_EL1 = X[t];
```

MRS <Xt>, TTBR0_EL12

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b11	0b101	0b0010	0b0000	0b000
~~0b11~~	~~0b0010~~	~~0b101~~	~~0b000~~	~~0b0000~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '101' then
        return NVMem[0x200];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if EL2Enabled() && HCR_EL2.E2H == '1' then
    if PSTATE.EL == EL2 then
        return TTBR0_EL1;
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if EL2Enabled() && HCR_EL2.E2H == '1' then
        return TTBR0_EL1;
else
    else
        UNDEFINED;
```

MSR TTBR0_EL12, <Xt>

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b11	0b101	0b0010	0b0000	0b000
~~0b11~~	~~0b0010~~	~~0b101~~	~~0b000~~	~~0b0000~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '101' then
        NVMem[0x200] = X[t];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if EL2Enabled() && HCR_EL2.E2H == '1' then
    if PSTATE.EL == EL2 then
        TTBR0_EL1 = X[t];
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if EL2Enabled() && HCR_EL2.E2H == '1' then
        TTBR0_EL1 = X[t];
else
    else
        UNDEFINED;
```

2713/0312/20192018 2116:5942; e5e4db499bf9867a4b93324c4dbac985d3da93766379d01e197f1d40720d32d0f84e419e9187e009

(old) htmldiff from- (new)

TTBR1_EL1, Translation Table Base Register 1 (EL1)

The TTBR1_EL1 characteristics are:

Purpose

Holds the base address of the translation table for the initial lookup for stage 1 of the translation of an address from the higher VA range in the EL1&0 stage 1 translation regime, and other information for this translation regime.

Configuration

AArch64 System register TTBR1_EL1 bits [63:0] are architecturally mapped to AArch32 System register TTBR1[63:0] .

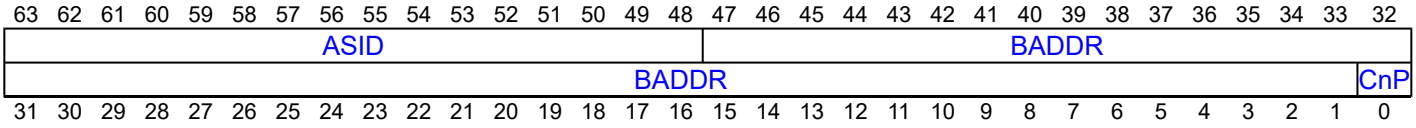
RW fields in this register reset to architecturally UNKNOWN values.

Attributes

TTBR1_EL1 is a 64-bit register.

Field descriptions

The TTBR1_EL1 bit assignments are:



ASID, bits [63:48]

An ASID for the translation table base address. The TCR_EL1.A1 field selects either TTBR0_EL1.ASID or TTBR1_EL1.ASID.

If the implementation has only 8 bits of ASID, then the upper 8 bits of this field are RES0.

This field resets to an architecturally UNKNOWN value.

BADDR, bits [47:1]

Translation table base address, A[47:x] or A[51:x], bits[47:1].

Note

- Translation table base addresses of 52 bits, A[51:x], are supported only in an implementation that includes ARMv8.2-LPA and is using the 64KB translation granule.
- A translation table must be aligned to the size of the table, except that when using a translation table base address larger than 48 bits the minimum alignment of a table containing fewer than eight entries is 64 bytes.

In an implementation that includes ARMv8.2-LPA, if the value of TCR_EL1.IPS is `0b110`, then:

- Register bits[47:z] hold bits[47:z] of the stage 1 translation table base address, where z is determined as follows:
  - If x >= 6 then z=x.
  - Otherwise, z=6.
- Register bits[5:2] hold bits[51:48] of the stage 1 translation table base address.
- When z>x register bits[(z-1):x] are RES0, and bits[(z-1):x] of the translation table base address are zero.
- When x>6 register bits[(x-1):6] are RES0.
- Register bit[1] is RES0.
- Bits[5:2] of the stage 1 translation table base address are zero.

- In an implementation that includes ARMv8.2-TTCNP bit[0] of the stage 1 translation table base address is zero.

Note

- In an implementation that includes ARMv8.2-LPA a TCR_EL1.IPS value of `0b110`, that selects an IPA size of 52 bits, is permitted only when using the 64KB translation granule.
- When the value of ID_AA64MMFR0_EL1.PARange indicates that the implementation does not support a 52 bit PA size, if a translation table lookup uses this register with the 64KB translation granule when the value of TCR_EL1.IPS is `0b110` and the value of register bits[5:2] is nonzero, an Address size fault is generated.

If the Effective value of TCR_EL1.IPS is not `0b110` then:

- Register bits[47:x] hold bits[47:x] of the stage 1 translation table base address.
- Register bits[(x-1):1] are RES0.
- If the implementation supports 52-bit PAs and IPAs, then bits[51:48] of the translation table base addresses used in this stage of translation are `0b0000`.

Note

This definition applies:

- To an implementation that includes ARMv8.2-LPA and is using a translation granule smaller than 64KB.
- To any implementation that does not include ARMv8.2-LPA.

If any TTBR1_EL1[47:0] bit that is defined as RES0 has the value 1 when a translation table walk is performed using TTBR1_EL1, then the translation table base address might be misaligned, with effects that are CONSTRAINED UNPREDICTABLE, and must be one of the following:

- Bits[x-1:0] of the translation table base address are treated as if all the bits are zero. The value read back from the corresponding register bits is either the value written to the register or zero.
- The result of the calculation of an address for a translation table walk using this register can be corrupted in those bits that are nonzero.

The AArch64 Virtual Memory System Architecture chapter describes how x is calculated based on the value of TCR_EL1.T1SZ, the stage of translation, and the translation granule size.

This field resets to an architecturally UNKNOWN value.

CnP, bit [0]

When ARMv8.2-TTCNP is implemented:

Common not Private. This bit indicates whether each entry that is pointed to by TBR1_EL1 is a member of a common set that can be used by every PE in the Inner Shareable domain for which the value of TTBR1_EL1.CnP is 1.

CnP	Meaning
`0b0`	The translation table entries pointed to by TTBR1_EL1, for the current translation regime and ASID, are permitted to differ from corresponding entries for TTBR1_EL1 for other PEs in the Inner Shareable domain. This is not affected by: • The value of TTBR1_EL1.CnP on those other PEs. • The value of the current ASID. • If EL2 is implemented and enabled in the current Security state, the value of the current VMID.
`0b1`	The translation table entries pointed to by TTBR1_EL1 are the same as the translation table entries for every other PE in the Inner Shareable domain for which the value of TTBR1_EL1.CnP is 1 and all of the following apply: • The translation table entries are pointed to by TTBR1_EL1. • The translation tables relate to the same translation regime. • The ASID is the same as the current ASID. • If EL2 is implemented and enabled in the current Security state, the value of the current VMID.

This field is permitted to be cached in a TLB.

When a TLB combines entries from stage 1 translation and stage 2 translation into a single entry, that entry can only be shared between different PEs if the value of the CnP bit is 1 for both stage 1 and stage 2.

Note

> If the value of the TTBR1_EL1.CnP bit is 1 on multiple PEs in the same Inner Shareable domain and those TTBR1_EL1s do not point to the same translation table entries when the other conditions specified for the case when the value of CnP is 1 apply, then the results of translations are CONSTRAINED UNPREDICTABLE, see 'CONSTRAINED UNPREDICTABLE behaviors due to caching of control or data values' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

This field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Accessing the TTBR1_EL1

When HCR_EL2.E2H is 1, without explicit synchronization, access from EL3 using the mnemonic TTBR1_EL1 or TTBR1_EL12 are not guaranteed to be ordered with respect to accesses using the other mnemonic.

Accesses to this register use the following encodings:

```
MRS <Xt>, TTBR1_EL1
```

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b11	0b000	0b0010	0b0000	0b001
~~0b11~~	~~0b0010~~	~~0b000~~	~~0b001~~	~~0b0000~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TRVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<NV2,NV1,NV> == '111' then
        return NVMem[0x210];
    else
        return TTBR1_EL1;
elsif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' then
        return TTBR1_EL2;
    else
        return TTBR1_EL1;
elsif PSTATE.EL == EL3 then
    return TTBR1_EL1;
```

```
MSR TTBR1_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b11	0b000	0b0010	0b0000	0b001
~~0b11~~	~~0b0010~~	~~0b000~~	~~0b001~~	~~0b0000~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<NV2,NV1,NV> == '111' then
        NVMem[0x210] = X[t];
    else
        TTBR1_EL1 = X[t];
elsif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' then
        TTBR1_EL2 = X[t];
    else
        TTBR1_EL1 = X[t];
elsif PSTATE.EL == EL3 then
    TTBR1_EL1 = X[t];
```

MRS <Xt>, TTBR1_EL12

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b11	0b101	0b0010	0b0000	0b001
~~0b11~~	~~0b0010~~	~~0b101~~	~~0b001~~	~~0b0000~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '101' then
        return NVMem[0x210];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if EL2Enabled() && HCR_EL2.E2H == '1' then
    if PSTATE.EL == EL2 then
        return TTBR1_EL1;
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if EL2Enabled() && HCR_EL2.E2H == '1' then
        return TTBR1_EL1;
else
    else
        UNDEFINED;
```

MSR TTBR1_EL12, <Xt>

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b11	0b101	0b0010	0b0000	0b001
~~0b11~~	~~0b0010~~	~~0b101~~	~~0b001~~	~~0b0000~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '101' then
        NVMem[0x210] = X[t];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if EL2Enabled() && HCR_EL2.E2H == '1' then
    if PSTATE.EL == EL2 then
        TTBR1_EL1 = X[t];
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if EL2Enabled() && HCR_EL2.E2H == '1' then
        TTBR1_EL1 = X[t];
else
    else
        UNDEFINED;
```

2713/0312/20192018 2116:5942; e5e4db499bf9867a4b93324c4dbac985d3da93766379d01e197f1d40720d32d0f84e419e9187e009

(old) htmldiff from- (new)

VBAR_EL1, Vector Base Address Register (EL1)

The VBAR_EL1 characteristics are:

Purpose

Holds the vector base address for any exception that is taken to EL1.

Configuration

AArch64 System register VBAR_EL1 bits [31:0] are architecturally mapped to AArch32 System register VBAR[31:0] .

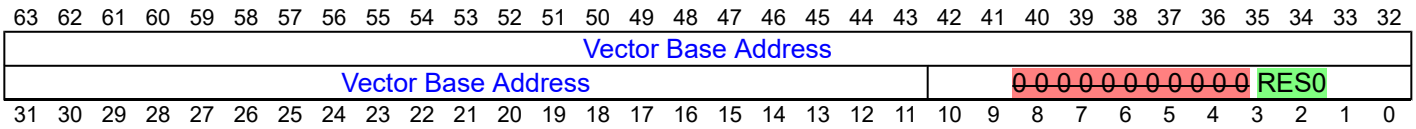
RW fields in this register reset to architecturally UNKNOWN values.

Attributes

VBAR_EL1 is a 64-bit register.

Field descriptions

The VBAR_EL1 bit assignments are:



Bits [63:11]

Vector Base Address. Base address of the exception vectors for exceptions taken to EL1.

If the implementation does not support ARMv8.2-LVA, then:

- If tagged addresses are being used, bits [55:48] of VBAR_EL1 must be the same or else the use of the vector address will result in a recursive exception.
- If tagged addresses are not being used, bits [63:48] of VBAR_EL1 must be the same or else the use of the vector address will result in a recursive exception.

If the implementation supports ARMv8.2-LVA, then:

- If tagged addresses are being used, bits [55:52] of VBAR_EL1 must be the same or else the use of the vector address will result in a recursive exception.
- If tagged addresses are not being used, bits [63:52] of VBAR_EL1 must be the same or else the use of the vector address will result in a recursive exception.

This field resets to an architecturally UNKNOWN value.

Bits [10:0]

Reserved, RES0.

Accessing the VBAR_EL1

When HCR_EL2.E2H is 1, without explicit synchronization, access from EL3 using the mnemonic VBAR_EL1 or VBAR_EL12 are not guaranteed to be ordered with respect to accesses using the other mnemonic.

Accesses to this register use the following encodings:

MRS <Xt>, VBAR_EL1

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b11	0b000	0b1100	0b0000	0b000
~~0b11~~	~~0b1100~~	~~0b000~~	~~0b000~~	~~0b0000~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV1> == '01' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<NV2,NV1,NV> == '111' then
        return NVMem[0x250];
    else
        return VBAR_EL1;
elsif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' then
        return VBAR_EL2;
    else
        return VBAR_EL1;
elsif PSTATE.EL == EL3 then
    return VBAR_EL1;
```

MSR VBAR_EL1, <Xt>

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b11	0b000	0b1100	0b0000	0b000
~~0b11~~	~~0b1100~~	~~0b000~~	~~0b000~~	~~0b0000~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV1> == '01' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<NV2,NV1,NV> == '111' then
        NVMem[0x250] = X[t];
    else
        VBAR_EL1 = X[t];
elsif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' then
        VBAR_EL2 = X[t];
    else
        VBAR_EL1 = X[t];
elsif PSTATE.EL == EL3 then
    VBAR_EL1 = X[t];
```

MRS <Xt>, VBAR_EL12

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b11	0b101	0b1100	0b0000	0b000
~~0b11~~	~~0b1100~~	~~0b101~~	~~0b000~~	~~0b0000~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '101' then
        return NVMem[0x250];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if EL2Enabled() && HCR_EL2.E2H == '1' then
    if PSTATE.EL == EL2 then
        return VBAR_EL1;
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if EL2Enabled() && HCR_EL2.E2H == '1' then
        return VBAR_EL1;
else
    else
        UNDEFINED;
```

```
MSR VBAR_EL12, <Xt>
```

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b11	0b101	0b1100	0b0000	0b000
~~0b11~~	~~0b1100~~	~~0b101~~	~~0b000~~	~~0b0000~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '101' then
        NVMem[0x250] = X[t];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if EL2Enabled() && HCR_EL2.E2H == '1' then
    if PSTATE.EL == EL2 then
        VBAR_EL1 = X[t];
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if EL2Enabled() && HCR_EL2.E2H == '1' then
        VBAR_EL1 = X[t];
else
    else
        UNDEFINED;
```

27~~13~~/03~~12~~/2019~~2018~~ 21~~16~~:59~~42~~; e5e4db499bf9867a4b93324c4dbac985d3da93766~~379d01e197f1d40720d32d0f84e419e9187e009~~

VDISR_EL2, Virtual Deferred Interrupt Status Register

The VDISR_EL2 characteristics are:

Purpose

Records that a virtual SError interrupt has been consumed by an ESB instruction executed at EL1.

An indirect write to VDISR_EL2 made by an ESB instruction does not require an explicit synchronization operation for the value written to be observed by a direct read of DISR_EL1 or DISR occurring in program order after the ESB instruction.

Configuration

AArch64 System register VDISR_EL2 bits [31:0] are architecturally mapped to AArch32 System register VDISR[31:0] .

This register is present only when RAS is implemented. Otherwise, direct accesses to VDISR_EL2 are UNDEFINED.

If EL2 is not implemented, this register is RES0 from EL3.

This register has no effect if EL2 is not enabled in the current Security state.

RW fields in this register reset to architecturally UNKNOWN values.

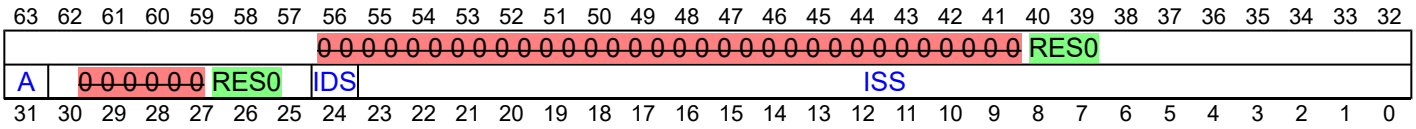
Attributes

VDISR_EL2 is a 64-bit register.

Field descriptions

The VDISR_EL2 bit assignments are:

When !ELUsingAArch32(EL1):



Bits [63:32]

Reserved, RES0.

A, bit [31]

Set to 1 when an ESB instruction defers a virtual SError interrupt.

This field resets to an architecturally UNKNOWN value.

Bits [30:25]

Reserved, RES0.

IDS, bit [24]

The value copied from VSESR_EL2.IDS.

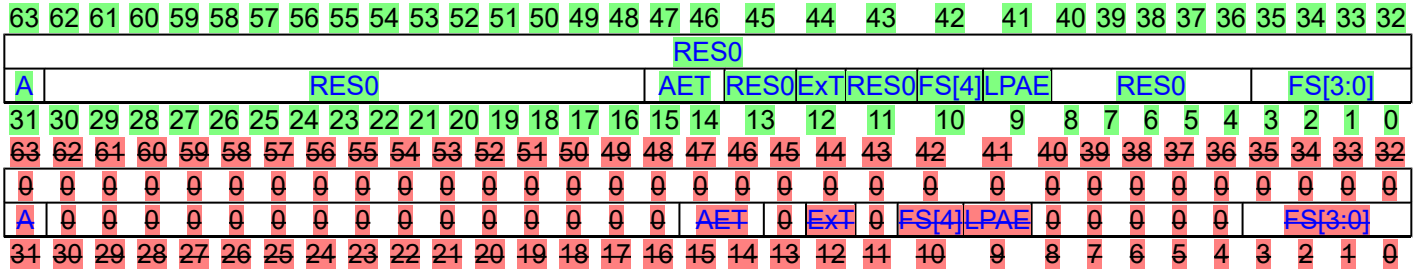
This field resets to an architecturally UNKNOWN value.

ISS, bits [23:0]

The value copied from VSESR_EL2.ISS.

This field resets to an architecturally UNKNOWN value.

When !ELUsingAArch32(EL1) and VDISR_EL2.LPAE == 0:



Bits [63:32]

Reserved, RES0.

A, bit [31]

Set to 1 when an ESB instruction defers a virtual SError interrupt.

This field resets to an architecturally UNKNOWN value.

Bits [30:16]

Reserved, RES0.

AET, bits [15:14]

The value copied from VSESR_EL2.AET.

This field resets to an architecturally UNKNOWN value.

Bit [13]

Reserved, RES0.

ExT, bit [12]

The value copied from VSESR_EL2.ExT.

This field resets to an architecturally UNKNOWN value.

Bit [11]

Reserved, RES0.

FS[4], bit [10]

This field is bit[4] of FS[4:0].

Fault status code. Set to `0b10110` when an ESB instruction defers a virtual SError interrupt.

FS	Meaning
0b10110	Asynchronous SError interrupt.

All other values are reserved.

The FS field is split as follows:

- FS[4] is VDISR_EL2[10].
- FS[3:0] is VDISR_EL2[3:0].

This field resets to an architecturally UNKNOWN value.

LPAE, bit [9]

Format.

Set to TTBCR.EAE when an ESB instruction defers a virtual SError interrupt.

LPAE	Meaning
0b0	Using the Short-descriptor translation table format.

This field resets to an architecturally UNKNOWN value.

Bits [8:4]

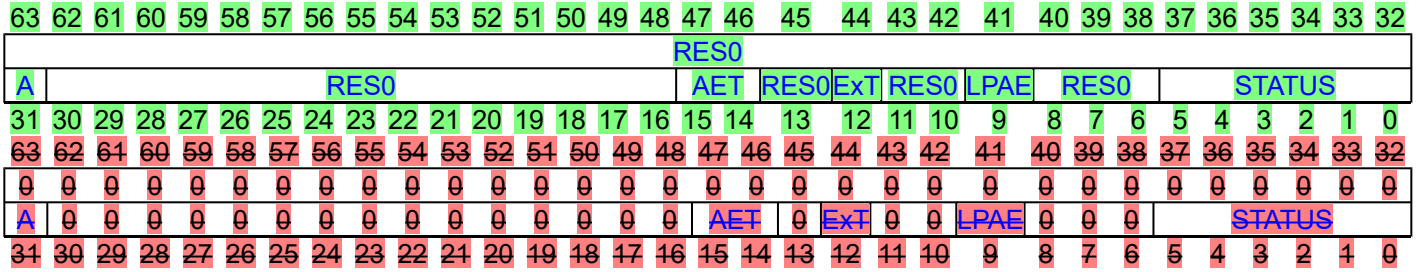
Reserved, RES0.

FS[3:0], bits [3:0]

This field is bits[3:0] of FS[4:0].

See FS[4] for the field description.

When ELUsingAArch32(EL1) and VDISR_EL2.LPAE == 1:



Bits [63:32]

Reserved, RES0.

A, bit [31]

Set to 1 when an ESB instruction defers a virtual SError interrupt.

This field resets to an architecturally UNKNOWN value.

Bits [30:16]

Reserved, RES0.

AET, bits [15:14]

The value copied from VSESR_EL2.AET.

This field resets to an architecturally UNKNOWN value.

Bit [13]

Reserved, RES0.

ExT, bit [12]

The value copied from VSESR_EL2.ExT.

This field resets to an architecturally UNKNOWN value.

Bits [11:10]

Reserved, RES0.

LPAE, bit [9]

Format.

Set to TTBCR.EAE when an ESB instruction defers a virtual SError interrupt.

LPAE	Meaning
0b1	Using the Long-descriptor translation table format.

This field resets to an architecturally UNKNOWN value.

Bits [8:6]

Reserved, RES0.

STATUS, bits [5:0]

Fault status code. Set to 0b010001 when an ESB instruction defers a virtual SError interrupt.

STATUS	Meaning
0b010001	Asynchronous SError interrupt.

All other values are reserved.

This field resets to an architecturally UNKNOWN value.

Accessing the VDISR_EL2

An indirect write to VDISR_EL2 made by an ESB instruction does not require an explicit synchronization operation for the value that is written to be observed by a direct read of DISR_EL1 or DISR occurring in program order after the ESB instruction.

Accesses to this register use the following encodings:

```
MRS <Xt>, VDISR_EL2
```

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b11	0b100	0b1100	0b0001	0b001
~~0b11~~	~~0b1100~~	~~0b100~~	~~0b001~~	~~0b0001~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
        return NVMem[0x500];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    return VDISR_EL2;
elsif PSTATE.EL == EL3 then
    return VDISR_EL2;
```

MSR VDISR_EL2, <Xt>

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b11	0b100	0b1100	0b0001	0b001
~~0b11~~	~~0b1100~~	~~0b100~~	~~0b001~~	~~0b0001~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
        NVMem[0x500] = X[t];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    VDISR_EL2 = X[t];
elsif PSTATE.EL == EL3 then
    VDISR_EL2 = X[t];
```

MRS <Xt>, DISR_EL1

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b11	0b000	0b1100	0b0001	0b001
~~0b11~~	~~0b1100~~	~~0b000~~	~~0b001~~	~~0b0001~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.AMO == '1' then
        return VDISR_EL2;
    else
        return DISR_EL1;
elsif PSTATE.EL == EL2 then
    return DISR_EL1;
elsif PSTATE.EL == EL3 then
    return DISR_EL1;
```

MSR DISR_EL1, <Xt>

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b11	0b000	0b1100	0b0001	0b001

0b11	0b1100	0b000	0b001	0b0001

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.AMO == '1' then
        VDISR_EL2 = X[t];
    else
        DISR_EL1 = X[t];
elsif PSTATE.EL == EL2 then
    DISR_EL1 = X[t];
elsif PSTATE.EL == EL3 then
    DISR_EL1 = X[t];
```

2713/0312/20192018 2116:5942: e5e4db499bf9867a4b93324c4dbac985d3da93766379d01e197f1d40720d32d0f84c419e9187e009

(old) htmldiff from- (new)

VSESR_EL2, Virtual SError~~Deferred~~ Exception~~Interrupt~~ Syndrome~~Status~~ Register

The VSESR_EL2 characteristics are:

Purpose

Provides the syndrome value reported to software on taking a virtual SError interrupt exception to EL1, or on executing an ESB instruction at EL1.

When the virtual SError interrupt is taken to EL1 using AArch64, then the syndrome value is reported in ESR_EL1.

When the virtual SError interrupt is taken to EL1 using AArch32, then the syndrome value is reported in DFSR.{AET, ExT} and the remainder of DFSR is set as defined by VMSAv8-32. For more information, see The AArch32 Virtual Memory System Architecture.

When the virtual SError interrupt is deferred by an ESB instruction, then the syndrome value is written to VDISR_EL2.

Configuration

AArch64 System register VSESR_EL2 bits [31:0] are architecturally mapped to AArch32 System register VDFSR[31:0] .

This register is present only when RAS is implemented. Otherwise, direct accesses to VSESR_EL2 are UNDEFINED.

If EL2 is not implemented, this register is RES0 from EL3.

This register has no effect if EL2 is not enabled in the current Security state.

RW fields in this register reset to architecturally UNKNOWN values.

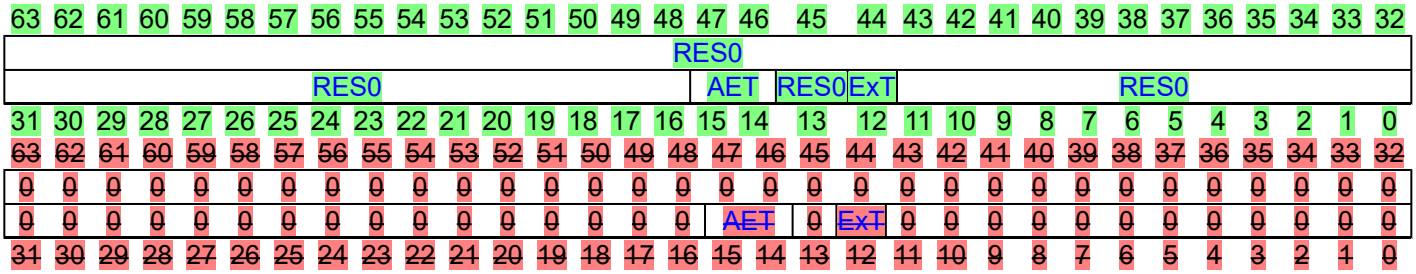
Attributes

VSESR_EL2 is a 64-bit register.

Field descriptions

The VSESR_EL2 bit assignments are:

When ELUsingAArch32(EL1):



Bits [63:16]

Reserved, RES0.

AET, bits [15:14]

When a virtual SError interrupt is taken to EL1 using AArch32, DFSR[15:4] is set to VSESR_EL2.AET.

When a virtual SError interrupt is deferred by an ESB instruction, VDISR_EL2[15:4] is set to VSESR_EL2.AET.

This field resets to an architecturally UNKNOWN value.

Bit [13]

Reserved, RES0.

ExT, bit [12]

When a virtual SError interrupt is taken to EL1 using AArch32, DFSR[12] is set to VSESR_EL2.ExT.

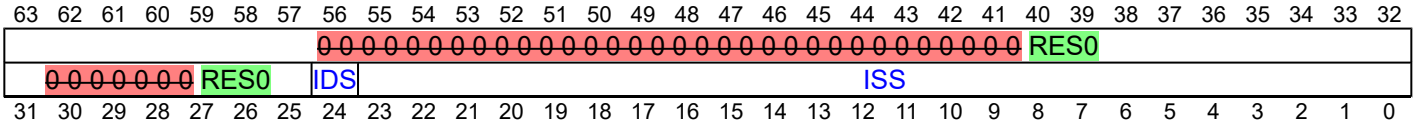
When a virtual SError interrupt is deferred by an ESB instruction, VDISR_EL2[12] is set to VSESR_EL2.ExT.

This field resets to an architecturally UNKNOWN value.

Bits [11:0]

Reserved, RES0.

When !ELUsingAArch32(EL1):



Bits [63:25]

Reserved, RES0.

IDS, bit [24]

When a virtual SError interrupt is taken to EL1 using AArch64, ESR_EL1[24] is set to VSESR_EL2.IDS.

When a virtual SError interrupt is deferred by an ESB instruction, VDISR_EL2[24] is set to VSESR_EL2.IDS.

This field resets to an architecturally UNKNOWN value.

ISS, bits [23:0]

When a virtual SError interrupt is taken to EL1 using AArch64, ESR_EL1[23:0] is set to VSESR_EL2.ISS.

When a virtual SError interrupt is deferred by an ESB instruction, VDISR_EL2[23:0] is set to VSESR_EL2.ISS.

This field resets to an architecturally UNKNOWN value.

Accessing the VSESR_EL2

Accesses to this register use the following encodings:

```
MRS <Xt>, VSESR_EL2
```

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b11	0b100	0b0101	0b0010	0b011
~~0b11~~	~~0b0101~~	~~0b100~~	~~0b011~~	~~0b0010~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
        return NVMem[0x508];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    return VSESR_EL2;
elsif PSTATE.EL == EL3 then
    return VSESR_EL2;
```

```
MSR VSESR_EL2, <Xt>
```

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b11	0b100	0b0101	0b0010	0b011
~~0b11~~	~~0b0101~~	~~0b100~~	~~0b011~~	~~0b0010~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
        NVMem[0x508] = X[t];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    VSESR_EL2 = X[t];
elsif PSTATE.EL == EL3 then
    VSESR_EL2 = X[t];
```

27~~13~~/03~~12~~/2019~~2018~~ 21~~16~~:59~~42~~; e5e4db499bf9867a4b93324c4dbac985d3da93766~~379d01e197f1d40720d32d0f84e419e9187e009~~

(old) htmldiff from- (new)

(old) htmldiff from- (new)

VSTCR_EL2, Virtualization Secure Translation Control Register

The VSTCR_EL2 characteristics are:

Purpose

The control register for stage 2 of the Secure EL1&0 translation regime.

Configuration

This register is present only when ARMv8.4-SecEL2 is implemented. Otherwise, direct accesses to VSTCR_EL2 are UNDEFINED.

This register has no effect if EL2 is not enabled in the current Security state.

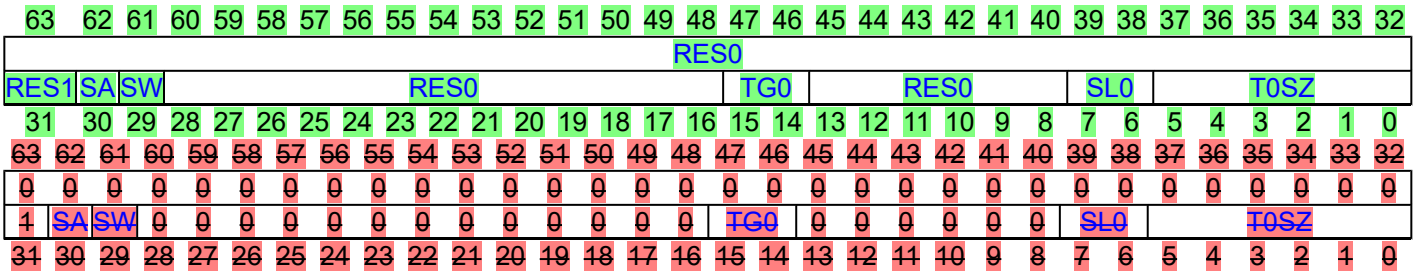
RW fields in this register reset to architecturally UNKNOWN values.

Attributes

VSTCR_EL2 is a 64-bit register.

Field descriptions

The VSTCR_EL2 bit assignments are:



Any of the bits in VSTCR_EL2 are permitted to be cached in a TLB.

Bits [63:32]

Reserved, RES0.

Bit [31]

Reserved, RES1.

SA, bit [30]

Secure stage 2 translation output address space.

SA	Meaning
0b0	All stage 2 translations for the Secure IPA space access the Secure PA space.
0b1	All stage 2 translations for the Secure IPA space access the Non-secure PA space.

When the value of VSTCR_EL2.SW is 1, this bit behaves as 1 for all purposes other than reading back the value of the bit.

This field resets to an architecturally UNKNOWN value.

SW, bit [29]

Secure stage 2 translation address space.

SW	Meaning
0b0	All stage 2 translation table walks for the Secure IPA space are to the Secure PA space.
0b1	All stage 2 translation table walks for the Secure IPA space are to the Non-secure PA space.

This field resets to an architecturally UNKNOWN value.

Bits [28:16]

Reserved, RES0.

TG0, bits [15:14]

Secure stage 2 granule size for VSTTBR_EL2.

TG0	Meaning
0b00	4KB.
0b01	64KB.
0b10	16KB.

Other values are reserved.

If the value is programmed to either a reserved value, or a size that has not been implemented, then for all purposes other than read back from this register, the hardware will treat the field as if it has been programmed to an IMPLEMENTATION DEFINED choice of the sizes that has been implemented.

It is IMPLEMENTATION DEFINED whether the value read back is the value programmed or the value that corresponds to the size chosen.

This field resets to an architecturally UNKNOWN value.

Bits [13:8]

Reserved, RES0.

SL0, bits [7:6]

From Armv8.4, when ARMv8.4-TTST is implemented:

Starting level of the Secure stage 2 translation lookup, controlled by VSTCR_EL2. The meaning of this field depends on the value of VSTCR_EL2.TG0.

SL0	Meaning
0b00	If VSTCR_EL2.TG0 is 0b00 (4KB granule), start at level 2. If VSTCR_EL2.TG0 is 0b10 (16KB granule) or 0b01 (64KB granule), start at level 3.
0b01	If VSTCR_EL2.TG0 is 0b00 (4KB granule), start at level 1. If VSTCR_EL2.TG0 is 0b10 (16KB granule) or 0b01 (64KB granule), start at level 2.
0b10	If VSTCR_EL2.TG0 is 0b00 (4KB granule), start at level 0. If VSTCR_EL2.TG0 is 0b10 (16KB granule) or 0b01 (64KB granule), start at level 1.
0b11	If VSTCR_EL2.TG0 is 0b00 (4KB granule), start at level 3.

All other values are reserved. If this field is programmed to a reserved value, or to a value that is not consistent with the programming of VSTCR_EL2.T0SZ, then a stage 2 level 0 Translation fault is generated.

This field resets to an architecturally UNKNOWN value.

Otherwise:

Starting level of the Secure stage 2 translation lookup, controlled by VSTCR_EL2. The meaning of this field depends on the value of VSTCR_EL2.TG0.

SL0	Meaning
0b00	If VSTCR_EL2.TG0 is 0b00 (4KB granule), start at level 2. If VSTCR_EL2.TG0 is 0b10 (16KB granule) or 0b01 (64KB granule), start at level 3.
0b01	If VSTCR_EL2.TG0 is 0b00 (4KB granule), start at level 1. If VSTCR_EL2.TG0 is 0b10 (16KB granule) or 0b01 (64KB granule), start at level 2.
0b10	If VSTCR_EL2.TG0 is 0b00 (4KB granule), start at level 0. If VSTCR_EL2.TG0 is 0b10 (16KB granule) or 0b01 (64KB granule), start at level 1.

All other values are reserved. If this field is programmed to a reserved value, or to a value that is not consistent with the programming of VSTCR_EL2.T0SZ, then a stage 2 level 0 Translation fault is generated.

This field resets to an architecturally UNKNOWN value.

T0SZ, bits [5:0]

The size offset of the memory region addressed by VSTTBR_EL2. The region size is $2^{(64-\text{VSTCR\_EL2.T0SZ})}$ bytes.

The maximum and minimum possible values for this field depend on the level of translation table and the memory translation granule size, as described in the AArch64 Virtual Memory System Architecture chapter.

If this field is programmed to a value that is not consistent with the programming of SL0, then a stage 2 level 0 Translation fault is generated.

This field resets to an architecturally UNKNOWN value.

Accessing the VSTCR_EL2

Any of the bits in VSTCR_EL2 are permitted to be cached in a TLB.

Accesses to this register use the following encodings:

```
MRS <Xt>, VSTCR_EL2
```

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b11	0b100	0b0010	0b0110	0b010
~~0b11~~	~~0b0010~~	~~0b100~~	~~0b010~~	~~0b0110~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
        return NVMem[0x048];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        if SCR_EL3.NS == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if SCR_EL3.NS == '1' then
        UNDEFINED;
    else
        return VSTCR_EL2;
elsif PSTATE.EL == EL3 then
    return VSTCR_EL2;
```

```
MSR VSTCR_EL2, <Xt>
```

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~

0b11	0b100	0b0010	0b0110	0b010
~~0b11~~	~~0b0010~~	~~0b100~~	~~0b010~~	~~0b0110~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
        NVMem[0x048] = X[t];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        if SCR_EL3.NS == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if SCR_EL3.NS == '1' then
        UNDEFINED;
    else
        VSTCR_EL2 = X[t];
elsif PSTATE.EL == EL3 then
    VSTCR_EL2 = X[t];
```

27~~13~~/03~~12~~/2019~~2018~~ 21~~16~~:59~~42~~; e5e4db499bf9867a4b93324c4dbac985d3da9376~~6379d01e197f1d40720d32d0f84e419e9187e009~~

(old) htmldiff from- (new)

(old) htmldiff from- (new)

VSTTBR_EL2, Virtualization Secure Translation Table Base Register

The VSTTBR_EL2 characteristics are:

Purpose

The base register for stage 2 of the Secure EL1&0 translation regime. Holds the base address of the translation table for the initial lookup for stage 2 of an address translation in the Secure EL1&0 translation regime, and other information for this translation stage.

Configuration

This register is present only when ARMv8.4-SecEL2 is implemented. Otherwise, direct accesses to VSTTBR_EL2 are UNDEFINED.

This register has no effect if EL2 is not enabled in the current Security state.

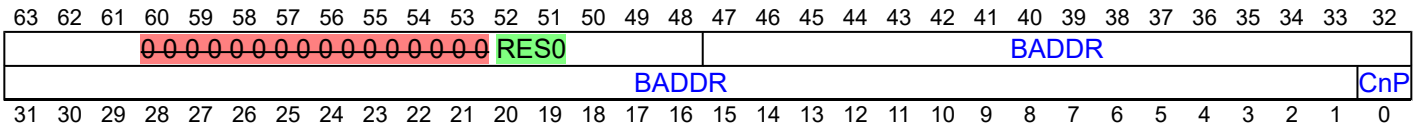
RW fields in this register reset to architecturally UNKNOWN values.

Attributes

VSTTBR_EL2 is a 64-bit register.

Field descriptions

The VSTTBR_EL2 bit assignments are:



Any of the bits in VSTTBR_EL2 are permitted to be cached in a TLB.

Bits [63:48]

Reserved, RES0.

BADDR, bits [47:1]

Translation table base address, A[47:x] or A[51:x].

Note

- Translation table base addresses of 52 bits, A[51:x], are supported only in an implementation that includes ARMv8.2-LPA and is using the 64KB translation granule.
- A translation table must be aligned to the size of the table, except that when using a translation table base address larger than 48 bits the minimum alignment of a table containing fewer than eight entries is 64 bytes.

If the value of VTCR_EL2.PS is `0b110`, then:

- Register bits[47:z] hold bits[47:z] of the stage 1 translation table base address, where z is determined as follows:
  - If x >= 6 then z=x.
  - Otherwise, z=6.
- Register bits[5:2] hold bits[51:48] of the stage 1 translation table base address.
- When z>x register bits[(z-1):x] are RES0, and bits[(z-1):x] of the translation table base address are zero.
- When x>6 register bits[(x-1):6] are RES0.

- Register bit[1] is RES0.
- Bits[5:2] of the stage 1 translation table base address are zero.

Note

When the value of ID_AA64MMFR0_EL1.PARange indicates that the implementation does not support a 52 bit PA size, if a translation table lookup uses this register with the 64KB translation granule when the value of VTCR_EL2.PS is `0b110` and the value of register bits[5:2] is nonzero, an Address size fault is generated.

If the Effective value of VTCR_EL2.PS is not `0b110` then:

- Register bits[47:x] hold bits[47:x] of the stage 1 translation table base address.
- Register bits[(x-1):1] are RES0.
- If the implementation supports 52-bit PAs and IPAs then bits[51:48] of the translation table base addresses used in this stage of translation are `0b0000`.

If any VSTTBR_EL2[47:1] bit that is defined as RES0 has the value 1 when a translation table walk is performed using VSTTBR_EL2, then the translation table base address might be misaligned, with effects that are CONSTRAINED UNPREDICTABLE, and must be one of the following:

- Bits[x-1:0] of the translation table base address are treated as if all the bits are zero. The value read back from the corresponding register bits is either the value written to the register or zero.
- The result of the calculation of an address for a translation table walk using this register can be corrupted in those bits that are nonzero.

The AArch64 Virtual Memory System Architecture chapter describes how x is calculated based on the value of VSTCR_EL2.T0SZ, the stage of translation, and the translation granule size.

This field resets to an architecturally UNKNOWN value.

CnP, bit [0]

Common not Private, for stage 2 of the Secure EL1&0 translation regime. In an implementation that includes ARMv8.2-TTCNP, indicates whether each entry that is pointed to by VSTTBR_EL2 is a member of a common set that can be used by every PE in the Inner Shareable domain for which the value of VSTTBR_EL2.CnP is 1.

CnP	Meaning
`0b0`	The translation table entries pointed to by VSTTBR_EL2 are permitted to differ from the entries for VSTTBR_EL2 for other PEs in the Inner Shareable domain. This is not affected by the value of the current VMID.
`0b1`	The translation table entries pointed to by VSTTBR_EL2 are the same as the translation table entries for every other PE in the Inner Shareable domain for which the value of VSTTBR_EL2.CnP is 1 and the VMID is the same as the current VMID.

Note

If the value of VSTTBR_EL2.CnP bit is 1 on multiple PEs in the same Inner Shareable domain and those VSTTBR_EL2s do not point to the same translation table entries when using the current VMID, then the results of translations using VSTTBR_EL2 are CONSTRAINED UNPREDICTABLE, see CONSTRAINED UNPREDICTABLE behaviors due to caching of control or data values on page K1-6254.

When this register has an architecturally-defined reset value, this field resets to a value that is architecturally UNKNOWN.

This field resets to an architecturally UNKNOWN value.

Accessing the VSTTBR_EL2

Any of the bits in VSTTBR_EL2 are permitted to be cached in a TLB.

Accesses to this register use the following encodings:

```
MRS <Xt>, VSTTBR_EL2
```

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~

0b11	0b100	0b0010	0b0110	0b000
~~0b11~~	~~0b0010~~	~~0b100~~	~~0b000~~	~~0b0110~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
        return NVMem[0x030];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        if SCR_EL3.NS == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if SCR_EL3.NS == '1' then
        UNDEFINED;
    else
        return VSTTBR_EL2;
elsif PSTATE.EL == EL3 then
    return VSTTBR_EL2;
```

```
MSR VSTTBR_EL2, <Xt>
```

op0	**op1**	**CRn**	**CRm**	**op2**
~~**op0**~~	~~**CRn**~~	~~**op1**~~	~~**op2**~~	~~**CRm**~~
0b11	0b100	0b0010	0b0110	0b000
~~0b11~~	~~0b0010~~	~~0b100~~	~~0b000~~	~~0b0110~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
        NVMem[0x030] = X[t];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        if SCR_EL3.NS == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if SCR_EL3.NS == '1' then
        UNDEFINED;
    else
        VSTTBR_EL2 = X[t];
elsif PSTATE.EL == EL3 then
    VSTTBR_EL2 = X[t];
```

27~~13~~/03~~12~~/20192018 21~~16~~:59~~42~~: e5e4db499bf9867a4b93324c4dbac985d3da93766379d01e197f1d40720d32d0f84e419e9187e009

(old) htmldiff from- (new)

VTCR_EL2, Virtualization Translation Control Register

The VTCR_EL2 characteristics are:

Purpose

The control register for stage 2 of the EL1&0 translation regime.

Configuration

AArch64 System register VTCR_EL2 bits [31:0] are architecturally mapped to AArch32 System register VTCR[31:0] .

If EL2 is not implemented, this register is RES0 from EL3.

This register has no effect if EL2 is not enabled in the current Security state.

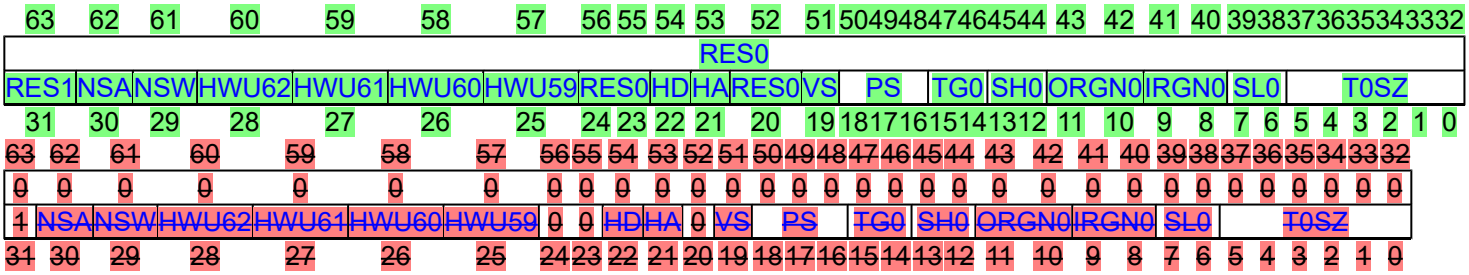
RW fields in this register reset to architecturally UNKNOWN values.

Attributes

VTCR_EL2 is a 64-bit register.

Field descriptions

The VTCR_EL2 bit assignments are:



Any of the bits in VTCR_EL2 are permitted to be cached in a TLB.

Bits [63:32]

Reserved, RES0.

Bit [31]

Reserved, RES1.

NSA, bit [30]

From Armv8.4:

Non-secure stage 2 translation output address space.

NSA	Meaning
0b0	All stage 2 translations for the Non-secure IPA space of the Secure EL1&0 translation regime access the Secure PA space.
0b1	All stage 2 translations for the Non-secure IPA space of the Secure EL1&0 translation regime access the Non-secure PA space.

This bit behaves as 1 for all purposes other than reading back the value of the bit when one of the following is true:

- The PE is executing in Non-secure state.
- The value of VTCR_EL2.NSW is 1.
- The value of VSTCR_EL2.SA is 1.

This field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

NSW, bit [29]

From Armv8.4:

Non-secure stage 2 translation table address space.

NSW	Meaning
0b0	All stage 2 translation table walks for the Non-secure IPA space of the Secure EL1&0 translation regime are to the Secure PA space.
0b1	All stage 2 translation table walks for the Non-secure IPA space of the Secure EL1&0 translation regime are to the Non-secure PA space.

When the PE is executing in Non-secure state, this bit behaves as 1 for all purposes other than reading back the value of the bit.

This field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HWU62, bit [28]

When ARMv8.2-TTPBHA is implemented:

Hardware Use. Indicates IMPLEMENTATION DEFINED hardware use of bit[62] of the stage 2 translation table Block or Page entry.

HWU62	Meaning
0b0	Bit[62] of each stage 2 translation table Block or Page entry cannot be used by hardware for an IMPLEMENTATION DEFINED purpose.
0b1	Bit[62] of each stage 2 translation table Block or Page entry can be used by hardware for an IMPLEMENTATION DEFINED purpose.

This field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HWU61, bit [27]

When ARMv8.2-TTPBHA is implemented:

Hardware Use. Indicates IMPLEMENTATION DEFINED hardware use of bit[61] of the stage 2 translation table Block or Page entry.

HWU61	Meaning
0b0	Bit[61] of each stage 2 translation table Block or Page entry cannot be used by hardware for an IMPLEMENTATION DEFINED purpose.
0b1	Bit[61] of each stage 2 translation table Block or Page entry can be used by hardware for an IMPLEMENTATION DEFINED purpose.

This field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HWU60, bit [26]

When ARMv8.2-TTPBHA is implemented:

Hardware Use. Indicates IMPLEMENTATION DEFINED hardware use of bit[60] of the stage 2 translation table Block or Page entry.

HWU60	Meaning
0b0	Bit[60] of each stage 2 translation table Block or Page entry cannot be used by hardware for an IMPLEMENTATION DEFINED purpose.
0b1	Bit[60] of each stage 2 translation table Block or Page entry can be used by hardware for an IMPLEMENTATION DEFINED purpose.

This field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HWU59, bit [25]

When ARMv8.2-TTPBHA is implemented:

Hardware Use. Indicates IMPLEMENTATION DEFINED hardware use of bit[59] of the stage 2 translation table Block or Page entry.

HWU59	Meaning
0b0	Bit[59] of each stage 2 translation table Block or Page entry cannot be used by hardware for an IMPLEMENTATION DEFINED purpose.
0b1	Bit[59] of each stage 2 translation table Block or Page entry can be used by hardware for an IMPLEMENTATION DEFINED purpose.

This field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [24:23]

Reserved, RES0.

HD, bit [22]

When ARMv8.1-TTHM is implemented:

Hardware management of dirty state in stage 2 translations when EL2 is enabled in the current Security state.

HD	Meaning
0b0	Stage 2 hardware management of dirty state disabled.
0b1	Stage 2 hardware management of dirty state enabled, only if the VTCR_EL2.HA bit is also set to 1.

This field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HA, bit [21]

When ARMv8.1-TTHM is implemented:

Hardware Access flag update in Non-secure and Secure stage 2 translations when EL2 is enabled in the current Security state.

HA	Meaning
0b0	Stage 2 Access flag update disabled.
0b1	Stage 2 Access flag update enabled.

Otherwise:

Reserved, RES0.

Bit [20]

Reserved, RES0.

VS, bit [19]

When ARMv8.1-VMID16 is implemented:

VMID Size.

VS	Meaning
0b0	8 bit - the upper 8 bits of VTTBR_EL2 and VSTTBR_EL2 are ignored by the hardware, and treated as if they are all zeros, for every purpose except when reading back the register.
0b1	16 bit - the upper 8 bits of VTTBR_EL2 and VSTTBR_EL2 are used for allocation and matching in the TLB.

If the implementation only supports an 8-bit VMID, this field is RES0.

This field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

PS, bits [18:16]

Physical address Size for the Second Stage of translation.

PS	Meaning
0b000	32 bits, 4GB.
0b001	36 bits, 64GB.
0b010	40 bits, 1TB.
0b011	42 bits, 4TB.
0b100	44 bits, 16TB.
0b101	48 bits, 256TB.
0b110	52 bits, 4PB.

Other values are reserved.

The reserved values behave in the same way as the 0b101 or 0b110 encoding, but software must not rely on this property as the behavior of the reserved values might change in a future revision of the architecture.

The value 0b110 is permitted only if ARMv8.2-LPA is implemented and the translation granule size is 64KB.

In an implementation that supports 52-bit PAs, if the value of this field is not 0b110 or a value treated as 0b110, then bits[51:48] of every translation table base address for the stage of translation controlled by VTCR_EL2 are 0b0000.

This field resets to an architecturally UNKNOWN value.

TG0, bits [15:14]

Granule size for the VTTBR_EL2.

TG0	Meaning
0b00	4KB.
0b01	64KB.
0b10	16KB.

Other values are reserved.

If the value is programmed to either a reserved value, or a size that has not been implemented, then the hardware will treat the field as if it has been programmed to an IMPLEMENTATION DEFINED choice of the sizes that has been implemented for all purposes other than the value read back from this register.

It is IMPLEMENTATION DEFINED whether the value read back is the value programmed or the value that corresponds to the size chosen.

This field resets to an architecturally UNKNOWN value.

SH0, bits [13:12]

Shareability attribute for memory associated with translation table walks using VTTBR_EL2 or VSTTBR_EL2.

SH0	Meaning
0b00	Non-shareable.
0b10	Outer Shareable.
0b11	Inner Shareable.

Other values are reserved. The effect of programming this field to a Reserved value is that behavior is CONSTRAINED UNPREDICTABLE, as described in 'Reserved values in AArch64 System registers and translation table entries' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section K1.2.2.

This field resets to an architecturally UNKNOWN value.

ORGN0, bits [11:10]

Outer cacheability attribute for memory associated with translation table walks using VTTBR_EL2 or VSTTBR_EL2.

ORGN0	Meaning
0b00	Normal memory, Outer Non-cacheable.
0b01	Normal memory, Outer Write-Back Read-Allocate Write-Allocate Cacheable.
0b10	Normal memory, Outer Write-Through Read-Allocate No Write-Allocate Cacheable.
0b11	Normal memory, Outer Write-Back Read-Allocate No Write-Allocate Cacheable.

This field resets to an architecturally UNKNOWN value.

IRGN0, bits [9:8]

Inner cacheability attribute for memory associated with translation table walks using VTTBR_EL2 or VSTTBR_EL2.

IRGN0	Meaning
0b00	Normal memory, Inner Non-cacheable.
0b01	Normal memory, Inner Write-Back Read-Allocate Write-Allocate Cacheable.
0b10	Normal memory, Inner Write-Through Read-Allocate No Write-Allocate Cacheable.
0b11	Normal memory, Inner Write-Back Read-Allocate No Write-Allocate Cacheable.

This field resets to an architecturally UNKNOWN value.

SL0, bits [7:6]

When ARMv8.4-TTST is implemented:

Starting level of the Secure stage 2 translation lookup, controlled by VTCR_EL2. The meaning of this field depends on the value of VTCR_EL2.TG0.

SL0	Meaning
0b00	If VTCR_EL2.TG0 is 0b00 (4KB granule), start at level 2. If VTCR_EL2.TG0 is 0b10 (16KB granule) or 0b01 (64KB granule), start at level 3.
0b01	If VTCR_EL2.TG0 is 0b00 (4KB granule), start at level 1. If VTCR_EL2.TG0 is 0b10 (16KB granule) or 0b01 (64KB granule), start at level 2.
0b10	If VTCR_EL2.TG0 is 0b00 (4KB granule), start at level 0. If VTCR_EL2.TG0 is 0b10 (16KB granule) or 0b01 (64KB granule), start at level 1.
0b11	If VTCR_EL2.TG0 is 0b00 (4KB granule), start at level 3.

All other values are reserved. If this field is programmed to a reserved value, or to a value that is not consistent with the programming of VTCR_EL2.T0SZ, then a stage 2 level 0 Translation fault is generated.

This field resets to an architecturally UNKNOWN value.

Otherwise:

Starting level of the Secure stage 2 translation lookup, controlled by VTCR_EL2. The meaning of this field depends on the value of VTCR_EL2.TG0.

SL0	Meaning
0b00	If VTCR_EL2.TG0 is 0b00 (4KB granule), start at level 2. If VTCR_EL2.TG0 is 0b10 (16KB granule) or 0b01 (64KB granule), start at level 3.
0b01	If VTCR_EL2.TG0 is 0b00 (4KB granule), start at level 1. If VTCR_EL2.TG0 is 0b10 (16KB granule) or 0b01 (64KB granule), start at level 2.
0b10	If VTCR_EL2.TG0 is 0b00 (4KB granule), start at level 0. If VTCR_EL2.TG0 is 0b10 (16KB granule) or 0b01 (64KB granule), start at level 1.

All other values are reserved. If this field is programmed to a reserved value, or to a value that is not consistent with the programming of VTCR_EL2.T0SZ, then a stage 2 level 0 Translation fault is generated.

This field resets to an architecturally UNKNOWN value.

T0SZ, bits [5:0]

The size offset of the memory region addressed by VTTBR_EL2. The region size is $2^{(64-T0SZ)}$ bytes.

The maximum and minimum possible values for T0SZ depend on the level of translation table and the memory translation granule size, as described in the AArch64 Virtual Memory System Architecture chapter.

If this field is programmed to a value that is not consistent with the programming of SL0 then a stage 2 level 0 Translation fault is generated.

This field resets to an architecturally UNKNOWN value.

Accessing the VTCR_EL2

Any of the bits in VTCR_EL2 are permitted to be cached in a TLB.

Accesses to this register use the following encodings:

```
MRS <Xt>, VTCR_EL2
```

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b11	0b100	0b0010	0b0001	0b010
~~0b11~~	~~0b0010~~	~~0b100~~	~~0b010~~	~~0b0001~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
        return NVMem[0x040];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    return VTCR_EL2;
elsif PSTATE.EL == EL3 then
    return VTCR_EL2;
```

```
MSR VTCR_EL2, <Xt>
```

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b11	0b100	0b0010	0b0001	0b010
~~0b11~~	~~0b0010~~	~~0b100~~	~~0b010~~	~~0b0001~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
        NVMem[0x040] = X[t];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    VTCR_EL2 = X[t];
elsif PSTATE.EL == EL3 then
    VTCR_EL2 = X[t];
```

27~~13~~/03~~12~~/2019~~2018~~ 21~~16~~:59~~42~~; e5e4db499bf9867a4b93324c4dbac985d3da93766~~379d01e197f1d40720d32d0f84e419e9187e009~~

(old) htmldiff from- (new)

(old) htmldiff from- (new)

ZCR_EL1, SVE Control Register for EL1

The ZCR_EL1 characteristics are:

Purpose

The SVE Control Register for EL1 is used to control aspects of SVE visible at Exception levels EL1 and EL0.

Configuration

This register is present only when SVE is implemented. Otherwise, direct accesses to ZCR_EL1 are UNDEFINED.

When HCR_EL2.{E2H, TGE} == {1, 1} and EL2 is enabled in the current Security state, the fields in this register have no effect on execution at EL0

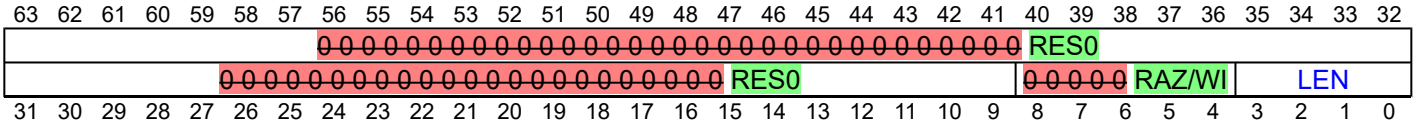
RW fields in this register reset to architecturally UNKNOWN values.

Attributes

ZCR_EL1 is a 64-bit register.

Field descriptions

The ZCR_EL1 bit assignments are:



Bits [63:9]

Reserved, RES0.

Bits [8:4]

Reserved, RAZ/WI.

LEN, bits [3:0]

Constrains the scalable vector register length for EL1 and EL0 to (LEN+1)x128 bits. For all purposes other than returning the result of a direct read of ZCR_EL1 then this field behaves as if it is set to the minimum of the stored value and the constrained length inherited from more privileged Exception levels in the current Security state, rounded down to the nearest implemented vector length.

An indirect read of ZCR_EL1.LEN appears to occur in program order relative to a direct write of the same register, without the need for explicit synchronization.

This field resets to an architecturally UNKNOWN value.

Accessing the ZCR_EL1

When HCR_EL2.E2H is 1, without explicit synchronization, access from EL3 using the mnemonic ZCR_EL1 or ZCR_EL12 are not guaranteed to be ordered with respect to accesses using the other mnemonic.

Accesses to this register use the following encodings:

MRS <Xt>, ZCR_EL1

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b11	0b000	0b0001	0b0010	0b000
~~0b11~~	~~0b0001~~	~~0b000~~	~~0b000~~	~~0b0010~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if CPACR_EL1.ZEN == 'x0' then
        AArch64.SystemAccessTrap(EL1, 0x19);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H != '1' && CPTR_EL2.TZ == '1' then
        AArch64.SystemAccessTrap(EL2, 0x19);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '1' && CPTR_EL2.ZEN == 'x0' then
        AArch64.SystemAccessTrap(EL2, 0x19);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.EZ == '0' then
        AArch64.SystemAccessTrap(EL3, 0x19);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<NV2,NV1,NV> == '111' then
        return NVMem[0x1E0];
    else
        return ZCR_EL1;
elsif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '0' && CPTR_EL2.TZ == '1' then
        AArch64.SystemAccessTrap(EL2, 0x19);
    elsif HCR_EL2.E2H == '1' && CPTR_EL2.ZEN == 'x0' then
        AArch64.SystemAccessTrap(EL2, 0x19);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.EZ == '0' then
        AArch64.SystemAccessTrap(EL3, 0x19);
    elsif HCR_EL2.E2H == '1' then
        return ZCR_EL2;
    else
        return ZCR_EL1;
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.EZ == '0' then
        AArch64.SystemAccessTrap(EL3, 0x19);
    else
        return ZCR_EL1;
```

MSR ZCR_EL1, <Xt>

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b11	0b000	0b0001	0b0010	0b000
~~0b11~~	~~0b0001~~	~~0b000~~	~~0b000~~	~~0b0010~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if CPACR_EL1.ZEN == 'x0' then
        AArch64.SystemAccessTrap(EL1, 0x19);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H != '1' && CPTR_EL2.TZ == '1' then
        AArch64.SystemAccessTrap(EL2, 0x19);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '1' && CPTR_EL2.ZEN == 'x0' then
        AArch64.SystemAccessTrap(EL2, 0x19);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.EZ == '0' then
        AArch64.SystemAccessTrap(EL3, 0x19);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<NV2,NV1,NV> == '111' then
        NVMem[0x1E0] = X[t];
    else
        ZCR_EL1 = X[t];
elsif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '0' && CPTR_EL2.TZ == '1' then
        AArch64.SystemAccessTrap(EL2, 0x19);
    elsif HCR_EL2.E2H == '1' && CPTR_EL2.ZEN == 'x0' then
        AArch64.SystemAccessTrap(EL2, 0x19);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.EZ == '0' then
        AArch64.SystemAccessTrap(EL3, 0x19);
    elsif HCR_EL2.E2H == '1' then
        ZCR_EL2 = X[t];
    else
        ZCR_EL1 = X[t];
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.EZ == '0' then
        AArch64.SystemAccessTrap(EL3, 0x19);
    else
        ZCR_EL1 = X[t];
```

MRS <Xt>, ZCR_EL12

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b11	0b101	0b0001	0b0010	0b000
~~0b11~~	~~0b0001~~	~~0b101~~	~~0b000~~	~~0b0010~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '101' then
        return NVMem[0x1E0];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if EL2Enabled() && HCR_EL2.E2H == '1' then
    if PSTATE.EL == EL2 then
        if HCR_EL2.E2H == '0' && CPTR_EL2.TZ == '1' then
            AArch64.SystemAccessTrap(EL2, 0x19);
        elsif HCR_EL2.E2H == '1' && CPTR_EL2.ZEN == 'x0' then
            AArch64.SystemAccessTrap(EL2, 0x19);
        elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.EZ == '0' then
            AArch64.SystemAccessTrap(EL3, 0x19);
        else
            return ZCR_EL1;
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if EL2Enabled() && HCR_EL2.E2H == '1' then
        if CPTR_EL3.EZ == '0' then
            AArch64.SystemAccessTrap(EL3, 0x19);
        else
            return ZCR_EL1;
else
    else
        UNDEFINED;
```

```
MSR ZCR_EL12, <Xt>
```

op0	op1	CRn	CRm	op2
~~op0~~	~~CRn~~	~~op1~~	~~op2~~	~~CRm~~
0b11	0b101	0b0001	0b0010	0b000
~~0b11~~	~~0b0001~~	~~0b101~~	~~0b000~~	~~0b0010~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '101' then
        NVMem[0x1E0] = X[t];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if EL2Enabled() && HCR_EL2.E2H == '1' then
    if PSTATE.EL == EL2 then
        if HCR_EL2.E2H == '0' && CPTR_EL2.TZ == '1' then
            AArch64.SystemAccessTrap(EL2, 0x19);
        elsif HCR_EL2.E2H == '1' && CPTR_EL2.ZEN == 'x0' then
            AArch64.SystemAccessTrap(EL2, 0x19);
        elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.EZ == '0' then
            AArch64.SystemAccessTrap(EL3, 0x19);
        else
            ZCR_EL1 = X[t];
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if EL2Enabled() && HCR_EL2.E2H == '1' then
        if CPTR_EL3.EZ == '0' then
            AArch64.SystemAccessTrap(EL3, 0x19);
        else
            ZCR_EL1 = X[t];
else
    else
        UNDEFINED;
```

27~~13~~/03~~12~~/2019~~2018~~ 21~~16~~:59~~42~~; e5e4db499bf9867a4b93324c4dbac985d3da93766~~379d01e197f1d40720d32d0f84e419e9187e009~~

(old) htmldiff from- (new)

AArch32 System Registers

ACTLR: Auxiliary Control Register

ACTLR2: Auxiliary Control Register 2

ADFSR: Auxiliary Data Fault Status Register

AIDR: Auxiliary ID Register

AIFSR: Auxiliary Instruction Fault Status Register

AMAIR0: Auxiliary Memory Attribute Indirection Register 0

AMAIR1: Auxiliary Memory Attribute Indirection Register 1

AMCFGR: Activity Monitors Configuration Register

AMCGCR: Activity Monitors Counter Group Configuration Register

AMCNTENCLR0: Activity Monitors Count Enable Clear Register 0

AMCNTENCLR1: Activity Monitors Count Enable Clear Register 1

AMCNTENSET0: Activity Monitors Count Enable Set Register 0

AMCNTENSET1: Activity Monitors Count Enable Set Register 1

AMCR: Activity Monitors Control Register

AMEVCNTR0<n>: Activity Monitors Event Counter Registers 0

AMEVCNTR1<n>: Activity Monitors Event Counter Registers 1

AMEVTYPER0<n>: Activity Monitors Event Type Registers 0

AMEVTYPER1<n>: Activity Monitors Event Type Registers 1

AMUSERENR: Activity Monitors User Enable Register

APSR: Application Program Status Register

CCSIDR: Current Cache Size ID Register

CCSIDR2: Current Cache Size ID Register 2

CLIDR: Cache Level ID Register

CNTFRQ: Counter-timer Frequency register

CNTHCTL: Counter-timer Hyp Control register

CNTHPS_CTL: Counter-timer Secure Physical Timer Control Register (EL2)

CNTHPS_CVAL: Counter-timer Secure Physical Timer CompareValue Register (EL2)

CNTHPS_TVAL: Counter-timer Secure Physical Timer TimerValue Register (EL2)

CNTHP_CTL: Counter-timer Hyp Physical Timer Control register

CNTHP_CVAL: Counter-timer Hyp Physical CompareValue register

CNTHP_TVAL: Counter-timer Hyp Physical Timer TimerValue register

CNTHVS_CTL: Counter-timer Secure Virtual Timer Control Register (EL2)

CNTHVS_CVAL: Counter-timer Secure Virtual Timer CompareValue Register (EL2)

CNTHVS_TVAL: Counter-timer Secure Virtual Timer TimerValue Register (EL2)

CNTHV_CTL: Counter-timer Virtual Timer Control register (EL2)

CNTHV_CVAL: Counter-timer Virtual Timer CompareValue register (EL2)

CNTHV_TVAL: Counter-timer Virtual Timer TimerValue register (EL2)

CNTKCTL: Counter-timer Kernel Control register

CNTPCT: Counter-timer Physical Count register

CNTP_CTL: Counter-timer Physical Timer Control register

CNTP_CVAL: Counter-timer Physical Timer CompareValue register

CNTP_TVAL: Counter-timer Physical Timer TimerValue register

CNTVCT: Counter-timer Virtual Count register

CNTVOFF: Counter-timer Virtual Offset register

CNTV_CTL: Counter-timer Virtual Timer Control register

CNTV_CVAL: Counter-timer Virtual Timer CompareValue register

CNTV_TVAL: Counter-timer Virtual Timer TimerValue register

CONTEXTIDR: Context ID Register

CPACR: Architectural Feature Access Control Register

CPSR: Current Program Status Register

CSSELR: Cache Size Selection Register

CTR: Cache Type Register

DACR: Domain Access Control Register

DBGAUTHSTATUS: Debug Authentication Status register

DBGBCR<n>: Debug Breakpoint Control Registers

DBGBVR<n>: Debug Breakpoint Value Registers

DBGBXVR<n>: Debug Breakpoint Extended Value Registers

DBGCLAIMCLR: Debug Claim Tag Clear register

DBGCLAIMSET: Debug Claim Tag Set register

DBGDCCINT: DCC Interrupt Enable Register

DBGDEVID: Debug Device ID register 0

DBGDEVID1: Debug Device ID register 1

DBGDEVID2: Debug Device ID register 2

DBGDIDR: Debug ID Register

DBGDRAR: Debug ROM Address Register

DBGDSAR: Debug Self Address Register

DBGDSCRext: Debug Status and Control Register, External View

DBGDSCRint: Debug Status and Control Register, Internal View

DBGDTRRXext: Debug OS Lock Data Transfer Register, Receive, External View

DBGDTRRXint: Debug Data Transfer Register, Receive

DBGDTRTXext: Debug OS Lock Data Transfer Register, Transmit

DBGDTRTXint: Debug Data Transfer Register, Transmit

DBGOSDLR: Debug OS Double Lock Register

DBGOSECCR: Debug OS Lock Exception Catch Control Register

DBGOSLAR: Debug OS Lock Access Register

DBGOSLSR: Debug OS Lock Status Register

DBGPRCR: Debug Power Control Register

DBGVCR: Debug Vector Catch Register

DBGWCR<n>: Debug Watchpoint Control Registers

DBGWFAR: Debug Watchpoint Fault Address Register

DBGWVR<n>: Debug Watchpoint Value Registers

DFAR: Data Fault Address Register

DFSR: Data Fault Status Register

DISR: Deferred Interrupt Status Register

DLR: Debug Link Register

DSPSR: Debug Saved Program Status Register

ELR_hyp: Exception Link Register (Hyp mode)

ERRIDR: Error Record ID Register

ERRSELR: Error Record Select Register

ERXADDR: Selected Error Record Address Register

ERXADDR2: Selected Error Record Address Register 2

ERXCTLR: Selected Error Record Control Register

ERXCTLR2: Selected Error Record Control Register 2

ERXFR: Selected Error Record Feature Register

ERXFR2: Selected Error Record Feature Register 2

ERXMISC0: Selected Error Record Miscellaneous Register 0

ERXMISC1: Selected Error Record Miscellaneous Register 1

ERXMISC2: Selected Error Record Miscellaneous Register 2

ERXMISC3: Selected Error Record Miscellaneous Register 3

ERXMISC4: Selected Error Record Miscellaneous Register 4

ERXMISC5: Selected Error Record Miscellaneous Register 5

ERXMISC6: Selected Error Record Miscellaneous Register 6

ERXMISC7: Selected Error Record Miscellaneous Register 7

ERXSTATUS: Selected Error Record Primary Status Register

FCSEIDR: FCSE Process ID register

FPEXC: Floating-Point Exception Control register

FPSCR: Floating-Point Status and Control Register

FPSID: Floating-Point System ID register

HACR: Hyp Auxiliary Configuration Register

HACTLR: Hyp Auxiliary Control Register

HACTLR2: Hyp Auxiliary Control Register 2

HADFSR: Hyp Auxiliary Data Fault Status Register

HAIFSR: Hyp Auxiliary Instruction Fault Status Register

HAMAIR0: Hyp Auxiliary Memory Attribute Indirection Register 0

HAMAIR1: Hyp Auxiliary Memory Attribute Indirection Register 1

HCPTR: Hyp Architectural Feature Trap Register

HCR: Hyp Configuration Register

HCR2: Hyp Configuration Register 2

HDCR: Hyp Debug Control Register

HDFAR: Hyp Data Fault Address Register

HIFAR: Hyp Instruction Fault Address Register

HMAIR0: Hyp Memory Attribute Indirection Register 0

HMAIR1: Hyp Memory Attribute Indirection Register 1

HPFAR: Hyp IPA Fault Address Register

HRMR: Hyp Reset Management Register

HSCTLR: Hyp System Control Register

HSR: Hyp Syndrome Register

HSTR: Hyp System Trap Register

HTCR: Hyp Translation Control Register

HTPIDR: Hyp Software Thread ID Register

HTRFCR: Hyp Trace Filter Control Register

HTTBR: Hyp Translation Table Base Register

HVBAR: Hyp Vector Base Address Register

ICC_AP0R<n>: Interrupt Controller Active Priorities Group 0 Registers

ICC_AP1R<n>: Interrupt Controller Active Priorities Group 1 Registers

ICC_ASGI1R: Interrupt Controller Alias Software Generated Interrupt Group 1 Register

ICC_BPR0: Interrupt Controller Binary Point Register 0

ICC_BPR1: Interrupt Controller Binary Point Register 1

ICC_CTLR: Interrupt Controller Control Register

ICC_DIR: Interrupt Controller Deactivate Interrupt Register

ICC_EOIR0: Interrupt Controller End Of Interrupt Register 0

ICC_EOIR1: Interrupt Controller End Of Interrupt Register 1

ICC_HPPIR0: Interrupt Controller Highest Priority Pending Interrupt Register 0

ICC_HPPIR1: Interrupt Controller Highest Priority Pending Interrupt Register 1

ICC_HSRE: Interrupt Controller Hyp System Register Enable register

ICC_IAR0: Interrupt Controller Interrupt Acknowledge Register 0

ICC_IAR1: Interrupt Controller Interrupt Acknowledge Register 1

ICC_IGRPEN0: Interrupt Controller Interrupt Group 0 Enable register

ICC_IGRPEN1: Interrupt Controller Interrupt Group 1 Enable register

ICC_MCTLR: Interrupt Controller Monitor Control Register

ICC_MGRPEN1: Interrupt Controller Monitor Interrupt Group 1 Enable register

ICC_MSRE: Interrupt Controller Monitor System Register Enable register

ICC_PMR: Interrupt Controller Interrupt Priority Mask Register

ICC_RPR: Interrupt Controller Running Priority Register

ICC_SGI0R: Interrupt Controller Software Generated Interrupt Group 0 Register

ICC_SGI1R: Interrupt Controller Software Generated Interrupt Group 1 Register

ICC_SRE: Interrupt Controller System Register Enable register

ICH_AP0R<n>: Interrupt Controller Hyp Active Priorities Group 0 Registers

ICH_AP1R<n>: Interrupt Controller Hyp Active Priorities Group 1 Registers

ICH_EISR: Interrupt Controller End of Interrupt Status Register

ICH_ELRSR: Interrupt Controller Empty List Register Status Register

ICH_HCR: Interrupt Controller Hyp Control Register

ICH_LR<n>: Interrupt Controller List Registers

ICH_LRC<n>: Interrupt Controller List Registers

ICH_MISR: Interrupt Controller Maintenance Interrupt State Register

ICH_VMCR: Interrupt Controller Virtual Machine Control Register

ICH_VTR: Interrupt Controller VGIC Type Register

ICV_AP0R<n>: Interrupt Controller Virtual Active Priorities Group 0 Registers

ICV_AP1R<n>: Interrupt Controller Virtual Active Priorities Group 1 Registers

ICV_BPR0: Interrupt Controller Virtual Binary Point Register 0

ICV_BPR1: Interrupt Controller Virtual Binary Point Register 1

ICV_CTLR: Interrupt Controller Virtual Control Register

ICV_DIR: Interrupt Controller Deactivate Virtual Interrupt Register

ICV_EOIR0: Interrupt Controller Virtual End Of Interrupt Register 0

ICV_EOIR1: Interrupt Controller Virtual End Of Interrupt Register 1

ICV_HPPIR0: Interrupt Controller Virtual Highest Priority Pending Interrupt Register 0

ICV_HPPIR1: Interrupt Controller Virtual Highest Priority Pending Interrupt Register 1

ICV_IAR0: Interrupt Controller Virtual Interrupt Acknowledge Register 0

ICV_IAR1: Interrupt Controller Virtual Interrupt Acknowledge Register 1

ICV_IGRPEN0: Interrupt Controller Virtual Interrupt Group 0 Enable register

ICV_IGRPEN1: Interrupt Controller Virtual Interrupt Group 1 Enable register

ICV_PMR: Interrupt Controller Virtual Interrupt Priority Mask Register

ICV_RPR: Interrupt Controller Virtual Running Priority Register

ID_AFR0: Auxiliary Feature Register 0

ID_DFR0: Debug Feature Register 0

ID_ISAR0: Instruction Set Attribute Register 0

ID_ISAR1: Instruction Set Attribute Register 1

ID_ISAR2: Instruction Set Attribute Register 2

ID_ISAR3: Instruction Set Attribute Register 3

ID_ISAR4: Instruction Set Attribute Register 4

ID_ISAR5: Instruction Set Attribute Register 5

ID_ISAR6: Instruction Set Attribute Register 6

ID_MMFR0: Memory Model Feature Register 0

ID_MMFR1: Memory Model Feature Register 1

ID_MMFR2: Memory Model Feature Register 2

ID_MMFR3: Memory Model Feature Register 3

ID_MMFR4: Memory Model Feature Register 4

ID_PFR0: Processor Feature Register 0

ID_PFR1: Processor Feature Register 1

ID_PFR2: Processor Feature Register 2

IFAR: Instruction Fault Address Register

IFSR: Instruction Fault Status Register

ISR: Interrupt Status Register

JIDR: Jazelle ID Register

JMCR: Jazelle Main Configuration Register

JOSCR: Jazelle OS Control Register

MAIR0: Memory Attribute Indirection Register 0

MAIR1: Memory Attribute Indirection Register 1

MIDR: Main ID Register

MPIDR: Multiprocessor Affinity Register

MVBAR: Monitor Vector Base Address Register

MVFR0: Media and VFP Feature Register 0

MVFR1: Media and VFP Feature Register 1

MVFR2: Media and VFP Feature Register 2

NMRR: Normal Memory Remap Register

NSACR: Non-Secure Access Control Register

PAR: Physical Address Register

PMCCFILTR: Performance Monitors Cycle Count Filter Register

PMCCNTR: Performance Monitors Cycle Count Register

PMCEID0: Performance Monitors Common Event Identification register 0

PMCEID1: Performance Monitors Common Event Identification register 1

PMCEID2: Performance Monitors Common Event Identification register 2

PMCEID3: Performance Monitors Common Event Identification register 3

PMCNTENCLR: Performance Monitors Count Enable Clear register

PMCNTENSET: Performance Monitors Count Enable Set register

PMCR: Performance Monitors Control Register

PMEVCNTR<n>: Performance Monitors Event Count Registers

PMEVTYPER<n>: Performance Monitors Event Type Registers

PMINTENCLR: Performance Monitors Interrupt Enable Clear register

PMINTENSET: Performance Monitors Interrupt Enable Set register

PMMIR: Performance Monitors Machine Identification Register

PMOVSR: Performance Monitors Overflow Flag Status Register

PMOVSSET: Performance Monitors Overflow Flag Status Set register

PMSELR: Performance Monitors Event Counter Selection Register

PMSWINC: Performance Monitors Software Increment register

PMUSERENR: Performance Monitors User Enable Register

PMXEVCNTR: Performance Monitors Selected Event Count Register

PMXEVTYPER: Performance Monitors Selected Event Type Register

PRRR: Primary Region Remap Register

REVIDR: Revision ID Register

RMR: Reset Management Register

RVBAR: Reset Vector Base Address Register

SCR: Secure Configuration Register

SCTLR: System Control Register

SDCR: Secure Debug Control Register

SDER: Secure Debug Enable Register

SPSR: Saved Program Status Register

SPSR_abt: Saved Program Status Register (Abort mode)

SPSR_fiq: Saved Program Status Register (FIQ mode)

SPSR_hyp: Saved Program Status Register (Hyp mode)

SPSR_irq: Saved Program Status Register (IRQ mode)

SPSR_mon: Saved Program Status Register (Monitor mode)

SPSR_svc: Saved Program Status Register (Supervisor mode)

SPSR_und: Saved Program Status Register (Undefined mode)

TCMTR: TCM Type Register

TLBTR: TLB Type Register

TPIDRPRW: PL1 Software Thread ID Register

TPIDRURO: PL0 Read-Only Software Thread ID Register

TPIDRURW: PL0 Read/Write Software Thread ID Register

TRFCR: Trace Filter Control Register

TTBCR: Translation Table Base Control Register

TTBCR2: Translation Table Base Control Register 2

TTBR0: Translation Table Base Register 0

TTBR1: Translation Table Base Register 1

VBAR: Vector Base Address Register

VDFSR: Virtual SError Exception Syndrome Register

VDISR: Virtual Deferred Interrupt Status Register

VMPIDR: Virtualization Multiprocessor ID Register

VPIDR: Virtualization Processor ID Register

VTCR: Virtualization Translation Control Register

VTTBR: Virtualization Translation Table Base Register

27~~13~~/03~~12~~/2019~~2018~~ 21~~16~~:59~~43~~

(old) htmldiff from- (new)

(old) htmldiff from- (new)

AArch32 System Instructions

ATS12NSOPR: Address Translate Stages 1 and 2 Non-secure Only PL1 Read

ATS12NSOPW: Address Translate Stages 1 and 2 Non-secure Only PL1 Write

ATS12NSOUR: Address Translate Stages 1 and 2 Non-secure Only Unprivileged Read

ATS12NSOUW: Address Translate Stages 1 and 2 Non-secure Only Unprivileged Write

ATS1CPR: Address Translate Stage 1 Current state PL1 Read

ATS1CPRP: Address Translate Stage 1 Current state PL1 Read PAN

ATS1CPW: Address Translate Stage 1 Current state PL1 Write

ATS1CPWP: Address Translate Stage 1 Current state PL1 Write PAN

ATS1CUR: Address Translate Stage 1 Current state Unprivileged Read

ATS1CUW: Address Translate Stage 1 Current state Unprivileged Write

ATS1HR: Address Translate Stage 1 Hyp mode Read

ATS1HW: Address Translate Stage 1 Hyp mode Write

BPIALL: Branch Predictor Invalidate All

BPIALLIS: Branch Predictor Invalidate All, Inner Shareable

BPIMVA: Branch Predictor Invalidate by VA

CFPRCTX: Control Flow Prediction Restriction by Context

CP15DMB: Data Memory Barrier System instruction

CP15DSB: Data Synchronization Barrier System instruction

CP15ISB: Instruction Synchronization Barrier System instruction

CPPRCTX: Cache Prefetch Prediction Restriction by Context

DCCIMVAC: Data Cache line Clean and Invalidate by VA to PoC

DCCISW: Data Cache line Clean and Invalidate by Set/Way

DCCMVAC: Data Cache line Clean by VA to PoC

DCCMVAU: Data Cache line Clean by VA to PoU

DCCSW: Data Cache line Clean by Set/Way

DCIMVAC: Data Cache line Invalidate by VA to PoC

DCISW: Data Cache line Invalidate by Set/Way

DTLBIALL: Data TLB Invalidate All

DTLBIASID: Data TLB Invalidate by ASID match

DTLBIMVA: Data TLB Invalidate by VA

DVPRCTX: Data Value Prediction Restriction by Context

ICIALLU: Instruction Cache Invalidate All to PoU

ICIALLUIS: Instruction Cache Invalidate All to PoU, Inner Shareable

ICIMVAU: Instruction Cache line Invalidate by VA to PoU

ITLBIALL: Instruction TLB Invalidate All

ITLBIASID: Instruction TLB Invalidate by ASID match

ITLBIMVA: Instruction TLB Invalidate by VA

TLBIALL: TLB Invalidate All

TLBIALLH: TLB Invalidate All, Hyp mode

TLBIALLHIS: TLB Invalidate All, Hyp mode, Inner Shareable

TLBIALLIS: TLB Invalidate All, Inner Shareable

TLBIALLNSNH: TLB Invalidate All, Non-Secure Non-Hyp

TLBIALLNSNHIS: TLB Invalidate All, Non-Secure Non-Hyp, Inner Shareable

TLBIASID: TLB Invalidate by ASID match

TLBIASIDIS: TLB Invalidate by ASID match, Inner Shareable

TLBIIPAS2: TLB Invalidate by Intermediate Physical Address, Stage 2

TLBIIPAS2IS: TLB Invalidate by Intermediate Physical Address, Stage 2, Inner Shareable

TLBIIPAS2L: TLB Invalidate by Intermediate Physical Address, Stage 2, Last level

TLBIIPAS2LIS: TLB Invalidate by Intermediate Physical Address, Stage 2, Last level, Inner Shareable

TLBIMVA: TLB Invalidate by VA

TLBIMVAA: TLB Invalidate by VA, All ASID

TLBIMVAAIS: TLB Invalidate by VA, All ASID, Inner Shareable

TLBIMVAAL: TLB Invalidate by VA, All ASID, Last level

TLBIMVAALIS: TLB Invalidate by VA, All ASID, Last level, Inner Shareable

TLBIMVAH: TLB Invalidate by VA, Hyp mode

TLBIMVAHIS: TLB Invalidate by VA, Hyp mode, Inner Shareable

TLBIMVAIS: TLB Invalidate by VA, Inner Shareable

TLBIMVAL: TLB Invalidate by VA, Last level

TLBIMVALH: TLB Invalidate by VA, Last level, Hyp mode

TLBIMVALHIS: TLB Invalidate by VA, Last level, Hyp mode, Inner Shareable

TLBIMVALIS: TLB Invalidate by VA, Last level, Inner Shareable

2713/0312/20192018 2116:5943

(old) htmldiff from- (new)

AMAIR0, Auxiliary Memory Attribute Indirection Register 0

The AMAIR0 characteristics are:

Purpose

When using the Long-descriptor format translation tables for stage 1 translations, provides IMPLEMENTATION DEFINED memory attributes for the memory regions specified by MAIR0.

Configuration

AArch32 System register AMAIR0 bits [31:0] are architecturally mapped to AArch64 System register AMAIR_EL1[31:0] .

~~When EL3 is using AArch32, write access to AMAIR0(S) is disabled when the CP15SDISABLE signal is asserted HIGH.~~

RW fields in this register reset to architecturally UNKNOWN values.

Attributes

AMAIR0 is a 32-bit register.

Field descriptions

The AMAIR0 bit assignments are:

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
IMPLEMENTATION DEFINED																															

This register is RES0 in the following cases:

- When an implementation does not provide any IMPLEMENTATION DEFINED memory attributes.
- When the Long-descriptor translation table format is not used.

If EL3 is implemented and is using AArch32:

- AMAIR0(S) gives the value for memory accesses from Secure state.
- AMAIR0(NS) gives the value for memory accesses from Non-secure states other than Hyp mode.

Any IMPLEMENTATION DEFINED memory attributes are additional qualifiers for the memory locations and must not change the architected behavior specified by MAIR0 and MAIR1.

In a typical implementation, AMAIR0 and AMAIR1 split into eight one-byte fields, corresponding to the MAIRn.Attr<n> fields, but the architecture does not require them to do so.

IMPLEMENTATION DEFINED, bits [31:0]

IMPLEMENTATION DEFINED.

This field resets to an architecturally UNKNOWN value.

Accessing the AMAIR0

Accesses to this register use the following encodings:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
~~opc1~~	~~opc2~~	~~CRn~~	~~coproc~~	~~CRm~~
0b1111	0b000	0b1010	0b0011	0b000
~~0b000~~	~~0b000~~	~~0b1010~~	~~0b1111~~	~~0b0011~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T10 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T10 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TRVM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TRVM == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) then
        if SCR.NS == '0' then
            return AMAIR0_S;
        else
            return AMAIR0_NS;
    else
        return AMAIR0;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && ELUsingAArch32(EL3) then
        return AMAIR0_NS;
    else
        return AMAIR0;
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        return AMAIR0_S;
    else
        return AMAIR0_NS;
```

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
~~opc1~~	~~opc2~~	~~CRn~~	~~coproc~~	~~CRm~~
0b1111	0b000	0b1010	0b0011	0b000
~~0b000~~	~~0b000~~	~~0b1010~~	~~0b1111~~	~~0b0011~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T10 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T10 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) && SCR.NS == '0' && CP15SDISABLE == HIGH then
        UNDEFINED;
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) && SCR.NS == '0' && CP15SDISABLE2 == HIGH then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TVM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TVM == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) then
        if SCR.NS == '0' then
            AMAIR0_S = R[t];
        else
            AMAIR0_NS = R[t];
    else
        AMAIR0 = R[t];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && ELUsingAArch32(EL3) then
        AMAIR0_NS = R[t];
    else
        AMAIR0 = R[t];
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' && CP15SDISABLE == HIGH then
        UNDEFINED;
    elsif SCR.NS == '0' && CP15SDISABLE2 == HIGH then
        UNDEFINED;
    else
        if SCR.NS == '0' then
            AMAIR0_S = R[t];
        else
            AMAIR0_NS = R[t];
```

2713/0312/20192018 2116:5942; e5e4db499bf9867a4b93324c4dbac985d3da93766379d01e197f1d40720d32d0f84e419e9187e009

(old) htmldiff from- (new)

AMAIR1, Auxiliary Memory Attribute Indirection Register 1

The AMAIR1 characteristics are:

Purpose

When using the Long-descriptor format translation tables for stage 1 translations, provides IMPLEMENTATION DEFINED memory attributes for the memory regions specified by MAIR1.

Configuration

AArch32 System register AMAIR1 bits [31:0] are architecturally mapped to AArch64 System register AMAIR_EL1[63:32] .

When EL3 is using AArch32, write access to AMAIR1(S) is disabled when the CP15SDISABLE signal is asserted HIGH.

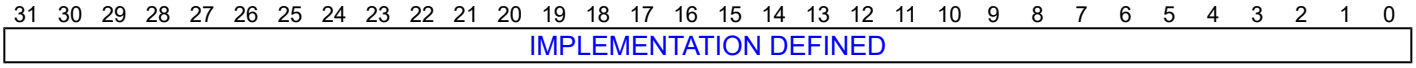
RW fields in this register reset to architecturally UNKNOWN values.

Attributes

AMAIR1 is a 32-bit register.

Field descriptions

The AMAIR1 bit assignments are:



This register is RES0 in the following cases:

- When an implementation does not provide any IMPLEMENTATION DEFINED memory attributes.
- When the Long-descriptor translation table format is not used.

If EL3 is implemented and is using AArch32:

- AMAIR1(S) gives the value for memory accesses from Secure state.
- AMAIR1(NS) gives the value for memory accesses from Non-secure states other than Hyp mode.

Any IMPLEMENTATION DEFINED memory attributes are additional qualifiers for the memory locations and must not change the architected behavior specified by MAIR0 and MAIR1.

In a typical implementation, AMAIR0 and AMAIR1 split into eight one-byte fields, corresponding to the MAIRn.Attr<n> fields, but the architecture does not require them to do so.

IMPLEMENTATION DEFINED, bits [31:0]

IMPLEMENTATION DEFINED.

This field resets to an architecturally UNKNOWN value.

Accessing the AMAIR1

Accesses to this register use the following encodings:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
~~opc1~~	~~opc2~~	~~CRn~~	~~coproc~~	~~CRm~~
0b1111	0b000	0b1010	0b0011	0b001
~~0b000~~	~~0b001~~	~~0b1010~~	~~0b1111~~	~~0b0011~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T10 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T10 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TRVM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TRVM == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) then
        if SCR.NS == '0' then
            return AMAIR1_S;
        else
            return AMAIR1_NS;
    else
        return AMAIR1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && ELUsingAArch32(EL3) then
        return AMAIR1_NS;
    else
        return AMAIR1;
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        return AMAIR1_S;
    else
        return AMAIR1_NS;
```

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
~~opc1~~	~~opc2~~	~~CRn~~	~~coproc~~	~~CRm~~
0b1111	0b000	0b1010	0b0011	0b001
~~0b000~~	~~0b001~~	~~0b1010~~	~~0b1111~~	~~0b0011~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T10 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T10 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) && SCR.NS == '0' && CP15SDISABLE == HIGH then
        UNDEFINED;
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) && SCR.NS == '0' && CP15SDISABLE2 == HIGH then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TVM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TVM == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) then
        if SCR.NS == '0' then
            AMAIR1_S = R[t];
        else
            AMAIR1_NS = R[t];
    else
        AMAIR1 = R[t];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && ELUsingAArch32(EL3) then
        AMAIR1_NS = R[t];
    else
        AMAIR1 = R[t];
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' && CP15SDISABLE == HIGH then
        UNDEFINED;
    elsif SCR.NS == '0' && CP15SDISABLE2 == HIGH then
        UNDEFINED;
    else
        if SCR.NS == '0' then
            AMAIR1_S = R[t];
        else
            AMAIR1_NS = R[t];
```

2713/0312/20192018 2116:5942; e5e4db499bf9867a4b93324c4dbac985d3da93766379d01e197f1d40720d32d0f84e419e9187e009

(old) htmldiff from- (new)

AMEVCNTR0<n>, Activity Monitors Event Counter Registers 0, n = 0 - 15

The AMEVCNTR0<n> characteristics are:

Purpose

Provides access to the architected activity monitor event counters.

Configuration

AArch32 System register AMEVCNTR0<n> bits [63:0] are architecturally mapped to AArch64 System register AMEVCNTR0<n>_EL0[63:0] .

AArch32 System register AMEVCNTR0<n> bits [63:0] are architecturally mapped to External register AMEVCNTR0<n>[63:0] .

This register is present only when AMUv1 is implemented. Otherwise, direct accesses to AMEVCNTR0<n> are UNDEFINED.

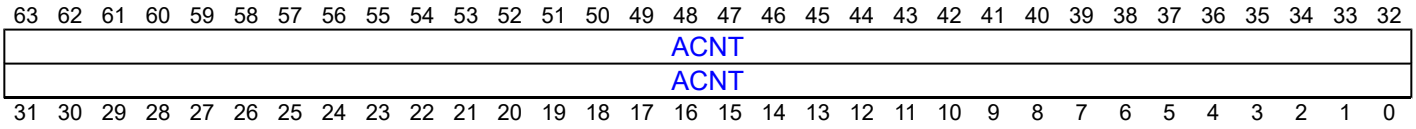
Some or all RW fields of this register have defined reset values. These apply only if the PE resets into an Exception level that is using AArch32. Otherwise, RW fields in this register reset to architecturally UNKNOWN values.

Attributes

AMEVCNTR0<n> is a 64-bit register.

Field descriptions

The AMEVCNTR0<n> bit assignments are:



ACNT, bits [63:0]

Architected activity monitor event counter n.

Value of architected activity monitor event counter n, where n is the number of this register and is a number from 0 to 15.

If the counter is enabled, writes to this register have UNPREDICTABLE results.

On a Cold reset, this field resets to 0.

Accessing the AMEVCNTR0<n>

If <n> is greater than or equal to the number of architected activity monitor event counters, reads and writes of AMEVCNTR0<n> are CONSTRAINED UNPREDICTABLE, and the following behaviors are permitted:

- Accesses to the register are UNDEFINED.
- Accesses to the register behave as RAZ/WI.
- Accesses to the register execute as a NOP.

Note

AMCGCR.CG0NC identifies the number of architected activity monitor event counters.

Accesses to this register use the following encodings:

```
MRRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <Rt2>, <CRm>
```

coproc	CRm	opc1
~~opc1~~	~~coproc~~	~~CRm~~
0b1111	0b000[n:3]	0b0[n:2:0]
~~0b[n:2:0]~~	~~0b1111~~	~~0b000[n:3]~~

```
if CRm == 0 then
    if PSTATE.EL == EL0 then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> != '11' && HSTR_EL2.T0 == '1'
then
            AArch64.AArch32SystemAccessTrap(EL2, 0x04);
        elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T0 == '1' then
            AArch32.TakeHypTrapException(0x04);
        elsif !ELUsingAArch32(EL1) && AMUSERENR_EL0.EN == '0' then
            if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
                AArch64.AArch32SystemAccessTrap(EL2, 0x04);
            else
                AArch64.AArch32SystemAccessTrap(EL1, 0x04);
        elsif EL2Enabled() && !ELUsingAArch32(EL2) && CPTR_EL2.TAM == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x04);
        elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TAM == '1' then
            AArch64.AArch32SystemAccessTrap(EL3, 0x04);
        else
            return AMEVCNTR0[UInt(CRm<0>:opc1<2:0>)];
    elsif PSTATE.EL == EL1 then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T0 == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x04);
        elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T0 == '1' then
            AArch32.TakeHypTrapException(0x04);
        elsif EL2Enabled() && !ELUsingAArch32(EL2) && CPTR_EL2.TAM == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x04);
        elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TAM == '1' then
            AArch64.AArch32SystemAccessTrap(EL3, 0x04);
        else
            return AMEVCNTR0[UInt(CRm<0>:opc1<2:0>)];
    elsif PSTATE.EL == EL2 then
        if HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TAM == '1' then
            AArch64.AArch32SystemAccessTrap(EL3, 0x04);
        else
            return AMEVCNTR0[UInt(CRm<0>:opc1<2:0>)];
    elsif PSTATE.EL == EL3 then
        return AMEVCNTR0[UInt(CRm<0>:opc1<2:0>)];
else
    UNDEFINED;
```

```
MCRR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <Rt2>, <CRm>
```

coproc	CRm	opc1
~~opc1~~	~~coproc~~	~~CRm~~
0b1111	0b000[n:3]	0b0[n:2:0]
~~0b[n:2:0]~~	~~0b1111~~	~~0b000[n:3]~~

```
if CRm == 0 then
    if PSTATE.EL == EL1 && EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T0 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    elsif PSTATE.EL == EL1 && EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T0 == '1' then
        AArch32.TakeHypTrapException(0x04);
    elsif IsHighestEL(PSTATE.EL) then
        AMEVCNTR0[UInt(CRm<0>:opc1<2:0>)] = R[t2]:R[t];
    else
        UNDEFINED;
else
    UNDEFINED;
```

27~~13~~/03~~12~~/2019~~2018~~ 21~~16~~:59~~42~~; e5e4db499bf9867a4b93324c4dbac985d3da9376~~6379d01e197f1d40720d32d0f84e419e9187e009~~

(old) htmldiff from- (new)

(old) htmldiff from- (new)

AMEVCNTR1<n>, Activity Monitors Event Counter Registers 1, n = 0 - 15

The AMEVCNTR1<n> characteristics are:

Purpose

Provides access to the auxiliary activity monitor event counters.

Configuration

AArch32 System register AMEVCNTR1<n> bits [63:0] are architecturally mapped to AArch64 System register AMEVCNTR1<n>_EL0[63:0] .

AArch32 System register AMEVCNTR1<n> bits [63:0] are architecturally mapped to External register AMEVCNTR1<n>[63:0] .

This register is present only when AMUv1 is implemented. Otherwise, direct accesses to AMEVCNTR1<n> are UNDEFINED.

Some or all RW fields of this register have defined reset values. These apply only if the PE resets into an Exception level that is using AArch32. Otherwise, RW fields in this register reset to architecturally UNKNOWN values.

Attributes

AMEVCNTR1<n> is a 64-bit register.

Field descriptions

The AMEVCNTR1<n> bit assignments are:

63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
ACNT																															
ACNT																															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

ACNT, bits [63:0]

Auxiliary activity monitor event counter n.

Value of auxiliary activity monitor event counter n, where n is the number of this register and is a number from 0 to 15.

If the counter is enabled, writes to this register have UNPREDICTABLE results.

On a Cold reset, this field resets to 0.

Accessing the AMEVCNTR1<n>

If <n> is greater than or equal to the number of auxiliary activity monitor event counters, reads and writes of AMEVCNTR1<n> are CONSTRAINED UNPREDICTABLE, and the following behaviors are permitted:

- Accesses to the register are UNDEFINED.
- Accesses to the register behave as RAZ/WI.
- Accesses to the register execute as a NOP.

Note

AMCGCR.CG1NC identifies the number of auxiliary activity monitor event counters.

Accesses to this register use the following encodings:

```
MRRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <Rt2>, <CRm>
```

coproc	CRm	opc1
~~opc1~~	~~coproc~~	~~CRm~~
0b1111	0b010[n:3]	0b0[n:2:0]
~~0b[n:2:0]~~	~~0b1111~~	~~0b010[n:3]~~

```
if CRm == 100 then
    if PSTATE.EL == EL0 then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> != '11' && HSTR_EL2.T4 == '1'
then
            AArch64.AArch32SystemAccessTrap(EL2, 0x04);
        elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T4 == '1' then
            AArch32.TakeHypTrapException(0x04);
        elsif !ELUsingAArch32(EL1) && AMUSERENR_EL0.EN == '0' then
            if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
                AArch64.AArch32SystemAccessTrap(EL2, 0x04);
            else
                AArch64.AArch32SystemAccessTrap(EL1, 0x04);
        elsif EL2Enabled() && !ELUsingAArch32(EL2) && CPTR_EL2.TAM == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x04);
        elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TAM == '1' then
            AArch64.AArch32SystemAccessTrap(EL3, 0x04);
        else
            return AMEVCNTR1[UInt(CRm<0>:opc1<2:0>)];
    elsif PSTATE.EL == EL1 then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T4 == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x04);
        elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T4 == '1' then
            AArch32.TakeHypTrapException(0x04);
        elsif EL2Enabled() && !ELUsingAArch32(EL2) && CPTR_EL2.TAM == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x04);
        elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TAM == '1' then
            AArch64.AArch32SystemAccessTrap(EL3, 0x04);
        else
            return AMEVCNTR1[UInt(CRm<0>:opc1<2:0>)];
    elsif PSTATE.EL == EL2 then
        if HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TAM == '1' then
            AArch64.AArch32SystemAccessTrap(EL3, 0x04);
        else
            return AMEVCNTR1[UInt(CRm<0>:opc1<2:0>)];
    elsif PSTATE.EL == EL3 then
        return AMEVCNTR1[UInt(CRm<0>:opc1<2:0>)];
elsif CRm == 101 then
    if PSTATE.EL == EL0 then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> != '11' && HSTR_EL2.T5 == '1'
then
            AArch64.AArch32SystemAccessTrap(EL2, 0x04);
        elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T5 == '1' then
            AArch32.TakeHypTrapException(0x04);
        elsif !ELUsingAArch32(EL1) && AMUSERENR_EL0.EN == '0' then
            if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
                AArch64.AArch32SystemAccessTrap(EL2, 0x04);
            else
                AArch64.AArch32SystemAccessTrap(EL1, 0x04);
        elsif EL2Enabled() && !ELUsingAArch32(EL2) && CPTR_EL2.TAM == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x04);
        elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TAM == '1' then
            AArch64.AArch32SystemAccessTrap(EL3, 0x04);
        else
            return AMEVCNTR1[UInt(CRm<0>:opc1<2:0>)];
    elsif PSTATE.EL == EL1 then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T5 == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x04);
        elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T5 == '1' then
            AArch32.TakeHypTrapException(0x04);
        elsif EL2Enabled() && !ELUsingAArch32(EL2) && CPTR_EL2.TAM == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x04);
        elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TAM == '1' then
            AArch64.AArch32SystemAccessTrap(EL3, 0x04);
        else
            return AMEVCNTR1[UInt(CRm<0>:opc1<2:0>)];
    elsif PSTATE.EL == EL2 then
        if HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TAM == '1' then
            AArch64.AArch32SystemAccessTrap(EL3, 0x04);
        else
            return AMEVCNTR1[UInt(CRm<0>:opc1<2:0>)];
```

```
    elsif PSTATE.EL == EL3 then
        return AMEVCNTR1[UInt(CRm<0>:opc1<2:0>)];
else
    UNDEFINED;
```

```
MCRR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <Rt2>, <CRm>
```

coproc	CRm	opc1
~~opc1~~	~~coproc~~	~~CRm~~
0b1111	0b010[n:3]	0b0[n:2:0]
~~0b[n:2:0]~~	~~0b1111~~	~~0b010[n:3]~~

```
if CRm == 100 then
    if PSTATE.EL == EL0 then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> != '11' && HSTR_EL2.T4 == '1'
then
            AArch64.AArch32SystemAccessTrap(EL2, 0x04);
        elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T4 == '1' then
            AArch32.TakeHypTrapException(0x04);
        elsif !ELUsingAArch32(EL1) && AMUSERENR_EL0.EN == '0' then
            if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
                AArch64.AArch32SystemAccessTrap(EL2, 0x04);
            else
                AArch64.AArch32SystemAccessTrap(EL1, 0x04);
        elsif EL2Enabled() && !ELUsingAArch32(EL2) && CPTR_EL2.TAM == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x04);
        elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TAM == '1' then
            AArch64.AArch32SystemAccessTrap(EL3, 0x04);
        else
            AMEVCNTR1[UInt(CRm<0>:opc1<2:0>)] = R[t2]:R[t];
    elsif PSTATE.EL == EL1 then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T4 == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x04);
        elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T4 == '1' then
            AArch32.TakeHypTrapException(0x04);
        elsif EL2Enabled() && !ELUsingAArch32(EL2) && CPTR_EL2.TAM == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x04);
        elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TAM == '1' then
            AArch64.AArch32SystemAccessTrap(EL3, 0x04);
        else
            AMEVCNTR1[UInt(CRm<0>:opc1<2:0>)] = R[t2]:R[t];
    elsif PSTATE.EL == EL2 then
        if HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TAM == '1' then
            AArch64.AArch32SystemAccessTrap(EL3, 0x04);
        else
            AMEVCNTR1[UInt(CRm<0>:opc1<2:0>)] = R[t2]:R[t];
    elsif PSTATE.EL == EL3 then
        AMEVCNTR1[UInt(CRm<0>:opc1<2:0>)] = R[t2]:R[t];
elsif CRm == 101 then
    if PSTATE.EL == EL0 then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> != '11' && HSTR_EL2.T5 == '1'
then
            AArch64.AArch32SystemAccessTrap(EL2, 0x04);
        elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T5 == '1' then
            AArch32.TakeHypTrapException(0x04);
        elsif !ELUsingAArch32(EL1) && AMUSERENR_EL0.EN == '0' then
            if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
                AArch64.AArch32SystemAccessTrap(EL2, 0x04);
            else
                AArch64.AArch32SystemAccessTrap(EL1, 0x04);
        elsif EL2Enabled() && !ELUsingAArch32(EL2) && CPTR_EL2.TAM == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x04);
        elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TAM == '1' then
            AArch64.AArch32SystemAccessTrap(EL3, 0x04);
        else
            AMEVCNTR1[UInt(CRm<0>:opc1<2:0>)] = R[t2]:R[t];
    elsif PSTATE.EL == EL1 then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T5 == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x04);
        elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T5 == '1' then
            AArch32.TakeHypTrapException(0x04);
        elsif EL2Enabled() && !ELUsingAArch32(EL2) && CPTR_EL2.TAM == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x04);
        elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TAM == '1' then
            AArch64.AArch32SystemAccessTrap(EL3, 0x04);
        else
            AMEVCNTR1[UInt(CRm<0>:opc1<2:0>)] = R[t2]:R[t];
    elsif PSTATE.EL == EL2 then
        if HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TAM == '1' then
            AArch64.AArch32SystemAccessTrap(EL3, 0x04);
        else
            AMEVCNTR1[UInt(CRm<0>:opc1<2:0>)] = R[t2]:R[t];
```

```
    elsif PSTATE.EL == EL3 then
        AMEVCNTR1[UInt(CRm<0>:opc1<2:0>)] = R[t2]:R[t];
else
    UNDEFINED;
```

2713/0312/20192018 2116:5942; e5e4db499bf9867a4b93324c4dbac985d3da93766379d01e197f1d40720d32d0f84e419e9187e009

(old) htmldiff from- (new)

(old) htmldiff from- (new)

AMEVTYPER0<n>, Activity Monitors Event Type Registers 0, n = 0 - 15

The AMEVTYPER0<n> characteristics are:

Purpose

Provides information on the events that an architected activity monitor event counter AMEVCNTR0<n> counts.

Configuration

AArch32 System register AMEVTYPER0<n> bits [31:0] are architecturally mapped to AArch64 System register AMEVTYPER0<n>_EL0[31:0] .

AArch32 System register AMEVTYPER0<n> bits [31:0] are architecturally mapped to External register AMEVTYPER0<n>[31:0] .

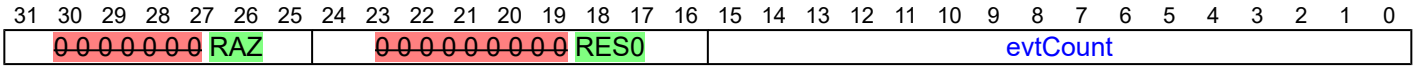
This register is present only when AMUv1 is implemented. Otherwise, direct accesses to AMEVTYPER0<n> are UNDEFINED.

Attributes

AMEVTYPER0<n> is a 32-bit register.

Field descriptions

The AMEVTYPER0<n> bit assignments are:



Bits [31:25]

Reserved, RAZ.

Bits [24:16]

Reserved, RES0.

evtCount, bits [15:0]

Event to count. The event number of the event that is counted by the architected activity monitor event counter AMEVCNTR0<n>. The value of this field is architecturally mandated for each architected counter.

The following table shows the mapping between required event numbers and the corresponding counters:

evtCount	Meaning	Applies when
0x0011	Processor frequency cycles	When n == 0
0x4004	Constant frequency cycles	When n == 1
0x0008	Instructions retired	When n == 2
0x4005	Memory stall cycles	When n == 3

Accessing the AMEVTYPER0<n>

If <n> is greater than or equal to the number of architected activity monitor event counters, reads and writes of AMEVTYPER0<n> are CONSTRAINED UNPREDICTABLE, and the following behaviors are permitted:

- Accesses to the register are UNDEFINED.
- Accesses to the register behave as RAZ/WI.
- Accesses to the register execute as a NOP.

Note

AMCGCR.CG0NC identifies the number of architected activity monitor event counters.

Accesses to this register use the following encodings:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
~~opc1~~	~~opc2~~	~~CRn~~	~~coproc~~	~~CRm~~
0b1111	0b000	0b1101	0b011[n:3]	0b[n:2:0]
~~0b000~~	~~0b[n:2:0]~~	~~0b1101~~	~~0b1111~~	~~0b011[n:3]~~

```
if PSTATE.EL == EL0 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> != '11' && HSTR_EL2.T13 == '1'
then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T13 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif !ELUsingAArch32(EL1) && AMUSERENR_EL0.EN == '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && CPTR_EL2.TAM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TAM == '1' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        return AMEVTYPER0[UInt(CRm<0>:opc2<2:0>)];
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T13 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T13 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && CPTR_EL2.TAM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TAM == '1' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        return AMEVTYPER0[UInt(CRm<0>:opc2<2:0>)];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TAM == '1' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        return AMEVTYPER0[UInt(CRm<0>:opc2<2:0>)];
elsif PSTATE.EL == EL3 then
    return AMEVTYPER0[UInt(CRm<0>:opc2<2:0>)];
```

27~~13~~/03~~12~~/20192~~018~~ 21~~16~~:59~~42~~; e5e4db499bf9867a4b93324c4dbac985d3da93766~~379d01e197f1d40720d32d0f84e419e9187e009~~

(old) htmldiff from- (new)

AMEVTYPER1<n>, Activity Monitors Event Type Registers 1, n = 0 - 15

The AMEVTYPER1<n> characteristics are:

Purpose

Provides information on the events that an auxiliary activity monitor event counter AMEVCNTR1<n> counts.

Configuration

AArch32 System register AMEVTYPER1<n> bits [31:0] are architecturally mapped to AArch64 System register AMEVTYPER1<n>_El0[31:0] .

AArch32 System register AMEVTYPER1<n> bits [31:0] are architecturally mapped to External register AMEVTYPER1<n>[31:0] .

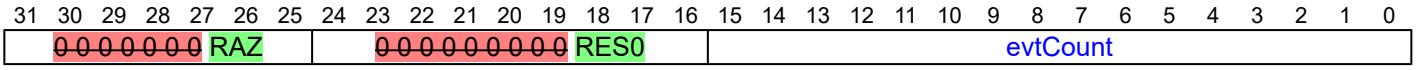
This register is present only when AMUv1 is implemented. Otherwise, direct accesses to AMEVTYPER1<n> are UNDEFINED.

Attributes

AMEVTYPER1<n> is a 32-bit register.

Field descriptions

The AMEVTYPER1<n> bit assignments are:



Bits [31:25]

Reserved, RAZ.

Bits [24:16]

Reserved, RES0.

evtCount, bits [15:0]

Event to count. The event number of the event that is counted by the auxiliary activity monitor event counter AMEVCNTR1<n>.

It is IMPLEMENTATION DEFINED what values are supported by each counter.

If software writes a value to this field which is not supported by the corresponding counter AMEVCNTR1<n>, then:

- It is UNPREDICTABLE which event will be counted.
- The value read back is UNKNOWN.

The event counted by AMEVCNTR1<n> might be fixed at implementation. In this case, the field is read-only and writes are UNDEFINED.

If the corresponding counter AMEVCNTR1<n> is enabled, writes to this register have UNPREDICTABLE results.

Accessing the AMEVTYPER1<n>

If <n> is greater than or equal to the number of auxiliary activity monitor event counters, reads and writes of AMEVTYPER1<n> are CONSTRAINED UNPREDICTABLE, and the following behaviors are permitted:

- Accesses to the register are UNDEFINED.
- Accesses to the register behave as RAZ/WI.
- Accesses to the register execute as a NOP.

Note

AMCGCR.CG1NC identifies the number of auxiliary activity monitor event counters.

Accesses to this register use the following encodings:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
~~opc1~~	~~opc2~~	~~CRn~~	~~coproc~~	~~CRm~~
0b1111	0b000	0b1101	0b111[n:3]	0b[n:2:0]
~~0b000~~	~~0b[n:2:0]~~	~~0b1101~~	~~0b1111~~	~~0b111[n:3]~~

```
if PSTATE.EL == EL0 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> != '11' && HSTR_EL2.T13 == '1'
then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T13 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif !ELUsingAArch32(EL1) && AMUSERENR_EL0.EN == '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && CPTR_EL2.TAM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TAM == '1' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        return AMEVTYPER1[UInt(CRm<0>:opc2<2:0>)];
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T13 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T13 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && CPTR_EL2.TAM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TAM == '1' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        return AMEVTYPER1[UInt(CRm<0>:opc2<2:0>)];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TAM == '1' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        return AMEVTYPER1[UInt(CRm<0>:opc2<2:0>)];
elsif PSTATE.EL == EL3 then
    return AMEVTYPER1[UInt(CRm<0>:opc2<2:0>)];
```

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
~~opc1~~	~~opc2~~	~~CRn~~	~~coproc~~	~~CRm~~

0b1111	0b000	0b1101	0b111[n:3]	0b[n:2:0]
~~0b000~~	~~0b[n:2:0]~~	~~0b1101~~	~~0b1111~~	~~0b111[n:3]~~

```
if PSTATE.EL == EL1 && EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T13 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
elsif PSTATE.EL == EL1 && EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T13 == '1' then
    AArch32.TakeHypTrapException(0x03);
elsif IsHighestEL(PSTATE.EL) then
    AMEVTYPER1[UInt(CRm<0>:opc2<2:0>)] = R[t];
else
    UNDEFINED;
```

27~~13~~/03~~12~~/20192018 21~~16~~:59~~42~~; e5e4db499bf9867a4b93324c4dbac985d3da93766379d01e197f1d40720d32d0f84e419e9187e009

(old) htmldiff from- (new)

AMUSERENR, Activity Monitors User Enable Register

The AMUSERENR characteristics are:

Purpose

Global user enable register for the activity monitors. Enables or disables EL0 access to the activity monitors. AMUSERENR is applicable to both the architected and the auxiliary counter groups.

Configuration

AArch32 System register AMUSERENR bits [31:0] are architecturally mapped to AArch64 System register AMUSERENR_EL0[31:0] .

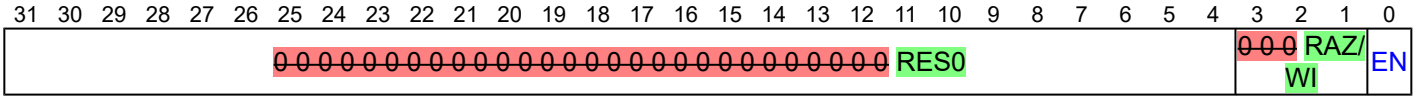
This register is present only when AMUv1 is implemented. Otherwise, direct accesses to AMUSERENR are UNDEFINED.

Attributes

AMUSERENR is a 32-bit register.

Field descriptions

The AMUSERENR bit assignments are:



Bits [31:4]

Reserved, RES0.

Bits [3:1]

Reserved, RAZ/WI.

EN, bit [0]

Traps EL0 accesses to the activity monitors registers to EL1.

EN	Meaning
0b0	EL0 accesses to the activity monitors registers are trapped to EL1.
0b1	This control does not cause any instructions to be trapped. Software can access all activity monitor registers at EL0.

Note

- AMUSERENR can always be read at EL0 and is not governed by this bit.

Accessing the AMUSERENR

Accesses to this register use the following encodings:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
~~opc1~~	~~opc2~~	~~CRn~~	~~coproc~~	~~CRm~~
0b1111	0b000	0b1101	0b0010	0b011
~~0b000~~	~~0b011~~	~~0b1101~~	~~0b1111~~	~~0b0010~~

```
if PSTATE.EL == EL0 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> != '11' && HSTR_EL2.T13 == '1'
then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T13 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && CPTR_EL2.TAM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TAM == '1' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        return AMUSERENRAMCFGR_EL0;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T13 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T13 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && CPTR_EL2.TAM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TAM == '1' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        return AMUSERENRAMCFGR_EL0;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TAM == '1' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        return AMUSERENRAMCFGR_EL0;
elsif PSTATE.EL == EL3 then
    return AMUSERENRAMCFGR_EL0;
```

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
~~opc1~~	~~opc2~~	~~CRn~~	~~coproc~~	~~CRm~~
0b1111	0b000	0b1101	0b0010	0b011
~~0b000~~	~~0b011~~	~~0b1101~~	~~0b1111~~	~~0b0010~~

```
if PSTATE.EL == EL0 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> != '11' && HSTR_EL2.T13 == '1'
then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T13 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && CPTR_EL2.TAM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TAM == '1' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        AMUSERENRAMCFGR_EL0 = R[t];
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T13 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T13 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && CPTR_EL2.TAM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TAM == '1' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        AMUSERENRAMCFGR_EL0 = R[t];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TAM == '1' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        AMUSERENRAMCFGR_EL0 = R[t];
elsif PSTATE.EL == EL3 then
    AMUSERENRAMCFGR_EL0 = R[t];
```

2713/0312/20192018 2116:5942; e5e4db499bf9867a4b93324c4dbac985d3da93766379d01e197f1d40720d32d0f84e419e9187e009

(old) htmldiff from- (new)

(old) htmldiff from- (new)

APSR, Application Program Status Register

The APSR characteristics are:

Purpose

Hold program status and control information.

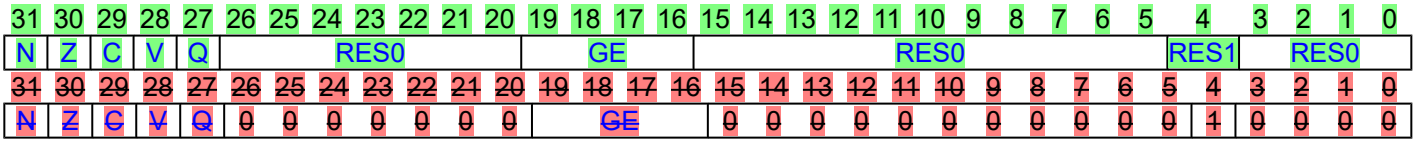
Configuration

Attributes

APSR is a 32-bit register.

Field descriptions

The APSR bit assignments are:



N, bit [31]

Negative condition flag. Set to bit[31] of the result of the last flag-setting instruction. If the result is regarded as a two's complement signed integer, then N is set to 1 if the result was negative, and N is set to 0 if the result was positive or zero.

Z, bit [30]

Zero condition flag. Set to 1 if the result of the last flag-setting instruction was zero, and to 0 otherwise. A result of zero often indicates an equal result from a comparison.

C, bit [29]

Carry condition flag. Set to 1 if the last flag-setting instruction resulted in a carry condition, for example an unsigned overflow on an addition.

V, bit [28]

Overflow condition flag. Set to 1 if the last flag-setting instruction resulted in an overflow condition, for example a signed overflow on an addition.

Q, bit [27]

Cumulative saturation bit. Set to 1 to indicate that overflow or saturation occurred in some instructions.

Bits [26:20]

Reserved, RES0.

GE, bits [19:16]

Greater than or Equal flags, for parallel addition and subtraction.

Bits [15:5]

Reserved, RES0.

Bit [4]

Reserved, RES1.

Bits [3:0]

Reserved, RES0.

It is permitted that, on a read of APSR:

- Bit[22] returns the value of PSTATE.PAN
- Bit[9] returns the value of PSTATE.E.
- Bits[8:6] return the value of PSTATE.{A, I, F}, the mask bits.
- Bit[4:0] returns the value of PSTATE.M[4:0]

Note

This is an exception to the general rule that an UNKNOWN field must not return information that cannot be obtained, at the current Privilege level, by an architected mechanism.

For more information see The Application Program State Register.

Accessing the APSR

APSR can be read using the MRS instruction and written using the MSR (register) or MSR (immediate) instructions. For more details, see MRS, MSR (register), and MSR (immediate) in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

2713/0312/20192018 2116:5942: e5e4db499bf9867a4b93324c4dbac985d3da93766379d01e197f1d40720d32d0f84e419e9187e009

(old) htmldiff from- (new)

ATS12NSOPR, Address Translate Stages 1 and 2 Non-secure Only PL1 Read

The ATS12NSOPR characteristics are:

Purpose

Performs stage 1 and 2 address translations as defined for PL1 and the Non-secure state, with permissions as if reading from the given virtual address.

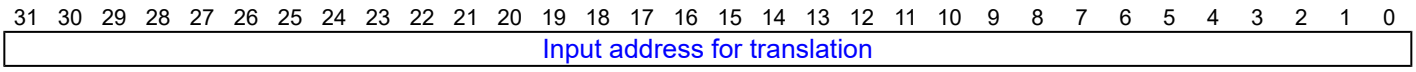
Configuration

Attributes

ATS12NSOPR is a 32-bit System instruction.

Field descriptions

The ATS12NSOPR input value bit assignments are:



Bits [31:0]

Input address for translation. The resulting address can be read from the PAR.

This System instruction takes a VA as input. The resulting address is the PA that is the output address of the stage 2 translation.

Executing the ATS12NSOPR instruction

Accesses to this instruction use the following encodings:

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
~~opc1~~	~~opc2~~	~~CRn~~	~~coproc~~	~~CRm~~
0b1111	0b000	0b0111	0b1000	0b100
~~0b000~~	~~0b100~~	~~0b0111~~	~~0b1111~~	~~0b1000~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T7 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T7 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif !ELUsingAArch32(EL2) && SCR_EL3.<NS,EEL2> == '01' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif !ELUsingAArch32(EL3) && SCR_EL3.NS == '0' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    ATS12NSOPR(R[t]);
elsif PSTATE.EL == EL3 then
    ATS12NSOPR(R[t]);
```

27~~13~~/03~~12~~/2019~~2018~~ 21~~16~~:59~~42~~; e5e4db499bf9867a4b93324c4dbac985d3da9376~~6379d01e197f1d40720d32d0f84e419e9187e009~~

(old) htmldiff from- (new)

(old) htmldiff from- (new)

ATS12NSOPW, Address Translate Stages 1 and 2 Non-secure Only PL1 Write

The ATS12NSOPW characteristics are:

Purpose

Performs stage 1 and 2 address translations as defined for PL1 and the Non-secure state, with permissions as if writing to the given virtual address.

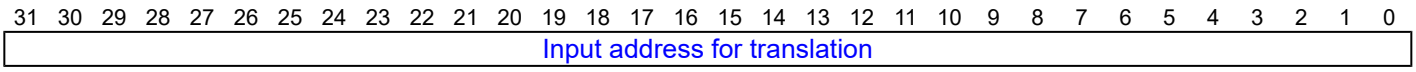
Configuration

Attributes

ATS12NSOPW is a 32-bit System instruction.

Field descriptions

The ATS12NSOPW input value bit assignments are:



Bits [31:0]

Input address for translation. The resulting address can be read from the PAR.

This System instruction takes a VA as input. The resulting address is the PA that is the output address of the stage 2 translation.

Executing the ATS12NSOPW instruction

Accesses to this instruction use the following encodings:

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
~~opc1~~	~~opc2~~	~~CRn~~	~~coproc~~	~~CRm~~
0b1111	0b000	0b0111	0b1000	0b101
~~0b000~~	~~0b101~~	~~0b0111~~	~~0b1111~~	~~0b1000~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T7 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T7 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif !ELUsingAArch32(EL2) && SCR_EL3.<NS,EEL2> == '01' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif !ELUsingAArch32(EL3) && SCR_EL3.NS == '0' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    ATS12NSOPW(R[t]);
elsif PSTATE.EL == EL3 then
    ATS12NSOPW(R[t]);
```

27~~13~~/03~~12~~/2019~~2018~~ 21~~16~~:59~~42~~; e5e4db499bf9867a4b93324c4dbac985d3da93766~~379d01e197f1d40720d32d0f84e419e9187e009~~

(old) htmldiff from- (new)

(old) htmldiff from- (new)

ATS12NSOUR, Address Translate Stages 1 and 2 Non-secure Only Unprivileged Read

The ATS12NSOUR characteristics are:

Purpose

Performs stage 1 and 2 address translations as defined for PL0 and the Non-secure state, with permissions as if reading from the given virtual address.

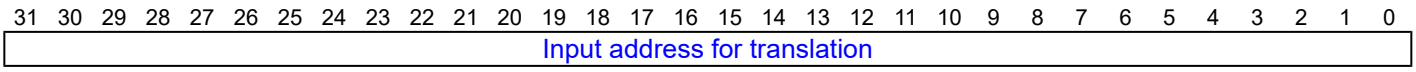
Configuration

Attributes

ATS12NSOUR is a 32-bit System instruction.

Field descriptions

The ATS12NSOUR input value bit assignments are:



Bits [31:0]

Input address for translation. The resulting address can be read from the PAR.

This System instruction takes a VA as input. The resulting address is the PA that is the output address of the stage 2 translation.

Executing the ATS12NSOUR instruction

Accesses to this instruction use the following encodings:

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
~~opc1~~	~~opc2~~	~~CRn~~	~~coproc~~	~~CRm~~
0b1111	0b000	0b0111	0b1000	0b110
~~0b000~~	~~0b110~~	~~0b0111~~	~~0b1111~~	~~0b1000~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T7 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T7 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif !ELUsingAArch32(EL2) && SCR_EL3.<NS,EEL2> == '01' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif !ELUsingAArch32(EL3) && SCR_EL3.NS == '0' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    ATS12NSOUR(R[t]);
elsif PSTATE.EL == EL3 then
    ATS12NSOUR(R[t]);
```

27~~13~~/03~~12~~/2019~~2018~~ 21~~16~~:59~~42~~; e5e4db499bf9867a4b93324c4dbac985d3da93766~~379d01e197f1d40720d32d0f84e419e9187e009~~

(old) htmldiff from- (new)

(old) htmldiff from- (new)

ATS12NSOUW, Address Translate Stages 1 and 2 Non-secure Only Unprivileged Write

The ATS12NSOUW characteristics are:

Purpose

Performs stage 1 and 2 address translations as defined for PL0 and the Non-secure state, with permissions as if writing to the given virtual address.

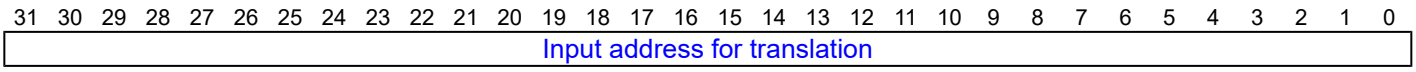
Configuration

Attributes

ATS12NSOUW is a 32-bit System instruction.

Field descriptions

The ATS12NSOUW input value bit assignments are:



Bits [31:0]

Input address for translation. The resulting address can be read from the PAR.

This System instruction takes a VA as input. The resulting address is the PA that is the output address of the stage 2 translation.

Executing the ATS12NSOUW instruction

Accesses to this instruction use the following encodings:

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
~~opc1~~	~~opc2~~	~~CRn~~	~~coproc~~	~~CRm~~
0b1111	0b000	0b0111	0b1000	0b111
~~0b000~~	~~0b111~~	~~0b0111~~	~~0b1111~~	~~0b1000~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T7 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T7 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif !ELUsingAArch32(EL2) && SCR_EL3.<NS,EEL2> == '01' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif !ELUsingAArch32(EL3) && SCR_EL3.NS == '0' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    ATS12NSOUW(R[t]);
elsif PSTATE.EL == EL3 then
    ATS12NSOUW(R[t]);
```

27~~13~~/03~~12~~/2019~~2018~~ 21~~16~~:59~~42~~; e5e4db499bf9867a4b93324c4dbac985d3da93766~~379d01e197f1d40720d32d0f84e419e9187e009~~

(old) htmldiff from- (new)

(old) htmldiff from- (new)

ATS1CPR, Address Translate Stage 1 Current state PL1 Read

The ATS1CPR characteristics are:

Purpose

Performs stage 1 address translation as defined for PL1 and the current Security state, with permissions as if reading from the given virtual address.

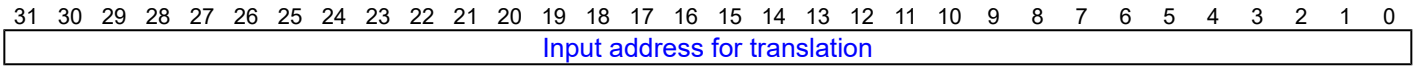
Configuration

Attributes

ATS1CPR is a 32-bit System instruction.

Field descriptions

The ATS1CPR input value bit assignments are:



Bits [31:0]

Input address for translation. The resulting address can be read from the PAR.

This System instruction takes a VA as input. If EL2 is implemented and enabled in the current Security state, the resulting address is the IPA that is the output address of the stage 1 translation. Otherwise, the resulting address is a PA.

Executing the ATS1CPR instruction

Accesses to this instruction use the following encodings:

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
~~opc1~~	~~opc2~~	~~CRn~~	~~coproc~~	~~CRm~~
0b1111	0b000	0b0111	0b1000	0b000
~~0b000~~	~~0b000~~	~~0b0111~~	~~0b1111~~	~~0b1000~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T7 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T7 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        ATS1CPR(R[t]);
elsif PSTATE.EL == EL2 then
    ATS1CPR(R[t]);
elsif PSTATE.EL == EL3 then
    ATS1CPR(R[t]);
```

2713/0312/20192018 2116:5942; e5e4db499bf9867a4b93324c4dbac985d3da93766379d01e197f1d40720d32d0f84e419e9187e009

(old) htmldiff from- (new)

(old) htmldiff from- (new)

ATS1CPRP, Address Translate Stage 1 Current state PL1 Read PAN

The ATS1CPRP characteristics are:

Purpose

Performs a stage 1 address translation at PL1 and in the current Security state, where the value of PSTATE.PAN determines if a read from a location will generate a permission fault for a privileged access.

Configuration

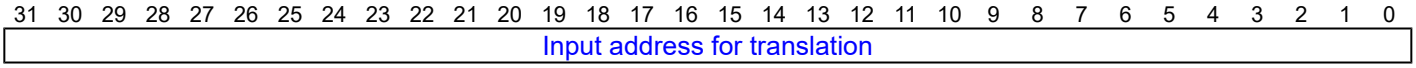
This instruction is present only when ARMv8.2-ATS1E1 is implemented. Otherwise, direct accesses to ATS1CPRP are UNDEFINED.

Attributes

ATS1CPRP is a 32-bit System instruction.

Field descriptions

The ATS1CPRP input value bit assignments are:



Bits [31:0]

Input address for translation. The resulting address can be read from the PAR.

This System instruction takes a VA as input. If EL2 is implemented and enabled in the current Security state, the resulting address is the IPA that is the output address of the stage 1 translation. Otherwise, the resulting address is a PA.

Executing the ATS1CPRP instruction

Accesses to this instruction use the following encodings:

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
~~opc1~~	~~opc2~~	~~CRn~~	~~coproc~~	~~CRm~~
0b1111	0b000	0b0111	0b1001	0b000
~~0b000~~	~~0b000~~	~~0b0111~~	~~0b1111~~	~~0b1001~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T7 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T7 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        ATS1CPRP(R[t]);
elsif PSTATE.EL == EL2 then
    ATS1CPRP(R[t]);
elsif PSTATE.EL == EL3 then
    ATS1CPRP(R[t]);
```

27~~13~~/03~~12~~/2019~~2018~~ 21~~16~~:59~~42~~; e5e4db499bf9867a4b93324c4dbac985d3da93766~~379d01e197f1d40720d32d0f84e419e9187e009~~

(old) htmldiff from- (new)

(old) htmldiff from- (new)

ATS1CPW, Address Translate Stage 1 Current state PL1 Write

The ATS1CPW characteristics are:

Purpose

Performs stage 1 address translation as defined for PL1 and the current Security state, with permissions as if writing to the given virtual address.

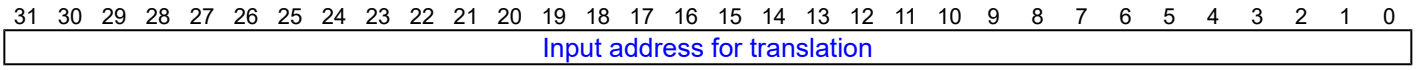
Configuration

Attributes

ATS1CPW is a 32-bit System instruction.

Field descriptions

The ATS1CPW input value bit assignments are:



Bits [31:0]

Input address for translation. The resulting address can be read from the PAR.

This System instruction takes a VA as input. If EL2 is implemented and enabled in the current Security state, the resulting address is the IPA that is the output address of the stage 1 translation. Otherwise, the resulting address is a PA.

Executing the ATS1CPW instruction

Accesses to this instruction use the following encodings:

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
~~opc1~~	~~opc2~~	~~CRn~~	~~coproc~~	~~CRm~~
0b1111	0b000	0b0111	0b1000	0b001
~~0b000~~	~~0b001~~	~~0b0111~~	~~0b1111~~	~~0b1000~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T7 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T7 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        ATS1CPW(R[t]);
elsif PSTATE.EL == EL2 then
    ATS1CPW(R[t]);
elsif PSTATE.EL == EL3 then
    ATS1CPW(R[t]);
```

2713/0312/20192018 2116:5942; e5e4db499bf9867a4b93324c4dbac985d3da93766379d01e197f1d40720d32d0f84e419e9187e009

(old) htmldiff from- (new)

(old) htmldiff from- (new)

ATS1CPWP, Address Translate Stage 1 Current state PL1 Write PAN

The ATS1CPWP characteristics are:

Purpose

When ARMv8.2-ATS1E1 is implemented, performs a stage 1 address translation at PL1 and in the current Security state, where the value of PSTATE.PAN determines if a write to the location will generate a permission fault for a privileged access.

Configuration

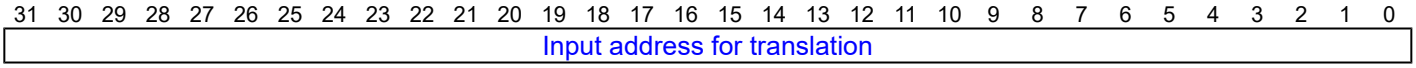
This instruction is present only when ARMv8.2-ATS1E1 is implemented. Otherwise, direct accesses to ATS1CPWP are UNDEFINED.

Attributes

ATS1CPWP is a 32-bit System instruction.

Field descriptions

The ATS1CPWP input value bit assignments are:



Bits [31:0]

Input address for translation. The resulting address can be read from the PAR.

This System instruction takes a VA as input. If EL2 is implemented and enabled in the current Security state, the resulting address is the IPA that is the output address of the stage 1 translation. Otherwise, the resulting address is a PA.

Executing the ATS1CPWP instruction

Accesses to this instruction use the following encodings:

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
~~opc1~~	~~opc2~~	~~CRn~~	~~coproc~~	~~CRm~~
0b1111	0b000	0b0111	0b1001	0b001
~~0b000~~	~~0b001~~	~~0b0111~~	~~0b1111~~	~~0b1001~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T7 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T7 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        ATS1CPWP(R[t]);
elsif PSTATE.EL == EL2 then
    ATS1CPWP(R[t]);
elsif PSTATE.EL == EL3 then
    ATS1CPWP(R[t]);
```

27~~13~~/03~~12~~/2019~~2018~~ 21~~16~~:59~~42~~; e5e4db499bf9867a4b93324c4dbac985d3da93766~~6379d01e197f1d40720d32d0f84e419e9187e009~~

(old) htmldiff from- (new)

(old) htmldiff from- (new)

ATS1CUR, Address Translate Stage 1 Current state Unprivileged Read

The ATS1CUR characteristics are:

Purpose

Performs stage 1 address translation as defined for PL0 and the current Security state, with permissions as if reading from the given virtual address.

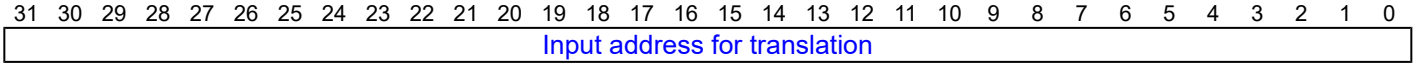
Configuration

Attributes

ATS1CUR is a 32-bit System instruction.

Field descriptions

The ATS1CUR input value bit assignments are:



Bits [31:0]

Input address for translation. The resulting address can be read from the PAR.

This System instruction takes a VA as input. If EL2 is implemented and enabled in the current Security state, the resulting address is the IPA that is the output address of the stage 1 translation. Otherwise, the resulting address is a PA.

Executing the ATS1CUR instruction

Accesses to this instruction use the following encodings:

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
~~opc1~~	~~opc2~~	~~CRn~~	~~coproc~~	~~CRm~~
0b1111	0b000	0b0111	0b1000	0b010
~~0b000~~	~~0b010~~	~~0b0111~~	~~0b1111~~	~~0b1000~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T7 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T7 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        ATS1CUR(R[t]);
elsif PSTATE.EL == EL2 then
    ATS1CUR(R[t]);
elsif PSTATE.EL == EL3 then
    ATS1CUR(R[t]);
```

27~~13~~/03~~12~~/2019~~2018~~ 21~~16~~:59~~42~~; e5e4db499bf9867a4b93324c4dbac985d3da93766~~379d01e197f1d40720d32d0f84e419e9187e009~~

(old) htmldiff from- (new)

(old) htmldiff from- (new)

ATS1CUW, Address Translate Stage 1 Current state Unprivileged Write

The ATS1CUW characteristics are:

Purpose

Performs stage 1 address translation as defined for PL0 and the current Security state, with permissions as if writing to the given virtual address.

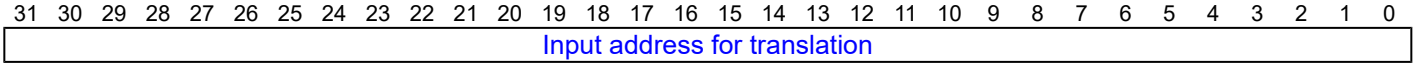
Configuration

Attributes

ATS1CUW is a 32-bit System instruction.

Field descriptions

The ATS1CUW input value bit assignments are:



Bits [31:0]

Input address for translation. The resulting address can be read from the PAR.

This System instruction takes a VA as input. If EL2 is implemented and enabled in the current Security state, the resulting address is the IPA that is the output address of the stage 1 translation. Otherwise, the resulting address is a PA.

Executing the ATS1CUW instruction

Accesses to this instruction use the following encodings:

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
~~opc1~~	~~opc2~~	~~CRn~~	~~coproc~~	~~CRm~~
0b1111	0b000	0b0111	0b1000	0b011
~~0b000~~	~~0b011~~	~~0b0111~~	~~0b1111~~	~~0b1000~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T7 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T7 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        ATS1CUW(R[t]);
elsif PSTATE.EL == EL2 then
    ATS1CUW(R[t]);
elsif PSTATE.EL == EL3 then
    ATS1CUW(R[t]);
```

27~~13~~/03~~12~~/2019~~2018~~ 21~~16~~:59~~42~~; e5e4db499bf9867a4b93324c4dbac985d3da93766~~379d01e197f1d40720d32d0f84e419e9187e009~~

(old) htmldiff from- (new)

(old) htmldiff from- (new)

ATS1HR, Address Translate Stage 1 Hyp mode Read

The ATS1HR characteristics are:

Purpose

Performs stage 1 address translation as defined for PL2 and the Non-secure state, with permissions as if reading from the given virtual address.

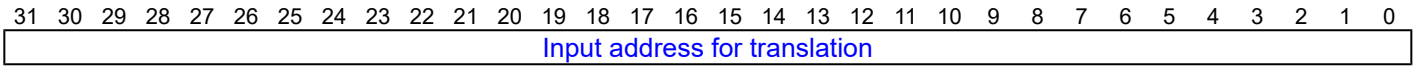
Configuration

Attributes

ATS1HR is a 32-bit System instruction.

Field descriptions

The ATS1HR input value bit assignments are:



Bits [31:0]

Input address for translation. The resulting address can be read from the PAR.

This System instruction takes a VA as input. The resulting address is the PA that is the output address of the translation.

Executing the ATS1HR instruction

If this instruction is executed in a Secure privileged mode other than Monitor mode, then the behavior is CONSTRAINED UNPREDICTABLE, and one of the following behaviors must occur:

- The instruction is UNDEFINED.
- The instruction is treated as a NOP.
- The instruction executes as if it had been executed in Monitor mode.

Accesses to this instruction use the following encodings:

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
~~opc1~~	~~opc2~~	~~CRn~~	~~coproc~~	~~CRm~~
0b1111	0b100	0b0111	0b1000	0b000
~~0b100~~	~~0b000~~	~~0b0111~~	~~0b1111~~	~~0b1000~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T7 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T7 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        ATS1HR(R[t]);
elsif PSTATE.EL == EL2 then
    ATS1HR(R[t]);
elsif PSTATE.EL == EL3 then
    ATS1HR(R[t]);
```

27~~13~~/03~~12~~/2019~~2018~~ 21~~16~~:59~~42~~; e5e4db499bf9867a4b93324c4dbac985d3da93766~~379d01e197f1d40720d32d0f84e419e9187e009~~

(old) htmldiff from- (new)

(old) htmldiff from- (new)

ATS1HW, Address Translate Stage 1 Hyp mode Write

The ATS1HW characteristics are:

Purpose

Performs stage 1 address translation as defined for PL2 and the Non-secure state, with permissions as if writing to the given virtual address.

Configuration

Attributes

ATS1HW is a 32-bit System instruction.

Field descriptions

The ATS1HW input value bit assignments are:

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Input address for translation																															

Bits [31:0]

Input address for translation. The resulting address can be read from the PAR.

This System instruction takes a VA as input. The resulting address is the PA that is the output address of the translation.

Executing the ATS1HW instruction

If this instruction is executed in a Secure privileged mode other than Monitor mode, then the behavior is CONSTRAINED UNPREDICTABLE, and one of the following behaviors must occur:

- The instruction is UNDEFINED.
- The instruction is treated as a NOP.
- The instruction executes as if it had been executed in Monitor mode.

Accesses to this instruction use the following encodings:

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
~~opc1~~	~~opc2~~	~~CRn~~	~~coproc~~	~~CRm~~
0b1111	0b100	0b0111	0b1000	0b001
~~0b100~~	~~0b001~~	~~0b0111~~	~~0b1111~~	~~0b1000~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T7 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T7 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        ATS1HW(R[t]);
elsif PSTATE.EL == EL2 then
    ATS1HW(R[t]);
elsif PSTATE.EL == EL3 then
    ATS1HW(R[t]);
```

27~~13~~/03~~12~~/2019~~2018~~ 21~~16~~:59~~42~~; e5e4db499bf9867a4b93324c4dbac985d3da93766~~6379d01e197f1d40720d32d0f84e419e9187e009~~

(old) htmldiff from- (new)

CCSIDR, Current Cache Size ID Register

The CCSIDR characteristics are:

Purpose

Provides information about the architecture of the currently selected cache.

When ARMv8.3-CCIDX is implemented, this register is used in conjunction with CCSIDR2.

Configuration

AArch32 System register CCSIDR bits [31:0] are architecturally mapped to AArch64 System register CCSIDR_EL1[31:0] .

The implementation includes one CCSIDR for each cache that it can access. CSSELR and the Security state select which Cache Size ID Register is accessible.

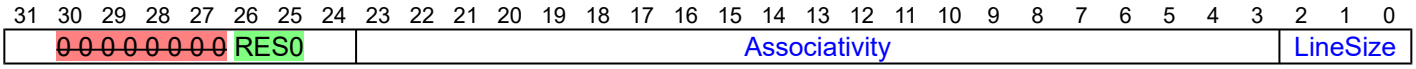
Attributes

CCSIDR is a 32-bit register.

Field descriptions

The CCSIDR bit assignments are:

When ARMv8.3-CCIDX is implemented:



Note

The parameters NumSets, Associativity, and LineSize in these registers define the architecturally visible parameters that are required for the cache maintenance by Set/Way instructions. They are not guaranteed to represent the actual microarchitectural features of a design. You cannot make any inference about the actual sizes of caches based on these parameters.

Bits [31:24]

Reserved, RES0.

Associativity, bits [23:3]

(Associativity of cache) - 1, therefore a value of 0 indicates an associativity of 1. The associativity does not have to be a power of 2.

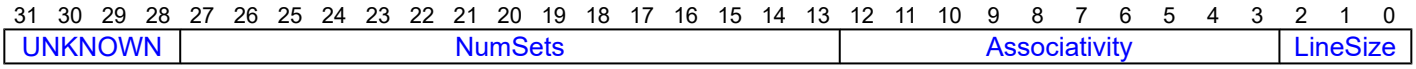
LineSize, bits [2:0]

(Log_2(Number of bytes in cache line)) - 4. For example:

For a line length of 16 bytes: $Log_2(16) = 4$, LineSize entry = 0. This is the minimum line length.

For a line length of 32 bytes: $Log_2(32) = 5$, LineSize entry = 1.

Otherwise:



Note

The parameters NumSets, Associativity, and LineSize in these registers define the architecturally visible parameters that are required for the cache maintenance by Set/Way instructions. They are not guaranteed to represent the actual microarchitectural features of a design. You cannot make any inference about the actual sizes of caches based on these parameters.

Bits~~UNKNOWN, bits~~ [31:28]

Reserved, UNKNOWN.

NumSets, bits [27:13]

(Number of sets in cache) - 1, therefore a value of 0 indicates 1 set in the cache. The number of sets does not have to be a power of 2.

Associativity, bits [12:3]

(Associativity of cache) - 1, therefore a value of 0 indicates an associativity of 1. The associativity does not have to be a power of 2.

LineSize, bits [2:0]

(Log_2(Number of bytes in cache line)) - 4. For example:

For a line length of 16 bytes: $Log_2(16) = 4$, LineSize entry = 0. This is the minimum line length.

For a line length of 32 bytes: $Log_2(32) = 5$, LineSize entry = 1.

Accessing the CCSIDR

If CSSELR.Level is programmed to a cache level that is not implemented, then on a read of the CCSIDR the behavior is CONSTRAINED UNPREDICTABLE, and can be one of the following:

- The CCSIDR read is treated as NOP.
- The CCSIDR read is UNDEFINED.
- The CCSIDR read returns an UNKNOWN value.

Accesses to this register use the following encodings:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
~~opc1~~	~~opc2~~	~~CRn~~	~~coproc~~	~~CRm~~
0b1111	0b001	0b0000	0b0000	0b000
~~0b001~~	~~0b000~~	~~0b0000~~	~~0b1111~~	~~0b0000~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T0 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T0 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TID2 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TID4 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TID2 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR2.TID4 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        return CCSIDR;
elsif PSTATE.EL == EL2 then
    return CCSIDR;
elsif PSTATE.EL == EL3 then
    return CCSIDR;
```

27~~13~~/03~~12~~/20192018 21~~16~~:59~~42~~; e5e4db499bf9867a4b93324c4dbac985d3da93766379d01e197f1d40720d32d0f84e419e9187e009

(old) htmldiff from- (new)

(old) htmldiff from- (new)

CNTHP_CTL, Counter-timer Hyp Physical Timer Control register

The CNTHP_CTL characteristics are:

Purpose

Control register for the Hyp mode physical timer.

Configuration

AArch32 System register CNTHP_CTL bits [31:0] are architecturally mapped to AArch64 System register CNTHP_CTL_EL2[31:0] .

If EL2 is not implemented, this register is RES0 from EL3.

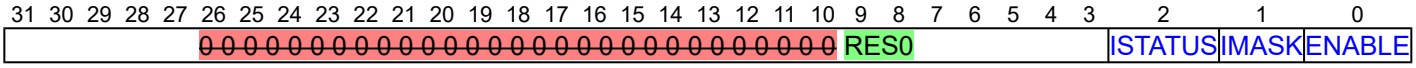
Some or all RW fields of this register have defined reset values. These apply only if the PE resets into EL2 with EL2 using AArch32, or into EL3 with EL3 using AArch32. Otherwise, RW fields in this register reset to architecturally UNKNOWN values.

Attributes

CNTHP_CTL is a 32-bit register.

Field descriptions

The CNTHP_CTL bit assignments are:



Bits [31:3]

Reserved, RES0.

ISTATUS, bit [2]

The status of the timer. This bit indicates whether the timer condition is met:

ISTATUS	Meaning
0b0	Timer condition is not met.
0b1	Timer condition is met.

When the value of the ENABLE bit is 1, ISTATUS indicates whether the timer condition is met. ISTATUS takes no account of the value of the IMASK bit. If the value of ISTATUS is 1 and the value of IMASK is 0 then the timer interrupt is asserted.

When the value of the ENABLE bit is 0, the ISTATUS field is UNKNOWN.

For more information see 'Operation of the CompareValue views of the timers' and 'Operation of the TimerValue views of the timers' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, chapter D6.

This bit is read-only.

~~In a system where the PE resets into EL2 or EL3, this field resets to an architecturally UNKNOWN value.~~

IMASK, bit [1]

Timer interrupt mask bit. Permitted values are:

IMASK	Meaning
0b0	Timer interrupt is not masked by the IMASK bit.
0b1	Timer interrupt is masked by the IMASK bit.

For more information, see the description of the ISTATUS bit.

In a system where the PE resets into EL2 or EL3, this field resets to an architecturally UNKNOWN value.

ENABLE, bit [0]

Enables the timer. Permitted values are:

ENABLE	Meaning
0b0	Timer disabled.
0b1	Timer enabled.

Setting this bit to 0 disables the timer output signal, but the timer value accessible from CNTHP_TVAL continues to count down.

Note

Disabling the output signal might be a power-saving option.

In a system where the PE resets into EL2 or EL3, this field resets to 0.

Accessing the CNTHP_CTL

Accesses to this register use the following encodings:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
~~opc1~~	~~opc2~~	~~CRn~~	~~coproc~~	~~CRm~~
0b1111	0b100	0b1110	0b0010	0b001
~~0b100~~	~~0b001~~	~~0b1110~~	~~0b1111~~	~~0b0010~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    return CNTHP_CTL;
elsif PSTATE.EL == EL3 then
    return CNTHP_CTL;
```

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
~~opc1~~	~~opc2~~	~~CRn~~	~~coproc~~	~~CRm~~
0b1111	0b100	0b1110	0b0010	0b001
~~0b100~~	~~0b001~~	~~0b1110~~	~~0b1111~~	~~0b0010~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    CNTHP_CTL = R[t];
elsif PSTATE.EL == EL3 then
    CNTHP_CTL = R[t];
```

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
~~opc1~~	~~opc2~~	~~CRn~~	~~coproc~~	~~CRm~~
0b1111	0b000	0b1110	0b0010	0b001
~~0b000~~	~~0b001~~	~~0b1110~~	~~0b1111~~	~~0b0010~~

```
if PSTATE.EL == EL0 then
    if !ELUsingAArch32(EL1) && !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') && CNTKCTL_EL1.EL0PTEN
== '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x03);
    elsif ELUsingAArch32(EL1) && CNTKCTL.PL0PTEN == '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TGE == '1' then
            AArch32.TakeHypTrapException(0x00);
        else
            UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '0' && CNTHCTL_EL2.EL1PCEN == '0'
then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '10' && CNTHCTL_EL2.EL1PTEN
== '0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && CNTHCTL_EL2.EL0PTEN
== '0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && CNTHCTL.PL1PCEN == '0' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '0'
then
        return CNTHPS_CTL_EL2;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '1'
then
        return CNTHP_CTL_EL2;
    else
        return CNTP_CTL;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '0' && CNTHCTL_EL2.EL1PCEN == '0'
then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '1' && CNTHCTL_EL2.EL1PTEN == '0'
then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && CNTHCTL.PL1PCEN == '0' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) then
        if SCR.NS == '0' then
            return CNTP_CTL_S;
        else
            return CNTP_CTL_NS;
    else
        return CNTP_CTL;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && ELUsingAArch32(EL3) then
        return CNTP_CTL_NS;
    else
        return CNTP_CTL;
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        return CNTP_CTL_S;
    else
        return CNTP_CTL_NS;
```

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
~~opc1~~	~~opc2~~	~~CRn~~	~~coproc~~	~~CRm~~
0b1111	0b000	0b1110	0b0010	0b001
~~0b000~~	~~0b001~~	~~0b1110~~	~~0b1111~~	~~0b0010~~

```
if PSTATE.EL == EL0 then
    if !ELUsingAArch32(EL1) && !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') && CNTKCTL_EL1.EL0PTEN
== '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x03);
    elsif ELUsingAArch32(EL1) && CNTKCTL.PL0PTEN == '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TGE == '1' then
            AArch32.TakeHypTrapException(0x00);
        else
            UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '0' && CNTHCTL_EL2.EL1PCEN == '0'
then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '10' && CNTHCTL_EL2.EL1PTEN
== '0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && CNTHCTL_EL2.EL0PTEN
== '0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && CNTHCTL.PL1PCEN == '0' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '0'
then
        CNTHPS_CTL_EL2 = R[t];
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '1'
then
        CNTHP_CTL_EL2 = R[t];
    else
        CNTP_CTL = R[t];
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '0' && CNTHCTL_EL2.EL1PCEN == '0'
then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '1' && CNTHCTL_EL2.EL1PTEN == '0'
then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && CNTHCTL.PL1PCEN == '0' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) then
        if SCR.NS == '0' then
            CNTP_CTL_S = R[t];
        else
            CNTP_CTL_NS = R[t];
    else
        CNTP_CTL = R[t];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && ELUsingAArch32(EL3) then
        CNTP_CTL_NS = R[t];
    else
        CNTP_CTL = R[t];
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        CNTP_CTL_S = R[t];
    else
        CNTP_CTL_NS = R[t];
```

27~~13~~/03~~12~~/2019~~2018~~ 21~~16~~:59~~42~~: e5e4db499bf9867a4b93324c4dbac985d3da93766379d01e197f1d40720d32d0f84e419e9187e009

(old) htmldiff from- (new)

CNTHPS_CTL, Counter-timer Secure Physical Timer Control Register (EL2)

The CNTHPS_CTL characteristics are:

Purpose

Provides AArch32 access to the Secure EL2 physical timer.

Note

The Secure EL2 timer is implemented by ARMv8.4-SecEL2. It is only accessible from AArch32 state when EL2 is using AArch64 and the value of SCR_EL3.{EEL2, NS} is {1, 0}.

Configuration

AArch32 System register CNTHPS_CTL bits [31:0] are architecturally mapped to AArch64 System register CNTHPS_CTL_EL2[31:0] .

This register is present only when ARMv8.4-SecEL2 is implemented. Otherwise, direct accesses to CNTHPS_CTL are UNDEFINED.

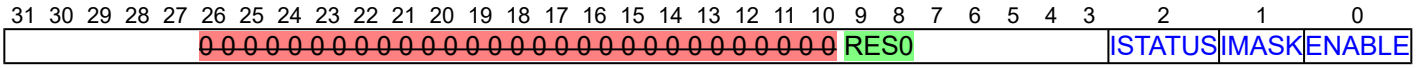
RW fields in this register reset to architecturally UNKNOWN values.

Attributes

CNTHPS_CTL is a 32-bit register.

Field descriptions

The CNTHPS_CTL bit assignments are:



Bits [31:3]

Reserved, RES0.

ISTATUS, bit [2]

The status of the timer. This bit indicates whether the timer condition is met:

ISTATUS	Meaning
0b0	Timer condition is not met.
0b1	Timer condition is met.

When the value of the CNTHPS_CTL.ENABLE bit is 1, ISTATUS indicates whether the timer condition is met. ISTATUS takes no account of the value of the IMASK bit. If the value of ISTATUS is 1 and the value of IMASK is 0 then the timer interrupt is asserted.

When the value of the CNTHPS_CTL.ENABLE bit is 0, the ISTATUS field is UNKNOWN.

For more information see 'Operation of the CompareValue views of the timers' and 'Operation of the TimerValue views of the timers' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, chapter D6.

This bit is read-only.

IMASK, bit [1]

Timer interrupt mask bit. Permitted values are:

IMASK	Meaning
0b0	Timer interrupt is not masked by the IMASK bit.
0b1	Timer interrupt is masked by the IMASK bit.

For more information, see the description of the ISTATUS bit.

This field resets to an architecturally UNKNOWN value.

ENABLE, bit [0]

Enables the timer. Permitted values are:

ENABLE	Meaning
0b0	Timer disabled.
0b1	Timer enabled.

Setting this bit to 0 disables the timer output signal, but the timer value accessible from CNTHPS_TVAL_EL2 continues to count down.

Note

Disabling the output signal might be a power-saving option.

This field resets to an architecturally UNKNOWN value.

Accessing the CNTHPS_CTL

This register is accessed using the encoding for CNTP_CTL.

Accesses to this register use the following encodings:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
~~opc1~~	~~opc2~~	~~CRn~~	~~coproc~~	~~CRm~~
0b1111	0b000	0b1110	0b0010	0b001
~~0b000~~	~~0b001~~	~~0b1110~~	~~0b1111~~	~~0b0010~~

```
if PSTATE.EL == EL0 then
    if !ELUsingAArch32(EL1) && !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') && CNTKCTL_EL1.EL0PTEN
== '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x03);
    elsif ELUsingAArch32(EL1) && CNTKCTL.PL0PTEN == '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TGE == '1' then
            AArch32.TakeHypTrapException(0x00);
        else
            UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '0' && CNTHCTL_EL2.EL1PCEN == '0'
then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '10' && CNTHCTL_EL2.EL1PTEN
== '0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && CNTHCTL_EL2.EL0PTEN
== '0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && CNTHCTL.PL1PCEN == '0' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '0'
then
        return CNTHPS_CTL_EL2;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '1'
then
        return CNTHP_CTL_EL2;
    else
        return CNTP_CTL;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '0' && CNTHCTL_EL2.EL1PCEN == '0'
then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '1' && CNTHCTL_EL2.EL1PTEN == '0'
then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && CNTHCTL.PL1PCEN == '0' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) then
        if SCR.NS == '0' then
            return CNTP_CTL_S;
        else
            return CNTP_CTL_NS;
    else
        return CNTP_CTL;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && ELUsingAArch32(EL3) then
        return CNTP_CTL_NS;
    else
        return CNTP_CTL;
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        return CNTP_CTL_S;
    else
        return CNTP_CTL_NS;
```

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
~~opc1~~	~~opc2~~	~~CRn~~	~~coproc~~	~~CRm~~
0b1111	0b000	0b1110	0b0010	0b001
~~0b000~~	~~0b001~~	~~0b1110~~	~~0b1111~~	~~0b0010~~

```
if PSTATE.EL == EL0 then
    if !ELUsingAArch32(EL1) && !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') && CNTKCTL_EL1.EL0PTEN
== '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x03);
    elsif ELUsingAArch32(EL1) && CNTKCTL.PL0PTEN == '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TGE == '1' then
            AArch32.TakeHypTrapException(0x00);
        else
            UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '0' && CNTHCTL_EL2.EL1PCEN == '0'
then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '10' && CNTHCTL_EL2.EL1PTEN
== '0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && CNTHCTL_EL2.EL0PTEN
== '0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && CNTHCTL.PL1PCEN == '0' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '0'
then
        CNTHPS_CTL_EL2 = R[t];
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '1'
then
        CNTHP_CTL_EL2 = R[t];
    else
        CNTP_CTL = R[t];
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '0' && CNTHCTL_EL2.EL1PCEN == '0'
then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '1' && CNTHCTL_EL2.EL1PTEN == '0'
then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && CNTHCTL.PL1PCEN == '0' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) then
        if SCR.NS == '0' then
            CNTP_CTL_S = R[t];
        else
            CNTP_CTL_NS = R[t];
    else
        CNTP_CTL = R[t];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && ELUsingAArch32(EL3) then
        CNTP_CTL_NS = R[t];
    else
        CNTP_CTL = R[t];
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        CNTP_CTL_S = R[t];
    else
        CNTP_CTL_NS = R[t];
```

27~~13~~/03~~12~~/2019~~2018~~ 21~~16~~:59~~42~~: e5e4db499bf9867a4b93324c4dbac985d3da93766379d01e197f1d40720d32d0f84e419e9187e009

(old) htmldiff from- (new)

CNTHPS_CVAL, Counter-timer Secure Physical Timer CompareValue Register (EL2)

The CNTHPS_CVAL characteristics are:

Purpose

Provides AArch32 access to the compare value for the Secure EL2 physical timer.

Note

The Secure EL2 timer is implemented by ARMv8.4-SecEL2. It is only accessible from AArch32 state when EL2 is using AArch64 and the value of SCR_EL3.{EEL2, NS} is {1, 0}.

Configuration

AArch32 System register CNTHPS_CVAL bits [63:0] are architecturally mapped to AArch64 System register CNTHPS_CVAL_EL2[63:0] .

This register is present only when ARMv8.4-SecEL2 is implemented. Otherwise, direct accesses to CNTHPS_CVAL are UNDEFINED.

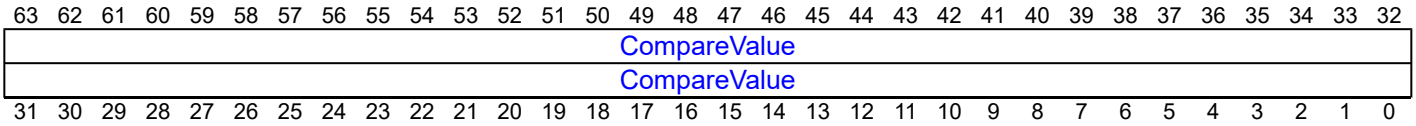
RW fields in this register reset to architecturally UNKNOWN values.

Attributes

CNTHPS_CVAL is a 64-bit register.

Field descriptions

The CNTHPS_CVAL bit assignments are:



CompareValue, bits [63:0]

Holds the EL2 physical timer CompareValue.

When CNTHPS_CTL_EL2.ENABLE is 1, the timer condition is met when (CNTPCT_EL0 - CompareValue) is greater than or equal to zero. This means that CompareValue acts like a 64-bit upcounter timer. When the timer condition is met:

- CNTHPS_CTL_EL2.ISTATUS is set to 1.
- If CNTHPS_CTL_EL2.IMASK is 0, an interrupt is generated.

When CNTHPS_CTL_EL2.ENABLE is 0, the timer condition is not met, but CNTPCT_EL0 continues to count.

This field resets to an architecturally UNKNOWN value.

Accessing the CNTHPS_CVAL

This register is accessed using the encoding for CNTP_CVAL.

Accesses to this register use the following encodings:

```
MRRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <Rt2>, <CRm>
```

coproc	CRm	opc1
~~opc1~~	~~coproc~~	~~CRm~~
0b1111	0b1110	0b0010
~~0b0010~~	~~0b1111~~	~~0b1110~~

```
if PSTATE.EL == EL0 then
    if !ELUsingAArch32(EL1) && !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') && CNTKCTL_EL1.EL0PTEN
== '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x04);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x04);
    elsif ELUsingAArch32(EL1) && CNTKCTL.PL0PTEN == '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x04);
        elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TGE == '1' then
            AArch32.TakeHypTrapException(0x00);
        else
            UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '0' && CNTHCTL_EL2.EL1PCEN == '0'
then
        AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '10' && CNTHCTL_EL2.EL1PTEN
== '0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && CNTHCTL_EL2.EL0PTEN
== '0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && CNTHCTL.PL1PCEN == '0' then
        AArch32.TakeHypTrapException(0x04);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '0'
then
        return CNTHPS_CVAL_EL2;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '1'
then
        return CNTHP_CVAL_EL2;
    else
        return CNTP_CVAL;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '0' && CNTHCTL_EL2.EL1PCEN == '0'
then
        AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '1' && CNTHCTL_EL2.EL1PTEN == '0'
then
        AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && CNTHCTL.PL1PCEN == '0' then
        AArch32.TakeHypTrapException(0x04);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) then
        if SCR.NS == '0' then
            return CNTP_CVAL_S;
        else
            return CNTP_CVAL_NS;
    else
        return CNTP_CVAL;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && ELUsingAArch32(EL3) then
        return CNTP_CVAL_NS;
    else
        return CNTP_CVAL;
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        return CNTP_CVAL_S;
    else
        return CNTP_CVAL_NS;
```

```
MCRR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <Rt2>, <CRm>
```

coproc	CRm	opc1
~~opc1~~	~~coproc~~	~~CRm~~
0b1111	0b1110	0b0010
~~0b0010~~	~~0b1111~~	~~0b1110~~

```
if PSTATE.EL == EL0 then
    if !ELUsingAArch32(EL1) && !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') && CNTKCTL_EL1.EL0PTEN
== '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x04);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x04);
    elsif ELUsingAArch32(EL1) && CNTKCTL.PL0PTEN == '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x04);
        elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TGE == '1' then
            AArch32.TakeHypTrapException(0x00);
        else
            UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '0' && CNTHCTL_EL2.EL1PCEN == '0'
then
        AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '10' && CNTHCTL_EL2.EL1PTEN
== '0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && CNTHCTL_EL2.EL0PTEN
== '0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && CNTHCTL.PL1PCEN == '0' then
        AArch32.TakeHypTrapException(0x04);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '0'
then
        CNTHPS_CVAL_EL2 = R[t2]:R[t];
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '1'
then
        CNTHP_CVAL_EL2 = R[t2]:R[t];
    else
        CNTP_CVAL = R[t2]:R[t];
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '0' && CNTHCTL_EL2.EL1PCEN == '0'
then
        AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '1' && CNTHCTL_EL2.EL1PTEN == '0'
then
        AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && CNTHCTL.PL1PCEN == '0' then
        AArch32.TakeHypTrapException(0x04);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) then
        if SCR.NS == '0' then
            CNTP_CVAL_S = R[t2]:R[t];
        else
            CNTP_CVAL_NS = R[t2]:R[t];
    else
        CNTP_CVAL = R[t2]:R[t];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && ELUsingAArch32(EL3) then
        CNTP_CVAL_NS = R[t2]:R[t];
    else
        CNTP_CVAL = R[t2]:R[t];
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        CNTP_CVAL_S = R[t2]:R[t];
    else
        CNTP_CVAL_NS = R[t2]:R[t];
```

2713/0312/20192018 2116:5942; e5e4db499bf9867a4b93324c4dbac985d3da93766379d01e197f1d40720d32d0f84e419e9187e009

(old) htmldiff from- (new)

(old) htmldiff from- (new)

CNTHPS_TVAL, Counter-timer Secure Physical Timer TimerValue Register (EL2)

The CNTHPS_TVAL characteristics are:

Purpose

Provides AArch32 access to the timer value for the Secure EL2 physical timer.

Note

The Secure EL2 timer is implemented by ARMv8.4-SecEL2. It is only accessible from AArch32 state when EL2 is using AArch64 and the value of SCR_EL3.{EEL2, NS} is {1, 0}.

Configuration

AArch32 System register CNTHPS_TVAL bits [31:0] are architecturally mapped to AArch64 System register CNTHPS_TVAL_EL2[31:0] .

This register is present only when ARMv8.4-SecEL2 is implemented. Otherwise, direct accesses to CNTHPS_TVAL are UNDEFINED.

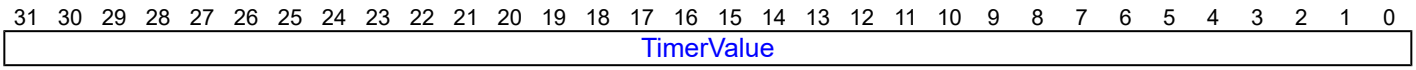
RW fields in this register reset to architecturally UNKNOWN values.

Attributes

CNTHPS_TVAL is a 32-bit register.

Field descriptions

The CNTHPS_TVAL bit assignments are:



TimerValue, bits [31:0]

The TimerValue view of the EL2 physical timer.

On a read of this register:

- If CNTHPS_CTL_EL2.ENABLE is 0, the value returned is UNKNOWN.
- If CNTHPS_CTL_EL2.ENABLE is 1, the value returned is (CNTHPS_CVAL_EL2 - CNTPCT_EL0).

On a write of this register, CNTHPS_CVAL_EL2 is set to (CNTPCT_EL0 + TimerValue), where TimerValue is treated as a signed 32-bit integer.

When CNTHPS_CTL_EL2.ENABLE is 1, the timer condition is met when (CNTPCT_EL0 - CNTHPS_CVAL_EL2) is greater than or equal to zero. This means that TimerValue acts like a 32-bit downcounter timer. When the timer condition is met:

- CNTHPS_CTL_EL2.ISTATUS is set to 1.
- If CNTHPS_CTL_EL2.IMASK is 0, an interrupt is generated.

When CNTHPS_CTL_EL2.ENABLE is 0, the timer condition is not met, but CNTPCT_EL0 continues to count, so the TimerValue view appears to continue to count down.

This field resets to an architecturally UNKNOWN value.

Accessing the CNTHPS_TVAL

This register is accessed using the encoding for CNTP_TVAL.

Accesses to this register use the following encodings:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
~~opc1~~	~~opc2~~	~~CRn~~	~~coproc~~	~~CRm~~
0b1111	0b000	0b1110	0b0010	0b000
~~0b000~~	~~0b000~~	~~0b1110~~	~~0b1111~~	~~0b0010~~

```
if PSTATE.EL == EL0 then
    if !ELUsingAArch32(EL1) && !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') && CNTKCTL_EL1.EL0PTEN
== '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x03);
    elsif ELUsingAArch32(EL1) && CNTKCTL.PL0PTEN == '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TGE == '1' then
            AArch32.TakeHypTrapException(0x00);
        else
            UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '0' && CNTHCTL_EL2.EL1PCEN == '0'
then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '10' && CNTHCTL_EL2.EL1PTEN
== '0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && CNTHCTL_EL2.EL0PTEN
== '0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && CNTHCTL.PL1PCEN == '0' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '0'
then
        return CNTHPS_TVAL_EL2;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '1'
then
        return CNTHP_TVAL_EL2;
    else
        return CNTP_TVAL;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '0' && CNTHCTL_EL2.EL1PCEN == '0'
then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '1' && CNTHCTL_EL2.EL1PTEN == '0'
then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && CNTHCTL.PL1PCEN == '0' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) then
        if SCR.NS == '0' then
            return CNTP_TVAL_S;
        else
            return CNTP_TVAL_NS;
    else
        return CNTP_TVAL;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && ELUsingAArch32(EL3) then
        return CNTP_TVAL_NS;
    else
        return CNTP_TVAL;
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        return CNTP_TVAL_S;
    else
        return CNTP_TVAL_NS;
```

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
~~opc1~~	~~opc2~~	~~CRn~~	~~coproc~~	~~CRm~~
0b1111	0b000	0b1110	0b0010	0b000
~~0b000~~	~~0b000~~	~~0b1110~~	~~0b1111~~	~~0b0010~~

```
if PSTATE.EL == EL0 then
    if !ELUsingAArch32(EL1) && !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') && CNTKCTL_EL1.EL0PTEN
== '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x03);
    elsif ELUsingAArch32(EL1) && CNTKCTL.PL0PTEN == '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TGE == '1' then
            AArch32.TakeHypTrapException(0x00);
        else
            UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '0' && CNTHCTL_EL2.EL1PCEN == '0'
then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '10' && CNTHCTL_EL2.EL1PTEN
== '0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && CNTHCTL_EL2.EL0PTEN
== '0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && CNTHCTL.PL1PCEN == '0' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '0'
then
        CNTHPS_TVAL_EL2 = R[t];
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '1'
then
        CNTHP_TVAL_EL2 = R[t];
    else
        CNTP_TVAL = R[t];
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '0' && CNTHCTL_EL2.EL1PCEN == '0'
then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '1' && CNTHCTL_EL2.EL1PTEN == '0'
then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && CNTHCTL.PL1PCEN == '0' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) then
        if SCR.NS == '0' then
            CNTP_TVAL_S = R[t];
        else
            CNTP_TVAL_NS = R[t];
    else
        CNTP_TVAL = R[t];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && ELUsingAArch32(EL3) then
        CNTP_TVAL_NS = R[t];
    else
        CNTP_TVAL = R[t];
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        CNTP_TVAL_S = R[t];
    else
        CNTP_TVAL_NS = R[t];
```

27~~13~~/03~~12~~/2019~~2018~~ 21~~16~~:59~~42~~: e5e4db499bf9867a4b93324c4dbac985d3da93766~~379d01e197f1d40720d32d0f84e419e9187e009~~

(old) htmldiff from- (new)

CNTHVS_CTL, Counter-timer Secure Virtual Timer Control Register (EL2)

The CNTHVS_CTL characteristics are:

Purpose

Provides AArch32 access to the Secure EL2 virtual timer.

Note

The Secure EL2 timer is implemented by ARMv8.4-SecEL2. It is only accessible from AArch32 state when EL2 is using AArch64 and the value of SCR_EL3.{EEL2, NS} is {1, 0}.

Configuration

AArch32 System register CNTHVS_CTL bits [31:0] are architecturally mapped to AArch64 System register CNTHVS_CTL_EL2[31:0] .

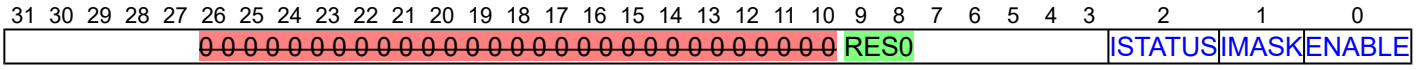
This register is present only when ARMv8.4-SecEL2 is implemented. Otherwise, direct accesses to CNTHVS_CTL are UNDEFINED.

Attributes

CNTHVS_CTL is a 32-bit register.

Field descriptions

The CNTHVS_CTL bit assignments are:



Bits [31:3]

Reserved, RES0.

ISTATUS, bit [2]

The status of the timer. This bit indicates whether the timer condition is met:

ISTATUS	Meaning
0b0	Timer condition is not met.
0b1	Timer condition is met.

When the value of the ENABLE bit is 1, ISTATUS indicates whether the timer condition is met. ISTATUS takes no account of the value of the IMASK bit. If the value of ISTATUS is 1 and the value of IMASK is 0 then the timer interrupt is asserted.

When the value of the ENABLE bit is 0, the ISTATUS field is UNKNOWN.

For more information see 'Operation of the CompareValue views of the timers' and 'Operation of the TimerValue views of the timers' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, chapter D8.

This bit is read-only.

IMASK, bit [1]

Timer interrupt mask bit. Permitted values are:

IMASK	Meaning
0b0	Timer interrupt is not masked by the IMASK bit.
0b1	Timer interrupt is masked by the IMASK bit.

For more information, see the description of the ISTATUS bit.

ENABLE, bit [0]

Enables the timer. Permitted values are:

ENABLE	Meaning
0b0	Timer disabled.
0b1	Timer enabled.

Setting this bit to 0 disables the timer output signal, but the timer value accessible from CNTHVS_TVAL continues to count down.

Note

Disabling the output signal might be a power-saving option.

Accessing the CNTHVS_CTL

This register is accessed using the encoding for CNTV_CTL.

Accesses to this register use the following encodings:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
~~opc1~~	~~opc2~~	~~CRn~~	~~coproc~~	~~CRm~~
0b1111	0b000	0b1110	0b0011	0b001
~~0b000~~	~~0b001~~	~~0b1110~~	~~0b1111~~	~~0b0011~~

```
if PSTATE.EL == EL0 then
    if !ELUsingAArch32(EL1) && !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') && CNTKCTL_EL1.EL0VTEN
== '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x03);
    elsif ELUsingAArch32(EL1) && CNTKCTL.PL0VTEN == '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TGE == '1' then
            AArch32.TakeHypTrapException(0x00);
        else
            UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && CNTHCTL_EL2.EL0VTEN
== '0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '0'
then
        return CNTHVS_CTL_EL2;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '1'
then
        return CNTHV_CTL_EL2;
    else
        return CNTV_CTL;
elsif PSTATE.EL == EL1 then
    return CNTV_CTL;
elsif PSTATE.EL == EL2 then
    return CNTV_CTL;
elsif PSTATE.EL == EL3 then
    return CNTV_CTL;
```

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
~~opc1~~	~~opc2~~	~~CRn~~	~~coproc~~	~~CRm~~
0b1111	0b000	0b1110	0b0011	0b001
~~0b000~~	~~0b001~~	~~0b1110~~	~~0b1111~~	~~0b0011~~

```
if PSTATE.EL == EL0 then
    if !ELUsingAArch32(EL1) && !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') && CNTKCTL_EL1.EL0VTEN
== '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x03);
    elsif ELUsingAArch32(EL1) && CNTKCTL.PL0VTEN == '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TGE == '1' then
            AArch32.TakeHypTrapException(0x00);
        else
            UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && CNTHCTL_EL2.EL0VTEN
== '0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '0'
then
        CNTHVS_CTL_EL2 = R[t];
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '1'
then
        CNTHV_CTL_EL2 = R[t];
    else
        CNTV_CTL = R[t];
elsif PSTATE.EL == EL1 then
    CNTV_CTL = R[t];
elsif PSTATE.EL == EL2 then
    CNTV_CTL = R[t];
elsif PSTATE.EL == EL3 then
    CNTV_CTL = R[t];
```

27~~13~~/03~~12~~/2019~~2018~~ 21~~16~~:59~~42~~; e5e4db499bf9867a4b93324c4dbac985d3da9376~~6379d01e197f1d40720d32d0f84e419e9187e009~~

(old) htmldiff from- (new)

(old) htmldiff from- (new)

CNTHVS_CVAL, Counter-timer Secure Virtual Timer CompareValue Register (EL2)

The CNTHVS_CVAL characteristics are:

Purpose

Provides AArch32 access to the compare value for the Secure EL2 virtual timer.

Note

The Secure EL2 timer is implemented by ARMv8.4-SecEL2. It is only accessible from AArch32 state when EL2 is using AArch64 and the value of SCR_EL3.{EEL2, NS} is {1, 0}.

Configuration

AArch32 System register CNTHVS_CVAL bits [63:0] are architecturally mapped to AArch64 System register CNTHVS_CVAL_EL2[63:0] .

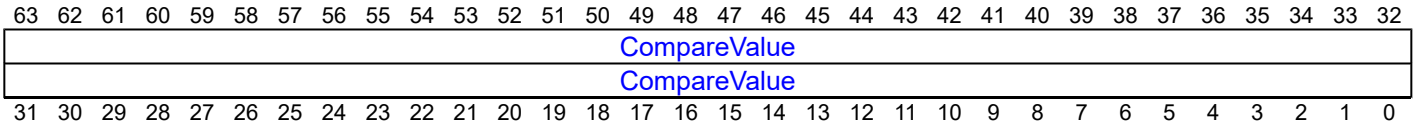
This register is present only when ARMv8.4-SecEL2 is implemented. Otherwise, direct accesses to CNTHVS_CVAL are UNDEFINED.

Attributes

CNTHVS_CVAL is a 64-bit register.

Field descriptions

The CNTHVS_CVAL bit assignments are:



CompareValue, bits [63:0]

Holds the EL2 virtual timer CompareValue.

When CNTHVS_CTL.ENABLE is 1, the timer condition is met when (CNTVCT - CompareValue) is greater than or equal to zero. This means that CompareValue acts like a 64-bit upcounter timer. When the timer condition is met:

- CNTHVS_CTL.ISTATUS is set to 1.
- If CNTHVS_CTL.IMASK is 0, an interrupt is generated.

When CNTHVS_CTL.ENABLE is 0, the timer condition is not met, but CNTVCT continues to count.

Accessing the CNTHVS_CVAL

This register is accessed using the encoding for CNTV_CVAL.

Accesses to this register use the following encodings:

```
MRRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <Rt2>, <CRm>
```

coproc	CRm	opc1
~~opc1~~	~~coproc~~	~~CRm~~
0b1111	0b1110	0b0011
~~0b0011~~	~~0b1111~~	~~0b1110~~

```
if PSTATE.EL == EL0 then
    if !ELUsingAArch32(EL1) && !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') && CNTKCTL_EL1.EL0VTEN
== '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x04);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x04);
    elsif ELUsingAArch32(EL1) && CNTKCTL.PL0VTEN == '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x04);
        elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TGE == '1' then
            AArch32.TakeHypTrapException(0x00);
        else
            UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && CNTHCTL_EL2.EL0VTEN
== '0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '0'
then
        return CNTHVS_CVAL_EL2;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '1'
then
        return CNTHV_CVAL_EL2;
    else
        return CNTV_CVAL;
elsif PSTATE.EL == EL1 then
    return CNTV_CVAL;
elsif PSTATE.EL == EL2 then
    return CNTV_CVAL;
elsif PSTATE.EL == EL3 then
    return CNTV_CVAL;
```

```
MCRR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <Rt2>, <CRm>
```

coproc	CRm	opc1
~~opc1~~	~~coproc~~	~~CRm~~
0b1111	0b1110	0b0011
~~0b0011~~	~~0b1111~~	~~0b1110~~

```
if PSTATE.EL == EL0 then
    if !ELUsingAArch32(EL1) && !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') && CNTKCTL_EL1.EL0VTEN
== '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x04);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x04);
    elsif ELUsingAArch32(EL1) && CNTKCTL.PL0VTEN == '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x04);
        elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TGE == '1' then
            AArch32.TakeHypTrapException(0x00);
        else
            UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && CNTHCTL_EL2.EL0VTEN
== '0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '0'
then
        CNTHVS_CVAL_EL2 = R[t2]:R[t];
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '1'
then
        CNTHV_CVAL_EL2 = R[t2]:R[t];
    else
        CNTV_CVAL = R[t2]:R[t];
elsif PSTATE.EL == EL1 then
    CNTV_CVAL = R[t2]:R[t];
elsif PSTATE.EL == EL2 then
    CNTV_CVAL = R[t2]:R[t];
elsif PSTATE.EL == EL3 then
    CNTV_CVAL = R[t2]:R[t];
```

27~~13~~/03~~12~~/2019~~2018~~ 21~~16~~:59~~42~~; e5e4db499bf9867a4b93324c4dbac985d3da93766~~379d01e197f1d40720d32d0f84e419e9187e009~~

(old) htmldiff from- (new)

(old) htmldiff from- (new)

CNTHVS_TVAL, Counter-timer Secure Virtual Timer TimerValue Register (EL2)

The CNTHVS_TVAL characteristics are:

Purpose

Provides AArch32 access to the timer value for the Secure EL2 virtual timer.

Note

The Secure EL2 timer is implemented by ARMv8.4-SecEL2. It is only accessible from AArch32 state when EL2 is using AArch64 and the value of SCR_EL3.{EEL2, NS} is {1, 0}.

Configuration

AArch32 System register CNTHVS_TVAL bits [31:0] are architecturally mapped to AArch64 System register CNTHVS_TVAL_EL2[31:0] .

This register is present only when ARMv8.4-SecEL2 is implemented. Otherwise, direct accesses to CNTHVS_TVAL are UNDEFINED.

Attributes

CNTHVS_TVAL is a 32-bit register.

Field descriptions

The CNTHVS_TVAL bit assignments are:

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
TimerValue																															

TimerValue, bits [31:0]

The TimerValue view of the EL2 virtual timer.

On a read of this register:

- If CNTHVS_CTL.ENABLE is 0, the value returned is UNKNOWN.
- If CNTHVS_CTL.ENABLE is 1, the value returned is (CNTHVS_CVAL - CNTVCT).

On a write of this register, CNTHVS_CVAL is set to (CNTVCT + TimerValue), where TimerValue is treated as a signed 32-bit integer.

When CNTHVS_CTL.ENABLE is 1, the timer condition is met when (CNTVCT - CNTHVS_CVAL) is greater than or equal to zero. This means that TimerValue acts like a 32-bit downcounter timer. When the timer condition is met:

- CNTHVS_CTL.ISTATUS is set to 1.
- If CNTHVS_CTL.IMASK is 0, an interrupt is generated.

When CNTHVS_CTL.ENABLE is 0, the timer condition is not met, but CNTVCT continues to count, so the TimerValue view appears to continue to count down.

Accessing the CNTHVS_TVAL

This register is accessed using the encoding for CNTV_TVAL.

Accesses to this register use the following encodings:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
~~opc1~~	~~opc2~~	~~CRn~~	~~coproc~~	~~CRm~~
0b1111	0b000	0b1110	0b0011	0b000
~~0b000~~	~~0b000~~	~~0b1110~~	~~0b1111~~	~~0b0011~~

```
if PSTATE.EL == EL0 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '0' then
        return CNTHVS_TVAL_EL2;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '1'
then
        return CNTHV_TVAL_EL2;
    else
        return CNTV_TVAL;
elsif PSTATE.EL == EL1 then
    return CNTV_TVAL;
elsif PSTATE.EL == EL2 then
    return CNTV_TVAL;
elsif PSTATE.EL == EL3 then
    return CNTV_TVAL;
```

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
~~opc1~~	~~opc2~~	~~CRn~~	~~coproc~~	~~CRm~~
0b1111	0b000	0b1110	0b0011	0b000
~~0b000~~	~~0b000~~	~~0b1110~~	~~0b1111~~	~~0b0011~~

```
if PSTATE.EL == EL0 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '0' then
        CNTHVS_TVAL_EL2 = R[t];
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '1'
then
        CNTHV_TVAL_EL2 = R[t];
    else
        CNTV_TVAL = R[t];
elsif PSTATE.EL == EL1 then
    CNTV_TVAL = R[t];
elsif PSTATE.EL == EL2 then
    CNTV_TVAL = R[t];
elsif PSTATE.EL == EL3 then
    CNTV_TVAL = R[t];
```

27~~13~~/03~~12~~/2019~~2018~~ 21~~16~~:59~~42~~; e5e4db499bf9867a4b93324c4dbac985d3da93766~~379d01e197f1d40720d32d0f84c419e9187e009~~

(old) htmldiff from- (new)

CNTP_CTL, Counter-timer Physical Timer Control register

The CNTP_CTL characteristics are:

Purpose

Control register for the EL1 physical timer.

Configuration

AArch32 System register CNTP_CTL bits [31:0] are architecturally mapped to AArch64 System register CNTP_CTL_EL0[31:0] .

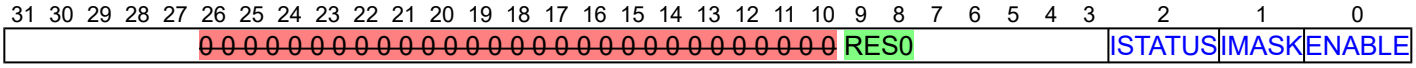
Some or all RW fields of this register have defined reset values. These apply only if the PE resets into an Exception level that is using AArch32. If the PE resets into EL3 using AArch32 they apply only to the Secure instance of the register. Otherwise, RW fields in this register reset to architecturally UNKNOWN values.

Attributes

CNTP_CTL is a 32-bit register.

Field descriptions

The CNTP_CTL bit assignments are:



Bits [31:3]

Reserved, RES0.

ISTATUS, bit [2]

The status of the timer. This bit indicates whether the timer condition is met:

ISTATUS	Meaning
0b0	Timer condition is not met.
0b1	Timer condition is met.

When the value of the ENABLE bit is 1, ISTATUS indicates whether the timer condition is met. ISTATUS takes no account of the value of the IMASK bit. If the value of ISTATUS is 1 and the value of IMASK is 0 then the timer interrupt is asserted.

When the value of the ENABLE bit is 0, the ISTATUS field is UNKNOWN.

For more information see 'Operation of the CompareValue views of the timers' and 'Operation of the TimerValue views of the timers' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, chapter D6.

This bit is read-only.

~~This field resets to an architecturally UNKNOWN value.~~

IMASK, bit [1]

Timer interrupt mask bit. Permitted values are:

IMASK	Meaning
0b0	Timer interrupt is not masked by the IMASK bit.
0b1	Timer interrupt is masked by the IMASK bit.

For more information, see the description of the ISTATUS bit.

This field resets to an architecturally UNKNOWN value.

ENABLE, bit [0]

Enables the timer. Permitted values are:

ENABLE	Meaning
0b0	Timer disabled.
0b1	Timer enabled.

Setting this bit to 0 disables the timer output signal, but the timer value accessible from CNTP_TVAL continues to count down.

Note

Disabling the output signal might be a power-saving option.

This field resets to 0.

Accessing the CNTP_CTL

Accesses to this register use the following encodings:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
~~opc1~~	~~opc2~~	~~CRn~~	~~coproc~~	~~CRm~~
0b1111	0b000	0b1110	0b0010	0b001
~~0b000~~	~~0b001~~	~~0b1110~~	~~0b1111~~	~~0b0010~~

```
if PSTATE.EL == EL0 then
    if !ELUsingAArch32(EL1) && !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') && CNTKCTL_EL1.EL0PTEN
== '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x03);
    elsif ELUsingAArch32(EL1) && CNTKCTL.PL0PTEN == '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TGE == '1' then
            AArch32.TakeHypTrapException(0x00);
        else
            UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '0' && CNTHCTL_EL2.EL1PCEN == '0'
then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '10' && CNTHCTL_EL2.EL1PTEN
== '0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && CNTHCTL_EL2.EL0PTEN
== '0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && CNTHCTL.PL1PCEN == '0' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '0'
then
        return CNTHPS_CTL_EL2;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '1'
then
        return CNTHP_CTL_EL2;
    else
        return CNTP_CTL;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '0' && CNTHCTL_EL2.EL1PCEN == '0'
then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '1' && CNTHCTL_EL2.EL1PTEN == '0'
then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && CNTHCTL.PL1PCEN == '0' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) then
        if SCR.NS == '0' then
            return CNTP_CTL_S;
        else
            return CNTP_CTL_NS;
    else
        return CNTP_CTL;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && ELUsingAArch32(EL3) then
        return CNTP_CTL_NS;
    else
        return CNTP_CTL;
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        return CNTP_CTL_S;
    else
        return CNTP_CTL_NS;
```

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
~~opc1~~	~~opc2~~	~~CRn~~	~~coproc~~	~~CRm~~
0b1111	0b000	0b1110	0b0010	0b001
~~0b000~~	~~0b001~~	~~0b1110~~	~~0b1111~~	~~0b0010~~

```
if PSTATE.EL == EL0 then
    if !ELUsingAArch32(EL1) && !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') && CNTKCTL_EL1.EL0PTEN
== '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x03);
    elsif ELUsingAArch32(EL1) && CNTKCTL.PL0PTEN == '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TGE == '1' then
            AArch32.TakeHypTrapException(0x00);
        else
            UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '0' && CNTHCTL_EL2.EL1PCEN == '0'
then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '10' && CNTHCTL_EL2.EL1PTEN
== '0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && CNTHCTL_EL2.EL0PTEN
== '0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && CNTHCTL.PL1PCEN == '0' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '0'
then
        CNTHPS_CTL_EL2 = R[t];
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '1'
then
        CNTHP_CTL_EL2 = R[t];
    else
        CNTP_CTL = R[t];
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '0' && CNTHCTL_EL2.EL1PCEN == '0'
then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '1' && CNTHCTL_EL2.EL1PTEN == '0'
then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && CNTHCTL.PL1PCEN == '0' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) then
        if SCR.NS == '0' then
            CNTP_CTL_S = R[t];
        else
            CNTP_CTL_NS = R[t];
    else
        CNTP_CTL = R[t];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && ELUsingAArch32(EL3) then
        CNTP_CTL_NS = R[t];
    else
        CNTP_CTL = R[t];
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        CNTP_CTL_S = R[t];
    else
        CNTP_CTL_NS = R[t];
```

2713/0312/20192018 2116:5942: e5e4db499bf9867a4b93324c4dbac985d3da93766379d01e197f1d40720d32d0f84e419e9187e009

(old) htmldiff from- (new)

CNTV_CTL, Counter-timer Virtual Timer Control register

The CNTV_CTL characteristics are:

Purpose

Control register for the virtual timer.

Configuration

AArch32 System register CNTV_CTL bits [31:0] are architecturally mapped to AArch64 System register CNTV_CTL_EL0[31:0] .

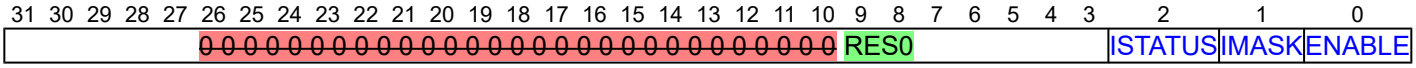
Some or all RW fields of this register have defined reset values. These apply only if the PE resets into an Exception level that is using AArch32. Otherwise, RW fields in this register reset to architecturally UNKNOWN values.

Attributes

CNTV_CTL is a 32-bit register.

Field descriptions

The CNTV_CTL bit assignments are:



Bits [31:3]

Reserved, RES0.

ISTATUS, bit [2]

The status of the timer. This bit indicates whether the timer condition is met:

ISTATUS	Meaning
0b0	Timer condition is not met.
0b1	Timer condition is met.

When the value of the ENABLE bit is 1, ISTATUS indicates whether the timer condition is met. ISTATUS takes no account of the value of the IMASK bit. If the value of ISTATUS is 1 and the value of IMASK is 0 then the timer interrupt is asserted.

When the value of the ENABLE bit is 0, the ISTATUS field is UNKNOWN.

For more information see 'Operation of the CompareValue views of the timers' and 'Operation of the TimerValue views of the timers' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, chapter D6.

This bit is read-only.

~~This field resets to an architecturally UNKNOWN value.~~

IMASK, bit [1]

Timer interrupt mask bit. Permitted values are:

IMASK	Meaning
0b0	Timer interrupt is not masked by the IMASK bit.
0b1	Timer interrupt is masked by the IMASK bit.

For more information, see the description of the ISTATUS bit.

This field resets to an architecturally UNKNOWN value.

ENABLE, bit [0]

Enables the timer. Permitted values are:

ENABLE	Meaning
0b0	Timer disabled.
0b1	Timer enabled.

Setting this bit to 0 disables the timer output signal, but the timer value accessible from CNTV_TVAL continues to count down.

Note

Disabling the output signal might be a power-saving option.

This field resets to 0.

Accessing the CNTV_CTL

Accesses to this register use the following encodings:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
~~opc1~~	~~opc2~~	~~CRn~~	~~coproc~~	~~CRm~~
0b1111	0b000	0b1110	0b0011	0b001
~~0b000~~	~~0b001~~	~~0b1110~~	~~0b1111~~	~~0b0011~~

```
if PSTATE.EL == EL0 then
    if !ELUsingAArch32(EL1) && !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') && CNTKCTL_EL1.EL0VTEN
== '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x03);
    elsif ELUsingAArch32(EL1) && CNTKCTL.PL0VTEN == '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TGE == '1' then
            AArch32.TakeHypTrapException(0x00);
        else
            UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && CNTHCTL_EL2.EL0VTEN
== '0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '0'
then
        return CNTHVS_CTL_EL2;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '1'
then
        return CNTHV_CTL_EL2;
    else
        return CNTV_CTL;
elsif PSTATE.EL == EL1 then
    return CNTV_CTL;
elsif PSTATE.EL == EL2 then
    return CNTV_CTL;
elsif PSTATE.EL == EL3 then
    return CNTV_CTL;
```

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
~~opc1~~	~~opc2~~	~~CRn~~	~~coproc~~	~~CRm~~
0b1111	0b000	0b1110	0b0011	0b001
~~0b000~~	~~0b001~~	~~0b1110~~	~~0b1111~~	~~0b0011~~

```
if PSTATE.EL == EL0 then
    if !ELUsingAArch32(EL1) && !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') && CNTKCTL_EL1.EL0VTEN
== '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x03);
    elsif ELUsingAArch32(EL1) && CNTKCTL.PL0VTEN == '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TGE == '1' then
            AArch32.TakeHypTrapException(0x00);
        else
            UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && CNTHCTL_EL2.EL0VTEN
== '0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '0'
then
        CNTHVS_CTL_EL2 = R[t];
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '1'
then
        CNTHV_CTL_EL2 = R[t];
    else
        CNTV_CTL = R[t];
elsif PSTATE.EL == EL1 then
    CNTV_CTL = R[t];
elsif PSTATE.EL == EL2 then
    CNTV_CTL = R[t];
elsif PSTATE.EL == EL3 then
    CNTV_CTL = R[t];
```

27~~13~~/03~~12~~/2019~~2018~~ 21~~16~~:59~~42~~; e5e4db499bf9867a4b93324c4dbac985d3da9376~~6379d01e197f1d40720d32d0f84e419e9187e009~~

(old) htmldiff from- (new)

CP15DMB, Data Memory Barrier System instruction

The CP15DMB characteristics are:

Purpose

Performs a Data Memory Barrier.

Arm deprecates any use of this System instruction~~operation~~, and strongly recommends that software use the DMB instruction instead.

Configuration

Attributes

CP15DMB is a 32-bit System instruction.

Field descriptions

CP15DMB ignores the value in the register specified by the instruction encoding. Software does not have to write a value to the register before issuing this instruction.

Executing the CP15DMB instruction

Accesses to this instruction use the following encodings:

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
~~opc1~~	~~opc2~~	~~CRn~~	~~coproc~~	~~CRm~~
0b1111	0b000	0b0111	0b1010	0b101
~~0b000~~	~~0b101~~	~~0b0111~~	~~0b1111~~	~~0b1010~~

```
if PSTATE.EL == EL0 then
    if !ELUsingAArch32(EL1) && !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') && SCTLR_EL1.CP15BEN
== '0' then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCTLR_EL2.CP15BEN
== '0' then
        UNDEFINED;
    elsif ELUsingAArch32(EL1) && SCTLR.CP15BEN == '0' then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> != '11' && HSTR_EL2.T7 == '1'
then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T7 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        CP15DMB();
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T7 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T7 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif ELUsingAArch32(EL1) && SCTLR.CP15BEN == '0' then
        UNDEFINED;
    else
        CP15DMB();
elsif PSTATE.EL == EL2 then
    if HSCTLR.CP15BEN == '0' then
        UNDEFINED;
    else
        CP15DMB();
elsif PSTATE.EL == EL3 then
    CP15DMB();
```

27~~13~~/03~~12~~/2019~~2018~~ 21~~16~~:59~~42~~; e5e4db499bf9867a4b93324c4dbac985d3da93766~~379d01e197f1d40720d32d0f84e419e9187e009~~

(old) htmldiff from- (new)

(old) htmldiff from- (new)

CP15DSB, Data Synchronization Barrier System instruction

The CP15DSB characteristics are:

Purpose

Performs a Data Synchronization Barrier.

Arm deprecates any use of this System instruction~~operation~~, and strongly recommends that software use the DSB instruction instead.

Configuration

Attributes

CP15DSB is a 32-bit System instruction.

Field descriptions

CP15DSB ignores the value in the register specified by the instruction encoding. Software does not have to write a value to the register before issuing this instruction.

Executing the CP15DSB instruction

Accesses to this instruction use the following encodings:

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
~~opc1~~	~~opc2~~	~~CRn~~	~~coproc~~	~~CRm~~
0b1111	0b000	0b0111	0b1010	0b100
~~0b000~~	~~0b100~~	~~0b0111~~	~~0b1111~~	~~0b1010~~

```
if PSTATE.EL == EL0 then
    if !ELUsingAArch32(EL1) && !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') && SCTLR_EL1.CP15BEN
== '0' then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCTLR_EL2.CP15BEN
== '0' then
        UNDEFINED;
    elsif ELUsingAArch32(EL1) && SCTLR.CP15BEN == '0' then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> != '11' && HSTR_EL2.T7 == '1'
then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T7 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        CP15DSB();
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T7 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T7 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif ELUsingAArch32(EL1) && SCTLR.CP15BEN == '0' then
        UNDEFINED;
    else
        CP15DSB();
elsif PSTATE.EL == EL2 then
    if HSCTLR.CP15BEN == '0' then
        UNDEFINED;
    else
        CP15DSB();
elsif PSTATE.EL == EL3 then
    CP15DSB();
```

27~~13~~/03~~12~~/2019~~2018~~ 21~~16~~:59~~42~~; e5e4db499bf9867a4b93324c4dbac985d3da93766~~379d01e197f1d40720d32d0f84e419e9187e009~~

(old) htmldiff from- (new)

CP15ISB, Instruction Synchronization Barrier System instruction

The CP15ISB characteristics are:

Purpose

Performs an Instruction Synchronization Barrier.

Arm deprecates any use of this System instruction~~operation~~, and strongly recommends that software use the ISB instruction instead.

Configuration

Attributes

CP15ISB is a 32-bit System instruction.

Field descriptions

CP15ISB ignores the value in the register specified by the instruction encoding. Software does not have to write a value to the register before issuing this instruction.

Executing the CP15ISB instruction

Accesses to this instruction use the following encodings:

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
~~opc1~~	~~opc2~~	~~CRn~~	~~coproc~~	~~CRm~~
0b1111	0b000	0b0111	0b0101	0b100
~~0b000~~	~~0b100~~	~~0b0111~~	~~0b1111~~	~~0b0101~~

```
if PSTATE.EL == EL0 then
    if !ELUsingAArch32(EL1) && !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') && SCTLR_EL1.CP15BEN
== '0' then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCTLR_EL2.CP15BEN
== '0' then
        UNDEFINED;
    elsif ELUsingAArch32(EL1) && SCTLR.CP15BEN == '0' then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> != '11' && HSTR_EL2.T7 == '1'
then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T7 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        CP15ISB();
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T7 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T7 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif ELUsingAArch32(EL1) && SCTLR.CP15BEN == '0' then
        UNDEFINED;
    else
        CP15ISB();
elsif PSTATE.EL == EL2 then
    if HSCTLR.CP15BEN == '0' then
        UNDEFINED;
    else
        CP15ISB();
elsif PSTATE.EL == EL3 then
    CP15ISB();
```

27~~13~~/03~~12~~/2019~~2018~~ 21~~16~~:59~~42~~; e5e4db499bf9867a4b93324c4dbac985d3da93766~~379d01e197f1d40720d32d0f84e419e9187e009~~

(old) htmldiff from- (new)

(old) htmldiff from- (new)

DACR, Domain Access Control Register

The DACR characteristics are:

Purpose

Defines the access permission for each of the sixteen memory domains.

Configuration

AArch32 System register DACR bits [31:0] are architecturally mapped to AArch64 System register DACR32_EL2[31:0] .

~~When EL3 is using AArch32, write access to DACR(S) is disabled when the CP15SDISABLE signal is asserted HIGH.~~

This register has no function when TTBCR.EAE is set to 1, to select the Long-descriptor translation table format.

RW fields in this register reset to architecturally UNKNOWN values.

Attributes

DACR is a 32-bit register.

Field descriptions

The DACR bit assignments are:

31 30	29 28	27 26	25 24	23 22	21 20	19 18	17 16	15 14	13 12	11 10	9 8	7 6	5 4	3 2	1 0
D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0

D<n>, bits [2n+1:2n], for n = 0 to 15

Domain n access permission, where n = 0 to 15. Permitted values are:

D<n>	Meaning
0b00	No access. Any access to the domain generates a Domain fault.
0b01	Client. Accesses are checked against the permission bits in the translation tables.
0b11	Manager. Accesses are not checked against the permission bits in the translation tables.

The value 0b10 is reserved.

This field resets to an architecturally UNKNOWN value.

Accessing the DACR

Accesses to this register use the following encodings:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
~~opc1~~	~~opc2~~	~~CRn~~	~~coproc~~	~~CRm~~
0b1111	0b000	0b0011	0b0000	0b000
~~0b000~~	~~0b000~~	~~0b0011~~	~~0b1111~~	~~0b0000~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T3 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T3 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TRVM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TRVM == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) then
        if SCR.NS == '0' then
            return DACR_S;
        else
            return DACR_NS;
    else
        return DACR;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && ELUsingAArch32(EL3) then
        return DACR_NS;
    else
        return DACR;
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        return DACR_S;
    else
        return DACR_NS;
```

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
~~opc1~~	~~opc2~~	~~CRn~~	~~coproc~~	~~CRm~~
0b1111	0b000	0b0011	0b0000	0b000
~~0b000~~	~~0b000~~	~~0b0011~~	~~0b1111~~	~~0b0000~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T3 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T3 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) && SCR.NS == '0' && CP15SDISABLE == HIGH then
        UNDEFINED;
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) && SCR.NS == '0' && CP15SDISABLE2 == HIGH then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TVM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TVM == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) then
        if SCR.NS == '0' then
            DACR_S = R[t];
        else
            DACR_NS = R[t];
    else
        DACR = R[t];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && ELUsingAArch32(EL3) then
        DACR_NS = R[t];
    else
        DACR = R[t];
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' && CP15SDISABLE == HIGH then
        UNDEFINED;
    elsif SCR.NS == '0' && CP15SDISABLE2 == HIGH then
        UNDEFINED;
    else
        if SCR.NS == '0' then
            DACR_S = R[t];
        else
            DACR_NS = R[t];
```

2713/0312/20192018 2116:5942; e5e4db499bf9867a4b93324c4dbac985d3da93766379d01e197f1d40720d32d0f84e419e9187e009

(old) htmldiff from- (new)

DBGBCR<n>, Debug Breakpoint Control Registers, n = 0 - 15

The DBGBCR<n> characteristics are:

Purpose

Holds control information for a breakpoint. Forms breakpoint n together with value register DBGBVR<n>. If EL2 is implemented and this breakpoint supports Context matching, DBGBVR<n> can be associated with a Breakpoint Extended Value Register DBGBXVR<n> for VMID matching.

Configuration

AArch32 System register DBGBCR<n> bits [31:0] are architecturally mapped to AArch64 System register DBGBCR<n>_EL1[31:0] .

AArch32 System register DBGBCR<n> bits [31:0] are architecturally mapped to External register DBGBCR<n>_EL1[31:0] .

If breakpoint n is not implemented then this register is unallocated.

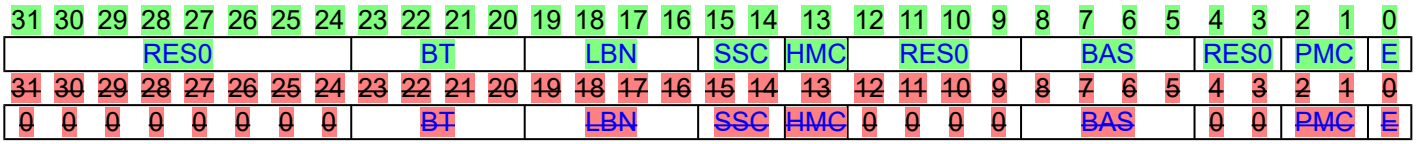
This register is in the Cold reset domain. On a Cold reset RW fields in this register reset to architecturally UNKNOWN values. The register is not affected by a Warm reset.

Attributes

DBGBCR<n> is a 32-bit register.

Field descriptions

The DBGBCR<n> bit assignments are:



When the E field is zero, all the other fields in the register are ignored.

Bits [31:24]

Reserved, RES0.

BT, bits [23:20]

Breakpoint Type. Possible values are:

BT	Meaning
0b0000	Unlinked instruction address match. DBGBVR<n> is the address of an instruction.
0b0001	As 0b0000 with linking enabled.
0b0010	Unlinked Context ID match. When ARMv8.1-VHE is implemented, EL2 is using AArch64, and the Effective value of HCR_EL2.E2H is 1, if either the PE is executing at EL0 with HCR_EL2.TGE set to 0 or the PE is executing at EL2, then DBGBVR<n>.ContextID must match the CONTEXTIDR_EL2 value. Otherwise DBGBVR<n>.ContextID must match the CONTEXTIDR value.
0b0011	As 0b0010 with linking enabled.
0b0100	Unlinked instruction address mismatch. DBGBVR<n> is the address of an instruction to be stepped.
0b0101	As 0b0100 with linking enabled.
0b0110	Unlinked CONTEXTIDR_EL1 match. DBGBVR<n>.ContextID is a Context ID compared against CONTEXTIDR.
0b0111	As 0b0110 with linking enabled.
0b1000	Unlinked VMID match. DBGBXVR<n>.VMID is a VMID compared against VTTBR.VMID.
0b1001	As 0b1000 with linking enabled.
0b1010	Unlinked VMID and Context ID match. DBGBVR<n>.ContextID is a Context ID compared against CONTEXTIDR, and DBGBXVR<n>.VMID is a VMID compared against VTTBR.VMID.
0b1011	As 0b1010 with linking enabled.
0b1100	Unlinked CONTEXTIDR_EL2 match. DBGBXVR<n>.ContextID2 is a Context ID compared against CONTEXTIDR_EL2.
0b1101	As 0b1100 with linking enabled.
0b1110	Unlinked Full Context ID match. DBGBVR<n>.ContextID is compared against CONTEXTIDR, and DBGBXVR<n>.ContextID2 is compared against CONTEXTIDR_EL2.
0b1111	As 0b1110 with linking enabled.

For more information on Breakpoints and their constraints, see 'Breakpoint exceptions' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section G2.9 and 'Reserved DBGBCR<n>.BT values' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section G2 (AArch32 Self-hosted Debug).

On a Cold reset, this field resets to an architecturally UNKNOWN value.

LBN, bits [19:16]

Linked breakpoint number. For Linked address matching breakpoints, this specifies the index of the Context-matching breakpoint linked to.

For all other breakpoint types this field is ignored and reads of the register return an UNKNOWN value.

This field is ignored when the value of DBGBCR<n>.E is 0.

On a Cold reset, this field resets to an architecturally UNKNOWN value.

SSC, bits [15:14]

Security state control. Determines the Security states under which a Breakpoint debug event for breakpoint n is generated. This field must be interpreted along with the HMC and PMC fields, and there are constraints on the permitted values of the {HMC, SSC, PMC} fields.

For more information, see 'Execution conditions for which a breakpoint generates Breakpoint exceptions' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile~~, section G2 (AArch32 Self-hosted Debug)~~, and 'Reserved DBGBCR<n>.{SSC~~HMC~~, HMC~~SSC~~, PMC} values' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile~~, section G2 (AArch32 Self-hosted Debug)~~.

On a Cold reset, this field resets to an architecturally UNKNOWN value.

HMC, bit [13]

Higher mode control. Determines the debug perspective for deciding when a Breakpoint debug event for breakpoint n is generated. This field must be interpreted along with the SSC and PMC fields, and there are constraints on the permitted values of the {HMC, SSC, PMC} fields. For more information see the SSC, bits [15:14] description.

For more information on the operation of the SSC, HMC, and PMC fields, see 'Execution conditions for which a breakpoint generates Breakpoint exceptions' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section G2 (AArch32 Self-hosted Debug).

On a Cold reset, this field resets to an architecturally UNKNOWN value.

Bits [12:9]

Reserved, RES0.

BAS, bits [8:5]

Byte address select. Defines which half-words an address-matching breakpoint matches, regardless of the instruction set and Execution state.

The permitted values depend on the breakpoint type.

For Address match breakpoints, the permitted values are:

BAS	Match instruction at	Constraint for debuggers
0b0011	DBGBVR<n>	Use for T32 instructions
0b1100	DBGBVR<n>+2	Use for T32 instructions
0b1111	DBGBVR<n>	Use for A32 instructions

All other values are reserved. For more information, see 'Reserved DBGBCR<n>.BAS values' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section G2 (AArch32 Self-hosted Debug).

For more information on using the BAS field in Address Match breakpoints, see 'Using the BAS field in Address Match breakpoints' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section G2 (AArch32 Self-hosted Debug).

For Address mismatch breakpoints in an AArch32 stage 1 translation regime, the permitted values are:

BAS	Step instruction at	Constraint for debuggers
0b0000	-	Use for a match anywhere breakpoint
0b0011	DBGBVR<n>	Use for T32 instructions
0b1100	DBGBVR<n>+2	Use for T32 instructions
0b1111	DBGBVR<n>	Use for A32 instructions

All other values are reserved. For more information, see 'Reserved DBGBCR<n>.BAS values' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section G2 (AArch32 Self-hosted Debug).

For more information on using the BAS field in address mismatch breakpoints, see 'Using the BAS field in Address Match breakpoints' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section G2 (AArch32 Self-hosted Debug).

For Context matching breakpoints, this field is RES1 and ignored.

On a Cold reset, this field resets to an architecturally UNKNOWN value.

Bits [4:3]

Reserved, RES0.

PMC, bits [2:1]

Privilege mode control. Determines the Exception level or levels at which a Breakpoint debug event for breakpoint n is generated. This field must be interpreted along with the SSC and HMC fields, and there are constraints on the permitted values of the {HMC, SSC, PMC} fields. For more information see the DBGBCR<n>.SSC description.

For more information on the operation of the SSC, HMC, and PMC fields, see 'Execution conditions for which a breakpoint generates Breakpoint exceptions' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section G2 (AArch32 Self-hosted Debug).

On a Cold reset, this field resets to an architecturally UNKNOWN value.

E, bit [0]

Enable breakpoint DBGBVR<n>. Possible values are:

E	Meaning
0b0	Breakpoint disabled.
0b1	Breakpoint enabled.

On a Cold reset, this field resets to an architecturally UNKNOWN value.

Accessing the DBGBCR<n>

Accesses to this register use the following encodings:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
~~opc1~~	~~opc2~~	~~CRn~~	~~coproc~~	~~CRm~~
0b1110	0b000	0b0000	0bnnnn	0b101
~~0b000~~	~~0b101~~	~~0b0000~~	~~0b1110~~	~~0bnnnn~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.<TDE,TDA> != '00' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x05);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.<TDE,TDA> != '00' then
        AArch32.TakeHypTrapException(0x05);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x05);
    elsif ELUsingAArch32(EL1) && DBGOSLSR.OSLK == '0' && HaltingAllowed() && EDSCR.TDA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        return DBGBCR[UInt(CRm<3:0>)];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x05);
    elsif ELUsingAArch32(EL1) && DBGOSLSR.OSLK == '0' && HaltingAllowed() && EDSCR.TDA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        return DBGBCR[UInt(CRm<3:0>)];
elsif PSTATE.EL == EL3 then
    if ELUsingAArch32(EL1) && DBGOSLSR.OSLK == '0' && HaltingAllowed() && EDSCR.TDA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        return DBGBCR[UInt(CRm<3:0>)];
```

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
~~opc1~~	~~opc2~~	~~CRn~~	~~coproc~~	~~CRm~~
0b1110	0b000	0b0000	0bnnnn	0b101
~~0b000~~	~~0b101~~	~~0b0000~~	~~0b1110~~	~~0bnnnn~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.<TDE,TDA> != '00' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x05);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.<TDE,TDA> != '00' then
        AArch32.TakeHypTrapException(0x05);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x05);
    elsif ELUsingAArch32(EL1) && DBGOSLSR.OSLK == '0' && HaltingAllowed() && EDSCR.TDA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        DBGBCR[UInt(CRm<3:0>)] = R[t];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x05);
    elsif ELUsingAArch32(EL1) && DBGOSLSR.OSLK == '0' && HaltingAllowed() && EDSCR.TDA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        DBGBCR[UInt(CRm<3:0>)] = R[t];
elsif PSTATE.EL == EL3 then
    if ELUsingAArch32(EL1) && DBGOSLSR.OSLK == '0' && HaltingAllowed() && EDSCR.TDA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        DBGBCR[UInt(CRm<3:0>)] = R[t];
```

(old) htmldiff from- (new)

DBGDSCRext, Debug Status and Control Register, External View

The DBGDSCRext characteristics are:

Purpose

Main control register for the debug implementation.

Configuration

AArch32 System register DBGDSCRext bits [31:0] are architecturally mapped to AArch64 System register MDSCR_EL1[31:0] .

AArch32 System register DBGDSCRext bits [15:2] are architecturally mapped to AArch32 System register DBGDSCRint[15:2] .

This register is required in all implementations.

This register is in the Warm reset domain. Some or all RW fields of this register have defined reset values. On a Warm or Cold reset these apply only if the PE resets into an Exception level that is using AArch32. Otherwise, on a Warm or Cold reset RW fields in this register reset to architecturally UNKNOWN values.

Attributes

DBGDSCRext is a 32-bit register.

Field descriptions

The DBGDSCRext bit assignments are:

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
TFO	RXfull	TXfull	RES0	RXO	TXU	RES0		INTdis		TDA	RES0		SC2	NS	SPNIDdis		SPIDdis		MDBGen	HDE	RES0		UDCCdis		RES0					ERR	MOE	
~~31~~	~~30~~	~~29~~	~~28~~	~~27~~	~~26~~	~~25~~	~~24~~	~~23~~	~~22~~	~~21~~	~~20~~	~~19~~	~~18~~	~~17~~	~~16~~	~~15~~	~~14~~	~~13~~	~~12~~	~~11~~	~~10~~	~~9~~	~~8~~	~~7~~	~~6~~	~~5~~	~~4~~	~~3~~	~~2~~	~~1~~	~~0~~
~~TFO~~	~~RXfull~~	~~TXfull~~	~~0~~	~~RXO~~	~~TXU~~	~~0~~	~~0~~	~~INTdis~~		~~TDA~~	~~0~~		~~SC2~~	~~NS~~	~~SPNIDdis~~		~~SPIDdis~~		~~MDBGen~~	~~HDE~~	~~0~~		~~UDCCdis~~		~~0~~	~~0~~	~~0~~	~~0~~	~~0~~	~~ERR~~	~~MOE~~	~~0~~

TFO, bit [31]

When ARMv8.4-Trace is implemented:

Trace Filter override. Used for save/restore of EDSCR.TFO.

When the OS Lock is unlocked, OSLSR_EL1.OSLK == 0, this bit ignores writes and software must treat it as UNK/SBZP.

When the OS Lock is locked, OSLSR_EL1.OSLK == 1, this bit is RW and holds the value of EDSCR.TFO.

Reads and writes of this bit are indirect accesses to EDSCR.TFO.

Otherwise:

Reserved, RES0.

RXfull, bit [30]

DTRRX full. Used for save/restore of EDSCR.RXfull.

When OSLSR_EL1.OSLK == 0, this bit is RO, and software must treat it as UNK/SBZP.

When OSLSR_EL1.OSLK == 1, this bit is RW and holds the value of EDSCR.RXfull.

Arm deprecates use of this bit other than for save/restore. Use DBGDSCRint to access the DTRRX full status.

Reads and writes of this bit are indirect accesses to EDSCR.RXfull.

The architected behavior of this field determines the value it returns after a reset.

TXfull, bit [29]

DTRTX full. Used for save/restore of EDSCR.TXfull.

When OSLSR_EL1.OSLK == 0, this bit is RO, and software must treat it as UNK/SBZP.

When DBGOSLSR.OSLK == 1, this bit is RW and holds the value of EDSCR.TXfull.

Arm deprecates use of this bit other than for save/restore. Use DBGDSCRint to access the DTRTX full status.

Reads and writes of this bit are indirect accesses to EDSCR.TXfull.

The architected behavior of this field determines the value it returns after a reset.

Bit [28]

Reserved, RES0.

RXO, bit [27]

Used for save/restore of EDSCR.RXO.

When OSLSR_EL1.OSLK == 0, this bit is RO, and software must treat it as UNK/SBZP.

When DBGOSLSR.OSLK == 1, this bit is RW and holds the value of EDSCR.RXO.

Reads and writes of this bit are indirect accesses to EDSCR.RXO.

The architected behavior of this field determines the value it returns after a reset.

TXU, bit [26]

Used for save/restore of EDSCR.TXU.

When OSLSR_EL1.OSLK == 0, this bit is RO, and software must treat it as UNK/SBZP.

When DBGOSLSR.OSLK == 1, this bit is RW and holds the value of EDSCR.TXU.

Reads and writes of this bit are indirect accesses to EDSCR.TXU.

The architected behavior of this field determines the value it returns after a reset.

Bits [25:24]

Reserved, RES0.

INTdis, bits [23:22]

Used for save/restore of EDSCR.INTdis.

When OSLSR_EL1.OSLK == 0, this field is RO, and software must treat it as UNK/SBZP.

When DBGOSLSR.OSLK == 1, this field is RW and holds the value of EDSCR.INTdis.

Reads and writes of this field are indirect accesses to EDSCR.INTdis.

The architected behavior of this field determines the value it returns after a reset.

TDA, bit [21]

Used for save/restore of EDSCR.TDA.

When OSLSR_EL1.OSLK == 0, this bit is RO, and software must treat it as UNK/SBZP.

When DBGOSLSR.OSLK == 1, this bit is RW and holds the value of EDSCR.TDA.

Reads and writes of this bit are indirect accesses to EDSCR.TDA.

The architected behavior of this field determines the value it returns after a reset.

Bit [20]

Reserved, RES0.

SC2, bit [19]

When ARMv8.0-PCSample is implemented, ARMv8.1-VHE is implemented and ARMv8.2-PCSampleARMv8.1-VHE is not implemented:

UsedIf ARMv8.2-PCSample is not implemented, used for save/restore of EDSCR.SC2.

When DBGOSLSR.OSLK == 0, this bit is RO, and software must treat it as UNK/SBZP.

When DBGOSLSR.OSLK == 1, this bit is RW and holds the value of EDSCR.SC2.

Reads and writes of this bit are indirect accesses to EDSCR.SC2.

Otherwise:

Reserved, RES0.

NS, bit [18]

Non-secure status. Returns the inverse of IsSecure(). This bit is RO.

Arm deprecates use of this field.

SPNIDdis, bit [17]

Secure privileged profiling disabled status bit. This bit is RO. Permitted values are:

SPNIDdis	Meaning
0b0	If EL3 is implemented, profiling allowed in Secure privileged modes.
0b1	If EL3 is implemented, profiling prohibited in Secure privileged modes.

This field is RES0 if EL3 is not implemented.

- Otherwise, the field reads as zero if any of the following applies, and reads as one otherwise:
  - ARMv8.2-Debug is not implemented and ExternalSecureNoninvasiveDebugEnabled() returns TRUE.
  - EL3 is using AArch32 and the value of SDCR.SPME is 1.
  - EL3 is using AArch64 and the value of MDCR_EL3.SPME is 1.

Arm deprecates use of this field.

SPIDdis, bit [16]

Secure privileged AArch32 invasive self-hosted debug disabled status bit. This bit is RO and depends on the value of SDCR.SPD and the pseudocode function AArch32.SelfHostedSecurePrivilegedInvasiveDebugEnabled(). Permitted values are:

SPIDdis	Meaning
0b0	Self-hosted debug enabled in Secure privileged AArch32 modes.
0b1	Self-hosted debug disabled in Secure privileged AArch32 modes.

This bit reads as 1 if any of the following is true and reads as 0 otherwise:

- SDCR.SPD has the value 0b10.
- SDCR.SPD has the value 0b00 and SelfHostedSecurePrivilegedInvasiveDebugEnabled() returns FALSE.

Arm deprecates use of this field.

MDBGen, bit [15]

Monitor debug events enable. Enable Breakpoint, Watchpoint, and Vector Catch exceptions.

MDBGen	Meaning
0b0	Breakpoint, Watchpoint, and Vector Catch exceptions disabled.
0b1	Breakpoint, Watchpoint, and Vector Catch exceptions enabled.

On a Warm reset, this field resets to 0.

HDE, bit [14]

Used for save/restore of EDSCR.HDE.

When OSLSR_EL1.OSLK == 0, this bit is RO, and software must treat it as UNK/SBZP.

When DBGOSLSR.OSLK == 1, this bit is RW and holds the value of EDSCR.HDE.

Reads and writes of this bit are indirect accesses to EDSCR.HDE.

The architected behavior of this field determines the value it returns after a reset.

Bit [13]

Reserved, RES0.

UDCCdis, bit [12]

Traps EL0 accesses to the DCC registers to Undefined mode.

UDCCdis	Meaning
0b0	This control does not cause any instructions to be trapped.
0b1	EL0 accesses to the DBGDSCRint, DBGDTRRXint, DBGDTRTXint, DBGDIDR, DBGDSAR, and DBGDRAR are trapped to Undefined mode.

Note

All accesses to these registers are trapped, including LDC and STC accesses to DBGDTRTXint and DBGDTRRXint, and MRRC accesses to DBGDSAR and DBGDRAR.

Traps of EL0 accesses to the DBGDTRRXint and DBGDTRTXint are ignored in Debug state.

On a Warm reset, this field resets to 0.

Bits [11:7]

Reserved, RES0.

ERR, bit [6]

Used for save/restore of EDSCR.ERR.

When OSLSR_EL1.OSLK == 0, this bit is RO, and software must treat it as UNK/SBZP.

When DBGOSLSR.OSLK == 1, this bit is RW and holds the value of EDSCR.ERR.

Reads and writes of this bit are indirect accesses to EDSCR.ERR.

The architected behavior of this field determines the value it returns after a reset.

MOE, bits [5:2]

Method of Entry for debug exception. When a debug exception is taken to an Exception level using AArch32, this field is set to indicate the event that caused the exception:

MOE	Meaning
0b0001	Breakpoint.
0b0011	Software breakpoint (BKPT) instruction.
0b0101	Vector catch.
0b1010	Watchpoint.

On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [1:0]

Reserved, RES0.

Accessing the DBGDSCRext

Individual fields within this register might have restricted accessibility when the OS lock is unlocked, DBGOSLSR.OSLK == 0. See the field descriptions for more detail.

Accesses to this register use the following encodings:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
~~opc1~~	~~opc2~~	~~CRn~~	~~coproc~~	~~CRm~~
0b1110	0b000	0b0000	0b0010	0b010
~~0b000~~	~~0b010~~	~~0b0000~~	~~0b1110~~	~~0b0010~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.<TDE,TDA> != '00' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x05);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.<TDE,TDA> != '00' then
        AArch32.TakeHypTrapException(0x05);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x05);
    else
        return DBGDSCRext;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x05);
    else
        return DBGDSCRext;
elsif PSTATE.EL == EL3 then
    return DBGDSCRext;
```

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
~~opc1~~	~~opc2~~	~~CRn~~	~~coproc~~	~~CRm~~
0b1110	0b000	0b0000	0b0010	0b010
~~0b000~~	~~0b010~~	~~0b0000~~	~~0b1110~~	~~0b0010~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.<TDE,TDA> != '00' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x05);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.<TDE,TDA> != '00' then
        AArch32.TakeHypTrapException(0x05);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x05);
    else
        DBGDSCRext = R[t];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x05);
    else
        DBGDSCRext = R[t];
elsif PSTATE.EL == EL3 then
    DBGDSCRext = R[t];
```

27~~13~~/03~~12~~/2019~~2018~~ 21~~16~~:59~~42~~; e5e4db499bf9867a4b93324c4dbac985d3da93766~~379d01e197f1d40720d32d0f84e419e9187e009~~

(old) htmldiff from- (new)

(old) htmldiff from- (new)

DBGOSDLR, Debug OS Double Lock Register

The DBGOSDLR characteristics are:

Purpose

Locks out the external debug interface.

Configuration

AArch32 System register DBGOSDLR bits [31:0] are architecturally mapped to AArch64 System register OSDLR_EL1[31:0] .

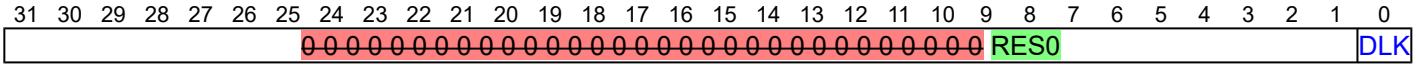
This register is in the Warm reset domain. Some or all RW fields of this register have defined reset values. On a Warm or Cold reset these apply only if the PE resets into an Exception level that is using AArch32. Otherwise, on a Warm or Cold reset RW fields in this register reset to architecturally UNKNOWN values.

Attributes

DBGOSDLR is a 32-bit register.

Field descriptions

The DBGOSDLR bit assignments are:



Bits [31:1]

Reserved, RES0.

DLK, bit [0]

When ARMv8.0-DoubleLock is implemented:

OS Double Lock control bit.

DLK	Meaning
0b0	OS Double Lock unlocked.
0b1	OS Double Lock locked, if DBGPRCR.CORENPDRQ (Core no powerdown request) bit is set to 0 and the PE is in Non-debug state.

On a Warm reset, this field resets to 0.

Otherwise:

Reserved, RAZ/WI.

Accessing the DBGOSDLR

Accesses to this register use the following encodings:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
~~opc1~~	~~opc2~~	~~CRn~~	~~coproc~~	~~CRm~~
0b1110	0b000	0b0001	0b0011	0b100
~~0b000~~	~~0b100~~	~~0b0001~~	~~0b1110~~	~~0b0011~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.<TDE,TDOSA> != '00' &&
(IsFeatureImplemented(ARMv8.0-DoubleLock) || boolean IMPLEMENTATION_DEFINED "Trapped by
MDCR_EL2.TDOSA") then
        AArch64.AArch32SystemAccessTrap(EL2, 0x05);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.<TDE,TDOSA> != '00' &&
(IsFeatureImplemented(ARMv8.0-DoubleLock) || boolean IMPLEMENTATION_DEFINED "Trapped by
HDCR.TDOSA") then
        AArch32.TakeHypTrapException(0x05);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDOSA == '1' &&
(IsFeatureImplemented(ARMv8.0-DoubleLock) || boolean IMPLEMENTATION_DEFINED "Trapped by
MDCR_EL3.TDOSA") then
        AArch64.AArch32SystemAccessTrap(EL3, 0x05);
    else
        return DBGOSDLR;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDOSA == '1' &&
(IsFeatureImplemented(ARMv8.0-DoubleLock) || boolean IMPLEMENTATION_DEFINED "Trapped by
MDCR_EL3.TDOSA") then
        AArch64.AArch32SystemAccessTrap(EL3, 0x05);
    else
        return DBGOSDLR;
elsif PSTATE.EL == EL3 then
    return DBGOSDLR;
```

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
~~opc1~~	~~opc2~~	~~CRn~~	~~coproc~~	~~CRm~~
0b1110	0b000	0b0001	0b0011	0b100
~~0b000~~	~~0b100~~	~~0b0001~~	~~0b1110~~	~~0b0011~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.<TDE,TDOSA> != '00' &&
(IsFeatureImplemented(ARMv8.0-DoubleLock) || boolean IMPLEMENTATION_DEFINED "Trapped by
MDCR_EL2.TDOSA") then
        AArch64.AArch32SystemAccessTrap(EL2, 0x05);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.<TDE,TDOSA> != '00' &&
(IsFeatureImplemented(ARMv8.0-DoubleLock) || boolean IMPLEMENTATION_DEFINED "Trapped by
HDCR.TDOSA") then
        AArch32.TakeHypTrapException(0x05);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDOSA == '1' &&
(IsFeatureImplemented(ARMv8.0-DoubleLock) || boolean IMPLEMENTATION_DEFINED "Trapped by
MDCR_EL3.TDOSA") then
        AArch64.AArch32SystemAccessTrap(EL3, 0x05);
    else
        DBGOSDLR = R[t];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDOSA == '1' &&
(IsFeatureImplemented(ARMv8.0-DoubleLock) || boolean IMPLEMENTATION_DEFINED "Trapped by
MDCR_EL3.TDOSA") then
        AArch64.AArch32SystemAccessTrap(EL3, 0x05);
    else
        DBGOSDLR = R[t];
elsif PSTATE.EL == EL3 then
    DBGOSDLR = R[t];
```

2713/0312/20192018 2116:5942: e5e4db499bf9867a4b93324c4dbac985d3da93766379d01e197f1d40720d32d0f84e419e9187e009

(old) htmldiff from- (new)

(old) htmldiff from- (new)

DBGWCR<n>, Debug Watchpoint Control Registers, n = 0 - 15

The DBGWCR<n> characteristics are:

Purpose

Holds control information for a watchpoint. Forms watchpoint n together with value register DBGWVR<n>.

Configuration

AArch32 System register DBGWCR<n> bits [31:0] are architecturally mapped to AArch64 System register DBGWCR<n>_EL1[31:0] .

AArch32 System register DBGWCR<n> bits [31:0] are architecturally mapped to External register DBGWCR<n>_EL1[31:0] .

If breakpoint n is not implemented then this register is unallocated.

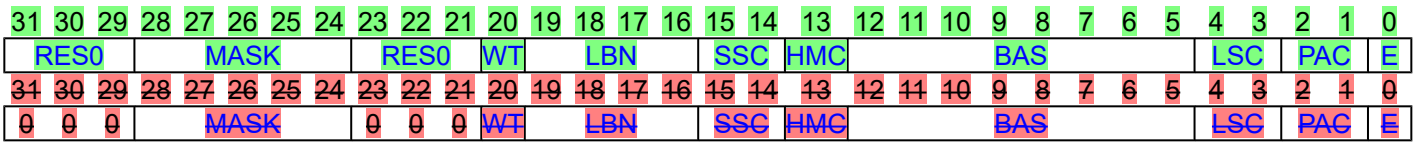
This register is in the Cold reset domain. On a Cold reset RW fields in this register reset to architecturally UNKNOWN values. The register is not affected by a Warm reset.

Attributes

DBGWCR<n> is a 32-bit register.

Field descriptions

The DBGWCR<n> bit assignments are:



When the E field is zero, all the other fields in the register are ignored.

Bits [31:29]

Reserved, RES0.

MASK, bits [28:24]

Address mask. Only objects up to 2GB can be watched using a single mask.

MASK	Meaning
0b00000	No mask.
0b00001	Reserved.
0b00010	Reserved.

If programmed with a reserved value, a watchpoint must behave as if either:

- MASK has been programmed with a defined value, which might be 0 (no mask), other than for a direct read of DBGWCRn_EL1.
- The watchpoint is disabled.

Software must not rely on this property because the behavior of reserved values might change in a future revision of the architecture.

Other values mask the corresponding number of address bits, from 0b00011 masking 3 address bits (0x00000007 mask for address) to 0b11111 masking 31 address bits (0x7FFFFFFF mask for address).

On a Cold reset, this field resets to an architecturally UNKNOWN value.

Bits [23:21]

Reserved, RES0.

WT, bit [20]

Watchpoint type. Possible values are:

WT	Meaning
0b0	Unlinked data address match.
0b1	Linked data address match.

On a Cold reset, this field resets to an architecturally UNKNOWN value.

LBN, bits [19:16]

Linked breakpoint number. For Linked data address watchpoints, this specifies the index of the Context-matching breakpoint linked to.

On a Cold reset, this field resets to an architecturally UNKNOWN value.

SSC, bits [15:14]

Security state control. Determines the Security states under which a Watchpoint debug event for watchpoint n is generated. This field must be interpreted along with the HMC and PAC fields.

For more information, see 'Execution conditions for which a breakpoint generates Breakpoint exceptions' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile~~, section G2 (AArch32 Self-hosted Debug)~~, and 'Reserved DBGBCR<n>.{SSC~~HMC~~, HMC~~SSC~~, PMC} values' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile~~, section G2 (AArch32 Self-hosted Debug)~~.

On a Cold reset, this field resets to an architecturally UNKNOWN value.

HMC, bit [13]

Higher mode control. Determines the debug perspective for deciding when a Watchpoint debug event for watchpoint n is generated. This field must be interpreted along with the SSC and PAC fields.

For more information on the operation of the SSC, HMC, and PAC fields, see 'Execution conditions for which a watchpoint generates Watchpoint exceptions' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section G2 (AArch32 Self-hosted Debug).

On a Cold reset, this field resets to an architecturally UNKNOWN value.

BAS, bits [12:5]

Byte address select. Each bit of this field selects whether a byte from within the word or double-word addressed by DBGWVR<n> is being watched.

BAS	Description
0bxxxxxxx1	Match byte at DBGWVR<n>
0bxxxxxx1x	Match byte at DBGWVR<n>+1
0bxxxxx1xx	Match byte at DBGWVR<n>+2
0bxxxx1xxx	Match byte at DBGWVR<n>+3

In cases where DBGWVR<n> addresses a double-word:

BAS	Description, if DBGWVR<n>[2] == 0
0bxxx1xxxx	Match byte at DBGWVR<n>+4
0bxx1xxxxx	Match byte at DBGWVR<n>+5
0bx1xxxxxx	Match byte at DBGWVR<n>+6
0b1xxxxxxx	Match byte at DBGWVR<n>+7

If DBGWVR<n>[2] == 1, only BAS[3:0] are used and BAS[7:4] are ignored. Arm deprecates setting DBGWVR<n>[2] == 1.

The valid values for BAS are non-zero binary numbers all of whose set bits are contiguous. All other values are reserved and must not be used by software. See 'Reserved DBGWCR<n>.BAS values' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section G2 (AArch32 Self-hosted Debug)

On a Cold reset, this field resets to an architecturally UNKNOWN value.

LSC, bits [4:3]

Load/store control. This field enables watchpoint matching on the type of access being made. Possible values of this field are:

LSC	Meaning
0b01	Match instructions that load from a watchpointed address.
0b10	Match instructions that store to a watchpointed address.
0b11	Match instructions that load from or store to a watchpointed address.

All other values are reserved, but must behave as if the watchpoint is disabled. Software must not rely on this property as the behavior of reserved values might change in a future revision of the architecture.

On a Cold reset, this field resets to an architecturally UNKNOWN value.

PAC, bits [2:1]

Privilege of access control. Determines the Exception level or levels at which a Watchpoint debug event for watchpoint n is generated. This field must be interpreted along with the SSC and HMC fields.

For more information on the operation of the SSC, HMC, and PAC fields, see 'Execution conditions for which a watchpoint generates Watchpoint exceptions' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section G2 (AArch32 Self-hosted Debug).

On a Cold reset, this field resets to an architecturally UNKNOWN value.

E, bit [0]

Enable watchpoint n. Possible values are:

E	Meaning
0b0	Watchpoint disabled.
0b1	Watchpoint enabled.

On a Cold reset, this field resets to an architecturally UNKNOWN value.

Accessing the DBGWCR<n>

Accesses to this register use the following encodings:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
~~opc1~~	~~opc2~~	~~CRn~~	~~coproc~~	~~CRm~~
0b1110	0b000	0b0000	0bnnnn	0b111
~~0b000~~	~~0b111~~	~~0b0000~~	~~0b1110~~	~~0bnnnn~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.<TDE,TDA> != '00' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x05);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.<TDE,TDA> != '00' then
        AArch32.TakeHypTrapException(0x05);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x05);
    elsif ELUsingAArch32(EL1) && DBGOSLSR.OSLK == '0' && HaltingAllowed() && EDSCR.TDA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        return DBGWCR[UInt(CRm<3:0>)];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x05);
    elsif ELUsingAArch32(EL1) && DBGOSLSR.OSLK == '0' && HaltingAllowed() && EDSCR.TDA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        return DBGWCR[UInt(CRm<3:0>)];
elsif PSTATE.EL == EL3 then
    if ELUsingAArch32(EL1) && DBGOSLSR.OSLK == '0' && HaltingAllowed() && EDSCR.TDA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        return DBGWCR[UInt(CRm<3:0>)];
```

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
~~opc1~~	~~opc2~~	~~CRn~~	~~coproc~~	~~CRm~~
0b1110	0b000	0b0000	0bnnnn	0b111
~~0b000~~	~~0b111~~	~~0b0000~~	~~0b1110~~	~~0bnnnn~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.<TDE,TDA> != '00' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x05);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.<TDE,TDA> != '00' then
        AArch32.TakeHypTrapException(0x05);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x05);
    elsif ELUsingAArch32(EL1) && DBGOSLSR.OSLK == '0' && HaltingAllowed() && EDSCR.TDA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        DBGWCR[UInt(CRm<3:0>)] = R[t];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x05);
    elsif ELUsingAArch32(EL1) && DBGOSLSR.OSLK == '0' && HaltingAllowed() && EDSCR.TDA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        DBGWCR[UInt(CRm<3:0>)] = R[t];
elsif PSTATE.EL == EL3 then
    if ELUsingAArch32(EL1) && DBGOSLSR.OSLK == '0' && HaltingAllowed() && EDSCR.TDA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        DBGWCR[UInt(CRm<3:0>)] = R[t];
```

(old) htmldiff from- (new)

(old) htmldiff from- (new)

DFAR, Data Fault Address Register

The DFAR characteristics are:

Purpose

Holds the virtual address of the faulting address that caused a synchronous Data Abort exception.

Configuration

~~AArch32 System register DFAR bits [31:0] (S) are architecturally mapped to AArch64 System register FAR_EL2[31:0] when HaveEL(EL2).~~

AArch32 System register DFAR bits [31:0] ~~(NS)~~ are architecturally mapped to AArch64 System register FAR_EL1[31:0] .

AArch32 System register DFAR bits [31:0] (S) are architecturally mapped to AArch32 System register HDFAR[31:0] when HaveEL(EL2), HaveEL(EL3) and HighestELUsingAArch32().~~)~~

RW fields in this register reset to architecturally UNKNOWN values.

Attributes

DFAR is a 32-bit register.

Field descriptions

The DFAR bit assignments are:

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
VA of faulting address of synchronous Data Abort exception																															

Bits [31:0]

VA of faulting address of synchronous Data Abort exception.

This field resets to an architecturally UNKNOWN value.

Accessing the DFAR

Accesses to this register use the following encodings:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
~~opc1~~	~~opc2~~	~~CRn~~	~~coproc~~	~~CRm~~
0b1111	0b000	0b0110	0b0000	0b000
~~0b000~~	~~0b000~~	~~0b0110~~	~~0b1111~~	~~0b0000~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T6 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T6 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TRVM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TRVM == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) then
        if SCR.NS == '0' then
            return DFAR_S;
        else
            return DFAR_NS;
    else
        return DFAR;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && ELUsingAArch32(EL3) then
        return DFAR_NS;
    else
        return DFAR;
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        return DFAR_S;
    else
        return DFAR_NS;
```

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
~~opc1~~	~~opc2~~	~~CRn~~	~~coproc~~	~~CRm~~
0b1111	0b000	0b0110	0b0000	0b000
~~0b000~~	~~0b000~~	~~0b0110~~	~~0b1111~~	~~0b0000~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T6 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T6 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TVM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TVM == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) then
        if SCR.NS == '0' then
            DFAR_S = R[t];
        else
            DFAR_NS = R[t];
    else
        DFAR = R[t];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && ELUsingAArch32(EL3) then
        DFAR_NS = R[t];
    else
        DFAR = R[t];
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        DFAR_S = R[t];
    else
        DFAR_NS = R[t];
```

2713/0312/20192018 2116:5942; e5e4db499bf9867a4b93324c4dbac985d3da93766379d01e197f1d40720d32d0f84e419e9187e009

(old) htmldiff from- (new)

(old) htmldiff from- (new)

DTLBIALL, Data TLB Invalidate All

The DTLBIALL characteristics are:

Purpose

Invalidate all cached copies of translation table entries from data TLBs that are from any level of the translation table walk. The entries that are invalidated are as follows:

- If executed at Secure EL1 when EL3 is using AArch64, all entries that would be required for the Secure EL1&0 translation regime.
- If executed in Secure state when EL3 is using AArch32, all entries that would be required for the Secure PL1&0 translation regime.
- If executed at Non-secure EL1, all stage 1 translation table entries that would be required for the Non-secure PL1&0 translation regime and, if EL2 is implemented, they must match the current VMID.
- If executed at EL2, the stage 1 or stage 2 translation table entries that would be required for the Non-secure PL1&0 translation regime and matches the current VMID.

The invalidation only applies to the PE that executes this System instruction.

Arm deprecates the use of this System instruction. It is only provided for backwards compatibility with earlier versions of the Arm architecture.

Configuration

Attributes

DTLBIALL is a 32-bit System instruction.

Field descriptions

DTLBIALL ignores the value in the register specified by the instruction encoding. Software does not have to write a value to the register before issuing this instruction.

Executing the DTLBIALL instruction

Accesses to this instruction use the following encodings:

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
~~opc1~~	~~opc2~~	~~CRn~~	~~coproc~~	~~CRm~~
0b1111	0b000	0b1000	0b0110	0b000
~~0b000~~	~~0b000~~	~~0b1000~~	~~0b1111~~	~~0b0110~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T8 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T8 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TTLB == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR.TTLB == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        DTLBIALL();
elsif PSTATE.EL == EL2 then
    DTLBIALL();
elsif PSTATE.EL == EL3 then
    DTLBIALL();
```

27~~13~~/03~~12~~/2019~~2018~~ 21~~16~~:59~~42~~; e5e4db499bf9867a4b93324c4dbac985d3da9376~~6379d01e197f1d40720d32d0f84e419e9187e009~~

(old) htmldiff from- (new)

(old) htmldiff from- (new)

DTLBIASID, Data TLB Invalidate by ASID match

The DTLBIASID characteristics are:

Purpose

Invalidate all cached copies of translation table entries from data TLBs that meet the following requirements:

- The entry is a stage 1 translation table entry.
- The entry would be used for the specified ASID, and either:
  - Is from a level of lookup above the final level.
  - Is a non-global entry from the final level of lookup.
- If EL2 is implemented, the entry would be used with the current VMID.

From the entries that match these requirement, the entries that are invalidated are required for the following translation regime:

- If executed at Secure EL1 when EL3 is using AArch64, the Secure EL1&0 translation regime.
- If executed in Secure state when EL3 is using AArch32, the Secure PL1&0 translation regime.
- If executed in Non-secure state, the Non-secure PL1&0 translation regime.

The invalidation only applies to the PE that executes this System instruction.

Arm deprecates the use of this System instruction. It is only provided for backwards compatibility with earlier versions of the Arm architecture.

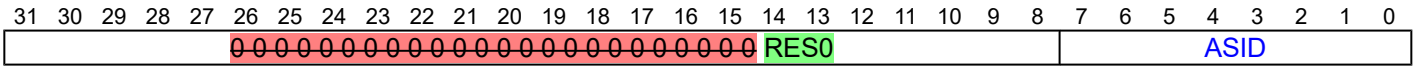
Configuration

Attributes

DTLBIASID is a 32-bit System instruction.

Field descriptions

The DTLBIASID input value bit assignments are:



Bits [31:8]

Reserved, RES0.

ASID, bits [7:0]

ASID value to match. Any TLB entries for non-global pages that match the ASID values will be affected by this System instruction.~~operation.~~

Executing the DTLBIASID instruction

Accesses to this instruction use the following encodings:

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
~~opc1~~	~~opc2~~	~~CRn~~	~~coproc~~	~~CRm~~
0b1111	0b000	0b1000	0b0110	0b010

0b000	0b010	0b1000	0b1111	0b0110

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T8 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T8 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TTLB == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR.TTLB == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        DTLBIASID(R[t]);
elsif PSTATE.EL == EL2 then
    DTLBIASID(R[t]);
elsif PSTATE.EL == EL3 then
    DTLBIASID(R[t]);
```

2713/0312/20192018 2116:5942; e5e4db499bf9867a4b93324c4dbac985d3da93766379d01e197f1d40720d32d0f84e419e9187e009

(old) htmldiff from- (new)

(old) htmldiff from- (new)

DTLBIMVA, Data TLB Invalidate by VA

The DTLBIMVA characteristics are:

Purpose

Invalidate all cached copies of translation table entries from data TLBs that meet the following requirements:

- The entry is a stage 1 translation table entry.
- The entry would be used to translate the specified address, and one of the following applies:
  - The entry is from a level of lookup above the final level and matches the specified ASID.
  - The entry is a global entry from the final level of lookup.
  - The entry is a non-global entry from the final level of lookup that matches the specified ASID.
- If EL2 is implemented, the entry would be used with the current VMID.

From the entries that match these requirement, the entries that are invalidated are required for the following translation regime:

- If executed at Secure EL1 when EL3 is using AArch64, the Secure EL1&0 translation regime.
- If executed in Secure state when EL3 is using AArch32, the Secure PL1&0 translation regime.
- If executed in Non-secure state, the Non-secure PL1&0 translation regime.

The invalidation only applies to the PE that executes this System instruction.

Arm deprecates the use of this System instruction. It is only provided for backwards compatibility with earlier versions of the Arm architecture.

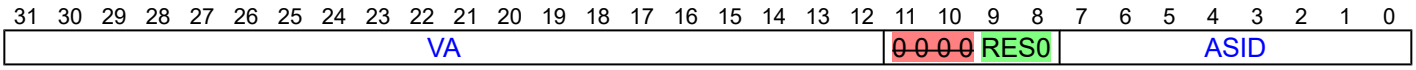
Configuration

Attributes

DTLBIMVA is a 32-bit System instruction.

Field descriptions

The DTLBIMVA input value bit assignments are:



VA, bits [31:12]

Virtual address to match. Any TLB entries that match the ASID value and VA value will be affected by this System instruction.~~operation.~~

Bits [11:8]

Reserved, RES0.

ASID, bits [7:0]

ASID value to match. Any TLB entries that match the ASID value and VA value will be affected by this System instruction.~~operation.~~

Global TLB entries that match the VA value will be affected by this System instruction~~operation~~, regardless of the value of the ASID field.

Executing the DTLBIMVA instruction

Accesses to this instruction use the following encodings:

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
~~opc1~~	~~opc2~~	~~CRn~~	~~coproc~~	~~CRm~~
0b1111	0b000	0b1000	0b0110	0b001
~~0b000~~	~~0b001~~	~~0b1000~~	~~0b1111~~	~~0b0110~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T8 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T8 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TTLB == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR.TTLB == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        DTLBIMVA(R[t]);
elsif PSTATE.EL == EL2 then
    DTLBIMVA(R[t]);
elsif PSTATE.EL == EL3 then
    DTLBIMVA(R[t]);
```

27~~13~~/03~~12~~/2019~~2018~~ 21~~16~~:59~~42~~; e5e4db499bf9867a4b93324c4dbac985d3da93766379d01e197f1d40720d32d0f84e419e9187e009

(old) htmldiff from- (new)

ELR_hyp, Exception Link Register (Hyp mode)

The ELR_hyp characteristics are:

Purpose

When taking an exception to Hyp mode, holds the address to return to.

Configuration

AArch32 System register ELR_hyp bits [31:0] are architecturally mapped to AArch64 System register ELR_EL2[31:0] .

On a reset into an Exception level that is using AArch32 ELR_hyp is UNKNOWN.

Attributes

ELR_hyp is a 32-bit register.

Field descriptions

The ELR_hyp bit assignments are:

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Return address																															

Bits [31:0]

Return address.

This field resets to an architecturally UNKNOWN value.

Accessing the ELR_hyp

ELR_hyp is accessible only at Hyp mode and Monitor mode. For more details, see MRS (banked register) and MSR (banked register) in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

Accesses to this register use the following encodings:

```
MRS{<c>}{<q>} <Rd>, ELR_hyp
```

R	M	M1
~~M~~	~~R~~	~~M1~~
0b0	0b1	0b1110
~~0b1~~	~~0b0~~	~~0b1110~~

```
MSR{<c>}{<q>} ELR_hyp, <Rn>
```

R	M	M1
~~M~~	~~R~~	~~M1~~
0b0	0b1	0b1110
~~0b1~~	~~0b0~~	~~0b1110~~

27~~13~~/03~~12~~/2019~~2018~~ 21~~16~~:59~~42~~; e5e4db499bf9867a4b93324c4dbac985d3da93766~~379d01e197f1d40720d32d0f84e419e9187e009~~

(old) htmldiff from- (new)

(old) htmldiff from- (new)

HCPTR, Hyp Architectural Feature Trap Register

The HCPTR characteristics are:

Purpose

Controls:

- Trapping to Hyp mode of Non-secure access, at EL1 or EL0, to trace, and to Advanced SIMD and floating-point functionality.
- Hyp mode access to trace, and to Advanced SIMD and floating-point functionality.

Note

Accesses to this functionality:

- From Non-secure modes other than Hyp mode are also affected by settings in the CPACR and NSACR.
- From Hyp mode are also affected by settings in the NSACR.

Exceptions generated by the CPACR and NSACR controls are higher priority than those generated by the HCPTR controls.

Configuration

AArch32 System register HCPTR bits [31:0] are architecturally mapped to AArch64 System register CPTR_EL2[31:0] .

If EL2 is not implemented, this register is RES0 from EL3.

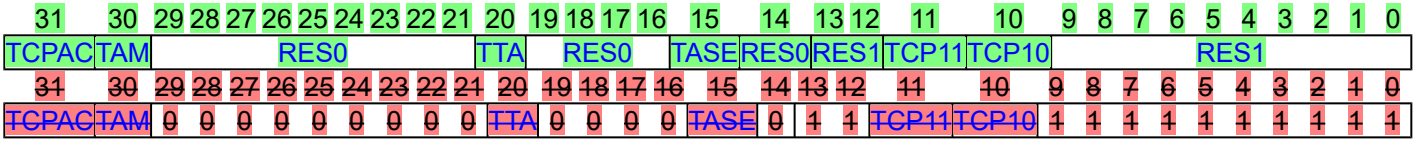
Some or all RW fields of this register have defined reset values. These apply only if the PE resets into EL2 with EL2 using AArch32, or into EL3 with EL3 using AArch32. Otherwise, RW fields in this register reset to architecturally UNKNOWN values.

Attributes

HCPTR is a 32-bit register.

Field descriptions

The HCPTR bit assignments are:



TCPAC, bit [31]

Traps Non-secure EL1 accesses to the CPACR to Hyp mode.

TCPAC	Meaning
0b0	This control does not cause any instructions to be trapped.
0b1	Non-secure EL1 accesses to the CPACR are trapped to Hyp mode.

Note

The CPACR is not accessible at EL0.

In a system where the PE resets into EL2 or EL3, this field resets to 0.

TAM, bit [30]

When AMUv1 is implemented:

Trap Activity Monitor access. Traps Non-secure EL1 and EL0 accesses to all Activity Monitor registers to EL2.

TAM	Meaning
0b0	Accesses from Non-secure EL1 and EL0 to Activity Monitor registers are not trapped.
0b1	Accesses from Non-secure EL1 and EL0 to Activity Monitor registers are trapped to Hyp mode.

In a system where the PE resets into EL2 or EL3, this field resets to ~~an architecturally~~ 0~~UNKNOWN~~.~~value.~~

Otherwise:

Reserved, RES0.

Bits [29:21]

Reserved, RES0.

TTA, bit [20]

Traps Non-secure System register accesses to all implemented trace registers to Hyp mode.

TTA	Meaning
0b0	This control does not cause any instructions to be trapped.
0b1	Any Non-secure System register access to an implemented trace register is trapped to Hyp mode, unless the access is trapped to EL1 by a CPACR or NSACR control, or the access is from Non-secure EL0 and the definition of the register in the appropriate trace architecture specification indicates that the register is not accessible from EL0. A trapped instruction generates: • A Hyp Trap exception, if the exception is taken from Non-secure EL0 or EL1. • An Undefined Instruction exception taken to Hyp mode, if the exception is taken from Hyp mode.

If the implementation does not include a PE trace unit, or does not include a System register interface to the PE trace unit registers, it is IMPLEMENTATION DEFINED whether this bit:

- Is RES0.
- Is RES1.
- Can be written from Hyp mode, and from Secure Monitor mode when SCR.NS is 1.

If EL3 is implemented and is using AArch32, and the value of NSACR.NSTRCDIS is 1, in Non-secure state this field behaves as RAO/WI, regardless of its actual value.

Note

- The ETMv4 architecture does not permit EL0 to access the trace registers. If the implementation includes an ETMv4 implementation, EL0 accesses to the trace registers are UNDEFINED, and a resulting Undefined Instruction exception is higher priority than a HCPTR.TTA Hyp Trap exception.
- The architecture does not provide traps on trace register accesses through the optional memory-mapped debug interface.

System register accesses to the trace registers can have side-effects. When a System register access is trapped, any side-effects that are normally associated with the access do not occur before the exception is taken.

In a system where the PE resets into EL2 or EL3, this field resets to 0.

Bits [19:16]

Reserved, RES0.

TASE, bit [15]

Traps Non-secure execution of Advanced SIMD instructions to Hyp mode when the value of HCPTR.TCP10 is 0.

TASE	Meaning
0b0	This control does not cause any instructions to be trapped.
0b1	When the value of HCPTR.TCP10 is 0, any attempt to execute an Advanced SIMD instruction in Non-secure state is trapped to Hyp mode, unless it is trapped to EL1 by a CPACR or NSACR control. A trapped instruction generates: • A Hyp Trap exception, if the exception is taken from Non-secure EL0 or EL1. • An Undefined Instruction exception taken to Hyp mode, if the exception is taken from Hyp mode.

When the value of HCPTR.TCP10 is 1, the value of this field is ignored.

If the implementation does not include Advanced SIMD and floating-point functionality, this field is RES1. Otherwise, it is IMPLEMENTATION DEFINED whether this field is implemented as a RW field. If it is not implemented as a RW field, then it is RAZ/WI.

If EL3 is implemented and is using AArch32, and the value of NSACR.NSASEDIS is 1, in Non-secure state this field behaves as RAO/WI, regardless of its actual value. This applies even if the field is implemented as RAZ/WI.

For the list of instructions affected by this field, see 'Controls of Advanced SIMD operation that do not apply to floating-point operation' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section E1.

In a system where the PE resets into EL2 or EL3, this field resets to 0.

Bit [14]

Reserved, RES0.

Bits [13:12]

Reserved, RES1.

TCP11, bit [11]

The value of this field is ignored. If this field is programmed with a different value to the TCP10 bit then this field is UNKNOWN on a direct read of the HCPTR.

If the implementation does not include Advanced SIMD and floating-point functionality, this field is RES1.

If EL3 is implemented and is using AArch32, and the value of NSACR.cp10 is 0, in Non-secure state this field behaves as RAO/WI, regardless of its actual value.

In a system where the PE resets into EL2 or EL3, this field resets to 0.

TCP10, bit [10]

Trap Non-secure accesses to Advanced SIMD and floating-point functionality to Hyp mode:

TCP10	Meaning
0b0	This control does not cause any instructions to be trapped.
0b1	Any attempted access to Advanced SIMD and floating-point functionality from Non-secure state is trapped to Hyp mode, unless it is trapped to EL1 by a CPACR or NSACR control. A trapped instruction generates: • A Hyp Trap exception, if the exception is taken from Non-secure EL0 or EL1. • An Undefined Instruction exception taken to Hyp mode, if the exception is taken from Hyp mode.

The Advanced SIMD and floating-point features controlled by these fields are:

- Execution of any floating-point or Advanced SIMD instruction.
- Any access to the Advanced SIMD and floating-point registers D0-D31 and their views as S0-S31 and Q0-Q15.
- Any access to the FPSCR, FPSID, MVFR0, MVFR1, MVFR2, or FPEXC System registers.

If the implementation does not include Advanced SIMD and floating-point functionality, this field is RES1.

If EL3 is implemented and is using AArch32, and the value of NSACR.cp10 is 0, in Non-secure state this field behaves as RAO/WI, regardless of its actual value.

In a system where the PE resets into EL2 or EL3, this field resets to `0`.

Bits [9:0]

Reserved, RES1.

Accessing the HCPTR

Accesses to this register use the following encodings:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
~~opc1~~	~~opc2~~	~~CRn~~	~~coproc~~	~~CRm~~
0b1111	0b100	0b0001	0b0001	0b010
~~0b100~~	~~0b010~~	~~0b0001~~	~~0b1111~~	~~0b0001~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T1 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T1 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TCPAC == '1' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        return HCPTR;
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        UNDEFINED;
    else
        return HCPTR;
```

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
~~opc1~~	~~opc2~~	~~CRn~~	~~coproc~~	~~CRm~~
0b1111	0b100	0b0001	0b0001	0b010
~~0b100~~	~~0b010~~	~~0b0001~~	~~0b1111~~	~~0b0001~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T1 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T1 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TCPAC == '1' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        HCPTR = R[t];
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        UNDEFINED;
    else
        HCPTR = R[t];
```

2713/0312/20192018 2116:5942; e5e4db499bf9867a4b93324c4dbac985d3da93766379d01e197f1d40720d32d0f84c419c9187c009

(old) htmldiff from- (new)

(old) htmldiff from- (new)

HDCR, Hyp Debug Control Register

The HDCR characteristics are:

Purpose

Controls the trapping to Hyp mode of Non-secure accesses, at EL1 or lower, to functions provided by the debug and trace architectures and the Performance Monitors Extension.

Configuration

AArch32 System register HDCR bits [31:0] are architecturally mapped to AArch64 System register MDCR_EL2[31:0] .

If EL2 is not implemented, this register is RES0 from EL3, and other than for a direct read of the register, the PE behaves as if HDCR.HPMN == PMCR.N.

This register is in the Warm reset domain. Some or all RW fields of this register have defined reset values. On a Warm or Cold reset these apply only if the PE resets into EL2 with EL2 using AArch32, or into EL3 with EL3 using AArch32. Otherwise, on a Warm or Cold reset RW fields in this register reset to architecturally UNKNOWN values.

Attributes

HDCR is a 32-bit register.

Field descriptions

The HDCR bit assignments are:

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RES0					HLP	RES0		HCCD	RES0			TTRF	RES0		HPMD	RES0				TDRA	TDOSA	TDA	TDE	HPME	TPM	TPMCR	HPMN					
~~0~~	~~0~~	~~0~~	~~0~~	~~0~~	~~HLP~~	~~0~~	~~0~~	~~HCCD~~	~~0~~	~~0~~	~~0~~	~~TTRF~~	~~0~~	~~HPMD~~	~~0~~	~~0~~	~~0~~	~~0~~	~~0~~	~~TDRA~~	~~TDOSA~~	~~TDA~~	~~TDE~~	~~HPME~~	~~TPM~~	~~TPMCR~~	~~HPMN~~					

Bits [31:27]

Reserved, RES0.

HLP, bit [26]

When ARMv8.5-PMU is implemented:

Hypervisor Long event counter enable. Determines when unsigned overflow is recorded by a counter overflow bit.

HLP	Meaning
0b0	Event counter overflow on increment that causes unsigned overflow of PMEVCNTR<n>[31:0].
0b1	Event counter overflow on increment that causes unsigned overflow of PMEVCNTR<n>[63:0].

If the highest implemented Exception level is using AArch32, it is IMPLEMENTATION DEFINED whether this bit is read/write or RAZ/WI.

If HDCR.HPMN is less than PMCR.N, this bit affects the operation of event counters in the range [HDCR.HPMN:(PMCR.N-1)]. Otherwise this bit has no effect on the operation of the event counters.

Note

The effect of HDCR.HPMN on the operation of this bit applies regardless of whether EL2 is enabled in the current Security state.

For more information see the description of the HPMN field.

PMEVCNTR<n>[63:32] cannot be accessed directly in AArch32 state.

On a Warm reset, in a system where the PE resets into EL2 or EL3, this field resets to `0`.

Otherwise:

Reserved, RES0.

Bits [25:24]

Reserved, RES0.

HCCD, bit [23]

When ARMv8.5-PMU is implemented:

Hypervisor Cycle Counter Disable. Prohibits PMCCNTR from counting at EL2.

HCCD	Meaning
`0b0`	Cycle counting by PMCCNTR is not affected by this bit.
`0b1`	Cycle counting by PMCCNTR is prohibited at EL2.

This bit does not affect the CPU_CYCLES event or any other event that counts cycles.

On a Warm reset, in a system where the PE resets into EL2 or EL3, this field resets to `0`.

Otherwise:

Reserved, RES0.

Bits [22:20]

Reserved, RES0.

TTRF, bit [19]

When ARMv8.4-Trace is implemented:

Traps use of the Trace Filter Control registers at EL1 to EL2.

TTRF	Meaning
`0b0`	Accesses to TRFCR at EL1 are not affected by this control bit.
`0b1`	Accesses to TRFCR at EL1 generate a Hyp Trap exception.

On a Warm reset, in a system where the PE resets into EL2 or EL3, this field resets to `0`.

Otherwise:

Reserved, RES0.

Bit [18]

Reserved, RES0.

HPMD, bit [17]

When ARMv8.1-PMU is implemented:

Guest Performance Monitors Disable. This control prohibits event counting at EL2.

HPMD	Meaning
0b0	Event counting allowed in Hyp mode.
0b1	Event counting prohibited in Hyp mode. In an Armv8.1 implementation, event counting is prohibited unless enabled by the IMPLEMENTATION DEFINED authentication interface ExternalSecureNoninvasiveDebugEnabled().

This control applies only to:

- The event counters in the range [0:(HDCR.HPMN-1)].
- If PMCR.DP is set to 1, PMCCNTR.

The other event counters are unaffected. When PMCR.DP is set to 0, PMCCNTR is unaffected.

On a Warm reset, in a system where the PE resets into EL2 or EL3, this field resets to 0.

Otherwise:

Reserved, RES0.

Bits [16:12]

Reserved, RES0.

TDRA, bit [11]

Trap Debug ROM Address register access. Traps Non-secure EL0 and EL1 System register accesses to the Debug ROM registers to Hyp mode.

TDRA	Meaning
0b0	This control does not cause any instructions to be trapped.
0b1	Non-secure EL0 and EL1 System register accesses to the DBGDRAR or DBGDSAR are trapped to Hyp mode, unless it is trapped by DBGDSCRext.UDCCdis.

If HCR.TGE or HDCR.TDE is 1, behavior is as if this bit is 1 other than for the purpose of a direct read.

On a Warm reset, in a system where the PE resets into EL2 or EL3, this field resets to 0.

TDOSA, bit [10]

When ARMv8.0-DoubleLock is implemented:

Trap debug OS-related register access. Traps Non-secure EL1 System register accesses to the powerdown debug registers to Hyp mode.

TDOSA	Meaning
0b0	This control does not cause any instructions to be trapped.
0b1	Non-secure EL1 System register accesses to the powerdown debug registers are trapped to Hyp mode.

The registers for which accesses are trapped are as follows:

- DBGOSLSR, DBGOSLAR, DBGOSDLR, and DBGPRCR.
- Any IMPLEMENTATION DEFINED register with similar functionality that the implementation specifies as trapped by this bit.

Note

These registers are not accessible at EL0.

If HCR.TGE or HDCR.TDE is 1, behavior is as if this bit is 1 other than for the purpose of a direct read.

On a Warm reset, in a system where the PE resets into EL2 or EL3, this field resets to 0.

Otherwise:

Trap debug OS-related register access. Traps Non-secure EL1 System register accesses to the powerdown debug registers to Hyp mode.

TDOSA	Meaning
0b0	This control does not cause any instructions to be trapped.
0b1	Non-secure EL1 System register accesses to the powerdown debug registers are trapped to Hyp mode.

The registers for which accesses are trapped are as follows:

- DBGOSLSR, DBGOSLAR, and DBGPRCR.
- Any IMPLEMENTATION DEFINED register with similar functionality that the implementation specifies as trapped by this bit.

It is IMPLEMENTATION DEFINED whether accesses to DBGOSDLR are trapped.

Note

These registers are not accessible at EL0.

If HCR.TGE or HDCR.TDE is 1, behavior is as if this bit is 1 other than for the purpose of a direct read.

On a Warm reset, in a system where the PE resets into EL2 or EL3, this field resets to 0.

TDA, bit [9]

Trap debug access. Traps Non-secure EL0 and EL1 System register accesses to those debug System registers in the (coproc==0b1110) encoding space that are not trapped by either of the following:

- HDCR.TDRA.
- HDCR.TDOSA.

TDA	Meaning
0b0	This control does not cause any instructions to be trapped.
0b1	Non-secure EL0 or EL1 System register accesses to the debug registers, other than the registers trapped by HDCR.TDRA and HDCR.TDOSA, are trapped to Hyp mode, unless it is trapped by DBGDSCRext.UDCCdis.

Traps of AArch32 accesses to DBGDTRRXint and DBGDTRTXint are ignored in Debug state.

If HCR.TGE or HDCR.TDE is 1, behavior is as if this bit is 1 other than for the purpose of a direct read.

On a Warm reset, in a system where the PE resets into EL2 or EL3, this field resets to 0.

TDE, bit [8]

Trap Debug exceptions. The possible values of this bit are:

TDE	Meaning
0b0	This control has no effect on the routing of debug exceptions, and has no effect on Non-secure accesses to debug registers.
0b1	Debug exceptions generated at EL1 or EL0 are routed to EL2 when enabled in the current Security state. The HDCR.{TDRA, TDOSA, TDA} fields are treated as being 1 for all purposes other than returning the result of a direct read of the register.

When HCR.TGE == 1, the PE behaves as if the value of this field is 1 for all purposes other than returning the value of a direct read of the register.

On a Warm reset, in a system where the PE resets into EL2 or EL3, this field resets to 0.

HPME, bit [7]

When PMUv3 is implemented:

[HDCR.HPMN:(N-1)] event counters enable.

HPME	Meaning
0b0	Event counters in the range [HDCR.HPMN:(PMCR.N-1)] are disabled.
0b1	Event counters in the range [HDCR.HPMN:(PMCR.N-1)] are enabled by PMCNTENSET.

If HDCR.HPMN is less than PMCR.N, the event counters in the range [HDCR.HPMN:(PMCR.N-1)], are enabled and disabled by this bit. Otherwise this bit has no effect on the operation of the event counters.

Note

The effect of HDCR.HPMN on the operation of this bit applies regardless of whether EL2 is enabled in the current Security state.

For more information see the description of the HPMN field.

On a Warm reset, in a system where the PE resets into EL2 or EL3, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TPM, bit [6]

When PMUv3 is implemented:

Trap Performance Monitors accesses. Traps Non-secure EL0 and EL1 accesses to all Performance Monitors registers to Hyp mode.

TPM	Meaning
0b0	This control does not cause any instructions to be trapped.
0b1	Non-secure EL0 and EL1 accesses to all Performance Monitors registers are trapped to Hyp mode.

Note

EL2 does not provide traps on Performance Monitor register accesses through the optional memory-mapped external debug interface.

On a Warm reset, in a system where the PE resets into EL2 or EL3, this field resets to 0.

Otherwise:

Reserved, RES0.

TPMCR, bit [5]

When PMUv3 is implemented:

Trap PMCR accesses. Traps Non-secure EL0 and EL1 accesses to the PMCR to Hyp mode.

TPMCR	Meaning
0b0	This control does not cause any instructions to be trapped.
0b1	Non-secure EL0 and EL1 accesses to the PMCR are trapped to Hyp mode, unless it is trapped by PMUSERENR.EN.

Note

EL2 does not provide traps on Performance Monitor register accesses through the optional memory-mapped external debug interface.

On a Warm reset, in a system where the PE resets into EL2 or EL3, this field resets to `0`.

Otherwise:

Reserved, RES0.

HPMN, bits [4:0]

When PMUv3 is implemented:

Defines the number of event counters that are accessible from Non-secure EL1 modes, and from Non-secure EL0 modes if unprivileged access is enabled.

If HPMN is less than PMCR.N, HPMN divides the event counters into two ranges, [0:(HPMN-1)] and [HPMN:(PMCR.N-1)].

For an event counter in the range [0:(HPMN-1)]:

- The counter is accessible from EL1 and EL2, and from EL0 if unprivileged access to the counters is enabled.
- If ARMv8.5-PMU is implemented, PMCR.LP determines whether the counter overflows at PMEVCNTR<n>[31:0] or PMEVCNTR<n>[63:0].
- PMCR.E enables the operation of counters in this range.

Note

If HPMN is equal to PMCR.N, this applies to all event counters.

If HPMN is less than PMCR.N, for an event counter in the range [HPMN:(PMCR.N-1)]:

- The counter is accessible only from EL2 and from Secure state.
- If ARMv8.5-PMU is implemented, HDCR.HLP determines whether the counter overflows at PMEVCNTR<n>[31:0] or PMEVCNTR<n>[63:0].
- HDCR.HPME enables the operation of counters in this range.

If this field is set to 0, or to a value larger than PMCR.N, then the following CONSTRAINED UNPREDICTABLE behavior applies:

- The value returned by a direct read of HDCR.HPMN is UNKNOWN.
- Either:
  - An UNKNOWN number of counters are reserved for EL2 use. That is, the PE behaves as if HDCR.HPMN is set to an UNKNOWN non-zero value less than or equal to PMCR.N.
  - All counters are reserved for EL2 use, meaning no counters are accessible from Non-secure EL1 and Non-secure EL0.

On a Warm reset, in a system where the PE resets into EL2 or EL3, this field resets to the value in PMCR.N.

Otherwise:

Reserved, RES0.

Accessing the HDCR

Accesses to this register use the following encodings:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
~~opc1~~	~~opc2~~	~~CRn~~	~~coproc~~	~~CRm~~
0b1111	0b100	0b0001	0b0001	0b001
~~0b100~~	~~0b001~~	~~0b0001~~	~~0b1111~~	~~0b0001~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T1 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T1 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        return HDCR;
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        UNDEFINED;
    else
        return HDCR;
```

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
~~opc1~~	~~opc2~~	~~CRn~~	~~coproc~~	~~CRm~~
0b1111	0b100	0b0001	0b0001	0b001
~~0b100~~	~~0b001~~	~~0b0001~~	~~0b1111~~	~~0b0001~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T1 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T1 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        HDCR = R[t];
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        UNDEFINED;
    else
        HDCR = R[t];
```

27~~13~~/03~~12~~/2019~~2018~~ 21~~16~~:59~~42~~: e5e4db499bf9867a4b93324c4dbac985d3da93766379d01e197f1d40720d32d0f84e419e9187e009

(old) htmldiff from- (new)

HDFAR, Hyp Data Fault Address Register

The HDFAR characteristics are:

Purpose

Holds the virtual address of the faulting address that caused a synchronous Data Abort exception that is taken to Hyp mode.

Configuration

AArch32 System register HDFAR bits [31:0] are architecturally mapped to AArch64 System register FAR_EL2[31:0] .

AArch32 System register HDFAR bits [31:0] are architecturally mapped to AArch32 System register DFAR[31:0] (S) when HaveEL(EL2), HaveEL(EL3) and HighestELUsingAArch32().

If EL2 is not implemented, this register is RES0 from EL3.

RW fields in this register reset to architecturally UNKNOWN values.

Attributes

HDFAR is a 32-bit register.

Field descriptions

The HDFAR bit assignments are:

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
VA of faulting address of synchronous Data Abort exception taken to Hyp mode																															

Bits [31:0]

VA of faulting address of synchronous Data Abort exception taken to Hyp mode.

On a Prefetch Abort exception, this register is UNKNOWN.

Any execution in a Non-secure EL1 or Non-secure EL0 mode makes this register UNKNOWN.

This field resets to an architecturally UNKNOWN value.

Accessing the HDFAR

Accesses to this register use the following encodings:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
~~opc1~~	~~opc2~~	~~CRn~~	~~coproc~~	~~CRm~~
0b1111	0b100	0b0110	0b0000	0b000
~~0b100~~	~~0b000~~	~~0b0110~~	~~0b1111~~	~~0b0000~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T6 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T6 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    return HDFAR;
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        UNDEFINED;
    else
        return HDFAR;
```

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
~~opc1~~	~~opc2~~	~~CRn~~	~~coproc~~	~~CRm~~
0b1111	0b100	0b0110	0b0000	0b000
~~0b100~~	~~0b000~~	~~0b0110~~	~~0b1111~~	~~0b0000~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T6 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T6 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    HDFAR = R[t];
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        UNDEFINED;
    else
        HDFAR = R[t];
```

27~~13~~/03~~12~~/2019~~2018~~ 21~~16~~:59~~42~~; e5e4db499bf9867a4b93324c4dbac985d3da9376~~6379d01e197f1d40720d32d0f84e419e9187e009~~

(old) htmldiff from- (new)

HIFAR, Hyp Instruction Fault Address Register

The HIFAR characteristics are:

Purpose

Holds the virtual address of the faulting address that caused a synchronous Prefetch Abort exception that is taken to Hyp mode.

Configuration

AArch32 System register HIFAR bits [31:0] are architecturally mapped to AArch64 System register FAR_EL2[63:32] .

AArch32 System register HIFAR bits [31:0] are architecturally mapped to AArch32 System register IFAR[31:0] (S) when HaveEL(EL2), HaveEL(EL3) and HighestELUsingAArch32().~~)~~.

If EL2 is not implemented, this register is RES0 from EL3.

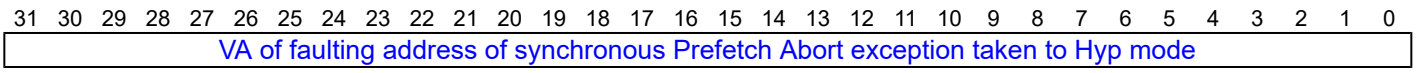
RW fields in this register reset to architecturally UNKNOWN values.

Attributes

HIFAR is a 32-bit register.

Field descriptions

The HIFAR bit assignments are:



Bits [31:0]

VA of faulting address of synchronous Prefetch Abort exception taken to Hyp mode.

On a Data Abort exception, this register is UNKNOWN.

Any execution in a Non-secure EL1 or Non-secure EL0 mode makes this register UNKNOWN.

This field resets to an architecturally UNKNOWN value.

Accessing the HIFAR

Accesses to this register use the following encodings:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
~~opc1~~	~~opc2~~	~~CRn~~	~~coproc~~	~~CRm~~
0b1111	0b100	0b0110	0b0000	0b010
~~0b100~~	~~0b010~~	~~0b0110~~	~~0b1111~~	~~0b0000~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T6 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T6 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    return HIFAR;
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        UNDEFINED;
    else
        return HIFAR;
```

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
~~opc1~~	~~opc2~~	~~CRn~~	~~coproc~~	~~CRm~~
0b1111	0b100	0b0110	0b0000	0b010
~~0b100~~	~~0b010~~	~~0b0110~~	~~0b1111~~	~~0b0000~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T6 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T6 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    HIFAR = R[t];
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        UNDEFINED;
    else
        HIFAR = R[t];
```

27~~13~~/03~~12~~/2019~~2018~~ 21~~16~~:59~~42~~; e5e4db499bf9867a4b93324c4dbac985d3da9376~~6379d01e197f1d40720d32d0f84e419e9187e009~~

(old) htmldiff from- (new)

ICC_MSRE, Interrupt Controller Monitor System Register Enable register

The ICC_MSRE characteristics are:

Purpose

Controls whether the System register interface or the memory-mapped interface to the GIC CPU interface is used for EL3.

Configuration

AArch32 System register ICC_MSRE bits [31:0] can be mapped to AArch64 System register ICC_SRE_EL3[31:0] , but this is not architecturally mandated.

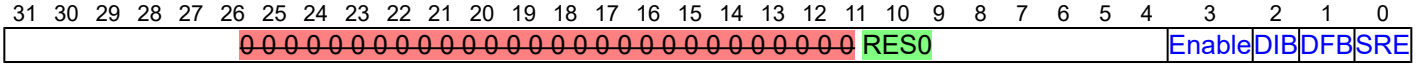
Some or all RW fields of this register have defined reset values. These apply only if the PE resets into an Exception level that is using AArch32. Otherwise, RW fields in this register reset to architecturally UNKNOWN values.

Attributes

ICC_MSRE is a 32-bit register.

Field descriptions

The ICC_MSRE bit assignments are:



Bits [31:4]

Reserved, RES0.

Enable, bit [3]

Enable. Enables lower Exception level access to ICC_SRE and ICC_HSRE.

Enable	Meaning
0b0	Secure EL1 accesses to Secure ICC_SRE trap to EL3. EL2 accesses to Non-secure ICC_SRE and ICC_HSRE trap to EL3. Non-secure EL1 accesses to ICC_SRE trap to EL3, unless these accesses are trapped to EL2 as a result of ICC_HSRE.Enable == 0.
0b1	Secure EL1 accesses to Secure ICC_SRE do not trap to EL3. EL2 accesses to Non-secure ICC_SRE and ICC_HSRE do not trap to EL3. Non-secure EL1 accesses to ICC_SRE do not trap to EL3.

If ICC_MSRE.SRE is RAO/WI, an implementation is permitted to make the Enable bit RAO/WI.

If ICC_MSRE.SRE is 0, the Enable bit behaves as 1 for all purposes other than reading the value of the bit.

This field resets to an architecturally UNKNOWN value.

DIB, bit [2]

Disable IRQ bypass.

DIB	Meaning
0b0	IRQ bypass enabled.
0b1	IRQ bypass disabled.

In systems that do not support IRQ bypass, this bit is RAO/WI.

This field resets to 0.

DFB, bit [1]

Disable FIQ bypass.

DFB	Meaning
0b0	FIQ bypass enabled.
0b1	FIQ bypass disabled.

In systems that do not support FIQ bypass, this bit is RAO/WI.

This field resets to 0.

SRE, bit [0]

System Register Enable.

SRE	Meaning
0b0	The memory-mapped interface must be used. Accesses at EL3 or below to any ICH_* System register, or any EL1, EL2, or EL3 ICC_* register other than ICC_SRE, ICC_HSRE, or ICC_MSRE, are UNDEFINED.
0b1	The System register interface to the ICH_* registers and the EL1, EL2, and EL3 ICC_* registers is enabled for EL3.

If software changes this bit from 1 to 0, the results are UNPREDICTABLE.

If an implementation supports only a System register interface to the GIC CPU interface, this bit is RAO/WI.

This field resets to 0.

Accessing the ICC_MSRE

This register is always System register accessible.

The GIC architecture permits, but does not require, that registers can be shared between memory-mapped registers and the equivalent System registers. This means that if the memory-mapped registers have been accessed while ICC_MSRE.SRE==0, then the System registers might be modified. Therefore, software must only rely on the reset values of the System registers if there has been no use of the GIC functionality while the memory-mapped registers are in use. Otherwise, the System register values must be treated as UNKNOWN.

This register is only accessible when executing in Monitor mode.

Accesses to this register use the following encodings:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
~~opc1~~	~~opc2~~	~~CRn~~	~~coproc~~	~~CRm~~
0b1111	0b110	0b1100	0b1100	0b101
~~0b110~~	~~0b101~~	~~0b1100~~	~~0b1111~~	~~0b1100~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T12 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T12 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    return ICC_MSRE;
```

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
~~opc1~~	~~opc2~~	~~CRn~~	~~coproc~~	~~CRm~~
0b1111	0b110	0b1100	0b1100	0b101
~~0b110~~	~~0b101~~	~~0b1100~~	~~0b1111~~	~~0b1100~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T12 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T12 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' && CP15SDISABLE2 == HIGH then
        UNDEFINED;
    else
        ICC_MSRE = R[t];
```

27~~13~~/03~~12~~/2019~~2018~~ 21~~16~~:59~~42~~; e5e4db499bf9867a4b93324c4dbac985d3da93766~~379d01e197f1d40720d32d0f84e419e9187e009~~

(old) htmldiff from- (new)

(old) htmldiff from- (new)

ICH_LRC<n>, Interrupt Controller List Registers, n = 0 - 15

The ICH_LRC<n> characteristics are:

Purpose

Provides interrupt context information for the virtual CPU interface.

Configuration

AArch32 System register ICH_LRC<n> bits [31:0] are architecturally mapped to AArch64 System register ICH_LR<n>_EL2[63:32] .

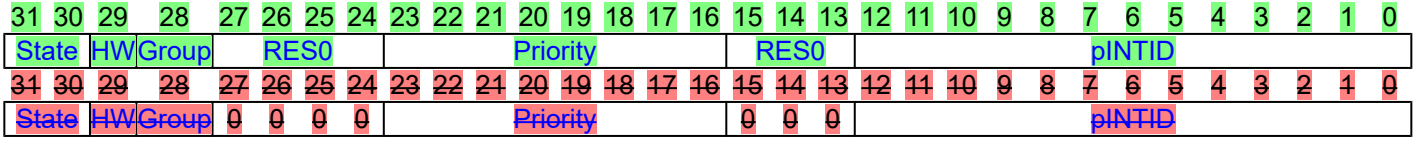
Some or all RW fields of this register have defined reset values. These apply only if the PE resets into an Exception level that is using AArch32. Otherwise, RW fields in this register reset to architecturally UNKNOWN values.

Attributes

ICH_LRC<n> is a 32-bit register.

Field descriptions

The ICH_LRC<n> bit assignments are:



State, bits [31:30]

The state of the interrupt:

State	Meaning
0b00	Invalid (Inactive).
0b01	Pending.
0b10	Active.
0b11	Pending and active.

The GIC updates these state bits as virtual interrupts proceed through the interrupt life cycle. Entries in the invalid state are ignored, except for the purpose of generating virtual maintenance interrupts.

For hardware interrupts, the pending and active state is held in the physical Distributor rather than the virtual CPU interface. A hypervisor must only use the pending and active state for software originated interrupts, which are typically associated with virtual devices, or SGIs.

This field resets to 0.

HW, bit [29]

Indicates whether this virtual interrupt maps directly to a hardware interrupt, meaning that it corresponds to a physical interrupt. Deactivation of the virtual interrupt also causes the deactivation of the physical interrupt with the INTID that the pINTID field indicates.

HW	Meaning
0b0	The interrupt is triggered entirely by software. No notification is sent to the Distributor when the virtual interrupt is deactivated.
0b1	The interrupt maps directly to a hardware interrupt. A deactivate interrupt request is sent to the Distributor when the virtual interrupt is deactivated, using the pINTID field from this register to indicate the physical INTID. If ICH_VMCR.VEOIM is 0, this request corresponds to a write to ICC_EOIR0 or ICC_EOIR1. Otherwise, it corresponds to a write to ICC_DIR.

This field resets to 0.

Group, bit [28]

Indicates the group for this virtual interrupt.

Group	Meaning
0b0	This is a Group 0 virtual interrupt. ICH_VMCR.VFIQEn determines whether it is signaled as a virtual IRQ or as a virtual FIQ, and ICH_VMCR.VENG0 enables signaling of this interrupt to the virtual machine.
0b1	This is a Group 1 virtual interrupt, signaled as a virtual IRQ. ICH_VMCR.VENG1 enables the signaling of this interrupt to the virtual machine. If ICH_VMCR.VCBPR is 0, then ICC_BPR1 determines if a pending Group 1 interrupt has sufficient priority to preempt current execution. Otherwise, ICH_LR<n> determines preemption.

This field resets to 0.

Bits [27:24]

Reserved, RES0.

Priority, bits [23:16]

The priority of this interrupt.

It is IMPLEMENTATION DEFINED how many bits of priority are implemented, though at least five bits must be implemented. Unimplemented bits are RES0 and start from bit[16] up to bit[18]. The number of implemented bits can be discovered from ICH_VTR.PRIbits.

This field resets to 0.

Bits [15:13]

Reserved, RES0.

pINTID, bits [12:0]

Physical INTID, for hardware interrupts.

When ICH_LRC<n>.HW is 0 (there is no corresponding physical interrupt), this field has the following meaning:

- Bits[12:10] : RES0.
- Bit[9] : EOI. If this bit is 1, then when the interrupt identified by vINTID is deactivated, an EOI maintenance interrupt is asserted.
- Bits[8:0] : Reserved, RES0.

When ICH_LRC<n>.HW is 1 (there is a corresponding physical interrupt):

- This field indicates the physical INTID. This field is only required to implement enough bits to hold a valid value for the implemented INTID size. Any unused higher order bits are RES0.
- When ICC_CTLR_EL1.ExtRange is 0, then bits[44:42] of this field are RES0.
- If the value of pINTID is not a valid INTID, behavior is UNPREDICTABLE. If the value of pINTID indicates a PPI, this field applies to the PPI associated with this same physical PE ID as the virtual CPU interface requesting the deactivation.

A hardware physical identifier is only required in List Registers for interrupts that require deactivation. This means only 13 bits of Physical INTID are required, regardless of the number specified by ICC_CTLR.IDbits.

This field resets to 0.

Accessing the ICH_LRC<n>

ICH_LR<n> and ICH_LRC<n> can be updated independently.

Accesses to this register use the following encodings:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
~~opc1~~	~~opc2~~	~~CRn~~	~~coproc~~	~~CRm~~
0b1111	0b100	0b1100	0b111[n:3]	0b[n:2:0]
~~0b100~~	~~0b[n:2:0]~~	~~0b1100~~	~~0b1111~~	~~0b111[n:3]~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T12 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T12 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ICC_HSRE.SRE == '0' then
        UNDEFINED;
    else
        return ICH_LRC~~ICH_AP0R~~[UInt(CRm<0>:opc2<2:0>)];
elsif PSTATE.EL == EL3 then
    if ICC_MSRE.SRE == '0' then
        UNDEFINED;
    else
        return ICH_LRC~~ICH_AP0R~~[UInt(CRm<0>:opc2<2:0>)];
```

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
~~opc1~~	~~opc2~~	~~CRn~~	~~coproc~~	~~CRm~~
0b1111	0b100	0b1100	0b111[n:3]	0b[n:2:0]
~~0b100~~	~~0b[n:2:0]~~	~~0b1100~~	~~0b1111~~	~~0b111[n:3]~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T12 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T12 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ICC_HSRE.SRE == '0' then
        UNDEFINED;
    else
        ICH_LRC~~ICH_AP0R~~[UInt(CRm<0>:opc2<2:0>)] = R[t];
elsif PSTATE.EL == EL3 then
    if ICC_MSRE.SRE == '0' then
        UNDEFINED;
    else
        ICH_LRC~~ICH_AP0R~~[UInt(CRm<0>:opc2<2:0>)] = R[t];
```

2713/0312/20192018 2116:5942; e5e4db499bf9867a4b93324c4dbac985d3da93766379d01e197f1d40720d32d0f84e419e9187e009

(old) htmldiff from- (new)

(old) htmldiff from- (new)

ICH_LR<n>, Interrupt Controller List Registers, n = 0 - 15

The ICH_LR<n> characteristics are:

Purpose

Provides interrupt context information for the virtual CPU interface.

Configuration

AArch32 System register ICH_LR<n> bits [31:0] are architecturally mapped to AArch64 System register ICH_LR<n>_EL2[31:0] .

If EL2 is not implemented, this register is RES0 from EL3.

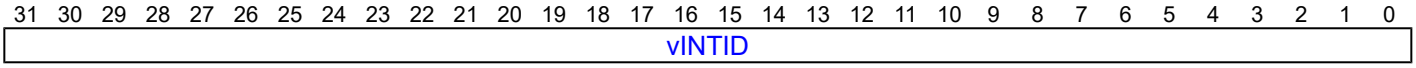
Some or all RW fields of this register have defined reset values. These apply only if the PE resets into an Exception level that is using AArch32. Otherwise, RW fields in this register reset to architecturally UNKNOWN values.

Attributes

ICH_LR<n> is a 32-bit register.

Field descriptions

The ICH_LR<n> bit assignments are:



vINTID, bits [31:0]

Virtual INTID of the interrupt.

If the value of vINTID is 1020-1023 and ICH_LRC<n>.State!=0b00 (Inactive), behavior is UNPREDICTABLE.

Behavior is UNPREDICTABLE if two or more List Registers specify the same vINTID when:

- ICH_LRC<n>.State == 01.
- ICH_LRC<n>.State == 10.
- ICH_LRC<n>.State == 11.

It is IMPLEMENTATION DEFINED how many bits are implemented, though at least 16 bits must be implemented. Unimplemented bits are RES0. The number of implemented bits can be discovered from ICH_VTR.IDbits.

Note

When a VM is using memory-mapped access to the GIC, software must ensure that the correct source PE ID is provided in bits[12:10].

This field resets to 0.

Accessing the ICH_LR<n>

ICH_LR<n> and ICH_LRC<n> can be updated independently.

Accesses to this register use the following encodings:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
~~opc1~~	~~opc2~~	~~CRn~~	~~coproc~~	~~CRm~~
0b1111	0b100	0b1100	0b110[n:3]	0b[n:2:0]
~~0b100~~	~~0b[n:2:0]~~	~~0b1100~~	~~0b1111~~	~~0b110[n:3]~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T12 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T12 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ICC_HSRE.SRE == '0' then
        UNDEFINED;
    else
        return ICH_LR[UInt(CRm<0>:opc2<2:0>)];
elsif PSTATE.EL == EL3 then
    if ICC_MSRE.SRE == '0' then
        UNDEFINED;
    else
        return ICH_LR[UInt(CRm<0>:opc2<2:0>)];
```

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
~~opc1~~	~~opc2~~	~~CRn~~	~~coproc~~	~~CRm~~
0b1111	0b100	0b1100	0b110[n:3]	0b[n:2:0]
~~0b100~~	~~0b[n:2:0]~~	~~0b1100~~	~~0b1111~~	~~0b110[n:3]~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T12 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T12 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ICC_HSRE.SRE == '0' then
        UNDEFINED;
    else
        ICH_LR[UInt(CRm<0>:opc2<2:0>)] = R[t];
elsif PSTATE.EL == EL3 then
    if ICC_MSRE.SRE == '0' then
        UNDEFINED;
    else
        ICH_LR[UInt(CRm<0>:opc2<2:0>)] = R[t];
```

27~~13~~/03~~12~~/2019~~2018~~ 21~~16~~:59~~42~~; e5e4db499bf9867a4b93324c4dbac985d3da9376~~6379d01e197f1d40720d32d0f84c419e9187e009~~

(old) htmldiff from- (new)

(old) htmldiff from- (new)

ID_MMFR4, Memory Model Feature Register 4

The ID_MMFR4 characteristics are:

Purpose

Provides information about the implemented memory model and memory management support in AArch32 state.

Must be interpreted with ID_MMFR0, ID_MMFR1, ID_MMFR2, and ID_MMFR3.

For general information about the interpretation of the ID registers see 'Principles of the ID scheme for fields in ID registers' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section G7.1.3.

Configuration

AArch32 System register ID_MMFR4 bits [31:0] are architecturally mapped to AArch64 System register ID_MMFR4_EL1[31:0] .

Attributes

ID_MMFR4 is a 32-bit register.

Field descriptions

The ID_MMFR4 bit assignments are:

31 30 29 28	27 26 25 24	23 22 21 20	19 18 17 16	15 14 13 12	11 10 9 8	7 6 5 4	3 2 1 0
EVTEVT–Enhanced Virtualization Traps	CCIDX	LSM	HPDS	CnP	XNX	AC2	SpecSEI

EVT–Enhanced Virtualization Traps, bits [31:28]

WhenFrom ARMv8.2-EVT is implementedArmv8.5:

Enhanced Virtualization Traps. If EL2 is implemented, indicates support for the TICAB, TOCU and TID4 traps. Defined values are: HCR2.{TTLBIS, TOCU, TICAB, TID4} traps. Defined values are:

EVT–Enhanced Virtualization Traps	Meaning
0b0000	HCR2.{TTLBISHCR2.TICAB, TOCUHCR2.TOCU, TICAB, TID4}HCR2.TID4 traps are not supported.
0b0001	HCR2.{TOCU, TICAB, TID4} traps are supported. HCR2.TTLBISHCR2.TICAB, HCR2.TOCU, HCR2.TID4 traps are supported. HCR2.TTLBIS trap is not supported
0b0010	HCR2.{TTLBISHCR2.TICAB, TOCUHCR2.TOCU, TICABHCR2.TID4, TID4}HCR2.TTLBIS traps are supported.

All other values are reserved.

In Armv8.0, the only permitted value is 0b0000.

From Armv8.1, the permitted values are 0b0000, 0b0001, and 0b0010.

From Armv8.5, the permitted values are:

- 0b0000 when EL2 is not implemented.
- 0b0010 when EL2 is implemented. This feature is identified as ARMv8.2-EVT.

Otherwise:

Reserved, RES0.

CCIDX, bits [27:24]

From Armv8.3:

Support for use of the revised CCSIDR format and the presence of the CCSIDR2 is indicated. Defined values are:

CCIDX	Meaning
0b0000	32-bit format implemented for all levels of the CCSIDR, and the CCSIDR2 register is not implemented.
0b0001	64-bit format implemented for all levels of the CCSIDR, and the CCSIDR2 register is implemented.

All other values are reserved.

From Armv8.3, the permitted values are 0b0000 and 0b0001. This feature is identified as ARMv8.3-CCIDX.

Otherwise:

Reserved, RAZ.

LSM, bits [23:20]

From Armv8.2:

Indicates support for LSMAOE and nTLSMD bits in HSCTLR and SCTLR. Defined values are:

LSM	Meaning
0b0000	LSMAOE and nTLSMD bits not supported.
0b0001	LSMAOE and nTLSMD bits supported.

All other values are reserved.

ARMv8.2-LSMAOC implements the functionality identified by the value 0b0001.

Otherwise:

Reserved, RAZ.

HPDS, bits [19:16]

From Armv8.2:

Hierarchical permission disables bits in translation tables. Defined values are:

HPDS	Meaning
0b0000	Disabling of hierarchical controls not supported.
0b0001	Supports disabling of hierarchical controls using the TTBCR2.HPD0, TTBCR2.HPD1, and HTCR.HPD bits.
0b0010	As for value 0b0001, and adds possible hardware allocation of bits[62:59] of the translation table descriptors from the final lookup level for IMPLEMENTATION DEFINED use.

All other values are reserved.

ARMv8.2-AA32HPD implements the functionality identified by the value 0b0001.

ARMv8.2-TTPBHA implements the functionality added by the value 0b0010.

Note

The value 0b0000 implies that the encoding for TTBCR2 is unallocated.

Otherwise:

Reserved, RAZ.

CnP, bits [15:12]

From Armv8.2:

Common not Private translations. Defined values are:

CnP	Meaning
0b0000	Common not Private translations not supported.
0b0001	Common not Private translations supported.

All other values are reserved.

ARMv8.2-TTCNP implements the functionality identified by the value 0b0001.

From Armv8.2 the only permitted value is 0b0001.

Otherwise:

Reserved, RAZ.

XNX, bits [11:8]

From Armv8.2:

Support for execute-never control distinction by Exception level at stage 2. Defined values are:

XNX	Meaning
0b0000	Distinction between EL0 and EL1 execute-never control at stage 2 not supported.
0b0001	Distinction between EL0 and EL1 execute-never control at stage 2 supported.

All other values are reserved.

ARMv8.2-TTS2UXN implements the functionality identified by the value 0b0001.

When ARMv8.2-TTS2UXN is implemented:

- If all of the following conditions are true it is IMPLEMENTATION DEFINED whether the value of ID_MMFR4.XNX is 0b0000 or 0b0001:
  - ID_AA64MMFR1_EL1.XNX ==1.
  - EL2 cannot use AArch32.
  - EL1 can use AArch32.
- If EL2 can use AArch32 then the only permitted value is 0b0001.

Otherwise:

Reserved, RAZ.

AC2, bits [7:4]

Indicates the extension of the ACTLR and HACTLR registers using ACTLR2 and HACTLR2. Defined values are:

AC2	Meaning
0b0000	ACTLR2 and HACTLR2 are not implemented.
0b0001	ACTLR2 and HACTLR2 are implemented.

All other values are reserved.

In Armv8.0 and Armv8.1 the permitted values are 0b0000 and 0b0001.

From Armv8.2, the only permitted value is 0b0001.

SpecSEI, bits [3:0]

When RAS is implemented:

Describes whether the PE can generate SError interrupt exceptions from speculative reads of memory, including speculative instruction fetches. The defined values of this field are:

SpecSEI	Meaning
0b0000	The PE never generates an SError interrupt due to an External abort on a speculative read.
0b0001	The PE might generate an SError interrupt due to an External abort on a speculative read.

All other values are reserved.

Otherwise:

Reserved, RES0. This provides no information about whether the PE generates a speculative SError interrupt.

Accessing the ID_MMFR4

Accesses to this register use the following encodings:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
~~opc1~~	~~opc2~~	~~CRn~~	~~coproc~~	~~CRm~~
0b1111	0b000	0b0000	0b0010	0b110
~~0b000~~	~~0b110~~	~~0b0000~~	~~0b1111~~	~~0b0010~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T0 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T0 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && (!IsZero(ID_MMFR4) || boolean
IMPLEMENTATION_DEFINED "ID_MMFR4_EL1 trapped by HCR_EL2.TID3 and ID_MMFR4 trapped by HCR_EL2.TID3
and HCR.TID3") && HCR_EL2.TID3 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TID3 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        return ID_MMFR4;
elsif PSTATE.EL == EL2 then
    return ID_MMFR4;
elsif PSTATE.EL == EL3 then
    return ID_MMFR4;
```

2713/0312/20192018 2116:5942: e5e4db499bf9867a4b93324c4dbac985d3da93766379d01e197f1d40720d32d0f84c419e9187e009

(old) htmldiff from- (new)

(old) htmldiff from- (new)

IFAR, Instruction Fault Address Register

The IFAR characteristics are:

Purpose

Holds the virtual address of the faulting address that caused a synchronous Prefetch Abort exception.

Configuration

AArch32 System register IFAR bits [31:0] ~~(NS)~~ are architecturally mapped to AArch64 System register FAR_EL1[63:32] .

AArch32 System register IFAR bits [31:0] (S) are architecturally mapped to AArch32 System register HIFAR[31:0] when HaveEL(EL2), HaveEL(EL3) and HighestELUsingAArch32().~~)~~.

AArch32 System register IFAR bits [31:0] (S) are architecturally mapped to AArch64 System register FAR_EL2[63:32] when HaveEL(EL2).

RW fields in this register reset to architecturally UNKNOWN values.

Attributes

IFAR is a 32-bit register.

Field descriptions

The IFAR bit assignments are:

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
VA of faulting address of synchronous Prefetch Abort exception																															

Bits [31:0]

VA of faulting address of synchronous Prefetch Abort exception.

This field resets to an architecturally UNKNOWN value.

Accessing the IFAR

Accesses to this register use the following encodings:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
~~opc1~~	~~opc2~~	~~CRn~~	~~coproc~~	~~CRm~~
0b1111	0b000	0b0110	0b0000	0b010
~~0b000~~	~~0b010~~	~~0b0110~~	~~0b1111~~	~~0b0000~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T6 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T6 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TRVM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TRVM == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) then
        if SCR.NS == '0' then
            return IFAR_S;
        else
            return IFAR_NS;
    else
        return IFAR;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && ELUsingAArch32(EL3) then
        return IFAR_NS;
    else
        return IFAR;
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        return IFAR_S;
    else
        return IFAR_NS;
```

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
~~opc1~~	~~opc2~~	~~CRn~~	~~coproc~~	~~CRm~~
0b1111	0b000	0b0110	0b0000	0b010
~~0b000~~	~~0b010~~	~~0b0110~~	~~0b1111~~	~~0b0000~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T6 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T6 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TVM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TVM == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) then
        if SCR.NS == '0' then
            IFAR_S = R[t];
        else
            IFAR_NS = R[t];
    else
        IFAR = R[t];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && ELUsingAArch32(EL3) then
        IFAR_NS = R[t];
    else
        IFAR = R[t];
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        IFAR_S = R[t];
    else
        IFAR_NS = R[t];
```

2713/0312/20192018 2116:5942; e5e4db499bf9867a4b93324c4dbac985d3da93766379d01e197f1d40720d32d0f84c419e9187c009

(old) htmldiff from- (new)

(old) htmldiff from- (new)

ITLBIALL, Instruction TLB Invalidate All

The ITLBIALL characteristics are:

Purpose

Invalidate all cached copies of translation table entries from instruction TLBs that are from any level of the translation table walk. The entries that are invalidated are as follows:

- If executed at Secure EL1 when EL3 is using AArch64, all entries that would be required for the Secure EL1&0 translation regime.
- If executed in Secure state when EL3 is using AArch32, all entries that would be required for the Secure PL1&0 translation regime.
- If executed at Non-secure EL1, all stage 1 translation table entries that would be required for the Non-secure PL1&0 translation regime and, if EL2 is implemented, they must match the current VMID.
- If executed at EL2, the stage 1 or stage 2 translation table entries that would be required for the Non-secure PL1&0 translation regime and matches the current VMID.

The invalidation only applies to the PE that executes this System instruction.

Arm deprecates the use of this System instruction. It is only provided for backwards compatibility with earlier versions of the Arm architecture.

Configuration

Attributes

ITLBIALL is a 32-bit System instruction.

Field descriptions

ITLBIALL ignores the value in the register specified by the instruction encoding. Software does not have to write a value to the register before issuing this instruction.

Executing the ITLBIALL instruction

Accesses to this instruction use the following encodings:

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
~~opc1~~	~~opc2~~	~~CRn~~	~~coproc~~	~~CRm~~
0b1111	0b000	0b1000	0b0101	0b000
~~0b000~~	~~0b000~~	~~0b1000~~	~~0b1111~~	~~0b0101~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T8 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T8 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TTLB == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR.TTLB == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        ITLBIALL();
elsif PSTATE.EL == EL2 then
    ITLBIALL();
elsif PSTATE.EL == EL3 then
    ITLBIALL();
```

27~~13~~/03~~12~~/2019~~2018~~ 21~~16~~:59~~42~~; e5e4db499bf9867a4b93324c4dbac985d3da9376~~6379d01e197f1d40720d32d0f84e419e9187e009~~

(old) htmldiff from- (new)

(old) htmldiff from- (new)

ITLBIASID, Instruction TLB Invalidate by ASID match

The ITLBIASID characteristics are:

Purpose

Invalidate all cached copies of translation table entries from instruction TLBs that meet the following requirements:

- The entry is a stage 1 translation table entry.
- The entry would be used for the specified ASID, and either:
  - Is from a level of lookup above the final level.
  - Is a non-global entry from the final level of lookup.
- If EL2 is implemented, the entry would be used with the current VMID.

From the entries that match these requirement, the entries that are invalidated are required for the following translation regime:

- If executed at Secure EL1 when EL3 is using AArch64, the Secure EL1&0 translation regime.
- If executed in Secure state when EL3 is using AArch32, the Secure PL1&0 translation regime.
- If executed in Non-secure state, the Non-secure PL1&0 translation regime.

The invalidation only applies to the PE that executes this System instruction.

Arm deprecates the use of this System instruction. It is only provided for backwards compatibility with earlier versions of the Arm architecture.

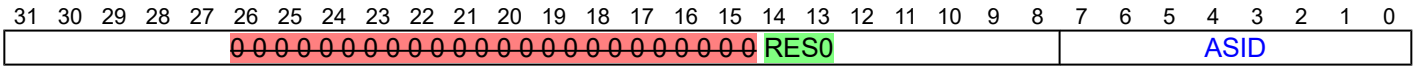
Configuration

Attributes

ITLBIASID is a 32-bit System instruction.

Field descriptions

The ITLBIASID input value bit assignments are:



Bits [31:8]

Reserved, RES0.

ASID, bits [7:0]

ASID value to match. Any TLB entries for non-global pages that match the ASID values will be affected by this System instruction.~~operation.~~

Executing the ITLBIASID instruction

Accesses to this instruction use the following encodings:

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
~~opc1~~	~~opc2~~	~~CRn~~	~~coproc~~	~~CRm~~
0b1111	0b000	0b1000	0b0101	0b010

0b000	0b010	0b1000	0b1111	0b0101

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T8 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T8 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TTLB == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR.TTLB == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        ITLBIASID(R[t]);
elsif PSTATE.EL == EL2 then
    ITLBIASID(R[t]);
elsif PSTATE.EL == EL3 then
    ITLBIASID(R[t]);
```

2713/0312/20192018 2116:5942; e5e4db499bf9867a4b93324c4dbac985d3da93766379d01e197f1d40720d32d0f84e419e9187e009

(old) htmldiff from- (new)

(old) htmldiff from- (new)

ITLBIMVA, Instruction TLB Invalidate by VA

The ITLBIMVA characteristics are:

Purpose

Invalidate all cached copies of translation table entries from instruction TLBs that meet the following requirements:

- The entry is a stage 1 translation table entry.
- The entry would be used to translate the specified address, and one of the following applies:
  - The entry is from a level of lookup above the final level and matches the specified ASID.
  - The entry is a global entry from the final level of lookup.
  - The entry is a non-global entry from the final level of lookup that matches the specified ASID.
- If EL2 is implemented, the entry would be used with the current VMID.

From the entries that match these requirement, the entries that are invalidated are required for the following translation regime:

- If executed at Secure EL1 when EL3 is using AArch64, the Secure EL1&0 translation regime.
- If executed in Secure state when EL3 is using AArch32, the Secure PL1&0 translation regime.
- If executed in Non-secure state, the Non-secure PL1&0 translation regime.

The invalidation only applies to the PE that executes this System instruction.

Arm deprecates the use of this System instruction. It is only provided for backwards compatibility with earlier versions of the Arm architecture.

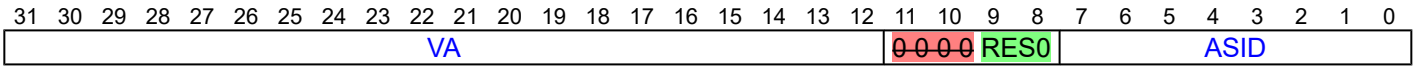
Configuration

Attributes

ITLBIMVA is a 32-bit System instruction.

Field descriptions

The ITLBIMVA input value bit assignments are:



VA, bits [31:12]

Virtual address to match. Any TLB entries that match the ASID value and VA value will be affected by this System instruction.~~operation.~~

Bits [11:8]

Reserved, RES0.

ASID, bits [7:0]

ASID value to match. Any TLB entries that match the ASID value and VA value will be affected by this System instruction.~~operation.~~

Global TLB entries that match the VA value will be affected by this operation, regardless of the value of the ASID field.

Executing the ITLBIMVA instruction

Accesses to this instruction use the following encodings:

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
~~opc1~~	~~opc2~~	~~CRn~~	~~coproc~~	~~CRm~~
0b1111	0b000	0b1000	0b0101	0b001
~~0b000~~	~~0b001~~	~~0b1000~~	~~0b1111~~	~~0b0101~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T8 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T8 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TTLB == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR.TTLB == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        ITLBIASID(R[t]);
elsif PSTATE.EL == EL2 then
    ITLBIASID(R[t]);
elsif PSTATE.EL == EL3 then
    ITLBIASID(R[t]);
```

27~~13~~/03~~12~~/2019~~2018~~ 21~~16~~:59~~42~~; e5e4db499bf9867a4b93324c4dbac985d3da93766~~379d01e197f1d40720d32d0f84e419e9187e009~~

(old) htmldiff from- (new)

(old) htmldiff from- (new)

JIDR, Jazelle ID Register

The JIDR characteristics are:

Purpose

A Jazelle register, which identified the Jazelle architecture version.

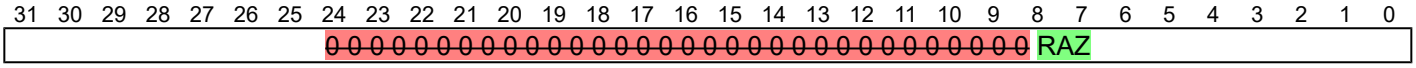
Configuration

Attributes

JIDR is a 32-bit register.

Field descriptions

The JIDR bit assignments are:



Bits [31:0]

Reserved, RAZ.

~~RO, RAZ at EL1, EL2, and EL3. It is IMPLEMENTATION DEFINED whether this field is RAZ or UNDEFINED at EL0.~~

Accessing the JIDR

Accesses to this register use the following encodings:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
~~opc1~~	~~opc2~~	~~CRn~~	~~coproc~~	~~CRm~~
0b1110	0b111	0b0000	0b0000	0b000
~~0b111~~	~~0b000~~	~~0b0000~~	~~0b1110~~	~~0b0000~~

```
if PSTATE.EL == EL0 then
    if boolean IMPLEMENTATION_DEFINED "JIDR UNDEFINED at EL0" then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> != '11' && HCR_EL2.TID0 == '1'
then
        AArch64.AArch32SystemAccessTrap(EL2, 0x05);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TID0 == '1' then
        AArch32.TakeHypTrapException(0x05);
    else
        return JIDR;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TID0 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x05);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TID0 == '1' then
        AArch32.TakeHypTrapException(0x05);
    else
        return JIDR;
elsif PSTATE.EL == EL2 then
    return JIDR;
elsif PSTATE.EL == EL3 then
    return JIDR;
```

27~~13~~/03~~12~~/20192018 21~~16~~:59~~42~~; e5e4db499bf9867a4b93324c4dbac985d3da93766379d01e197f1d40720d32d0f84e419e9187e009

(old) htmldiff from- (new)

(old) htmldiff from- (new)

JMCR, Jazelle Main Configuration Register

The JMCR characteristics are:

Purpose

A Jazelle register, which provides control of the Jazelle extension.

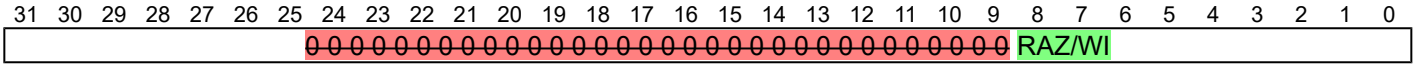
Configuration

Attributes

JMCR is a 32-bit register.

Field descriptions

The JMCR bit assignments are:



Bits [31:0]

Reserved, RAZ/WI.

~~RAZ/WI at EL1, EL2, and EL3. It is IMPLEMENTATION DEFINED whether this field is RAZ/WI or UNDEFINED at EL0.~~

Accessing the JMCR

For accesses from EL0 it is IMPLEMENTATION DEFINED whether the register is RW or UNDEFINED.

Accesses to this register use the following encodings:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
~~opc1~~	~~opc2~~	~~CRn~~	~~coproc~~	~~CRm~~
0b1110	0b111	0b0010	0b0000	0b000
~~0b111~~	~~0b000~~	~~0b0010~~	~~0b1110~~	~~0b0000~~

```
if PSTATE.EL == EL0 then
    if boolean IMPLEMENTATION_DEFINED "JMCR UNDEFINED at EL0" then
        UNDEFINED;
    else
        return JMCR;
elsif PSTATE.EL == EL1 then
    return JMCR;
elsif PSTATE.EL == EL2 then
    return JMCR;
elsif PSTATE.EL == EL3 then
    return JMCR;
```

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
~~opc1~~	~~opc2~~	~~CRn~~	~~coproc~~	~~CRm~~
0b1110	0b111	0b0010	0b0000	0b000
~~0b111~~	~~0b000~~	~~0b0010~~	~~0b1110~~	~~0b0000~~

```
if PSTATE.EL == EL0 then
    if boolean IMPLEMENTATION_DEFINED "JMCR UNDEFINED at EL0" then
        UNDEFINED;
    else
        //no operation
elsif PSTATE.EL == EL1 then
    //no operation
elsif PSTATE.EL == EL2 then
    //no operation
elsif PSTATE.EL == EL3 then
    //no operation
```

27~~13~~/03~~12~~/2019~~2018~~ 21~~16~~:59~~42~~; e5e4db499bf9867a4b93324c4dbac985d3da93766~~379d01e197f1d40720d32d0f84e419e9187e009~~

(old) htmldiff from- (new)

(old) htmldiff from- (new)

JOSCR, Jazelle OS Control Register

The JOSCR characteristics are:

Purpose

A Jazelle register, which provides operating system control of the Jazelle Extension.

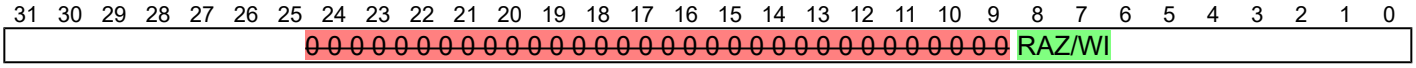
Configuration

Attributes

JOSCR is a 32-bit register.

Field descriptions

The JOSCR bit assignments are:



Bits [31:0]

Reserved, RAZ/WI.

~~RAZ/WI at EL1, EL2, and EL3. It is IMPLEMENTATION DEFINED whether this field is RAZ/WI or UNDEFINED at EL0.~~

Accessing the JOSCR

Accesses to this register use the following encodings:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
~~opc1~~	~~opc2~~	~~CRn~~	~~coproc~~	~~CRm~~
0b1110	0b111	0b0001	0b0000	0b000
~~0b111~~	~~0b000~~	~~0b0001~~	~~0b1110~~	~~0b0000~~

```
if PSTATE.EL == EL0 then
    if boolean IMPLEMENTATION_DEFINED "JOSCR UNDEFINED at EL0" then
        UNDEFINED;
    else
        return JOSCR;
elsif PSTATE.EL == EL1 then
    return JOSCR;
elsif PSTATE.EL == EL2 then
    return JOSCR;
elsif PSTATE.EL == EL3 then
    return JOSCR;
```

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
~~opc1~~	~~opc2~~	~~CRn~~	~~coproc~~	~~CRm~~
0b1110	0b111	0b0001	0b0000	0b000
~~0b111~~	~~0b000~~	~~0b0001~~	~~0b1110~~	~~0b0000~~

```
if PSTATE.EL == EL0 then
    if boolean IMPLEMENTATION_DEFINED "JOSCR UNDEFINED at EL0" then
        UNDEFINED;
    else
        //no operation
elsif PSTATE.EL == EL1 then
    //no operation
elsif PSTATE.EL == EL2 then
    //no operation
elsif PSTATE.EL == EL3 then
    //no operation
```

27~~13~~/03~~12~~/2019~~2018~~ 21~~16~~:59~~42~~: e5e4db499bf9867a4b93324c4dbac985d3da93766379d01e197f1d40720d32d0f84e419e9187e009

(old) htmldiff from- (new)

MIDR, Main ID Register

The MIDR characteristics are:

Purpose

Provides identification information for the PE, including an implementer code for the device and a device ID number.

Configuration

AArch32 System register MIDR bits [31:0] are architecturally mapped to AArch64 System register MIDR_EL1[31:0] .

AArch32 System register MIDR bits [31:0] are architecturally mapped to External register MIDR_EL1[31:0] .

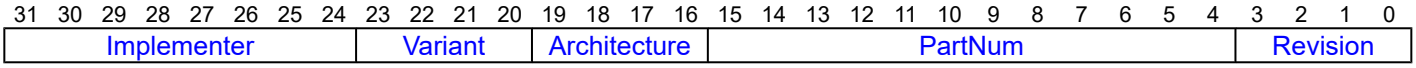
Some fields of the MIDR are IMPLEMENTATION DEFINED. For details of the values of these fields for a particular Armv8 implementation, and any implementation-specific significance of these values, see the product documentation.

Attributes

MIDR is a 32-bit register.

Field descriptions

The MIDR bit assignments are:



Implementer, bits [31:24]

The Implementer code. This field must hold an implementer code that has been assigned by Arm. Assigned codes include the following:

Hex representation	Implementer
0x00~~0x0~~	Reserved for software use
0xC0	Ampere Computing
0x41	Arm Limited
0x42	Broadcom Corporation
0x43	Cavium Inc.
0x44	Digital Equipment Corporation
0x49	Infineon Technologies AG
0x4D	Motorola or Freescale Semiconductor Inc.
0x4E	NVIDIA Corporation
0x50	Applied Micro Circuits Corporation
0x51	Qualcomm Inc.
0x56	Marvell International Ltd.
0x69	Intel Corporation

Arm can assign codes that are not published in this manual. All values not assigned by Arm are reserved and must not be used.

Variant, bits [23:20]

An IMPLEMENTATION DEFINED variant number. Typically, this field is used to distinguish between different product variants, or major revisions of a product.

Architecture, bits [19:16]

The permitted values of this field are:

Architecture	Meaning
0b0001	Armv4.
0b0010	Armv4T.
0b0011	Armv5 (obsolete).
0b0100	Armv5T.
0b0101	Armv5TE.
0b0110	Armv5TEJ.
0b0111	Armv6.
0b1111	Architectural features are individually identified in the ID_* registers, see 'ID registers' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section K12.5.3.

All other values are reserved.

PartNum, bits [15:4]

An IMPLEMENTATION DEFINED primary part number for the device.

On processors implemented by Arm, if the top four bits of the primary part number are 0x0 or 0x7, the variant and architecture are encoded differently.

Revision, bits [3:0]

An IMPLEMENTATION DEFINED revision number for the device.

Accessing the MIDR

Accesses to this register use the following encodings:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
~~opc1~~	~~opc2~~	~~CRn~~	~~coproc~~	~~CRm~~
0b1111	0b000	0b0000	0b0000	0b000
~~0b000~~	~~0b000~~	~~0b0000~~	~~0b1111~~	~~0b0000~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T0 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T0 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) then
        return VPIDR_EL2<31:0>;
    elsif EL2Enabled() && ELUsingAArch32(EL2) then
        return VPIDR;
    else
        return MIDR;
elsif PSTATE.EL == EL2 then
    return MIDR;
elsif PSTATE.EL == EL3 then
    return MIDR;
```

(old) htmldiff from- (new)

(old) htmldiff from- (new)

MPIDR, Multiprocessor Affinity Register

The MPIDR characteristics are:

Purpose

In a multiprocessor system, provides an additional PE identification mechanism for scheduling purposes.

Configuration

AArch32 System register MPIDR bits [31:0] are architecturally mapped to AArch64 System register MPIDR_EL1[31:0] .

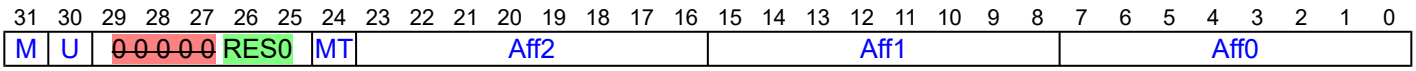
In a uniprocessor system Arm recommends that each Aff<n> field of this register returns a value of 0.

Attributes

MPIDR is a 32-bit register.

Field descriptions

The MPIDR bit assignments are:



M, bit [31]

Indicates whether this implementation includes the functionality introduced by the ARMv7 Multiprocessing Extensions. The possible values of this bit are:

M	Meaning
0b0	This implementation does not include the ARMv7 Multiprocessing Extensions functionality.
0b1	This implementation includes the ARMv7 Multiprocessing Extensions functionality.

In Armv8, this bit is RAO.

~~In Armv8 this bit is RES1.~~

U, bit [30]

Indicates a Uniprocessor system, as distinct from PE 0 in a multiprocessor system. The possible values of this bit are:

U	Meaning
0b0	Processor is part of a multiprocessor system.
0b1	Processor is part of a uniprocessor system.

Bits [29:25]

Reserved, RES0.

MT, bit [24]

Indicates whether the lowest level of affinity consists of logical PEs that are implemented using a multithreading type approach. See the description of Aff0 for more information about affinity levels. The possible values of this bit are:

MT	Meaning
0b0	Performance of PEs at the lowest affinity level is largely independent.
0b1	Performance of PEs at the lowest affinity level is very interdependent.

Aff2, bits [23:16]

Affinity level 2. See the description of Aff0 for more information.

Aff1, bits [15:8]

Affinity level 1. See the description of Aff0 for more information.

Aff0, bits [7:0]

Affinity level 0. This is the affinity level that is most significant for determining PE behavior. Higher affinity levels are increasingly less significant in determining PE behavior. The assigned value of the MPIDR.{Aff2, Aff1, Aff0} or MPIDR_EL1.{Aff3, Aff2, Aff1, Aff0} set of fields of each PE must be unique within the system as a whole.

Accessing the MPIDR

Accesses to this register use the following encodings:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
~~opc1~~	~~opc2~~	~~CRn~~	~~coproc~~	~~CRm~~
0b1111	0b000	0b0000	0b0000	0b101
~~0b000~~	~~0b101~~	~~0b0000~~	~~0b1111~~	~~0b0000~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T0 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T0 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) then
        return VMPIDR_EL2<31:0>;
    elsif EL2Enabled() && ELUsingAArch32(EL2) then
        return VMPIDR;
    else
        return MPIDR;
elsif PSTATE.EL == EL2 then
    return MPIDR;
elsif PSTATE.EL == EL3 then
    return MPIDR;
```

27~~13~~/03~~12~~/2019~~2018~~ 21~~16~~:59~~42~~; e5e4db499bf9867a4b93324c4dbac985d3da93766~~379d01e197f1d40720d32d0f84e419e9187e009~~

(old) htmldiff from- (new)

MVBAR, Monitor Vector Base Address Register

The MVBAR characteristics are:

Purpose

When EL3 is implemented and can use AArch32, holds the vector base address for any exception that is taken to Monitor mode.

Secure software must program the MVBAR with the required initial value as part of the PE boot sequence.

Configuration

It is IMPLEMENTATION DEFINED whether MVBAR[0] has a fixed value and ignored writes, or takes the last value written to it.

~~Write access to MVBAR is disabled when the CP15SDISABLE signal is asserted HIGH.~~

On a reset into EL3 using AArch32, the reset value of MVBAR is an IMPLEMENTATION DEFINED choice between:

- MVBAR[31:5] = an IMPLEMENTATION DEFINED value, which might be UNKNOWN.
- MVBAR[4:1] = RES0.
- MVBAR[0] = 0.

And:

- MVBAR[31:1] = an IMPLEMENTATION DEFINED value that is bits[31:1] of the AArch32 reset address.
- MVBAR[0] = 1.

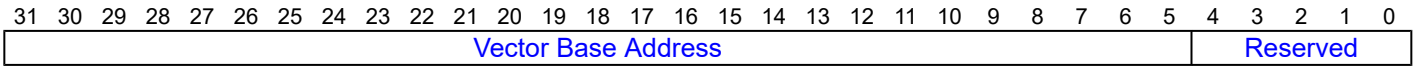
Attributes

MVBAR is a 32-bit register.

Field descriptions

The MVBAR bit assignments are:

When programmed with a vector base address:



Bits [31:5]

Vector Base Address. Bits[31:5] of the base address of the exception vectors for exceptions taken to this Exception level. Bits[4:0] of an exception vector are the exception offset.

Reserved, bits [4:0]

Reserved, see Configurations.

Accessing the MVBAR

Accesses to this register use the following encodings:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
~~opc1~~	~~opc2~~	~~CRn~~	~~coproc~~	~~CRm~~
0b1111	0b000	0b1100	0b0000	0b001
~~0b000~~	~~0b001~~	~~0b1100~~	~~0b1111~~	~~0b0000~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T12 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T12 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif !ELUsingAArch32(EL2) && SCR_EL3.<NS,EEL2> == '01' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif !ELUsingAArch32(EL3) && SCR_EL3.NS == '0' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    return MVBAR;
```

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
~~opc1~~	~~opc2~~	~~CRn~~	~~coproc~~	~~CRm~~
0b1111	0b000	0b1100	0b0000	0b001
~~0b000~~	~~0b001~~	~~0b1100~~	~~0b1111~~	~~0b0000~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T12 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T12 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif !ELUsingAArch32(EL2) && SCR_EL3.<NS,EEL2> == '01' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif !ELUsingAArch32(EL3) && SCR_EL3.NS == '0' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' && CP15SDISABLE == HIGH then
        UNDEFINED;
    elsif SCR.NS == '0' && CP15SDISABLE2 == HIGH then
        UNDEFINED;
    else
        MVBAR = R[t];
```

27~~13~~/03~~12~~/2019~~2018~~ 21~~16~~:59~~42~~; e5e4db499bf9867a4b93324c4dbac985d3da93766~~379d01e197f1d40720d32d0f84c419e9187c009~~

(old) htmldiff from- (new)

(old) htmldiff from- (new)

NMRR, Normal Memory Remap Register

The NMRR characteristics are:

Purpose

Provides additional mapping controls for memory regions that are mapped as Normal memory by their entry in the PRRR.

Used in conjunction with the PRRR.

Configuration

AArch32 System register NMRR bits [31:0] are architecturally mapped to AArch64 System register MAIR_EL1[63:32] when TTBCR.EAE == 0.

MAIR1 and NMRR are the same register, with a different view depending on the value of TTBCR.EAE:

- When it is set to 0, the register is as described in NMRR.
- When it is set to 1, the register is as described in MAIR1.

~~When EL3 is using AArch32, write access to NMRR(S) is disabled when the CP15SDISABLE signal is asserted HIGH.~~

RW fields in this register reset to architecturally UNKNOWN values.

Attributes

NMRR is a 32-bit register.

Field descriptions

The NMRR bit assignments are:

When TTBCR.EAE == 0:

31 30	29 28	27 26	25 24	23 22	21 20	19 18	17 16	15 14	13 12	11 10	9 8	7 6	5 4	3 2	1 0
OR7	OR6	OR5	OR4	OR3	OR2	OR1	OR0	IR7	IR6	IR5	IR4	IR3	IR2	IR1	IR0

OR<n>, bits [2n+17:2n+16], for n = 0 to 7

Outer Cacheable property mapping for memory attributes n, if the region is mapped as Normal memory by the PRRR.TR<n> entry. n is the value of the TEX[0], C, and B bits concatenated. The possible values of this field are:

OR<n>	Meaning
0b00	Region is Non-cacheable.
0b01	Region is Write-Back, Write-Allocate.
0b10	Region is Write-Through, no Write-Allocate.
0b11	Region is Write-Back, no Write-Allocate.

The meaning of the field with n = 6 is IMPLEMENTATION DEFINED and might differ from the meaning given here. This is because the meaning of the attribute combination {TEX[0] = 1, C = 1, B = 0} is IMPLEMENTATION DEFINED.

This field resets to an architecturally UNKNOWN value.

IR<n>, bits [2n+1:2n], for n = 0 to 7

Inner Cacheable property mapping for memory attributes n, if the region is mapped as Normal memory by the PRRR.TR<n> entry. n is the value of the TEX[0], C, and B bits concatenated. The possible values of this field are:

IR<n>	Meaning
0b00	Region is Non-cacheable.
0b01	Region is Write-Back, Write-Allocate.
0b10	Region is Write-Through, no Write-Allocate.
0b11	Region is Write-Back, no Write-Allocate.

The meaning of the field with n = 6 is IMPLEMENTATION DEFINED and might differ from the meaning given here. This is because the meaning of the attribute combination {TEX[0] = 1, C = 1, B = 0} is IMPLEMENTATION DEFINED.

This field resets to an architecturally UNKNOWN value.

Accessing the NMRR

Accesses to this register use the following encodings:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
~~opc1~~	~~opc2~~	~~CRn~~	~~coproc~~	~~CRm~~
0b1111	0b000	0b1010	0b0010	0b001
~~0b000~~	~~0b001~~	~~0b1010~~	~~0b1111~~	~~0b0010~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T10 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T10 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TRVM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TRVM == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) then
        if SCR.NS == '0' then
            return NMRR_S;
        else
            return NMRR_NS;
    else
        return NMRR;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && ELUsingAArch32(EL3) then
        return NMRR_NS;
    else
        return NMRR;
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        return NMRR_S;
    else
        return NMRR_NS;
```

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
~~opc1~~	~~opc2~~	~~CRn~~	~~coproc~~	~~CRm~~
0b1111	0b000	0b1010	0b0010	0b001
~~0b000~~	~~0b001~~	~~0b1010~~	~~0b1111~~	~~0b0010~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T10 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T10 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) && SCR.NS == '0' && CP15SDISABLE == HIGH then
        UNDEFINED;
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) && SCR.NS == '0' && CP15SDISABLE2 == HIGH then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TVM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TVM == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) then
        if SCR.NS == '0' then
            NMRR_S = R[t];
        else
            NMRR_NS = R[t];
    else
        NMRR = R[t];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && ELUsingAArch32(EL3) then
        NMRR_NS = R[t];
    else
        NMRR = R[t];
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' && CP15SDISABLE == HIGH then
        UNDEFINED;
    elsif SCR.NS == '0' && CP15SDISABLE2 == HIGH then
        UNDEFINED;
    else
        if SCR.NS == '0' then
            NMRR_S = R[t];
        else
            NMRR_NS = R[t];
```

2713/0312/20192018 2116:5942; e5e4db499bf9867a4b93324c4dbac985d3da93766379d01e197f1d40720d32d0f84e419e9187e009

(old) htmldiff from- (new)

NSACR, Non-Secure Access Control Register

The NSACR characteristics are:

Purpose

When EL3 is implemented and can use AArch32, defines the Non-secure access permissions to Trace, Advanced SIMD and floating-point functionality. Also includes IMPLEMENTATION DEFINED bits that can define Non-secure access permissions for IMPLEMENTATION DEFINED functionality.

Configuration

Note

In AArch64 state, the NSACR controls are replaced by controls in CPTR_EL3.

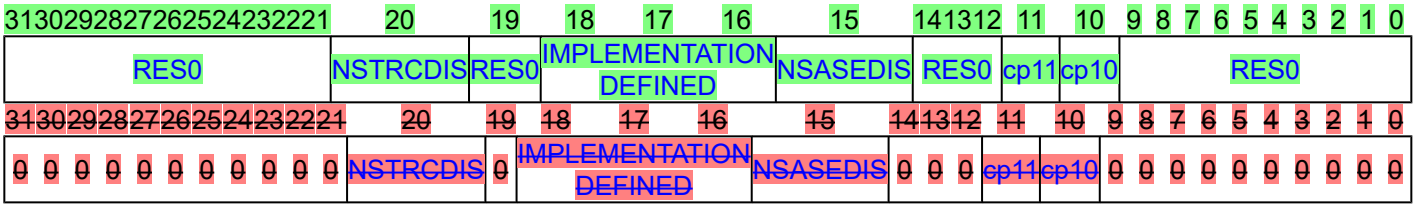
Some or all RW fields of this register have defined reset values. These apply whenever the register is accessible. This means they apply when the PE resets into EL3 using AArch32.

Attributes

NSACR is a 32-bit register.

Field descriptions

The NSACR bit assignments are:



If EL3 is implemented and is using AArch64 then:

- Any read of the NSACR from Non-secure EL2 or Non-secure EL1 returns a value of `0x00000C00`.
- Any read or write to NSACR from Secure EL1 is trapped as an exception to EL3.

If EL3 is not implemented, then any read of the NSACR from EL2 or EL1 returns a value of `0x00000C00`.

Bits [31:21]

Reserved, RES0.

NSTRCDIS, bit [20]

Disables Non-secure System register accesses to all implemented trace registers.

NSTRCDIS	Meaning
0b0	This control has no effect on: • System register access to implemented trace registers. • The behavior of CPACR.TRCDIS and HCPTR.TTA.
0b1	Non-secure System register accesses to all implemented trace registers are disabled, meaning: • CPACR.TRCDIS behaves as RAO/WI in Non-secure state, regardless of its actual value. • HCPTR.TTA behaves as RAO/WI, regardless of its actual value.

The implementation of this field must correspond to the implementation of the CPACR.TRCDIS field:

- If CPACR.TRCDIS is RAZ/WI, this field is RAZ/WI.
- If CPACR.TRCDIS is RW, this field is RW.

Note

- The ETMv4 architecture does not permit EL0 to access the trace registers. If the implementation includes an ETMv4 implementation, EL0 accesses to the trace registers are UNDEFINED.
- The architecture does not provide Non-secure access controls on trace register accesses through the optional memory-mapped external debug interface.

System register accesses to the trace registers can have side-effects. When a System register access is trapped, any side-effects that are normally associated with the access do not occur before the exception is taken.

In a system where the PE resets into EL3, this field resets to 0.

Bit [19]

Reserved, RES0.

IMPLEMENTATION DEFINED, bits [18:16]

IMPLEMENTATION DEFINED.

NSASEDIS, bit [15]

Disables Non-secure access to the Advanced SIMD functionality.

NSASEDIS	Meaning
0b0	This control has no effect on: • Non-secure access to Advanced SIMD functionality. • The behavior of CPACR.ASEDIS and HCPTR.TASE.
0b1	Non-secure access to the Advanced SIMD functionality is disabled, meaning: • CPACR.ASEDIS behaves as RAO/WI in Non-secure state, regardless of its actual value. • HCPTR.TASE behaves as RAO/WI, regardless of its actual value.

The implementation of this field must correspond to the implementation of the CPACR.ASEDIS field:

- If CPACR.ASEDIS is RES0, this field is RES0. If the implementation does not include Advanced SIMD and floating-point functionality, this field is RES0.
- If CPACR.ASEDIS is RAZ/WI, this field is RAZ/WI.
- If CPACR.ASEDIS is RW, this field is RW.

In a system where the PE resets into EL3, this field resets to 0.

Bits [14:12]

Reserved, RES0.

cp11, bit [11]

The value of this field is ignored. If this field is programmed with a different value to the cp10 field then this field is UNKNOWN on a direct read of the NSACR.

If the implementation does not include Advanced SIMD and floating-point functionality, this field is RES0.

In a system where the PE resets into EL3, this field resets to an architecturally UNKNOWN value.

cp10, bit [10]

Enable Non-secure access to the Advanced SIMD and floating-point features. Possible values of the fields are:

cp10	Meaning
0b0	Advanced SIMD and floating-point features can be accessed only from Secure state. Any attempt to access this functionality from Non-secure state is UNDEFINED. When the PE is in Non-secure state: • The CPACR.{cp11, cp10} fields ignore writes and read as 0b00, access denied. • The HCPTR.{TCP11, TCP10} fields behave as RAO/WI, regardless of their actual values.
0b1	Advanced SIMD and floating-point features can be accessed from both Security states.

If Non-secure access to the Advanced SIMD and floating-point functionality is enabled, the CPACR must be checked to determine the level of access that is permitted.

The Advanced SIMD and floating-point features controlled by these fields are:

- Execution of any floating-point or Advanced SIMD instruction.
- Any access to the Advanced SIMD and floating-point registers D0-D31 and their views as S0-S31 and Q0-Q15.
- Any access to the FPSCR, FPSID, MVFR0, MVFR1, MVFR2, or FPEXC System registers.

If the implementation does not include Advanced SIMD and floating-point functionality, this field is RES0.

In a system where the PE resets into EL3, this field resets to an architecturally UNKNOWN value.

Bits [9:0]

Reserved, RES0.

Accessing the NSACR

Accesses to this register use the following encodings:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
~~opc1~~	~~opc2~~	~~CRn~~	~~coproc~~	~~CRm~~
0b1111	0b000	0b0001	0b0001	0b010
~~0b000~~	~~0b010~~	~~0b0001~~	~~0b1111~~	~~0b0001~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T1 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T1 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif !ELUsingAArch32(EL2) && SCR_EL3.<NS,EEL2> == '01' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif !ELUsingAArch32(EL3) && SCR_EL3.NS == '0' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif !HaveEL(EL3) || (!ELUsingAArch32(EL3) && SCR_EL3.NS == '1') then
        return Zeros(20):'1100':Zeros(8);
    else
        return NSACR;
elsif PSTATE.EL == EL2 then
    if !HaveEL(EL3) || (!ELUsingAArch32(EL3) && SCR_EL3.NS == '1') then
        return Zeros(20):'1100':Zeros(8);
    else
        return NSACR;
elsif PSTATE.EL == EL3 then
    return NSACR;
```

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
~~opc1~~	~~opc2~~	~~CRn~~	~~coproc~~	~~CRm~~
0b1111	0b000	0b0001	0b0001	0b010
~~0b000~~	~~0b010~~	~~0b0001~~	~~0b1111~~	~~0b0001~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T1 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T1 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif !ELUsingAArch32(EL2) && SCR_EL3.<NS,EEL2> == '01' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif !ELUsingAArch32(EL3) && SCR_EL3.NS == '0' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' && CP15SDISABLE2 == HIGH then
        UNDEFINED;
    else
        NSACR = R[t];
```

27~~13~~/03~~12~~/2019~~2018~~ 21~~16~~:59~~42~~; e5e4db499bf9867a4b93324c4dbac985d3da9376~~6379d01e197f1d40720d32d0f84e419e9187e009~~

(old) htmldiff from- (new)

PMCCNTR, Performance Monitors Cycle Count Register

The PMCCNTR characteristics are:

Purpose

Holds the value of the processor Cycle Counter, CCNT, that counts processor clock cycles. See 'Time as measured by the Performance Monitors cycle counter' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile~~, section D5~~ for more information.

PMCCFILTR determines the modes and states in which the PMCCNTR can increment.

Configuration

AArch32 System register PMCCNTR bits [63:0] are architecturally mapped to AArch64 System register PMCCNTR_EL0[63:0] .

AArch32 System register PMCCNTR bits [63:0] are architecturally mapped to External register PMCCNTR_EL0[63:0] .

All counters are subject to any changes in clock frequency, including clock stopping caused by the WFI and WFE instructions. This means that it is CONSTRAINED UNPREDICTABLE whether or not PMCCNTR continues to increment when clocks are stopped by WFI and WFE instructions.

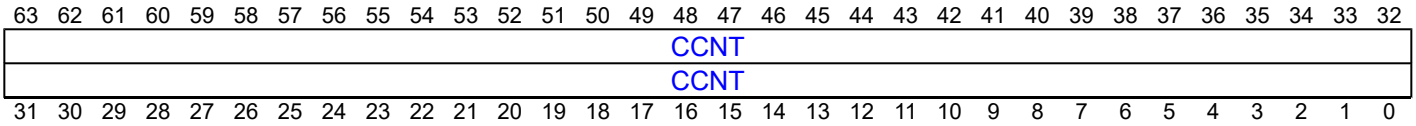
This register is in the Warm reset domain. On a Warm or Cold reset RW fields in this register reset to architecturally UNKNOWN values.

Attributes

PMCCNTR is a 64-bit register that can also be accessed as a 32-bit value. If it is accessed as a 32-bit register, accesses read and write bits [31:0] and do not modify bits [63:32].

Field descriptions

The PMCCNTR bit assignments are:



CCNT, bits [63:0]

Cycle count. Depending on the values of PMCR.{LC,D}, this field increments in one of the following ways:

- Every processor clock cycle.
- Every 64th processor clock cycle.

Writing 1 to PMCR.C sets this field to 0.

On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing the PMCCNTR

Accesses to this register use the following encodings:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
~~opc1~~	~~opc2~~	~~CRn~~	~~coproc~~	~~CRm~~

0b1111	0b000	0b1001	0b1101	0b000
~~0b000~~	~~0b000~~	~~0b1001~~	~~0b1111~~	~~0b1101~~

```
if PSTATE.EL == EL0 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> != '11' && HSTR_EL2.T9 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T9 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif !ELUsingAArch32(EL1) && PMUSERENR_EL0.<CR,EN> == '00' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x03);
    elsif ELUsingAArch32(EL1) && PMUSERENR.<CR,EN> == '00' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TGE == '1' then
            AArch32.TakeHypTrapException(0x00);
        else
            UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.TPM == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        return PMCCNTR<31:0>;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T9 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T9 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.TPM == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        return PMCCNTR<31:0>;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        return PMCCNTR<31:0>;
elsif PSTATE.EL == EL3 then
    return PMCCNTR<31:0>;
```

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
~~opc1~~	~~opc2~~	~~CRn~~	~~coproc~~	~~CRm~~
0b1111	0b000	0b1001	0b1101	0b000
~~0b000~~	~~0b000~~	~~0b1001~~	~~0b1111~~	~~0b1101~~

```
if PSTATE.EL == EL0 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> != '11' && HSTR_EL2.T9 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T9 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif !ELUsingAArch32(EL1) && PMUSERENR_EL0.EN == '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x03);
    elsif ELUsingAArch32(EL1) && PMUSERENR.EN == '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TGE == '1' then
            AArch32.TakeHypTrapException(0x00);
        else
            UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.TPM == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        PMCCNTR = ZeroExtend(R[t]);
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T9 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T9 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.TPM == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        PMCCNTR = ZeroExtend(R[t]);
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        PMCCNTR = ZeroExtend(R[t]);
elsif PSTATE.EL == EL3 then
    PMCCNTR = ZeroExtend(R[t]);
```

```
MRRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <Rt2>, <CRm>
```

coproc	CRm	opc1
~~opc1~~	~~coproc~~	~~CRm~~
0b1111	0b1001	0b0000
~~0b0000~~	~~0b1111~~	~~0b1001~~

```
if PSTATE.EL == EL0 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> != '11' && HSTR_EL2.T9 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T9 == '1' then
        AArch32.TakeHypTrapException(0x04);
    elsif !ELUsingAArch32(EL1) && PMUSERENR_EL0.<CR,EN> == '00' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x04);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x04);
    elsif ELUsingAArch32(EL1) && PMUSERENR.<CR,EN> == '00' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x04);
        elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TGE == '1' then
            AArch32.TakeHypTrapException(0x00);
        else
            UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.TPM == '1' then
        AArch32.TakeHypTrapException(0x04);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x04);
    else
        return PMCCNTR;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T9 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T9 == '1' then
        AArch32.TakeHypTrapException(0x04);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.TPM == '1' then
        AArch32.TakeHypTrapException(0x04);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x04);
    else
        return PMCCNTR;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x04);
    else
        return PMCCNTR;
elsif PSTATE.EL == EL3 then
    return PMCCNTR;
```

```
MCRR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <Rt2>, <CRm>
```

coproc	CRm	opc1
~~opc1~~	~~coproc~~	~~CRm~~
0b1111	0b1001	0b0000
~~0b0000~~	~~0b1111~~	~~0b1001~~

```
if PSTATE.EL == EL0 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> != '11' && HSTR_EL2.T9 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T9 == '1' then
        AArch32.TakeHypTrapException(0x04);
    elsif !ELUsingAArch32(EL1) && PMUSERENR_EL0.EN == '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x04);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x04);
    elsif ELUsingAArch32(EL1) && PMUSERENR.EN == '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x04);
        elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TGE == '1' then
            AArch32.TakeHypTrapException(0x00);
        else
            UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.TPM == '1' then
        AArch32.TakeHypTrapException(0x04);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x04);
    else
        PMCCNTR = R[t2]:R[t];
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T9 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T9 == '1' then
        AArch32.TakeHypTrapException(0x04);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.TPM == '1' then
        AArch32.TakeHypTrapException(0x04);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x04);
    else
        PMCCNTR = R[t2]:R[t];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x04);
    else
        PMCCNTR = R[t2]:R[t];
elsif PSTATE.EL == EL3 then
    PMCCNTR = R[t2]:R[t];
```

2713/0312/20192018 2116:5942; e5e4db499bf9867a4b93324c4dbac985d3da93766379d01e197f1d40720d32d0f84c419e9187e009

(old) htmldiff from- (new)

PMCR, Performance Monitors Control Register

The PMCR characteristics are:

Purpose

Provides details of the Performance Monitors implementation, including the number of counters implemented, and configures and controls the counters.

Configuration

AArch32 System register PMCR bits [31:0] are architecturally mapped to AArch64 System register PMCR_EL0[31:0] .

AArch32 System register PMCR bits [76:0] are architecturally mapped to External register PMCR_EL0[76:0] .

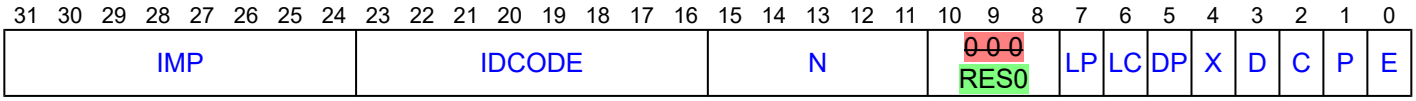
This register is in the Warm reset domain. Some or all RW fields of this register have defined reset values. On a Warm or Cold reset these apply only if the PE resets into an Exception level that is using AArch32. Otherwise, on a Warm or Cold reset RW fields in this register reset to architecturally UNKNOWN values.

Attributes

PMCR is a 32-bit register.

Field descriptions

The PMCR bit assignments are:



IMP, bits [31:24]

Implementer code. This field is RO with an IMPLEMENTATION DEFINED value.

If this field is zero, then PMCR.IDCODE is PMCR.IDCODE is RES0 and software must use the MIDR to identify the PE.

Otherwise this field and PMCR.IDCODE identify the PMU implementation to software. The implementer codes are allocated by Arm. A non-zero value has the same interpretation as MIDR.Implementer.

IDCODE, bits [23:16]

When PMCR.IMP != 0x00:

Identification code. This field is RO with an IMPLEMENTATION DEFINED value.

Each implementer must maintain a list of identification codes that areis specific to the implementer. A specific implementation is identified by the combination of the implementer code and the identification code.

Otherwise:

Reserved, RES0.

N, bits [15:11]

An RO field that indicates the number of event counters implemented. This value is in the range of `0b00000-0b111111`. If the value is `0b00000` then only PMCCNTR_EL0 is implemented. If the value is `0b111111` PMCCNTR_EL0 and 31 event counters are implemented.

In an implementation that includes EL2:

- If EL2 is using AArch32, reads of this field from Non-secure EL1 and Non-secure EL0 return the value of HDCR.HPMN.
- If EL2 is using AArch64 and enabled in the current Security state, reads of this field from EL1 and EL0 return the value of MDCR_EL2.HPMN.

Access to this field is **RO**.

Bits [10:8]

Reserved, RES0.

LP, bit [7]

When ARMv8.5-PMU is implemented:

Long event counter enable. Determines when unsigned overflow is recorded by a counter overflow bit.

LP	Meaning
`0b0`	Event counter overflow on increment that causes unsigned overflow of PMEVCNTR<n>[31:0].
`0b1`	Event counter overflow on increment that causes unsigned overflow of PMEVCNTR<n>[63:0].

If the highest implemented Exception level is using AArch32, it is IMPLEMENTATION DEFINED whether this bit is read/write or RAZ/WI.

If EL2 is implemented and HDCR.HPMN or MDCR_EL2.HPMN is less than PMCR.N, this bit does not affect the operation of event counters in the range [HDCR.HPMN:(PMCR.N-1)] or [MDCR_EL2.HPMN:(PMCR.N-1)].

PMEVCNTR<n>[63:32] cannot be accessed directly in AArch32 state.

Note

The effect of HDCR.HPMN or MDCR_EL2.HPMN on the operation of this bit applies if EL2 is implemented regardless of whether EL2 is enabled in the current Security state. For more information, see the description of HDCR.HPMN or MDCR_EL2.HPMN.

On a Warm reset, this field resets to `0`.

Otherwise:

Reserved, RES0.

LC, bit [6]

Long cycle counter enable. Determines when unsigned overflow is recorded by the cycle counter overflow bit.

LC	Meaning
`0b0`	Cycle counter overflow on increment that causes unsigned overflow of PMCCNTR[31:0].
`0b1`	Cycle counter overflow on increment that causes unsigned overflow of PMCCNTR[63:0].

Arm deprecates use of PMCR.LC = 0.

On a Warm reset, this field resets to an architecturally UNKNOWN value.

DP, bit [5]

Disable cycle counter when event counting is prohibited. The possible values of this bit are:

DP	Meaning
0b0	Cycle counting by PMCCNTR is, notif affectedenabled, bycounts thiswhen bit.event counting is prohibited.
0b1	When event counting for counters in the range [0..(HDCR.HPMN-1)] or [0..(MDCR_EL2.HPMN-1)] is prohibited, cycle counting by PMCCNTR does not count when event counting is disabled.prohibited.

Counting events is never prohibited in Non-secure state. However, there are some restrictions on counting events in Secure state. For more information about the interaction between the Performance Monitors and EL3, see 'EffectInteraction ofwith EL3 and EL2' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section D5.5.1

When EL3 is not implemented, this field is RES0:

- When ARMv8.1-PMU is not implemented.
- When ARMv8.1-PMU is implemented, only if EL2 is not implemented.

Otherwise this field is RW.

On a Warm reset, this field resets to 0.

X, bit [4]

Enable export of events in an IMPLEMENTATION DEFINED event stream. The possible values of this bit are:

X	Meaning
0b0	Do not export events.
0b1	Export events where not prohibited.

This field enables the exporting of events over an event bus to another device, for example to an OPTIONAL PE trace unit. If the implementation does not include such an event bus then this field is RAZ/WI, otherwise it is an RW field.

In an implementation that includes an event bus, no events are exported when counting is prohibited.

This field does not affect the generation of Performance Monitors overflow interrupt requests or signaling to a cross-trigger interface (CTI) that can be implemented as signals exported from the PE.

On a Warm reset, this field resets to 0.

D, bit [3]

Clock divider. The possible values of this bit are:

D	Meaning
0b0	When enabled, PMCCNTR counts every clock cycle.
0b1	When enabled, PMCCNTR counts once every 64 clock cycles.

This bit is RW.

If PMCR.LC == 1, this bit is ignored and the cycle counter counts every clock cycle.

Arm deprecates use of PMCR.D = 1.

On a Warm reset, this field resets to 0.

C, bit [2]

Cycle counter reset. This bit is WO. The effects of writing to this bit are:

C	Meaning
0b0	No action.
0b1	Reset PMCCNTR to zero.

This bit is always RAZ.

Note

Resetting PMCCNTR does not change the cycle counter overflow bit.

P, bit [1]

Event counter reset. This bit is WO. The effects of writing to this bit are:

P	Meaning
0b0	No action.
0b1	Reset all event counters accessible in the current Exception level, not including PMCCNTR, to zero.

This bit is always RAZ.

In EL0 and EL1:

- If EL2 is implemented and enabled in the current Security state, and HDCR.HPMN or MDCR_EL2.HPMN is less than PMCR_EL0.N, a write of 1 to this bit does not reset event counters in the range [HDCR.HPMN:(PMCR.N-1)] or [MDCR_EL2.HPMN:(PMCR.N-1)].
- If EL2 is not implemented, EL2 is disabled in the current Security state, or HDCR.HPMN or MDCR_EL2.HPMN is equal to PMCR_EL0.N, a write of 1 to this bit resets all the event counters.

In EL2 and EL3, a write of 1 to this bit resets all the event counters.

Note

Resetting the event counters does not change the event counter overflow bits.

If ARMv8.5-PMU is implemented, the value of HDCR.HLP, or PMCR.LP is ignored and bits [63:0] of all affected event counters are reset.

E, bit [0]

Enable.

E	Meaning
0b0	All event counters in the range [0:(PMN-1)] and PMCCNTR, are disabled.
0b1	All event counters in the range [0:(PMN-1)] and PMCCNTR, are enabled by PMCNTENSET.

This bit is RW.

If EL2 is implemented then:

- If EL2 is using AArch32, PMN is HDCR.HPMN.
- If EL2 is using AArch64, PMN is MDCR_EL2.HPMN.
- If PMN is less than PMCR.N, this bit does not affect the operation of event counters in the range [PMN:(PMCR.N-1)].

If EL2 is not implemented, PMN is PMCR.N.

Note

The effect of MDCR_EL2.HPMN or HDCR.HPMN on the operation of this bit applies if EL2 is implemented regardless of whether EL2 is enabled in the current Security state. For more information, see the description of MDCR_EL2.HPMN or HDCR.HPMN.

On a Warm reset, this field resets to 0.

Accessing the PMCR

Accesses to this register use the following encodings:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
~~opc1~~	~~opc2~~	~~CRn~~	~~coproc~~	~~CRm~~
0b1111	0b000	0b1001	0b1100	0b000
~~0b000~~	~~0b000~~	~~0b1001~~	~~0b1111~~	~~0b1100~~

```
if PSTATE.EL == EL0 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> != '11' && HSTR_EL2.T9 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T9 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif !ELUsingAArch32(EL1) && PMUSERENR_EL0.EN == '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x03);
    elsif ELUsingAArch32(EL1) && PMUSERENR.EN == '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TGE == '1' then
            AArch32.TakeHypTrapException(0x00);
        else
            UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPMCR == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.TPM == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.TPMCR == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        return PMCR;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T9 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T9 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPMCR == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.TPM == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.TPMCR == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        return PMCR;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        return PMCR;
elsif PSTATE.EL == EL3 then
    return PMCR;
```

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
~~opc1~~	~~opc2~~	~~CRn~~	~~coproc~~	~~CRm~~

0b1111	0b000	0b1001	0b1100	0b000
~~0b000~~	~~0b000~~	~~0b1001~~	~~0b1111~~	~~0b1100~~

```
if PSTATE.EL == EL0 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> != '11' && HSTR_EL2.T9 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T9 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif !ELUsingAArch32(EL1) && PMUSERENR_EL0.EN == '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x03);
    elsif ELUsingAArch32(EL1) && PMUSERENR.EN == '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TGE == '1' then
            AArch32.TakeHypTrapException(0x00);
        else
            UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPMCR == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.TPM == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.TPMCR == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        PMCR = R[t];
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T9 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T9 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPMCR == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.TPM == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.TPMCR == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        PMCR = R[t];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        PMCR = R[t];
elsif PSTATE.EL == EL3 then
    PMCR = R[t];
```

27~~13~~/03~~12~~/2019~~2018~~ 21~~16~~:59~~42~~; e5e4db499bf9867a4b93324c4dbac985d3da9376~~6379d01e197f1d40720d32d0f84e419e9187e009~~

(old) htmldiff from- (new)

PMEVCNTR<n>, Performance Monitors Event Count Registers, n = 0 - 30

The PMEVCNTR<n> characteristics are:

Purpose

Holds event counter n, which counts events, where n is 0 to 30.

Configuration

AArch32 System register PMEVCNTR<n> bits [31:0] are architecturally mapped to AArch64 System register PMEVCNTR<n>_EL0[31:0] .

AArch32 System register PMEVCNTR<n> bits [31:0] are architecturally mapped to External register PMEVCNTR<n>_EL0[31:0] .

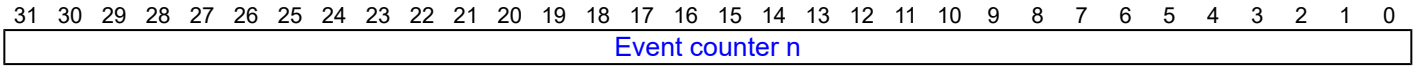
This register is in the Warm reset domain. On a Warm or Cold reset RW fields in this register reset to architecturally UNKNOWN values.

Attributes

PMEVCNTR<n> is a 32-bit register.

Field descriptions

The PMEVCNTR<n> bit assignments are:



Bits [31:0]

Event counter n. Value of event counter n, where n is the number of this register and is a number from 0 to 30.

If ARMv8.5-PMU is implemented, the event counter is 64 bits and only the least-significant part of the event counter is accessible in AArch32 state:

- Reads from PMEVCNTR<n> return bits [31:0] of the counter.
- Writes to PMEVCNTR<n> update bits [31:0] and leave bits [63:32] unchanged.
- There is no means to access bits [63:32] directly from AArch32 state.
- If the implementation does not support AArch64 at any Exception level, bits [63:32] are not required to be implemented.

If ARMv8.5-PMU is not implemented, the event counter is 32 bits.

On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing the PMEVCNTR<n>

PMEVCNTR<n> can also be accessed by using PMXEVCNTR with PMSELR.SEL set to the value of <n>.

If <n> is greater than or equal to the number of accessible counters, reads and writes of PMEVCNTR<n> are CONSTRAINED UNPREDICTABLE, and the following behaviors are permitted:

- Accesses to the register are UNDEFINED.
- Accesses to the register behave as RAZ/WI.
- Accesses to the register execute as a NOP

- If EL2 is implemented and enabled in the current Security state, and <n> is less than the number of implemented counters, accesses from EL1 or permitted accesses from EL0 are trapped to EL2.

Note

In EL0, an access is permitted if it is enabled by PMUSERENR.{ER,EN} or PMUSERER_EL0.{ER,EN}.

If EL2 is implemented and enabled in the current Security state, at EL0 and EL1:

- If EL2 is using AArch32, HDCR.HPMN identifies the number of accessible counters.
- If EL2 is using AArch64, MDCR_EL2.HPMN identifies the number of accessible counters.

Otherwise, the number of accessible counters is the number of implemented counters. See HDCR.HPMN and MDCR_EL2.HPMN for more details.

Accesses to this register use the following encodings:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
~~opc1~~	~~opc2~~	~~CRn~~	~~coproc~~	~~CRm~~
0b1111	0b000	0b1110	0b10[n:4:3]	0b[n:2:0]
~~0b000~~	~~0b[n:2:0]~~	~~0b1110~~	~~0b1111~~	~~0b10[n:4:3]~~

```
if PSTATE.EL == EL0 then
    if !ELUsingAArch32(EL1) && PMUSERENR_EL0.<ER,EN> == '00' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x03);
    elsif ELUsingAArch32(EL1) && PMUSERENR.<ER,EN> == '00' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TGE == '1' then
            AArch32.TakeHypTrapException(0x00);
        else
            UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.TPM == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        return PMEVCNTR[UInt(CRm<1:0>:opc2<2:0>)];
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.TPM == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        return PMEVCNTR[UInt(CRm<1:0>:opc2<2:0>)];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        return PMEVCNTR[UInt(CRm<1:0>:opc2<2:0>)];
elsif PSTATE.EL == EL3 then
    return PMEVCNTR[UInt(CRm<1:0>:opc2<2:0>)];
```

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
~~opc1~~	~~opc2~~	~~CRn~~	~~coproc~~	~~CRm~~
0b1111	0b000	0b1110	0b10[n:4:3]	0b[n:2:0]
~~0b000~~	~~0b[n:2:0]~~	~~0b1110~~	~~0b1111~~	~~0b10[n:4:3]~~

```
if PSTATE.EL == EL0 then
    if !ELUsingAArch32(EL1) && PMUSERENR_EL0.EN == '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x03);
    elsif ELUsingAArch32(EL1) && PMUSERENR.EN == '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TGE == '1' then
            AArch32.TakeHypTrapException(0x00);
        else
            UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.TPM == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        PMEVCNTR[UInt(CRm<1:0>:opc2<2:0>)] = R[t];
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.TPM == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        PMEVCNTR[UInt(CRm<1:0>:opc2<2:0>)] = R[t];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        PMEVCNTR[UInt(CRm<1:0>:opc2<2:0>)] = R[t];
elsif PSTATE.EL == EL3 then
    PMEVCNTR[UInt(CRm<1:0>:opc2<2:0>)] = R[t];
```

27~~13~~/03~~12~~/2019~~2018~~ 21~~16~~:59~~42~~; e5e4db499bf9867a4b93324c4dbac985d3da93766~~379d01e197f1d40720d32d0f84e419e9187e009~~

(old) htmldiff from- (new)

(old) htmldiff from- (new)

PMINTENCLR, Performance Monitors Interrupt Enable Clear register

The PMINTENCLR characteristics are:

Purpose

Disables the generation of interrupt requests on overflows from the Cycle Count Register, PMCCNTR, and the event counters PMEVCNTR<n>. Reading the register shows which overflow interrupt requests are enabled.

PMINTENCLR is used in conjunction with the PMINTENSET register.

Configuration

AArch32 System register PMINTENCLR bits [31:0] are architecturally mapped to AArch64 System register PMINTENCLR_EL1[31:0] .

AArch32 System register PMINTENCLR bits [31:0] are architecturally mapped to External register PMINTENCLR_EL1[31:0] .

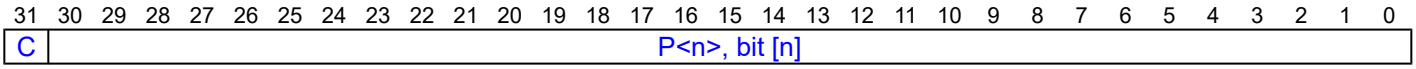
This register is in the Warm reset domain. On a Warm or Cold reset RW fields in this register reset to architecturally UNKNOWN values.

Attributes

PMINTENCLR is a 32-bit register.

Field descriptions

The PMINTENCLR bit assignments are:



C, bit [31]

PMCCNTR overflow interrupt request disable bit. Possible values are:

C	Meaning
0b0	When read, means the cycle counter overflow interrupt request is disabled. When written, has no effect.
0b1	When read, means the cycle counter overflow interrupt request is enabled. When written, disables the cycle count overflow interrupt request.

On a Warm reset, this field resets to an architecturally UNKNOWN value.

P<n>, bit [n], for n = 0 to 30

Event counter overflow interrupt request disable bit for PMEVCNTR<n>.

If N is less than 31, then bits [30:N] are RAZ/WI. When EL2 is implemented and enabled in the current Security state, in EL1 ~~and EL0~~, N is the value in MDCR_EL2.HPMN if EL2 is using AArch64, or in HDCR.HPMN if EL2 is using AArch32. Otherwise, N is the value in PMCR.N.

P<n>	Meaning
0b0	When read, means that the PMEVCNTR<n> event counter interrupt request is disabled. When written, has no effect.
0b1	When read, means that the PMEVCNTR<n> event counter interrupt request is enabled. When written, disables the PMEVCNTR<n> interrupt request.

On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing the PMINTENCLR

Accesses to this register use the following encodings:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
~~opc1~~	~~opc2~~	~~CRn~~	~~coproc~~	~~CRm~~
0b1111	0b000	0b1001	0b1110	0b010
~~0b000~~	~~0b010~~	~~0b1001~~	~~0b1111~~	~~0b1110~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T9 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T9 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.TPM == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        return PMINTENCLR;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        return PMINTENCLR;
elsif PSTATE.EL == EL3 then
    return PMINTENCLR;
```

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
~~opc1~~	~~opc2~~	~~CRn~~	~~coproc~~	~~CRm~~
0b1111	0b000	0b1001	0b1110	0b010
~~0b000~~	~~0b010~~	~~0b1001~~	~~0b1111~~	~~0b1110~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T9 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T9 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.TPM == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        PMINTENCLR = R[t];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        PMINTENCLR = R[t];
elsif PSTATE.EL == EL3 then
    PMINTENCLR = R[t];
```

27~~13~~/03~~12~~/20192018 21~~16~~:59~~42~~; e5e4db499bf9867a4b93324c4dbac985d3da93766379d01e197f1d40720d32d0f84e419e9187e009

(old) htmldiff from- (new)

(old) htmldiff from- (new)

PMINTENSET, Performance Monitors Interrupt Enable Set register

The PMINTENSET characteristics are:

Purpose

Enables the generation of interrupt requests on overflows from the Cycle Count Register, PMCCNTR, and the event counters PMEVCNTR<n>. Reading the register shows which overflow interrupt requests are enabled.

PMINTENSET is used in conjunction with the PMINTENCLR register.

Configuration

AArch32 System register PMINTENSET bits [31:0] are architecturally mapped to AArch64 System register PMINTENSET_EL1[31:0] .

AArch32 System register PMINTENSET bits [31:0] are architecturally mapped to External register PMINTENSET_EL1[31:0] .

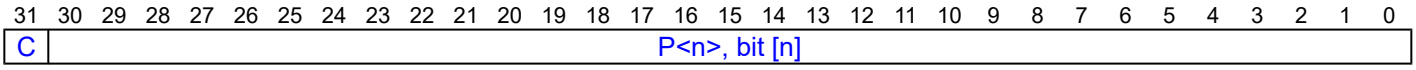
This register is in the Warm reset domain. On a Warm or Cold reset RW fields in this register reset to architecturally UNKNOWN values.

Attributes

PMINTENSET is a 32-bit register.

Field descriptions

The PMINTENSET bit assignments are:



C, bit [31]

PMCCNTR overflow interrupt request enable bit. Possible values are:

C	Meaning
0b0	When read, means the cycle counter overflow interrupt request is disabled. When written, has no effect.
0b1	When read, means the cycle counter overflow interrupt request is enabled. When written, enables the cycle count overflow interrupt request.

On a Warm reset, this field resets to an architecturally UNKNOWN value.

P<n>, bit [n], for n = 0 to 30

Event counter overflow interrupt request enable bit for PMEVCNTR<n>.

If N is less than 31, then bits [30:N] are RAZ/WI. When EL2 is implemented and enabled in the current Security state, in EL1 ~~and EL0~~, N is the value in MDCR_EL2.HPMN if EL2 is using AArch64, or in HDCR.HPMN if EL2 is using AArch32. Otherwise, N is the value in PMCR.N.

P<n>	Meaning
0b0	When read, means that the PMEVCNTR<n> event counter interrupt request is disabled. When written, has no effect.
0b1	When read, means that the PMEVCNTR<n> event counter interrupt request is enabled. When written, enables the PMEVCNTR<n> interrupt request.

On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing the PMINTENSET

Accesses to this register use the following encodings:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
~~opc1~~	~~opc2~~	~~CRn~~	~~coproc~~	~~CRm~~
0b1111	0b000	0b1001	0b1110	0b001
~~0b000~~	~~0b001~~	~~0b1001~~	~~0b1111~~	~~0b1110~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T9 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T9 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.TPM == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        return PMINTENSET;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        return PMINTENSET;
elsif PSTATE.EL == EL3 then
    return PMINTENSET;
```

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
~~opc1~~	~~opc2~~	~~CRn~~	~~coproc~~	~~CRm~~
0b1111	0b000	0b1001	0b1110	0b001
~~0b000~~	~~0b001~~	~~0b1001~~	~~0b1111~~	~~0b1110~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T9 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T9 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.TPM == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        PMINTENSET = R[t];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        PMINTENSET = R[t];
elsif PSTATE.EL == EL3 then
    PMINTENSET = R[t];
```

27~~13~~/03~~12~~/20192018 21~~16~~:59~~42~~; e5e4db499bf9867a4b93324c4dbac985d3da93766379d01e197f1d40720d32d0f84e419e9187e009

(old) htmldiff from- (new)

(old) htmldiff from- (new)

PMSELR, Performance Monitors Event Counter Selection Register

The PMSELR characteristics are:

Purpose

Selects the current event counter PMEVCNTR<n> or the cycle counter, CCNT.

PMSELR is used in conjunction with PMXEVTYPER to determine the event that increments a selected event counter, and the modes and states in which the selected counter increments.

It is also used in conjunction with PMXEVCNTR, to determine the value of a selected event counter.

Configuration

AArch32 System register PMSELR bits [31:0] are architecturally mapped to AArch64 System register PMSELR_EL0[31:0] .

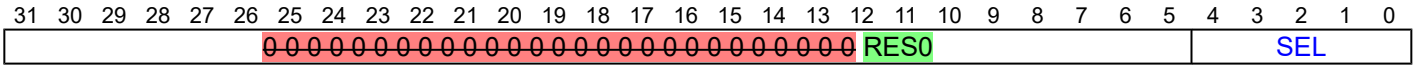
This register is in the Warm reset domain. On a Warm or Cold reset RW fields in this register reset to architecturally UNKNOWN values.

Attributes

PMSELR is a 32-bit register.

Field descriptions

The PMSELR bit assignments are:



Bits [31:5]

Reserved, RES0.

SEL, bits [4:0]

Selects event counter, PMEVCNTR<n>, where n is the value held in this field. This value identifies which event counter is accessed when a subsequent access to PMXEVTYPER or PMXEVCNTR occurs.

This field can take any value from 0 (`0b00000`) to (PMCR.N)-1, or 31 (`0b11111`).

When PMSELR.SEL is `0b11111`, it selects the cycle counter and:

- A read of the PMXEVTYPER returns the value of PMCCFILTR.
- A write of the PMXEVTYPER writes to PMCCFILTR.
- A read or write of PMXEVCNTR has CONSTRAINED UNPREDICTABLE effects.~~effects,~~ See~~that can be one of the following:~~
  - ~~Access to PMXEVCNTR is UNDEFINED.~~
  - ~~Access to PMXEVCNTR behaves as a NOP.~~
  - ~~Access to PMXEVCNTR behaves as if the register is RAZ/WI.~~
  - ~~Access to PMXEVCNTR behaves as if the PMSELR.SEL field contains an UNKNOWN value.~~

  PMXEVCNTR for more details.

If this field is set to a value greater than or equal to the number of ~~implemented~~ counters accessible at the current Exception level, but not equal to 31:

- Direct reads of this field return an UNKNOWN value.
- The results of access to PMXEVTYPER or PMXEVCNTR are CONSTRAINED UNPREDICTABLE.~~,~~ See~~and can be one of the following:~~

- ~~Access to PMXEVTYPER or PMXEVCNTR is UNDEFINED.~~
- ~~Access to PMXEVTYPER or PMXEVCNTR behaves as a NOP.~~
- ~~Access to PMXEVTYPER or PMXEVCNTR behaves as if the register is RAZ/WI.~~
- ~~Access to PMXEVTYPER or PMXEVCNTR behaves as if the PMSELR.SEL field contains an UNKNOWN value.~~
- ~~Access to PMXEVTYPER behaves as if the PMSELR.SEL field contains 0b11111.~~

PMXEVTYPER or PMXEVCNTR for more details.

For information about the number of counters accessible at each Exception level, see HDCR.HPMN and MDCR_EL2.HPMN.

On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing the PMSELR

Accesses to this register use the following encodings:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
~~opc1~~	~~opc2~~	~~CRn~~	~~coproc~~	~~CRm~~
0b1111	0b000	0b1001	0b1100	0b101
~~0b000~~	~~0b101~~	~~0b1001~~	~~0b1111~~	~~0b1100~~

```
if PSTATE.EL == EL0 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> != '11' && HSTR_EL2.T9 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T9 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif !ELUsingAArch32(EL1) && PMUSERENR_EL0.<ER,EN> == '00' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x03);
    elsif ELUsingAArch32(EL1) && PMUSERENR.<ER,EN> == '00' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TGE == '1' then
            AArch32.TakeHypTrapException(0x00);
        else
            UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.TPM == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        return PMSELR;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T9 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T9 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.TPM == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        return PMSELR;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        return PMSELR;
elsif PSTATE.EL == EL3 then
    return PMSELR;
```

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
~~opc1~~	~~opc2~~	~~CRn~~	~~coproc~~	~~CRm~~
0b1111	0b000	0b1001	0b1100	0b101
~~0b000~~	~~0b101~~	~~0b1001~~	~~0b1111~~	~~0b1100~~

```
if PSTATE.EL == EL0 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> != '11' && HSTR_EL2.T9 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T9 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif !ELUsingAArch32(EL1) && PMUSERENR_EL0.<ER,EN> == '00' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x03);
    elsif ELUsingAArch32(EL1) && PMUSERENR.<ER,EN> == '00' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TGE == '1' then
            AArch32.TakeHypTrapException(0x00);
        else
            UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.TPM == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        PMSELR = R[t];
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T9 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T9 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.TPM == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        PMSELR = R[t];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        PMSELR = R[t];
elsif PSTATE.EL == EL3 then
    PMSELR = R[t];
```

2713/0312/20192018 2116:5942; e5e4db499bf9867a4b93324c4dbac985d3da93766379d01e197f1d40720d32d0f84c419e9187e009

(old) htmldiff from- (new)

PMXEVCNTR, Performance Monitors Selected Event Count Register

The PMXEVCNTR characteristics are:

Purpose

Reads or writes the value of the selected event counter, PMEVCNTR<n>. PMSELR.SEL determines which event counter is selected.

Configuration

AArch32 System register PMXEVCNTR bits [31:0] are architecturally mapped to AArch64 System register PMXEVCNTR_EL0[31:0] .

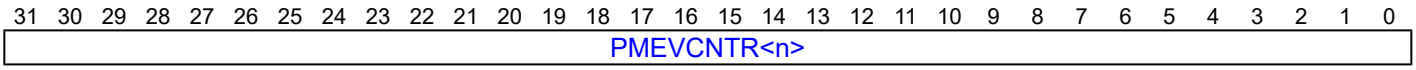
This register is in the Warm reset domain. On a Warm or Cold reset RW fields in this register reset to architecturally UNKNOWN values.

Attributes

PMXEVCNTR is a 32-bit register.

Field descriptions

The PMXEVCNTR bit assignments are:



PMEVCNTR<n>, bits [31:0]

Value of the selected event counter, PMEVCNTR<n>, where n is the value stored in PMSELR.SEL.

If ARMv8.5-PMU is implemented, the event counter is 64 bits and only the least-significant part of the event counter is accessible in AArch32 state:

- Reads from PMXEVCNTR return bits [31:0] of the counter.
- Writes to PMXEVCNTR update bits [31:0] and leave bits [63:32] unchanged.
- There is no means to access bits [63:32] directly from AArch32 state.
- If the implementation does not support AArch64 at any Exception level, bits [63:32] are not required to be implemented.

If ARMv8.5-PMU is not implemented, the event counter is 32 bits.

On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing the PMXEVCNTR

If PMSELR.SEL is greater than or equal to the number of accessible counters then reads and writes of PMXEVCNTR are CONSTRAINED UNPREDICTABLE, and the following behaviors are permitted:

- Accesses to the register are UNDEFINED.
- Accesses to the register behave as RAZ/WI.
- Accesses to the register execute as a NOP
- Accesses to the register behave as if PMSELR.SEL has an UNKNOWN value less than the number of counters accessible at the current Exception level and Security state.
- If EL2 is implemented and enabled in the current Security state, and PMSELR.SEL is less than the number of implemented counters, accesses from EL1 or permitted accesses from EL0 are trapped to EL2.

Note

In EL0, an access is permitted if it is enabled by PMUSERENR.{ER,EN} or PMUSERENR_EL0.{ER,EN}.

If EL2 is implemented and enabled in the current Security state, at EL0 and EL1:

- If EL2 is using AArch32, HDCR.HPMN identifies the number of accessible counters.
- If EL2 is using AArch64, MDCR_EL2.HPMN identifies the number of accessible counters.

Otherwise, the number of accessible counters is the number of implemented counters. See HDCR.HPMN and MDCR_EL2.HPMN for more details.

Accesses to this register use the following encodings:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
~~opc1~~	~~opc2~~	~~CRn~~	~~coproc~~	~~CRm~~
0b1111	0b000	0b1001	0b1101	0b010
~~0b000~~	~~0b010~~	~~0b1001~~	~~0b1111~~	~~0b1101~~

```
if PSTATE.EL == EL0 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> != '11' && HSTR_EL2.T9 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T9 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif !ELUsingAArch32(EL1) && PMUSERENR_EL0.<ER,EN> == '00' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x03);
    elsif ELUsingAArch32(EL1) && PMUSERENR.<ER,EN> == '00' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TGE == '1' then
            AArch32.TakeHypTrapException(0x00);
        else
            UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.TPM == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        return PMXEVCNTR;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T9 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T9 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.TPM == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        return PMXEVCNTR;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        return PMXEVCNTR;
elsif PSTATE.EL == EL3 then
    return PMXEVCNTR;
```

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
~~opc1~~	~~opc2~~	~~CRn~~	~~coproc~~	~~CRm~~
0b1111	0b000	0b1001	0b1101	0b010
~~0b000~~	~~0b010~~	~~0b1001~~	~~0b1111~~	~~0b1101~~

```
if PSTATE.EL == EL0 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> != '11' && HSTR_EL2.T9 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T9 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif !ELUsingAArch32(EL1) && PMUSERENR_EL0.EN == '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x03);
    elsif ELUsingAArch32(EL1) && PMUSERENR.EN == '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TGE == '1' then
            AArch32.TakeHypTrapException(0x00);
        else
            UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.TPM == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        PMXEVCNTR = R[t];
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T9 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T9 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.TPM == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        PMXEVCNTR = R[t];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        PMXEVCNTR = R[t];
elsif PSTATE.EL == EL3 then
    PMXEVCNTR = R[t];
```

2713/0312/20192018 2116:5942; e5e4db499bf9867a4b93324c4dbac985d3da93766379d01e197f1d40720d32d0f84c419e9187e009

(old) htmldiff from- (new)

PRRR, Primary Region Remap Register

The PRRR characteristics are:

Purpose

Controls the top level mapping of the TEX[0], C, and B memory region attributes.

Configuration

AArch32 System register PRRR bits [31:0] are architecturally mapped to AArch64 System register MAIR_EL1[31:0] when TTBCR.EAE == 0.

MAIR0 and PRRR are the same register, with a different view depending on the value of TTBCR.EAE:

- When it is set to 0, the register is as described in PRRR.
- When it is set to 1, the register is as described in MAIR0.

~~When EL3 is using AArch32, write access to PRRR(S) is disabled when the CP15SDISABLE signal is asserted HIGH.~~

RW fields in this register reset to architecturally UNKNOWN values.

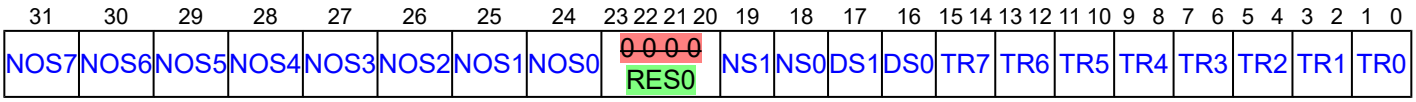
Attributes

PRRR is a 32-bit register.

Field descriptions

The PRRR bit assignments are:

When TTBCR.EAE == 0:



NOS<n>, bit [n+24], for n = 0 to 7

Not Outer Shareable. NOS<n> is the Outer Shareable property for memory attributes n, if the region is mapped as Normal memory that is not Inner Non-cacheable, Outer Non-cacheable, and the appropriate PRRR.{NS0, NS1} field identifies the region as shareable. n is the value of the concatenation of the {TEX[0], C, B} bits from the translation table descriptor. The possible values of each NOS<n> field other than NOS6 are:

NOS<n>	Meaning
0b0	Memory region is Outer Shareable.
0b1	Memory region is Inner Shareable.

The value of this bit is ignored if the region is:

- Device memory
- Normal memory that is at least one of:
  - Inner Non-cacheable, Outer Non-cacheable.
  - Identified by the appropriate PRRR.{NS0, NS1} field as Non-shareable.

The meaning of the NOS6 field is IMPLEMENTATION DEFINED.

This field resets to an architecturally UNKNOWN value.

Bits [23:20]

Reserved, RES0.

NS1, bit [19]

Mapping of S = 1 attribute for Normal memory regions. This field is used in determining the Shareability of a memory region that is mapped to Normal memory and both:

- Is not Inner Non-cacheable, Outer Non-cacheable.
- Has the S bit in the translation table descriptor set to 1.

The possible values of this bit are:

NS1	Meaning
0b0	Region is Non-shareable.
0b1	Region is shareable. The value of the appropriate PRRR.NOS<n> field determines whether the region is Inner Shareable or Outer Shareable.

This field resets to an architecturally UNKNOWN value.

NS0, bit [18]

Mapping of S = 0 attribute for Normal memory regions. This field is used in determining the Shareability of a memory region that is mapped to Normal memory and both:

- Is not Inner Non-cacheable, Outer Non-cacheable.
- Has the S bit in the translation table descriptor set to 0.

The possible values of this bit are:

NS0	Meaning
0b0	Region is Non-shareable.
0b1	Region is shareable. The value of the appropriate PRRR.NOS<n> field determines whether the region is Inner Shareable or Outer Shareable.

This field resets to an architecturally UNKNOWN value.

DS1, bit [17]

Mapping of S = 1 attribute for Device memory. In Armv8, all types of Device memory are Outer Shareable, and therefore this bit is RES1.

This field resets to an architecturally UNKNOWN value.

DS0, bit [16]

Mapping of S = 0 attribute for Device memory. In Armv8, all types of Device memory are Outer Shareable, and therefore this bit is RES1.

This field resets to an architecturally UNKNOWN value.

TR<n>, bits [2n+1:2n], for n = 0 to 7

TR<n> is the primary TEX mapping for memory attributes n, and defines the mapped memory type for a region with attributes n. n is the value of the concatenation of the {TEX[0], C, B} bits from the translation table descriptor. The possible values for each field other than TR6 are:

TR<n>	Meaning
0b00	Device-nGnRnE memory
0b01	Device-nGnRE memory
0b10	Normal memory

The value 0b11 is reserved. The effect of programming a field to 0b11 is CONSTRAINED UNPREDICTABLE, see 'Unallocated values in fields of AArch32 System registers and translation table entries' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section K1.1.11.

The meaning of the TR6 field is IMPLEMENTATION DEFINED.

This field resets to an architecturally UNKNOWN value.

Accessing the PRRR

Accesses to this register use the following encodings:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
~~opc1~~	~~opc2~~	~~CRn~~	~~coproc~~	~~CRm~~
0b1111	0b000	0b1010	0b0010	0b000
~~0b000~~	~~0b000~~	~~0b1010~~	~~0b1111~~	~~0b0010~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T10 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T10 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TRVM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TRVM == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) then
        if SCR.NS == '0' then
            return PRRR_S;
        else
            return PRRR_NS;
    else
        return PRRR;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && ELUsingAArch32(EL3) then
        return PRRR_NS;
    else
        return PRRR;
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        return PRRR_S;
    else
        return PRRR_NS;
```

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
~~opc1~~	~~opc2~~	~~CRn~~	~~coproc~~	~~CRm~~
0b1111	0b000	0b1010	0b0010	0b000
~~0b000~~	~~0b000~~	~~0b1010~~	~~0b1111~~	~~0b0010~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T10 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T10 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) && SCR.NS == '0' && CP15SDISABLE == HIGH then
        UNDEFINED;
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) && SCR.NS == '0' && CP15SDISABLE2 == HIGH then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TVM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TVM == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) then
        if SCR.NS == '0' then
            PRRR_S = R[t];
        else
            PRRR_NS = R[t];
    else
        PRRR = R[t];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && ELUsingAArch32(EL3) then
        PRRR_NS = R[t];
    else
        PRRR = R[t];
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' && CP15SDISABLE == HIGH then
        UNDEFINED;
    elsif SCR.NS == '0' && CP15SDISABLE2 == HIGH then
        UNDEFINED;
    else
        if SCR.NS == '0' then
            PRRR_S = R[t];
        else
            PRRR_NS = R[t];
```

2713/0312/20192018 2116:5942; e5e4db499bf9867a4b93324c4dbac985d3da93766379d01e197f1d40720d32d0f84e419e9187e009

(old) htmldiff from- (new)

SCTLR, System Control Register

The SCTLR characteristics are:

Purpose

Provides the top level control of the system, including its memory system.

Configuration

AArch32 System register SCTLR bits [31:0] are architecturally mapped to AArch64 System register SCTLR_EL1[31:0] .

When EL3 is using AArch32, write access to SCTLR(S) is disabled when the CP15SDISABLE signal is asserted HIGH.

Some bits in the register are read-only. These bits relate to non-configurable features of an implementation, and are provided for compatibility with previous versions of the architecture.

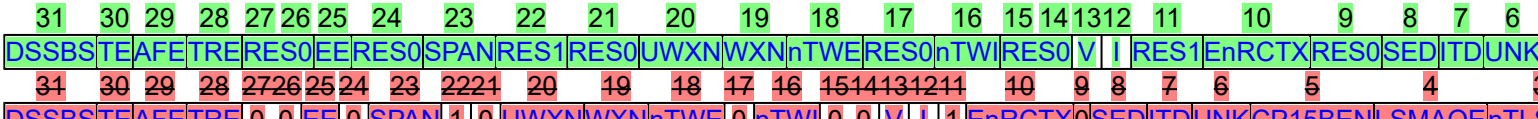
Some or all RW fields of this register have defined reset values. These apply only if the PE resets into an Exception level that is using AArch32. If the PE resets into EL3 using AArch32 they apply only to the Secure instance of the register. Otherwise, RW fields in this register reset to architecturally UNKNOWN values.

Attributes

SCTLR is a 32-bit register.

Field descriptions

The SCTLR bit assignments are:



DSSBS, bit [31]

From Armv8.5:

Default PSTATE.SSBS value on Exception Entry. The defined values are:

DSSBS	Meaning
0b0	PSTATE.SSBS is set to 0 on an exception to any mode in this security state except Hyp mode
0b1	PSTATE.SSBS is set to 1 on an exception to any mode in this security state except Hyp mode

Note

When EL3 is implemented and is using AArch32, this bit is banked between the two Security states.

This field resets to an IMPLEMENTATION DEFINED value.

Otherwise:

Reserved, RES0.

TE, bit [30]

T32 Exception Enable. This bit controls whether exceptions to an Exception Level that is executing at PL1 are taken to A32 or T32 state:

TE	Meaning
0b0	Exceptions, including reset, taken to A32 state.
0b1	Exceptions, including reset, taken to T32 state.

This field resets to an IMPLEMENTATION DEFINED choice between:

- 0.
- A value determined by an input configuration signal.

AFE, bit [29]

Access Flag Enable. When using the Short-descriptor translation table format for the PL1&0 translation regime, this bit enables use of the AP[0] bit in the translation descriptors as the Access flag, and restricts access permissions in the translation descriptors to the simplified model. The possible values of this bit are:

AFE	Meaning
0b0	In the translation table descriptors, AP[0] is an access permissions bit. The full range of access permissions is supported. No Access flag is implemented.
0b1	In the translation table descriptors, AP[0] is the Access flag. Only the simplified model for access permissions is supported.

When using the Long-descriptor translation table format, the VMSA behaves as if this bit is set to 1, regardless of the value of this bit.

The AFE bit is permitted to be cached in a TLB.

This field resets to 0.

TRE, bit [28]

TEX remap enable. This bit enables remapping of the TEX[2:1] bits in the PL1&0 translation regime for use as two translation table bits that can be managed by the operating system. Enabling this remapping also changes the scheme used to describe the memory region attributes in the VMSA. The possible values of this bit are:

TRE	Meaning
0b0	TEX remap disabled. TEX[2:0] are used, with the C and B bits, to describe the memory region attributes.
0b1	TEX remap enabled. TEX[2:1] are reassigned for use as bits managed by the operating system. The TEX[0], C, and B bits are used to describe the memory region attributes, with the MMU remap registers.

When the value of TTBCR.EAE is 1, this bit is RES1.

The TRE bit is permitted to be cached in a TLB.

This field resets to 0.

Bits [27:26]

Reserved, RES0.

EE, bit [25]

The value of the PSTATE.E bit on branch to an exception vector or coming out of reset, and the endianness of stage 1 translation table walks in the PL1&0 translation regime.

The possible values of this bit are:

EE	Meaning
0b0	Little-endian. PSTATE.E is cleared to 0 on taking an exception or coming out of reset. Stage 1 translation table walks in the PL1&0 translation regime are little-endian.
0b1	Big-endian. PSTATE.E is set to 1 on taking an exception or coming out of reset. Stage 1 translation table walks in the PL1&0 translation regime are big-endian.

If an implementation does not provide Big-endian support for data accesses at Exception Levels higher than EL0, this bit is RES0.

If an implementation does not provide Little-endian support for data accesses at Exception Levels higher than EL0, this bit is RES1.

This field resets to an IMPLEMENTATION DEFINED choice between:

- 0.
- A value determined by an input configuration signal.

Bit [24]

Reserved, RES0.

SPAN, bit [23]

When ARMv8.1-PAN is implemented:

Set Privileged Access Never, on taking an exception to EL1 from either Secure or Non-secure state, or to EL3 from Secure state when EL3 is using AArch32.

SPAN	Meaning
0b0	CPSR.PAN is set to 1 in the following situations: • In Non-secure state, on taking an exception to EL1. • In Secure state, when EL3 is using AArch64, on taking an exception to EL1. • In Secure state, when EL3 is using AArch32, on taking an exception to EL3.
0b1	The value of CPSR.PAN is left unchanged on taking an exception to EL1.

This field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES1.

Bit [22]

Reserved, RES1.

Bit [21]

Reserved, RES0.

UWXN, bit [20]

Unprivileged write permission implies PL1 XN (Execute-never). This bit can force all memory regions that are writable at PL0 to be treated as XN for accesses from software executing at PL1. The possible values of this bit are:

UWXN	Meaning
0b0	This control has no effect on memory access permissions.
0b1	Any region that is writable at PL0 forced to XN for accesses from software executing at PL1.

The UWXN bit is permitted to be cached in a TLB.

This field resets to 0.

WXN, bit [19]

Write permission implies XN (Execute-never). For the PL1&0 translation regime, this bit can force all memory regions that are writable to be treated as XN. The possible values of this bit are:

WXN	Meaning
0b0	This control has no effect on memory access permissions.
0b1	Any region that is writable in the PL1&0 translation regime is forced to XN for accesses from software executing at PL1 or PL0.

The WXN bit is permitted to be cached in a TLB.

This field resets to 0.

nTWE, bit [18]

Traps EL0 execution of WFE instructions to Undefined mode.

nTWE	Meaning
0b0	Any attempt to execute a WFE instruction at EL0 is trapped to Undefined mode, if the instruction would otherwise have caused the PE to enter a low-power state.
0b1	This control does not cause any instructions to be trapped.

The attempted execution of a conditional WFE instruction is only trapped if the instruction passes its condition code check.

Note

Since a WFE or WFI can complete at any time, even without a Wakeup event, the traps on WFE of WFI are not guaranteed to be taken, even if the WFE or WFI is executed when there is no Wakeup event. The only guarantee is that if the instruction does not complete in finite time in the absence of a Wakeup event, the trap will be taken.

This field resets to 1.

Bit [17]

Reserved, RES0.

nTWI, bit [16]

Traps EL0 execution of WFI instructions to Undefined mode.

nTWI	Meaning
0b0	Any attempt to execute a WFI instruction at EL0 is trapped to Undefined mode, if the instruction would otherwise have caused the PE to enter a low-power state.
0b1	This control does not cause any instructions to be trapped.

The attempted execution of a conditional WFI instruction is only trapped if the instruction passes its condition code check.

Note

Since a WFE or WFI can complete at any time, even without a Wakeup event, the traps on WFE of WFI are not guaranteed to be taken, even if the WFE or WFI is executed when there is no Wakeup event. The only guarantee is that if the instruction does not complete in finite time in the absence of a Wakeup event, the trap will be taken.

This field resets to 1.

Bits [15:14]

Reserved, RES0.

V, bit [13]

Vectors bit. This bit selects the base address of the exception vectors for exceptions taken to a PE mode other than Monitor mode or Hyp mode:

V	Meaning
0b0	Normal exception vectors. Base address is held in VBAR.
0b1	High exception vectors (Hivecs), base address 0xFFFF0000. This base address cannot be remapped.

This field resets to an IMPLEMENTATION DEFINED choice between:

- 0.
- A value determined by an input configuration signal.

I, bit [12]

Instruction access Cacheability control, for accesses at EL1 and EL0:

I	Meaning
0b0	All instruction access to Normal memory from PL1 and PL0 are Non-cacheable for all levels of instruction and unified cache. If the value of SCTLR.M is 0, instruction accesses from stage 1 of the PL1&0 translation regime are to Normal, Outer Shareable, Inner Non-cacheable, Outer Non-cacheable memory.
0b1	All instruction access to Normal memory from PL1 and PL0 can be cached at all levels of instruction and unified cache. If the value of SCTLR.M is 0, instruction accesses from stage 1 of the PL1&0 translation regime are to Normal, Outer Shareable, Inner Write-Through, Outer Write-Through memory.

Instruction accesses to Normal memory from EL1 and EL0 are Cacheable regardless of the value of the SCTLR.I bit if either:

- EL2 is using AArch32 and the value of HCR.DC is 1.
- EL2 is using AArch64 and the value of HCR_EL2.DC is 1.

This field resets to 0.

Bit [11]

Reserved, RES1.

EnRCTX, bit [10]

From Armv8.5:

Enable EL0 Access to the AArch32 CFPRCTX, DVPRCTX and CPPRCTX instructions. The defined values are:

EnRCTX	Meaning
0b0	EL0 access to these instructions is disabled, and these instructions are trapped to EL1.
0b1	EL0 access to these instructions is enabled.

Note

When EL3 is implemented and is using AArch32, this bit is banked between the two Security states.

This field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bit [9]

Reserved, RES0.

SED, bit [8]

SETEND instruction disable. Disables SETEND instructions at PL0 and PL1.

SED	Meaning
0b0	SETEND instruction execution is enabled at PL0 and PL1.
0b1	SETEND instructions are UNDEFINED at PL0 and PL1.

If the implementation does not support mixed-endian operation at any Exception level, this bit is RES1.

This field resets to 0.

ITD, bit [7]

IT Disable. Disables some uses of IT instructions at PL1 and PL0.

ITD	Meaning
0b0	All IT instruction functionality is enabled at PL1 and PL0.
0b1	Any attempt at PL1 or PL0 to execute any of the following is UNDEFINED: • All encodings of the IT instruction with hw1[3:0]!=1000. • All encodings of the subsequent instruction with the following values for hw1: ◦ 11xxxxxxxxxxxxxx: All 32-bit instructions, and the 16-bit instructions B, UDF, SVC, LDM, and STM. ◦ 1011xxxxxxxxxxxx: All instructions in Miscellaneous 16-bit instructions. ◦ 10100xxxxxxxxxx: ADD Rd, PC, #imm ◦ 01001xxxxxxxxxx: LDR Rd, [PC, #imm] ◦ 0100x1xxx1111xxx: ADD Rdn, PC; CMP Rn, PC; MOV Rd, PC; BX PC; BLX PC. ◦ 010001xx1xxxx111: ADD PC, Rm; CMP PC, Rm; MOV PC, Rm. This pattern also covers unpredictable cases with BLX Rn. These instructions are always UNDEFINED, regardless of whether they would pass or fail the condition code check that applies to them as a result of being in an IT block. It is IMPLEMENTATION DEFINED whether the IT instruction is treated as: • A 16-bit instruction, that can only be followed by another 16-bit instruction. • The first half of a 32-bit instruction. This means that, for the situations that are UNDEFINED, either the second 16-bit instruction or the 32-bit instruction is UNDEFINED. An implementation might vary dynamically as to whether IT is treated as a 16-bit instruction or the first half of a 32-bit instruction.

If an instruction in an active IT block that would be disabled by this field sets this field to 1 then behavior is CONSTRAINED UNPREDICTABLE. For more information see 'Changes to an ITD control by an instruction in an IT block' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section E1.2.4.

ITD is optional, but if it is implemented in the SCTLR then it must also be implemented in the SCTLR_EL1. If it is not implemented then this bit is RAZ/WI.

This field resets to 0.

UNK, bit [6]

Writes to this bit are IGNORED. Reads of this bit return an UNKNOWN value.

This field resets to an architecturally UNKNOWN value.

CP15BEN, bit [5]

System instruction memory barrier enable. Enables accesses to the DMB, DSB, and ISB System instructions in the (coproc==0b1111) encoding space from PL1 and PL0:

CP15BEN	Meaning
0b0	PL0 and PL1 execution of the CP15DMB, CP15DSB, and CP15ISB instructions is UNDEFINED.
0b1	PL0 and PL1 execution of the CP15DMB, CP15DSB, and CP15ISB instructions is enabled.

CP15BEN is optional, but if it is implemented in the SCTLR then it must also be implemented in the SCTLR_EL1. If it is not implemented then this bit is RAO/WI.

This field resets to 1.

LSMAOE, bit [4]

When ARMv8.2-LSMAOC is implemented:

Load Multiple and Store Multiple Atomicity and Ordering Enable.

LSMAOE	Meaning
0b0	For all memory accesses at EL1 or EL0, A32 and T32 Load Multiple and Store Multiple can have an interrupt taken during the sequence memory accesses, and the memory accesses are not required to be ordered.
0b1	The ordering and interrupt behavior of A32 and T32 Load Multiple and Store Multiple at EL1 or EL0 is as defined for Armv8.0.

This bit is permitted to be cached in a TLB.

This field resets to 1.

Otherwise:

Reserved, RES1.

nTLSMD, bit [3]

When ARMv8.2-LSMAOC is implemented:

No Trap Load Multiple and Store Multiple to Device-nGRE/Device-nGnRE/Device-nGnRnE memory.

nTLSMD	Meaning
0b0	All memory accesses by A32 and T32 Load Multiple and Store Multiple at EL1 or EL0 that are marked at stage 1 as Device-nGRE/Device-nGnRE/Device-nGnRnE memory are trapped and generate a stage 1 Alignment fault.
0b1	All memory accesses by A32 and T32 Load Multiple and Store Multiple at EL1 or EL0 that are marked at stage 1 as Device-nGRE/Device-nGnRE/Device-nGnRnE memory are not trapped.

This bit is permitted to be cached in a TLB.

This field resets to 1.

Otherwise:

Reserved, RES1.

C, bit [2]

Cacheability control, for data accesses at EL1 and EL0:

C	Meaning
0b0	All data access to Normal memory from PL1 and PL0, and all accesses to the PL1&0 stage 1 translation tables, are Non-cacheable for all levels of data and unified cache.
0b1	All data access to Normal memory from PL1 and PL0, and all accesses to the PL1&0 stage 1 translation tables, can be cached at all levels of data and unified cache.

The PE ignores SCLTR.C for Non-secure state and data accesses to Normal memory from EL1 and EL0 are Cacheable if either:

- EL2 is using AArch32 and the value of HCR.DC is 1.
- EL2 is using AArch64 and the value of HCR_EL2.DC is 1.

This field resets to 0.

A, bit [1]

Alignment check enable. This is the enable bit for Alignment fault checking at PL1 and PL0:

A	Meaning
0b0	Alignment fault checking disabled when executing at PL1 or PL0. Instructions that load or store one or more registers, other than load/store exclusive and load-acquire/store-release, do not check that the address being accessed is aligned to the size of the data element(s) being accessed.
0b1	Alignment fault checking enabled when executing at PL1 or PL0. All instructions that load or store one or more registers have an alignment check that the address being accessed is aligned to the size of the data element(s) being accessed. If this check fails it causes an Alignment fault, which is taken as a Data Abort exception.

Load/store exclusive and load-acquire/store-release instructions have an alignment check regardless of the value of the A bit.

This field resets to 0.

M, bit [0]

MMU enable for EL1 and EL0 stage 1 address translation. Possible values of this bit are:

M	Meaning
0b0	EL1 and EL0 stage 1 address translation disabled. See the SCTLR.I field for the behavior of instruction accesses to Normal memory.
0b1	EL1 and EL0 stage 1 address translation enabled.

In the Non-secure state the PE behaves as if the value of the SCTLR.M field is 0 for all purposes other than returning the value of a direct read of the field if either:

- EL2 is using AArch32 and the value of HCR.{DC, TGE} is not {0, 0}.
- EL2 is using AArch64 and the value of HCR_EL2.{DC, TGE} is not {0, 0}.

This field resets to 0.

Accessing the SCTLR

Accesses to this register use the following encodings:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
~~opc1~~	~~opc2~~	~~CRn~~	~~coproc~~	~~CRm~~
0b1111	0b000	0b0001	0b0000	0b000
~~0b000~~	~~0b000~~	~~0b0001~~	~~0b1111~~	~~0b0000~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T1 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T1 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TRVM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TRVM == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) then
        if SCR.NS == '0' then
            return SCTLR_S;
        else
            return SCTLR_NS;
    else
        return SCTLR;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && ELUsingAArch32(EL3) then
        return SCTLR_NS;
    else
        return SCTLR;
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        return SCTLR_S;
    else
        return SCTLR_NS;
```

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
~~opc1~~	~~opc2~~	~~CRn~~	~~coproc~~	~~CRm~~
0b1111	0b000	0b0001	0b0000	0b000
~~0b000~~	~~0b000~~	~~0b0001~~	~~0b1111~~	~~0b0000~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T1 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T1 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) && SCR.NS == '0' && CP15SDISABLE == HIGH then
        UNDEFINED;
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) && SCR.NS == '0' && CP15SDISABLE2 == HIGH then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TVM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TVM == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) then
        if SCR.NS == '0' then
            SCTLR_S = R[t];
        else
            SCTLR_NS = R[t];
    else
        SCTLR = R[t];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && ELUsingAArch32(EL3) then
        SCTLR_NS = R[t];
    else
        SCTLR = R[t];
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' && CP15SDISABLE == HIGH then
        UNDEFINED;
    elsif SCR.NS == '0' && CP15SDISABLE2 == HIGH then
        UNDEFINED;
    else
        if SCR.NS == '0' then
            SCTLR_S = R[t];
        else
            SCTLR_NS = R[t];
```

2713/0312/20192018 2116:5942; e5e4db499bf9867a4b93324c4dbac985d3da93766379d01e197f1d40720d32d0f84e419e9187e009

(old) htmldiff from- (new)

SDCR, Secure Debug Control Register

The SDCR characteristics are:

Purpose

Provides EL3 configuration options for self-hosted debug, trace, and the Performance Monitors Extension.

Configuration

AArch32 System register SDCR bits [31:0] can be mapped to AArch64 System register MDCR_EL3[31:0] , but this is not architecturally mandated.

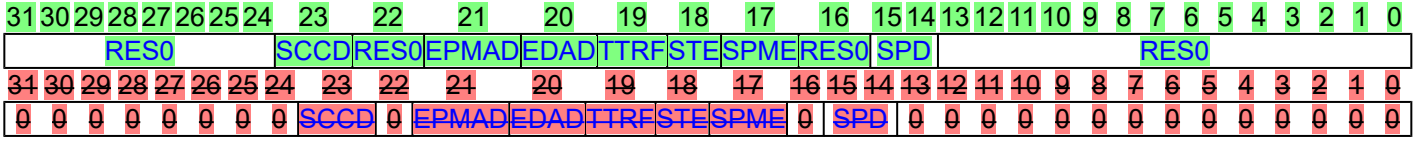
Some or all RW fields of this register have defined reset values. These apply whenever the register is accessible. This means they apply when the PE resets into EL3 using AArch32.

Attributes

SDCR is a 32-bit register.

Field descriptions

The SDCR bit assignments are:



Bits [31:24]

Reserved, RES0.

SCCD, bit [23]

When ARMv8.5-PMU is implemented:

Secure Cycle Counter Disable. Prohibits PMCCNTR from counting in Secure state.

SCCD	Meaning
0b0	Cycle counting by PMCCNTR is not affected by this bit.
0b1	Cycle counting by PMCCNTR is prohibited in Secure state.

This bit does not affect the CPU_CYCLES event or any other event that counts cycles.

In a system where the PE resets into EL3, this field resets to 0.

Otherwise:

Reserved, RES0.

Bit [22]

Reserved, RES0.

EPMAD, bit [21]

When ARMv8.4-Debug is implemented and PMUv3 is implemented:

External debug interface Performance Monitors registers disable. This controls Non-secure access to Performance Monitors registers by an external debugger.

EPMAD	Meaning
0b0	Non-secure access to the Performance Monitors registers from an external debugger is permitted.
0b1	Non-secure access to the Performance Monitors registers from an external debugger is not permitted.

If the Performance Monitors Extension does not support external debug interface accesses this bit is RES0.

In a system where the PE resets into EL3, this field resets to 0.

When PMUv3 is implemented:

External debug interface Performance Monitors registers disable. This controls access to Performance Monitors registers by an external debugger.

EPMAD	Meaning
0b0	Access to Performance Monitors registers from an external debugger is permitted.
0b1	Access to Performance Monitors registers from an external debugger is disabled, unless overridden by the IMPLEMENTATION DEFINED authentication interface.

If the Performance Monitors Extension does not support external debug interface accesses this bit is RES0.

In a system where the PE resets into EL3, this field resets to 0.

Otherwise:

Reserved, RES0.

EDAD, bit [20]

When ARMv8.4-Debug is implemented:

External debug register Non-secure access disable. Controls access to debug registers by an external debugger.

EDAD	Meaning
0b0	Non-secure access to debug registers from an external debugger is permitted.
0b1	Non-secure access to breakpoint registers, watchpoint registers, and OSLAR_EL1 from an external debugger is not permitted.

In a system where the PE resets into EL3, this field resets to 0.

When ARMv8.2-Debug is implemented:

External debug access disable. This disables access to debug registers by an external debugger.

EDAD	Meaning
0b0	Access to debug registers from an external debugger is permitted.
0b1	Access to breakpoint registers, watchpoint registers and OSLAR_EL1 from an external debugger is disabled, unless overridden by the IMPLEMENTATION DEFINED authentication interface.

In a system where the PE resets into EL3, this field resets to 0.

Otherwise:

External debug access disable. This disables access to debug registers by an external debugger.

EDAD	Meaning
0b0	Access to debug registers from an external debugger is permitted.
0b1	Access to breakpoint registers and watchpoint registers from an external debugger is not permitted, unless overridden by the IMPLEMENTATION DEFINED authentication interface. It is IMPLEMENTATION DEFINED whether access to the OSLAR_EL1 register from an external debugger is also not permitted.

In a system where the PE resets into EL3, this field resets to 0.

TTRF, bit [19]

When ARMv8.4-Trace is implemented:

Trap Trace Filter controls. Controls whether accesses at EL2 and EL1 to the trace filter control registers are trapped to EL3.

TTRF	Meaning
0b0	Accesses to HTRFCR and TRFCR registers are not affected by this control bit.
0b1	When not in Monitor mode, accesses to HTRFCR and TRFCR registers generate a Monitor trap exception.

In a system where the PE resets into EL3, this field resets to 0.

Otherwise:

Reserved, RES0.

STE, bit [18]

When ARMv8.4-Trace is implemented:

Secure Trace Enable. This bit enables tracing in Secure state and controls the level of authentication required by an external debugger to enable external tracing.

STE	Meaning
0b0	Trace is prohibited in Secure state unless overridden by the IMPLEMENTATION DEFINED authentication interface.
0b1	Trace is allowed in Secure state unless prohibited by the Trace Filter control registers.

If EL3 is not implemented and the PE executes in Secure state, the PE behaves as if this bit is set to 1.

In a system where the PE resets into EL3, this field resets to 0.

Otherwise:

Reserved, RES0.

SPME, bit [17]

When ARMv8.2-Debug is implemented and PMUv3 is implemented:

Secure Performance Monitors enable. This allows event counting in Secure state.

SPME	Meaning
0b0	Event counting prohibited in Secure state.
0b1	Event counting allowed in Secure state.

If EL3 is not implemented and the PE is executing in Secure state, then the Effective value of this bit is 0b1.

In a system where the PE resets into EL3, this field resets to 0.

When PMUv3 is implemented:

Secure Performance Monitors enable. This allows event counting in Secure state.

SPME	Meaning
0b0	Event counting prohibited in Secure state, unless ExternalSecureNoninvasiveDebugEnabled() is TRUE.
0b1	Event counting allowed in Secure state.

If EL3 is not implemented and the PE is executing in Secure state, then the Effective value of this bit is 0b1.

In a system where the PE resets into EL3, this field resets to 0.

Otherwise:

Reserved, RES0.

Bit [16]

Reserved, RES0.

SPD, bits [15:14]

AArch32 Secure privileged debug. Enables or disables debug exceptions from Secure state, other than Breakpoint Instruction exceptions. Valid values for this field are:

SPD	Meaning
0b00	Legacy mode. Debug exceptions from Secure EL1 are enabled by the authentication interface.
0b10	Secure privileged debug disabled. Debug exceptions from Secure EL1 are disabled.
0b11	Secure privileged debug enabled. Debug exceptions from Secure EL1 are enabled.

Other values are reserved, and have the CONSTRAINED UNPREDICTABLE behavior that they must have the same behavior as 0b00. Software must not rely on this property as the behavior of reserved values might change in a future revision of the architecture.

If debug exceptions from Secure EL1 are enabled, then debug exceptions from Secure EL0 are also enabled.

Otherwise, debug exceptions from Secure EL0 are enabled only if SDER32_EL3.SUIDEN == 1.

If EL3 is using AArch32, SDCR.SPD controls debug exceptions from EL3 and not from Secure EL1, as there is no Secure EL1.

Ignored in Non-secure state. Debug exceptions from Breakpoint Instruction exceptions are always enabled.

In a system where the PE resets into EL3, this field resets to 0.

Bits [13:0]

Reserved, RES0.

Accessing the SDCR

Accesses to this register use the following encodings:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
~~opc1~~	~~opc2~~	~~CRn~~	~~coproc~~	~~CRm~~
0b1111	0b000	0b0001	0b0011	0b001

0b000	0b001	0b0001	0b1111	0b0011

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T1 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T1 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif !ELUsingAArch32(EL2) && SCR_EL3.<NS,EEL2> == '01' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif !ELUsingAArch32(EL3) && SCR_EL3.NS == '0' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    return SDCR;
```

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
~~opc1~~	~~opc2~~	~~CRn~~	~~coproc~~	~~CRm~~
0b1111	0b000	0b0001	0b0011	0b001
~~0b000~~	~~0b001~~	~~0b0001~~	~~0b1111~~	~~0b0011~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T1 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T1 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif !ELUsingAArch32(EL2) && SCR_EL3.<NS,EEL2> == '01' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif !ELUsingAArch32(EL3) && SCR_EL3.NS == '0' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' && CP15SDISABLE2 == HIGH then
        UNDEFINED;
    else
        SDCR = R[t];
```

27~~13~~/03~~12~~/20192~~018~~ 21~~16~~:59~~42~~; e5e4db499bf9867a4b93324c4dbac985d3da93766379d01e197f1d40720d32d0f84e419e9187e009

(old) htmldiff from- (new)

SDER, Secure Debug Enable Register

The SDER characteristics are:

Purpose

Controls invasive and non-invasive debug in the Secure EL0 mode.

Configuration

AArch32 System register SDER bits [31:0] are architecturally mapped to AArch64 System register SDER32_EL3[31:0] .

If EL3 is not implemented and EL1 supports AArch32, SDER is implemented only if the implemented Security state is Secure state.

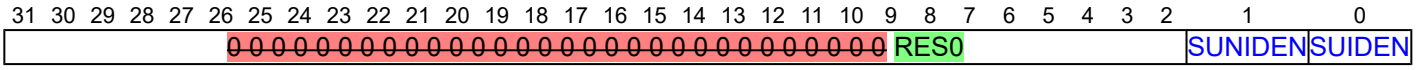
This register is in the Warm reset domain. Some or all RW fields of this register have defined reset values. On a Warm or Cold reset these apply only if the PE resets into an Exception level that is using AArch32. Otherwise, on a Warm or Cold reset RW fields in this register reset to architecturally UNKNOWN values.

Attributes

SDER is a 32-bit register.

Field descriptions

The SDER bit assignments are:



Bits [31:2]

Reserved, RES0.

SUNIDEN, bit [1]

Secure User Non-Invasive Debug Enable:

SUNIDEN	Meaning
0b0	Performance Monitors event counting prohibited in Secure EL0 unless allowed by AArc64-MDCR_EL3.SPME, SDCR.SPME. In an Armv8.0 or Armv8.1 implementation, event counting can also be allowed using the IMPLEMENTATION DEFINED authentication interface ExternalSecureNoninvasiveDebugEnabled().
0b1	Performance Monitors event counting allowed in Secure EL0.

On a Warm reset, this field resets to 0.

SUIDEN, bit [0]

Secure User Invasive Debug Enable:

SUIDEN	Meaning
0b0	Debug exceptions other than Breakpoint Instruction exceptions from Secure EL0 are disabled, unless enabled by MDCR_EL3.SPD32 or SDCR.SPD.
0b1	Debug exceptions from Secure EL0 are enabled.

On a Warm reset, this field resets to `0`.

Accessing the SDER

Accesses to this register use the following encodings:

`MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}`

coproc	opc1	CRn	CRm	opc2
~~opc1~~	~~opc2~~	~~CRn~~	~~coproc~~	~~CRm~~
0b1111	0b000	0b0001	0b0001	0b001
~~0b000~~	~~0b001~~	~~0b0001~~	~~0b1111~~	~~0b0001~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T1 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T1 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif (!HaveEL(EL3) || !ELUsingAArch32(EL3)) && SCR_EL3.NS == '0' then
        return SDER;
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    return SDER;
```

`MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}`

coproc	opc1	CRn	CRm	opc2
~~opc1~~	~~opc2~~	~~CRn~~	~~coproc~~	~~CRm~~
0b1111	0b000	0b0001	0b0001	0b001
~~0b000~~	~~0b001~~	~~0b0001~~	~~0b1111~~	~~0b0001~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T1 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T1 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif (!HaveEL(EL3) || !ELUsingAArch32(EL3)) && SCR_EL3.NS == '0' then
        SDER = R[t];
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' && CP15SDISABLE2 == HIGH then
        UNDEFINED;
    else
        SDER = R[t];
```

27~~13~~/03~~12~~/2019~~2018~~ 21~~16~~:59~~42~~: e5e4db499bf9867a4b93324c4dbac985d3da93766379d01e197f1d40720d32d0f84e419e9187e009

(old) htmldiff from- (new)

(old) htmldiff from- (new)

TLBIALL, TLB Invalidate All

The TLBIALL characteristics are:

Purpose

Invalidate all cached copies of translation table entries from TLBs that are from any level of the translation table walk. The entries that are invalidated are as follows:

- If executed at Secure EL1 when EL3 is using AArch64, all entries that would be required for the Secure EL1&0 translation regime.
- If executed in Secure state when EL3 is using AArch32, all entries that would be required for the Secure PL1&0 translation regime.
- If executed at Non-secure EL1, all stage 1 translation table entries that would be required for the Non-secure PL1&0 translation regime and, if EL2 is implemented, they must match the current VMID.
- If executed at EL2, and if EL2 is enabled in the current Security state, the stage 1 or stage 2 translation table entries that would be required for the PL1&0 translation regime and matches the current VMID.

The invalidation only applies to the PE that executes this System instruction.

Configuration

Attributes

TLBIALL is a 32-bit System instruction.

Field descriptions

TLBIALL ignores the value in the register specified by the instruction encoding. Software does not have to write a value to the register before issuing this instruction.

Executing the TLBIALL instruction

Accesses to this instruction use the following encodings:

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
~~opc1~~	~~opc2~~	~~CRn~~	~~coproc~~	~~CRm~~
0b1111	0b000	0b1000	0b0111	0b000
~~0b000~~	~~0b000~~	~~0b1000~~	~~0b1111~~	~~0b0111~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T8 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T8 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TTLB == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR.TTLB == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.FB == '1' then
        TLBIALLIS();
    else
        TLBIALL();
elsif PSTATE.EL == EL2 then
    TLBIALL();
elsif PSTATE.EL == EL3 then
    TLBIALL();
```

27~~13~~/03~~12~~/20192018 21~~16~~:59~~42~~; e5e4db499bf9867a4b93324c4dbac985d3da93766379d01e197f1d40720d32d0f84c419c9187c009

(old) htmldiff from- (new)

(old) htmldiff from- (new)

TLBIALLH, TLB Invalidate All, Hyp mode

The TLBIALLH characteristics are:

Purpose

If EL2 is implemented, invalidate all cached copies of translation table entries from TLBs that are from any level of the translation table walk that would be required for the Non-secure EL2 translation regime.

The invalidation only applies to the PE that executes this System instruction.

Configuration

Attributes

TLBIALLH is a 32-bit System instruction.

Field descriptions

TLBIALLH ignores the value in the register specified by the instruction encoding. Software does not have to write a value to the register before issuing this instruction.

Executing the TLBIALLH instruction

If this instruction is executed in a Secure privileged mode other than Monitor mode, then the behavior is CONSTRAINED UNPREDICTABLE, and one of the following behaviors must occur:

- The instruction is UNDEFINED.
- The instruction is treated as a NOP.
- The instruction executes as if it had been executed in Monitor mode.

Accesses to this instruction use the following encodings:

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
~~opc1~~	~~opc2~~	~~CRn~~	~~coproc~~	~~CRm~~
0b1111	0b100	0b1000	0b0111	0b000
~~0b100~~	~~0b000~~	~~0b1000~~	~~0b1111~~	~~0b0111~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T8 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T8 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    TLBIALLH();
elsif PSTATE.EL == EL3 then
    if !HaveEL(EL2) then
        UNDEFINED;
    else
        TLBIALLH();
```

27~~13~~/03~~12~~/2019~~2018~~ 21~~16~~:59~~42~~; e5e4db499bf9867a4b93324c4dbac985d3da9376~~6379d01e197f1d40720d32d0f84e419e9187e009~~

(old) htmldiff from- (new)

(old) htmldiff from- (new)

TLBIALLHIS, TLB Invalidate All, Hyp mode, Inner Shareable

The TLBIALLHIS characteristics are:

Purpose

If EL2 is implemented, invalidate all cached copies of translation table entries from TLBs that are from any level of the translation table walk that would be required for the Non-secure EL2 translation regime.

The invalidation applies to all PEs in the same Inner Shareable shareability domain as the PE that executes this System instruction.~~instructions.~~

Configuration

Attributes

TLBIALLHIS is a 32-bit System instruction.

Field descriptions

TLBIALLHIS ignores the value in the register specified by the instruction encoding. Software does not have to write a value to the register before issuing this instruction.

Executing the TLBIALLHIS instruction

If this instruction is executed in a Secure privileged mode other than Monitor mode, then the behavior is CONSTRAINED UNPREDICTABLE, and one of the following behaviors must occur:

- The instruction is UNDEFINED.
- The instruction is treated as a NOP.
- The instruction executes as if it had been executed in Monitor mode.

Accesses to this instruction use the following encodings:

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
~~opc1~~	~~opc2~~	~~CRn~~	~~coproc~~	~~CRm~~
0b1111	0b100	0b1000	0b0011	0b000
~~0b100~~	~~0b000~~	~~0b1000~~	~~0b1111~~	~~0b0011~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T8 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T8 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    TLBIALLHIS();
elsif PSTATE.EL == EL3 then
    if !HaveEL(EL2) then
        UNDEFINED;
    else
        TLBIALLHIS();
```

27~~13~~/03~~12~~/2019~~2018~~ 21~~16~~:59~~42~~; e5e4db499bf9867a4b93324c4dbac985d3da9376~~6379d01e197f1d40720d32d0f84e419e9187e009~~

(old) htmldiff from- (new)

(old) htmldiff from- (new)

TLBIALLIS, TLB Invalidate All, Inner Shareable

The TLBIALLIS characteristics are:

Purpose

Invalidate all cached copies of translation table entries from TLBs that are from any level of the translation table walk. The entries that are invalidated are as follows:

- If executed at Secure EL1 when EL3 is using AArch64, all entries that would be required for the Secure EL1&0 translation regime.
- If executed in Secure state when EL3 is using AArch32, all entries that would be required for the Secure PL1&0 translation regime.
- If executed at Non-secure EL1, all stage 1 translation table entries that would be required for the Non-secure PL1&0 translation regime and, if EL2 is implemented, they must match the current VMID.
- If executed at EL2 and if EL2 is enabled in the current Security state, the stage 1 or stage 2 translation table entries that would be required for the PL1&0 translation regime and matches the current VMID.

The invalidation applies to all PEs in the same Inner Shareable shareability domain as the PE that executes this System instruction.~~instructions.~~

Configuration

Attributes

TLBIALLIS is a 32-bit System instruction.

Field descriptions

TLBIALLIS ignores the value in the register specified by the instruction encoding. Software does not have to write a value to the register before issuing this instruction.

Executing the TLBIALLIS instruction

Accesses to this instruction use the following encodings:

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
~~opc1~~	~~opc2~~	~~CRn~~	~~coproc~~	~~CRm~~
0b1111	0b000	0b1000	0b0011	0b000
~~0b000~~	~~0b000~~	~~0b1000~~	~~0b1111~~	~~0b0011~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T8 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T8 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TTLB == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TTLBIS == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR.TTLB == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR2.TTLBIS == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        TLBIALLIS();
elsif PSTATE.EL == EL2 then
    TLBIALLIS();
elsif PSTATE.EL == EL3 then
    TLBIALLIS();
```

27~~13~~/03~~12~~/20192018 21~~16~~:59~~42~~; e5e4db499bf9867a4b93324c4dbac985d3da93766379d01e197f1d40720d32d0f84e419e9187e009

(old) htmldiff from- (new)

(old) htmldiff from- (new)

TLBIALLNSNH, TLB Invalidate All, Non-Secure Non-Hyp

The TLBIALLNSNH characteristics are:

Purpose

If EL2 is implemented, invalidate all cached copies of translation table entries from TLBs that are from any level of the translation table walk that would be required for stage 1 or stage 2 of the Non-secure PL1&0 translation regime, regardless of the associated VMID.

The invalidation only applies to the PE that executes this System instruction.

Configuration

Attributes

TLBIALLNSNH is a 32-bit System instruction.

Field descriptions

TLBIALLNSNH ignores the value in the register specified by the instruction encoding. Software does not have to write a value to the register before issuing this instruction.

Executing the TLBIALLNSNH instruction

If this instruction is executed in a Secure privileged mode other than Monitor mode, then the behavior is CONSTRAINED UNPREDICTABLE, and one of the following behaviors must occur:

- The instruction is UNDEFINED.
- The instruction is treated as a NOP.
- The instruction executes as if it had been executed in Monitor mode.

Accesses to this instruction use the following encodings:

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
~~opc1~~	~~opc2~~	~~CRn~~	~~coproc~~	~~CRm~~
0b1111	0b100	0b1000	0b0111	0b100
~~0b100~~	~~0b100~~	~~0b1000~~	~~0b1111~~	~~0b0111~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T8 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T8 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    TLBIALLNSNH();
elsif PSTATE.EL == EL3 then
    if !HaveEL(EL2) then
        UNDEFINED;
    else
        TLBIALLNSNH();
```

27~~13~~/03~~12~~/2019~~2018~~ 21~~16~~:59~~42~~; e5e4db499bf9867a4b93324c4dbac985d3da9376~~6379d01e197f1d40720d32d0f84e419e9187e009~~

(old) htmldiff from- (new)

(old) htmldiff from- (new)

TLBIALLNSNHIS, TLB Invalidate All, Non-Secure Non-Hyp, Inner Shareable

The TLBIALLNSNHIS characteristics are:

Purpose

If EL2 is implemented, invalidate all cached copies of translation table entries from TLBs that are from any level of the translation table walk that would be required for stage 1 or stage 2 of the Non-secure PL1&0 translation regime, regardless of the associated VMID.

The invalidation applies to all PEs in the same Inner Shareable shareability domain as the PE that executes this System instruction.~~instructions.~~

Configuration

Attributes

TLBIALLNSNHIS is a 32-bit System instruction.

Field descriptions

TLBIALLNSNHIS ignores the value in the register specified by the instruction encoding. Software does not have to write a value to the register before issuing this instruction.

Executing the TLBIALLNSNHIS instruction

If this instruction is executed in a Secure privileged mode other than Monitor mode, then the behavior is CONSTRAINED UNPREDICTABLE, and one of the following behaviors must occur:

- The instruction is UNDEFINED.
- The instruction is treated as a NOP.
- The instruction executes as if it had been executed in Monitor mode.

Accesses to this instruction use the following encodings:

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
~~opc1~~	~~opc2~~	~~CRn~~	~~coproc~~	~~CRm~~
0b1111	0b100	0b1000	0b0011	0b100
~~0b100~~	~~0b100~~	~~0b1000~~	~~0b1111~~	~~0b0011~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T8 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T8 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    TLBIALLNSNHIS();
elsif PSTATE.EL == EL3 then
    if !HaveEL(EL2) then
        UNDEFINED;
    else
        TLBIALLNSNHIS();
```

27~~13~~/03~~12~~/2019~~2018~~ 21~~16~~:59~~42~~; e5e4db499bf9867a4b93324c4dbac985d3da9376~~6379d01e197f1d40720d32d0f84e419e9187e009~~

(old) htmldiff from- (new)

(old) htmldiff from- (new)

TLBIASID, TLB Invalidate by ASID match

The TLBIASID characteristics are:

Purpose

Invalidate all cached copies of translation table entries from TLBs that meet the following requirements:

- The entry is a stage 1 translation table entry.
- The entry would be used for the specified ASID, and either:
  - Is from a level of lookup above the final level.
  - Is a non-global entry from the final level of lookup.
- If EL2 is implemented and enabled in the current Security state, the entry would be used with the current VMID.

From the entries that match these requirement, the entries that are invalidated are required for the following translation regime:

- If executed at Secure EL1 when EL3 is using AArch64, the Secure EL1&0 translation regime.
- If executed in Secure state when EL3 is using AArch32, the Secure PL1&0 translation regime.
- If executed in Non-secure state, the Non-secure PL1&0 translation regime.

The invalidation only applies to the PE that executes this System instruction.

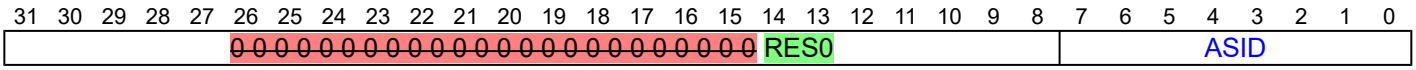
Configuration

Attributes

TLBIASID is a 32-bit System instruction.

Field descriptions

The TLBIASID input value bit assignments are:



Bits [31:8]

Reserved, RES0.

ASID, bits [7:0]

ASID value to match. Any TLB entries for non-global pages that match the ASID values will be affected by this System instruction.~~operation.~~

Executing the TLBIASID instruction

Accesses to this instruction use the following encodings:

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
~~opc1~~	~~opc2~~	~~CRn~~	~~coproc~~	~~CRm~~
0b1111	0b000	0b1000	0b0111	0b010
~~0b000~~	~~0b010~~	~~0b1000~~	~~0b1111~~	~~0b0111~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T8 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T8 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TTLB == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR.TTLB == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.FB == '1' then
        TLBIASIDIS(R[t]);
    else
        TLBIASID(R[t]);
elsif PSTATE.EL == EL2 then
    TLBIASID(R[t]);
elsif PSTATE.EL == EL3 then
    TLBIASID(R[t]);
```

27~~13~~/03~~12~~/20192018 21~~16~~:59~~42~~; e5e4db499bf9867a4b93324c4dbac985d3da93766379d01e197f1d40720d32d0f84c419c9187c009

(old) htmldiff from- (new)

(old) htmldiff from- (new)

TLBIASIDIS, TLB Invalidate by ASID match, Inner Shareable

The TLBIASIDIS characteristics are:

Purpose

Invalidate all cached copies of translation table entries from TLBs that meet the following requirements:

- The entry is a stage 1 translation table entry.
- The entry would be used for the specified ASID, and either:
  - Is from a level of lookup above the final level.
  - Is a non-global entry from the final level of lookup.
- If EL2 is implemented and enabled in the current Security state, the entry would be used with the current VMID.

From the entries that match these requirement, the entries that are invalidated are required for the following translation regime:

- If executed at Secure EL1 when EL3 is using AArch64, the Secure EL1&0 translation regime.
- If executed in Secure state when EL3 is using AArch32, the Secure PL1&0 translation regime.
- If executed in Non-secure state, the Non-secure PL1&0 translation regime.

The invalidation applies to all PEs in the same Inner Shareable shareability domain as the PE that executes this System instruction.~~instructions.~~

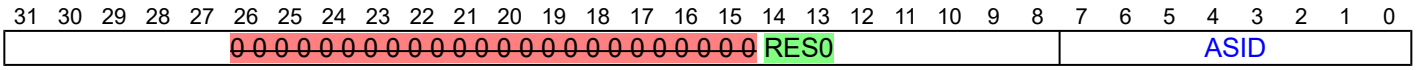
Configuration

Attributes

TLBIASIDIS is a 32-bit System instruction.

Field descriptions

The TLBIASIDIS input value bit assignments are:



Bits [31:8]

Reserved, RES0.

ASID, bits [7:0]

ASID value to match. Any TLB entries for non-global pages that match the ASID values will be affected by this System instruction.~~operation.~~

Executing the TLBIASIDIS instruction

Accesses to this instruction use the following encodings:

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
~~opc1~~	~~opc2~~	~~CRn~~	~~coproc~~	~~CRm~~
0b1111	0b000	0b1000	0b0011	0b010
~~0b000~~	~~0b010~~	~~0b1000~~	~~0b1111~~	~~0b0011~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T8 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T8 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TTLB == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TTLBIS == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR.TTLB == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR2.TTLBIS == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        TLBIASIDIS(R[t]);
elsif PSTATE.EL == EL2 then
    TLBIASIDIS(R[t]);
elsif PSTATE.EL == EL3 then
    TLBIASIDIS(R[t]);
```

27~~13~~/03~~12~~/20192018 21~~16~~:59~~42~~; e5e4db499bf9867a4b93324c4dbac985d3da93766379d01e197f1d40720d32d0f84e419e9187e009

(old) htmldiff from- (new)

(old) htmldiff from- (new)

TLBIIPAS2, TLB Invalidate by Intermediate Physical Address, Stage 2

The TLBIIPAS2 characteristics are:

Purpose

If EL2 is implemented, invalidate all cached copies of translation table entries from TLBs that meet the following requirements:

- The entry is a stage 2 only translation table entry, from any level of the translation table walk.
- SCR.NS is 1.
- The entry would be used for the specified IPA.
- The entry would be used with the current VMID.
- The entry would be required for the PL1&0 translation regime.

The invalidation is not required to apply to caching structures that combine stage 1 and stage 2 translation table entries.

The invalidation only applies to the PE that executes this System instruction.

Configuration

Note

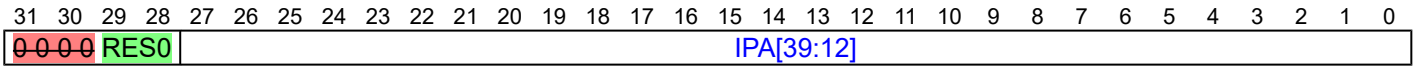
This System instruction is not implemented in architecture versions before Armv8.

Attributes

TLBIIPAS2 is a 32-bit System instruction.

Field descriptions

The TLBIIPAS2 input value bit assignments are:



Bits [31:28]

Reserved, RES0.

IPA[39:12], bits [27:0]

Bits[39:12] of the intermediate physical address to match.

Executing the TLBIIPAS2 instruction

If this instruction is executed in a Secure privileged mode other than Monitor mode, then the behavior is CONSTRAINED UNPREDICTABLE, and one of the following behaviors must occur:

- The instruction is UNDEFINED.
- The instruction is treated as a NOP.
- The instruction executes as if it had been executed in Monitor mode.

Accesses to this instruction use the following encodings:

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
~~opc1~~	~~opc2~~	~~CRn~~	~~coproc~~	~~CRm~~
0b1111	0b100	0b1000	0b0100	0b001
~~0b100~~	~~0b001~~	~~0b1000~~	~~0b1111~~	~~0b0100~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T8 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T8 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    TLBIIPAS2(R[t]);
elsif PSTATE.EL == EL3 then
    if !HaveEL(EL2) then
        UNDEFINED;
    elsif SCR.NS == '0' then
        //no operation
    else
        TLBIIPAS2(R[t]);
```

(old) htmldiff from- (new)

TLBIIPAS2IS, TLB Invalidate by Intermediate Physical Address, Stage 2, Inner Shareable

The TLBIIPAS2IS characteristics are:

Purpose

If EL2 is implemented, invalidate all cached copies of translation table entries from TLBs that meet the following requirements:

- The entry is a stage 2 only translation table entry, from any level of the translation table walk.
- SCR.NS is 1.
- The entry would be used for the specified IPA.
- The entry would be used with the current VMID.
- The entry would be required for the PL1&0 translation regime.

The invalidation is not required to apply to caching structures that combine stage 1 and stage 2 translation table entries.

The invalidation applies to all PEs in the same Inner Shareable shareability domain as the PE that executes this System instruction.~~instructions.~~

Configuration

Note

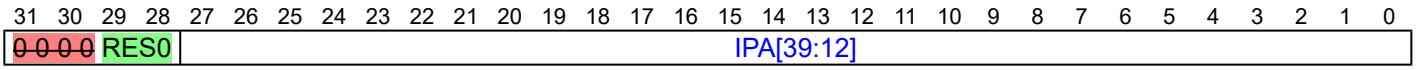
This System instruction is not implemented in architecture versions before Armv8.

Attributes

TLBIIPAS2IS is a 32-bit System instruction.

Field descriptions

The TLBIIPAS2IS input value bit assignments are:



Bits [31:28]

Reserved, RES0.

IPA[39:12], bits [27:0]

Bits[39:12] of the intermediate physical address to match.

Executing the TLBIIPAS2IS instruction

If this instruction is executed in a Secure privileged mode other than Monitor mode, then the behavior is CONSTRAINED UNPREDICTABLE, and one of the following behaviors must occur:

- The instruction is UNDEFINED.
- The instruction is treated as a NOP.
- The instruction executes as if it had been executed in Monitor mode.

Accesses to this instruction use the following encodings:

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
~~opc1~~	~~opc2~~	~~CRn~~	~~coproc~~	~~CRm~~
0b1111	0b100	0b1000	0b0000	0b001
~~0b100~~	~~0b001~~	~~0b1000~~	~~0b1111~~	~~0b0000~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T8 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T8 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    TLBIIPAS2(R[t]);
elsif PSTATE.EL == EL3 then
    if !HaveEL(EL2) then
        UNDEFINED;
    elsif SCR.NS == '0' then
        //no operation
    else
        TLBIIPAS2(R[t]);
```

27~~13~~/03~~12~~/2019~~2018~~ 21~~16~~:59~~42~~; e5e4db499bf9867a4b93324c4dbac985d3da93766~~379d01e197f1d40720d32d0f84e419e9187e009~~

(old) htmldiff from- (new)

TLBIIPAS2L, TLB Invalidate by Intermediate Physical Address, Stage 2, Last level

The TLBIIPAS2L characteristics are:

Purpose

If EL2 is implemented, invalidate all cached copies of translation table entries from TLBs that meet the following requirements:

- The entry is a stage 2 only translation table entry, from the final level of the translation table walk.
- SCR.NS is 1.
- The entry would be used for the specified IPA.
- The entry would be used with the current VMID.
- The entry would be required for the PL1&0 translation regime.

The invalidation is not required to apply to caching structures that combine stage 1 and stage 2 translation table entries.

The invalidation only applies to the PE that executes this System instruction.

Configuration

Note

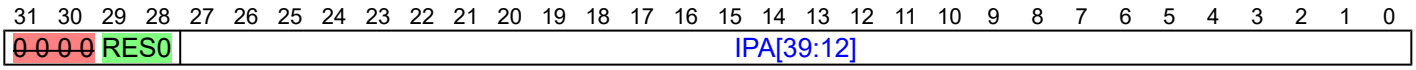
This System instruction is not implemented in architecture versions before Armv8.

Attributes

TLBIIPAS2L is a 32-bit System instruction.

Field descriptions

The TLBIIPAS2L input value bit assignments are:



Bits [31:28]

Reserved, RES0.

IPA[39:12], bits [27:0]

Bits[39:12] of the intermediate physical address to match.

Executing the TLBIIPAS2L instruction

If this instruction is executed in a Secure privileged mode other than Monitor mode, then the behavior is CONSTRAINED UNPREDICTABLE, and one of the following behaviors must occur:

- The instruction is UNDEFINED.
- The instruction is treated as a NOP.
- The instruction executes as if it had been executed in Monitor mode.

Accesses to this instruction use the following encodings:

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
~~opc1~~	~~opc2~~	~~CRn~~	~~coproc~~	~~CRm~~
0b1111	0b100	0b1000	0b0100	0b101
~~0b100~~	~~0b101~~	~~0b1000~~	~~0b1111~~	~~0b0100~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T8 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T8 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    TLBIIPAS2(R[t]);
elsif PSTATE.EL == EL3 then
    if !HaveEL(EL2) then
        UNDEFINED;
    elsif SCR.NS == '0' then
        //no operation
    else
        TLBIIPAS2(R[t]);
```

27~~13~~/03~~12~~/2019~~2018~~ 21~~16~~:59~~42~~; e5e4db499bf9867a4b93324c4dbac985d3da93766~~379d01e197f1d40720d32d0f84e419e9187e009~~

(old) htmldiff from- (new)

TLBIIPAS2LIS, TLB Invalidate by Intermediate Physical Address, Stage 2, Last level, Inner Shareable

The TLBIIPAS2LIS characteristics are:

Purpose

If EL2 is implemented, invalidate all cached copies of translation table entries from TLBs that meet the following requirements:

- The entry is a stage 2 only translation table entry, from the final level of the translation table walk.
- SCR.NS is 1.
- The entry would be used for the specified IPA.
- The entry would be used with the current VMID.
- The entry would be required for the PL1&0 translation regime.

The invalidation is not required to apply to caching structures that combine stage 1 and stage 2 translation table entries.

The invalidation applies to all PEs in the same Inner Shareable shareability domain as the PE that executes this System instruction.~~instructions.~~

Configuration

Note

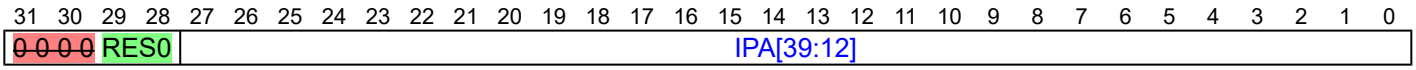
This System instruction is not implemented in architecture versions before Armv8.

Attributes

TLBIIPAS2LIS is a 32-bit System instruction.

Field descriptions

The TLBIIPAS2LIS input value bit assignments are:



Bits [31:28]

Reserved, RES0.

IPA[39:12], bits [27:0]

Bits[39:12] of the intermediate physical address to match.

Executing the TLBIIPAS2LIS instruction

If this instruction is executed in a Secure privileged mode other than Monitor mode, then the behavior is CONSTRAINED UNPREDICTABLE, and one of the following behaviors must occur:

- The instruction is UNDEFINED.
- The instruction is treated as a NOP.
- The instruction executes as if it had been executed in Monitor mode.

Accesses to this instruction use the following encodings:

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
~~opc1~~	~~opc2~~	~~CRn~~	~~coproc~~	~~CRm~~
0b1111	0b100	0b1000	0b0000	0b101
~~0b100~~	~~0b101~~	~~0b1000~~	~~0b1111~~	~~0b0000~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T8 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T8 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    TLBIIPAS2IS(R[t]);
elsif PSTATE.EL == EL3 then
    if !HaveEL(EL2) then
        UNDEFINED;
    elsif SCR.NS == '0' then
        //no operation
    else
        TLBIIPAS2IS(R[t]);
```

27~~13~~/03~~12~~/2019~~2018~~ 21~~16~~:59~~42~~; e5e4db499bf9867a4b93324c4dbac985d3da93766~~379d01e197f1d40720d32d0f84e419e9187e009~~

(old) htmldiff from- (new)

TLBIMVA, TLB Invalidate by VA

The TLBIMVA characteristics are:

Purpose

Invalidate all cached copies of translation table entries from TLBs that meet the following requirements:

- The entry is a stage 1 translation table entry.
- The entry would be used to translate the specified address, and one of the following applies:
  - The entry is from a level of lookup above the final level and matches the specified ASID.
  - The entry is a global entry from the final level of lookup.
  - The entry is a non-global entry from the final level of lookup that matches the specified ASID.
- If EL2 is implemented and enabled in the current Security state, the entry would be used with the current VMID.

From the entries that match these requirement, the entries that are invalidated are required for the following translation regime:

- If executed at Secure EL1 when EL3 is using AArch64, the Secure EL1&0 translation regime.
- If executed in Secure state when EL3 is using AArch32, the Secure PL1&0 translation regime.
- If executed in Non-secure state, the Non-secure PL1&0 translation regime.

The invalidation only applies to the PE that executes this System instruction.

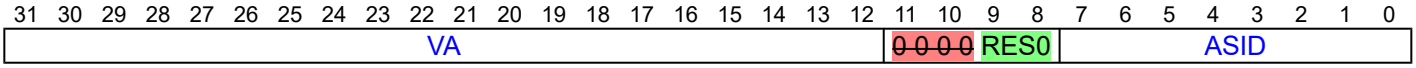
Configuration

Attributes

TLBIMVA is a 32-bit System instruction.

Field descriptions

The TLBIMVA input value bit assignments are:



VA, bits [31:12]

Virtual address to match. Any TLB entries that match the ASID value and VA value will be affected by this operation.

Bits [11:8]

Reserved, RES0.

ASID, bits [7:0]

ASID value to match. Any TLB entries that match the ASID value and VA value will be affected by this System instruction.~~operation.~~

Global TLB entries that match the VA value will be affected by this System instruction~~operation~~, regardless of the value of the ASID field.

Executing the TLBIMVA instruction

Accesses to this instruction use the following encodings:

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
~~opc1~~	~~opc2~~	~~CRn~~	~~coproc~~	~~CRm~~
0b1111	0b000	0b1000	0b0111	0b001
~~0b000~~	~~0b001~~	~~0b1000~~	~~0b1111~~	~~0b0111~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T8 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T8 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TTLB == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR.TTLB == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.FB == '1' then
        TLBIMVAIS(R[t]);
    else
        TLBIMVA(R[t]);
elsif PSTATE.EL == EL2 then
    TLBIMVA(R[t]);
elsif PSTATE.EL == EL3 then
    TLBIMVA(R[t]);
```

27~~13~~/03~~12~~/2019~~2018~~ 21~~16~~:59~~42~~; e5e4db499bf9867a4b93324c4dbac985d3da93766~~379d01e197f1d40720d32d0f84e419e9187e009~~

(old) htmldiff from- (new)

TLBIMVAA, TLB Invalidate by VA, All ASID

The TLBIMVAA characteristics are:

Purpose

Invalidate all cached copies of translation table entries from TLBs that meet the following requirements:

- The entry is a stage 1 translation table entry, from any level of the translation table walk.
- The entry would be used to translate the specified address.
- If EL2 is implemented and enabled in the current Security state, the entry would be used with the current VMID.

From the entries that match these requirement, the entries that are invalidated are required for the following translation regime:

- If executed at Secure EL1 when EL3 is using AArch64, the Secure EL1&0 translation regime.
- If executed in Secure state when EL3 is using AArch32, the Secure PL1&0 translation regime.
- If executed in Non-secure state, the Non-secure PL1&0 translation regime.

The invalidation only applies to the PE that executes this System instruction.

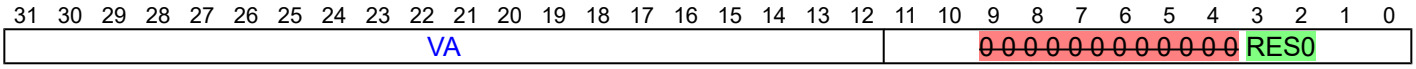
Configuration

Attributes

TLBIMVAA is a 32-bit System instruction.

Field descriptions

The TLBIMVAA input value bit assignments are:



VA, bits [31:12]

Virtual address to match. Any unlocked TLB entries that match the VA will be affected by this System instruction~~operation~~, regardless of the ASID.

Bits [11:0]

Reserved, RES0.

Executing the TLBIMVAA instruction

Accesses to this instruction use the following encodings:

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
~~opc1~~	~~opc2~~	~~CRn~~	~~coproc~~	~~CRm~~
0b1111	0b000	0b1000	0b0111	0b011
~~0b000~~	~~0b011~~	~~0b1000~~	~~0b1111~~	~~0b0111~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T8 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T8 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TTLB == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR.TTLB == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.FB == '1' then
        TLBIMVAAIS(R[t]);
    else
        TLBIMVAA(R[t]);
elsif PSTATE.EL == EL2 then
    TLBIMVAA(R[t]);
elsif PSTATE.EL == EL3 then
    TLBIMVAA(R[t]);
```

27~~13~~/03~~12~~/2019~~2018~~ 21~~16~~:59~~42~~; e5e4db499bf9867a4b93324c4dbac985d3da93766~~379d01e197f1d40720d32d0f84c419c9187c009~~

(old) htmldiff from- (new)

(old) htmldiff from- (new)

TLBIMVAAIS, TLB Invalidate by VA, All ASID, Inner Shareable

The TLBIMVAAIS characteristics are:

Purpose

Invalidate all cached copies of translation table entries from TLBs that meet the following requirements:

- The entry is a stage 1 translation table entry, from any level of the translation table walk.
- The entry would be used to translate the specified address.
- If EL2 is implemented and enabled in the current Security state, the entry would be used with the current VMID.

From the entries that match these requirement, the entries that are invalidated are required for the following translation regime:

- If executed at Secure EL1 when EL3 is using AArch64, the Secure EL1&0 translation regime.
- If executed in Secure state when EL3 is using AArch32, the Secure PL1&0 translation regime.
- If executed in Non-secure state, the Non-secure PL1&0 translation regime.

The invalidation applies to all PEs in the same Inner Shareable shareability domain as the PE that executes this System instruction.~~instructions.~~

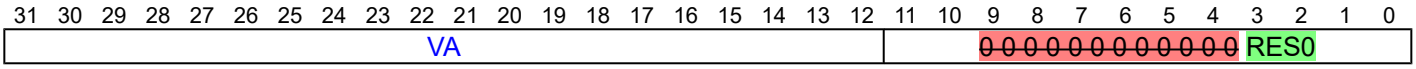
Configuration

Attributes

TLBIMVAAIS is a 32-bit System instruction.

Field descriptions

The TLBIMVAAIS input value bit assignments are:



VA, bits [31:12]

Virtual address to match. Any unlocked TLB entries that match the VA will be affected by this System instruction~~operation~~, regardless of the ASID.

Bits [11:0]

Reserved, RES0.

Executing the TLBIMVAAIS instruction

Accesses to this instruction use the following encodings:

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
~~opc1~~	~~opc2~~	~~CRn~~	~~coproc~~	~~CRm~~
0b1111	0b000	0b1000	0b0011	0b011
~~0b000~~	~~0b011~~	~~0b1000~~	~~0b1111~~	~~0b0011~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T8 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T8 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TTLB == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TTLBIS == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR.TTLB == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR2.TTLBIS == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        TLBIMVAAIS(R[t]);
elsif PSTATE.EL == EL2 then
    TLBIMVAAIS(R[t]);
elsif PSTATE.EL == EL3 then
    TLBIMVAAIS(R[t]);
```

27~~13~~/03~~12~~/2019~~2018~~ 21~~16~~:59~~42~~; e5e4db499bf9867a4b93324c4dbac985d3da93766~~379d01e197f1d40720d32d0f84e419e9187e009~~

(old) htmldiff from- (new)

(old) htmldiff from- (new)

TLBIMVAAL, TLB Invalidate by VA, All ASID, Last level

The TLBIMVAAL characteristics are:

Purpose

Invalidate all cached copies of translation table entries from TLBs that meet the following requirements:

- The entry is a stage 1 translation table entry, from the final level of the translation table walk.
- The entry would be used to translate the specified address.
- If EL2 is implemented and enabled in the current Security state, the entry would be used with the current VMID.

From the entries that match these requirement, the entries that are invalidated are required for the following translation regime:

- If executed at Secure EL1 when EL3 is using AArch64, the Secure EL1&0 translation regime.
- If executed in Secure state when EL3 is using AArch32, the Secure PL1&0 translation regime.
- If executed in Non-secure state, the Non-secure PL1&0 translation regime.

The invalidation only applies to the PE that executes this System instruction.

Configuration

Note

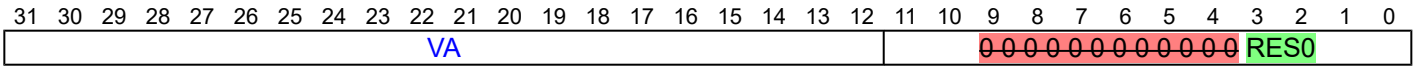
This System instruction is not implemented in architecture versions before Armv8.

Attributes

TLBIMVAAL is a 32-bit System instruction.

Field descriptions

The TLBIMVAAL input value bit assignments are:



VA, bits [31:12]

Virtual address to match. Any unlocked TLB entries that match the VA will be affected by this System instruction~~operation~~, regardless of the ASID.

Bits [11:0]

Reserved, RES0.

Executing the TLBIMVAAL instruction

Accesses to this instruction use the following encodings:

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
~~opc1~~	~~opc2~~	~~CRn~~	~~coproc~~	~~CRm~~

0b1111	0b000	0b1000	0b0111	0b111
~~0b000~~	~~0b111~~	~~0b1000~~	~~0b1111~~	~~0b0111~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T8 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T8 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TTLB == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR.TTLB == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.FB == '1' then
        TLBIMVAALIS(R[t]);
    else
        TLBIMVAAL(R[t]);
elsif PSTATE.EL == EL2 then
    TLBIMVAAL(R[t]);
elsif PSTATE.EL == EL3 then
    TLBIMVAAL(R[t]);
```

27~~13~~/03~~12~~/2019~~2018~~ 21~~16~~:59~~42~~; e5e4db499bf9867a4b93324c4dbac985d3da93766~~379d01e197f1d40720d32d0f84e419e9187e009~~

(old) htmldiff from- (new)

(old) htmldiff from- (new)

TLBIMVAALIS, TLB Invalidate by VA, All ASID, Last level, Inner Shareable

The TLBIMVAALIS characteristics are:

Purpose

Invalidate all cached copies of translation table entries from TLBs that meet the following requirements:

- The entry is a stage 1 translation table entry, from the final level of the translation table walk.
- The entry would be used to translate the specified address.
- If EL2 is implemented and enabled in the current Security state, the entry would be used with the current VMID.

From the entries that match these requirement, the entries that are invalidated are required for the following translation regime:

- If executed at Secure EL1 when EL3 is using AArch64, the Secure EL1&0 translation regime.
- If executed in Secure state when EL3 is using AArch32, the Secure PL1&0 translation regime.
- If executed in Non-secure state, the Non-secure PL1&0 translation regime.

The invalidation applies to all PEs in the same Inner Shareable shareability domain as the PE that executes this System instruction.~~instructions.~~

Configuration

Note

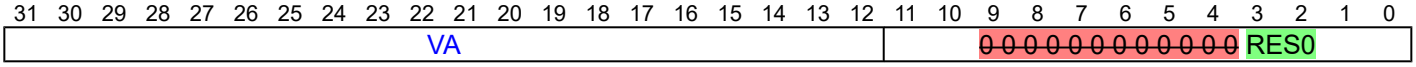
This System instruction is not implemented in architecture versions before Armv8.

Attributes

TLBIMVAALIS is a 32-bit System instruction.

Field descriptions

The TLBIMVAALIS input value bit assignments are:



VA, bits [31:12]

Virtual address to match. Any unlocked TLB entries that match the VA will be affected by this System instruction~~operation~~, regardless of the ASID.

Bits [11:0]

Reserved, RES0.

Executing the TLBIMVAALIS instruction

Accesses to this instruction use the following encodings:

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
~~opc1~~	~~opc2~~	~~CRn~~	~~coproc~~	~~CRm~~
0b1111	0b000	0b1000	0b0011	0b111
~~0b000~~	~~0b111~~	~~0b1000~~	~~0b1111~~	~~0b0011~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T8 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T8 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TTLB == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TTLBIS == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR.TTLB == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR2.TTLBIS == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        TLBIMVAALIS(R[t]);
elsif PSTATE.EL == EL2 then
    TLBIMVAALIS(R[t]);
elsif PSTATE.EL == EL3 then
    TLBIMVAALIS(R[t]);
```

27~~13~~/03~~12~~/2019~~2018~~ 21~~16~~:59~~42~~; e5e4db499bf9867a4b93324c4dbac985d3da9376~~6379d01e197f1d40720d32d0f84e419e9187e009~~

(old) htmldiff from- (new)

TLBIMVAH, TLB Invalidate by VA, Hyp mode

The TLBIMVAH characteristics are:

Purpose

If EL2 is implemented, invalidate all cached copies of translation table entries from TLBs that are from any level of the translation table walk that would be required for the Non-secure EL2 translation regime and used to translate the specified address.

The invalidation only applies to the PE that executes this System instruction.

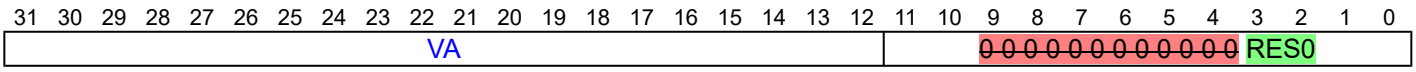
Configuration

Attributes

TLBIMVAH is a 32-bit System instruction.

Field descriptions

The TLBIMVAH input value bit assignments are:



VA, bits [31:12]

Virtual address to match. Any TLB entries that match the ASID value and VA value will be affected by this System instruction.~~operation.~~

Bits [11:0]

Reserved, RES0.

Executing the TLBIMVAH instruction

If this instruction is executed in a Secure privileged mode other than Monitor mode, then the behavior is CONSTRAINED UNPREDICTABLE, and one of the following behaviors must occur:

- The instruction is UNDEFINED.
- The instruction is treated as a NOP.
- The instruction executes as if it had been executed in Monitor mode.

Accesses to this instruction use the following encodings:

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
~~opc1~~	~~opc2~~	~~CRn~~	~~coproc~~	~~CRm~~
0b1111	0b100	0b1000	0b0111	0b001
~~0b100~~	~~0b001~~	~~0b1000~~	~~0b1111~~	~~0b0111~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T8 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T8 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    TLBIMVAH(R[t]);
elsif PSTATE.EL == EL3 then
    if !HaveEL(EL2) then
        UNDEFINED;
    else
        TLBIMVAH(R[t]);
```

27~~13~~/03~~12~~/2019~~2018~~ 21~~16~~:59~~42~~; e5e4db499bf9867a4b93324c4dbac985d3da9376~~6379d01e197f1d40720d32d0f84e419e9187e009~~

(old) htmldiff from- (new)

(old) htmldiff from- (new)

TLBIMVAHIS, TLB Invalidate by VA, Hyp mode, Inner Shareable

The TLBIMVAHIS characteristics are:

Purpose

If EL2 is implemented, invalidate all cached copies of translation table entries from TLBs that are from any level of the translation table walk that would be required for the Non-secure EL2 translation regime and used to translate the specified address.

The invalidation applies to all PEs in the same Inner Shareable shareability domain as the PE that executes this System instruction.~~instructions.~~

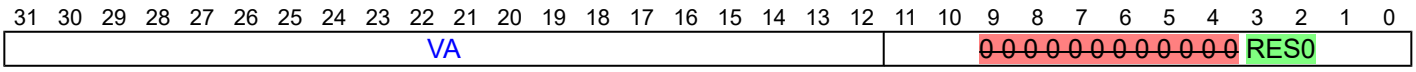
Configuration

Attributes

TLBIMVAHIS is a 32-bit System instruction.

Field descriptions

The TLBIMVAHIS input value bit assignments are:



VA, bits [31:12]

Virtual address to match. Any TLB entries that match the ASID value and VA value will be affected by this System instruction.~~operation.~~

Bits [11:0]

Reserved, RES0.

Executing the TLBIMVAHIS instruction

If this instruction is executed in a Secure privileged mode other than Monitor mode, then the behavior is CONSTRAINED UNPREDICTABLE, and one of the following behaviors must occur:

- The instruction is UNDEFINED.
- The instruction is treated as a NOP.
- The instruction executes as if it had been executed in Monitor mode.

Accesses to this instruction use the following encodings:

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
~~opc1~~	~~opc2~~	~~CRn~~	~~coproc~~	~~CRm~~
0b1111	0b100	0b1000	0b0011	0b001
~~0b100~~	~~0b001~~	~~0b1000~~	~~0b1111~~	~~0b0011~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T8 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T8 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    TLBIMVAHIS(R[t]);
elsif PSTATE.EL == EL3 then
    if !HaveEL(EL2) then
        UNDEFINED;
    else
        TLBIMVAHIS(R[t]);
```

2713/0312/20192018 2116:5942; e5e4db499bf9867a4b93324c4dbac985d3da93766379d01e197f1d40720d32d0f84e419e9187e009

(old) htmldiff from- (new)

(old) htmldiff from- (new)

TLBIMVAIS, TLB Invalidate by VA, Inner Shareable

The TLBIMVAIS characteristics are:

Purpose

Invalidate all cached copies of translation table entries from TLBs that meet the following requirements:

- The entry is a stage 1 translation table entry.
- The entry would be used to translate the specified address, and one of the following applies:
  - The entry is from a level of lookup above the final level and matches the specified ASID.
  - The entry is a global entry from the final level of lookup.
  - The entry is a non-global entry from the final level of lookup that matches the specified ASID.
- If EL2 is implemented and enabled in the current Security state, the entry would be used with the current VMID.

From the entries that match these requirement, the entries that are invalidated are required for the following translation regime:

- If executed at Secure EL1 when EL3 is using AArch64, the Secure EL1&0 translation regime.
- If executed in Secure state when EL3 is using AArch32, the Secure PL1&0 translation regime.
- If executed in Non-secure state, the Non-secure PL1&0 translation regime.

The invalidation applies to all PEs in the same Inner Shareable shareability domain as the PE that executes this System instruction.~~instructions.~~

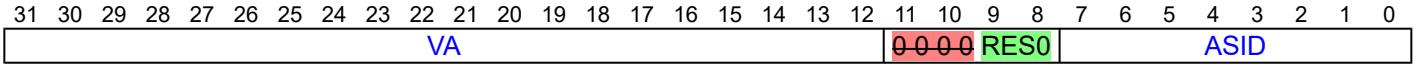
Configuration

Attributes

TLBIMVAIS is a 32-bit System instruction.

Field descriptions

The TLBIMVAIS input value bit assignments are:



VA, bits [31:12]

Virtual address to match. Any TLB entries that match the ASID value and VA value will be affected by this System instruction.~~operation.~~

Bits [11:8]

Reserved, RES0.

ASID, bits [7:0]

ASID value to match. Any TLB entries that match the ASID value and VA value will be affected by this System instruction.~~operation.~~

Global TLB entries that match the VA value will be affected by this System instruction~~operation~~, regardless of the value of the ASID field.

Executing the TLBIMVAIS instruction

Accesses to this instruction use the following encodings:

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
~~opc1~~	~~opc2~~	~~CRn~~	~~coproc~~	~~CRm~~
0b1111	0b000	0b1000	0b0011	0b001
~~0b000~~	~~0b001~~	~~0b1000~~	~~0b1111~~	~~0b0011~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T8 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T8 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TTLB == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TTLBIS == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR.TTLB == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR2.TTLBIS == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        TLBIMVAIS(R[t]);
elsif PSTATE.EL == EL2 then
    TLBIMVAIS(R[t]);
elsif PSTATE.EL == EL3 then
    TLBIMVAIS(R[t]);
```

27~~13~~/03~~12~~/2019~~2018~~ 21~~16~~:59~~42~~; e5e4db499bf9867a4b93324c4dbac985d3da93766~~379d01e197f1d40720d32d0f84e419e9187e009~~

(old) htmldiff from- (new)

(old) htmldiff from- (new)

TLBIMVAL, TLB Invalidate by VA, Last level

The TLBIMVAL characteristics are:

Purpose

Invalidate all cached copies of translation table entries from TLBs that meet the following requirements:

- The entry is a stage 1 translation table entry.
- The entry would be used to translate the specified address, and one of the following applies:
  - The entry is a global entry from the final level of lookup.
  - The entry is a non-global entry from the final level of lookup that matches the specified ASID.
- If EL2 is implemented and enabled in the current Security state, the entry would be used with the current VMID.

From the entries that match these requirement, the entries that are invalidated are required for the following translation regime:

- If executed at Secure EL1 when EL3 is using AArch64, the Secure EL1&0 translation regime.
- If executed in Secure state when EL3 is using AArch32, the Secure PL1&0 translation regime.
- If executed in Non-secure state, the Non-secure PL1&0 translation regime.

The invalidation only applies to the PE that executes this System instruction.

Configuration

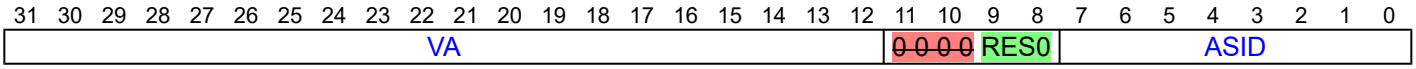
This System instruction is not implemented in architecture versions before Armv8.

Attributes

TLBIMVAL is a 32-bit System instruction.

Field descriptions

The TLBIMVAL input value bit assignments are:



VA, bits [31:12]

Virtual address to match. Any TLB entries that match the ASID value and VA value will be affected by this System instruction.~~operation.~~

Bits [11:8]

Reserved, RES0.

ASID, bits [7:0]

ASID value to match. Any TLB entries that match the ASID value and VA value will be affected by this System instruction.~~operation.~~

Global TLB entries that match the VA value will be affected by this System instruction~~operation~~, regardless of the value of the ASID field.

Executing the TLBIMVAL instruction

Accesses to this instruction use the following encodings:

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
~~opc1~~	~~opc2~~	~~CRn~~	~~coproc~~	~~CRm~~
0b1111	0b000	0b1000	0b0111	0b101
~~0b000~~	~~0b101~~	~~0b1000~~	~~0b1111~~	~~0b0111~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T8 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T8 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TTLB == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR.TTLB == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.FB == '1' then
        TLBIMVALIS(R[t]);
    else
        TLBIMVAL(R[t]);
elsif PSTATE.EL == EL2 then
    TLBIMVAL(R[t]);
elsif PSTATE.EL == EL3 then
    TLBIMVAL(R[t]);
```

27~~13~~/03~~12~~/2019~~2018~~ 21~~16~~:59~~42~~; e5e4db499bf9867a4b93324c4dbac985d3da93766~~379d01e197f1d40720d32d0f84e419e9187e009~~

(old) htmldiff from- (new)

(old) htmldiff from- (new)

TLBIMVALH, TLB Invalidate by VA, Last level, Hyp mode

The TLBIMVALH characteristics are:

Purpose

If EL2 is implemented, invalidate all cached copies of translation table entries from TLBs that are from the final level of the translation table walk that would be required for the Non-secure EL2 translation regime and used to translate the specified address.

The invalidation only applies to the PE that executes this System instruction.

Configuration

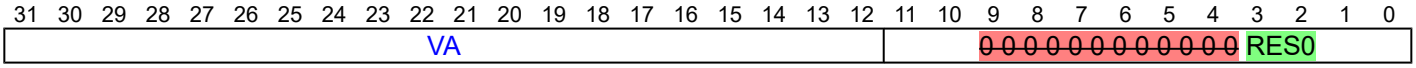
This System instruction is not implemented in architecture versions before Armv8.

Attributes

TLBIMVALH is a 32-bit System instruction.

Field descriptions

The TLBIMVALH input value bit assignments are:



VA, bits [31:12]

Virtual address to match. Any TLB entries that match the ASID value and VA value will be affected by this System instruction.~~operation.~~

Bits [11:0]

Reserved, RES0.

Executing the TLBIMVALH instruction

If this instruction is executed in a Secure privileged mode other than Monitor mode, then the behavior is CONSTRAINED UNPREDICTABLE, and one of the following behaviors must occur:

- The instruction is UNDEFINED.
- The instruction is treated as a NOP.
- The instruction executes as if it had been executed in Monitor mode.

Accesses to this instruction use the following encodings:

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
~~opc1~~	~~opc2~~	~~CRn~~	~~coproc~~	~~CRm~~
0b1111	0b100	0b1000	0b0111	0b101
~~0b100~~	~~0b101~~	~~0b1000~~	~~0b1111~~	~~0b0111~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T8 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T8 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    TLBIMVALH(R[t]);
elsif PSTATE.EL == EL3 then
    if !HaveEL(EL2) then
        UNDEFINED;
    else
        TLBIMVALH(R[t]);
```

2713/0312/20192018 2116:5942; e5e4db499bf9867a4b93324c4dbac985d3da93766379d01e197f1d40720d32d0f84e419e9187e009

(old) htmldiff from- (new)

(old) htmldiff from- (new)

TLBIMVALHIS, TLB Invalidate by VA, Last level, Hyp mode, Inner Shareable

The TLBIMVALHIS characteristics are:

Purpose

If EL2 is implemented, invalidate all cached copies of translation table entries from TLBs that are from the final level of the translation table walk that would be required for the Non-secure EL2 translation regime and used to translate the specified address.

The invalidation applies to all PEs in the same Inner Shareable shareability domain as the PE that executes this System instruction.~~instructions.~~

Configuration

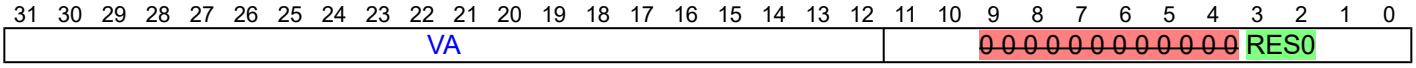
This System instruction is not implemented in architecture versions before Armv8.

Attributes

TLBIMVALHIS is a 32-bit System instruction.

Field descriptions

The TLBIMVALHIS input value bit assignments are:



VA, bits [31:12]

Virtual address to match. Any TLB entries that match the ASID value and VA value will be affected by this System instruction.~~operation.~~

Bits [11:0]

Reserved, RES0.

Executing the TLBIMVALHIS instruction

If this instruction is executed in a Secure privileged mode other than Monitor mode, then the behavior is CONSTRAINED UNPREDICTABLE, and one of the following behaviors must occur:

- The instruction is UNDEFINED.
- The instruction is treated as a NOP.
- The instruction executes as if it had been executed in Monitor mode.

Accesses to this instruction use the following encodings:

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
~~opc1~~	~~opc2~~	~~CRn~~	~~coproc~~	~~CRm~~
0b1111	0b100	0b1000	0b0011	0b101
~~0b100~~	~~0b101~~	~~0b1000~~	~~0b1111~~	~~0b0011~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T8 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T8 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    TLBIMVALHIS(R[t]);
elsif PSTATE.EL == EL3 then
    if !HaveEL(EL2) then
        UNDEFINED;
    else
        TLBIMVALHIS(R[t]);
```

27~~13~~/03~~12~~/2019~~2018~~ 21~~16~~:59~~42~~; e5e4db499bf9867a4b93324c4dbac985d3da9376~~6379d01e197f1d40720d32d0f84e419e9187e009~~

(old) htmldiff from- (new)

(old) htmldiff from- (new)

TLBIMVALIS, TLB Invalidate by VA, Last level, Inner Shareable

The TLBIMVALIS characteristics are:

Purpose

Invalidate all cached copies of translation table entries from TLBs that meet the following requirements:

- The entry is a stage 1 translation table entry.
- The entry would be used to translate the specified address, and one of the following applies:
  - The entry is a global entry from the final level of lookup.
  - The entry is a non-global entry from the final level of lookup that matches the specified ASID.
- If EL2 is implemented and enabled in the current Security state, the entry would be used with the current VMID.

From the entries that match these requirement, the entries that are invalidated are required for the following translation regime:

- If executed at Secure EL1 when EL3 is using AArch64, the Secure EL1&0 translation regime.
- If executed in Secure state when EL3 is using AArch32, the Secure PL1&0 translation regime.
- If executed in Non-secure state, the Non-secure PL1&0 translation regime.

The invalidation applies to all PEs in the same Inner Shareable shareability domain as the PE that executes this System instruction.~~instructions.~~

Configuration

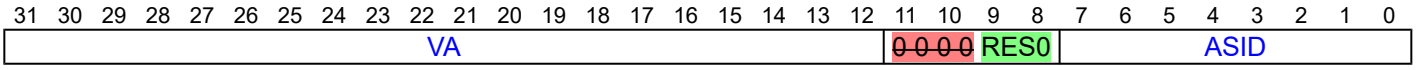
This System instruction is not implemented in architecture versions before Armv8.

Attributes

TLBIMVALIS is a 32-bit System instruction.

Field descriptions

The TLBIMVALIS input value bit assignments are:



VA, bits [31:12]

Virtual address to match. Any TLB entries that match the ASID value and VA value will be affected by this System instruction.~~operation.~~

Bits [11:8]

Reserved, RES0.

ASID, bits [7:0]

ASID value to match. Any TLB entries that match the ASID value and VA value will be affected by this System instruction.~~operation.~~

Global TLB entries that match the VA value will be affected by this System instruction~~operation~~, regardless of the value of the ASID field.

Executing the TLBIMVALIS instruction

Accesses to this instruction use the following encodings:

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
~~opc1~~	~~opc2~~	~~CRn~~	~~coproc~~	~~CRm~~
0b1111	0b000	0b1000	0b0011	0b101
~~0b000~~	~~0b101~~	~~0b1000~~	~~0b1111~~	~~0b0011~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T8 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T8 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TTLB == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TTLBIS == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR.TTLB == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR2.TTLBIS == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        TLBIMVAL(R[t]);
elsif PSTATE.EL == EL2 then
    TLBIMVAL(R[t]);
elsif PSTATE.EL == EL3 then
    TLBIMVAL(R[t]);
```

27~~13~~/03~~12~~/2019~~2018~~ 21~~16~~:59~~42~~; e5e4db499bf9867a4b93324c4dbac985d3da9376~~6379d01e197f1d40720d32d0f84e419e9187e009~~

(old) htmldiff from- (new)

TTBCR, Translation Table Base Control Register

The TTBCR characteristics are:

Purpose

The control register for stage 1 of the PL1&0 translation regime. Its controls include:

- Where the VA range is split between addresses translated using TTBR0 and addresses translated using TTBR1.
- The translation table format used by this stage of translation.

In Armv8.2, when the value of TTBCR.{EAE, T2E} is {1, 1}, TTBCR is used with TTBCR2.

Configuration

AArch32 System register TTBCR bits [31:0] are architecturally mapped to AArch64 System register TCR_EL1[31:0] .

The current translation table format determines which format of the register is used.

~~When EL3 is using AArch32, write access to TTBCR(S) is disabled when the CP15SDISABLE signal is asserted HIGH.~~

Some RW fields of this register have defined reset values. These apply only if the PE resets into an Exception level that is using AArch32. If the PE resets into EL3 using AArch32 then:

- The EAE bit resets to 0 in both the Secure and the Non-secure instances of the register.
- Other reset values apply only to the Secure instance of the register.

Attributes

TTBCR is a 32-bit register.

Field descriptions

The TTBCR bit assignments are:

When TTBCR.EAE == 0:

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
EAE	RES0																									PD1	PD0	RES0		N	
~~31~~	~~30~~	~~29~~	~~28~~	~~27~~	~~26~~	~~25~~	~~24~~	~~23~~	~~22~~	~~21~~	~~20~~	~~19~~	~~18~~	~~17~~	~~16~~	~~15~~	~~14~~	~~13~~	~~12~~	~~11~~	~~10~~	~~9~~	~~8~~	~~7~~	~~6~~	~~5~~	~~4~~	~~3~~	~~2~~	~~1~~	~~0~~
~~EAE~~	~~0~~	~~0~~	~~0~~	~~0~~	~~0~~	~~0~~	~~0~~	~~0~~	~~0~~	~~0~~	~~0~~	~~0~~	~~0~~	~~0~~	~~0~~	~~0~~	~~0~~	~~0~~	~~0~~	~~0~~	~~0~~	~~0~~	~~0~~	~~0~~	~~0~~	~~PD1~~	~~PD0~~	~~0~~		N	

EAE, bit [31]

Extended Address Enable. The meanings of the possible values of this bit are:

EAE	Meaning
0b0	Use the VMSAv8-32 translation system with the Short-descriptor translation table format.

This field resets to 0.

Bits [30:6]

Reserved, RES0.

PD1, bit [5]

Translation table walk disable for translations using TTBR1. This bit controls whether a translation table walk is performed on a TLB miss, for an address that is translated using TTBR1. The encoding of this bit is:

PD1	Meaning
0b0	Perform translation table walks using TTBR1.
0b1	A TLB miss on an address that is translated using TTBR1 generates a Translation fault. No translation table walk is performed.

This field resets to 0.

PD0, bit [4]

Translation table walk disable for translations using TTBR0. This bit controls whether a translation table walk is performed on a TLB miss for an address that is translated using TTBR0. The encoding of this bit is:

PD0	Meaning
0b0	Perform translation table walks using TTBR0.
0b1	A TLB miss on an address that is translated using TTBR0 generates a Translation fault. No translation table walk is performed.

This field resets to 0.

Bit [3]

Reserved, RES0.

N, bits [2:0]

Indicate the width of the base address held in TTBR0. In TTBR0, the base address field is bits[31:14-N]. The value of N also determines:

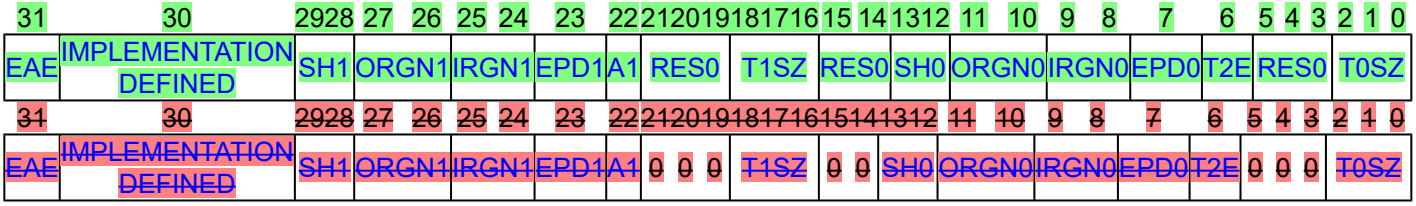
- Whether TTBR0 or TTBR1 is used as the base address for translation table walks.
- The size of the translation table pointed to by TTBR0.

N can take any value from 0 to 7, that is, from 0b000 to 0b111.

When N has its reset value of 0, the translation table base is compatible with Armv5 and Armv6.

This field resets to 0.

When TTBCR.EAE == 1:



EAE, bit [31]

Extended Address Enable. The meanings of the possible values of this bit are:

EAE	Meaning
0b1	Use the VMSAv8-32 translation system with the Long-descriptor translation table format.

This field resets to 0.

IMPLEMENTATION DEFINED, bit [30]

IMPLEMENTATION DEFINED.

This field resets to 0.

SH1, bits [29:28]

Shareability attribute for memory associated with translation table walks using TTBR1. Defined values are:

SH1	Meaning
0b00	Non-shareable.
0b10	Outer Shareable.
0b11	Inner Shareable.

Other values are reserved. The effect of programming this field to a Reserved value is that behavior is CONSTRAINED UNPREDICTABLE, as described in 'Unallocated values in fields of AArch32 System registers and translation table entries' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section K1.1.11.

This field resets to 0.

ORGN1, bits [27:26]

Outer cacheability attribute for memory associated with translation table walks using TTBR1.

ORGN1	Meaning
0b00	Normal memory, Outer Non-cacheable.
0b01	Normal memory, Outer Write-Back Read-Allocate Write-Allocate Cacheable.
0b10	Normal memory, Outer Write-Through Read-Allocate No Write-Allocate Cacheable.
0b11	Normal memory, Outer Write-Back Read-Allocate No Write-Allocate Cacheable.

This field resets to 0.

IRGN1, bits [25:24]

Inner cacheability attribute for memory associated with translation table walks using TTBR1.

IRGN1	Meaning
0b00	Normal memory, Inner Non-cacheable.
0b01	Normal memory, Inner Write-Back Read-Allocate Write-Allocate Cacheable.
0b10	Normal memory, Inner Write-Through Read-Allocate No Write-Allocate Cacheable.
0b11	Normal memory, Inner Write-Back Read-Allocate No Write-Allocate Cacheable.

This field resets to 0.

EPD1, bit [23]

Translation table walk disable for translations using TTBR1. This bit controls whether a translation table walk is performed on a TLB miss, for an address that is translated using TTBR1. The encoding of this bit is:

EPD1	Meaning
0b0	Perform translation table walks using TTBR1.
0b1	A TLB miss on an address that is translated using TTBR1 generates a Translation fault. No translation table walk is performed.

This field resets to 0.

A1, bit [22]

Selects whether TTBR0 or TTBR1 defines the ASID. The encoding of this bit is:

A1	Meaning
0b0	TTBR0.ASID defines the ASID.
0b1	TTBR1.ASID defines the ASID.

This field resets to 0.

Bits [21:19]

Reserved, RES0.

T1SZ, bits [18:16]

See 'Selecting between TTBR0 and TTBR1, VMSAv8-32 Long-descriptor translation table format' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile for how TTBCR.{T1SZ, T0SZ} determine the input address ranges and memory region sizes translated using TTBR0 and TTBR1.

This field resets to 0.

Bits [15:14]

Reserved, RES0.

SH0, bits [13:12]

Shareability attribute for memory associated with translation table walks using TTBR0.

SH0	Meaning
0b00	Non-shareable
0b10	Outer Shareable
0b11	Inner Shareable

Other values are reserved. The effect of programming this field to a Reserved value is that behavior is CONSTRAINED UNPREDICTABLE, as described in 'Unallocated values in fields of AArch32 System registers and translation table entries' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section K1.1.11.

This field resets to 0.

ORGN0, bits [11:10]

Outer cacheability attribute for memory associated with translation table walks using TTBR0.

ORGN0	Meaning
0b00	Normal memory, Outer Non-cacheable.
0b01	Normal memory, Outer Write-Back Read-Allocate Write-Allocate Cacheable.
0b10	Normal memory, Outer Write-Through Read-Allocate No Write-Allocate Cacheable.
0b11	Normal memory, Outer Write-Back Read-Allocate No Write-Allocate Cacheable.

This field resets to 0.

IRGN0, bits [9:8]

Inner cacheability attribute for memory associated with translation table walks using TTBR0.

IRGN0	Meaning
0b00	Normal memory, Inner Non-cacheable.
0b01	Normal memory, Inner Write-Back Read-Allocate Write-Allocate Cacheable.
0b10	Normal memory, Inner Write-Through Read-Allocate No Write-Allocate Cacheable.
0b11	Normal memory, Inner Write-Back Read-Allocate No Write-Allocate Cacheable.

This field resets to 0.

EPD0, bit [7]

Translation table walk disable for translations using TTBR0. This bit controls whether a translation table walk is performed on a TLB miss, for an address that is translated using TTBR0. The encoding of this bit is:

EPD0	Meaning
0b0	Perform translation table walks using TTBR0.
0b1	A TLB miss on an address that is translated using TTBR0 generates a Translation fault. No translation table walk is performed.

This field resets to 0.

T2E, bit [6]

When ARMv8.2-AA32HPD is implemented:

TTBCR2 Enable.

T2E	Meaning
0b0	TTBCR2 is disabled. The contents of TTBCR2 are treated as 0 for all purposes other than reading or writing the register.
0b1	TTBCR2 is enabled.

If TTBCR.EAE==0, then the behavior is as if the bit is 0.

Otherwise:

Reserved, RES0.

Bits [5:3]

Reserved, RES0.

T0SZ, bits [2:0]

See 'Selecting between TTBR0 and TTBR1, VMSAv8-32 Long-descriptor translation table format' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile for how TTBCR.{T1SZ, T0SZ} determine the input address ranges and memory region sizes translated using TTBR0 and TTBR1.

This field resets to 0.

Accessing the TTBCR

Accesses to this register use the following encodings:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
~~opc1~~	~~opc2~~	~~CRn~~	~~coproc~~	~~CRm~~
0b1111	0b000	0b0010	0b0000	0b010
~~0b000~~	~~0b010~~	~~0b0010~~	~~0b1111~~	~~0b0000~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T2 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T2 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TRVM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TRVM == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) then
        if SCR.NS == '0' then
            return TTBCR_S;
        else
            return TTBCR_NS;
    else
        return TTBCR;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && ELUsingAArch32(EL3) then
        return TTBCR_NS;
    else
        return TTBCR;
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        return TTBCR_S;
    else
        return TTBCR_NS;
```

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
~~opc1~~	~~opc2~~	~~CRn~~	~~coproc~~	~~CRm~~
0b1111	0b000	0b0010	0b0000	0b010
~~0b000~~	~~0b010~~	~~0b0010~~	~~0b1111~~	~~0b0000~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T2 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T2 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) && SCR.NS == '0' && CP15SDISABLE == HIGH then
        UNDEFINED;
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) && SCR.NS == '0' && CP15SDISABLE2 == HIGH then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TVM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TVM == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) then
        if SCR.NS == '0' then
            TTBCR_S = R[t];
        else
            TTBCR_NS = R[t];
    else
        TTBCR = R[t];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && ELUsingAArch32(EL3) then
        TTBCR_NS = R[t];
    else
        TTBCR = R[t];
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' && CP15SDISABLE == HIGH then
        UNDEFINED;
    elsif SCR.NS == '0' && CP15SDISABLE2 == HIGH then
        UNDEFINED;
    else
        if SCR.NS == '0' then
            TTBCR_S = R[t];
        else
            TTBCR_NS = R[t];
```

2713/0312/20192018 2116:5942; e5e4db499bf9867a4b93324c4dbac985d3da93766379d01e197f1d40720d32d0f84e419e9187e009

(old) htmldiff from- (new)

TTBCR2, Translation Table Base Control Register 2

The TTBCR2 characteristics are:

Purpose

The second control register for stage 1 of the PL1&0 translation regime.

If ARMv8.2-AA32HPD is not implemented then this register is not implemented and its encoding is unallocated. Otherwise:

- When the value of TTBCR.{EAE, T2E} is not {1, 1} the contents of TTBCR2 are treated as zero for all purposes other than reading or writing the register.
- When the value of TTBCR.{EAE, T2E} is {1, 1} TTBCR2 is used with TTBCR.

Configuration

AArch32 System register TTBCR2 bits [31:0] are architecturally mapped to AArch64 System register TCR_EL1[63:32] .

This register is present only from Armv8.2. Otherwise, direct accesses to TTBCR2 are UNDEFINED.

~~When EL3 is using AArch32, write access to TTBCR2(S) is disabled when the CP15SDISABLE signal is asserted HIGH.~~

RW fields in this register reset to architecturally UNKNOWN values.

Attributes

TTBCR2 is a 32-bit register.

Field descriptions

The TTBCR2 bit assignments are:

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RES0													HWU162	HWU161	HWU160	HWU159	HWU062	HWU061	HWU060	HWU059	HPD1	HPD0	RES0							
~~0~~	~~0~~	~~0~~	~~0~~	~~0~~	~~0~~	~~0~~	~~0~~	~~0~~	~~0~~	~~0~~	~~0~~	~~0~~	HWU162	HWU161	HWU160	HWU159	HWU062	HWU061	HWU060	HWU059	HPD1	HPD0	0	0	0	0	0	0	0	0

Bits [31:19]

Reserved, RES0.

HWU162, bit [18]

When ARMv8.2-TTPBHA is implemented:

Hardware Use. Indicates IMPLEMENTATION DEFINED hardware use of bit[62] of the stage 1 translation table Block or Page entry for translations using TTBR1.

HWU162	Meaning
0b0	For translations using TTBR1, bit[62] of each stage 1 translation table Block or Page entry cannot be used by hardware for an IMPLEMENTATION DEFINED purpose.
0b1	For translations using TTBR1, bit[62] of each stage 1 translation table Block or Page entry can be used by hardware for an IMPLEMENTATION DEFINED purpose if the value of TTBCR2.HPD1 is 1.

The Effective value of this field is 0 if the value of TTBCR2.HPD1 is 0 or the value of TTBCR.T2E is 0.

This field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HWU161, bit [17]

When ARMv8.2-TTPBHA is implemented:

Hardware Use. Indicates IMPLEMENTATION DEFINED hardware use of bit[61] of the stage 1 translation table Block or Page entry for translations using TTBR1.

HWU161	Meaning
0b0	For translations using TTBR1, bit[61] of each stage 1 translation table Block or Page entry cannot be used by hardware for an IMPLEMENTATION DEFINED purpose.
0b1	For translations using TTBR1, bit[61] of each stage 1 translation table Block or Page entry can be used by hardware for an IMPLEMENTATION DEFINED purpose if the value of TTBCR2.HPD1 is 1.

The Effective value of this field is 0 if the value of TTBCR2.HPD1 is 0 or the value of TTBCR.T2E is 0.

This field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HWU160, bit [16]

When ARMv8.2-TTPBHA is implemented:

Hardware Use. Indicates IMPLEMENTATION DEFINED hardware use of bit[60] of the stage 1 translation table Block or Page entry for translations using TTBR1.

HWU160	Meaning
0b0	For translations using TTBR1, bit[60] of each stage 1 translation table Block or Page entry cannot be used by hardware for an IMPLEMENTATION DEFINED purpose.
0b1	For translations using TTBR1, bit[60] of each stage 1 translation table Block or Page entry can be used by hardware for an IMPLEMENTATION DEFINED purpose if the value of TTBCR2.HPD1 is 1.

The Effective value of this field is 0 if the value of TTBCR2.HPD1 is 0 or the value of TTBCR.T2E is 0.

This field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HWU159, bit [15]

When ARMv8.2-TTPBHA is implemented:

Hardware Use. Indicates IMPLEMENTATION DEFINED hardware use of bit[59] of the stage 1 translation table Block or Page entry for translations using TTBR1.

HWU159	Meaning
0b0	For translations using TTBR1, bit[59] of each stage 1 translation table Block or Page entry cannot be used by hardware for an IMPLEMENTATION DEFINED purpose.
0b1	For translations using TTBR1, bit[59] of each stage 1 translation table Block or Page entry can be used by hardware for an IMPLEMENTATION DEFINED purpose if the value of TTBCR2.HPD1 is 1.

The Effective value of this field is 0 if the value of TTBCR2.HPD1 is 0 or the value of TTBCR.T2E is 0.

This field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HWU062, bit [14]

When ARMv8.2-TTPBHA is implemented:

Hardware Use. Indicates IMPLEMENTATION DEFINED hardware use of bit[62] of the stage 1 translation table Block or Page entry for translations using TTBR0.

HWU062	Meaning
0b0	For translations using TTBR0, bit[62] of each stage 1 translation table Block or Page entry cannot be used by hardware for an IMPLEMENTATION DEFINED purpose.
0b1	For translations using TTBR0, bit[62] of each stage 1 translation table Block or Page entry can be used by hardware for an IMPLEMENTATION DEFINED purpose if the value of TTBCR2.HPD0 is 1.

The Effective value of this field is 0 if the value of TTBCR2.HPD0 is 0 or the value of TTBCR.T2E is 0.

This field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HWU061, bit [13]

When ARMv8.2-TTPBHA is implemented:

Hardware Use. Indicates IMPLEMENTATION DEFINED hardware use of bit[61] of the stage 1 translation table Block or Page entry for translations using TTBR0.

HWU061	Meaning
0b0	For translations using TTBR0, bit[61] of each stage 1 translation table Block or Page entry cannot be used by hardware for an IMPLEMENTATION DEFINED purpose.
0b1	For translations using TTBR0, bit[61] of each stage 1 translation table Block or Page entry can be used by hardware for an IMPLEMENTATION DEFINED purpose if the value of TTBCR2.HPD0 is 1.

The Effective value of this field is 0 if the value of TTBCR2.HPD0 is 0 or the value of TTBCR.T2E is 0.

This field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HWU060, bit [12]

When ARMv8.2-TTPBHA is implemented:

Hardware Use. Indicates IMPLEMENTATION DEFINED hardware use of bit[60] of the stage 1 translation table Block or Page entry for translations using TTBR0.

HWU060	Meaning
0b0	For translations using TTBR0, bit[60] of each stage 1 translation table Block or Page entry cannot be used by hardware for an IMPLEMENTATION DEFINED purpose.
0b1	For translations using TTBR0, bit[60] of each stage 1 translation table Block or Page entry can be used by hardware for an IMPLEMENTATION DEFINED purpose if the value of TTBCR2.HPD0 is 1.

The Effective value of this field is 0 if the value of TTBCR2.HPD0 is 0 or the value of TTBCR.T2E is 0.

This field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HWU059, bit [11]

When ARMv8.2-TTPBHA is implemented:

Hardware Use. Indicates IMPLEMENTATION DEFINED hardware use of bit[59] of the stage 1 translation table Block or Page entry for translations using TTBR0.

HWU059	Meaning
0b0	For translations using TTBR0, bit[59] of each stage 1 translation table Block or Page entry cannot be used by hardware for an IMPLEMENTATION DEFINED purpose.
0b1	For translations using TTBR0, bit[59] of each stage 1 translation table Block or Page entry can be used by hardware for an IMPLEMENTATION DEFINED purpose if the value of TTBCR2.HPD0 is 1.

The Effective value of this field is 0 if the value of TTBCR2.HPD0 is 0 or the value of TTBCR.T2E is 0.

This field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HPD1, bit [10]

When ARMv8.2-AA32HPD is implemented:

Hierarchical Permission Disables. This affects the hierarchical control bits, APTable, XNTable, and PXNTable, in the translation tables pointed to by TTBR1.

HPD1	Meaning
0b0	Hierarchical permissions are enabled.
0b1	Hierarchical permissions are disabled if TTBCR.T2E == 1.

When disabled, the permissions are treated as if the bits are 0.

The Effective value of this field is 0 if the value of TTBCR.T2E is 0.

This field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HPD0, bit [9]

When ARMv8.2-AA32HPD is implemented:

Hierarchical Permission Disables. This affects the hierarchical control bits, APTable, XNTable, and PXNTable, in the translation tables pointed to by TTBR0.

HPD0	Meaning
0b0	Hierarchical permissions are enabled.
0b1	Hierarchical permissions are disabled if TTBCR.T2E ==1.

When disabled, the permissions are treated is as if the bits are 0.

The Effective value of this field is 0 if the value of TTBCR.T2E is 0.

This field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [8:0]

Reserved, RES0.

Accessing the TTBCR2

Accesses to this register use the following encodings:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
~~opc1~~	~~opc2~~	~~CRn~~	~~coproc~~	~~CRm~~
0b1111	0b000	0b0010	0b0000	0b011
~~0b000~~	~~0b011~~	~~0b0010~~	~~0b1111~~	~~0b0000~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T2 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T2 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TRVM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TRVM == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) then
        if SCR.NS == '0' then
            return TTBCR2_S;
        else
            return TTBCR2_NS;
    else
        return TTBCR2;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && ELUsingAArch32(EL3) then
        return TTBCR2_NS;
    else
        return TTBCR2;
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        return TTBCR2_S;
    else
        return TTBCR2_NS;
```

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
~~opc1~~	~~opc2~~	~~CRn~~	~~coproc~~	~~CRm~~
0b1111	0b000	0b0010	0b0000	0b011
~~0b000~~	~~0b011~~	~~0b0010~~	~~0b1111~~	~~0b0000~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T2 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T2 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) && SCR.NS == '0' && CP15SDISABLE == HIGH then
        UNDEFINED;
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) && SCR.NS == '0' && CP15SDISABLE2 == HIGH then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TVM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TVM == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) then
        if SCR.NS == '0' then
            TTBCR2_S = R[t];
        else
            TTBCR2_NS = R[t];
    else
        TTBCR2 = R[t];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && ELUsingAArch32(EL3) then
        TTBCR2_NS = R[t];
    else
        TTBCR2 = R[t];
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' && CP15SDISABLE == HIGH then
        UNDEFINED;
    elsif SCR.NS == '0' && CP15SDISABLE2 == HIGH then
        UNDEFINED;
    else
        if SCR.NS == '0' then
            TTBCR2_S = R[t];
        else
            TTBCR2_NS = R[t];
```

2713/0312/20192018 2116:5942; e5e4db499bf9867a4b93324c4dbac985d3da93766379d01e197f1d40720d32d0f84e419e9187e009

(old) htmldiff from- (new)

TTBR1, Translation Table Base Register 1

The TTBR1 characteristics are:

Purpose

Holds the base address of the translation table for the initial lookup for stage 1 of the translation of an address from the higher VA range in the PL1&0 translation regime, and other information for this translation regime.

Configuration

AArch32 System register TTBR1 bits [63:0] are architecturally mapped to AArch64 System register TTBR1_EL1[63:0] .

TTBCR.EAE determines which TTBR1 format is used:

- TTBCR.EAE == 0b0: 32-bit format is used. TTBR1[63:32] are ignored.
- TTBCR.EAE == 0b1: 64-bit format is used.

Used in conjunction with the TTBCR. When the 64-bit TTBR1 format is used, cacheability and shareability information is held in the TTBCR, not in TTBR1.

RW fields in this register reset to architecturally UNKNOWN values.

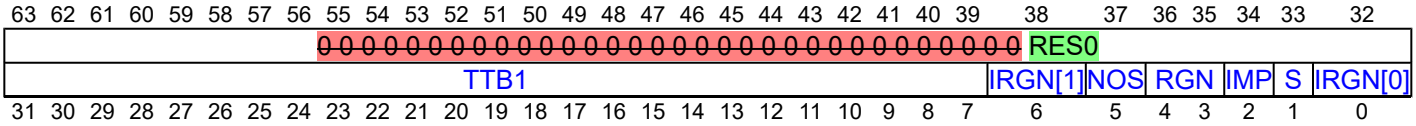
Attributes

TTBR1 is a 64-bit register that can also be accessed as a 32-bit value. If it is accessed as a 32-bit register, accesses read and write bits [31:0] and do not modify bits [63:32].

Field descriptions

The TTBR1 bit assignments are:

When TTBCR.EAE == 0:



Bits [63:32]

Reserved, RES0.

TTB1, bits [31:7]

Translation table base address, bits[31:14]. Register bits [13:7] are RES0, with the additional requirement that if these bits are not all zero, this is a misaligned translation table base address, with effects that are CONSTRAINED UNPREDICTABLE, and must be one of the following:

- Register bits [13:7] are treated as if all the bits are zero. The value read back from these bits is either the value written or zero.
- The result of the calculation of an address for a translation table walk using this register can be corrupted in those bits that are nonzero.

This field resets to an architecturally UNKNOWN value.

IRGN[1], bit [6]

This field is bit[1] of IRGN[1:0].

Inner region bits. IRGN[1:0] indicate the Inner Cacheability attributes for the memory associated with the translation table walks. The possible values of IRGN[1:0] are:

IRGN	Meaning
0b00	Normal memory, Inner Non-cacheable.
0b01	Normal memory, Inner Write-Back Write-Allocate Cacheable.
0b10	Normal memory, Inner Write-Through Cacheable.
0b11	Normal memory, Inner Write-Back no Write-Allocate Cacheable.

Note

The encoding of the IRGN bits is counter-intuitive, with register bit[6] being IRGN[0] and register bit[0] being IRGN[1]. This encoding is chosen to give a consistent encoding of memory region types and to ensure that software written for Armv7 without the Multiprocessing Extensions can run unmodified on an implementation that includes the functionality introduced by the ARMv7 Multiprocessing Extensions.

The IRGN field is split as follows:

- IRGN[1] is TTBR1[6].
- IRGN[0] is TTBR1[0].

This field resets to an architecturally UNKNOWN value.

NOS, bit [5]

Not Outer Shareable. When the value of TTBR1.S is 1, indicates whether the memory associated with a translation table walk is Inner Shareable or Outer Shareable:

NOS	Meaning
0b0	Memory is Outer Shareable.
0b1	Memory is Inner Shareable.

This bit is ignored when the value of TTBR1.S is 0.

This field resets to an architecturally UNKNOWN value.

RGN, bits [4:3]

Region bits. Indicates the Outer cacheability attributes for the memory associated with the translation table walks:

RGN	Meaning
0b00	Normal memory, Outer Non-cacheable.
0b01	Normal memory, Outer Write-Back Write-Allocate Cacheable.
0b10	Normal memory, Outer Write-Through Cacheable.
0b11	Normal memory, Outer Write-Back no Write-Allocate Cacheable.

This field resets to an architecturally UNKNOWN value.

IMP, bit [2]

The effect of this bit is IMPLEMENTATION DEFINED. If the translation table implementation does not include any IMPLEMENTATION DEFINED features this bit is RES0.

This field resets to an architecturally UNKNOWN value.

S, bit [1]

Shareable. Indicates whether the memory associated with the translation table walks is Non-shareable:

S	Meaning
0b0	Memory is Non-shareable.
0b1	Memory is shareable. The TTBR1.NOS field indicates whether the memory is Inner Shareable or Outer Shareable.

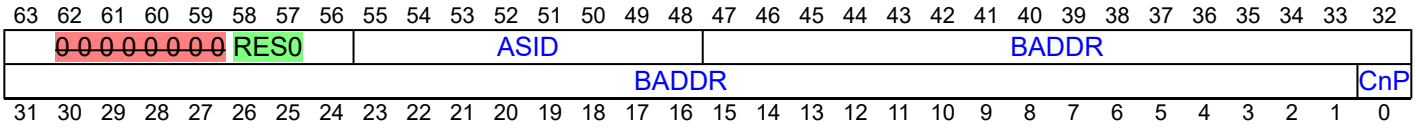
This field resets to an architecturally UNKNOWN value.

IRGN[0], bit [0]

This field is bit[0] of IRGN[1:0].

See IRGN[1] for the field description.

When TTBCR.EAE == 1:



Bits [63:56]

Reserved, RES0.

ASID, bits [55:48]

An ASID for the translation table base address. The TTBCR.A1 field selects either TTBR0.ASID or TTBR1.ASID.

This field resets to an architecturally UNKNOWN value.

BADDR, bits [47:1]

Translation table base address, bits[47:x], Bits [x-1:1] are RES0, with the additional requirement that if bits[x-1:3] are not all zero, this is a misaligned translation table base address, with effects that are CONSTRAINED UNPREDICTABLE, and must be one of the following:

- Register bits [x-1:3] are treated as if all the bits are zero. The value read back from these bits is either the value written or zero.
- The result of the calculation of an address for a translation table walk using this register can be corrupted in those bits that are nonzero.

x is determined from the value of TTBCR.T1SZ as follows:

- If TTBCR.T1SZ is 0 or 1, x = 5 - TTBCR.T1SZ.
- If TTBCR.T1SZ is greater than 1, x = 14 - TTBCR.T1SZ.

If bits[47:40] of the translation table base address are not zero, an Address size fault is generated.

This field resets to an architecturally UNKNOWN value.

CnP, bit [0]

When ARMv8.2-TTCNP is implemented:

Common not Private. When TTBCR.EAE ==1, this bit indicates whether each entry that is pointed to by TTBR1 is a member of a common set that can be used by every PE in the Inner Shareable domain for which the value of TTBR1.CnP is 1.

CnP	Meaning
0b0	The translation table entries pointed to by this instance of TTBR1, for the current ASID, are permitted to differ from corresponding entries for this instance of TTBR1 for other PEs in the Inner Shareable domain. This is not affected by: • The value of TTBR1.CnP on those other PEs. • The value of TTBCR.EAE on those other PEs. • The value of the current ASID or, for the Non-secure instance of TTBR1, the value of the current VMID.
0b1	The translation table entries pointed to by this instance of TTBR1 are the same as the translation table entries for every other PE in the Inner Shareable domain for which the value of TTBR1.CnP is 1 for this instance of TTBR1 and all of the following apply: • The translation table entries are pointed to by this instance of TTBR1. • The value of the applicable TTBCR.EAE field is 1. • The ASID is the same as the current ASID. • For the Non-secure instance of TTBR1, the VMID is the same as the current VMID.

When a TLB combines entries from stage 1 translation and stage 2 translation into a single entry, that entry can only be shared between different PEs if the value of the CnP bit is 1 for both stage 1 and stage 2.

Note

If the value of the TTBR1.CnP bit is 1 on multiple PEs in the same Inner Shareable domain and those TTBR1s do not point to the same translation table entries when the other conditions specified for the case when the value of CnP is 1 apply, then the results of translations are CONSTRAINED UNPREDICTABLE, see 'CONSTRAINED UNPREDICTABLE behaviors due to caching of control or data values' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

This field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Accessing the TTBR1

Accesses to this register use the following encodings:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
~~opc1~~	~~opc2~~	~~CRn~~	~~coproc~~	~~CRm~~
0b1111	0b000	0b0010	0b0000	0b001
~~0b000~~	~~0b001~~	~~0b0010~~	~~0b1111~~	~~0b0000~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T2 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T2 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TRVM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TRVM == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) then
        if SCR.NS == '0' then
            return TTBR1_S<31:0>;
        else
            return TTBR1_NS<31:0>;
    else
        return TTBR1<31:0>;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && ELUsingAArch32(EL3) then
        return TTBR1_NS<31:0>;
    else
        return TTBR1<31:0>;
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        return TTBR1_S<31:0>;
    else
        return TTBR1_NS<31:0>;
```

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
~~opc1~~	~~opc2~~	~~CRn~~	~~coproc~~	~~CRm~~
0b1111	0b000	0b0010	0b0000	0b001
~~0b000~~	~~0b001~~	~~0b0010~~	~~0b1111~~	~~0b0000~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T2 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T2 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) && SCR.NS == '0' && CP15SDISABLE == HIGH then
        UNDEFINED;
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) && SCR.NS == '0' && CP15SDISABLE2 == HIGH then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TVM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TVM == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) then
        if SCR.NS == '0' then
            TTBR1_S = ZeroExtend(R[t]);
        else
            TTBR1_NS = ZeroExtend(R[t]);
    else
        TTBR1 = ZeroExtend(R[t]);
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && ELUsingAArch32(EL3) then
        TTBR1_NS = ZeroExtend(R[t]);
    else
        TTBR1 = ZeroExtend(R[t]);
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' && CP15SDISABLE == HIGH then
        UNDEFINED;
    elsif SCR.NS == '0' && CP15SDISABLE2 == HIGH then
        UNDEFINED;
    else
        if SCR.NS == '0' then
            TTBR1_S = ZeroExtend(R[t]);
        else
            TTBR1_NS = ZeroExtend(R[t]);
```

```
MRRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <Rt2>, <CRm>
```

coproc	CRm	opc1
~~opc1~~	~~coproc~~	~~CRm~~
0b1111	0b0010	0b0001
~~0b0001~~	~~0b1111~~	~~0b0010~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T2 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T2 == '1' then
        AArch32.TakeHypTrapException(0x04);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TRVM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TRVM == '1' then
        AArch32.TakeHypTrapException(0x04);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) then
        if SCR.NS == '0' then
            return TTBR1_S;
        else
            return TTBR1_NS;
    else
        return TTBR1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && ELUsingAArch32(EL3) then
        return TTBR1_NS;
    else
        return TTBR1;
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        return TTBR1_S;
    else
        return TTBR1_NS;
```

```
MCRR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <Rt2>, <CRm>
```

coproc	CRm	opc1
~~opc1~~	~~coproc~~	~~CRm~~
0b1111	0b0010	0b0001
~~0b0001~~	~~0b1111~~	~~0b0010~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T2 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T2 == '1' then
        AArch32.TakeHypTrapException(0x04);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) && SCR.NS == '0' && CP15SDISABLE == HIGH then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TVM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TVM == '1' then
        AArch32.TakeHypTrapException(0x04);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) then
        if SCR.NS == '0' then
            TTBR1_S = R[t2]:R[t];
        else
            TTBR1_NS = R[t2]:R[t];
    else
        TTBR1 = R[t2]:R[t];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && ELUsingAArch32(EL3) then
        TTBR1_NS = R[t2]:R[t];
    else
        TTBR1 = R[t2]:R[t];
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' && CP15SDISABLE == HIGH then
        UNDEFINED;
    else
        if SCR.NS == '0' then
            TTBR1_S = R[t2]:R[t];
        else
            TTBR1_NS = R[t2]:R[t];
```

27~~13~~/03~~12~~/2019~~2018~~ 21~~16~~:59~~42~~; e5e4db499bf9867a4b93324c4dbac985d3da9376~~6379d01e197f1d40720d32d0f84e419e9187e009~~

(old) htmldiff from- (new)

(old) htmldiff from- (new)

VBAR, Vector Base Address Register

The VBAR characteristics are:

Purpose

When high exception vectors are not selected, holds the vector base address for exceptions that are not taken to Monitor mode or to Hyp mode.

Software must program VBAR(NS) with the required initial value as part of the PE boot sequence.

Configuration

AArch32 System register VBAR bits [31:0] are architecturally mapped to AArch64 System register VBAR_EL1[31:0] .

~~When EL3 is using AArch32, write access to VBAR(S) is disabled when the CP15SDISABLE signal is asserted HIGH.~~

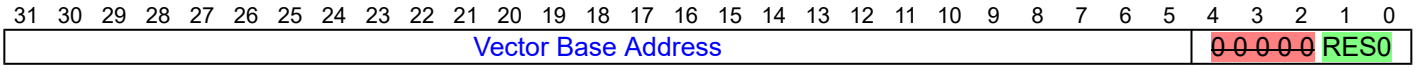
Some or all RW fields of this register have defined reset values. These apply only if the PE resets into an Exception level that is using AArch32. If the PE resets into EL3 using AArch32 they apply only to the Secure instance of the register. Otherwise, RW fields in this register reset to architecturally UNKNOWN values.

Attributes

VBAR is a 32-bit register.

Field descriptions

The VBAR bit assignments are:



Bits [31:5]

Vector Base Address. Bits[31:5] of the base address of the exception vectors for exceptions taken to this Exception level. Bits[4:0] of an exception vector are the exception offset.

This field resets to an IMPLEMENTATION DEFINED value.

Bits [4:0]

Reserved, RES0.

Accessing the VBAR

Accesses to this register use the following encodings:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
~~opc1~~	~~opc2~~	~~CRn~~	~~coproc~~	~~CRm~~
0b1111	0b000	0b1100	0b0000	0b000
~~0b000~~	~~0b000~~	~~0b1100~~	~~0b1111~~	~~0b0000~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T12 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T12 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) then
        if SCR.NS == '0' then
            return VBAR_S;
        else
            return VBAR_NS;
    else
        return VBAR;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && ELUsingAArch32(EL3) then
        return VBAR_NS;
    else
        return VBAR;
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        return VBAR_S;
    else
        return VBAR_NS;
```

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
~~opc1~~	~~opc2~~	~~CRn~~	~~coproc~~	~~CRm~~
0b1111	0b000	0b1100	0b0000	0b000
~~0b000~~	~~0b000~~	~~0b1100~~	~~0b1111~~	~~0b0000~~

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T12 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T12 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) && SCR.NS == '0' && CP15SDISABLE == HIGH then
        UNDEFINED;
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) && SCR.NS == '0' && CP15SDISABLE2 == HIGH then
        UNDEFINED;
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) then
        if SCR.NS == '0' then
            VBAR_S = R[t];
        else
            VBAR_NS = R[t];
    else
        VBAR = R[t];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && ELUsingAArch32(EL3) then
        VBAR_NS = R[t];
    else
        VBAR = R[t];
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' && CP15SDISABLE == HIGH then
        UNDEFINED;
    elsif SCR.NS == '0' && CP15SDISABLE2 == HIGH then
        UNDEFINED;
    else
        if SCR.NS == '0' then
            VBAR_S = R[t];
        else
            VBAR_NS = R[t];
```

2713/0312/20192018 2116:5942; e5e4db499bf9867a4b93324c4dbac985d3da93766379d01e197f1d40720d32d0f84e419e9187e009

(old) htmldiff from- (new)

(old) htmldiff from- (new)

System Register index by instruction and encoding

Below are indexes for registers and operations accessed in the following ways:

For AArch32

- MCR/MRC
- MCRR/MRRC
- MRS/MSR
- VMRS/VMSR

For AArch64

- AT
- CFP
- CPP
- DC
- DVP
- IC
- MRS/MSR
- TLBI

Registers and operations in AArch32

Accessed using MCR/MRC:

		Register selectors				
coproc	opc1	CRn	CRm	opc2	Name	Description
0b1110	0b000	0b0000	0b0000	0b000	DBGDIDR	Debug ID Register
0b1110	0b000	0b0000	0b0000	0b010	DBGDTRRXext	Debug OS Lock Data Transfer Register, Receive, External View
0b1110	0b000	0b0000	0b0001	0b000	DBGDSCRint	Debug Status and Control Register, Internal View
0b1110	0b000	0b0000	0b0010	0b000	DBGDCCINT	DCC Interrupt Enable Register
0b1110	0b000	0b0000	0b0010	0b010	DBGDSCRext	Debug Status and Control Register, External View
0b1110	0b000	0b0000	0b0011	0b010	DBGDTRTXext	Debug OS Lock Data Transfer Register, Transmit
0b1110	0b000	0b0000	0b0101	0b000	DBGDTRRXint	Debug Data Transfer Register, Receive
0b1110	0b000	0b0000	0b0101	0b000	DBGDTRTXint	Debug Data Transfer Register, Transmit
0b1110	0b000	0b0000	0b0110	0b000	DBGWFAR	Debug Watchpoint

Register selectors						
coproc	opc1	CRn	CRm	opc2	Name	Description
						Fault Address Register
0b1110	0b000	0b0000	0b0110	0b010	DBGOSECCR	Debug OS Lock Exception Catch Control Register
0b1110	0b000	0b0000	0b0111	0b000	DBGVCR	Debug Vector Catch Register
0b1110	0b000	0b0000	0bnnnn	0b100	DBGBVR<n>	Debug Breakpoint Value Registers
0b1110	0b000	0b0000	0bnnnn	0b101	DBGBCR<n>	Debug Breakpoint Control Registers
0b1110	0b000	0b0000	0bnnnn	0b110	DBGWVR<n>	Debug Watchpoint Value Registers
0b1110	0b000	0b0000	0bnnnn	0b111	DBGWCR<n>	Debug Watchpoint Control Registers
0b1110	0b000	0b0001	0b0000	0b000	DBGDRAR	Debug ROM Address Register
0b1110	0b000	0b0001	0b0000	0b100	DBGOSLAR	Debug OS Lock Access Register
0b1110	0b000	0b0001	0b0001	0b100	DBGOSLSR	Debug OS Lock Status Register
0b1110	0b000	0b0001	0b0011	0b100	DBGOSDLR	Debug OS Double Lock Register
0b1110	0b000	0b0001	0b0100	0b100	DBGPRCR	Debug Power Control Register
0b1110	0b000	0b0001	0bnnnn	0b001	DBGBXVR<n>	Debug Breakpoint Extended Value Registers
0b1110	0b000	0b0010	0b0000	0b000	DBGDSAR	Debug Self Address Register
0b1110	0b000	0b0111	0b0000	0b111	DBGDEVID2	Debug Device ID register 2
0b1110	0b000	0b0111	0b0001	0b111	DBGDEVID1	Debug Device ID register 1
0b1110	0b000	0b0111	0b0010	0b111	DBGDEVID	Debug Device ID register 0
0b1110	0b000	0b0111	0b1000	0b110	DBGCLAIMSET	Debug Claim Tag Set register
0b1110	0b000	0b0111	0b1001	0b110	DBGCLAIMCLR	Debug Claim Tag Clear register
0b1110	0b000	0b0111	0b1110	0b110	DBGAUTHSTATUS	Debug Authentication Status register
0b1110	0b111	0b0000	0b0000	0b000	JIDR	Jazelle ID Register
0b1110	0b111	0b0001	0b0000	0b000	JOSCR	Jazelle OS Control Register

Register selectors					Name	Description
coproc	opc1	CRn	CRm	opc2		
0b1110	0b111	0b0010	0b0000	0b000	JMCR	Jazelle Main Configuration Register
0b1111	0b000	0b0000	0b0000	0b000	MIDR	Main ID Register
0b1111	0b000	0b0000	0b0000	0b001	CTR	Cache Type Register
0b1111	0b000	0b0000	0b0000	0b010	TCMTR	TCM Type Register
0b1111	0b000	0b0000	0b0000	0b011	TLBTR	TLB Type Register
0b1111	0b000	0b0000	0b0000	0b101	MPIDR	Multiprocessor Affinity Register
0b1111	0b000	0b0000	0b0000	0b110	REVIDR	Revision ID Register
0b1111	0b000	0b0000	0b0001	0b000	ID_PFR0	Processor Feature Register 0
0b1111	0b000	0b0000	0b0001	0b001	ID_PFR1	Processor Feature Register 1
0b1111	0b000	0b0000	0b0001	0b010	ID_DFR0	Debug Feature Register 0
0b1111	0b000	0b0000	0b0001	0b011	ID_AFR0	Auxiliary Feature Register 0
0b1111	0b000	0b0000	0b0001	0b100	ID_MMFR0	Memory Model Feature Register 0
0b1111	0b000	0b0000	0b0001	0b101	ID_MMFR1	Memory Model Feature Register 1
0b1111	0b000	0b0000	0b0001	0b110	ID_MMFR2	Memory Model Feature Register 2
0b1111	0b000	0b0000	0b0001	0b111	ID_MMFR3	Memory Model Feature Register 3
0b1111	0b000	0b0000	0b0010	0b000	ID_ISAR0	Instruction Set Attribute Register 0
0b1111	0b000	0b0000	0b0010	0b001	ID_ISAR1	Instruction Set Attribute Register 1
0b1111	0b000	0b0000	0b0010	0b010	ID_ISAR2	Instruction Set Attribute Register 2
0b1111	0b000	0b0000	0b0010	0b011	ID_ISAR3	Instruction Set Attribute Register 3
0b1111	0b000	0b0000	0b0010	0b100	ID_ISAR4	Instruction Set Attribute Register 4
0b1111	0b000	0b0000	0b0010	0b101	ID_ISAR5	Instruction Set Attribute Register 5
0b1111	0b000	0b0000	0b0010	0b110	ID_MMFR4	Memory Model Feature Register 4

Register selectors					Name	Description
coproc	opc1	CRn	CRm	opc2		
0b1111	0b000	0b0000	0b0010	0b111	ID_ISAR6	Instruction Set Attribute Register 6
0b1111	0b000	0b0000	0b0011	0b100	ID_PFR2	Processor Feature Register 2
0b1111	0b000	0b0001	0b0000	0b000	SCTLR	System Control Register
0b1111	0b000	0b0001	0b0000	0b001	ACTLR	Auxiliary Control Register
0b1111	0b000	0b0001	0b0000	0b010	CPACR	Architectural Feature Access Control Register
0b1111	0b000	0b0001	0b0000	0b011	ACTLR2	Auxiliary Control Register 2
0b1111	0b000	0b0001	0b0001	0b000	SCR	Secure Configuration Register
0b1111	0b000	0b0001	0b0001	0b001	SDER	Secure Debug Enable Register
0b1111	0b000	0b0001	0b0001	0b010	NSACR	Non-Secure Access Control Register
0b1111	0b000	0b0001	0b0010	0b001	TRFCR	Trace Filter Control Register
0b1111	0b000	0b0001	0b0011	0b001	SDCR	Secure Debug Control Register
0b1111	0b000	0b0010	0b0000	0b000	TTBR0	Translation Table Base Register 0
0b1111	0b000	0b0010	0b0000	0b001	TTBR1	Translation Table Base Register 1
0b1111	0b000	0b0010	0b0000	0b010	TTBCR	Translation Table Base Control Register
0b1111	0b000	0b0010	0b0000	0b011	TTBCR2	Translation Table Base Control Register 2
0b1111	0b000	0b0011	0b0000	0b000	DACR	Domain Access Control Register
0b1111	0b000	0b0100	0b0110	0b000	ICC_PMR	Interrupt Controller Interrupt Priority Mask Register
~~0b1111~~	~~0b000~~	~~0b0100~~	~~0b0110~~	~~0b000~~	~~ICV_PMR~~	~~Interrupt Controller Virtual Interrupt Priority Mask Register~~
0b1111	0b000	0b0101	0b0000	0b000	DFSR	Data Fault Status Register

Register selectors					Name	Description
coproc	opc1	CRn	CRm	opc2		
0b1111	0b000	0b0101	0b0000	0b001	IFSR	Instruction Fault Status Register
0b1111	0b000	0b0101	0b0001	0b000	ADFSR	Auxiliary Data Fault Status Register
0b1111	0b000	0b0101	0b0001	0b001	AIFSR	Auxiliary Instruction Fault Status Register
0b1111	0b000	0b0101	0b0011	0b000	ERRIDR	Error Record ID Register
0b1111	0b000	0b0101	0b0011	0b001	ERRSELR	Error Record Select Register
0b1111	0b000	0b0101	0b0100	0b000	ERXFR	Selected Error Record Feature Register
0b1111	0b000	0b0101	0b0100	0b001	ERXCTLR	Selected Error Record Control Register
0b1111	0b000	0b0101	0b0100	0b010	ERXSTATUS	Selected Error Record Primary Status Register
0b1111	0b000	0b0101	0b0100	0b011	ERXADDR	Selected Error Record Address Register
0b1111	0b000	0b0101	0b0100	0b100	ERXFR2	Selected Error Record Feature Register 2
0b1111	0b000	0b0101	0b0100	0b101	ERXCTLR2	Selected Error Record Control Register 2
0b1111	0b000	0b0101	0b0100	0b111	ERXADDR2	Selected Error Record Address Register 2
0b1111	0b000	0b0101	0b0101	0b000	ERXMISC0	Selected Error Record Miscellaneous Register 0
0b1111	0b000	0b0101	0b0101	0b001	ERXMISC1	Selected Error Record Miscellaneous Register 1
0b1111	0b000	0b0101	0b0101	0b010	ERXMISC4	Selected Error Record Miscellaneous Register 4
0b1111	0b000	0b0101	0b0101	0b011	ERXMISC5	Selected Error Record Miscellaneous Register 5
0b1111	0b000	0b0101	0b0101	0b100	ERXMISC2	Selected Error Record Miscellaneous Register 2
0b1111	0b000	0b0101	0b0101	0b101	ERXMISC3	Selected Error Record Miscellaneous Register 3
0b1111	0b000	0b0101	0b0101	0b110	ERXMISC6	Selected Error Record Miscellaneous Register 6

Register selectors					Name	Description
coproc	opc1	CRn	CRm	opc2		
0b1111	0b000	0b0101	0b0101	0b111	ERXMISC7	Selected Error Record Miscellaneous Register 7
0b1111	0b000	0b0110	0b0000	0b000	DFAR	Data Fault Address Register
0b1111	0b000	0b0110	0b0000	0b010	IFAR	Instruction Fault Address Register
0b1111	0b000	0b0111	0b0001	0b000	ICIALLUIS	Instruction Cache Invalidate All to PoU, Inner Shareable
0b1111	0b000	0b0111	0b0001	0b110	BPIALLIS	Branch Predictor Invalidate All, Inner Shareable
0b1111	0b000	0b0111	0b0011	0b100	CFPRCTX	Control Flow Prediction Restriction by Context
0b1111	0b000	0b0111	0b0011	0b101	DVPRCTX	Data Value Prediction Restriction by Context
0b1111	0b000	0b0111	0b0011	0b111	CPPRCTX	Cache Prefetch Prediction Restriction by Context
0b1111	0b000	0b0111	0b0100	0b000	PAR	Physical Address Register
0b1111	0b000	0b0111	0b0101	0b000	ICIALLU	Instruction Cache Invalidate All to PoU
0b1111	0b000	0b0111	0b0101	0b001	ICIMVAU	Instruction Cache line Invalidate by VA to PoU
0b1111	0b000	0b0111	0b0101	0b100	CP15ISB	Instruction Synchronization Barrier System instruction
0b1111	0b000	0b0111	0b0101	0b110	BPIALL	Branch Predictor Invalidate All
0b1111	0b000	0b0111	0b0101	0b111	BPIMVA	Branch Predictor Invalidate by VA
0b1111	0b000	0b0111	0b0110	0b001	DCIMVAC	Data Cache line Invalidate by VA to PoC
0b1111	0b000	0b0111	0b0110	0b010	DCISW	Data Cache line Invalidate by Set/Way
0b1111	0b000	0b0111	0b1000	0b000	ATS1CPR	Address Translate Stage 1 Current state PL1 Read

Register selectors					Name	Description
coproc	opc1	CRn	CRm	opc2		
0b1111	0b000	0b0111	0b1000	0b001	ATS1CPW	Address Translate Stage 1 Current state PL1 Write
0b1111	0b000	0b0111	0b1000	0b010	ATS1CUR	Address Translate Stage 1 Current state Unprivileged Read
0b1111	0b000	0b0111	0b1000	0b011	ATS1CUW	Address Translate Stage 1 Current state Unprivileged Write
0b1111	0b000	0b0111	0b1000	0b100	ATS12NSOPR	Address Translate Stages 1 and 2 Non-secure Only PL1 Read
0b1111	0b000	0b0111	0b1000	0b101	ATS12NSOPW	Address Translate Stages 1 and 2 Non-secure Only PL1 Write
0b1111	0b000	0b0111	0b1000	0b110	ATS12NSOUR	Address Translate Stages 1 and 2 Non-secure Only Unprivileged Read
0b1111	0b000	0b0111	0b1000	0b111	ATS12NSOUW	Address Translate Stages 1 and 2 Non-secure Only Unprivileged Write
0b1111	0b000	0b0111	0b1001	0b000	ATS1CPRP	Address Translate Stage 1 Current state PL1 Read PAN
0b1111	0b000	0b0111	0b1001	0b001	ATS1CPWP	Address Translate Stage 1 Current state PL1 Write PAN
0b1111	0b000	0b0111	0b1010	0b001	DCCMVAC	Data Cache line Clean by VA to PoC
0b1111	0b000	0b0111	0b1010	0b010	DCCSW	Data Cache line Clean by Set/ Way
0b1111	0b000	0b0111	0b1010	0b100	CP15DSB	Data Synchronization Barrier System instruction
0b1111	0b000	0b0111	0b1010	0b101	CP15DMB	Data Memory Barrier System instruction
0b1111	0b000	0b0111	0b1011	0b001	DCCMVAU	Data Cache line Clean by VA to PoU
0b1111	0b000	0b0111	0b1110	0b001	DCCIMVAC	Data Cache line Clean and Invalidate by VA to PoC

Register selectors					Name	Description
coproc	opc1	CRn	CRm	opc2		
0b1111	0b000	0b0111	0b1110	0b010	DCCISW	Data Cache line Clean and Invalidate by Set/Way
0b1111	0b000	0b1000	0b0011	0b000	TLBIALLIS	TLB Invalidate All, Inner Shareable
0b1111	0b000	0b1000	0b0011	0b001	TLBIMVAIS	TLB Invalidate by VA, Inner Shareable
0b1111	0b000	0b1000	0b0011	0b010	TLBIASIDIS	TLB Invalidate by ASID match, Inner Shareable
0b1111	0b000	0b1000	0b0011	0b011	TLBIMVAAIS	TLB Invalidate by VA, All ASID, Inner Shareable
0b1111	0b000	0b1000	0b0011	0b101	TLBIMVALIS	TLB Invalidate by VA, Last level, Inner Shareable
0b1111	0b000	0b1000	0b0011	0b111	TLBIMVAALIS	TLB Invalidate by VA, All ASID, Last level, Inner Shareable
0b1111	0b000	0b1000	0b0101	0b000	ITLBIALL	Instruction TLB Invalidate All
0b1111	0b000	0b1000	0b0101	0b001	ITLBIMVA	Instruction TLB Invalidate by VA
0b1111	0b000	0b1000	0b0101	0b010	ITLBIASID	Instruction TLB Invalidate by ASID match
0b1111	0b000	0b1000	0b0110	0b000	DTLBIALL	Data TLB Invalidate All
0b1111	0b000	0b1000	0b0110	0b001	DTLBIMVA	Data TLB Invalidate by VA
0b1111	0b000	0b1000	0b0110	0b010	DTLBIASID	Data TLB Invalidate by ASID match
0b1111	0b000	0b1000	0b0111	0b000	TLBIALL	TLB Invalidate All
0b1111	0b000	0b1000	0b0111	0b001	TLBIMVA	TLB Invalidate by VA
0b1111	0b000	0b1000	0b0111	0b010	TLBIASID	TLB Invalidate by ASID match
0b1111	0b000	0b1000	0b0111	0b011	TLBIMVAA	TLB Invalidate by VA, All ASID
0b1111	0b000	0b1000	0b0111	0b101	TLBIMVAL	TLB Invalidate by VA, Last level
0b1111	0b000	0b1000	0b0111	0b111	TLBIMVAAL	TLB Invalidate by VA, All ASID, Last level
0b1111	0b000	0b1001	0b1100	0b000	PMCR	Performance Monitors Control Register

Register selectors					Name	Description
coproc	opc1	CRn	CRm	opc2		
0b1111	0b000	0b1001	0b1100	0b001	PMCNTENSET	Performance Monitors Count Enable Set register
0b1111	0b000	0b1001	0b1100	0b010	PMCNTENCLR	Performance Monitors Count Enable Clear register
0b1111	0b000	0b1001	0b1100	0b011	PMOVSR	Performance Monitors Overflow Flag Status Register
0b1111	0b000	0b1001	0b1100	0b100	PMSWINC	Performance Monitors Software Increment register
0b1111	0b000	0b1001	0b1100	0b101	PMSELR	Performance Monitors Event Counter Selection Register
0b1111	0b000	0b1001	0b1100	0b110	PMCEID0	Performance Monitors Common Event Identification register 0
0b1111	0b000	0b1001	0b1100	0b111	PMCEID1	Performance Monitors Common Event Identification register 1
0b1111	0b000	0b1001	0b1101	0b000	PMCCNTR	Performance Monitors Cycle Count Register
0b1111	0b000	0b1001	0b1101	0b001	PMXEVTYPER	Performance Monitors Selected Event Type Register
0b1111	0b000	0b1001	0b1101	0b010	PMXEVCNTR	Performance Monitors Selected Event Count Register
0b1111	0b000	0b1001	0b1110	0b000	PMUSERENR	Performance Monitors User Enable Register
0b1111	0b000	0b1001	0b1110	0b001	PMINTENSET	Performance Monitors Interrupt Enable Set register
0b1111	0b000	0b1001	0b1110	0b010	PMINTENCLR	Performance Monitors Interrupt Enable Clear register
0b1111	0b000	0b1001	0b1110	0b011	PMOVSSET	Performance Monitors Overflow Flag Status Set register
0b1111	0b000	0b1001	0b1110	0b100	PMCEID2	Performance Monitors Common Event Identification register 2

Register selectors					Name	Description
coproc	opc1	CRn	CRm	opc2		
0b1111	0b000	0b1001	0b1110	0b101	PMCEID3	Performance Monitors Common Event Identification register 3
0b1111	0b000	0b1001	0b1110	0b110	PMMIR	Performance Monitors Machine Identification Register
0b1111	0b000	0b1010	0b0010	0b000	MAIR0	Memory Attribute Indirection Register 0
0b1111	0b000	0b1010	0b0010	0b000	PRRR	Primary Region Remap Register
0b1111	0b000	0b1010	0b0010	0b001	MAIR1	Memory Attribute Indirection Register 1
0b1111	0b000	0b1010	0b0010	0b001	NMRR	Normal Memory Remap Register
0b1111	0b000	0b1010	0b0011	0b000	AMAIR0	Auxiliary Memory Attribute Indirection Register 0
0b1111	0b000	0b1010	0b0011	0b001	AMAIR1	Auxiliary Memory Attribute Indirection Register 1
0b1111	0b000	0b1100	0b0000	0b000	VBAR	Vector Base Address Register
0b1111	0b000	0b1100	0b0000	0b001	MVBAR	Monitor Vector Base Address Register
0b1111	0b000	0b1100	0b0000	0b001	RVBAR	Reset Vector Base Address Register
0b1111	0b000	0b1100	0b0000	0b010	RMR	Reset Management Register
0b1111	0b000	0b1100	0b0001	0b000	ISR	Interrupt Status Register
0b1111	0b000	0b1100	0b0001	0b001	DISR	Deferred Interrupt Status Register
0b1111	0b000	0b1100	0b1000	0b000	ICC_IAR0	Interrupt Controller Interrupt Acknowledge Register 0
0b1111	0b000	0b1100	0b1000	0b000	ICV_IAR0	Interrupt Controller Virtual Interrupt Acknowledge Register 0
0b1111	0b000	0b1100	0b1000	0b001	ICC_EOIR0	Interrupt Controller End Of Interrupt Register 0

Register selectors					Name	Description
coproc	opc1	CRn	CRm	opc2		
~~0b1111~~	~~0b000~~	~~0b1100~~	~~0b1000~~	~~0b001~~	~~ICV_EOIR0~~	~~Interrupt Controller Virtual End Of Interrupt Register 0~~
0b1111	0b000	0b1100	0b1000	0b010	ICC_HPPIR0	Interrupt Controller Highest Priority Pending Interrupt Register 0
~~0b1111~~	~~0b000~~	~~0b1100~~	~~0b1000~~	~~0b010~~	~~ICV_HPPIR0~~	~~Interrupt Controller Virtual Highest Priority Pending Interrupt Register 0~~
0b1111	0b000	0b1100	0b1000	0b011	ICC_BPR0	Interrupt Controller Binary Point Register 0
~~0b1111~~	~~0b000~~	~~0b1100~~	~~0b1000~~	~~0b011~~	~~ICV_BPR0~~	~~Interrupt Controller Virtual Binary Point Register 0~~
0b1111	0b000	0b1100	0b1000	0b1[n:1:0]	ICC_AP0R<n>	Interrupt Controller Active Priorities Group 0 Registers
~~0b1111~~	~~0b000~~	~~0b1100~~	~~0b1000~~	~~0b1[n:1:0]~~	~~ICV_AP0R<n>~~	~~Interrupt Controller Virtual Active Priorities Group 0 Registers~~
0b1111	0b000	0b1100	0b1001	0b0[n:1:0]	ICC_AP1R<n>	Interrupt Controller Active Priorities Group 1 Registers
0b1111	0b000	0b1100	0b1011	0b001	ICC_DIR	Interrupt Controller Deactivate Interrupt Register
0b1111	0b000	0b1100	0b1011	0b011	ICC_RPR	Interrupt Controller Running Priority Register
~~0b1111~~	~~0b000~~	~~0b1100~~	~~0b1011~~	~~0b011~~	~~ICV_RPR~~	~~Interrupt Controller Virtual Running Priority Register~~
0b1111	0b000	0b1100	0b1100	0b000	ICC_IAR1	Interrupt Controller Interrupt Acknowledge Register 1
~~0b1111~~	~~0b000~~	~~0b1100~~	~~0b1100~~	~~0b000~~	~~ICV_IAR1~~	~~Interrupt Controller Virtual Interrupt~~

Register selectors					Name	Description
coproc	opc1	CRn	CRm	opc2		
						~~Acknowledge Register 1~~
0b1111	0b000	0b1100	0b1100	0b001	ICC_EOIR1	Interrupt Controller End Of Interrupt Register 1
0b1111	0b000	0b1100	0b1100	0b010	ICC_HPPIR1	Interrupt Controller Highest Priority Pending Interrupt Register 1
~~0b1111~~	~~0b000~~	~~0b1100~~	~~0b1100~~	~~0b010~~	~~ICV_HPPIR1~~	~~Interrupt Controller Virtual Highest Priority Pending Interrupt Register 1~~
0b1111	0b000	0b1100	0b1100	0b011	ICC_BPR1	Interrupt Controller Binary Point Register 1
0b1111	0b000	0b1100	0b1100	0b100	ICC_CTLR	Interrupt Controller Control Register
0b1111	0b000	0b1100	0b1100	0b101	ICC_SRE	Interrupt Controller System Register Enable register
0b1111	0b000	0b1100	0b1100	0b110	ICC_IGRPEN0	Interrupt Controller Interrupt Group 0 Enable register
~~0b1111~~	~~0b000~~	~~0b1100~~	~~0b1100~~	~~0b110~~	~~ICV_IGRPEN0~~	~~Interrupt Controller Virtual Interrupt Group 0 Enable register~~
0b1111	0b000	0b1100	0b1100	0b111	ICC_IGRPEN1	Interrupt Controller Interrupt Group 1 Enable register
0b1111	0b000	0b1101	0b0000	0b000	FCSEIDR	FCSE Process ID register
0b1111	0b000	0b1101	0b0000	0b001	CONTEXTIDR	Context ID Register
0b1111	0b000	0b1101	0b0000	0b010	TPIDRURW	PL0 Read/Write Software Thread ID Register
0b1111	0b000	0b1101	0b0000	0b011	TPIDRURO	PL0 Read-Only Software Thread ID Register
0b1111	0b000	0b1101	0b0000	0b100	TPIDRPRW	PL1 Software Thread ID Register
0b1111	0b000	0b1101	0b0010	0b000	AMCR	Activity Monitors

Register selectors					Name	Description
coproc	opc1	CRn	CRm	opc2		
						Control Register
0b1111	0b000	0b1101	0b0010	0b001	AMCFGR	Activity Monitors Configuration Register
0b1111	0b000	0b1101	0b0010	0b010	AMCGCR	Activity Monitors Counter Group Configuration Register
0b1111	0b000	0b1101	0b0010	0b011	AMUSERENR	Activity Monitors User Enable Register
0b1111	0b000	0b1101	0b0010	0b100	AMCNTENCLR0	Activity Monitors Count Enable Clear Register 0
0b1111	0b000	0b1101	0b0010	0b101	AMCNTENSET0	Activity Monitors Count Enable Set Register 0
0b1111	0b000	0b1101	0b0011	0b000	AMCNTENCLR1	Activity Monitors Count Enable Clear Register 1
0b1111	0b000	0b1101	0b0011	0b001	AMCNTENSET1	Activity Monitors Count Enable Set Register 1
0b1111	0b000	0b1101	0b011[n:3]	0b[n:2:0]	AMEVTYPER0<n>	Activity Monitors Event Type Registers 0
0b1111	0b000	0b1101	0b111[n:3]	0b[n:2:0]	AMEVTYPER1<n>	Activity Monitors Event Type Registers 1
0b1111	0b000	0b1110	0b0000	0b000	CNTFRQ	Counter-timer Frequency register
0b1111	0b000	0b1110	0b0001	0b000	CNTKCTL	Counter-timer Kernel Control register
0b1111	0b000	0b1110	0b0010	0b000	CNTP_TVAL	Counter-timer Physical Timer TimerValue register
0b1111	0b000	0b1110	0b0010	0b001	CNTP_CTL	Counter-timer Physical Timer Control register
0b1111	0b000	0b1110	0b0011	0b000	CNTV_TVAL	Counter-timer Virtual Timer TimerValue register
0b1111	0b000	0b1110	0b0011	0b001	CNTV_CTL	Counter-timer Virtual Timer Control register
0b1111	0b000	0b1110	0b10[n:4:3]	0b[n:2:0]	PMEVCNTR<n>	Performance Monitors Event Count Registers
0b1111	0b000	0b1110	0b1111	0b111	PMCCFILTR	Performance Monitors Cycle

Register selectors					Name	Description
coproc	opc1	CRn	CRm	opc2		
						Count Filter Register
0b1111	0b000	0b1110	0b11[n:4:3]	0b[n:2:0]	PMEVTYPER<n>	Performance Monitors Event Type Registers
0b1111	0b001	0b0000	0b0000	0b000	CCSIDR	Current Cache Size ID Register
0b1111	0b001	0b0000	0b0000	0b001	CLIDR	Cache Level ID Register
0b1111	0b001	0b0000	0b0000	0b010	CCSIDR2	Current Cache Size ID Register 2
0b1111	0b001	0b0000	0b0000	0b111	AIDR	Auxiliary ID Register
0b1111	0b010	0b0000	0b0000	0b000	CSSELR	Cache Size Selection Register
0b1111	0b011	0b0100	0b0101	0b000	DSPSR	Debug Saved Program Status Register
0b1111	0b011	0b0100	0b0101	0b001	DLR	Debug Link Register
0b1111	0b100	0b0000	0b0000	0b000	VPIDR	Virtualization Processor ID Register
0b1111	0b100	0b0000	0b0000	0b101	VMPIDR	Virtualization Multiprocessor ID Register
0b1111	0b100	0b0001	0b0000	0b000	HSCTLR	Hyp System Control Register
0b1111	0b100	0b0001	0b0000	0b001	HACTLR	Hyp Auxiliary Control Register
0b1111	0b100	0b0001	0b0000	0b011	HACTLR2	Hyp Auxiliary Control Register 2
0b1111	0b100	0b0001	0b0001	0b000	HCR	Hyp Configuration Register
0b1111	0b100	0b0001	0b0001	0b001	HDCR	Hyp Debug Control Register
0b1111	0b100	0b0001	0b0001	0b010	HCPTR	Hyp Architectural Feature Trap Register
0b1111	0b100	0b0001	0b0001	0b011	HSTR	Hyp System Trap Register
0b1111	0b100	0b0001	0b0001	0b100	HCR2	Hyp Configuration Register 2
0b1111	0b100	0b0001	0b0001	0b111	HACR	Hyp Auxiliary Configuration Register
0b1111	0b100	0b0001	0b0010	0b001	HTRFCR	Hyp Trace Filter Control Register
0b1111	0b100	0b0010	0b0000	0b010	HTCR	Hyp Translation Control Register

Register selectors					Name	Description
coproc	opc1	CRn	CRm	opc2		
0b1111	0b100	0b0010	0b0001	0b010	VTCR	Virtualization Translation Control Register
0b1111	0b100	0b0101	0b0001	0b000	HADFSR	Hyp Auxiliary Data Fault Status Register
0b1111	0b100	0b0101	0b0001	0b001	HAIFSR	Hyp Auxiliary Instruction Fault Status Register
0b1111	0b100	0b0101	0b0010	0b000	HSR	Hyp Syndrome Register
0b1111	0b100	0b0101	0b0010	0b011	VDFSR	Virtual SError Exception Syndrome Register
0b1111	0b100	0b0110	0b0000	0b000	HDFAR	Hyp Data Fault Address Register
0b1111	0b100	0b0110	0b0000	0b010	HIFAR	Hyp Instruction Fault Address Register
0b1111	0b100	0b0110	0b0000	0b100	HPFAR	Hyp IPA Fault Address Register
0b1111	0b100	0b0111	0b1000	0b000	ATS1HR	Address Translate Stage 1 Hyp mode Read
0b1111	0b100	0b0111	0b1000	0b001	ATS1HW	Address Translate Stage 1 Hyp mode Write
0b1111	0b100	0b1000	0b0000	0b001	TLBIIPAS2IS	TLB Invalidate by Intermediate Physical Address, Stage 2, Inner Shareable
0b1111	0b100	0b1000	0b0000	0b101	TLBIIPAS2LIS	TLB Invalidate by Intermediate Physical Address, Stage 2, Last level, Inner Shareable
0b1111	0b100	0b1000	0b0011	0b000	TLBIALLHIS	TLB Invalidate All, Hyp mode, Inner Shareable
0b1111	0b100	0b1000	0b0011	0b001	TLBIMVAHIS	TLB Invalidate by VA, Hyp mode, Inner Shareable
0b1111	0b100	0b1000	0b0011	0b100	TLBIALLNSNHIS	TLB Invalidate All, Non-Secure Non-Hyp, Inner Shareable
0b1111	0b100	0b1000	0b0011	0b101	TLBIMVALHIS	TLB Invalidate by VA, Last level, Hyp mode, Inner Shareable
0b1111	0b100	0b1000	0b0100	0b001	TLBIIPAS2	TLB Invalidate by Intermediate

Register selectors					Name	Description
coproc	opc1	CRn	CRm	opc2		
						Physical Address, Stage 2
0b1111	0b100	0b1000	0b0100	0b101	TLBIIPAS2L	TLB Invalidate by Intermediate Physical Address, Stage 2, Last level
0b1111	0b100	0b1000	0b0111	0b000	TLBIALLH	TLB Invalidate All, Hyp mode
0b1111	0b100	0b1000	0b0111	0b001	TLBIMVAH	TLB Invalidate by VA, Hyp mode
0b1111	0b100	0b1000	0b0111	0b100	TLBIALLNSNH	TLB Invalidate All, Non-Secure Non-Hyp
0b1111	0b100	0b1000	0b0111	0b101	TLBIMVALH	TLB Invalidate by VA, Last level, Hyp mode
0b1111	0b100	0b1010	0b0010	0b000	HMAIR0	Hyp Memory Attribute Indirection Register 0
0b1111	0b100	0b1010	0b0010	0b001	HMAIR1	Hyp Memory Attribute Indirection Register 1
0b1111	0b100	0b1010	0b0011	0b000	HAMAIR0	Hyp Auxiliary Memory Attribute Indirection Register 0
0b1111	0b100	0b1010	0b0011	0b001	HAMAIR1	Hyp Auxiliary Memory Attribute Indirection Register 1
0b1111	0b100	0b1100	0b0000	0b000	HVBAR	Hyp Vector Base Address Register
0b1111	0b100	0b1100	0b0000	0b010	HRMR	Hyp Reset Management Register
0b1111	0b100	0b1100	0b0001	0b001	VDISR	Virtual Deferred Interrupt Status Register
0b1111	0b100	0b1100	0b1000	0b0[n:1:0]	ICH_AP0R<n>	Interrupt Controller Hyp Active Priorities Group 0 Registers
0b1111	0b100	0b1100	0b1001	0b0[n:1:0]	ICH_AP1R<n>	Interrupt Controller Hyp Active Priorities Group 1 Registers
0b1111	0b100	0b1100	0b1001	0b101	ICC_HSRE	Interrupt Controller Hyp System Register Enable register
0b1111	0b100	0b1100	0b1011	0b000	ICH_HCR	Interrupt Controller Hyp

Register selectors					Name	Description
coproc	opc1	CRn	CRm	opc2		
						Control Register
0b1111	0b100	0b1100	0b1011	0b001	ICH_VTR	Interrupt Controller VGIC Type Register
0b1111	0b100	0b1100	0b1011	0b010	ICH_MISR	Interrupt Controller Maintenance Interrupt State Register
0b1111	0b100	0b1100	0b1011	0b011	ICH_EISR	Interrupt Controller End of Interrupt Status Register
0b1111	0b100	0b1100	0b1011	0b101	ICH_ELRSR	Interrupt Controller Empty List Register Status Register
0b1111	0b100	0b1100	0b1011	0b111	ICH_VMCR	Interrupt Controller Virtual Machine Control Register
0b1111	0b100	0b1100	0b110[n:3]	0b[n:2:0]	ICH_LR<n>	Interrupt Controller List Registers
0b1111	0b100	0b1100	0b111[n:3]	0b[n:2:0]	ICH_LRC<n>	Interrupt Controller List Registers
0b1111	0b100	0b1101	0b0000	0b010	HTPIDR	Hyp Software Thread ID Register
0b1111	0b100	0b1110	0b0001	0b000	CNTHCTL	Counter-timer Hyp Control register
0b1111	0b100	0b1110	0b0010	0b000	CNTHP_TVAL	Counter-timer Hyp Physical Timer TimerValue register
0b1111	0b100	0b1110	0b0010	0b001	CNTHP_CTL	Counter-timer Hyp Physical Timer Control register
0b1111	0b110	0b1100	0b1100	0b100	ICC_MCTLR	Interrupt Controller Monitor Control Register
0b1111	0b110	0b1100	0b1100	0b101	ICC_MSRE	Interrupt Controller Monitor System Register Enable register
0b1111	0b110	0b1100	0b1100	0b111	ICC_MGRPEN1	Interrupt Controller Monitor Interrupt Group 1 Enable register

Accessed using MCRR/MRRC:

Register selectors			Name	Description
coproc	CRm	opc1		
0b1110	0b0001	0b0000	DBGDRAR	Debug ROM Address Register
0b1110	0b0010	0b0000	DBGDSAR	Debug Self Address Register
0b1111	0b000[n:3]	0b00[n:2:0]	AMEVCNTR0<n>	Activity Monitors Event Counter Registers 0
0b1111	0b0010	0b0000	TTBR0	Translation Table Base Register 0
0b1111	0b0010	0b0001	TTBR1	Translation Table Base Register 1
0b1111	0b0010	0b0100	HTTBR	Hyp Translation Table Base Register
0b1111	0b0010	0b0110	VTTBR	Virtualization Translation Table Base Register
0b1111	0b010[n:3]	0b00[n:2:0]	AMEVCNTR1<n>	Activity Monitors Event Counter Registers 1
0b1111	0b0111	0b0000	PAR	Physical Address Register
0b1111	0b1001	0b0000	PMCCNTR	Performance Monitors Cycle Count Register
0b1111	0b1100	0b0000	ICC_SGI1R	Interrupt Controller Software Generated Interrupt Group 1 Register
0b1111	0b1100	0b0001	ICC_ASGI1R	Interrupt Controller Alias Software Generated Interrupt Group 1 Register
0b1111	0b1100	0b0010	ICC_SGI0R	Interrupt Controller Software Generated Interrupt Group 0 Register
0b1111	0b1110	0b0000	CNTPCT	Counter-timer Physical Count register
0b1111	0b1110	0b0001	CNTVCT	Counter-timer Virtual Count register
0b1111	0b1110	0b0010	CNTP_CVAL	Counter-timer Physical Timer CompareValue register
0b1111	0b1110	0b0011	CNTV_CVAL	Counter-timer Virtual Timer CompareValue register
0b1111	0b1110	0b0100	CNTVOFF	Counter-timer Virtual Offset register
0b1111	0b1110	0b0110	CNTHP_CVAL	Counter-timer Hyp Physical CompareValue register

Accessed using MRS/MSR:

Register selectors			Name	Description
R	M	M1		
0b0	0b1	0b1110	ELR_hyp	Exception Link Register (Hyp mode)
0b1	0b0	0b1110	SPSR_fiq	Saved Program Status Register (FIQ mode)
0b1	0b1	0b0000	SPSR_irq	Saved Program Status Register (IRQ mode)
0b1	0b1	0b0010	SPSR_svc	Saved Program Status Register (Supervisor mode)
0b1	0b1	0b0100	SPSR_abt	Saved Program Status Register (Abort mode)
0b1	0b1	0b0110	SPSR_und	Saved Program Status Register (Undefined mode)
0b1	0b1	0b1100	SPSR_mon	Saved Program Status Register (Monitor mode)
0b1	0b1	0b1110	SPSR_hyp	Saved Program Status Register (Hyp mode)

Accessed using VMRS/VMSR:

Register selectors reg	Name	Description
0b0000	FPSID	Floating-Point System ID register
0b0001	FPSCR	Floating-Point Status and Control Register
0b0101	MVFR2	Media and VFP Feature Register 2
0b0110	MVFR1	Media and VFP Feature Register 1
0b0111	MVFR0	Media and VFP Feature Register 0
0b1000	FPEXC	Floating-Point Exception Control register

Registers and operations in AArch64

Accessed using AT:

Register selectors					Name	Description
op0	op1	CRn	CRm	op2		
0b01	0b000	0b0111	0b1000	0b000	AT S1E1R	Address Translate Stage 1 EL1 Read
0b01	0b000	0b0111	0b1000	0b001	AT S1E1W	Address Translate Stage 1 EL1 Write
0b01	0b000	0b0111	0b1000	0b010	AT S1E0R	Address Translate Stage 1 EL0 Read
0b01	0b000	0b0111	0b1000	0b011	AT S1E0W	Address Translate Stage 1 EL0 Write
0b01	0b000	0b0111	0b1001	0b000	AT S1E1RP	Address Translate Stage 1 EL1 Read PAN
0b01	0b000	0b0111	0b1001	0b001	AT S1E1WP	Address Translate Stage 1 EL1 Write PAN
0b01	0b100	0b0111	0b1000	0b000	AT S1E2R	Address Translate Stage 1 EL2 Read
0b01	0b100	0b0111	0b1000	0b001	AT S1E2W	Address Translate Stage 1 EL2 Write
0b01	0b100	0b0111	0b1000	0b100	AT S12E1R	Address Translate Stages 1 and 2 EL1 Read
0b01	0b100	0b0111	0b1000	0b101	AT S12E1W	Address Translate Stages 1 and 2 EL1 Write
0b01	0b100	0b0111	0b1000	0b110	AT S12E0R	Address Translate Stages 1 and 2 EL0 Read
0b01	0b100	0b0111	0b1000	0b111	AT S12E0W	Address Translate Stages 1 and 2 EL0 Write
0b01	0b110	0b0111	0b1000	0b000	AT S1E3R	Address Translate Stage 1 EL3 Read
0b01	0b110	0b0111	0b1000	0b001	AT S1E3W	Address Translate Stage 1 EL3 Write

Accessed using CFP:

Register selectors					Name	Description
op0	op1	CRn	CRm	op2		
0b01	0b011	0b0111	0b0011	0b100	CFP RCTX	Control Flow Prediction Restriction by Context

Accessed using CPP:

Register selectors					Name	Description
op0	op1	CRn	CRm	op2		
0b01	0b011	0b0111	0b0011	0b111	CPP RCTX	Cache Prefetch Prediction Restriction by Context

Accessed using DC:

Register selectors					Name	Description
op0	op1	CRn	CRm	op2		
0b01	0b000	0b0111	0b0110	0b001	DC IVAC	Data or unified Cache line Invalidate by VA to PoC
0b01	0b000	0b0111	0b0110	0b010	DC ISW	Data or unified Cache line Invalidate by Set/Way
0b01	0b000	0b0111	0b0110	0b011	DC IGVAC	Data, Allocation Tag or unified Cache line Invalidate of Allocation Tags by VA to PoC
0b01	0b000	0b0111	0b0110	0b100	DC IGSW	Data, Allocation Tag or unified Cache line Invalidate of Allocation Tags by Set/Way
0b01	0b000	0b0111	0b0110	0b101	DC IGDVAC	Data, Allocation Tag or unified Cache line Invalidate of Allocation Tags by VA to PoC
0b01	0b000	0b0111	0b0110	0b110	DC IGDSW	Data, Allocation Tag or unified Cache line Invalidate of Data and Allocation Tags by Set/Way
0b01	0b000	0b0111	0b1010	0b010	DC CSW	Data or unified Cache line Clean by Set/Way
0b01	0b000	0b0111	0b1010	0b100	DC CGSW	Data, Allocation Tag or unified Cache line Clean of Allocation Tags by Set/Way

Register selectors					Name	Description
op0	op1	CRn	CRm	op2		
0b01	0b000	0b0111	0b1010	0b110	DC CGDSW	Data, Allocation Tag or unified Cache line Clean of Data and Allocation Tags by Set/Way
0b01	0b000	0b0111	0b1110	0b010	DC CISW	Data or unified Cache line Clean and Invalidate by Set/Way
0b01	0b000	0b0111	0b1110	0b100	DC CIGSW	Data, Allocation Tag or unified Cache line Clean and Invalidate of Allocation Tags by Set/Way
0b01	0b000	0b0111	0b1110	0b110	DC CIGDSW	Data, Allocation Tag or unified Cache line Clean and Invalidate of Data and Allocation Tags by Set/Way
0b01	0b011	0b0111	0b0100	0b001	DC ZVA	Data Cache Zero by VA
0b01	0b011	0b0111	0b0100	0b011	DC GVA	Data Cache set Allocation Tag by VA
0b01	0b011	0b0111	0b0100	0b100	DC GZVA	Data Cache set Allocation Tags and Zero by VA
0b01	0b011	0b0111	0b1010	0b001	DC CVAC	Data or unified Cache line Clean by VA to PoC
0b01	0b011	0b0111	0b1010	0b011	DC CGVAC	Data, Allocation Tag or unified Cache line Clean of Allocation Tags by VA to PoC
0b01	0b011	0b0111	0b1010	0b101	DC CGDVAC	Data, Allocation Tag or unified Cache line Clean of Allocation Tags by VA to PoC
0b01	0b011	0b0111	0b1011	0b001	DC CVAU	Data or unified Cache line Clean by VA to PoU
0b01	0b011	0b0111	0b1100	0b001	DC CVAP	Data or unified Cache line Clean by VA to PoP
0b01	0b011	0b0111	0b1100	0b011	DC CGVAP	Data, Allocation Tag or unified Cache line Clean of Allocation Tags by VA to PoP
0b01	0b011	0b0111	0b1100	0b101	DC CGDVAP	Data, Allocation Tag or unified Cache line Clean of Data and Allocation Tags by VA to PoP
0b01	0b011	0b0111	0b1101	0b001	DC CVADP	Data or unified Cache line Clean by VA to PoDP
0b01	0b011	0b0111	0b1101	0b011	DC CGVADP	Data, Allocation Tag or unified Cache line Clean of Data and Allocation Tags by VA to PoDP
0b01	0b011	0b0111	0b1101	0b101	DC CGDVADP	Data, Allocation Tag or unified Cache line Clean of Allocation Tags by VA to PoDP
0b01	0b011	0b0111	0b1110	0b001	DC CIVAC	Data or unified Cache line Clean and Invalidate by VA to PoC
0b01	0b011	0b0111	0b1110	0b011	DC CIGVAC	Data, Allocation Tag or unified Cache line Clean and Invalidate of Allocation Tags by VA to PoC
0b01	0b011	0b0111	0b1110	0b101	DC CIGDVAC	Data, Allocation Tag or unified Cache line Clean and Invalidate of Data and Allocation Tags by VA to PoC

Accessed using DVP:

Register selectors					Name	Description
op0	op1	CRn	CRm	op2		
0b01	0b011	0b0111	0b0011	0b101	DVP RCTX	Data Value Prediction Restriction by Context

Accessed using IC:

Register selectors						Name	Description
op0	op1	CRn	CRm	op2	Rt		
0b01	0b000	0b0111	0b0001	0b000	0b11111	IC IALLUIS	Instruction Cache Invalidate All to PoU, Inner Shareable
0b01	0b000	0b0111	0b0101	0b000	0b11111	IC IALLU	Instruction Cache Invalidate All to PoU
0b01	0b011	0b0111	0b0101	0b001	–	IC IVAU	Instruction Cache line Invalidate by VA to PoU

Accessed using MRS/MSR:

Register selectors					Name	Description
op0	op1	CRn	CRm	op2		
0b10	0b000	0b0000	0b0000	0b010	OSDTRRX_EL1	OS Lock Data Transfer Register Receive
0b10	0b000	0b0000	0b0010	0b000	MDCCINT_EL1	Monitor DCC Interrupt Enable Register
0b10	0b000	0b0000	0b0010	0b010	MDSCR_EL1	Monitor Debug System Control Register
0b10	0b000	0b0000	0b0011	0b010	OSDTRTX_EL1	OS Lock Data Transfer Register Transmit
0b10	0b000	0b0000	0b0110	0b010	OSECCR_EL1	OS Lock Exception Catch Control Register
0b10	0b000	0b0000	0bnnnn	0b100	DBGBVR<n>_EL1	Debug Breakpoi Value Registers
0b10	0b000	0b0000	0bnnnn	0b101	DBGBCR<n>_EL1	Debug Breakpoi Control Register
0b10	0b000	0b0000	0bnnnn	0b110	DBGWVR<n>_EL1	Debug Watchpoi Value Registers
0b10	0b000	0b0000	0bnnnn	0b111	DBGWCR<n>_EL1	Debug Watchpoi Control Register
0b10	0b000	0b0001	0b0000	0b000	MDRAR_EL1	Monitor Debug ROM Address Register
0b10	0b000	0b0001	0b0000	0b100	OSLAR_EL1	OS Lock Access Register
0b10	0b000	0b0001	0b0001	0b100	OSLSR_EL1	OS Lock Status Register
0b10	0b000	0b0001	0b0011	0b100	OSDLR_EL1	OS Double Lock Register
0b10	0b000	0b0001	0b0100	0b100	DBGPRCR_EL1	Debug Power Control Register
0b10	0b000	0b0111	0b1000	0b110	DBGCLAIMSET_EL1	Debug Claim Tag Set register
0b10	0b000	0b0111	0b1001	0b110	DBGCLAIMCLR_EL1	Debug Claim Tag Clear register
0b10	0b000	0b0111	0b1110	0b110	DBGAUTHSTATUS_EL1	Debug Authentication Status register
0b10	0b011	0b0000	0b0001	0b000	MDCCSR_EL0	Monitor DCC Status Register
0b10	0b011	0b0000	0b0100	0b000	DBGDTR_EL0	Debug Data Transfer Register half-duplex

Register selectors					Name	Description
op0	op1	CRn	CRm	op2		
0b10	0b011	0b0000	0b0101	0b000	DBGDTRRX_EL0	Debug Data Transfer Register Receive
0b10	0b011	0b0000	0b0101	0b000	DBGDTRTX_EL0	Debug Data Transfer Register Transmit
0b10	0b100	0b0000	0b0111	0b000	DBGVCR32_EL2	Debug Vector Catch Register
0b11	0b000	0b0000	0b0000	0b000	MIDR_EL1	Main ID Register
0b11	0b000	0b0000	0b0000	0b101	MPIDR_EL1	Multiprocessor Affinity Register
0b11	0b000	0b0000	0b0000	0b110	REVIDR_EL1	Revision ID Register
0b11	0b000	0b0000	0b0001	0b000	ID_PFR0_EL1	AArch32 Processor Feature Register 0
0b11	0b000	0b0000	0b0001	0b001	ID_PFR1_EL1	AArch32 Processor Feature Register 1
0b11	0b000	0b0000	0b0001	0b010	ID_DFR0_EL1	AArch32 Debug Feature Register
0b11	0b000	0b0000	0b0001	0b011	ID_AFR0_EL1	AArch32 Auxiliary Feature Register 0
0b11	0b000	0b0000	0b0001	0b100	ID_MMFR0_EL1	AArch32 Memory Model Feature Register 0
0b11	0b000	0b0000	0b0001	0b101	ID_MMFR1_EL1	AArch32 Memory Model Feature Register 1
0b11	0b000	0b0000	0b0001	0b110	ID_MMFR2_EL1	AArch32 Memory Model Feature Register 2
0b11	0b000	0b0000	0b0001	0b111	ID_MMFR3_EL1	AArch32 Memory Model Feature Register 3
0b11	0b000	0b0000	0b0010	0b000	ID_ISAR0_EL1	AArch32 Instruction Set Attribute Register 0
0b11	0b000	0b0000	0b0010	0b001	ID_ISAR1_EL1	AArch32 Instruction Set Attribute Register 1
0b11	0b000	0b0000	0b0010	0b010	ID_ISAR2_EL1	AArch32 Instruction Set Attribute Register 2
0b11	0b000	0b0000	0b0010	0b011	ID_ISAR3_EL1	AArch32 Instruction Set Attribute Register 3
0b11	0b000	0b0000	0b0010	0b100	ID_ISAR4_EL1	AArch32 Instruction Set Attribute Register 4
0b11	0b000	0b0000	0b0010	0b101	ID_ISAR5_EL1	AArch32 Instruction Set Attribute Register 5
0b11	0b000	0b0000	0b0010	0b110	ID_MMFR4_EL1	AArch32 Memory Model Feature Register 4

Register selectors					Name	Description
op0	op1	CRn	CRm	op2		
0b11	0b000	0b0000	0b0010	0b111	ID_ISAR6_EL1	AArch32 Instruction Set Attribute Registe 6
0b11	0b000	0b0000	0b0011	0b000	MVFR0_EL1	AArch32 Media and VFP Feature Register 0
0b11	0b000	0b0000	0b0011	0b001	MVFR1_EL1	AArch32 Media and VFP Feature Register 1
0b11	0b000	0b0000	0b0011	0b010	MVFR2_EL1	AArch32 Media and VFP Feature Register 2
0b11	0b000	0b0000	0b0011	0b100	ID_PFR2_EL1	AArch32 Processor Featur Register 2
0b11	0b000	0b0000	0b0100	0b000	ID_AA64PFR0_EL1	AArch64 Processor Featur Register 0
0b11	0b000	0b0000	0b0100	0b001	ID_AA64PFR1_EL1	AArch64 Processor Featur Register 1
0b11	0b000	0b0000	0b0100	0b100	ID_AA64ZFR0_EL1	SVE Feature ID register 0
0b11	0b000	0b0000	0b0101	0b000	ID_AA64DFR0_EL1	AArch64 Debug Feature Register
0b11	0b000	0b0000	0b0101	0b001	ID_AA64DFR1_EL1	AArch64 Debug Feature Register
0b11	0b000	0b0000	0b0101	0b100	ID_AA64AFR0_EL1	AArch64 Auxiliary Feature Register 0
0b11	0b000	0b0000	0b0101	0b101	ID_AA64AFR1_EL1	AArch64 Auxiliary Feature Register 1
0b11	0b000	0b0000	0b0110	0b000	ID_AA64ISAR0_EL1	AArch64 Instruction Set Attribute Registe 0
0b11	0b000	0b0000	0b0110	0b001	ID_AA64ISAR1_EL1	AArch64 Instruction Set Attribute Registe 1
0b11	0b000	0b0000	0b0111	0b000	ID_AA64MMFR0_EL1	AArch64 Memor Model Feature Register 0
0b11	0b000	0b0000	0b0111	0b001	ID_AA64MMFR1_EL1	AArch64 Memor Model Feature Register 1
0b11	0b000	0b0000	0b0111	0b010	ID_AA64MMFR2_EL1	AArch64 Memor Model Feature Register 2
0b11	0b000	0b0001	0b0000	0b000	SCTLR_EL1	System Control Register (EL1)
0b11	0b000	0b0001	0b0000	0b001	ACTLR_EL1	Auxiliary Contro Register (EL1)
0b11	0b000	0b0001	0b0000	0b010	CPACR_EL1	Architectural Feature Access Control Register
0b11	0b000	0b0001	0b0000	0b101	RGSR_EL1	Random Allocation Tag Seed Register.

Register selectors					Name	Description
op0	op1	CRn	CRm	op2		
0b11	0b000	0b0001	0b0000	0b110	GCR_EL1	Tag Control Register.
0b11	0b000	0b0001	0b0010	0b000	ZCR_EL1	SVE Control Register for EL1
0b11	0b000	0b0001	0b0010	0b001	TRFCR_EL1	Trace Filter Control Register (EL1)
0b11	0b000	0b0010	0b0000	0b000	TTBR0_EL1	Translation Table Base Register 0 (EL1)
0b11	0b000	0b0010	0b0000	0b001	TTBR1_EL1	Translation Table Base Register 1 (EL1)
0b11	0b000	0b0010	0b0000	0b010	TCR_EL1	Translation Control Register (EL1)
0b11	0b000	0b0010	0b0001	0b000	APIAKeyLo_EL1	Pointer Authentication Key A for Instruction (bits[63:0])
0b11	0b000	0b0010	0b0001	0b001	APIAKeyHi_EL1	Pointer Authentication Key A for Instruction (bits[127:64])
0b11	0b000	0b0010	0b0001	0b010	APIBKeyLo_EL1	Pointer Authentication Key B for Instruction (bits[63:0])
0b11	0b000	0b0010	0b0001	0b011	APIBKeyHi_EL1	Pointer Authentication Key B for Instruction (bits[127:64])
0b11	0b000	0b0010	0b0010	0b000	APDAKeyLo_EL1	Pointer Authentication Key A for Data (bits[63:0])
0b11	0b000	0b0010	0b0010	0b001	APDAKeyHi_EL1	Pointer Authentication Key A for Data (bits[127:64])
0b11	0b000	0b0010	0b0010	0b010	APDBKeyLo_EL1	Pointer Authentication Key B for Data (bits[63:0])
0b11	0b000	0b0010	0b0010	0b011	APDBKeyHi_EL1	Pointer Authentication Key B for Data (bits[127:64])
0b11	0b000	0b0010	0b0011	0b000	APGAKeyLo_EL1	Pointer Authentication Key A for Code (bits[63:0])
0b11	0b000	0b0010	0b0011	0b001	APGAKeyHi_EL1	Pointer Authentication Key A for Code (bits[127:64])
0b11	0b000	0b0100	0b0000	0b000	SPSR_EL1	Saved Program Status Register (EL1)

Register selectors					Name	Description
op0	op1	CRn	CRm	op2		
0b11	0b000	0b0100	0b0000	0b001	ELR_EL1	Exception Link Register (EL1)
0b11	0b000	0b0100	0b0001	0b000	SP_EL0	Stack Pointer (EL0)
0b11	0b000	0b0100	0b0010	0b000	SPSel	Stack Pointer Select
0b11	0b000	0b0100	0b0010	0b010	CurrentEL	Current Exceptio Level
0b11	0b000	0b0100	0b0010	0b011	PAN	Privileged Acces Never
0b11	0b000	0b0100	0b0010	0b100	UAO	User Access Override
0b11	0b000	0b0100	0b0110	0b000	ICC_PMR_EL1	Interrupt Controller Interrupt Priority Mask Register
0b11	0b000	0b0101	0b0001	0b000	AFSR0_EL1	Auxiliary Fault Status Register 0 (EL1)
0b11	0b000	0b0101	0b0001	0b001	AFSR1_EL1	Auxiliary Fault Status Register 1 (EL1)
0b11	0b000	0b0101	0b0010	0b000	ESR_EL1	Exception Syndrome Register (EL1)
0b11	0b000	0b0101	0b0011	0b000	ERRIDR_EL1	Error Record ID Register
0b11	0b000	0b0101	0b0011	0b001	ERRSELR_EL1	Error Record Select Register
0b11	0b000	0b0101	0b0100	0b000	ERXFR_EL1	Selected Error Record Feature Register
0b11	0b000	0b0101	0b0100	0b001	ERXCTLR_EL1	Selected Error Record Control Register
0b11	0b000	0b0101	0b0100	0b010	ERXSTATUS_EL1	Selected Error Record Primary Status Register
0b11	0b000	0b0101	0b0100	0b011	ERXADDR_EL1	Selected Error Record Address Register
0b11	0b000	0b0101	0b0100	0b100	ERXPFGF_EL1	Selected Pseudo-fault Generation Feature Register
0b11	0b000	0b0101	0b0100	0b101	ERXPFGCTL_EL1	Selected Pseudo-fault Generation Control Register
0b11	0b000	0b0101	0b0100	0b110	ERXPFGCDN_EL1	Selected Pseudo-fault Generation Countdown Register
0b11	0b000	0b0101	0b0101	0b000	ERXMISC0_EL1	Selected Error Record Miscellaneous Register 0
0b11	0b000	0b0101	0b0101	0b001	ERXMISC1_EL1	Selected Error Record Miscellaneous Register 1
0b11	0b000	0b0101	0b0101	0b010	ERXMISC2_EL1	Selected Error Record

Register selectors					Name	Description
op0	op1	CRn	CRm	op2		
						Miscellaneous Register 2
0b11	0b000	0b0101	0b0101	0b011	ERXMISC3_EL1	Selected Error Record Miscellaneous Register 3
0b11	0b000	0b0110	0b0000	0b000	FAR_EL1	Fault Address Register (EL1)
0b11	0b000	0b0110	0b0101	0b000	TFSR_EL1	Tag Fail Status Register (EL1).
0b11	0b000	0b0110	0b0110	0b001	TFSRE0_EL1	Tag Fail Status Register (EL0).
0b11	0b000	0b0111	0b0100	0b000	PAR_EL1	Physical Address Register
0b11	0b000	0b1001	0b1001	0b000	PMSCR_EL1	Statistical Profiling Control Register (EL1)
0b11	0b000	0b1001	0b1001	0b010	PMSICR_EL1	Sampling Interva Counter Register
0b11	0b000	0b1001	0b1001	0b011	PMSIRR_EL1	Sampling Interva Reload Register
0b11	0b000	0b1001	0b1001	0b100	PMSFCR_EL1	Sampling Filter Control Register
0b11	0b000	0b1001	0b1001	0b101	PMSEVFR_EL1	Sampling Event Filter Register
0b11	0b000	0b1001	0b1001	0b110	PMSLATFR_EL1	Sampling Latenc Filter Register
0b11	0b000	0b1001	0b1001	0b111	PMSIDR_EL1	Sampling Profiling ID Register
0b11	0b000	0b1001	0b1010	0b000	PMBLIMITR_EL1	Profiling Buffer Limit Address Register
0b11	0b000	0b1001	0b1010	0b001	PMBPTR_EL1	Profiling Buffer Write Pointer Register
0b11	0b000	0b1001	0b1010	0b011	PMBSR_EL1	Profiling Buffer Status/syndrome Register
0b11	0b000	0b1001	0b1010	0b111	PMBIDR_EL1	Profiling Buffer ID Register
0b11	0b000	0b1001	0b1110	0b001	PMINTENSET_EL1	Performance Monitors Interru Enable Set regist
0b11	0b000	0b1001	0b1110	0b010	PMINTENCLR_EL1	Performance Monitors Interru Enable Clear register
0b11	0b000	0b1001	0b1110	0b110	PMMIR_EL1	Performance Monitors Machir Identification Register
0b11	0b000	0b1010	0b0010	0b000	MAIR_EL1	Memory Attribut Indirection Register (EL1)
0b11	0b000	0b1010	0b0011	0b000	AMAIR_EL1	Auxiliary Memo Attribute Indirection Register (EL1)
0b11	0b000	0b1010	0b0100	0b000	LORSA_EL1	LORegion Start Address (EL1)

Register selectors					Name	Description
op0	op1	CRn	CRm	op2		
0b11	0b000	0b1010	0b0100	0b001	LOREA_EL1	LORegion End Address (EL1)
0b11	0b000	0b1010	0b0100	0b010	LORN_EL1	LORegion Number (EL1)
0b11	0b000	0b1010	0b0100	0b011	LORC_EL1	LORegion Contr (EL1)
0b11	0b000	0b1010	0b0100	0b100	MPAMIDR_EL1	MPAM ID Register (EL1)
0b11	0b000	0b1010	0b0100	0b111	LORID_EL1	LORegionID (EL1)
0b11	0b000	0b1010	0b0101	0b000	MPAM1_EL1	MPAM1 Registe (EL1)
0b11	0b000	0b1010	0b0101	0b001	MPAM0_EL1	MPAM0 Registe (EL1)
0b11	0b000	0b1100	0b0000	0b000	VBAR_EL1	Vector Base Address Register (EL1)
0b11	0b000	0b1100	0b0000	0b001	RVBAR_EL1	Reset Vector Bas Address Register (if EL2 and EL3 not implemented
0b11	0b000	0b1100	0b0000	0b010	RMR_EL1	Reset Management Register (EL1)
0b11	0b000	0b1100	0b0001	0b000	ISR_EL1	Interrupt Status Register
0b11	0b000	0b1100	0b0001	0b001	DISR_EL1	Deferred Interrup Status Register
0b11	0b000	0b1100	0b1000	0b000	ICC_IAR0_EL1	Interrupt Controller Interrupt Acknowledge Register 0
0b11	0b000	0b1100	0b1000	0b001	ICC_EOIR0_EL1	Interrupt Controller End C Interrupt Registe 0
0b11	0b000	0b1100	0b1000	0b010	ICC_HPPIR0_EL1	Interrupt Controller Highe Priority Pending Interrupt Registe 0
0b11	0b000	0b1100	0b1000	0b011	ICC_BPR0_EL1	Interrupt Controller Binary Point Register 0
0b11	0b000	0b1100	0b1000	0b1[n:1:0]	ICC_AP0R<n>_EL1	Interrupt Controller Activ Priorities Group Registers
0b11	0b000	0b1100	0b1001	0b0[n:1:0]	ICC_AP1R<n>_EL1	Interrupt Controller Activ Priorities Group Registers
0b11	0b000	0b1100	0b1011	0b001	ICC_DIR_EL1	Interrupt Controller Deactivate Interrupt Registe
0b11	0b000	0b1100	0b1011	0b011	ICC_RPR_EL1	Interrupt Controller Running Priority Register
0b11	0b000	0b1100	0b1011	0b101	ICC_SGI1R_EL1	Interrupt Controller

Register selectors					Name	Description
op0	op1	CRn	CRm	op2		
						Software Generated Interrupt Group Register
0b11	0b000	0b1100	0b1011	0b110	ICC_ASGI1R_EL1	Interrupt Controller Alias Software Generated Interrupt Group Register
0b11	0b000	0b1100	0b1011	0b111	ICC_SGI0R_EL1	Interrupt Controller Software Generated Interrupt Group Register
0b11	0b000	0b1100	0b1100	0b000	ICC_IAR1_EL1	Interrupt Controller Interrupt Acknowledge Register 1
0b11	0b000	0b1100	0b1100	0b001	ICC_EOIR1_EL1	Interrupt Controller End Interrupt Register 1
0b11	0b000	0b1100	0b1100	0b010	ICC_HPPIR1_EL1	Interrupt Controller Highe Priority Pending Interrupt Register 1
0b11	0b000	0b1100	0b1100	0b011	ICC_BPR1_EL1	Interrupt Controller Binary Point Register 1
0b11	0b000	0b1100	0b1100	0b100	ICC_CTLR_EL1	Interrupt Controller Contr Register (EL1)
0b11	0b000	0b1100	0b1100	0b101	ICC_SRE_EL1	Interrupt Controller Syste Register Enable register (EL1)
0b11	0b000	0b1100	0b1100	0b110	ICC_IGRPEN0_EL1	Interrupt Controller Interrupt Group Enable register
0b11	0b000	0b1100	0b1100	0b111	ICC_IGRPEN1_EL1	Interrupt Controller Interrupt Group Enable register
0b11	0b000	0b1101	0b0000	0b001	CONTEXTIDR_EL1	Context ID Register (EL1)
0b11	0b000	0b1101	0b0000	0b100	TPIDR_EL1	EL1 Software Thread ID Register
0b11	0b000	0b1101	0b0000	0b111	SCXTNUM_EL1	EL1 Read/Write Software Contex Number
0b11	0b000	0b1110	0b0001	0b000	CNTKCTL_EL1	Counter-timer Kernel Control register
0b11	0b001	0b0000	0b0000	0b000	CCSIDR_EL1	Current Cache Size ID Register
0b11	0b001	0b0000	0b0000	0b001	CLIDR_EL1	Cache Level ID Register

Register selectors					Name	Description
op0	op1	CRn	CRm	op2		
0b11	0b001	0b0000	0b0000	0b010	CCSIDR2_EL1	Current Cache Size ID Register
0b11	0b001	0b0000	0b0000	0b100	GMID_EL1	Multiple tag transfer ID register
0b11	0b001	0b0000	0b0000	0b111	AIDR_EL1	Auxiliary ID Register
0b11	0b010	0b0000	0b0000	0b000	CSSELR_EL1	Cache Size Selection Registe
0b11	0b011	0b0000	0b0000	0b001	CTR_EL0	Cache Type Register
0b11	0b011	0b0000	0b0000	0b111	DCZID_EL0	Data Cache Zero ID register
0b11	0b011	0b0010	0b0100	0b000	RNDR	Random Number
0b11	0b011	0b0010	0b0100	0b001	RNDRRS	Reseeded Rando Number
0b11	0b011	0b0100	0b0010	0b000	NZCV	Condition Flags
0b11	0b011	0b0100	0b0010	0b001	DAIF	Interrupt Mask Bits
0b11	0b011	0b0100	0b0010	0b101	DIT	Data Independen Timing
0b11	0b011	0b0100	0b0010	0b110	SSBS	Speculative Stor Bypass Safe
0b11	0b011	0b0100	0b0010	0b111	TCO	Tag Check Override
0b11	0b011	0b0100	0b0100	0b000	FPCR	Floating-point Control Register
0b11	0b011	0b0100	0b0100	0b001	FPSR	Floating-point Status Register
0b11	0b011	0b0100	0b0101	0b000	DSPSR_EL0	Debug Saved Program Status Register
0b11	0b011	0b0100	0b0101	0b001	DLR_EL0	Debug Link Register
0b11	0b011	0b1001	0b1100	0b000	PMCR_EL0	Performance Monitors Contro Register
0b11	0b011	0b1001	0b1100	0b001	PMCNTENSET_EL0	Performance Monitors Count Enable Set regist
0b11	0b011	0b1001	0b1100	0b010	PMCNTENCLR_EL0	Performance Monitors Count Enable Clear register
0b11	0b011	0b1001	0b1100	0b011	PMOVSCLR_EL0	Performance Monitors Overflow Flag Status Clear Register
0b11	0b011	0b1001	0b1100	0b100	PMSWINC_EL0	Performance Monitors Softwa Increment regist
0b11	0b011	0b1001	0b1100	0b101	PMSELR_EL0	Performance Monitors Event Counter Selectio Register
0b11	0b011	0b1001	0b1100	0b110	PMCEID0_EL0	Performance Monitors Common Event Identification register 0

Register selectors					Name	Description
op0	op1	CRn	CRm	op2		
0b11	0b011	0b1001	0b1100	0b111	PMCEID1_EL0	Performance Monitors Common Event Identification register 1
0b11	0b011	0b1001	0b1101	0b000	PMCCNTR_EL0	Performance Monitors Cycle Count Register
0b11	0b011	0b1001	0b1101	0b001	PMXEVTYPER_EL0	Performance Monitors Selecte Event Type Register
0b11	0b011	0b1001	0b1101	0b010	PMXEVCNTR_EL0	Performance Monitors Selecte Event Count Register
0b11	0b011	0b1001	0b1110	0b000	PMUSERENR_EL0	Performance Monitors User Enable Register
0b11	0b011	0b1001	0b1110	0b011	PMOVSSET_EL0	Performance Monitors Overflow Flag Status Set registe
0b11	0b011	0b1101	0b0000	0b010	TPIDR_EL0	EL0 Read/Write Software Thread ID Register
0b11	0b011	0b1101	0b0000	0b011	TPIDRRO_EL0	EL0 Read-Only Software Thread ID Register
0b11	0b011	0b1101	0b0000	0b111	SCXTNUM_EL0	EL0 Read/Write Software Contex Number
0b11	0b011	0b1101	0b0010	0b000	AMCR_EL0	Activity Monitor Control Register
0b11	0b011	0b1101	0b0010	0b001	AMCFGR_EL0	Activity Monitor Configuration Register
0b11	0b011	0b1101	0b0010	0b010	AMCGCR_EL0	Activity Monitor Counter Group Configuration Register
0b11	0b011	0b1101	0b0010	0b011	AMUSERENR_EL0	Activity Monitor User Enable Register
0b11	0b011	0b1101	0b0010	0b100	AMCNTENCLR0_EL0	Activity Monitor Count Enable Clear Register 0
0b11	0b011	0b1101	0b0010	0b101	AMCNTENSET0_EL0	Activity Monitor Count Enable Se Register 0
0b11	0b011	0b1101	0b0011	0b000	AMCNTENCLR1_EL0	Activity Monitor Count Enable Clear Register 1
0b11	0b011	0b1101	0b0011	0b001	AMCNTENSET1_EL0	Activity Monitor Count Enable Se Register 1
0b11	0b011	0b1101	0b010[n:3]	0b[n:2:0]	AMEVCNTR0<n>_EL0	Activity Monitor Event Counter Registers 0
0b11	0b011	0b1101	0b011[n:3]	0b[n:2:0]	AMEVTYPER0<n>_EL0	Activity Monitor Event Type Registers 0

Register selectors					Name	Description
op0	op1	CRn	CRm	op2		
0b11	0b011	0b1101	0b110[n:3]	0b[n:2:0]	AMEVCNTR1<n>_EL0	Activity Monitor Event Counter Registers 1
0b11	0b011	0b1101	0b111[n:3]	0b[n:2:0]	AMEVTYPER1<n>_EL0	Activity Monitor Event Type Registers 1
0b11	0b011	0b1110	0b0000	0b000	CNTFRQ_EL0	Counter-timer Frequency regist
0b11	0b011	0b1110	0b0000	0b001	CNTPCT_EL0	Counter-timer Physical Count register
0b11	0b011	0b1110	0b0000	0b010	CNTVCT_EL0	Counter-timer Virtual Count register
0b11	0b011	0b1110	0b0010	0b000	CNTP_TVAL_EL0	Counter-timer Physical Timer TimerValue register
0b11	0b011	0b1110	0b0010	0b001	CNTP_CTL_EL0	Counter-timer Physical Timer Control register
0b11	0b011	0b1110	0b0010	0b010	CNTP_CVAL_EL0	Counter-timer Physical Timer CompareValue register
0b11	0b011	0b1110	0b0011	0b000	CNTV_TVAL_EL0	Counter-timer Virtual Timer TimerValue register
0b11	0b011	0b1110	0b0011	0b001	CNTV_CTL_EL0	Counter-timer Virtual Timer Control register
0b11	0b011	0b1110	0b0011	0b010	CNTV_CVAL_EL0	Counter-timer Virtual Timer CompareValue register
0b11	0b011	0b1110	0b10[n:4:3]	0b[n:2:0]	PMEVCNTR<n>_EL0	Performance Monitors Event Count Registers
0b11	0b011	0b1110	0b1111	0b111	PMCCFILTR_EL0	Performance Monitors Cycle Count Filter Register
0b11	0b011	0b1110	0b11[n:4:3]	0b[n:2:0]	PMEVTYPER<n>_EL0	Performance Monitors Event Type Registers
0b11	0b100	0b0000	0b0000	0b000	VPIDR_EL2	Virtualization Processor ID Register
0b11	0b100	0b0000	0b0000	0b101	VMPIDR_EL2	Virtualization Multiprocessor I Register
0b11	0b100	0b0001	0b0000	0b000	SCTLR_EL2	System Control Register (EL2)
0b11	0b100	0b0001	0b0000	0b001	ACTLR_EL2	Auxiliary Contro Register (EL2)
0b11	0b100	0b0001	0b0001	0b000	HCR_EL2	Hypervisor Configuration Register
0b11	0b100	0b0001	0b0001	0b001	MDCR_EL2	Monitor Debug Configuration Register (EL2)

Register selectors					Name	Description
op0	op1	CRn	CRm	op2		
0b11	0b100	0b0001	0b0001	0b010	CPTR_EL2	Architectural Feature Trap Register (EL2)
0b11	0b100	0b0001	0b0001	0b011	HSTR_EL2	Hypervisor System Trap Register
0b11	0b100	0b0001	0b0001	0b111	HACR_EL2	Hypervisor Auxiliary Contro Register
0b11	0b100	0b0001	0b0010	0b000	ZCR_EL2	SVE Control Register for EL2
0b11	0b100	0b0001	0b0010	0b001	TRFCR_EL2	Trace Filter Control Register (EL2)
0b11	0b100	0b0001	0b0011	0b001	SDER32_EL2	AArch32 Secure Debug Enable Register
0b11	0b100	0b0010	0b0000	0b000	TTBR0_EL2	Translation Table Base Register 0 (EL2)
0b11	0b100	0b0010	0b0000	0b001	TTBR1_EL2	Translation Table Base Register 1 (EL2)
0b11	0b100	0b0010	0b0000	0b010	TCR_EL2	Translation Control Register (EL2)
0b11	0b100	0b0010	0b0001	0b000	VTTBR_EL2	Virtualization Translation Table Base Register
0b11	0b100	0b0010	0b0001	0b010	VTCR_EL2	Virtualization Translation Control Register
0b11	0b100	0b0010	0b0010	0b000	VNCR_EL2	Virtual Nested Control Register
0b11	0b100	0b0010	0b0110	0b000	VSTTBR_EL2	Virtualization Secure Translatio Table Base Register
0b11	0b100	0b0010	0b0110	0b010	VSTCR_EL2	Virtualization Secure Translatio Control Register
0b11	0b100	0b0011	0b0000	0b000	DACR32_EL2	Domain Access Control Register
0b11	0b100	0b0100	0b0000	0b000	SPSR_EL2	Saved Program Status Register (EL2)
0b11	0b100	0b0100	0b0000	0b001	ELR_EL2	Exception Link Register (EL2)
0b11	0b100	0b0100	0b0001	0b000	SP_EL1	Stack Pointer (EL1)
0b11	0b100	0b0100	0b0011	0b000	SPSR_irq	Saved Program Status Register (IRQ mode)
0b11	0b100	0b0100	0b0011	0b001	SPSR_abt	Saved Program Status Register (Abort mode)
0b11	0b100	0b0100	0b0011	0b010	SPSR_und	Saved Program Status Register (Undefined mod
0b11	0b100	0b0100	0b0011	0b011	SPSR_fiq	Saved Program Status Register (FIQ mode)

Register selectors					Name	Description
op0	op1	CRn	CRm	op2		
0b11	0b100	0b0101	0b0000	0b001	IFSR32_EL2	Instruction Fault Status Register (EL2)
0b11	0b100	0b0101	0b0001	0b000	AFSR0_EL2	Auxiliary Fault Status Register 0 (EL2)
0b11	0b100	0b0101	0b0001	0b001	AFSR1_EL2	Auxiliary Fault Status Register 1 (EL2)
0b11	0b100	0b0101	0b0010	0b000	ESR_EL2	Exception Syndrome Register (EL2)
0b11	0b100	0b0101	0b0010	0b011	VSESR_EL2	Virtual SErrorDeferred ExceptionInterru SyndromeStatus Register
0b11	0b100	0b0101	0b0011	0b000	FPEXC32_EL2	Floating-Point Exception Contr register
0b11	0b100	0b0110	0b0000	0b000	FAR_EL2	Fault Address Register (EL2)
0b11	0b100	0b0110	0b0000	0b100	HPFAR_EL2	Hypervisor IPA Fault Address Register
0b11	0b100	0b0110	0b0101	0b000	TFSR_EL2	Tag Fail Status Register (EL2).
0b11	0b100	0b1001	0b1001	0b000	PMSCR_EL2	Statistical Profiling Control Register (EL2)
0b11	0b100	0b1010	0b0010	0b000	MAIR_EL2	Memory Attribut Indirection Register (EL2)
0b11	0b100	0b1010	0b0011	0b000	AMAIR_EL2	Auxiliary Memo Attribute Indirection Register (EL2)
0b11	0b100	0b1010	0b0100	0b000	MPAMHCR_EL2	MPAM Hypervisor Control Register (EL2)
0b11	0b100	0b1010	0b0100	0b001	MPAMVPMV_EL2	MPAM Virtual Partition Mappin Valid Register
0b11	0b100	0b1010	0b0101	0b000	MPAM2_EL2	MPAM2 Registe (EL2)
0b11	0b100	0b1010	0b0110	0b000	MPAMVPM0_EL2	MPAM Virtual PARTID Mappin Register 0
0b11	0b100	0b1010	0b0110	0b001	MPAMVPM1_EL2	MPAM Virtual PARTID Mappin Register 1
0b11	0b100	0b1010	0b0110	0b010	MPAMVPM2_EL2	MPAM Virtual PARTID Mappin Register 2
0b11	0b100	0b1010	0b0110	0b011	MPAMVPM3_EL2	MPAM Virtual PARTID Mappin Register 3
0b11	0b100	0b1010	0b0110	0b100	MPAMVPM4_EL2	MPAM Virtual PARTID Mappin Register 4

Register selectors					Name	Description
op0	op1	CRn	CRm	op2		
0b11	0b100	0b1010	0b0110	0b101	MPAMVPM5_EL2	MPAM Virtual PARTID Mappin Register 5
0b11	0b100	0b1010	0b0110	0b110	MPAMVPM6_EL2	MPAM Virtual PARTID Mappin Register 6
0b11	0b100	0b1010	0b0110	0b111	MPAMVPM7_EL2	MPAM Virtual PARTID Mappin Register 7
0b11	0b100	0b1100	0b0000	0b000	VBAR_EL2	Vector Base Address Register (EL2)
0b11	0b100	0b1100	0b0000	0b001	RVBAR_EL2	Reset Vector Bas Address Register (if EL3 not implemented)
0b11	0b100	0b1100	0b0000	0b010	RMR_EL2	Reset Management Register (EL2)
0b11	0b100	0b1100	0b0001	0b001	VDISR_EL2	Virtual Deferred Interrupt Status Register
0b11	0b100	0b1100	0b1000	0b0[n:1:0]	ICH_AP0R<n>_EL2	Interrupt Controller Hyp Active Priorities Group 0 Register
0b11	0b100	0b1100	0b1001	0b0[n:1:0]	ICH_AP1R<n>_EL2	Interrupt Controller Hyp Active Priorities Group 1 Register
0b11	0b100	0b1100	0b1001	0b101	ICC_SRE_EL2	Interrupt Controller Syster Register Enable register (EL2)
0b11	0b100	0b1100	0b1011	0b000	ICH_HCR_EL2	Interrupt Controller Hyp Control Register
0b11	0b100	0b1100	0b1011	0b001	ICH_VTR_EL2	Interrupt Controller VGIC Type Register
0b11	0b100	0b1100	0b1011	0b010	ICH_MISR_EL2	Interrupt Controller Maintenance Interrupt State Register
0b11	0b100	0b1100	0b1011	0b011	ICH_EISR_EL2	Interrupt Controller End o Interrupt Status Register
0b11	0b100	0b1100	0b1011	0b101	ICH_ELRSR_EL2	Interrupt Controller Empty List Register Status Register
0b11	0b100	0b1100	0b1011	0b111	ICH_VMCR_EL2	Interrupt Controller Virtua Machine Control Register
0b11	0b100	0b1100	0b110[n:3]	0b[n:2:0]	ICH_LR<n>_EL2	Interrupt Controller List Registers
0b11	0b100	0b1101	0b0000	0b001	CONTEXTIDR_EL2	Context ID Register (EL2)

Register selectors					Name	Description
op0	op1	CRn	CRm	op2		
0b11	0b100	0b1101	0b0000	0b010	TPIDR_EL2	EL2 Software Thread ID Register
0b11	0b100	0b1101	0b0000	0b111	SCXTNUM_EL2	EL2 Read/Write Software Contex Number
0b11	0b100	0b1110	0b0000	0b011	CNTVOFF_EL2	Counter-timer Virtual Offset register
0b11	0b100	0b1110	0b0001	0b000	CNTHCTL_EL2	Counter-timer Hypervisor Control register
0b11	0b100	0b1110	0b0010	0b000	CNTHP_TVAL_EL2	Counter-timer Physical Timer TimerValue register (EL2)
0b11	0b100	0b1110	0b0010	0b001	CNTHP_CTL_EL2	Counter-timer Hypervisor Physical Timer Control register
0b11	0b100	0b1110	0b0010	0b010	CNTHP_CVAL_EL2	Counter-timer Physical Timer CompareValue register (EL2)
0b11	0b100	0b1110	0b0011	0b000	CNTHV_TVAL_EL2	Counter-timer Virtual Timer TimerValue Register (EL2)
0b11	0b100	0b1110	0b0011	0b001	CNTHV_CTL_EL2	Counter-timer Virtual Timer Control register (EL2)
0b11	0b100	0b1110	0b0011	0b010	CNTHV_CVAL_EL2	Counter-timer Virtual Timer CompareValue register (EL2)
0b11	0b100	0b1110	0b0100	0b000	CNTHVS_TVAL_EL2	Counter-timer Secure Virtual Timer TimerValu register (EL2)
0b11	0b100	0b1110	0b0100	0b001	CNTHVS_CTL_EL2	Counter-timer Secure Virtual Timer Control register (EL2)
0b11	0b100	0b1110	0b0100	0b010	CNTHVS_CVAL_EL2	Counter-timer Secure Virtual Timer CompareValue register (EL2)
0b11	0b100	0b1110	0b0101	0b000	CNTHPS_TVAL_EL2	Counter-timer Secure Physical Timer TimerValu register (EL2)
0b11	0b100	0b1110	0b0101	0b001	CNTHPS_CTL_EL2	Counter-timer Secure Physical Timer Control register (EL2)
0b11	0b100	0b1110	0b0101	0b010	CNTHPS_CVAL_EL2	Counter-timer Secure Physical Timer CompareValue register (EL2)
0b11	0b110	0b0001	0b0000	0b000	SCTLR_EL3	System Control Register (EL3)

Register selectors					Name	Description
op0	op1	CRn	CRm	op2		
0b11	0b110	0b0001	0b0000	0b001	ACTLR_EL3	Auxiliary Contro Register (EL3)
0b11	0b110	0b0001	0b0001	0b000	SCR_EL3	Secure Configuration Register
0b11	0b110	0b0001	0b0001	0b001	SDER32_EL3	AArch32 Secure Debug Enable Register
0b11	0b110	0b0001	0b0001	0b010	CPTR_EL3	Architectural Feature Trap Register (EL3)
0b11	0b110	0b0001	0b0010	0b000	ZCR_EL3	SVE Control Register for EL3
0b11	0b110	0b0001	0b0011	0b001	MDCR_EL3	Monitor Debug Configuration Register (EL3)
0b11	0b110	0b0010	0b0000	0b000	TTBR0_EL3	Translation Table Base Register 0 (EL3)
0b11	0b110	0b0010	0b0000	0b010	TCR_EL3	Translation Control Register (EL3)
0b11	0b110	0b0100	0b0000	0b000	SPSR_EL3	Saved Program Status Register (EL3)
0b11	0b110	0b0100	0b0000	0b001	ELR_EL3	Exception Link Register (EL3)
0b11	0b110	0b0100	0b0001	0b000	SP_EL2	Stack Pointer (EL2)
0b11	0b110	0b0101	0b0001	0b000	AFSR0_EL3	Auxiliary Fault Status Register 0 (EL3)
0b11	0b110	0b0101	0b0001	0b001	AFSR1_EL3	Auxiliary Fault Status Register 1 (EL3)
0b11	0b110	0b0101	0b0010	0b000	ESR_EL3	Exception Syndrome Register (EL3)
0b11	0b110	0b0110	0b0000	0b000	FAR_EL3	Fault Address Register (EL3)
0b11	0b110	0b0110	0b0110~~0b0101~~	0b000	TFSR_EL3	Tag Fail Status Register (EL3).
0b11	0b110	0b1010	0b0010	0b000	MAIR_EL3	Memory Attribut Indirection Register (EL3)
0b11	0b110	0b1010	0b0011	0b000	AMAIR_EL3	Auxiliary Memo Attribute Indirection Register (EL3)
0b11	0b110	0b1010	0b0101	0b000	MPAM3_EL3	MPAM3 Registe (EL3)
0b11	0b110	0b1100	0b0000	0b000	VBAR_EL3	Vector Base Address Register (EL3)
0b11	0b110	0b1100	0b0000	0b001	RVBAR_EL3	Reset Vector Bas Address Register (if EL3 implemented)
0b11	0b110	0b1100	0b0000	0b010	RMR_EL3	Reset Management Register (EL3)

Register selectors					Name	Description
op0	op1	CRn	CRm	op2		
0b11	0b110	0b1100	0b1100	0b100	ICC_CTLR_EL3	Interrupt Controller Contr Register (EL3)
0b11	0b110	0b1100	0b1100	0b101	ICC_SRE_EL3	Interrupt Controller Syste Register Enable register (EL3)
0b11	0b110	0b1100	0b1100	0b111	ICC_IGRPEN1_EL3	Interrupt Controller Interrupt Group Enable register (EL3)
0b11	0b110	0b1101	0b0000	0b010	TPIDR_EL3	EL3 Software Thread ID Register
0b11	0b110	0b1101	0b0000	0b111	SCXTNUM_EL3	EL3 Read/Write Software Contex Number
0b11	0b111	0b1110	0b0010	0b000	CNTPS_TVAL_EL1	Counter-timer Physical Secure Timer TimerValu register
0b11	0b111	0b1110	0b0010	0b001	CNTPS_CTL_EL1	Counter-timer Physical Secure Timer Control register
0b11	0b111	0b1110	0b0010	0b010	CNTPS_CVAL_EL1	Counter-timer Physical Secure Timer CompareValue register

Accessed using TLBI:

Register selectors						Name	Description
op0	op1	CRn	CRm	op2	Rt		
0b01	0b000	0b1000	0b0001	0b000	0b11111	TLBI VMALLE1OS	TLB Invalidate by VMID, All at stage 1, EL1, Outer Shareable
0b01	0b000	0b1000	0b0001	0b001	-	TLBI VAE1OS	TLB Invalidate by VA, EL1, Outer Shareable
0b01	0b000	0b1000	0b0001	0b010	-	TLBI ASIDE1OS	TLB Invalidate by ASID, EL1, Outer Shareable
0b01	0b000	0b1000	0b0001	0b011	-	TLBI VAAE1OS	TLB Invalidate by VA, All ASID, EL1, Outer Shareable
0b01	0b000	0b1000	0b0001	0b101	-	TLBI VALE1OS	TLB Invalidate by VA, Last level, EL1, Outer Shareable
0b01	0b000	0b1000	0b0001	0b111	-	TLBI VAALE1OS	TLB Invalidate by VA, All ASID, Last Level, EL1, Outer Shareable
0b01	0b000	0b1000	0b0010	0b001	-	TLBI RVAE1IS	TLB Range Invalidate by VA, EL1, Inner Shareable

Register selectors						Name	Description
op0	op1	CRn	CRm	op2	Rt		
0b01	0b000	0b1000	0b0010	0b011	-	TLBI RVAAE1IS	TLB Range Invalidate by VA, All ASID, EL1, Inner Shareable
0b01	0b000	0b1000	0b0010	0b101	-	TLBI RVALE1IS	TLB Range Invalidate by VA, Last level, EL1, Inner Shareable
0b01	0b000	0b1000	0b0010	0b111	-	TLBI RVAALE1IS	TLB Range Invalidate by VA, All ASID, Last Level, EL1, Inner Shareable
0b01	0b000	0b1000	0b0011	0b000	0b11111	TLBI VMALLE1IS	TLB Invalidate by VMID, All at stage 1, EL1, Inner Shareable
0b01	0b000	0b1000	0b0011	0b001	-	TLBI VAE1IS	TLB Invalidate by VA, EL1, Inner Shareable
0b01	0b000	0b1000	0b0011	0b010	-	TLBI ASIDE1IS	TLB Invalidate by ASID, EL1, Inner Shareable
0b01	0b000	0b1000	0b0011	0b011	-	TLBI VAAE1IS	TLB Invalidate by VA, All ASID, EL1, Inner Shareable
0b01	0b000	0b1000	0b0011	0b101	-	TLBI VALE1IS	TLB Invalidate by VA, Last level, EL1, Inner Shareable
0b01	0b000	0b1000	0b0011	0b111	-	TLBI VAALE1IS	TLB Invalidate by VA, All ASID, Last Level, EL1, Inner Shareable
0b01	0b000	0b1000	0b0101	0b001	-	TLBI RVAE1OS	TLB Range Invalidate by VA, EL1, Outer Shareable
0b01	0b000	0b1000	0b0101	0b011	-	TLBI RVAAE1OS	TLB Range Invalidate by VA, All ASID, EL1, Outer Shareable
0b01	0b000	0b1000	0b0101	0b101	-	TLBI RVALE1OS	TLB Range Invalidate by VA, Last level, EL1, Outer Shareable
0b01	0b000	0b1000	0b0101	0b111	-	TLBI RVAALE1OS	TLB Range Invalidate by VA, All ASID, Last Level, EL1, Outer Shareable
0b01	0b000	0b1000	0b0110	0b001	-	TLBI RVAE1	TLB Range Invalidate by VA, EL1
0b01	0b000	0b1000	0b0110	0b011	-	TLBI RVAAE1	TLB Range Invalidate by VA, All ASID, EL1
0b01	0b000	0b1000	0b0110	0b101	-	TLBI RVALE1	TLB Range Invalidate by VA, Last level, EL1
0b01	0b000	0b1000	0b0110	0b111	-	TLBI RVAALE1	TLB Range Invalidate by VA,

Register selectors						Name	Description
op0	op1	CRn	CRm	op2	Rt		
							All ASID, Last level, EL1
0b01	0b000	0b1000	0b0111	0b000	0b11111	TLBI VMALLE1	TLB Invalidate by VMID, All at stage 1, EL1
0b01	0b000	0b1000	0b0111	0b001	-	TLBI VAE1	TLB Invalidate by VA, EL1
0b01	0b000	0b1000	0b0111	0b010	-	TLBI ASIDE1	TLB Invalidate by ASID, EL1
0b01	0b000	0b1000	0b0111	0b011	-	TLBI VAAE1	TLB Invalidate by VA, All ASID, EL1
0b01	0b000	0b1000	0b0111	0b101	-	TLBI VALE1	TLB Invalidate by VA, Last level, EL1
0b01	0b000	0b1000	0b0111	0b111	-	TLBI VAALE1	TLB Invalidate by VA, All ASID, Last level, EL1
0b01	0b100	0b1000	0b0000	0b001	-	TLBI IPAS2E1IS	TLB Invalidate by Intermediate Physical Address, Stage 2, EL1, Inner Shareable
0b01	0b100	0b1000	0b0000	0b010	-	TLBI RIPAS2E1IS	TLB Range Invalidate by Intermediate Physical Address, Stage 2, EL1, Inner Shareable
0b01	0b100	0b1000	0b0000	0b101	-	TLBI IPAS2LE1IS	TLB Invalidate by Intermediate Physical Address, Stage 2, Last level, EL1, Inner Shareable
0b01	0b100	0b1000	0b0000	0b110	-	TLBI RIPAS2LE1IS	TLB Range Invalidate by Intermediate Physical Address, Stage 2, Last level, EL1, Inner Shareable
0b01	0b100	0b1000	0b0001	0b000	0b11111	TLBI ALLE2OS	TLB Invalidate All, EL2, Outer Shareable
0b01	0b100	0b1000	0b0001	0b001	-	TLBI VAE2OS	TLB Invalidate by VA, EL2, Outer Shareable
0b01	0b100	0b1000	0b0001	0b100	0b11111	TLBI ALLE1OS	TLB Invalidate All, EL1, Outer Shareable
0b01	0b100	0b1000	0b0001	0b101	-	TLBI VALE2OS	TLB Invalidate by VA, Last level, EL2, Outer Shareable
0b01	0b100	0b1000	0b0001	0b110	0b11111	TLBI VMALLS12E1OS	TLB Invalidate by VMID, All at Stage 1 and 2, EL1, Outer Shareable

Register selectors						Name	Description
op0	op1	CRn	CRm	op2	Rt		
0b01	0b100	0b1000	0b0010	0b001	-	TLBI RVAE2IS	TLB Range Invalidate by VA, EL2, Inner Shareable
0b01	0b100	0b1000	0b0010	0b101	-	TLBI RVALE2IS	TLB Range Invalidate by VA, Last level, EL2, Inner Shareable
0b01	0b100	0b1000	0b0011	0b000	0b11111	TLBI ALLE2IS	TLB Invalidate All, EL2, Inner Shareable
0b01	0b100	0b1000	0b0011	0b001	-	TLBI VAE2IS	TLB Invalidate by VA, EL2, Inner Shareable
0b01	0b100	0b1000	0b0011	0b100	0b11111	TLBI ALLE1IS	TLB Invalidate All, EL1, Inner Shareable
0b01	0b100	0b1000	0b0011	0b101	-	TLBI VALE2IS	TLB Invalidate by VA, Last level, EL2, Inner Shareable
0b01	0b100	0b1000	0b0011	0b110	0b11111	TLBI VMALLS12E1IS	TLB Invalidate by VMID, All at Stage 1 and 2, EL1, Inner Shareable
0b01	0b100	0b1000	0b0100	0b000	-	TLBI IPAS2E1OS	TLB Invalidate by Intermediate Physical Address, Stage 2, EL1, Outer Shareable
0b01	0b100	0b1000	0b0100	0b001	-	TLBI IPAS2E1	TLB Invalidate by Intermediate Physical Address, Stage 2, EL1
0b01	0b100	0b1000	0b0100	0b010	-	TLBI RIPAS2E1	TLB Range Invalidate by Intermediate Physical Address, Stage 2, EL1
0b01	0b100	0b1000	0b0100	0b011	-	TLBI RIPAS2E1OS	TLB Range Invalidate by Intermediate Physical Address, Stage 2, EL1, Outer Shareable
0b01	0b100	0b1000	0b0100	0b100	-	TLBI IPAS2LE1OS	TLB Invalidate by Intermediate Physical Address, Stage 2, Last level, EL1, Outer Shareable
0b01	0b100	0b1000	0b0100	0b101	-	TLBI IPAS2LE1	TLB Invalidate by Intermediate Physical Address, Stage 2, Last level, EL1
0b01	0b100	0b1000	0b0100	0b110	-	TLBI RIPAS2LE1	TLB Range Invalidate by Intermediate

Register selectors						Name	Description
op0	op1	CRn	CRm	op2	Rt		
							Physical Address, Stage 2, Last level, EL1
0b01	0b100	0b1000	0b0100	0b111	-	TLBI RIPAS2LE1OS	TLB Range Invalidate by Intermediate Physical Address, Stage 2, Last level, EL1, Outer Shareable
0b01	0b100	0b1000	0b0101	0b001	-	TLBI RVAE2OS	TLB Range Invalidate by VA, EL2, Outer Shareable
0b01	0b100	0b1000	0b0101	0b101	-	TLBI RVALE2OS	TLB Range Invalidate by VA, Last level, EL2, Outer Shareable
0b01	0b100	0b1000	0b0110	0b001	-	TLBI RVAE2	TLB Range Invalidate by VA, EL2
0b01	0b100	0b1000	0b0110	0b101	-	TLBI RVALE2	TLB Range Invalidate by VA, Last level, EL2
0b01	0b100	0b1000	0b0111	0b000	0b11111	TLBI ALLE2	TLB Invalidate All, EL2
0b01	0b100	0b1000	0b0111	0b001	-	TLBI VAE2	TLB Invalidate by VA, EL2
0b01	0b100	0b1000	0b0111	0b100	0b11111	TLBI ALLE1	TLB Invalidate All, EL1
0b01	0b100	0b1000	0b0111	0b101	-	TLBI VALE2	TLB Invalidate by VA, Last level, EL2
0b01	0b100	0b1000	0b0111	0b110	0b11111	TLBI VMALLS12E1	TLB Invalidate by VMID, All at Stage 1 and 2, EL1
0b01	0b110	0b1000	0b0001	0b000	0b11111	TLBI ALLE3OS	TLB Invalidate All, EL3, Outer Shareable
0b01	0b110	0b1000	0b0001	0b001	-	TLBI VAE3OS	TLB Invalidate by VA, EL3, Outer Shareable
0b01	0b110	0b1000	0b0001	0b101	-	TLBI VALE3OS	TLB Invalidate by VA, Last level, EL3, Outer Shareable
0b01	0b110	0b1000	0b0010	0b001	-	TLBI RVAE3IS	TLB Range Invalidate by VA, EL3, Inner Shareable
0b01	0b110	0b1000	0b0010	0b101	-	TLBI RVALE3IS	TLB Range Invalidate by VA, Last level, EL3, Inner Shareable
0b01	0b110	0b1000	0b0011	0b000	0b11111	TLBI ALLE3IS	TLB Invalidate All, EL3, Inner Shareable
0b01	0b110	0b1000	0b0011	0b001	-	TLBI VAE3IS	TLB Invalidate by VA, EL3, Inner Shareable
0b01	0b110	0b1000	0b0011	0b101	-	TLBI VALE3IS	TLB Invalidate by VA, Last

Register selectors						Name	Description
op0	op1	CRn	CRm	op2	Rt		
							level, EL3, Inner Shareable
0b01	0b110	0b1000	0b0101	0b001	-	TLBI RVAE3OS	TLB Range Invalidate by VA, EL3, Outer Shareable
0b01	0b110	0b1000	0b0101	0b101	-	TLBI RVALE3OS	TLB Range Invalidate by VA, Last level, EL3, Outer Shareable
0b01	0b110	0b1000	0b0110	0b001	-	TLBI RVAE3	TLB Range Invalidate by VA, EL3
0b01	0b110	0b1000	0b0110	0b101	-	TLBI RVALE3	TLB Range Invalidate by VA, Last level, EL3
0b01	0b110	0b1000	0b0111	0b000	0b11111	TLBI ALLE3	TLB Invalidate All, EL3
0b01	0b110	0b1000	0b0111	0b001	-	TLBI VAE3	TLB Invalidate by VA, EL3
0b01	0b110	0b1000	0b0111	0b101	-	TLBI VALE3	TLB Invalidate by VA, Last level, EL3

27~~13~~/03~~12~~/2019~~2018~~ 21~~16~~:59~~43~~

(old) htmldiff from- (new)

(old) htmldiff from- (new)

System Register index by functional group

Below are indexes for registers with the following main functional groups:

In the ID functional group:

Exec state	Name	Description
AArch32	CCSIDR	Current Cache Size ID Register
AArch32	CCSIDR2	Current Cache Size ID Register 2
AArch32	CLIDR	Cache Level ID Register
AArch32	CSSELR	Cache Size Selection Register
AArch32	CTR	Cache Type Register
AArch32	ID_AFR0	Auxiliary Feature Register 0
AArch32	ID_DFR0	Debug Feature Register 0
AArch32	ID_ISAR0	Instruction Set Attribute Register 0
AArch32	ID_ISAR1	Instruction Set Attribute Register 1
AArch32	ID_ISAR2	Instruction Set Attribute Register 2
AArch32	ID_ISAR3	Instruction Set Attribute Register 3
AArch32	ID_ISAR4	Instruction Set Attribute Register 4
AArch32	ID_ISAR5	Instruction Set Attribute Register 5
AArch32	ID_ISAR6	Instruction Set Attribute Register 6
AArch32	ID_MMFR0	Memory Model Feature Register 0
AArch32	ID_MMFR1	Memory Model Feature Register 1
AArch32	ID_MMFR2	Memory Model Feature Register 2
AArch32	ID_MMFR3	Memory Model Feature Register 3
AArch32	ID_MMFR4	Memory Model Feature Register 4
AArch32	ID_PFR0	Processor Feature Register 0
AArch32	ID_PFR1	Processor Feature Register 1
AArch32	ID_PFR2	Processor Feature Register 2
AArch32	MIDR	Main ID Register
AArch32	MPIDR	Multiprocessor Affinity Register

Exec state	Name	Description
AArch32	REVIDR	Revision ID Register
AArch32	TCMTR	TCM Type Register
AArch32	TLBTR	TLB Type Register
AArch64	CCSIDR2_EL1	Current Cache Size ID Register 2
AArch64	CCSIDR_EL1	Current Cache Size ID Register
AArch64	CLIDR_EL1	Cache Level ID Register
AArch64	CSSELR_EL1	Cache Size Selection Register
AArch64	CTR_EL0	Cache Type Register
AArch64	DCZID_EL0	Data Cache Zero ID register
AArch64	GMID_EL1	Multiple tag transfer ID register
AArch64	ID_AA64AFR0_EL1	AArch64 Auxiliary Feature Register 0
AArch64	ID_AA64AFR1_EL1	AArch64 Auxiliary Feature Register 1
AArch64	ID_AA64DFR0_EL1	AArch64 Debug Feature Register 0
AArch64	ID_AA64DFR1_EL1	AArch64 Debug Feature Register 1
AArch64	ID_AA64ISAR0_EL1	AArch64 Instruction Set Attribute Register 0
AArch64	ID_AA64ISAR1_EL1	AArch64 Instruction Set Attribute Register 1
AArch64	ID_AA64MMFR0_EL1	AArch64 Memory Model Feature Register 0
AArch64	ID_AA64MMFR1_EL1	AArch64 Memory Model Feature Register 1
AArch64	ID_AA64MMFR2_EL1	AArch64 Memory Model Feature Register 2
AArch64	ID_AA64PFR0_EL1	AArch64 Processor Feature Register 0
AArch64	ID_AA64PFR1_EL1	AArch64 Processor Feature Register 1
AArch64	ID_AFR0_EL1	AArch32 Auxiliary Feature Register 0
AArch64	ID_DFR0_EL1	AArch32 Debug Feature Register 0
AArch64	ID_ISAR0_EL1	AArch32 Instruction Set Attribute Register 0
AArch64	ID_ISAR1_EL1	AArch32 Instruction Set Attribute Register 1
AArch64	ID_ISAR2_EL1	AArch32 Instruction Set Attribute Register 2
AArch64	ID_ISAR3_EL1	AArch32 Instruction Set Attribute Register 3
AArch64	ID_ISAR4_EL1	AArch32 Instruction Set Attribute Register 4
AArch64	ID_ISAR5_EL1	AArch32 Instruction Set Attribute Register 5
AArch64	ID_ISAR6_EL1	AArch32 Instruction Set Attribute Register 6
AArch64	ID_MMFR0_EL1	AArch32 Memory Model Feature Register 0
AArch64	ID_MMFR1_EL1	AArch32 Memory Model Feature Register 1
AArch64	ID_MMFR2_EL1	AArch32 Memory Model Feature Register 2
AArch64	ID_MMFR3_EL1	AArch32 Memory Model Feature Register 3
AArch64	ID_MMFR4_EL1	AArch32 Memory Model Feature Register 4
AArch64	ID_PFR0_EL1	AArch32 Processor Feature Register 0
AArch64	ID_PFR1_EL1	AArch32 Processor Feature Register 1
AArch64	ID_PFR2_EL1	AArch32 Processor Feature Register 2
AArch64	MIDR_EL1	Main ID Register
AArch64	MPIDR_EL1	Multiprocessor Affinity Register
AArch64	REVIDR_EL1	Revision ID Register
External	EDAA32PFR	External Debug AArch32 Processor Feature Register
External	EDDFR	External Debug Feature Register
External	EDPFR	External Debug Processor Feature Register
External	MIDR_EL1	Main ID Register

In the Memory functional group:

Exec state	Name	Description
AArch32	AMAIR0	Auxiliary Memory Attribute Indirection Register 0
AArch32	AMAIR1	Auxiliary Memory Attribute Indirection Register 1
AArch32	CONTEXTIDR	Context ID Register
AArch32	DACR	Domain Access Control Register
AArch32	HAMAIR0	Hyp Auxiliary Memory Attribute Indirection Register 0
AArch32	HAMAIR1	Hyp Auxiliary Memory Attribute Indirection Register 1
AArch32	HMAIR0	Hyp Memory Attribute Indirection Register 0
AArch32	HMAIR1	Hyp Memory Attribute Indirection Register 1
AArch32	HTCR	Hyp Translation Control Register
AArch32	HTTBR	Hyp Translation Table Base Register
AArch32	MAIR0	Memory Attribute Indirection Register 0
AArch32	MAIR1	Memory Attribute Indirection Register 1
AArch32	NMRR	Normal Memory Remap Register
AArch32	PRRR	Primary Region Remap Register

Exec state	Name	Description
AArch32	TTBCR	Translation Table Base Control Register
AArch32	TTBCR2	Translation Table Base Control Register 2
AArch32	TTBR0	Translation Table Base Register 0
AArch32	TTBR1	Translation Table Base Register 1
AArch32	VTCR	Virtualization Translation Control Register
AArch32	VTTBR	Virtualization Translation Table Base Register
AArch64	AMAIR_EL1	Auxiliary Memory Attribute Indirection Register (EL1)
AArch64	AMAIR_EL2	Auxiliary Memory Attribute Indirection Register (EL2)
AArch64	AMAIR_EL3	Auxiliary Memory Attribute Indirection Register (EL3)
AArch64	CONTEXTIDR_EL1	Context ID Register (EL1)
AArch64	CONTEXTIDR_EL2	Context ID Register (EL2)
AArch64	DACR32_EL2	Domain Access Control Register
AArch64	LORC_EL1	LORegion Control (EL1)
AArch64	LOREA_EL1	LORegion End Address (EL1)
AArch64	LORID_EL1	LORegionID (EL1)
AArch64	LORN_EL1	LORegion Number (EL1)
AArch64	LORSA_EL1	LORegion Start Address (EL1)
AArch64	MAIR_EL1	Memory Attribute Indirection Register (EL1)
AArch64	MAIR_EL2	Memory Attribute Indirection Register (EL2)
AArch64	MAIR_EL3	Memory Attribute Indirection Register (EL3)
AArch64	TCR_EL1	Translation Control Register (EL1)
AArch64	TCR_EL2	Translation Control Register (EL2)
AArch64	TCR_EL3	Translation Control Register (EL3)
AArch64	TTBR0_EL1	Translation Table Base Register 0 (EL1)
AArch64	TTBR0_EL2	Translation Table Base Register 0 (EL2)
AArch64	TTBR0_EL3	Translation Table Base Register 0 (EL3)
AArch64	TTBR1_EL1	Translation Table Base Register 1 (EL1)
AArch64	TTBR1_EL2	Translation Table Base Register 1 (EL2)
AArch64	VTCR_EL2	Virtualization Translation Control Register
AArch64	VTTBR_EL2	Virtualization Translation Table Base Register

In the Other functional group:

Exec state	Name	Description
AArch32	CPACR	Architectural Feature Access Control Register
AArch32	SCTLR	System Control Register
AArch64	CPACR_EL1	Architectural Feature Access Control Register
AArch64	SCTLR_EL1	System Control Register (EL1)
AArch64	SCTLR_EL3	System Control Register (EL3)
AArch64	ZCR_EL1	SVE Control Register for EL1
AArch64	ZCR_EL2	SVE Control Register for EL2
AArch64	ZCR_EL3	SVE Control Register for EL3

In the Exception functional group:

Exec state	Name	Description
AArch32	ADFSR	Auxiliary Data Fault Status Register
AArch32	AIFSR	Auxiliary Instruction Fault Status Register
AArch32	DFAR	Data Fault Address Register
AArch32	DFSR	Data Fault Status Register
AArch32	HADFSR	Hyp Auxiliary Data Fault Status Register
AArch32	HAIFSR	Hyp Auxiliary Instruction Fault Status Register
AArch32	HDFAR	Hyp Data Fault Address Register
AArch32	HIFAR	Hyp Instruction Fault Address Register
AArch32	HPFAR	Hyp IPA Fault Address Register
AArch32	HSR	Hyp Syndrome Register
AArch32	HVBAR	Hyp Vector Base Address Register
AArch32	IFAR	Instruction Fault Address Register
AArch32	IFSR	Instruction Fault Status Register
AArch32	ISR	Interrupt Status Register
AArch32	MVBAR	Monitor Vector Base Address Register
AArch32	VBAR	Vector Base Address Register

Exec state	Name	Description
AArch64	AFSR0_EL1	Auxiliary Fault Status Register 0 (EL1)
AArch64	AFSR0_EL2	Auxiliary Fault Status Register 0 (EL2)
AArch64	AFSR0_EL3	Auxiliary Fault Status Register 0 (EL3)
AArch64	AFSR1_EL1	Auxiliary Fault Status Register 1 (EL1)
AArch64	AFSR1_EL2	Auxiliary Fault Status Register 1 (EL2)
AArch64	AFSR1_EL3	Auxiliary Fault Status Register 1 (EL3)
AArch64	ESR_EL1	Exception Syndrome Register (EL1)
AArch64	ESR_EL2	Exception Syndrome Register (EL2)
AArch64	ESR_EL3	Exception Syndrome Register (EL3)
AArch64	FAR_EL1	Fault Address Register (EL1)
AArch64	FAR_EL2	Fault Address Register (EL2)
AArch64	FAR_EL3	Fault Address Register (EL3)
AArch64	HPFAR_EL2	Hypervisor IPA Fault Address Register
AArch64	IFSR32_EL2	Instruction Fault Status Register (EL2)
AArch64	ISR_EL1	Interrupt Status Register
AArch64	VBAR_EL1	Vector Base Address Register (EL1)
AArch64	VBAR_EL2	Vector Base Address Register (EL2)
AArch64	VBAR_EL3	Vector Base Address Register (EL3)

In the Special functional group:

Exec state	Name	Description
AArch32	DLR	Debug Link Register
AArch32	DSPSR	Debug Saved Program Status Register
AArch32	ELR_hyp	Exception Link Register (Hyp mode)
AArch32	SPSR	Saved Program Status Register
AArch32	SPSR_abt	Saved Program Status Register (Abort mode)
AArch32	SPSR_fiq	Saved Program Status Register (FIQ mode)
AArch32	SPSR_hyp	Saved Program Status Register (Hyp mode)
AArch32	SPSR_irq	Saved Program Status Register (IRQ mode)
AArch32	SPSR_mon	Saved Program Status Register (Monitor mode)
AArch32	SPSR_svc	Saved Program Status Register (Supervisor mode)
AArch32	SPSR_und	Saved Program Status Register (Undefined mode)
AArch64	ELR_EL1	Exception Link Register (EL1)
AArch64	ELR_EL2	Exception Link Register (EL2)
AArch64	ELR_EL3	Exception Link Register (EL3)
AArch64	SPSR_EL1	Saved Program Status Register (EL1)
AArch64	SPSR_EL2	Saved Program Status Register (EL2)
AArch64	SPSR_EL3	Saved Program Status Register (EL3)
AArch64	SPSR_abt	Saved Program Status Register (Abort mode)
AArch64	SPSR_fiq	Saved Program Status Register (FIQ mode)
AArch64	SPSR_irq	Saved Program Status Register (IRQ mode)
AArch64	SPSR_und	Saved Program Status Register (Undefined mode)
AArch64	SP_EL0	Stack Pointer (EL0)
AArch64	SP_EL1	Stack Pointer (EL1)
AArch64	SP_EL2	Stack Pointer (EL2)
AArch64	SP_EL3	Stack Pointer (EL3)

In the PSTATE functional group:

Exec state	Name	Description
AArch32	APSR	Application Program Status Register
AArch32	CPSR	Current Program Status Register
AArch64	CurrentEL	Current Exception Level
AArch64	DAIF	Interrupt Mask Bits
AArch64	DIT	Data Independent Timing
AArch64	NZCV	Condition Flags
AArch64	PAN	Privileged Access Never
AArch64	SPSel	Stack Pointer Select
AArch64	SSBS	Speculative Store Bypass Safe
AArch64	TCO	Tag Check Override
AArch64	UAO	User Access Override

In the Cache functional group:

Exec state	Name	Description
AArch32	BPIALL	Branch Predictor Invalidate All
AArch32	BPIALLIS	Branch Predictor Invalidate All, Inner Shareable
AArch32	BPIMVA	Branch Predictor Invalidate by VA
AArch32	DCCIMVAC	Data Cache line Clean and Invalidate by VA to PoC
AArch32	DCCISW	Data Cache line Clean and Invalidate by Set/Way
AArch32	DCCMVAC	Data Cache line Clean by VA to PoC
AArch32	DCCMVAU	Data Cache line Clean by VA to PoU
AArch32	DCCSW	Data Cache line Clean by Set/Way
AArch32	DCIMVAC	Data Cache line Invalidate by VA to PoC
AArch32	DCISW	Data Cache line Invalidate by Set/Way
AArch32	ICIALLU	Instruction Cache Invalidate All to PoU
AArch32	ICIALLUIS	Instruction Cache Invalidate All to PoU, Inner Shareable
AArch32	ICIMVAU	Instruction Cache line Invalidate by VA to PoU
AArch64	DC CGDSW	Data, Allocation Tag or unified Cache line Clean of Data and Allocation Tags by Set/Way
AArch64	DC CGDVAC	Data, Allocation Tag or unified Cache line Clean of Allocation Tags by VA to PoC
AArch64	DC CGDVADP	Data, Allocation Tag or unified Cache line Clean of Allocation Tags by VA to PoDP
AArch64	DC CGDVAP	Data, Allocation Tag or unified Cache line Clean of Data and Allocation Tags by VA to PoP
AArch64	DC CGSW	Data, Allocation Tag or unified Cache line Clean of Allocation Tags by Set/Way
AArch64	DC CGVAC	Data, Allocation Tag or unified Cache line Clean of Allocation Tags by VA to PoC
AArch64	DC CGVADP	Data, Allocation Tag or unified Cache line Clean of Data and Allocation Tags by VA to PoDP
AArch64	DC CGVAP	Data, Allocation Tag or unified Cache line Clean of Allocation Tags by VA to PoP
AArch64	DC CIGDSW	Data, Allocation Tag or unified Cache line Clean and Invalidate of Data and Allocation Tags by Set/Way
AArch64	DC CIGDVAC	Data, Allocation Tag or unified Cache line Clean and Invalidate of Data and Allocation Tags by VA to PoC
AArch64	DC CIGSW	Data, Allocation Tag or unified Cache line Clean and Invalidate of Allocation Tags by Set/Way
AArch64	DC CIGVAC	Data, Allocation Tag or unified Cache line Clean and Invalidate of Allocation Tags by VA to PoC
AArch64	DC CISW	Data or unified Cache line Clean and Invalidate by Set/Way
AArch64	DC CIVAC	Data or unified Cache line Clean and Invalidate by VA to PoC
AArch64	DC CSW	Data or unified Cache line Clean by Set/Way
AArch64	DC CVAC	Data or unified Cache line Clean by VA to PoC
AArch64	DC CVADP	Data or unified Cache line Clean by VA to PoDP
AArch64	DC CVAP	Data or unified Cache line Clean by VA to PoP
AArch64	DC CVAU	Data or unified Cache line Clean by VA to PoU
AArch64	DC GVA	Data Cache set Allocation Tag by VA
AArch64	DC GZVA	Data Cache set Allocation Tags and Zero by VA
AArch64	DC IGDSW	Data, Allocation Tag or unified Cache line Invalidate of Data and Allocation Tags by Set/Way
AArch64	DC IGDVAC	Data, Allocation Tag or unified Cache line Invalidate of Allocation Tags by VA to PoC
AArch64	DC IGSW	Data, Allocation Tag or unified Cache line Invalidate of Allocation Tags by Set/Way
AArch64	DC IGVAC	Data, Allocation Tag or unified Cache line Invalidate of Allocation Tags by VA to PoC
AArch64	DC ISW	Data or unified Cache line Invalidate by Set/Way
AArch64	DC IVAC	Data or unified Cache line Invalidate by VA to PoC
AArch64	DC ZVA	Data Cache Zero by VA
AArch64	IC IALLU	Instruction Cache Invalidate All to PoU
AArch64	IC IALLUIS	Instruction Cache Invalidate All to PoU, Inner Shareable
AArch64	IC IVAU	Instruction Cache line Invalidate by VA to PoU

In the Address functional group:

Exec state	Name	Description
AArch32	ATS12NSOPR	Address Translate Stages 1 and 2 Non-secure Only PL1 Read
AArch32	ATS12NSOPW	Address Translate Stages 1 and 2 Non-secure Only PL1 Write
AArch32	ATS12NSOUR	Address Translate Stages 1 and 2 Non-secure Only Unprivileged Read
AArch32	ATS12NSOUW	Address Translate Stages 1 and 2 Non-secure Only Unprivileged Write
AArch32	ATS1CPR	Address Translate Stage 1 Current state PL1 Read
AArch32	ATS1CPRP	Address Translate Stage 1 Current state PL1 Read PAN
AArch32	ATS1CPW	Address Translate Stage 1 Current state PL1 Write
AArch32	ATS1CPWP	Address Translate Stage 1 Current state PL1 Write PAN
AArch32	ATS1CUR	Address Translate Stage 1 Current state Unprivileged Read
AArch32	ATS1CUW	Address Translate Stage 1 Current state Unprivileged Write
AArch32	ATS1HR	Address Translate Stage 1 Hyp mode Read
AArch32	ATS1HW	Address Translate Stage 1 Hyp mode Write
AArch32	PAR	Physical Address Register

Exec state	Name	Description
AArch64	AT S12E0R	Address Translate Stages 1 and 2 EL0 Read
AArch64	AT S12E0W	Address Translate Stages 1 and 2 EL0 Write
AArch64	AT S12E1R	Address Translate Stages 1 and 2 EL1 Read
AArch64	AT S12E1W	Address Translate Stages 1 and 2 EL1 Write
AArch64	AT S1E0R	Address Translate Stage 1 EL0 Read
AArch64	AT S1E0W	Address Translate Stage 1 EL0 Write
AArch64	AT S1E1R	Address Translate Stage 1 EL1 Read
AArch64	AT S1E1RP	Address Translate Stage 1 EL1 Read PAN
AArch64	AT S1E1W	Address Translate Stage 1 EL1 Write
AArch64	AT S1E1WP	Address Translate Stage 1 EL1 Write PAN
AArch64	AT S1E2R	Address Translate Stage 1 EL2 Read
AArch64	AT S1E2W	Address Translate Stage 1 EL2 Write
AArch64	AT S1E3R	Address Translate Stage 1 EL3 Read
AArch64	AT S1E3W	Address Translate Stage 1 EL3 Write
AArch64	PAR_EL1	Physical Address Register

In the TLB functional group:

Exec state	Name	Description
AArch32	CFPRCTX	Control Flow Prediction Restriction by Context
AArch32	CPPRCTX	Cache Prefetch Prediction Restriction by Context
AArch32	DTLBIALL	Data TLB Invalidate All
AArch32	DTLBIASID	Data TLB Invalidate by ASID match
AArch32	DTLBIMVA	Data TLB Invalidate by VA
AArch32	DVPRCTX	Data Value Prediction Restriction by Context
AArch32	ITLBIALL	Instruction TLB Invalidate All
AArch32	ITLBIASID	Instruction TLB Invalidate by ASID match
AArch32	ITLBIMVA	Instruction TLB Invalidate by VA
AArch32	TLBIALL	TLB Invalidate All
AArch32	TLBIALLH	TLB Invalidate All, Hyp mode
AArch32	TLBIALLHIS	TLB Invalidate All, Hyp mode, Inner Shareable
AArch32	TLBIALLIS	TLB Invalidate All, Inner Shareable
AArch32	TLBIALLNSNH	TLB Invalidate All, Non-Secure Non-Hyp
AArch32	TLBIALLNSNHIS	TLB Invalidate All, Non-Secure Non-Hyp, Inner Shareable
AArch32	TLBIASID	TLB Invalidate by ASID match
AArch32	TLBIASIDIS	TLB Invalidate by ASID match, Inner Shareable
AArch32	TLBIIPAS2	TLB Invalidate by Intermediate Physical Address, Stage 2
AArch32	TLBIIPAS2IS	TLB Invalidate by Intermediate Physical Address, Stage 2, Inner Shareable
AArch32	TLBIIPAS2L	TLB Invalidate by Intermediate Physical Address, Stage 2, Last level
AArch32	TLBIIPAS2LIS	TLB Invalidate by Intermediate Physical Address, Stage 2, Last level, Inner Shareable
AArch32	TLBIMVA	TLB Invalidate by VA
AArch32	TLBIMVAA	TLB Invalidate by VA, All ASID
AArch32	TLBIMVAAIS	TLB Invalidate by VA, All ASID, Inner Shareable
AArch32	TLBIMVAAL	TLB Invalidate by VA, All ASID, Last level
AArch32	TLBIMVAALIS	TLB Invalidate by VA, All ASID, Last level, Inner Shareable
AArch32	TLBIMVAH	TLB Invalidate by VA, Hyp mode
AArch32	TLBIMVAHIS	TLB Invalidate by VA, Hyp mode, Inner Shareable
AArch32	TLBIMVAIS	TLB Invalidate by VA, Inner Shareable
AArch32	TLBIMVAL	TLB Invalidate by VA, Last level
AArch32	TLBIMVALH	TLB Invalidate by VA, Last level, Hyp mode
AArch32	TLBIMVALHIS	TLB Invalidate by VA, Last level, Hyp mode, Inner Shareable
AArch32	TLBIMVALIS	TLB Invalidate by VA, Last level, Inner Shareable
AArch64	TLBI ALLE1	TLB Invalidate All, EL1
AArch64	TLBI ALLE1IS	TLB Invalidate All, EL1, Inner Shareable
AArch64	TLBI ALLE1OS	TLB Invalidate All, EL1, Outer Shareable
AArch64	TLBI ALLE2	TLB Invalidate All, EL2
AArch64	TLBI ALLE2IS	TLB Invalidate All, EL2, Inner Shareable
AArch64	TLBI ALLE2OS	TLB Invalidate All, EL2, Outer Shareable
AArch64	TLBI ALLE3	TLB Invalidate All, EL3
AArch64	TLBI ALLE3IS	TLB Invalidate All, EL3, Inner Shareable
AArch64	TLBI ALLE3OS	TLB Invalidate All, EL3, Outer Shareable
AArch64	TLBI ASIDE1	TLB Invalidate by ASID, EL1
AArch64	TLBI ASIDE1IS	TLB Invalidate by ASID, EL1, Inner Shareable

Exec state	Name	Description
AArch64	TLBI ASIDE1OS	TLB Invalidate by ASID, EL1, Outer Shareable
AArch64	TLBI IPAS2E1	TLB Invalidate by Intermediate Physical Address, Stage 2, EL1
AArch64	TLBI IPAS2E1IS	TLB Invalidate by Intermediate Physical Address, Stage 2, EL1, Inner Shareable
AArch64	TLBI IPAS2E1OS	TLB Invalidate by Intermediate Physical Address, Stage 2, EL1, Outer Shareable
AArch64	TLBI IPAS2LE1	TLB Invalidate by Intermediate Physical Address, Stage 2, Last level, EL1
AArch64	TLBI IPAS2LE1IS	TLB Invalidate by Intermediate Physical Address, Stage 2, Last level, EL1, Inner Shareable
AArch64	TLBI IPAS2LE1OS	TLB Invalidate by Intermediate Physical Address, Stage 2, Last level, EL1, Outer Shareable
AArch64	TLBI RIPAS2E1	TLB Range Invalidate by Intermediate Physical Address, Stage 2, EL1
AArch64	TLBI RIPAS2E1IS	TLB Range Invalidate by Intermediate Physical Address, Stage 2, EL1, Inner Shareable
AArch64	TLBI RIPAS2E1OS	TLB Range Invalidate by Intermediate Physical Address, Stage 2, EL1, Outer Shareable
AArch64	TLBI RIPAS2LE1	TLB Range Invalidate by Intermediate Physical Address, Stage 2, Last level, EL1
AArch64	TLBI RIPAS2LE1IS	TLB Range Invalidate by Intermediate Physical Address, Stage 2, Last level, EL1, Inner Shareable
AArch64	TLBI RIPAS2LE1OS	TLB Range Invalidate by Intermediate Physical Address, Stage 2, Last level, EL1, Outer Shareable
AArch64	TLBI RVAAE1	TLB Range Invalidate by VA, All ASID, EL1
AArch64	TLBI RVAAE1IS	TLB Range Invalidate by VA, All ASID, EL1, Inner Shareable
AArch64	TLBI RVAAE1OS	TLB Range Invalidate by VA, All ASID, EL1, Outer Shareable
AArch64	TLBI RVAALE1	TLB Range Invalidate by VA, All ASID, Last level, EL1
AArch64	TLBI RVAALE1IS	TLB Range Invalidate by VA, All ASID, Last Level, EL1, Inner Shareable
AArch64	TLBI RVAALE1OS	TLB Range Invalidate by VA, All ASID, Last Level, EL1, Outer Shareable
AArch64	TLBI RVAE1	TLB Range Invalidate by VA, EL1
AArch64	TLBI RVAE1IS	TLB Range Invalidate by VA, EL1, Inner Shareable
AArch64	TLBI RVAE1OS	TLB Range Invalidate by VA, EL1, Outer Shareable
AArch64	TLBI RVAE2	TLB Range Invalidate by VA, EL2
AArch64	TLBI RVAE2IS	TLB Range Invalidate by VA, EL2, Inner Shareable
AArch64	TLBI RVAE2OS	TLB Range Invalidate by VA, EL2, Outer Shareable
AArch64	TLBI RVAE3	TLB Range Invalidate by VA, EL3
AArch64	TLBI RVAE3IS	TLB Range Invalidate by VA, EL3, Inner Shareable
AArch64	TLBI RVAE3OS	TLB Range Invalidate by VA, EL3, Outer Shareable
AArch64	TLBI RVALE1	TLB Range Invalidate by VA, Last level, EL1
AArch64	TLBI RVALE1IS	TLB Range Invalidate by VA, Last level, EL1, Inner Shareable
AArch64	TLBI RVALE1OS	TLB Range Invalidate by VA, Last level, EL1, Outer Shareable
AArch64	TLBI RVALE2	TLB Range Invalidate by VA, Last level, EL2
AArch64	TLBI RVALE2IS	TLB Range Invalidate by VA, Last level, EL2, Inner Shareable
AArch64	TLBI RVALE2OS	TLB Range Invalidate by VA, Last level, EL2, Outer Shareable
AArch64	TLBI RVALE3	TLB Range Invalidate by VA, Last level, EL3
AArch64	TLBI RVALE3IS	TLB Range Invalidate by VA, Last level, EL3, Inner Shareable
AArch64	TLBI RVALE3OS	TLB Range Invalidate by VA, Last level, EL3, Outer Shareable
AArch64	TLBI VAAE1	TLB Invalidate by VA, All ASID, EL1
AArch64	TLBI VAAE1IS	TLB Invalidate by VA, All ASID, EL1, Inner Shareable
AArch64	TLBI VAAE1OS	TLB Invalidate by VA, All ASID, EL1, Outer Shareable
AArch64	TLBI VAALE1	TLB Invalidate by VA, All ASID, Last level, EL1
AArch64	TLBI VAALE1IS	TLB Invalidate by VA, All ASID, Last Level, EL1, Inner Shareable
AArch64	TLBI VAALE1OS	TLB Invalidate by VA, All ASID, Last Level, EL1, Outer Shareable
AArch64	TLBI VAE1	TLB Invalidate by VA, EL1
AArch64	TLBI VAE1IS	TLB Invalidate by VA, EL1, Inner Shareable
AArch64	TLBI VAE1OS	TLB Invalidate by VA, EL1, Outer Shareable
AArch64	TLBI VAE2	TLB Invalidate by VA, EL2
AArch64	TLBI VAE2IS	TLB Invalidate by VA, EL2, Inner Shareable
AArch64	TLBI VAE2OS	TLB Invalidate by VA, EL2, Outer Shareable
AArch64	TLBI VAE3	TLB Invalidate by VA, EL3
AArch64	TLBI VAE3IS	TLB Invalidate by VA, EL3, Inner Shareable
AArch64	TLBI VAE3OS	TLB Invalidate by VA, EL3, Outer Shareable
AArch64	TLBI VALE1	TLB Invalidate by VA, Last level, EL1
AArch64	TLBI VALE1IS	TLB Invalidate by VA, Last level, EL1, Inner Shareable
AArch64	TLBI VALE1OS	TLB Invalidate by VA, Last level, EL1, Outer Shareable
AArch64	TLBI VALE2	TLB Invalidate by VA, Last level, EL2
AArch64	TLBI VALE2IS	TLB Invalidate by VA, Last level, EL2, Inner Shareable
AArch64	TLBI VALE2OS	TLB Invalidate by VA, Last level, EL2, Outer Shareable
AArch64	TLBI VALE3	TLB Invalidate by VA, Last level, EL3
AArch64	TLBI VALE3IS	TLB Invalidate by VA, Last level, EL3, Inner Shareable
AArch64	TLBI VALE3OS	TLB Invalidate by VA, Last level, EL3, Outer Shareable
AArch64	TLBI VMALLE1	TLB Invalidate by VMID, All at stage 1, EL1
AArch64	TLBI VMALLE1IS	TLB Invalidate by VMID, All at stage 1, EL1, Inner Shareable
AArch64	TLBI VMALLE1OS	TLB Invalidate by VMID, All at stage 1, EL1, Outer Shareable

Exec state	Name	Description
AArch64	TLBI VMALLS12E1	TLB Invalidate by VMID, All at Stage 1 and 2, EL1
AArch64	TLBI VMALLS12E1IS	TLB Invalidate by VMID, All at Stage 1 and 2, EL1, Inner Shareable
AArch64	TLBI VMALLS12E1OS	TLB Invalidate by VMID, All at Stage 1 and 2, EL1, Outer Shareable

In the PMU functional group:

Exec state	Name	Description
AArch32	PMCCFILTR	Performance Monitors Cycle Count Filter Register
AArch32	PMCCNTR	Performance Monitors Cycle Count Register
AArch32	PMCEID0	Performance Monitors Common Event Identification register 0
AArch32	PMCEID1	Performance Monitors Common Event Identification register 1
AArch32	PMCEID2	Performance Monitors Common Event Identification register 2
AArch32	PMCEID3	Performance Monitors Common Event Identification register 3
AArch32	PMCNTENCLR	Performance Monitors Count Enable Clear register
AArch32	PMCNTENSET	Performance Monitors Count Enable Set register
AArch32	PMCR	Performance Monitors Control Register
AArch32	PMEVCNTR<n>	Performance Monitors Event Count Registers
AArch32	PMEVTYPER<n>	Performance Monitors Event Type Registers
AArch32	PMINTENCLR	Performance Monitors Interrupt Enable Clear register
AArch32	PMINTENSET	Performance Monitors Interrupt Enable Set register
AArch32	PMMIR	Performance Monitors Machine Identification Register
AArch32	PMOVSR	Performance Monitors Overflow Flag Status Register
AArch32	PMOVSSET	Performance Monitors Overflow Flag Status Set register
AArch32	PMSELR	Performance Monitors Event Counter Selection Register
AArch32	PMSWINC	Performance Monitors Software Increment register
AArch32	PMUSERENR	Performance Monitors User Enable Register
AArch32	PMXEVCNTR	Performance Monitors Selected Event Count Register
AArch32	PMXEVTYPER	Performance Monitors Selected Event Type Register
AArch64	PMCCFILTR_EL0	Performance Monitors Cycle Count Filter Register
AArch64	PMCCNTR_EL0	Performance Monitors Cycle Count Register
AArch64	PMCEID0_EL0	Performance Monitors Common Event Identification register 0
AArch64	PMCEID1_EL0	Performance Monitors Common Event Identification register 1
AArch64	PMCNTENCLR_EL0	Performance Monitors Count Enable Clear register
AArch64	PMCNTENSET_EL0	Performance Monitors Count Enable Set register
AArch64	PMCR_EL0	Performance Monitors Control Register
AArch64	PMEVCNTR<n>_EL0	Performance Monitors Event Count Registers
AArch64	PMEVTYPER<n>_EL0	Performance Monitors Event Type Registers
AArch64	PMINTENCLR_EL1	Performance Monitors Interrupt Enable Clear register
AArch64	PMINTENSET_EL1	Performance Monitors Interrupt Enable Set register
AArch64	PMMIR_EL1	Performance Monitors Machine Identification Register
AArch64	PMOVSCLR_EL0	Performance Monitors Overflow Flag Status Clear Register
AArch64	PMOVSSET_EL0	Performance Monitors Overflow Flag Status Set register
AArch64	PMSELR_EL0	Performance Monitors Event Counter Selection Register
AArch64	PMSWINC_EL0	Performance Monitors Software Increment register
AArch64	PMUSERENR_EL0	Performance Monitors User Enable Register
AArch64	PMXEVCNTR_EL0	Performance Monitors Selected Event Count Register
AArch64	PMXEVTYPER_EL0	Performance Monitors Selected Event Type Register
External	PMAUTHSTATUS	Performance Monitors Authentication Status register
External	PMCCFILTR_EL0	Performance Monitors Cycle Counter Filter Register
External	PMCCNTR_EL0	Performance Monitors Cycle Counter
External	PMCEID0	Performance Monitors Common Event Identification register 0
External	PMCEID1	Performance Monitors Common Event Identification register 1
External	PMCEID2	Performance Monitors Common Event Identification register 2
External	PMCEID3	Performance Monitors Common Event Identification register 3
External	PMCFGR	Performance Monitors Configuration Register
External	PMCID1SR	CONTEXTIDR_EL1 Sample Register
External	PMCID2SR	CONTEXTIDR_EL2 Sample Register
External	PMCIDR0	Performance Monitors Component Identification Register 0
External	PMCIDR1	Performance Monitors Component Identification Register 1
External	PMCIDR2	Performance Monitors Component Identification Register 2
External	PMCIDR3	Performance Monitors Component Identification Register 3
External	PMCNTENCLR_EL0	Performance Monitors Count Enable Clear register
External	PMCNTENSET_EL0	Performance Monitors Count Enable Set register

Exec state	Name	Description
External	PMCR_EL0	Performance Monitors Control Register
External	PMDEVAFF0	Performance Monitors Device Affinity register 0
External	PMDEVAFF1	Performance Monitors Device Affinity register 1
External	PMDEVARCH	Performance Monitors Device Architecture register
External	PMDEVID	Performance Monitors Device ID register
External	PMDEVTYPE	Performance Monitors Device Type register
External	PMEVCNTR<n>_EL0	Performance Monitors Event Count Registers
External	PMEVTYPER<n>_EL0	Performance Monitors Event Type Registers
External	PMINTENCLR_EL1	Performance Monitors Interrupt Enable Clear register
External	PMINTENSET_EL1	Performance Monitors Interrupt Enable Set register
External	PMITCTRL	Performance Monitors Integration mode Control register
External	PMLAR	Performance Monitors Lock Access Register
External	PMLSR	Performance Monitors Lock Status Register
External	PMOVSCLR_EL0	Performance Monitors Overflow Flag Status Clear register
External	PMOVSSET_EL0	Performance Monitors Overflow Flag Status Set register
External	PMPCSR	Program Counter Sample Register
External	PMPIDR0	Performance Monitors Peripheral Identification Register 0
External	PMPIDR1	Performance Monitors Peripheral Identification Register 1
External	PMPIDR2	Performance Monitors Peripheral Identification Register 2
External	PMPIDR3	Performance Monitors Peripheral Identification Register 3
External	PMPIDR4	Performance Monitors Peripheral Identification Register 4
External	PMSWINC_EL0	Performance Monitors Software Increment register
External	PMVIDSR	VMID Sample Register

In the Reset functional group:

Exec state	Name	Description
AArch32	HRMR	Hyp Reset Management Register
AArch32	RMR	Reset Management Register
AArch32	RVBAR	Reset Vector Base Address Register
AArch64	RMR_EL1	Reset Management Register (EL1)
AArch64	RMR_EL2	Reset Management Register (EL2)
AArch64	RMR_EL3	Reset Management Register (EL3)
AArch64	RVBAR_EL1	Reset Vector Base Address Register (if EL2 and EL3 not implemented)
AArch64	RVBAR_EL2	Reset Vector Base Address Register (if EL3 not implemented)
AArch64	RVBAR_EL3	Reset Vector Base Address Register (if EL3 implemented)

In the Thread functional group:

Exec state	Name	Description
AArch32	HTPIDR	Hyp Software Thread ID Register
AArch32	TPIDRPRW	PL1 Software Thread ID Register
AArch32	TPIDRURO	PL0 Read-Only Software Thread ID Register
AArch32	TPIDRURW	PL0 Read/Write Software Thread ID Register
AArch64	SCXTNUM_EL0	EL0 Read/Write Software Context Number
AArch64	SCXTNUM_EL1	EL1 Read/Write Software Context Number
AArch64	SCXTNUM_EL2	EL2 Read/Write Software Context Number
AArch64	SCXTNUM_EL3	EL3 Read/Write Software Context Number
AArch64	TPIDRRO_EL0	EL0 Read-Only Software Thread ID Register
AArch64	TPIDR_EL0	EL0 Read/Write Software Thread ID Register
AArch64	TPIDR_EL1	EL1 Software Thread ID Register
AArch64	TPIDR_EL2	EL2 Software Thread ID Register
AArch64	TPIDR_EL3	EL3 Software Thread ID Register

In the IMP DEF functional group:

Exec state	Name	Description
AArch32	ACTLR	Auxiliary Control Register
AArch32	ACTLR2	Auxiliary Control Register 2
AArch32	ADFSR	Auxiliary Data Fault Status Register
AArch32	AIDR	Auxiliary ID Register

Exec state	Name	Description
AArch32	AIFSR	Auxiliary Instruction Fault Status Register
AArch32	AMAIR0	Auxiliary Memory Attribute Indirection Register 0
AArch32	AMAIR1	Auxiliary Memory Attribute Indirection Register 1
AArch32	HACTLR	Hyp Auxiliary Control Register
AArch32	HACTLR2	Hyp Auxiliary Control Register 2
AArch32	HADFSR	Hyp Auxiliary Data Fault Status Register
AArch32	HAIFSR	Hyp Auxiliary Instruction Fault Status Register
AArch32	HAMAIR0	Hyp Auxiliary Memory Attribute Indirection Register 0
AArch32	HAMAIR1	Hyp Auxiliary Memory Attribute Indirection Register 1
AArch64	ACTLR_EL1	Auxiliary Control Register (EL1)
AArch64	ACTLR_EL2	Auxiliary Control Register (EL2)
AArch64	ACTLR_EL3	Auxiliary Control Register (EL3)
AArch64	AFSR0_EL1	Auxiliary Fault Status Register 0 (EL1)
AArch64	AFSR0_EL2	Auxiliary Fault Status Register 0 (EL2)
AArch64	AFSR0_EL3	Auxiliary Fault Status Register 0 (EL3)
AArch64	AFSR1_EL1	Auxiliary Fault Status Register 1 (EL1)
AArch64	AFSR1_EL2	Auxiliary Fault Status Register 1 (EL2)
AArch64	AFSR1_EL3	Auxiliary Fault Status Register 1 (EL3)
AArch64	AIDR_EL1	Auxiliary ID Register
AArch64	AMAIR_EL1	Auxiliary Memory Attribute Indirection Register (EL1)
AArch64	AMAIR_EL2	Auxiliary Memory Attribute Indirection Register (EL2)
AArch64	AMAIR_EL3	Auxiliary Memory Attribute Indirection Register (EL3)
AArch64	HACR_EL2	Hypervisor Auxiliary Control Register
AArch64	S1_<op1>_<Cn>_<Cm>_<op2>	IMPLEMENTATION DEFINED maintenance instructions
AArch64	S3_<op1>_<Cn>_<Cm>_<op2>	IMPLEMENTATION DEFINED registers

In the Timer functional group:

Exec state	Name	Description
AArch32	CNTFRQ	Counter-timer Frequency register
AArch32	CNTHPS_CTL	Counter-timer Secure Physical Timer Control Register (EL2)
AArch32	CNTHPS_CVAL	Counter-timer Secure Physical Timer CompareValue Register (EL2)
AArch32	CNTHPS_TVAL	Counter-timer Secure Physical Timer TimerValue Register (EL2)
AArch32	CNTHP_CTL	Counter-timer Hyp Physical Timer Control register
AArch32	CNTHVS_CTL	Counter-timer Secure Virtual Timer Control Register (EL2)
AArch32	CNTHVS_CVAL	Counter-timer Secure Virtual Timer CompareValue Register (EL2)
AArch32	CNTHVS_TVAL	Counter-timer Secure Virtual Timer TimerValue Register (EL2)
AArch32	CNTHV_CTL	Counter-timer Virtual Timer Control register (EL2)
AArch32	CNTHV_CVAL	Counter-timer Virtual Timer CompareValue register (EL2)
AArch32	CNTHV_TVAL	Counter-timer Virtual Timer TimerValue register (EL2)
AArch32	CNTKCTL	Counter-timer Kernel Control register
AArch32	CNTPCT	Counter-timer Physical Count register
AArch32	CNTP_CTL	Counter-timer Physical Timer Control register
AArch32	CNTP_CVAL	Counter-timer Physical Timer CompareValue register
AArch32	CNTP_TVAL	Counter-timer Physical Timer TimerValue register
AArch32	CNTVCT	Counter-timer Virtual Count register
AArch32	CNTV_CTL	Counter-timer Virtual Timer Control register
AArch32	CNTV_CVAL	Counter-timer Virtual Timer CompareValue register
AArch32	CNTV_TVAL	Counter-timer Virtual Timer TimerValue register
AArch64	CNTFRQ_EL0	Counter-timer Frequency register
AArch64	CNTHVS_CTL_EL2	Counter-timer Secure Virtual Timer Control register (EL2)
AArch64	CNTHVS_CVAL_EL2	Counter-timer Secure Virtual Timer CompareValue register (EL2)
AArch64	CNTHVS_TVAL_EL2	Counter-timer Secure Virtual Timer TimerValue register (EL2)
AArch64	CNTHV_CTL_EL2	Counter-timer Virtual Timer Control register (EL2)
AArch64	CNTHV_CVAL_EL2	Counter-timer Virtual Timer CompareValue register (EL2)
AArch64	CNTHV_TVAL_EL2	Counter-timer Virtual Timer TimerValue Register (EL2)
AArch64	CNTKCTL_EL1	Counter-timer Kernel Control register
AArch64	CNTPCT_EL0	Counter-timer Physical Count register
AArch64	CNTPS_CTL_EL1	Counter-timer Physical Secure Timer Control register
AArch64	CNTPS_CVAL_EL1	Counter-timer Physical Secure Timer CompareValue register
AArch64	CNTPS_TVAL_EL1	Counter-timer Physical Secure Timer TimerValue register
AArch64	CNTP_CTL_EL0	Counter-timer Physical Timer Control register
AArch64	CNTP_CVAL_EL0	Counter-timer Physical Timer CompareValue register

Exec state	Name	Description
AArch64	CNTP_TVAL_EL0	Counter-timer Physical Timer TimerValue register
AArch64	CNTVCT_EL0	Counter-timer Virtual Count register
AArch64	CNTV_CTL_EL0	Counter-timer Virtual Timer Control register
AArch64	CNTV_CVAL_EL0	Counter-timer Virtual Timer CompareValue register
AArch64	CNTV_TVAL_EL0	Counter-timer Virtual Timer TimerValue register
External	CNTACR<n>	Counter-timer Access Control Registers
External	CNTCR	Counter Control Register
External	CNTCV	Counter Count Value register
External	CNTEL0ACR	Counter-timer EL0 Access Control Register
External	CNTFID0	Counter Frequency ID
External	CNTFID<n>	Counter Frequency IDs, n > 0
External	CNTFRQ	Counter-timer Frequency
External	CNTID	Counter Identification Register
External	CNTNSAR	Counter-timer Non-secure Access Register
External	CNTPCT	Counter-timer Physical Count
External	CNTP_CTL	Counter-timer Physical Timer Control
External	CNTP_CVAL	Counter-timer Physical Timer CompareValue
External	CNTP_TVAL	Counter-timer Physical Timer TimerValue
External	CNTSCR	Counter Scale Register
External	CNTSR	Counter Status Register
External	CNTTIDR	Counter-timer Timer ID Register
External	CNTVCT	Counter-timer Virtual Count
External	CNTVOFF	Counter-timer Virtual Offset
External	CNTVOFF<n>	Counter-timer Virtual Offsets
External	CNTV_CTL	Counter-timer Virtual Timer Control
External	CNTV_CVAL	Counter-timer Virtual Timer CompareValue
External	CNTV_TVAL	Counter-timer Virtual Timer TimerValue
External	CounterID<n>	Counter ID registers

In the Debug functional group:

Exec state	Name	Description
AArch32	DBGAUTHSTATUS	Debug Authentication Status register
AArch32	DBGBCR<n>	Debug Breakpoint Control Registers
AArch32	DBGBVR<n>	Debug Breakpoint Value Registers
AArch32	DBGBXVR<n>	Debug Breakpoint Extended Value Registers
AArch32	DBGCLAIMCLR	Debug Claim Tag Clear register
AArch32	DBGCLAIMSET	Debug Claim Tag Set register
AArch32	DBGDCCINT	DCC Interrupt Enable Register
AArch32	DBGDEVID	Debug Device ID register 0
AArch32	DBGDEVID1	Debug Device ID register 1
AArch32	DBGDEVID2	Debug Device ID register 2
AArch32	DBGDIDR	Debug ID Register
AArch32	DBGDRAR	Debug ROM Address Register
AArch32	DBGDSAR	Debug Self Address Register
AArch32	DBGDSCRext	Debug Status and Control Register, External View
AArch32	DBGDSCRint	Debug Status and Control Register, Internal View
AArch32	DBGDTRRXext	Debug OS Lock Data Transfer Register, Receive, External View
AArch32	DBGDTRRXint	Debug Data Transfer Register, Receive
AArch32	DBGDTRTXext	Debug OS Lock Data Transfer Register, Transmit
AArch32	DBGDTRTXint	Debug Data Transfer Register, Transmit
AArch32	DBGOSDLR	Debug OS Double Lock Register
AArch32	DBGOSECCR	Debug OS Lock Exception Catch Control Register
AArch32	DBGOSLAR	Debug OS Lock Access Register
AArch32	DBGOSLSR	Debug OS Lock Status Register
AArch32	DBGPRCR	Debug Power Control Register
AArch32	DBGVCR	Debug Vector Catch Register
AArch32	DBGWCR<n>	Debug Watchpoint Control Registers
AArch32	DBGWFAR	Debug Watchpoint Fault Address Register
AArch32	DBGWVR<n>	Debug Watchpoint Value Registers
AArch32	TRFCR	Trace Filter Control Register
AArch64	DBGAUTHSTATUS_EL1	Debug Authentication Status register
AArch64	DBGBCR<n>_EL1	Debug Breakpoint Control Registers

Exec state	Name	Description
AArch64	DBGBVR<n>_EL1	Debug Breakpoint Value Registers
AArch64	DBGCLAIMCLR_EL1	Debug Claim Tag Clear register
AArch64	DBGCLAIMSET_EL1	Debug Claim Tag Set register
AArch64	DBGDTRRX_EL0	Debug Data Transfer Register, Receive
AArch64	DBGDTRTX_EL0	Debug Data Transfer Register, Transmit
AArch64	DBGDTR_EL0	Debug Data Transfer Register, half-duplex
AArch64	DBGPRCR_EL1	Debug Power Control Register
AArch64	DBGVCR32_EL2	Debug Vector Catch Register
AArch64	DBGWCR<n>_EL1	Debug Watchpoint Control Registers
AArch64	DBGWVR<n>_EL1	Debug Watchpoint Value Registers
AArch64	DLR_EL0	Debug Link Register
AArch64	DSPSR_EL0	Debug Saved Program Status Register
AArch64	MDCCINT_EL1	Monitor DCC Interrupt Enable Register
AArch64	MDCCSR_EL0	Monitor DCC Status Register
AArch64	MDRAR_EL1	Monitor Debug ROM Address Register
AArch64	MDSCR_EL1	Monitor Debug System Control Register
AArch64	OSDLR_EL1	OS Double Lock Register
AArch64	OSDTRRX_EL1	OS Lock Data Transfer Register, Receive
AArch64	OSDTRTX_EL1	OS Lock Data Transfer Register, Transmit
AArch64	OSECCR_EL1	OS Lock Exception Catch Control Register
AArch64	OSLAR_EL1	OS Lock Access Register
AArch64	OSLSR_EL1	OS Lock Status Register
AArch64	TRFCR_EL1	Trace Filter Control Register (EL1)
AArch64	TRFCR_EL2	Trace Filter Control Register (EL2)
External	DBGAUTHSTATUS_EL1	Debug Authentication Status register
External	DBGBCR<n>_EL1	Debug Breakpoint Control Registers
External	DBGBVR<n>_EL1	Debug Breakpoint Value Registers
External	DBGCLAIMCLR_EL1	Debug Claim Tag Clear register
External	DBGCLAIMSET_EL1	Debug Claim Tag Set register
External	DBGDTRRX_EL0	Debug Data Transfer Register, Receive
External	DBGDTRTX_EL0	Debug Data Transfer Register, Transmit
External	DBGWCR<n>_EL1	Debug Watchpoint Control Registers
External	DBGWVR<n>_EL1	Debug Watchpoint Value Registers
External	EDACR	External Debug Auxiliary Control Register
External	EDCIDR0	External Debug Component Identification Register 0
External	EDCIDR1	External Debug Component Identification Register 1
External	EDCIDR2	External Debug Component Identification Register 2
External	EDCIDR3	External Debug Component Identification Register 3
External	EDCIDSR	External Debug Context ID Sample Register
External	EDDEVAFF0	External Debug Device Affinity register 0
External	EDDEVAFF1	External Debug Device Affinity register 1
External	EDDEVARCH	External Debug Device Architecture register
External	EDDEVID	External Debug Device ID register 0
External	EDDEVID1	External Debug Device ID register 1
External	EDDEVID2	External Debug Device ID register 2
External	EDDEVTYPE	External Debug Device Type register
External	EDECCR	External Debug Exception Catch Control Register
External	EDECR	External Debug Execution Control Register
External	EDESR	External Debug Event Status Register
External	EDITCTRL	External Debug Integration mode Control register
External	EDITR	External Debug Instruction Transfer Register
External	EDLAR	External Debug Lock Access Register
External	EDLSR	External Debug Lock Status Register
External	EDPCSR	External Debug Program Counter Sample Register
External	EDPIDR0	External Debug Peripheral Identification Register 0
External	EDPIDR1	External Debug Peripheral Identification Register 1
External	EDPIDR2	External Debug Peripheral Identification Register 2
External	EDPIDR3	External Debug Peripheral Identification Register 3
External	EDPIDR4	External Debug Peripheral Identification Register 4
External	EDPRCR	External Debug Power/Reset Control Register
External	EDPRSR	External Debug Processor Status Register
External	EDRCR	External Debug Reserve Control Register
External	EDSCR	External Debug Status and Control Register
External	EDVIDSR	External Debug Virtual Context Sample Register

Exec state	Name	Description
External	EDWAR	External Debug Watchpoint Address Register
External	OSLAR_EL1	OS Lock Access Register

In the CTI functional group:

Exec state	Name	Description
External	ASICCTL	CTI External Multiplexer Control register
External	CTIAPPCLEAR	CTI Application Trigger Clear register
External	CTIAPPPULSE	CTI Application Pulse register
External	CTIAPPSET	CTI Application Trigger Set register
External	CTIAUTHSTATUS	CTI Authentication Status register
External	CTICHINSTATUS	CTI Channel In Status register
External	CTICHOUTSTATUS	CTI Channel Out Status register
External	CTICIDR0	CTI Component Identification Register 0
External	CTICIDR1	CTI Component Identification Register 1
External	CTICIDR2	CTI Component Identification Register 2
External	CTICIDR3	CTI Component Identification Register 3
External	CTICLAIMCLR	CTI Claim Tag Clear register
External	CTICLAIMSET	CTI Claim Tag Set register
External	CTICONTROL	CTI Control register
External	CTIDEVAFF0	CTI Device Affinity register 0
External	CTIDEVAFF1	CTI Device Affinity register 1
External	CTIDEVARCH	CTI Device Architecture register
External	CTIDEVCTL	CTI Device Control register
External	CTIDEVID	CTI Device ID register 0
External	CTIDEVID1	CTI Device ID register 1
External	CTIDEVID2	CTI Device ID register 2
External	CTIDEVTYPE	CTI Device Type register
External	CTIGATE	CTI Channel Gate Enable register
External	CTIINEN<n>	CTI Input Trigger to Output Channel Enable registers
External	CTIINTACK	CTI Output Trigger Acknowledge register
External	CTIITCTRL	CTI Integration mode Control register
External	CTILAR	CTI Lock Access Register
External	CTILSR	CTI Lock Status Register
External	CTIOUTEN<n>	CTI Input Channel to Output Trigger Enable registers
External	CTIPIDR0	CTI Peripheral Identification Register 0
External	CTIPIDR1	CTI Peripheral Identification Register 1
External	CTIPIDR2	CTI Peripheral Identification Register 2
External	CTIPIDR3	CTI Peripheral Identification Register 3
External	CTIPIDR4	CTI Peripheral Identification Register 4
External	CTITRIGINSTATUS	CTI Trigger In Status register
External	CTITRIGOUTSTATUS	CTI Trigger Out Status register

In the Virt functional group:

Exec state	Name	Description
AArch32	ATS1HR	Address Translate Stage 1 Hyp mode Read
AArch32	ATS1HW	Address Translate Stage 1 Hyp mode Write
AArch32	CNTHCTL	Counter-timer Hyp Control register
AArch32	CNTHP_CVAL	Counter-timer Hyp Physical CompareValue register
AArch32	CNTHP_TVAL	Counter-timer Hyp Physical Timer TimerValue register
AArch32	CNTVOFF	Counter-timer Virtual Offset register
AArch32	HACR	Hyp Auxiliary Configuration Register
AArch32	HACTLR	Hyp Auxiliary Control Register
AArch32	HACTLR2	Hyp Auxiliary Control Register 2
AArch32	HADFSR	Hyp Auxiliary Data Fault Status Register
AArch32	HAIFSR	Hyp Auxiliary Instruction Fault Status Register
AArch32	HAMAIR0	Hyp Auxiliary Memory Attribute Indirection Register 0
AArch32	HAMAIR1	Hyp Auxiliary Memory Attribute Indirection Register 1
AArch32	HCPTR	Hyp Architectural Feature Trap Register
AArch32	HCR	Hyp Configuration Register
AArch32	HCR2	Hyp Configuration Register 2

Exec state	Name	Description
AArch32	HDCR	Hyp Debug Control Register
AArch32	HDFAR	Hyp Data Fault Address Register
AArch32	HIFAR	Hyp Instruction Fault Address Register
AArch32	HMAIR0	Hyp Memory Attribute Indirection Register 0
AArch32	HMAIR1	Hyp Memory Attribute Indirection Register 1
AArch32	HPFAR	Hyp IPA Fault Address Register
AArch32	HRMR	Hyp Reset Management Register
AArch32	HSCTLR	Hyp System Control Register
AArch32	HSR	Hyp Syndrome Register
AArch32	HSTR	Hyp System Trap Register
AArch32	HTCR	Hyp Translation Control Register
AArch32	HTPIDR	Hyp Software Thread ID Register
AArch32	HTRFCR	Hyp Trace Filter Control Register
AArch32	HTTBR	Hyp Translation Table Base Register
AArch32	HVBAR	Hyp Vector Base Address Register
AArch32	ICC_HSRE	Interrupt Controller Hyp System Register Enable register
AArch32	ICH_AP0R<n>	Interrupt Controller Hyp Active Priorities Group 0 Registers
AArch32	ICH_AP1R<n>	Interrupt Controller Hyp Active Priorities Group 1 Registers
AArch32	ICH_EISR	Interrupt Controller End of Interrupt Status Register
AArch32	ICH_ELRSR	Interrupt Controller Empty List Register Status Register
AArch32	ICH_HCR	Interrupt Controller Hyp Control Register
AArch32	ICH_LR<n>	Interrupt Controller List Registers
AArch32	ICH_LRC<n>	Interrupt Controller List Registers
AArch32	ICH_MISR	Interrupt Controller Maintenance Interrupt State Register
AArch32	ICH_VMCR	Interrupt Controller Virtual Machine Control Register
AArch32	ICH_VTR	Interrupt Controller VGIC Type Register
AArch32	TLBIALLH	TLB Invalidate All, Hyp mode
AArch32	TLBIALLHIS	TLB Invalidate All, Hyp mode, Inner Shareable
AArch32	TLBIIPAS2	TLB Invalidate by Intermediate Physical Address, Stage 2
AArch32	TLBIIPAS2IS	TLB Invalidate by Intermediate Physical Address, Stage 2, Inner Shareable
AArch32	TLBIIPAS2L	TLB Invalidate by Intermediate Physical Address, Stage 2, Last level
AArch32	TLBIIPAS2LIS	TLB Invalidate by Intermediate Physical Address, Stage 2, Last level, Inner Shareable
AArch32	TLBIMVAH	TLB Invalidate by VA, Hyp mode
AArch32	TLBIMVAHIS	TLB Invalidate by VA, Hyp mode, Inner Shareable
AArch32	TLBIMVALH	TLB Invalidate by VA, Last level, Hyp mode
AArch32	TLBIMVALHIS	TLB Invalidate by VA, Last level, Hyp mode, Inner Shareable
AArch32	VMPIDR	Virtualization Multiprocessor ID Register
AArch32	VPIDR	Virtualization Processor ID Register
AArch32	VTCR	Virtualization Translation Control Register
AArch32	VTTBR	Virtualization Translation Table Base Register
AArch64	ACTLR_EL2	Auxiliary Control Register (EL2)
AArch64	AFSR0_EL2	Auxiliary Fault Status Register 0 (EL2)
AArch64	AFSR1_EL2	Auxiliary Fault Status Register 1 (EL2)
AArch64	AMAIR_EL2	Auxiliary Memory Attribute Indirection Register (EL2)
AArch64	CNTHCTL_EL2	Counter-timer Hypervisor Control register
AArch64	CNTHPS_CTL_EL2	Counter-timer Secure Physical Timer Control register (EL2)
AArch64	CNTHPS_CVAL_EL2	Counter-timer Secure Physical Timer CompareValue register (EL2)
AArch64	CNTHPS_TVAL_EL2	Counter-timer Secure Physical Timer TimerValue register (EL2)
AArch64	CNTHP_CTL_EL2	Counter-timer Hypervisor Physical Timer Control register
AArch64	CNTHP_CVAL_EL2	Counter-timer Physical Timer CompareValue register (EL2)
AArch64	CNTHP_TVAL_EL2	Counter-timer Physical Timer TimerValue register (EL2)
AArch64	CNTVOFF_EL2	Counter-timer Virtual Offset register
AArch64	CPTR_EL2	Architectural Feature Trap Register (EL2)
AArch64	ESR_EL2	Exception Syndrome Register (EL2)
AArch64	FAR_EL2	Fault Address Register (EL2)
AArch64	HACR_EL2	Hypervisor Auxiliary Control Register
AArch64	HCR_EL2	Hypervisor Configuration Register
AArch64	HPFAR_EL2	Hypervisor IPA Fault Address Register
AArch64	HSTR_EL2	Hypervisor System Trap Register
AArch64	ICC_SRE_EL2	Interrupt Controller System Register Enable register (EL2)
AArch64	ICH_AP0R<n>_EL2	Interrupt Controller Hyp Active Priorities Group 0 Registers
AArch64	ICH_AP1R<n>_EL2	Interrupt Controller Hyp Active Priorities Group 1 Registers
AArch64	ICH_EISR_EL2	Interrupt Controller End of Interrupt Status Register
AArch64	ICH_ELRSR_EL2	Interrupt Controller Empty List Register Status Register

Exec state	Name	Description
AArch64	ICH_HCR_EL2	Interrupt Controller Hyp Control Register
AArch64	ICH_LR<n>_EL2	Interrupt Controller List Registers
AArch64	ICH_MISR_EL2	Interrupt Controller Maintenance Interrupt State Register
AArch64	ICH_VMCR_EL2	Interrupt Controller Virtual Machine Control Register
AArch64	ICH_VTR_EL2	Interrupt Controller VGIC Type Register
AArch64	MAIR_EL2	Memory Attribute Indirection Register (EL2)
AArch64	MDCR_EL2	Monitor Debug Configuration Register (EL2)
AArch64	MPAMHCR_EL2	MPAM Hypervisor Control Register (EL2)
AArch64	MPAMVPM0_EL2	MPAM Virtual PARTID Mapping Register 0
AArch64	MPAMVPM1_EL2	MPAM Virtual PARTID Mapping Register 1
AArch64	MPAMVPM2_EL2	MPAM Virtual PARTID Mapping Register 2
AArch64	MPAMVPM3_EL2	MPAM Virtual PARTID Mapping Register 3
AArch64	MPAMVPM4_EL2	MPAM Virtual PARTID Mapping Register 4
AArch64	MPAMVPM5_EL2	MPAM Virtual PARTID Mapping Register 5
AArch64	MPAMVPM6_EL2	MPAM Virtual PARTID Mapping Register 6
AArch64	MPAMVPM7_EL2	MPAM Virtual PARTID Mapping Register 7
AArch64	MPAMVPMV_EL2	MPAM Virtual Partition Mapping Valid Register
AArch64	RMR_EL2	Reset Management Register (EL2)
AArch64	SCTLR_EL2	System Control Register (EL2)
AArch64	TCR_EL2	Translation Control Register (EL2)
AArch64	TLBI IPAS2E1	TLB Invalidate by Intermediate Physical Address, Stage 2, EL1
AArch64	TLBI IPAS2E1IS	TLB Invalidate by Intermediate Physical Address, Stage 2, EL1, Inner Shareable
AArch64	TLBI IPAS2E1OS	TLB Invalidate by Intermediate Physical Address, Stage 2, EL1, Outer Shareable
AArch64	TLBI IPAS2LE1	TLB Invalidate by Intermediate Physical Address, Stage 2, Last level, EL1
AArch64	TLBI IPAS2LE1IS	TLB Invalidate by Intermediate Physical Address, Stage 2, Last level, EL1, Inner Shareable
AArch64	TLBI IPAS2LE1OS	TLB Invalidate by Intermediate Physical Address, Stage 2, Last level, EL1, Outer Shareable
AArch64	TLBI RIPAS2E1	TLB Range Invalidate by Intermediate Physical Address, Stage 2, EL1
AArch64	TLBI RIPAS2E1IS	TLB Range Invalidate by Intermediate Physical Address, Stage 2, EL1, Inner Shareable
AArch64	TLBI RIPAS2E1OS	TLB Range Invalidate by Intermediate Physical Address, Stage 2, EL1, Outer Shareable
AArch64	TLBI RIPAS2LE1	TLB Range Invalidate by Intermediate Physical Address, Stage 2, Last level, EL1
AArch64	TLBI RIPAS2LE1IS	TLB Range Invalidate by Intermediate Physical Address, Stage 2, Last level, EL1, Inner Shareable
AArch64	TLBI RIPAS2LE1OS	TLB Range Invalidate by Intermediate Physical Address, Stage 2, Last level, EL1, Outer Shareable
AArch64	TPIDR_EL2	EL2 Software Thread ID Register
AArch64	TTBR0_EL2	Translation Table Base Register 0 (EL2)
AArch64	TTBR1_EL2	Translation Table Base Register 1 (EL2)
AArch64	VBAR_EL2	Vector Base Address Register (EL2)
AArch64	VMPIDR_EL2	Virtualization Multiprocessor ID Register
AArch64	VPIDR_EL2	Virtualization Processor ID Register
AArch64	VTCR_EL2	Virtualization Translation Control Register
AArch64	VTTBR_EL2	Virtualization Translation Table Base Register

In the Secure functional group:

Exec state	Name	Description
AArch32	ICC_MCTLR	Interrupt Controller Monitor Control Register
AArch32	ICC_MSRE	Interrupt Controller Monitor System Register Enable register
AArch32	MVBAR	Monitor Vector Base Address Register
AArch32	NSACR	Non-Secure Access Control Register
AArch32	SCR	Secure Configuration Register
AArch32	SDCR	Secure Debug Control Register
AArch32	SDER	Secure Debug Enable Register
AArch64	ACTLR_EL3	Auxiliary Control Register (EL3)
AArch64	AFSR0_EL3	Auxiliary Fault Status Register 0 (EL3)
AArch64	AFSR1_EL3	Auxiliary Fault Status Register 1 (EL3)
AArch64	AMAIR_EL3	Auxiliary Memory Attribute Indirection Register (EL3)
AArch64	CPTR_EL3	Architectural Feature Trap Register (EL3)
AArch64	ICC_CTLR_EL3	Interrupt Controller Control Register (EL3)
AArch64	ICC_SRE_EL3	Interrupt Controller System Register Enable register (EL3)
AArch64	MDCR_EL3	Monitor Debug Configuration Register (EL3)
AArch64	SCR_EL3	Secure Configuration Register
AArch64	SDER32_EL3	AArch32 Secure Debug Enable Register
AArch64	VBAR_EL3	Vector Base Address Register (EL3)

In the Float functional group:

Exec state	Name	Description
AArch32	FPEXC	Floating-Point Exception Control register
AArch32	FPSCR	Floating-Point Status and Control Register
AArch32	FPSID	Floating-Point System ID register
AArch32	MVFR0	Media and VFP Feature Register 0
AArch32	MVFR1	Media and VFP Feature Register 1
AArch32	MVFR2	Media and VFP Feature Register 2
AArch64	FPCR	Floating-point Control Register
AArch64	FPEXC32_EL2	Floating-Point Exception Control register
AArch64	FPSR	Floating-point Status Register
AArch64	MVFR0_EL1	AArch32 Media and VFP Feature Register 0
AArch64	MVFR1_EL1	AArch32 Media and VFP Feature Register 1
AArch64	MVFR2_EL1	AArch32 Media and VFP Feature Register 2

In the Legacy functional group:

Exec state	Name	Description
AArch32	CP15DMB	Data Memory Barrier System instruction
AArch32	CP15DSB	Data Synchronization Barrier System instruction
AArch32	CP15ISB	Instruction Synchronization Barrier System instruction
AArch32	FCSEIDR	FCSE Process ID register
AArch32	JIDR	Jazelle ID Register
AArch32	JMCR	Jazelle Main Configuration Register
AArch32	JOSCR	Jazelle OS Control Register

In the GIC functional group:

Exec state	Name	Description
AArch32	ICC_AP0R<n>	Interrupt Controller Active Priorities Group 0 Registers
AArch32	ICC_AP1R<n>	Interrupt Controller Active Priorities Group 1 Registers
AArch32	ICC_ASGI1R	Interrupt Controller Alias Software Generated Interrupt Group 1 Register
AArch32	ICC_BPR0	Interrupt Controller Binary Point Register 0
AArch32	ICC_BPR1	Interrupt Controller Binary Point Register 1
AArch32	ICC_CTLR	Interrupt Controller Control Register
AArch32	ICC_DIR	Interrupt Controller Deactivate Interrupt Register
AArch32	ICC_EOIR0	Interrupt Controller End Of Interrupt Register 0
AArch32	ICC_EOIR1	Interrupt Controller End Of Interrupt Register 1
AArch32	ICC_HPPIR0	Interrupt Controller Highest Priority Pending Interrupt Register 0
AArch32	ICC_HPPIR1	Interrupt Controller Highest Priority Pending Interrupt Register 1
AArch32	ICC_HSRE	Interrupt Controller Hyp System Register Enable register
AArch32	ICC_IAR0	Interrupt Controller Interrupt Acknowledge Register 0
AArch32	ICC_IAR1	Interrupt Controller Interrupt Acknowledge Register 1
AArch32	ICC_IGRPEN0	Interrupt Controller Interrupt Group 0 Enable register
AArch32	ICC_IGRPEN1	Interrupt Controller Interrupt Group 1 Enable register
AArch32	ICC_MCTLR	Interrupt Controller Monitor Control Register
AArch32	ICC_MGRPEN1	Interrupt Controller Monitor Interrupt Group 1 Enable register
AArch32	ICC_MSRE	Interrupt Controller Monitor System Register Enable register
AArch32	ICC_PMR	Interrupt Controller Interrupt Priority Mask Register
AArch32	ICC_RPR	Interrupt Controller Running Priority Register
AArch32	ICC_SGI0R	Interrupt Controller Software Generated Interrupt Group 0 Register
AArch32	ICC_SGI1R	Interrupt Controller Software Generated Interrupt Group 1 Register
AArch32	ICC_SRE	Interrupt Controller System Register Enable register
AArch32	ICH_AP0R<n>	Interrupt Controller Hyp Active Priorities Group 0 Registers
AArch32	ICH_AP1R<n>	Interrupt Controller Hyp Active Priorities Group 1 Registers
AArch32	ICH_EISR	Interrupt Controller End of Interrupt Status Register
AArch32	ICH_ELRSR	Interrupt Controller Empty List Register Status Register
AArch32	ICH_HCR	Interrupt Controller Hyp Control Register
AArch32	ICH_LR<n>	Interrupt Controller List Registers
AArch32	ICH_LRC<n>	Interrupt Controller List Registers
AArch32	ICH_MISR	Interrupt Controller Maintenance Interrupt State Register
AArch32	ICH_VMCR	Interrupt Controller Virtual Machine Control Register

Exec state	Name	Description
AArch32	ICH_VTR	Interrupt Controller VGIC Type Register
AArch32	ICV_AP0R<n>	Interrupt Controller Virtual Active Priorities Group 0 Registers
AArch32	ICV_AP1R<n>	Interrupt Controller Virtual Active Priorities Group 1 Registers
AArch32	ICV_BPR0	Interrupt Controller Virtual Binary Point Register 0
AArch32	ICV_BPR1	Interrupt Controller Virtual Binary Point Register 1
AArch32	ICV_CTLR	Interrupt Controller Virtual Control Register
AArch32	ICV_DIR	Interrupt Controller Deactivate Virtual Interrupt Register
AArch32	ICV_EOIR0	Interrupt Controller Virtual End Of Interrupt Register 0
AArch32	ICV_EOIR1	Interrupt Controller Virtual End Of Interrupt Register 1
AArch32	ICV_HPPIR0	Interrupt Controller Virtual Highest Priority Pending Interrupt Register 0
AArch32	ICV_HPPIR1	Interrupt Controller Virtual Highest Priority Pending Interrupt Register 1
AArch32	ICV_IAR0	Interrupt Controller Virtual Interrupt Acknowledge Register 0
AArch32	ICV_IAR1	Interrupt Controller Virtual Interrupt Acknowledge Register 1
AArch32	ICV_IGRPEN0	Interrupt Controller Virtual Interrupt Group 0 Enable register
AArch32	ICV_IGRPEN1	Interrupt Controller Virtual Interrupt Group 1 Enable register
AArch32	ICV_PMR	Interrupt Controller Virtual Interrupt Priority Mask Register
AArch32	ICV_RPR	Interrupt Controller Virtual Running Priority Register
AArch64	ICC_AP0R<n>_EL1	Interrupt Controller Active Priorities Group 0 Registers
AArch64	ICC_AP1R<n>_EL1	Interrupt Controller Active Priorities Group 1 Registers
AArch64	ICC_ASGI1R_EL1	Interrupt Controller Alias Software Generated Interrupt Group 1 Register
AArch64	ICC_BPR0_EL1	Interrupt Controller Binary Point Register 0
AArch64	ICC_BPR1_EL1	Interrupt Controller Binary Point Register 1
AArch64	ICC_CTLR_EL1	Interrupt Controller Control Register (EL1)
AArch64	ICC_CTLR_EL3	Interrupt Controller Control Register (EL3)
AArch64	ICC_DIR_EL1	Interrupt Controller Deactivate Interrupt Register
AArch64	ICC_EOIR0_EL1	Interrupt Controller End Of Interrupt Register 0
AArch64	ICC_EOIR1_EL1	Interrupt Controller End Of Interrupt Register 1
AArch64	ICC_HPPIR0_EL1	Interrupt Controller Highest Priority Pending Interrupt Register 0
AArch64	ICC_HPPIR1_EL1	Interrupt Controller Highest Priority Pending Interrupt Register 1
AArch64	ICC_IAR0_EL1	Interrupt Controller Interrupt Acknowledge Register 0
AArch64	ICC_IAR1_EL1	Interrupt Controller Interrupt Acknowledge Register 1
AArch64	ICC_IGRPEN0_EL1	Interrupt Controller Interrupt Group 0 Enable register
AArch64	ICC_IGRPEN1_EL1	Interrupt Controller Interrupt Group 1 Enable register
AArch64	ICC_IGRPEN1_EL3	Interrupt Controller Interrupt Group 1 Enable register (EL3)
AArch64	ICC_PMR_EL1	Interrupt Controller Interrupt Priority Mask Register
AArch64	ICC_RPR_EL1	Interrupt Controller Running Priority Register
AArch64	ICC_SGI0R_EL1	Interrupt Controller Software Generated Interrupt Group 0 Register
AArch64	ICC_SGI1R_EL1	Interrupt Controller Software Generated Interrupt Group 1 Register
AArch64	ICC_SRE_EL1	Interrupt Controller System Register Enable register (EL1)
AArch64	ICC_SRE_EL2	Interrupt Controller System Register Enable register (EL2)
AArch64	ICC_SRE_EL3	Interrupt Controller System Register Enable register (EL3)
AArch64	ICH_AP0R<n>_EL2	Interrupt Controller Hyp Active Priorities Group 0 Registers
AArch64	ICH_AP1R<n>_EL2	Interrupt Controller Hyp Active Priorities Group 1 Registers
AArch64	ICH_EISR_EL2	Interrupt Controller End of Interrupt Status Register
AArch64	ICH_ELRSR_EL2	Interrupt Controller Empty List Register Status Register
AArch64	ICH_HCR_EL2	Interrupt Controller Hyp Control Register
AArch64	ICH_LR<n>_EL2	Interrupt Controller List Registers
AArch64	ICH_MISR_EL2	Interrupt Controller Maintenance Interrupt State Register
AArch64	ICH_VMCR_EL2	Interrupt Controller Virtual Machine Control Register
AArch64	ICH_VTR_EL2	Interrupt Controller VGIC Type Register
AArch64	ICV_AP0R<n>_EL1	Interrupt Controller Virtual Active Priorities Group 0 Registers
AArch64	ICV_AP1R<n>_EL1	Interrupt Controller Virtual Active Priorities Group 1 Registers
AArch64	ICV_BPR0_EL1	Interrupt Controller Virtual Binary Point Register 0
AArch64	ICV_BPR1_EL1	Interrupt Controller Virtual Binary Point Register 1
AArch64	ICV_CTLR_EL1	Interrupt Controller Virtual Control Register
AArch64	ICV_DIR_EL1	Interrupt Controller Deactivate Virtual Interrupt Register
AArch64	ICV_EOIR0_EL1	Interrupt Controller Virtual End Of Interrupt Register 0
AArch64	ICV_EOIR1_EL1	Interrupt Controller Virtual End Of Interrupt Register 1
AArch64	ICV_HPPIR0_EL1	Interrupt Controller Virtual Highest Priority Pending Interrupt Register 0
AArch64	ICV_HPPIR1_EL1	Interrupt Controller Virtual Highest Priority Pending Interrupt Register 1
AArch64	ICV_IAR0_EL1	Interrupt Controller Virtual Interrupt Acknowledge Register 0
AArch64	ICV_IAR1_EL1	Interrupt Controller Virtual Interrupt Acknowledge Register 1
AArch64	ICV_IGRPEN0_EL1	Interrupt Controller Virtual Interrupt Group 0 Enable register
AArch64	ICV_IGRPEN1_EL1	Interrupt Controller Virtual Interrupt Group 1 Enable register

Exec state	Name	Description
AArch64	ICV_PMR_EL1	Interrupt Controller Virtual Interrupt Priority Mask Register
AArch64	ICV_RPR_EL1	Interrupt Controller Virtual Running Priority Register

In the GICD functional group:

Exec state	Name	Description
External	GICD_CLRSPI_NSR	Clear Non-secure SPI Pending Register
External	GICD_CLRSPI_SR	Clear Secure SPI Pending Register
External	GICD_CPENDSGIR<n>	SGI Clear-Pending Registers
External	GICD_CTLR	Distributor Control Register
External	GICD_ICACTIVER<n>	Interrupt Clear-Active Registers
External	GICD_ICACTIVER<n>E	Interrupt Clear-Active Registers (extended SPI range)
External	GICD_ICENABLER<n>	Interrupt Clear-Enable Registers
External	GICD_ICENABLER<n>E	Interrupt Clear-Enable Registers
External	GICD_ICFGR<n>	Interrupt Configuration Registers
External	GICD_ICFGR<n>E	Interrupt Configuration Registers (Extended SPI Range)
External	GICD_ICPENDR<n>	Interrupt Clear-Pending Registers
External	GICD_ICPENDR<n>E	Interrupt Clear-Pending Registers (extended SPI range)
External	GICD_IGROUPR<n>	Interrupt Group Registers
External	GICD_IGROUPR<n>E	Interrupt Group Registers (extended SPI range)
External	GICD_IGRPMODR<n>	Interrupt Group Modifier Registers
External	GICD_IGRPMODR<n>E	Interrupt Group Modifier Registers (extended SPI range)
External	GICD_IIDR	Distributor Implementer Identification Register
External	GICD_IPRIORITYR<n>	Interrupt Priority Registers
External	GICD_IPRIORITYR<n>E	Holds the priority of the corresponding interrupt for each extended SPI supported by the GIC.
External	GICD_IROUTER<n>	Interrupt Routing Registers
External	GICD_IROUTER<n>E	Interrupt Routing Registers (Extended SPI Range)
External	GICD_ISACTIVER<n>	Interrupt Set-Active Registers
External	GICD_ISACTIVER<n>E	Interrupt Set-Active Registers (extended SPI range)
External	GICD_ISENABLER<n>	Interrupt Set-Enable Registers
External	GICD_ISENABLER<n>E	Interrupt Set-Enable Registers
External	GICD_ISPENDR<n>	Interrupt Set-Pending Registers
External	GICD_ISPENDR<n>E	Interrupt Set-Pending Registers (extended SPI range)
External	GICD_ITARGETSR<n>	Interrupt Processor Targets Registers
External	GICD_NSACR<n>	Non-secure Access Control Registers
External	GICD_NSACR<n>E	Non-secure Access Control Registers
External	GICD_SETSPI_NSR	Set Non-secure SPI Pending Register
External	GICD_SETSPI_SR	Set Secure SPI Pending Register
External	GICD_SGIR	Software Generated Interrupt Register
External	GICD_SPENDSGIR<n>	SGI Set-Pending Registers
External	GICD_STATUSR	Error Reporting Status Register
External	GICD_TYPER	Interrupt Controller Type Register

In the GICR functional group:

Exec state	Name	Description
External	GICR_CLRLPIR	Clear LPI Pending Register
External	GICR_CTLR	Redistributor Control Register
External	GICR_ICACTIVER0	Interrupt Clear-Active Register 0
External	GICR_ICACTIVER<n>E	Interrupt Clear-Active Registers
External	GICR_ICENABLER0	Interrupt Clear-Enable Register 0
External	GICR_ICENABLER<n>E	Interrupt Clear-Enable Registers
External	GICR_ICFGR0	Interrupt Configuration Register 0
External	GICR_ICFGR1	Interrupt Configuration Register 1
External	GICR_ICFGR<n>E	Interrupt configuration registers
External	GICR_ICPENDR0	Interrupt Clear-Pending Register 0
External	GICR_ICPENDR<n>E	Interrupt Clear-Pending Registers
External	GICR_IGROUPR0	Interrupt Group Register 0
External	GICR_IGROUPR<n>E	Interrupt Group Registers
External	GICR_IGRPMODR0	Interrupt Group Modifier Register 0
External	GICR_IGRPMODR<n>E	Interrupt Group Modifier Registers
External	GICR_IIDR	Redistributor Implementer Identification Register

Exec state	Name	Description
External	GICR_INVALLR	Redistributor Invalidate All Register
External	GICR_INVLPIR	Redistributor Invalidate LPI Register
External	GICR_IPRIORITYR<n>	Interrupt Priority Registers
External	GICR_IPRIORITYR<n>E	Interrupt Priority Registers (extended PPI range)
External	GICR_ISACTIVER0	Interrupt Set-Active Register 0
External	GICR_ISACTIVER<n>E	Interrupt Set-Active Registers
External	GICR_ISENABLER0	Interrupt Set-Enable Register 0
External	GICR_ISENABLER<n>E	Interrupt Set-Enable Registers
External	GICR_ISPENDR0	Interrupt Set-Pending Register 0
External	GICR_ISPENDR<n>E	Interrupt Set-Pending Registers
External	GICR_MPAMIDR	Report maximum PARTID and PMG Register
External	GICR_NSACR	Non-secure Access Control Register
External	GICR_PARTIDR	Set PARTID and PMG Register
External	GICR_PENDBASER	Redistributor LPI Pending Table Base Address Register
External	GICR_PROPBASER	Redistributor Properties Base Address Register
External	GICR_SETLPIR	Set LPI Pending Register
External	GICR_STATUSR	Error Reporting Status Register
External	GICR_SYNCR	Redistributor Synchronize Register
External	GICR_TYPER	Redistributor Type Register
External	GICR_VPENDBASER	Virtual Redistributor LPI Pending Table Base Address Register
External	GICR_VPROPBASER	Virtual Redistributor Properties Base Address Register
External	GICR_WAKER	Redistributor Wake Register

In the GICC functional group:

Exec state	Name	Description
External	GICC_ABPR	CPU Interface Aliased Binary Point Register
External	GICC_AEOIR	CPU Interface Aliased End Of Interrupt Register
External	GICC_AHPPIR	CPU Interface Aliased Highest Priority Pending Interrupt Register
External	GICC_AIAR	CPU Interface Aliased Interrupt Acknowledge Register
External	GICC_APR<n>	CPU Interface Active Priorities Registers
External	GICC_BPR	CPU Interface Binary Point Register
External	GICC_CTLR	CPU Interface Control Register
External	GICC_DIR	CPU Interface Deactivate Interrupt Register
External	GICC_EOIR	CPU Interface End Of Interrupt Register
External	GICC_HPPIR	CPU Interface Highest Priority Pending Interrupt Register
External	GICC_IAR	CPU Interface Interrupt Acknowledge Register
External	GICC_IIDR	CPU Interface Identification Register
External	GICC_NSAPR<n>	CPU Interface Non-secure Active Priorities Registers
External	GICC_PMR	CPU Interface Priority Mask Register
External	GICC_RPR	CPU Interface Running Priority Register
External	GICC_STATUSR	CPU Interface Status Register

In the GICV functional group:

Exec state	Name	Description
External	GICV_ABPR	Virtual Machine Aliased Binary Point Register
External	GICV_AEOIR	Virtual Machine Aliased End Of Interrupt Register
External	GICV_AHPPIR	Virtual Machine Aliased Highest Priority Pending Interrupt Register
External	GICV_AIAR	Virtual Machine Aliased Interrupt Acknowledge Register
External	GICV_APR<n>	Virtual Machine Active Priorities Registers
External	GICV_BPR	Virtual Machine Binary Point Register
External	GICV_CTLR	Virtual Machine Control Register
External	GICV_DIR	Virtual Machine Deactivate Interrupt Register
External	GICV_EOIR	Virtual Machine End Of Interrupt Register
External	GICV_HPPIR	Virtual Machine Highest Priority Pending Interrupt Register
External	GICV_IAR	Virtual Machine Interrupt Acknowledge Register
External	GICV_IIDR	Virtual Machine CPU Interface Identification Register
External	GICV_PMR	Virtual Machine Priority Mask Register
External	GICV_RPR	Virtual Machine Running Priority Register
External	GICV_STATUSR	Virtual Machine Error Reporting Status Register

In the GICH functional group:

Exec state	Name	Description
External	GICH_APR<n>	Active Priorities Registers
External	GICH_EISR	End Interrupt Status Register
External	GICH_ELRSR	Empty List Register Status Register
External	GICH_HCR	Hypervisor Control Register
External	GICH_LR<n>	List Registers
External	GICH_MISR	Maintenance Interrupt Status Register
External	GICH_VMCR	Virtual Machine Control Register
External	GICH_VTR	Virtual Type Register

In the GITS functional group:

Exec state	Name	Description
External	GITS_BASER<n>	ITS Translation Table Descriptors
External	GITS_CBASER	ITS Command Queue Descriptor
External	GITS_CREADR	ITS Read Register
External	GITS_CTLR	ITS Control Register
External	GITS_CWRITER	ITS Write Register
External	GITS_IIDR	ITS Identification Register
External	GITS_MPAMIDR	Report maximum PARTID and PMG Register
External	GITS_PARTIDR	Set PARTID and PMG Register
External	GITS_TRANSLATER	ITS Translation Register
External	GITS_TYPER	ITS Type Register

In the RAS functional group:

Exec state	Name	Description
AArch32	DISR	Deferred Interrupt Status Register
AArch32	ERRIDR	Error Record ID Register
AArch32	ERRSELR	Error Record Select Register
AArch32	ERXADDR	Selected Error Record Address Register
AArch32	ERXADDR2	Selected Error Record Address Register 2
AArch32	ERXCTLR	Selected Error Record Control Register
AArch32	ERXCTLR2	Selected Error Record Control Register 2
AArch32	ERXFR	Selected Error Record Feature Register
AArch32	ERXFR2	Selected Error Record Feature Register 2
AArch32	ERXMISC0	Selected Error Record Miscellaneous Register 0
AArch32	ERXMISC1	Selected Error Record Miscellaneous Register 1
AArch32	ERXMISC2	Selected Error Record Miscellaneous Register 2
AArch32	ERXMISC3	Selected Error Record Miscellaneous Register 3
AArch32	ERXMISC4	Selected Error Record Miscellaneous Register 4
AArch32	ERXMISC5	Selected Error Record Miscellaneous Register 5
AArch32	ERXMISC6	Selected Error Record Miscellaneous Register 6
AArch32	ERXMISC7	Selected Error Record Miscellaneous Register 7
AArch32	ERXSTATUS	Selected Error Record Primary Status Register
AArch32	VDFSR	Virtual SError Exception Syndrome Register
AArch32	VDISR	Virtual Deferred Interrupt Status Register
AArch64	DISR_EL1	Deferred Interrupt Status Register
AArch64	ERRIDR_EL1	Error Record ID Register
AArch64	ERRSELR_EL1	Error Record Select Register
AArch64	ERXADDR_EL1	Selected Error Record Address Register
AArch64	ERXCTLR_EL1	Selected Error Record Control Register
AArch64	ERXFR_EL1	Selected Error Record Feature Register
AArch64	ERXMISC0_EL1	Selected Error Record Miscellaneous Register 0
AArch64	ERXMISC1_EL1	Selected Error Record Miscellaneous Register 1
AArch64	ERXMISC2_EL1	Selected Error Record Miscellaneous Register 2
AArch64	ERXMISC3_EL1	Selected Error Record Miscellaneous Register 3
AArch64	ERXPFGCDN_EL1	Selected Pseudo-fault Generation Countdown Register
AArch64	ERXPFGCTL_EL1	Selected Pseudo-fault Generation Control Register
AArch64	ERXPFGF_EL1	Selected Pseudo-fault Generation Feature Register
AArch64	ERXSTATUS_EL1	Selected Error Record Primary Status Register

Exec state	Name	Description
AArch64	VDISR_EL2	Virtual Deferred Interrupt Status Register
AArch64	VSESR_EL2	Virtual SErrorDeferred ExceptionInterrupt SyndromeStatus Register
External	ERR<n>ADDR	Error Record Address Register
External	ERR<n>CTLR	Error Record Control Register
External	ERR<n>FR	Error Record Feature Register
External	ERR<n>MISC0	Error Record Miscellaneous Register 0
External	ERR<n>MISC1	Error Record Miscellaneous Register 1
External	ERR<n>MISC2	Error Record Miscellaneous Register 2
External	ERR<n>MISC3	Error Record Miscellaneous Register 3
External	ERR<n>PFGCDN	Pseudo-fault Generation Countdown Register
External	ERR<n>PFGCTL	Pseudo-fault Generation Control Register
External	ERR<n>PFGF	Pseudo-fault Generation Feature Register
External	ERR<n>STATUS	Error Record Primary Status Register
External	ERRCIDR0	Component Identification Register 0
External	ERRCIDR1	Component Identification Register 1
External	ERRCIDR2	Component Identification Register 2
External	ERRCIDR3	Component Identification Register 3
External	ERRCRICR0	Critical Error Interrupt Configuration Register 0
External	ERRCRICR1	Critical Error Interrupt Configuration Register 1
External	ERRCRICR2	Critical Error Interrupt Configuration Register 2
External	ERRDEVAFF	Device Affinity Register
External	ERRDEVARCH	Device Architecture Register
External	ERRDEVID	Device Configuration Register
External	ERRERICR0	Error Recovery Interrupt Configuration Register 0
External	ERRERICR1	Error Recovery Interrupt Configuration Register 1
External	ERRERICR2	Error Recovery Interrupt Configuration Register 2
External	ERRFHICR0	Fault-Handling Interrupt Configuration Register 0
External	ERRFHICR1	Fault-Handling Interrupt Configuration Register 1
External	ERRFHICR2	Fault-Handling Interrupt Configuration Register 2
External	ERRGSR	Error Group Status Register
External	ERRIIDR	Implementation Identification Register
External	ERRIRQCR<n>	Generic Error Interrupt Configuration Register
External	ERRIRQSR	Error Interrupt Status Register
External	ERRPIDR0	Peripheral Identification Register 0
External	ERRPIDR1	Peripheral Identification Register 1
External	ERRPIDR2	Peripheral Identification Register 2
External	ERRPIDR3	Peripheral Identification Register 3
External	ERRPIDR4	Peripheral Identification Register 4

In the Ptr Auth functional group:

Exec state	Name	Description
AArch64	APDAKeyHi_EL1	Pointer Authentication Key A for Data (bits[127:64])
AArch64	APDAKeyLo_EL1	Pointer Authentication Key A for Data (bits[63:0])
AArch64	APDBKeyHi_EL1	Pointer Authentication Key B for Data (bits[127:64])
AArch64	APDBKeyLo_EL1	Pointer Authentication Key B for Data (bits[63:0])
AArch64	APGAKeyHi_EL1	Pointer Authentication Key A for Code (bits[127:64])
AArch64	APGAKeyLo_EL1	Pointer Authentication Key A for Code (bits[63:0])
AArch64	APIAKeyHi_EL1	Pointer Authentication Key A for Instruction (bits[127:64])
AArch64	APIAKeyLo_EL1	Pointer Authentication Key A for Instruction (bits[63:0])
AArch64	APIBKeyHi_EL1	Pointer Authentication Key B for Instruction (bits[127:64])
AArch64	APIBKeyLo_EL1	Pointer Authentication Key B for Instruction (bits[63:0])

In the Resource monitoring configuration functional group:

Exec state	Name	Description
External	MSMON_CAPT_EVNT	MPAM Capture Event Generation Register
External	MSMON_CFG_CSU_CTL	MPAM Memory System Monitor Configure Cache Storage Usage Monitor Control Register
External	MSMON_CFG_CSU_FLT	MPAM Memory System Monitor Configure Cache Storage Usage Monitor Filter Register
External	MSMON_CFG_MBWU_CTL	MPAM Memory System Monitor Configure Memory Bandwidth Usage Monitor Control Register

Exec state	Name	Description
External	MSMON_CFG_MBWU_FLT	MPAM Memory System Monitor Configure Memory Bandwidth Usage Monitor Filter Register
External	MSMON_CFG_MON_SEL	MPAM Partion Configuration Selection Register
External	MSMON_CSU	MPAM Cache Storage Usage Monitor Register
External	MSMON_CSU_CAPTURE	MPAM Cache Storage Usage Monitor Capture Register
External	MSMON_MBWU	MPAM Memory Bandwdith Usage Monitor Register
External	MSMON_MBWU_CAPTURE	MPAM Memory Bandwidth Usage Monitor Capture Register

(old) htmldiff from- (new)

External System registers

AMCFGR: Activity Monitors Configuration Register

AMCGCR: Activity Monitors Counter Group Configuration Register

AMCIDR0: Activity Monitors Component Identification Register 0

AMCIDR1: Activity Monitors Component Identification Register 1

AMCIDR2: Activity Monitors Component Identification Register 2

AMCIDR3: Activity Monitors Component Identification Register 3

AMCNTENCLR0: Activity Monitors Count Enable Clear Register 0

AMCNTENCLR1: Activity Monitors Count Enable Clear Register 1

AMCNTENSET0: Activity Monitors Count Enable Set Register 0

AMCNTENSET1: Activity Monitors Count Enable Set Register 1

AMCR: Activity Monitors Control Register

AMDEVAFF0: Activity Monitors Device Affinity Register 0

AMDEVAFF1: Activity Monitors Device Affinity Register 1

AMDEVARCH: Activity Monitors Device Architecture Register

AMDEVTYPE: Activity Monitors Device Type Register

AMEVCNTR0<n>: Activity Monitors Event Counter Registers 0

AMEVCNTR1<n>: Activity Monitors Event Counter Registers 1

AMEVTYPER0<n>: Activity Monitors Event Type Registers 0

AMEVTYPER1<n>: Activity Monitors Event Type Registers 1

AMIIDR: Activity Monitors Implementation Identification Register

AMPIDR0: Activity Monitors Peripheral Identification Register 0

AMPIDR1: Activity Monitors Peripheral Identification Register 1

AMPIDR2: Activity Monitors Peripheral Identification Register 2

AMPIDR3: Activity Monitors Peripheral Identification Register 3

AMPIDR4: Activity Monitors Peripheral Identification Register 4

ASICCTL: CTI External Multiplexer Control register

CNTACR<n>: Counter-timer Access Control Registers

CNTCR: Counter Control Register

CNTCV: Counter Count Value register

CNTEL0ACR: Counter-timer EL0 Access Control Register

CNTFID0: Counter Frequency ID

CNTFID<n>: Counter Frequency IDs, n > 0

CNTFRQ: Counter-timer Frequency

CNTID: Counter Identification Register

CNTNSAR: Counter-timer Non-secure Access Register

CNTPCT: Counter-timer Physical Count

CNTP_CTL: Counter-timer Physical Timer Control

CNTP_CVAL: Counter-timer Physical Timer CompareValue

CNTP_TVAL: Counter-timer Physical Timer TimerValue

CNTSCR: Counter Scale Register

CNTSR: Counter Status Register

CNTTIDR: Counter-timer Timer ID Register

CNTVCT: Counter-timer Virtual Count

CNTVOFF: Counter-timer Virtual Offset

CNTVOFF<n>: Counter-timer Virtual Offsets

CNTV_CTL: Counter-timer Virtual Timer Control

CNTV_CVAL: Counter-timer Virtual Timer CompareValue

CNTV_TVAL: Counter-timer Virtual Timer TimerValue

CTIAPPCLEAR: CTI Application Trigger Clear register

CTIAPPPULSE: CTI Application Pulse register

CTIAPPSET: CTI Application Trigger Set register

CTIAUTHSTATUS: CTI Authentication Status register

CTICHINSTATUS: CTI Channel In Status register

CTICHOUTSTATUS: CTI Channel Out Status register

CTICIDR0: CTI Component Identification Register 0

CTICIDR1: CTI Component Identification Register 1

CTICIDR2: CTI Component Identification Register 2

CTICIDR3: CTI Component Identification Register 3

CTICLAIMCLR: CTI Claim Tag Clear register

CTICLAIMSET: CTI Claim Tag Set register

CTICONTROL: CTI Control register

CTIDEVAFF0: CTI Device Affinity register 0

CTIDEVAFF1: CTI Device Affinity register 1

CTIDEVARCH: CTI Device Architecture register

CTIDEVCTL: CTI Device Control register

CTIDEVID: CTI Device ID register 0

CTIDEVID1: CTI Device ID register 1

CTIDEVID2: CTI Device ID register 2

CTIDEVTYPE: CTI Device Type register

CTIGATE: CTI Channel Gate Enable register

CTIINEN<n>: CTI Input Trigger to Output Channel Enable registers

CTIINTACK: CTI Output Trigger Acknowledge register

CTIITCTRL: CTI Integration mode Control register

CTILAR: CTI Lock Access Register

CTILSR: CTI Lock Status Register

CTIOUTEN<n>: CTI Input Channel to Output Trigger Enable registers

CTIPIDR0: CTI Peripheral Identification Register 0

CTIPIDR1: CTI Peripheral Identification Register 1

CTIPIDR2: CTI Peripheral Identification Register 2

CTIPIDR3: CTI Peripheral Identification Register 3

CTIPIDR4: CTI Peripheral Identification Register 4

CTITRIGINSTATUS: CTI Trigger In Status register

CTITRIGOUTSTATUS: CTI Trigger Out Status register

CounterID<n>: Counter ID registers

DBGAUTHSTATUS_EL1: Debug Authentication Status register

DBGBCR<n>_EL1: Debug Breakpoint Control Registers

DBGBVR<n>_EL1: Debug Breakpoint Value Registers

DBGCLAIMCLR_EL1: Debug Claim Tag Clear register

DBGCLAIMSET_EL1: Debug Claim Tag Set register

DBGDTRRX_EL0: Debug Data Transfer Register, Receive

DBGDTRTX_EL0: Debug Data Transfer Register, Transmit

DBGWCR<n>_EL1: Debug Watchpoint Control Registers

DBGWVR<n>_EL1: Debug Watchpoint Value Registers

EDAA32PFR: External Debug AArch32 Processor Feature Register

EDACR: External Debug Auxiliary Control Register

EDCIDR0: External Debug Component Identification Register 0

EDCIDR1: External Debug Component Identification Register 1

EDCIDR2: External Debug Component Identification Register 2

EDCIDR3: External Debug Component Identification Register 3

EDCIDSR: External Debug Context ID Sample Register

EDDEVAFF0: External Debug Device Affinity register 0

EDDEVAFF1: External Debug Device Affinity register 1

EDDEVARCH: External Debug Device Architecture register

EDDEVID: External Debug Device ID register 0

EDDEVID1: External Debug Device ID register 1

EDDEVID2: External Debug Device ID register 2

EDDEVTYPE: External Debug Device Type register

EDDFR: External Debug Feature Register

EDECCR: External Debug Exception Catch Control Register

EDECR: External Debug Execution Control Register

EDESR: External Debug Event Status Register

EDITCTRL: External Debug Integration mode Control register

EDITR: External Debug Instruction Transfer Register

EDLAR: External Debug Lock Access Register

EDLSR: External Debug Lock Status Register

EDPCSR: External Debug Program Counter Sample Register

EDPFR: External Debug Processor Feature Register

EDPIDR0: External Debug Peripheral Identification Register 0

EDPIDR1: External Debug Peripheral Identification Register 1

EDPIDR2: External Debug Peripheral Identification Register 2

EDPIDR3: External Debug Peripheral Identification Register 3

EDPIDR4: External Debug Peripheral Identification Register 4

EDPRCR: External Debug Power/Reset Control Register

EDPRSR: External Debug Processor Status Register

EDRCR: External Debug Reserve Control Register

EDSCR: External Debug Status and Control Register

EDVIDSR: External Debug Virtual Context Sample Register

EDWAR: External Debug Watchpoint Address Register

ERR<n>ADDR: Error Record Address Register

ERR<n>CTLR: Error Record Control Register

ERR<n>FR: Error Record Feature Register

ERR<n>MISC0: Error Record Miscellaneous Register 0

ERR<n>MISC1: Error Record Miscellaneous Register 1

ERR<n>MISC2: Error Record Miscellaneous Register 2

ERR<n>MISC3: Error Record Miscellaneous Register 3

ERR<n>PFGCDN: Pseudo-fault Generation Countdown Register

ERR<n>PFGCTL: Pseudo-fault Generation Control Register

ERR<n>PFGF: Pseudo-fault Generation Feature Register

ERR<n>STATUS: Error Record Primary Status Register

ERRCIDR0: Component Identification Register 0

ERRCIDR1: Component Identification Register 1

ERRCIDR2: Component Identification Register 2

ERRCIDR3: Component Identification Register 3

ERRCRICR0: Critical Error Interrupt Configuration Register 0

ERRCRICR1: Critical Error Interrupt Configuration Register 1

ERRCRICR2: Critical Error Interrupt Configuration Register 2

ERRDEVAFF: Device Affinity Register

ERRDEVARCH: Device Architecture Register

ERRDEVID: Device Configuration Register

ERRERICR0: Error Recovery Interrupt Configuration Register 0

ERRERICR1: Error Recovery Interrupt Configuration Register 1

ERRERICR2: Error Recovery Interrupt Configuration Register 2

ERRFHICR0: Fault-Handling Interrupt Configuration Register 0

ERRFHICR1: Fault-Handling Interrupt Configuration Register 1

ERRFHICR2: Fault-Handling Interrupt Configuration Register 2

ERRGSR: Error Group Status Register

ERRIIDR: Implementation Identification Register

ERRIRQCR<n>: Generic Error Interrupt Configuration Register

ERRIRQSR: Error Interrupt Status Register

ERRPIDR0: Peripheral Identification Register 0

ERRPIDR1: Peripheral Identification Register 1

ERRPIDR2: Peripheral Identification Register 2

ERRPIDR3: Peripheral Identification Register 3

ERRPIDR4: Peripheral Identification Register 4

GICC_ABPR: CPU Interface Aliased Binary Point Register

GICC_AEOIR: CPU Interface Aliased End Of Interrupt Register

GICC_AHPPIR: CPU Interface Aliased Highest Priority Pending Interrupt Register

GICC_AIAR: CPU Interface Aliased Interrupt Acknowledge Register

GICC_APR<n>: CPU Interface Active Priorities Registers

GICC_BPR: CPU Interface Binary Point Register

GICC_CTLR: CPU Interface Control Register

GICC_DIR: CPU Interface Deactivate Interrupt Register

GICC_EOIR: CPU Interface End Of Interrupt Register

GICC_HPPIR: CPU Interface Highest Priority Pending Interrupt Register

GICC_IAR: CPU Interface Interrupt Acknowledge Register

GICC_IIDR: CPU Interface Identification Register

GICC_NSAPR<n>: CPU Interface Non-secure Active Priorities Registers

GICC_PMR: CPU Interface Priority Mask Register

GICC_RPR: CPU Interface Running Priority Register

GICC_STATUSR: CPU Interface Status Register

GICD_CLRSPI_NSR: Clear Non-secure SPI Pending Register

GICD_CLRSPI_SR: Clear Secure SPI Pending Register

GICD_CPENDSGIR<n>: SGI Clear-Pending Registers

GICD_CTLR: Distributor Control Register

GICD_ICACTIVER<n>: Interrupt Clear-Active Registers

GICD_ICACTIVER<n>E: Interrupt Clear-Active Registers (extended SPI range)

GICD_ICENABLER<n>: Interrupt Clear-Enable Registers

GICD_ICENABLER<n>E: Interrupt Clear-Enable Registers

GICD_ICFGR<n>: Interrupt Configuration Registers

GICD_ICFGR<n>E: Interrupt Configuration Registers (Extended SPI Range)

GICD_ICPENDR<n>: Interrupt Clear-Pending Registers

GICD_ICPENDR<n>E: Interrupt Clear-Pending Registers (extended SPI range)

GICD_IGROUPR<n>: Interrupt Group Registers

GICD_IGROUPR<n>E: Interrupt Group Registers (extended SPI range)

GICD_IGRPMODR<n>: Interrupt Group Modifier Registers

GICD_IGRPMODR<n>E: Interrupt Group Modifier Registers (extended SPI range)

GICD_IIDR: Distributor Implementer Identification Register

GICD_IPRIORITYR<n>: Interrupt Priority Registers

GICD_IPRIORITYR<n>E: Holds the priority of the corresponding interrupt for each extended SPI supported by the GIC.

GICD_IROUTER<n>: Interrupt Routing Registers

GICD_IROUTER<n>E: Interrupt Routing Registers (Extended SPI Range)

GICD_ISACTIVER<n>: Interrupt Set-Active Registers

GICD_ISACTIVER<n>E: Interrupt Set-Active Registers (extended SPI range)

GICD_ISENABLER<n>: Interrupt Set-Enable Registers

GICD_ISENABLER<n>E: Interrupt Set-Enable Registers

GICD_ISPENDR<n>: Interrupt Set-Pending Registers

GICD_ISPENDR<n>E: Interrupt Set-Pending Registers (extended SPI range)

GICD_ITARGETSR<n>: Interrupt Processor Targets Registers

GICD_NSACR<n>: Non-secure Access Control Registers

GICD_NSACR<n>E: Non-secure Access Control Registers

GICD_SETSPI_NSR: Set Non-secure SPI Pending Register

GICD_SETSPI_SR: Set Secure SPI Pending Register

GICD_SGIR: Software Generated Interrupt Register

GICD_SPENDSGIR<n>: SGI Set-Pending Registers

GICD_STATUSR: Error Reporting Status Register

GICD_TYPER: Interrupt Controller Type Register

GICH_APR<n>: Active Priorities Registers

GICH_EISR: End Interrupt Status Register

GICH_ELRSR: Empty List Register Status Register

GICH_HCR: Hypervisor Control Register

GICH_LR<n>: List Registers

GICH_MISR: Maintenance Interrupt Status Register

GICH_VMCR: Virtual Machine Control Register

GICH_VTR: Virtual Type Register

GICR_CLRLPIR: Clear LPI Pending Register

GICR_CTLR: Redistributor Control Register

GICR_ICACTIVER0: Interrupt Clear-Active Register 0

GICR_ICACTIVER<n>E: Interrupt Clear-Active Registers

GICR_ICENABLER0: Interrupt Clear-Enable Register 0

GICR_ICENABLER<n>E: Interrupt Clear-Enable Registers

GICR_ICFGR0: Interrupt Configuration Register 0

GICR_ICFGR1: Interrupt Configuration Register 1

GICR_ICFGR<n>E: Interrupt configuration registers

GICR_ICPENDR0: Interrupt Clear-Pending Register 0

GICR_ICPENDR<n>E: Interrupt Clear-Pending Registers

GICR_IGROUPR0: Interrupt Group Register 0

GICR_IGROUPR<n>E: Interrupt Group Registers

GICR_IGRPMODR0: Interrupt Group Modifier Register 0

GICR_IGRPMODR<n>E: Interrupt Group Modifier Registers

GICR_IIDR: Redistributor Implementer Identification Register

GICR_INVALLR: Redistributor Invalidate All Register

GICR_INVLPIR: Redistributor Invalidate LPI Register

GICR_IPRIORITYR<n>: Interrupt Priority Registers

GICR_IPRIORITYR<n>E: Interrupt Priority Registers (extended PPI range)

GICR_ISACTIVER0: Interrupt Set-Active Register 0

GICR_ISACTIVER<n>E: Interrupt Set-Active Registers

GICR_ISENABLER0: Interrupt Set-Enable Register 0

GICR_ISENABLER<n>E: Interrupt Set-Enable Registers

GICR_ISPENDR0: Interrupt Set-Pending Register 0

GICR_ISPENDR<n>E: Interrupt Set-Pending Registers

GICR_MPAMIDR: Report maximum PARTID and PMG Register

GICR_NSACR: Non-secure Access Control Register

GICR_PARTIDR: Set PARTID and PMG Register

GICR_PENDBASER: Redistributor LPI Pending Table Base Address Register

GICR_PROPBASER: Redistributor Properties Base Address Register

GICR_SETLPIR: Set LPI Pending Register

GICR_STATUSR: Error Reporting Status Register

GICR_SYNCR: Redistributor Synchronize Register

GICR_TYPER: Redistributor Type Register

GICR_VPENDBASER: Virtual Redistributor LPI Pending Table Base Address Register

GICR_VPROPBASER: Virtual Redistributor Properties Base Address Register

GICR_WAKER: Redistributor Wake Register

GICV_ABPR: Virtual Machine Aliased Binary Point Register

GICV_AEOIR: Virtual Machine Aliased End Of Interrupt Register

GICV_AHPPIR: Virtual Machine Aliased Highest Priority Pending Interrupt Register

GICV_AIAR: Virtual Machine Aliased Interrupt Acknowledge Register

GICV_APR<n>: Virtual Machine Active Priorities Registers

GICV_BPR: Virtual Machine Binary Point Register

GICV_CTLR: Virtual Machine Control Register

GICV_DIR: Virtual Machine Deactivate Interrupt Register

GICV_EOIR: Virtual Machine End Of Interrupt Register

GICV_HPPIR: Virtual Machine Highest Priority Pending Interrupt Register

GICV_IAR: Virtual Machine Interrupt Acknowledge Register

GICV_IIDR: Virtual Machine CPU Interface Identification Register

GICV_PMR: Virtual Machine Priority Mask Register

GICV_RPR: Virtual Machine Running Priority Register

GICV_STATUSR: Virtual Machine Error Reporting Status Register

GITS_BASER<n>: ITS Translation Table Descriptors

GITS_CBASER: ITS Command Queue Descriptor

GITS_CREADR: ITS Read Register

GITS_CTLR: ITS Control Register

GITS_CWRITER: ITS Write Register

GITS_IIDR: ITS Identification Register

GITS_MPAMIDR: Report maximum PARTID and PMG Register

GITS_PARTIDR: Set PARTID and PMG Register

GITS_TRANSLATER: ITS Translation Register

GITS_TYPER: ITS Type Register

MIDR_EL1: Main ID Register

MPAMCFG_CMAX: MPAM Cache Maximum Capacity Partition Configuration Register

MPAMCFG_CPBM: MPAM Cache Portion Bitmap Partition Configuration Register

MPAMCFG_INTPARTID: MPAM Internal PARTID Narrowing Configuration Register

MPAMCFG_MBW_MAX: MPAM Memory Bandwidth Maximum Partition Configuration Register

MPAMCFG_MBW_MIN: MPAM Cache Maximum Capacity Partition Configuration Register

MPAMCFG_MBW_PBM: MPAM Bandwidth Portion Bitmap Parition Configuration Register

MPAMCFG_MBW_PROP: MPAM Memory Bandwidth Proportional Stride Partition Configuration Register

MPAMCFG_MBW_WINWD: MPAM Memory Bandwidth Partitioning Window Width Configuration Register

MPAMCFG_PART_SEL: MPAM Partion Configuration Selection Register

MPAMCFG_PRI: MPAM Priority Partition Configuration Register

MPAMF_AIDR: MPAM Architecture Identification Register

MPAMF_CCAP_IDR: MPAM Features Cache Capacity Partitioning ID register

MPAMF_CPOR_IDR: MPAM Features Cache Portion Partitioning ID register

MPAMF_CSUMON_IDR: MPAM Features Cache Storage Usage Monitoring ID register

MPAMF_ECR: MPAM Error Control Register

MPAMF_ESR: MPAM Error Status Register

MPAMF_IDR: MPAM Features Identification Register

MPAMF_IIDR: MPAM Implemenation Identification Register

MPAMF_IMPL_IDR: MPAM Implementation-Specific Partitioning Feature Identification Register

MPAMF_MBWUMON_IDR: MPAM Features Memory Bandwidth Usage Monitoring ID register

MPAMF_MBW_IDR: MPAM Memory Bandwidth Partitioning Identification Register

MPAMF_MSMON_IDR: MPAM Resource Monitoring Identification Register

MPAMF_PARTID_NRW_IDR: MPAM PARTID Narrowing ID register

MPAMF_PRI_IDR: MPAM Priority Partitioning Identification Register

MPAMF_SIDR: MPAM Features Secure Identification Register

MSMON_CAPT_EVNT: MPAM Capture Event Generation Register

MSMON_CFG_CSU_CTL: MPAM Memory System Monitor Configure Cache Storage Usage Monitor Control Register

MSMON_CFG_CSU_FLT: MPAM Memory System Monitor Configure Cache Storage Usage Monitor Filter Register

MSMON_CFG_MBWU_CTL: MPAM Memory System Monitor Configure Memory Bandwidth Usage Monitor Control Register

MSMON_CFG_MBWU_FLT: MPAM Memory System Monitor Configure Memory Bandwidth Usage Monitor Filter Register

MSMON_CFG_MON_SEL: MPAM Partion Configuration Selection Register

MSMON_CSU: MPAM Cache Storage Usage Monitor Register

MSMON_CSU_CAPTURE: MPAM Cache Storage Usage Monitor Capture Register

MSMON_MBWU: MPAM Memory Bandwdith Usage Monitor Register

MSMON_MBWU_CAPTURE: MPAM Memory Bandwidth Usage Monitor Capture Register

OSLAR_EL1: OS Lock Access Register

PMAUTHSTATUS: Performance Monitors Authentication Status register

PMCCFILTR_EL0: Performance Monitors Cycle Counter Filter Register

PMCCNTR_EL0: Performance Monitors Cycle Counter

PMCEID0: Performance Monitors Common Event Identification register 0

PMCEID1: Performance Monitors Common Event Identification register 1

PMCEID2: Performance Monitors Common Event Identification register 2

PMCEID3: Performance Monitors Common Event Identification register 3

PMCFGR: Performance Monitors Configuration Register

PMCID1SR: CONTEXTIDR_EL1 Sample Register

PMCID2SR: CONTEXTIDR_EL2 Sample Register

PMCIDR0: Performance Monitors Component Identification Register 0

PMCIDR1: Performance Monitors Component Identification Register 1

PMCIDR2: Performance Monitors Component Identification Register 2

PMCIDR3: Performance Monitors Component Identification Register 3

PMCNTENCLR_EL0: Performance Monitors Count Enable Clear register

PMCNTENSET_EL0: Performance Monitors Count Enable Set register

PMCR_EL0: Performance Monitors Control Register

PMDEVAFF0: Performance Monitors Device Affinity register 0

PMDEVAFF1: Performance Monitors Device Affinity register 1

PMDEVARCH: Performance Monitors Device Architecture register

PMDEVID: Performance Monitors Device ID register

PMDEVTYPE: Performance Monitors Device Type register

PMEVCNTR<n>_EL0: Performance Monitors Event Count Registers

PMEVTYPER<n>_EL0: Performance Monitors Event Type Registers

PMINTENCLR_EL1: Performance Monitors Interrupt Enable Clear register

PMINTENSET_EL1: Performance Monitors Interrupt Enable Set register

PMITCTRL: Performance Monitors Integration mode Control register

PMLAR: Performance Monitors Lock Access Register

PMLSR: Performance Monitors Lock Status Register

PMMIR: Performance Monitors Machine Identification Register

PMOVSCLR_EL0: Performance Monitors Overflow Flag Status Clear register

PMOVSSET_EL0: Performance Monitors Overflow Flag Status Set register

PMPCSR: Program Counter Sample Register

PMPIDR0: Performance Monitors Peripheral Identification Register 0

PMPIDR1: Performance Monitors Peripheral Identification Register 1

PMPIDR2: Performance Monitors Peripheral Identification Register 2

PMPIDR3: Performance Monitors Peripheral Identification Register 3

PMPIDR4: Performance Monitors Peripheral Identification Register 4

PMSWINC_EL0: Performance Monitors Software Increment register

PMVIDSR: VMID Sample Register

(old) htmldiff from- (new)

External register index by offset

Below are indexes for external registers in the following blocks:

In the AMU block:

Offset	Name	Description
0x000 + 8n	AMEVCNTR0<n>[31:0]	Activity Monitors Event Counter Registers 0
0x004 + 8n	AMEVCNTR0<n>[63:32]	Activity Monitors Event Counter Registers 0
0x100 + 8n	AMEVCNTR1<n>[31:0]	Activity Monitors Event Counter Registers 1
0x104 + 8n	AMEVCNTR1<n>[63:32]	Activity Monitors Event Counter Registers 1
0x400 + 4n	AMEVTYPER0<n>	Activity Monitors Event Type Registers 0
0x480 + 4n	AMEVTYPER1<n>	Activity Monitors Event Type Registers 1
0xC00	AMCNTENSET0	Activity Monitors Count Enable Set Register 0
0xC04	AMCNTENSET1	Activity Monitors Count Enable Set Register 1
0xC20	AMCNTENCLR0	Activity Monitors Count Enable Clear Register 0
0xC24	AMCNTENCLR1	Activity Monitors Count Enable Clear Register 1
0xCE0	AMCGCR	Activity Monitors Counter Group Configuration Register
0xE00	AMCFGR	Activity Monitors Configuration Register
0xE04	AMCR	Activity Monitors Control Register
0xE08	AMIIDR	Activity Monitors Implementation Identification Register
0xFA8	AMDEVAFF0	Activity Monitors Device Affinity Register 0
0xFAC	AMDEVAFF1	Activity Monitors Device Affinity Register 1
0xFBC	AMDEVARCH	Activity Monitors Device Architecture Register
0xFCC	AMDEVTYPE	Activity Monitors Device Type Register
0xFD0	AMPIDR4	Activity Monitors Peripheral Identification Register 4
0xFE0	AMPIDR0	Activity Monitors Peripheral Identification Register 0
0xFE4	AMPIDR1	Activity Monitors Peripheral Identification Register 1
0xFE8	AMPIDR2	Activity Monitors Peripheral Identification Register 2
0xFEC	AMPIDR3	Activity Monitors Peripheral Identification Register 3
0xFF0	AMCIDR0	Activity Monitors Component Identification Register 0
0xFF4	AMCIDR1	Activity Monitors Component Identification Register 1
0xFF8	AMCIDR2	Activity Monitors Component Identification Register 2
0xFFC	AMCIDR3	Activity Monitors Component Identification Register 3

In the CTI block:

Offset	Name	Description
0x000	CTICONTROL	CTI Control register
0x010	CTIINTACK	CTI Output Trigger Acknowledge register
0x014	CTIAPPSET	CTI Application Trigger Set register
0x018	CTIAPPCLEAR	CTI Application Trigger Clear register
0x01C	CTIAPPPULSE	CTI Application Pulse register
0x020 + 4n	CTIINEN<n>	CTI Input Trigger to Output Channel Enable registers
0x0A0 + 4n	CTIOUTEN<n>	CTI Input Channel to Output Trigger Enable registers
0x130	CTITRIGINSTATUS	CTI Trigger In Status register
0x134	CTITRIGOUTSTATUS	CTI Trigger Out Status register
0x138	CTICHINSTATUS	CTI Channel In Status register
0x13C	CTICHOUTSTATUS	CTI Channel Out Status register
0x140	CTIGATE	CTI Channel Gate Enable register
0x144	ASICCTL	CTI External Multiplexer Control register
0x150	CTIDEVCTL	CTI Device Control register
0xF00	CTIITCTRL	CTI Integration mode Control register
0xFA0	CTICLAIMSET	CTI Claim Tag Set register
0xFA4	CTICLAIMCLR	CTI Claim Tag Clear register
0xFA8	CTIDEVAFF0	CTI Device Affinity register 0
0xFAC	CTIDEVAFF1	CTI Device Affinity register 1
0xFB0	CTILAR	CTI Lock Access Register
0xFB4	CTILSR	CTI Lock Status Register
0xFB8	CTIAUTHSTATUS	CTI Authentication Status register
0xFBC	CTIDEVARCH	CTI Device Architecture register
0xFC0	CTIDEVID2	CTI Device ID register 2
0xFC4	CTIDEVID1	CTI Device ID register 1
0xFC8	CTIDEVID	CTI Device ID register 0
0xFCC	CTIDEVTYPE	CTI Device Type register
0xFD0	CTIPIDR4	CTI Peripheral Identification Register 4
0xFE0	CTIPIDR0	CTI Peripheral Identification Register 0
0xFE4	CTIPIDR1	CTI Peripheral Identification Register 1
0xFE8	CTIPIDR2	CTI Peripheral Identification Register 2
0xFEC	CTIPIDR3	CTI Peripheral Identification Register 3
0xFF0	CTICIDR0	CTI Component Identification Register 0
0xFF4	CTICIDR1	CTI Component Identification Register 1
0xFF8	CTICIDR2	CTI Component Identification Register 2
0xFFC	CTICIDR3	CTI Component Identification Register 3

In the Debug block:

Offset	Name	Description
0x020	EDESR	External Debug Event Status Register
0x024	EDECR	External Debug Execution Control Register
0x030	EDWAR[31:0]	External Debug Watchpoint Address Register
0x034	EDWAR[63:32]	External Debug Watchpoint Address Register
0x080	DBGDTRRX_EL0	Debug Data Transfer Register, Receive
0x084	EDITR	External Debug Instruction Transfer Register
0x088	EDSCR	External Debug Status and Control Register

Offset	Name	Description
0x08C	DBGDTRTX_EL0	Debug Data Transfer Register, Transmit
0x090	EDRCR	External Debug Reserve Control Register
0x094	EDACR	External Debug Auxiliary Control Register
0x098	EDECCR	External Debug Exception Catch Control Register
0x0A0	EDPCSR[31:0]	External Debug Program Counter Sample Register
0x0A4	EDCIDSR	External Debug Context ID Sample Register
0x0A8	EDVIDSR	External Debug Virtual Context Sample Register
0x0AC	EDPCSR[63:32]	External Debug Program Counter Sample Register
0x300	OSLAR_EL1	OS Lock Access Register
0x310	EDPRCR	External Debug Power/Reset Control Register
0x314	EDPRSR	External Debug Processor Status Register
0x400 + 16n	DBGBVR<n>_EL1[63:0]	Debug Breakpoint Value Registers
0x408 + 16n	DBGBCR<n>_EL1	Debug Breakpoint Control Registers
0x800 + 16n	DBGWVR<n>_EL1[63:0]	Debug Watchpoint Value Registers
0x808 + 16n	DBGWCR<n>_EL1	Debug Watchpoint Control Registers
0xD00	MIDR_EL1	Main ID Register
0xD20	EDPFR[31:0]	External Debug Processor Feature Register
0xD24	EDPFR[63:32]	External Debug Processor Feature Register
0xD28	EDDFR[31:0]	External Debug Feature Register
0xD2C	EDDFR[63:32]	External Debug Feature Register
0xD60	EDAA32PFR	External Debug AArch32 Processor Feature Register
0xF00	EDITCTRL	External Debug Integration mode Control register
0xFA0	DBGCLAIMSET_EL1	Debug Claim Tag Set register
0xFA4	DBGCLAIMCLR_EL1	Debug Claim Tag Clear register
0xFA8	EDDEVAFF0	External Debug Device Affinity register 0
0xFAC	EDDEVAFF1	External Debug Device Affinity register 1
0xFB0	EDLAR	External Debug Lock Access Register
0xFB4	EDLSR	External Debug Lock Status Register
0xFB8	DBGAUTHSTATUS_EL1	Debug Authentication Status register
0xFBC	EDDEVARCH	External Debug Device Architecture register
0xFC0	EDDEVID2	External Debug Device ID register 2
0xFC4	EDDEVID1	External Debug Device ID register 1
0xFC8	EDDEVID	External Debug Device ID register 0
0xFCC	EDDEVTYPE	External Debug Device Type register
0xFD0	EDPIDR4	External Debug Peripheral Identification Register 4
0xFE0	EDPIDR0	External Debug Peripheral Identification Register 0
0xFE4	EDPIDR1	External Debug Peripheral Identification Register 1
0xFE8	EDPIDR2	External Debug Peripheral Identification Register 2
0xFEC	EDPIDR3	External Debug Peripheral Identification Register 3
0xFF0	EDCIDR0	External Debug Component Identification Register 0
0xFF4	EDCIDR1	External Debug Component Identification Register 1
0xFF8	EDCIDR2	External Debug Component Identification Register 2
0xFFC	EDCIDR3	External Debug Component Identification Register 3

In the GIC CPU interface block:

Offset	Name	Description
0x0000	GICC_CTLR	CPU Interface Control Register
0x0004	GICC_PMR	CPU Interface Priority Mask Register

Offset	Name	Description
0x0008	GICC_BPR	CPU Interface Binary Point Register
0x000C	GICC_IAR	CPU Interface Interrupt Acknowledge Register
0x0010	GICC_EOIR	CPU Interface End Of Interrupt Register
0x0014	GICC_RPR	CPU Interface Running Priority Register
0x0018	GICC_HPPIR	CPU Interface Highest Priority Pending Interrupt Register
0x001C	GICC_ABPR	CPU Interface Aliased Binary Point Register
0x0020	GICC_AIAR	CPU Interface Aliased Interrupt Acknowledge Register
0x0024	GICC_AEOIR	CPU Interface Aliased End Of Interrupt Register
0x0028	GICC_AHPPIR	CPU Interface Aliased Highest Priority Pending Interrupt Register
0x002C	GICC_STATUSR	CPU Interface Status Register
0x002C	GICC_STATUSR	CPU Interface Status Register
0x00D0 + 4n	GICC_APR<n>	CPU Interface Active Priorities Registers
0x00E0 + 4n	GICC_NSAPR<n>	CPU Interface Non-secure Active Priorities Registers
0x00FC	GICC_IIDR	CPU Interface Identification Register
0x1000	GICC_DIR	CPU Interface Deactivate Interrupt Register

In the GIC Distributor block:

Offset	Name	Description
0x0000	GICD_CTLR	Distributor Control Register
0x0004	GICD_TYPER	Interrupt Controller Type Register
0x0008	GICD_IIDR	Distributor Implementer Identification Register
0x0010	GICD_STATUSR	Error Reporting Status Register
0x0010	GICD_STATUSR	Error Reporting Status Register
0x0040	GICD_SETSPI_NSR	Set Non-secure SPI Pending Register
0x0048	GICD_CLRSPI_NSR	Clear Non-secure SPI Pending Register
0x0050	GICD_SETSPI_SR	Set Secure SPI Pending Register
0x0058	GICD_CLRSPI_SR	Clear Secure SPI Pending Register
0x0080 + 4n	GICD_IGROUPR<n>	Interrupt Group Registers
0x0100 + 4n	GICD_ISENABLER<n>	Interrupt Set-Enable Registers
0x0180 + 4n	GICD_ICENABLER<n>	Interrupt Clear-Enable Registers
0x0200 + 4n	GICD_ISPENDR<n>	Interrupt Set-Pending Registers
0x0280 + 4n	GICD_ICPENDR<n>	Interrupt Clear-Pending Registers
0x0300 + 4n	GICD_ISACTIVER<n>	Interrupt Set-Active Registers
0x0380 + 4n	GICD_ICACTIVER<n>	Interrupt Clear-Active Registers
0x0400 + 4n	GICD_IPRIORITYR<n>	Interrupt Priority Registers
0x0800 + 4n	GICD_ITARGETSR<n>	Interrupt Processor Targets Registers
0x0C00 + 4n	GICD_ICFGR<n>	Interrupt Configuration Registers
0x0D00 + 4n	GICD_IGRPMODR<n>	Interrupt Group Modifier Registers
0x0E00 + 4n	GICD_NSACR<n>	Non-secure Access Control Registers
0x0F00	GICD_SGIR	Software Generated Interrupt Register
0x0F10 + 4n	GICD_CPENDSGIR<n>	SGI Clear-Pending Registers
0x0F20 + 4n	GICD_SPENDSGIR<n>	SGI Set-Pending Registers
0x1000 + 4n	GICD_IGROUPR<n>E	Interrupt Group Registers (extended SPI range)
0x1200 + 4n	GICD_ISENABLER<n>E	Interrupt Set-Enable Registers
0x1400 + 4n	GICD_ICENABLER<n>E	Interrupt Clear-Enable Registers
0x1600 + 4n	GICD_ISPENDR<n>E	Interrupt Set-Pending Registers (extended SPI range)
0x1800 + 4n	GICD_ICPENDR<n>E	Interrupt Clear-Pending Registers (extended SPI range)
0x1A00 + 4n	GICD_ISACTIVER<n>E	Interrupt Set-Active Registers (extended SPI range)

Offset	Name	Description
0x1C00 + 4n	GICD_ICACTIVER<n>E	Interrupt Clear-Active Registers (extended SPI range)
0x1E00 + 4n	GICD_ICFGR<n>E	Interrupt Configuration Registers (Extended SPI Range)
0x2000 + 4n	GICD_IPRIORITYR<n>E	Holds the priority of the corresponding interrupt for each extended SPI supported by the GIC.
0x3000 + 4n	GICD_IGRPMODR<n>E	Interrupt Group Modifier Registers (extended SPI range)
0x3600 + 4n	GICD_NSACR<n>E	Non-secure Access Control Registers
0x6000 + 8n	GICD_IROUTER<n>	Interrupt Routing Registers
0x8000 + 8n	GICD_IROUTER<n>E	Interrupt Routing Registers (Extended SPI Range)

In the GIC ITS control block:

Offset	Name	Description
0x0000	GITS_CTLR	ITS Control Register
0x0004	GITS_IIDR	ITS Identification Register
0x0008	GITS_TYPER	ITS Type Register
0x0010	GITS_MPAMIDR	Report maximum PARTID and PMG Register
0x0014	GITS_PARTIDR	Set PARTID and PMG Register
0x0080	GITS_CBASER	ITS Command Queue Descriptor
0x0088	GITS_CWRITER	ITS Write Register
0x0090	GITS_CREADR	ITS Read Register
0x0100 + 8n	GITS_BASER<n>	ITS Translation Table Descriptors

In the GIC ITS translation block:

Offset	Name	Description
0x0040	GITS_TRANSLATER	ITS Translation Register

In the GIC Redistributor block:

Frame	Offset	Name	Description
RD_base	0x0000	GICR_CTLR	Redistributor Control Register
RD_base	0x0004	GICR_IIDR	Redistributor Implementer Identification Register
RD_base	0x0008	GICR_TYPER	Redistributor Type Register
RD_base	0x0010	GICR_STATUSR	Error Reporting Status Register
RD_base	0x0010	GICR_STATUSR	Error Reporting Status Register
RD_base	0x0014	GICR_WAKER	Redistributor Wake Register
RD_base	0x0018	GICR_MPAMIDR	Report maximum PARTID and PMG Register
RD_base	0x001C	GICR_PARTIDR	Set PARTID and PMG Register
RD_base	0x0040	GICR_SETLPIR	Set LPI Pending Register
RD_base	0x0048	GICR_CLRLPIR	Clear LPI Pending Register
RD_base	0x0070	GICR_PROPBASER	Redistributor Properties Base Address Register
RD_base	0x0078	GICR_PENDBASER	Redistributor LPI Pending Table Base Address Register
RD_base	0x00A0	GICR_INVLPIR	Redistributor Invalidate LPI Register
RD_base	0x00B0	GICR_INVALLR	Redistributor Invalidate All Register
RD_base	0x00C0	GICR_SYNCR	Redistributor Synchronize Register
SGI_base	0x0080	GICR_IGROUPR0	Interrupt Group Register 0
SGI_base	0x0080 + 4n	GICR_IGROUPR<n>E	Interrupt Group Registers
SGI_base	0x0100	GICR_ISENABLER0	Interrupt Set-Enable Register 0
SGI_base	0x0100 + 4n	GICR_ISENABLER<n>E	Interrupt Set-Enable Registers
SGI_base	0x0180	GICR_ICENABLER0	Interrupt Clear-Enable Register 0

Frame	Offset	Name	Description
SGI_base	0x0180 + 4n	GICR_ICENABLER<n>E	Interrupt Clear-Enable Registers
SGI_base	0x0200	GICR_ISPENDR0	Interrupt Set-Pending Register 0
SGI_base	0x0200 + 4n	GICR_ISPENDR<n>E	Interrupt Set-Pending Registers
SGI_base	0x0280	GICR_ICPENDR0	Interrupt Clear-Pending Register 0
SGI_base	0x0280 + 4n	GICR_ICPENDR<n>E	Interrupt Clear-Pending Registers
SGI_base	0x0300	GICR_ISACTIVER0	Interrupt Set-Active Register 0
SGI_base	0x0300 + 4n	GICR_ISACTIVER<n>E	Interrupt Set-Active Registers
SGI_base	0x0380	GICR_ICACTIVER0	Interrupt Clear-Active Register 0
SGI_base	0x0380 + 4n	GICR_ICACTIVER<n>E	Interrupt Clear-Active Registers
SGI_base	0x0400 + 4n	GICR_IPRIORITYR<n>	Interrupt Priority Registers
SGI_base	0x0400 + 4n	GICR_IPRIORITYR<n>E	Interrupt Priority Registers (extended PPI range)
SGI_base	0x0C00	GICR_ICFGR0	Interrupt Configuration Register 0
SGI_base	0x0C00 + 4n	GICR_ICFGR<n>E	Interrupt configuration registers
SGI_base	0x0C04	GICR_ICFGR1	Interrupt Configuration Register 1
SGI_base	0x0D00	GICR_IGRPMODR0	Interrupt Group Modifier Register 0
SGI_base	0x0D00 + 4n	GICR_IGRPMODR<n>E	Interrupt Group Modifier Registers
SGI_base	0x0E00	GICR_NSACR	Non-secure Access Control Register
VLPI_base	0x0070	GICR_VPROPBASER	Virtual Redistributor Properties Base Address Register
VLPI_base	0x0078	GICR_VPENDBASER	Virtual Redistributor LPI Pending Table Base Address Register

In the GIC Virtual CPU interface block:

Offset	Name	Description
0x0000	GICV_CTLR	Virtual Machine Control Register
0x0004	GICV_PMR	Virtual Machine Priority Mask Register
0x0008	GICV_BPR	Virtual Machine Binary Point Register
0x000C	GICV_IAR	Virtual Machine Interrupt Acknowledge Register
0x0010	GICV_EOIR	Virtual Machine End Of Interrupt Register
0x0014	GICV_RPR	Virtual Machine Running Priority Register
0x0018	GICV_HPPIR	Virtual Machine Highest Priority Pending Interrupt Register
0x001C	GICV_ABPR	Virtual Machine Aliased Binary Point Register
0x0020	GICV_AIAR	Virtual Machine Aliased Interrupt Acknowledge Register
0x0024	GICV_AEOIR	Virtual Machine Aliased End Of Interrupt Register
0x0028	GICV_AHPPIR	Virtual Machine Aliased Highest Priority Pending Interrupt Register
0x002C	GICV_STATUSR	Virtual Machine Error Reporting Status Register
0x00D0 + 4n	GICV_APR<n>	Virtual Machine Active Priorities Registers
0x00FC	GICV_IIDR	Virtual Machine CPU Interface Identification Register
0x1000	GICV_DIR	Virtual Machine Deactivate Interrupt Register

In the GIC Virtual interface control block:

Offset	Name	Description
0x0000	GICH_HCR	Hypervisor Control Register
0x0004	GICH_VTR	Virtual Type Register
0x0008	GICH_VMCR	Virtual Machine Control Register
0x0010	GICH_MISR	Maintenance Interrupt Status Register
0x0020	GICH_EISR	End Interrupt Status Register
0x0030	GICH_ELRSR	Empty List Register Status Register
0x00F0 + 4n	GICH_APR<n>	Active Priorities Registers

Offset	Name	Description
0x0100 + 4n	GICH_LR<n>	List Registers

In the MPAM.any block:

Frame	Offset	Name	Description
MPAMF_BASE_ns	0x0000	MPAMF_IDR	MPAM Features Identification Register
MPAMF_BASE_ns	0x0018	MPAMF_IIDR	MPAM Implemenation Identification Register
MPAMF_BASE_ns	0x0020	MPAMF_AIDR	MPAM Architecture Identification Register
MPAMF_BASE_ns	0x0028	MPAMF_IMPL_IDR	MPAM Implementation-Specific Partitioning Feature Identification Register
MPAMF_BASE_ns	0x0030	MPAMF_CPOR_IDR	MPAM Features Cache Portion Partitioning ID register
MPAMF_BASE_ns	0x0038	MPAMF_CCAP_IDR	MPAM Features Cache Capacity Partitioning ID register
MPAMF_BASE_ns	0x0040	MPAMF_MBW_IDR	MPAM Memory Bandwidth Partitioning Identification Register
MPAMF_BASE_ns	0x0048	MPAMF_PRI_IDR	MPAM Priority Partitioning Identification Register
MPAMF_BASE_ns	0x0050	MPAMF_PARTID_NRW_IDR	MPAM PARTID Narrowing ID register
MPAMF_BASE_ns	0x0080	MPAMF_MSMON_IDR	MPAM Resource Monitoring Identification Register
MPAMF_BASE_ns	0x0088	MPAMF_CSUMON_IDR	MPAM Features Cache Storage Usage Monitoring ID register
MPAMF_BASE_ns	0x0090	MPAMF_MBWUMON_IDR	MPAM Features Memory Bandwidth Usage Monitoring ID register
MPAMF_BASE_ns	0x00F0	MPAMF_ECR	MPAM Error Control Register
MPAMF_BASE_ns	0x00F8	MPAMF_ESR	MPAM Error Status Register
MPAMF_BASE_ns	0x0100	MPAMCFG_PART_SEL	MPAM Partion Configuration Selection Register
MPAMF_BASE_ns	0x0108	MPAMCFG_CMAX	MPAM Cache Maximum Capacity Partition Configuration Register
MPAMF_BASE_ns	0x0200	MPAMCFG_MBW_MIN	MPAM Cache Maximum Capacity Partition Configuration Register
MPAMF_BASE_ns	0x0208	MPAMCFG_MBW_MAX	MPAM Memory Bandwidth Maximum Partition Configuration Register
MPAMF_BASE_ns	0x0220	MPAMCFG_MBW_WINWD	MPAM Memory Bandwidth Partitioning Window Width Configuration Register
MPAMF_BASE_ns	0x0400	MPAMCFG_PRI	MPAM Priority Partition Configuration Register
MPAMF_BASE_ns	0x0500	MPAMCFG_MBW_PROP	MPAM Memory Bandwidth Proportional Stride Partition Configuration Register
MPAMF_BASE_ns	0x0600	MPAMCFG_INTPARTID	MPAM Internal PARTID Narrowing Configuration Register
MPAMF_BASE_ns	0x0800	MSMON_CFG_MON_SEL	MPAM Partion Configuration Selection Register
MPAMF_BASE_ns	0x0808	MSMON_CAPT_EVNT	MPAM Capture Event Generation Register
MPAMF_BASE_ns	0x0810	MSMON_CFG_CSU_FLT	MPAM Memory System Monitor Configure Cache Storage Usage Monitor Filter Register
MPAMF_BASE_ns	0x0818	MSMON_CFG_CSU_CTL	MPAM Memory System Monitor Configure Cache Storage Usage Monitor Control Register
MPAMF_BASE_ns	0x0820	MSMON_CFG_MBWU_FLT	MPAM Memory System Monitor Configure Memory Bandwidth Usage Monitor Filter Register
MPAMF_BASE_ns	0x0828	MSMON_CFG_MBWU_CTL	MPAM Memory System Monitor Configure Memory Bandwidth Usage Monitor Control Register
MPAMF_BASE_ns	0x0840	MSMON_CSU	MPAM Cache Storage Usage Monitor Register
MPAMF_BASE_ns	0x0848	MSMON_CSU_CAPTURE	MPAM Cache Storage Usage Monitor Capture Register
MPAMF_BASE_ns	0x0860	MSMON_MBWU	MPAM Memory Bandwdith Usage Monitor Register
MPAMF_BASE_ns	0x0868	MSMON_MBWU_CAPTURE	MPAM Memory Bandwidth Usage Monitor Capture Register
MPAMF_BASE_ns	0x1000	MPAMCFG_CPBM	MPAM Cache Portion Bitmap Partition Configuration Register
MPAMF_BASE_ns	0x2000	MPAMCFG_MBW_PBM	MPAM Bandwidth Portion Bitmap Parition Configuration Register
MPAMF_BASE_s	0x0000	MPAMF_IDR	MPAM Features Identification Register
MPAMF_BASE_s	0x0008	MPAMF_SIDR	MPAM Features Secure Identification Register
MPAMF_BASE_s	0x0018	MPAMF_IIDR	MPAM Implemenation Identification Register
MPAMF_BASE_s	0x0020	MPAMF_AIDR	MPAM Architecture Identification Register
MPAMF_BASE_s	0x0028	MPAMF_IMPL_IDR	MPAM Implementation-Specific Partitioning Feature Identification Register

Frame	Offset	Name	Description
MPAMF_BASE_s	0x0030	MPAMF_CPOR_IDR	MPAM Features Cache Portion Partitioning ID register
MPAMF_BASE_s	0x0038	MPAMF_CCAP_IDR	MPAM Features Cache Capacity Partitioning ID register
MPAMF_BASE_s	0x0040	MPAMF_MBW_IDR	MPAM Memory Bandwidth Partitioning Identification Register
MPAMF_BASE_s	0x0048	MPAMF_PRI_IDR	MPAM Priority Partitioning Identification Register
MPAMF_BASE_s	0x0050	MPAMF_PARTID_NRW_IDR	MPAM PARTID Narrowing ID register
MPAMF_BASE_s	0x0080	MPAMF_MSMON_IDR	MPAM Resource Monitoring Identification Register
MPAMF_BASE_s	0x0088	MPAMF_CSUMON_IDR	MPAM Features Cache Storage Usage Monitoring ID register
MPAMF_BASE_s	0x0090	MPAMF_MBWUMON_IDR	MPAM Features Memory Bandwidth Usage Monitoring ID register
MPAMF_BASE_s	0x00F0	MPAMF_ECR	MPAM Error Control Register
MPAMF_BASE_s	0x00F8	MPAMF_ESR	MPAM Error Status Register
MPAMF_BASE_s	0x0100	MPAMCFG_PART_SEL	MPAM Partion Configuration Selection Register
MPAMF_BASE_s	0x0108	MPAMCFG_CMAX	MPAM Cache Maximum Capacity Partition Configuration Register
MPAMF_BASE_s	0x0200	MPAMCFG_MBW_MIN	MPAM Cache Maximum Capacity Partition Configuration Register
MPAMF_BASE_s	0x0208	MPAMCFG_MBW_MAX	MPAM Memory Bandwidth Maximum Partition Configuration Register
MPAMF_BASE_s	0x0220	MPAMCFG_MBW_WINWD	MPAM Memory Bandwidth Partitioning Window Width Configuration Register
MPAMF_BASE_s	0x0400	MPAMCFG_PRI	MPAM Priority Partition Configuration Register
MPAMF_BASE_s	0x0500	MPAMCFG_MBW_PROP	MPAM Memory Bandwidth Proportional Stride Partition Configuration Register
MPAMF_BASE_s	0x0600	MPAMCFG_INTPARTID	MPAM Internal PARTID Narrowing Configuration Register
MPAMF_BASE_s	0x0800	MSMON_CFG_MON_SEL	MPAM Partion Configuration Selection Register
MPAMF_BASE_s	0x0808	MSMON_CAPT_EVNT	MPAM Capture Event Generation Register
MPAMF_BASE_s	0x0810	MSMON_CFG_CSU_FLT	MPAM Memory System Monitor Configure Cache Storage Usage Monitor Filter Register
MPAMF_BASE_s	0x0818	MSMON_CFG_CSU_CTL	MPAM Memory System Monitor Configure Cache Storage Usage Monitor Control Register
MPAMF_BASE_s	0x0820	MSMON_CFG_MBWU_FLT	MPAM Memory System Monitor Configure Memory Bandwidth Usage Monitor Filter Register
MPAMF_BASE_s	0x0828	MSMON_CFG_MBWU_CTL	MPAM Memory System Monitor Configure Memory Bandwidth Usage Monitor Control Register
MPAMF_BASE_s	0x0840	MSMON_CSU	MPAM Cache Storage Usage Monitor Register
MPAMF_BASE_s	0x0848	MSMON_CSU_CAPTURE	MPAM Cache Storage Usage Monitor Capture Register
MPAMF_BASE_s	0x0860	MSMON_MBWU	MPAM Memory Bandwdith Usage Monitor Register
MPAMF_BASE_s	0x0868	MSMON_MBWU_CAPTURE	MPAM Memory Bandwidth Usage Monitor Capture Register
MPAMF_BASE_s	0x1000	MPAMCFG_CPBM	MPAM Cache Portion Bitmap Partition Configuration Register
MPAMF_BASE_s	0x2000	MPAMCFG_MBW_PBM	MPAM Bandwidth Portion Bitmap Parition Configuration Register

In the PMU block:

Offset	Name	Description
0x000 + 8n	PMEVCNTR<n>_EL0	Performance Monitors Event Count Registers
0x0F8	PMCCNTR_EL0[31:0]	Performance Monitors Cycle Counter
0x0FC	PMCCNTR_EL0[63:32]	Performance Monitors Cycle Counter
0x200	PMPCSR[31:0]	Program Counter Sample Register
0x204	PMPCSR[63:32]	Program Counter Sample Register
0x208	PMCID1SR	CONTEXTIDR_EL1 Sample Register
0x20C	PMVIDSR	VMID Sample Register
0x220	PMPCSR[31:0]	Program Counter Sample Register
0x224	PMPCSR[63:32]	Program Counter Sample Register
0x228	PMCID1SR	CONTEXTIDR_EL1 Sample Register
0x22C	PMCID2SR	CONTEXTIDR_EL2 Sample Register

Offset	Name	Description
0x400 + 4n	PMEVTYPER<n>_EL0	Performance Monitors Event Type Registers
0x47C	PMCCFILTR_EL0	Performance Monitors Cycle Counter Filter Register
0xC00	PMCNTENSET_EL0	Performance Monitors Count Enable Set register
0xC20	PMCNTENCLR_EL0	Performance Monitors Count Enable Clear register
0xC40	PMINTENSET_EL1	Performance Monitors Interrupt Enable Set register
0xC60	PMINTENCLR_EL1	Performance Monitors Interrupt Enable Clear register
0xC80	PMOVSCLR_EL0	Performance Monitors Overflow Flag Status Clear register
0xCA0	PMSWINC_EL0	Performance Monitors Software Increment register
0xCC0	PMOVSSET_EL0	Performance Monitors Overflow Flag Status Set register
0xE00	PMCFGR	Performance Monitors Configuration Register
0xE04	PMCR_EL0	Performance Monitors Control Register
0xE20	PMCEID0	Performance Monitors Common Event Identification register 0
0xE24	PMCEID1	Performance Monitors Common Event Identification register 1
0xE28	PMCEID2	Performance Monitors Common Event Identification register 2
0xE2C	PMCEID3	Performance Monitors Common Event Identification register 3
0xE40	PMMIR	Performance Monitors Machine Identification Register
0xF00	PMITCTRL	Performance Monitors Integration mode Control register
0xFA8	PMDEVAFF0	Performance Monitors Device Affinity register 0
0xFAC	PMDEVAFF1	Performance Monitors Device Affinity register 1
0xFB0	PMLAR	Performance Monitors Lock Access Register
0xFB4	PMLSR	Performance Monitors Lock Status Register
0xFB8	PMAUTHSTATUS	Performance Monitors Authentication Status register
0xFBC	PMDEVARCH	Performance Monitors Device Architecture register
0xFC8	PMDEVID	Performance Monitors Device ID register
0xFCC	PMDEVTYPE	Performance Monitors Device Type register
0xFD0	PMPIDR4	Performance Monitors Peripheral Identification Register 4
0xFE0	PMPIDR0	Performance Monitors Peripheral Identification Register 0
0xFE4	PMPIDR1	Performance Monitors Peripheral Identification Register 1
0xFE8	PMPIDR2	Performance Monitors Peripheral Identification Register 2
0xFEC	PMPIDR3	Performance Monitors Peripheral Identification Register 3
0xFF0	PMCIDR0	Performance Monitors Component Identification Register 0
0xFF4	PMCIDR1	Performance Monitors Component Identification Register 1
0xFF8	PMCIDR2	Performance Monitors Component Identification Register 2
0xFFC	PMCIDR3	Performance Monitors Component Identification Register 3

In the RAS block:

Offset	Name	Description
0x000 + 64n	ERR<n>FR	Error Record Feature Register
0x008 + 64n	ERR<n>CTLR	Error Record Control Register
0x010 + 64n	ERR<n>STATUS	Error Record Primary Status Register
0x018 + 64n	ERR<n>ADDR	Error Record Address Register
0x020 + 64n	ERR<n>MISC0	Error Record Miscellaneous Register 0
0x028 + 64n	ERR<n>MISC1	Error Record Miscellaneous Register 1
0x030 + 64n	ERR<n>MISC2	Error Record Miscellaneous Register 2
0x038 + 64n	ERR<n>MISC3	Error Record Miscellaneous Register 3
0x800 + 64n	ERR<n>PFGF	Pseudo-fault Generation Feature Register
0x808 + 64n	ERR<n>PFGCTL	Pseudo-fault Generation Control Register
0x810 + 64n	ERR<n>PFGCDN	Pseudo-fault Generation Countdown Register

Offset	Name	Description
0xE00	ERRGSR	Error Group Status Register
0xE10	ERRIIDR	Implementation Identification Register
0xE80	ERRFHICR0	Fault-Handling Interrupt Configuration Register 0
0xE80 + 8n	ERRIRQCR<n>	Generic Error Interrupt Configuration Register
0xE88	ERRFHICR1	Fault-Handling Interrupt Configuration Register 1
0xE8C	ERRFHICR2	Fault-Handling Interrupt Configuration Register 2
0xE90	ERRERICR0	Error Recovery Interrupt Configuration Register 0
0xE98	ERRERICR1	Error Recovery Interrupt Configuration Register 1
0xE9C	ERRERICR2	Error Recovery Interrupt Configuration Register 2
0xEA0	ERRCRICR0	Critical Error Interrupt Configuration Register 0
0xEA8	ERRCRICR1	Critical Error Interrupt Configuration Register 1
0xEAC	ERRCRICR2	Critical Error Interrupt Configuration Register 2
0xEF8	ERRIRQSR	Error Interrupt Status Register
0xFA8	ERRDEVAFF	Device Affinity Register
0xFBC	ERRDEVARCH	Device Architecture Register
0xFC8	ERRDEVID	Device Configuration Register
0xFD0	ERRPIDR4	Peripheral Identification Register 4
0xFE0	ERRPIDR0	Peripheral Identification Register 0
0xFE4	ERRPIDR1	Peripheral Identification Register 1
0xFE8	ERRPIDR2	Peripheral Identification Register 2
0xFEC	ERRPIDR3	Peripheral Identification Register 3
0xFF0	ERRCIDR0	Component Identification Register 0
0xFF4	ERRCIDR1	Component Identification Register 1
0xFF8	ERRCIDR2	Component Identification Register 2
0xFFC	ERRCIDR3	Component Identification Register 3

In the Timer block:

Frame	Offset	Name	Description
CNTBaseN	0x000	CNTPCT[31:0]	Counter-timer Physical Count
CNTBaseN	0x004	CNTPCT[63:32]	Counter-timer Physical Count
CNTBaseN	0x008	CNTVCT[31:0]	Counter-timer Virtual Count
CNTBaseN	0x00C	CNTVCT[63:32]	Counter-timer Virtual Count
CNTBaseN	0x010	CNTFRQ	Counter-timer Frequency
CNTBaseN	0x014	CNTEL0ACR	Counter-timer EL0 Access Control Register
CNTBaseN	0x018	CNTVOFF[31:0]	Counter-timer Virtual Offset
CNTBaseN	0x01C	CNTVOFF[63:32]	Counter-timer Virtual Offset
CNTBaseN	0x020	CNTP_CVAL[31:0]	Counter-timer Physical Timer CompareValue
CNTBaseN	0x024	CNTP_CVAL[63:32]	Counter-timer Physical Timer CompareValue
CNTBaseN	0x028	CNTP_TVAL	Counter-timer Physical Timer TimerValue
CNTBaseN	0x02C	CNTP_CTL	Counter-timer Physical Timer Control
CNTBaseN	0x030	CNTV_CVAL[31:0]	Counter-timer Virtual Timer CompareValue
CNTBaseN	0x034	CNTV_CVAL[63:32]	Counter-timer Virtual Timer CompareValue
CNTBaseN	0x038	CNTV_TVAL	Counter-timer Virtual Timer TimerValue
CNTBaseN	0x03C	CNTV_CTL	Counter-timer Virtual Timer Control
CNTBaseN	0xFD0 + 4n	CounterID<n>	Counter ID registers
CNTCTLBase	0x000	CNTFRQ	Counter-timer Frequency
CNTCTLBase	0x004	CNTNSAR	Counter-timer Non-secure Access Register
CNTCTLBase	0x008	CNTTIDR	Counter-timer Timer ID Register

Frame	Offset	Name	Description
CNTCTLBase	0x040 + 4n	CNTACR<n>	Counter-timer Access Control Registers
CNTCTLBase	0x080 + 8n	CNTVOFF<n>[31:0]	Counter-timer Virtual Offsets
CNTCTLBase	0x084 + 8n	CNTVOFF<n>[63:32]	Counter-timer Virtual Offsets
CNTCTLBase	0xFD0 + 4n	CounterID<n>	Counter ID registers
CNTControlBase	0x000	CNTCR	Counter Control Register
CNTControlBase	0x004	CNTSR	Counter Status Register
CNTControlBase	0x008	CNTCV[63:0]	Counter Count Value register
CNTControlBase	0x020	CNTFID0	Counter Frequency ID
CNTControlBase	0x020 + 4n	CNTFID<n>	Counter Frequency IDs, n > 0
CNTControlBase	0x10	CNTSCR	Counter Scale Register
CNTControlBase	0x1C	CNTID	Counter Identification Register
CNTControlBase	0xFD0 + 4n	CounterID<n>	Counter ID registers
CNTEL0BaseN	0x000	CNTPCT[31:0]	Counter-timer Physical Count
CNTEL0BaseN	0x004	CNTPCT[63:32]	Counter-timer Physical Count
CNTEL0BaseN	0x008	CNTVCT[31:0]	Counter-timer Virtual Count
CNTEL0BaseN	0x00C	CNTVCT[63:32]	Counter-timer Virtual Count
CNTEL0BaseN	0x010	CNTFRQ	Counter-timer Frequency
CNTEL0BaseN	0x020	CNTP_CVAL[31:0]	Counter-timer Physical Timer CompareValue
CNTEL0BaseN	0x024	CNTP_CVAL[63:32]	Counter-timer Physical Timer CompareValue
CNTEL0BaseN	0x028	CNTP_TVAL	Counter-timer Physical Timer TimerValue
CNTEL0BaseN	0x02C	CNTP_CTL	Counter-timer Physical Timer Control
CNTEL0BaseN	0x030	CNTV_CVAL[31:0]	Counter-timer Virtual Timer CompareValue
CNTEL0BaseN	0x034	CNTV_CVAL[63:32]	Counter-timer Virtual Timer CompareValue
CNTEL0BaseN	0x038	CNTV_TVAL	Counter-timer Virtual Timer TimerValue
CNTEL0BaseN	0x03C	CNTV_CTL	Counter-timer Virtual Timer Control
CNTEL0BaseN	0xFD0 + 4n	CounterID<n>	Counter ID registers
CNTReadBase	0x000	CNTCV[63:0]	Counter Count Value register
CNTReadBase	0xFD0 + 4n	CounterID<n>	Counter ID registers

(old) htmldiff from- (new)

(old) htmldiff from- (new)

AMEVTYPER0<n>, Activity Monitors Event Type Registers 0, n = 0 - 15

The AMEVTYPER0<n> characteristics are:

Purpose

Provides information on the events that an architected activity monitor event counter AMEVCNTR0<n> counts.

Configuration

External register AMEVTYPER0<n> bits [31:0] are architecturally mapped to AArch64 System register AMEVTYPER0<n>_EL0[31:0] .

External register AMEVTYPER0<n> bits [31:0] are architecturally mapped to AArch32 System register AMEVTYPER0<n>[31:0] .

The power domain of AMEVTYPER0<n> is IMPLEMENTATION DEFINED.

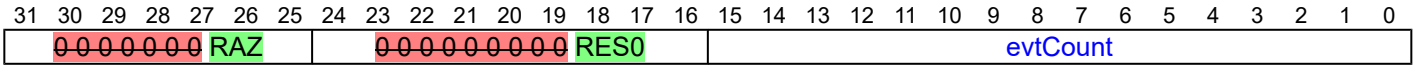
This register is present only when AMUv1 is implemented. Otherwise, direct accesses to AMEVTYPER0<n> are RES0.

Attributes

AMEVTYPER0<n> is a 32-bit register.

Field descriptions

The AMEVTYPER0<n> bit assignments are:



Bits [31:25]

Reserved, RAZ.

Bits [24:16]

Reserved, RES0.

evtCount, bits [15:0]

Event to count. The event number of the event that is counted by the architected activity monitor event counter AMEVCNTR0<n>. The value of this field is architecturally mandated for each architected counter.

The following table shows the mapping between required event numbers and the corresponding counters:

evtCount	Meaning	Applies when
0x0011	Processor frequency cycles	When n == 0
0x4004	Constant frequency cycles	When n == 1
0x0008	Instructions retired	When n == 2
0x4005~~0x4004~~	Memory stall cycles	When n == 3

Accessing the AMEVTYPER0<n>

If <n> is greater than or equal to the number of architected activity monitor event counters, reads and writes of AMEVTYPER0<n> are CONSTRAINED UNPREDICTABLE, and accesses to the register behave as RAZ/WI.

Note

AMCGCR.CG0NC identifies the number of architected activity monitor event counters.

AMEVTYPER0<n> can be accessed through the memory-mapped interfaces:

Component	Offset	Instance
AMU	`0x400` + 4n	AMEVTYPER0<n>

Access on this interface is **RO**.

(old) htmldiff from- (new)

CTICLAIMCLR, CTI Claim Tag Clear register

The CTICLAIMCLR characteristics are:

Purpose

Used by software to read the values of the CLAIM bits, and to clear these bits to 0.

Configuration

CTICLAIMCLR is in the Debug power domain.

This register is not affected by a Warm reset, and is not affected by a Cold reset.

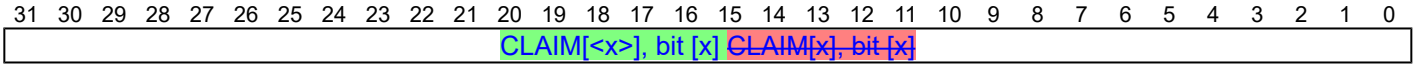
Implementation of this register is OPTIONAL.

Attributes

CTICLAIMCLR is a 32-bit register.

Field descriptions

The CTICLAIMCLR bit assignments are:



CLAIM[<[x>],], bit [x], for x = 0 to 31

CLAIM tag clear bit.

For values of x greater than or equal to the IMPLEMENTATION DEFINED number of CLAIM tags, this bit is RAZ/SBZ. Software can rely on these bits reading as zero, and must use a Should-Be-Zero policy on writes. Implementations must ignore writes.

For other values of x, reads return the value of CLAIM[x] and the behavior on writes is:

CLAIM[<[x>]]	Meaning
0b0	No action.
0b1	Indirectly clear CLAIM[x] to 0.

A single write to CTICLAIMCLR can clear multiple tags to 0.

An External Debug reset clears the CLAIM tag bits to 0.

An External Debug reset clears the CLAIM tag bits to 0.

Accessing the CTICLAIMCLR

CTICLAIMCLR can be accessed through the external debug interface:

Component	Offset	Instance
CTI	0xFA4	CTICLAIMCLR

This interface is accessible as follows:

- When SoftwareLockStatus() access to this register is **RO**.

- When !SoftwareLockStatus() access to this register is **RW**.

(old) htmldiff from- (new)

(old) htmldiff from- (new)

CTICLAIMSET, CTI Claim Tag Set register

The CTICLAIMSET characteristics are:

Purpose

Used by software to set CLAIM bits to 1.

Configuration

CTICLAIMSET is in the Debug power domain. Some or all RW fields of this register have defined reset values. These apply only on an External debug reset. The register is not affected by a Warm reset and is not affected by a Cold reset.

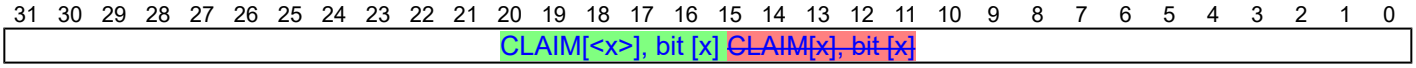
Implementation of this register is OPTIONAL.

Attributes

CTICLAIMSET is a 32-bit register.

Field descriptions

The CTICLAIMSET bit assignments are:



CLAIM[<x>], bit [x], for x = 0 to 31

CLAIM tag set bit.

For values of x greater than or equal to the IMPLEMENTATION DEFINED number of CLAIM tags, this bit is RAZ/SBZ. Software can rely on these bits reading as zero, and must use a Should-Be-Zero policy on writes. Implementations must ignore writes.

For other values of x, the bit is RAO and the behavior on writes is:

CLAIM[<x>]	Meaning
0b0	No action.
0b1	Indirectly set CLAIM[x] tag to 1.

A single write to CTICLAIMSET can set multiple tags to 1.

An External Debug reset clears the CLAIM tag bits to 0.

Accessing the CTICLAIMSET

CTICLAIMSET can be accessed through the external debug interface:

Component	Offset	Instance
CTI	0xFA0	CTICLAIMSET

This interface is accessible as follows:

- When SoftwareLockStatus() access to this register is **RO**.
- When !SoftwareLockStatus() access to this register is **RW**.

27~~13~~/03~~12~~/2019~~2018~~ 21~~16~~:59~~42~~; e5e4db499bf9867a4b93324c4dbac985d3da93766~~379d01e197f1d40720d32d0f84e419e9187e009~~

(old) htmldiff from- (new)

(old) htmldiff from- (new)

EDDEVID, External Debug Device ID register 0

The EDDEVID characteristics are:

Purpose

Provides extra information for external debuggers about features of the debug implementation.

Configuration

It is IMPLEMENTATION DEFINED whether EDDEVID is implemented in the Core power domain or in the Debug power domain.

If ARMv8.3-DoPD is implemented, this register is in the Core power domain. If ARMv8.3-DoPD is not implemented, this register is in the Debug power domain.

Attributes

EDDEVID is a 32-bit register.

Field descriptions

The EDDEVID bit assignments are:

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RES0				AuxRegs				RES0																DebugPower				PCSample			
~~0~~	~~0~~	~~0~~	~~0~~	~~AuxRegs~~				~~0~~	~~0~~	~~0~~	~~0~~	~~0~~	~~0~~	~~0~~	~~0~~	~~0~~	~~0~~	~~0~~	~~0~~	~~0~~	~~0~~	~~0~~	~~0~~	~~DebugPower~~				~~PCSample~~			

Bits [31:28]

Reserved, RES0.

AuxRegs, bits [27:24]

Indicates support for Auxiliary registers. Permitted values for this field are:

AuxRegs	Meaning
0b0000	None supported.
0b0001	Support for External Debug Auxiliary Control Register, EDACR.

All other values are reserved.

Bits [23:8]

Reserved, RES0.

DebugPower, bits [7:4]

When ARMv8.3-DoPD is implemented:

Indicates support for the ~~Armv8.3 debug over powerdown feature. Defined values of this field are:~~ ARMv8.3-DoPD feature. Defined values of this field are:

DebugPower	Meaning
0b0000	ARMv8.3-DoPD ~~Armv8.3 debug over powerdown~~ not implemented. Registers in the external debug interface register map are implemented in a mix of the Debug and Core power domains.
0b0001	ARMv8.3-DoPD ~~Armv8.3 debug over powerdown~~ implemented. All registers in the external debug interface register map are implemented in the Core power domain.

ARMv8.3-DoPD implements the functionality added by the value 0b0001.

All other values are reserved.

Otherwise:

Reserved, RES0.

PCSample, bits [3:0]

Indicates the level of PC Sample-based Profiling support using external debug registers. Permitted values of this field are:

PCSample	Meaning
0b0000	PC Sample-based Profiling Extension is not implemented in the external debug registers space.
0b0010	Only EDPCSR and EDCIDSR are implemented. This option is only permitted if EL3 and EL2 are not implemented.
0b0011	EDPCSR, EDCIDSR, and EDVIDSR are implemented.

All other values are reserved.

When ARMv8.2-PCSample is implemented, the only permitted value is 0b0000.

Note

ARMv8.2-PCSample implements the PC Sample-based Profiling Extension in the Performance Monitors register space, as indicated by the value of PMDEVID.PCSample.

Accessing the EDDEVID

EDDEVID can be accessed through the external debug interface:

Component	Offset	Instance
Debug	0xFC8	EDDEVID

This interface is accessible as follows:

- When ARMv8.3-DoPD is not implemented or IsCorePowered() access to this register is **RO**.
- Otherwise access to this register returns an Error.

(old) htmldiff from- (new)

EDDFR, External Debug Feature Register

The EDDFR characteristics are:

Purpose

Provides top level information about the debug system.

Note

Debuggers must use EDDEVARCH to determine the Debug architecture version.

For general information about the interpretation of the ID registers, see 'Principles of the ID scheme for fields in ID registers' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

Configuration

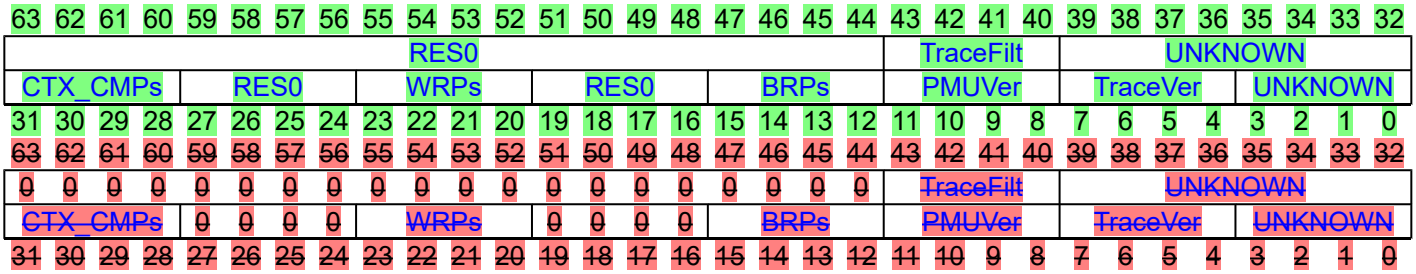
It is IMPLEMENTATION DEFINED whether EDDFR is implemented in the Core power domain or in the Debug power domain.

Attributes

EDDFR is a 64-bit register.

Field descriptions

The EDDFR bit assignments are:



Bits [63:44]

Reserved, RES0.

TraceFilt, bits [43:40]

When ARMv8.4-Trace is implemented:

Armv8.4 Self-hosted Trace Extension version. The defined values of this field are:

TraceFilt	Meaning
0b0000	Armv8.4 Self-hosted Trace Extension is not implemented.
0b0001	Armv8.4 Self-hosted Trace Extension is implemented.

All other values are reserved.

Otherwise:

Reserved, RES0.

Bits~~UNKNOWN, bits~~ [39:32]

Reserved, UNKNOWN.

CTX_CMPs, bits [31:28]

Number of breakpoints that are context-aware, minus 1. These are the highest numbered breakpoints.

In an Armv8-A implementation that supports AArch64 state in at least one Exception level, this field returns the value of ID_AA64DFR0_EL1.CTX_CMPs.

Bits [27:24]

Reserved, RES0.

WRPs, bits [23:20]

Number of watchpoints, minus 1. The value of `0b0000` is reserved.

In an Armv8-A implementation that supports AArch64 state in at least one Exception level, this field returns the value of ID_AA64DFR0_EL1.WRPs.

Bits [19:16]

Reserved, RES0.

BRPs, bits [15:12]

Number of breakpoints, minus 1. The value of `0b0000` is reserved.

In an Armv8-A implementation that supports AArch64 state in at least one Exception level, this field returns the value of ID_AA64DFR0_EL1.BRPs.

PMUVer, bits [11:8]

Performance Monitors Extension version.

This field does not follow the standard ID scheme, but uses the Alternative ID scheme described in 'Alternative ID scheme used for the Performance Monitors Extension version' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section D10.1.4.

Defined values are:

PMUVer	Meaning
`0b0000`	Performance Monitors Extension not implemented.
`0b0001`	Performance Monitors Extension implemented, PMUv3.
`0b0100`	PMUv3 for Armv8.1. As `0b0001`, and also includes support for: • Extended 16-bit PMEVTYPER<n>_EL0.evtCount field. • If EL2 is implemented, the MDCR_EL2.HPMD control bit.
`0b0101`	PMUv3 for Armv8.4. As `0b0100` and also includes support for the PMMIR register.
`0b0110`	PMUv3 for Armv8.5. As `0b0101` and also includes support for: • 64-bit event counters. • If EL2 is implemented, the MDCR_EL2.HCCD control bit. • If EL3 is implemented, the MDCR_EL3.SCCD control bit.
`0b1111`	IMPLEMENTATION DEFINED form of performance monitors supported, PMUv3 not supported. Arm does not recommend this value in new implementations.

ARMv8.1-PMU implements the functionality added by the value `0b0100`.

ARMv8.4-PMU implements the functionality added by the value `0b0101`.

ARMv8.5-PMU implements the functionality added by the value 0b0110.

All other values are reserved.

From Armv8.1, the value 0b0001 is not permitted.

From Armv8.4, the value 0b0100 is not permitted.

From Armv8.5, the value 0b0101 is not permitted.

In an Armv8-A implementation that supports AArch64 state in at least one Exception level, this field returns the value of ID_AA64DFR0_EL1.PMUVer.

TraceVer, bits [7:4]

Trace support. Indicates whether System register interface to a PE trace unit is implemented. Defined values are:

TraceVer	Meaning
0b0000	PE trace unit System registers not implemented.
0b0001	PE trace unit System registers implemented.

All other values are reserved.

A value of 0b0000 only indicates that no System register interface to a PE trace unit is implemented. A PE trace unit might nevertheless be implemented without a System register interface.

In an Armv8-A implementation that supports AArch64 state in at least one Exception level, this field returns the value of ID_AA64DFR0_EL1.TraceVer.

Bits~~UNKNOWN, bits~~ [3:0]

Reserved, UNKNOWN.

Accessing the EDDFR

EDDFR can be accessed through the external debug interface:

Component	Offset	Instance	Range
Debug	0xD28	EDDFR	31:0

This interface is accessible as follows:

- When IsCorePowered() and !DoubleLockStatus() access to this register is **RO**.
- Otherwise access to this register is **IMPDEF**.

Component	Offset	Instance	Range
Debug	0xD2C	EDDFR	63:32

This interface is accessible as follows:

- When IsCorePowered() and !DoubleLockStatus() access to this register is **RO**.
- Otherwise access to this register is **IMPDEF**.

27~~13~~/03~~12~~/2019~~2018~~ 21~~16~~:59~~42~~; e5e4db499bf9867a4b93324c4dbac985d3da93766379d01e197f1d40720d32d0f84e419e9187e009

(old) htmldiff from- (new)

EDECCR, External Debug Exception Catch Control Register

The EDECCR characteristics are:

Purpose

Controls Exception Catch debug events.

Configuration

External register EDECCR bits [31:0] are architecturally mapped to AArch64 System register OSECCR_EL1[31:0] .

External register EDECCR bits [31:0] are architecturally mapped to AArch32 System register DBGOSECCR[31:0] .

EDECCR is in the Core power domain. Some or all RW fields of this register have defined reset values. These apply only on a Cold reset. The register is not affected by a Warm reset and is not affected by an External debug reset.

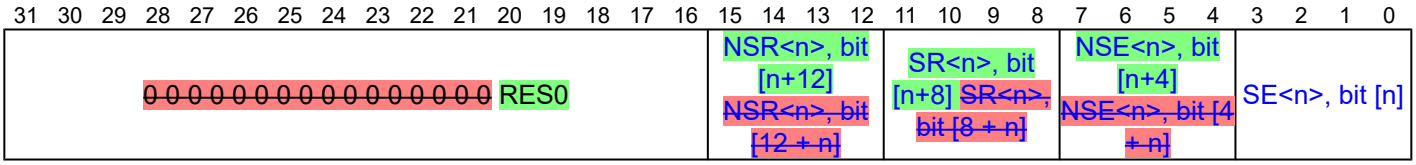
Attributes

EDECCR is a 32-bit register.

Field descriptions

The EDECCR bit assignments are:

When ARMv8.2-Debug is implemented:



Bits [31:16]

Reserved, RES0.

NSR<n>, bit [12 + n+12], for n = 0 to 3

Controls Non-secure exception catch on exception return to EL<n> in conjunction with NSE<n>. See the summary of Exception Catch debug event control for information.

If EL3 is not implemented and the PE behaves as if SCR_EL3.NS is set to 0, this field is reserved, RES0. Otherwise, possible values for this field are:

NSR<n>	Meaning
0b0	If the corresponding NSE<n> bit is 0, then Exception Catch debug events are disabled for Non-secure Exception level <n>. If the corresponding NSE<n> bit is 1, then Exception Catch debug events are enabled for exception entry, reset entry and exception return to Non-secure Exception level <n>.
0b1	If the corresponding NSE<n> bit is 0, then Exception Catch debug events are enabled for exception returns to Non-secure Exception level <n>. If the corresponding NSE<n> bit is 1, then Exception Catch debug events are enabled for exception entry and reset entry to Non-secure Exception level <n>.

Note

It is IMPLEMENTATION DEFINED whether a reset entry to an Exception level is permitted to generate an Exception Catch debug event.

A value of the NSR field that enables an Exception Catch debug event for an Exception level that is not implemented is reserved. If the NSR field is programmed with a reserved value then:

- The PE behaves as if it is programmed with a defined value, other than for a read of EDECCR.
- The value returned for NSR by a read of EDECCR is UNKNOWN.

On a Cold reset, this field resets to `0`.

SR<n>, bit [8 + n+8], for n = 0 to 3

Controls Secure exception catch on exception return to EL<n> in conjunction with SE<n>. See the summary of Exception Catch debug event control for information.

If EL3 is not implemented and the PE behaves as if SCR_EL3.NS is set to 1, this field is reserved, RES0. Otherwise, possible values for this field are:

SR<n>	Meaning
`0b0`	If the corresponding SE<n> bit is 0, then Exception Catch debug events are disabled for Secure Exception level <n>. If the corresponding SE<n> bit is 1, then Exception Catch debug events are enabled for exception entry, reset entry and exception return to Secure Exception level <n>.
`0b1`	If the corresponding SE<n> bit is 0, then Exception Catch debug events are enabled for exception returns to Secure Exception level <n>. If the corresponding SE<n> bit is 1, then Exception Catch debug events are enabled for exception entry and reset entry to Secure Exception level <n>.

Note

It is IMPLEMENTATION DEFINED whether a reset entry to an Exception level is permitted to generate an Exception Catch debug event.

A value of the SR field that enables an Exception Catch debug event for an Exception level that is not implemented is reserved. If the SR field is programmed with a reserved value then:

- The PE behaves as if it is programmed with a defined value, other than for a read of EDECCR.
- The value returned for SR by a read of EDECCR is UNKNOWN.

On a Cold reset, this field resets to `0`.

NSE<n>, bit [4 + n+4], for n = 0 to 3

Coarse-grained Non-secure exception catch for EL<n>. This controls whether Exception Catch debug events are enabled for Non-secure EL<n>. This also controls:

- The behavior of exception catch on exception entry to EL<n>.
- The behavior of exception catch on exception return to EL<n> in conjunction with NSR<n>.

If EL3 is not implemented and the PE behaves as if SCR_EL3.NS is set to 0, this field is reserved, RES0. Otherwise, possible values for this field are:

NSE<n>	Meaning
`0b0`	If the corresponding NSR<n> bit is 0, then Exception Catch debug events are disabled for Non-secure Exception level <n>. If the corresponding NSR<n> bit is 1, then Exception Catch debug events are enabled for exception returns to Non-secure Exception level <n>.
`0b1`	If the corresponding NSR<n> bit is 0, then Exception Catch debug events are enabled for exception entry, reset entry and exception return to Non-secure Exception level <n>. If the corresponding NSR<n> bit is 1, then Exception Catch debug events are enabled for exception entry and reset entry to Non-secure Exception level <n>.

A value of the NSE field that enables an Exception Catch debug event for an Exception level that is not implemented is reserved. If the NSE field is programmed with a reserved value then:

- The PE behaves as if it is programmed with a defined value, other than for a read of EDECCR.
- The value returned for NSE by a read of EDECCR is UNKNOWN.

On a Cold reset, this field resets to 0.

SE<n>, bit [n], for n = 0 to 3

Coarse-grained Secure exception catch for EL<n>. This field controls whether Exception Catch debug events are enabled for Secure EL<n>.

- The behavior of exception catch on exception entry to EL<n>.
- The behavior of exception catch on exception return to EL<n> in conjunction with SR<n>.

If EL3 is not implemented and the PE behaves as if SCR_EL3.NS is set to 1, this field is reserved, RES0. Otherwise, possible values for this field are:

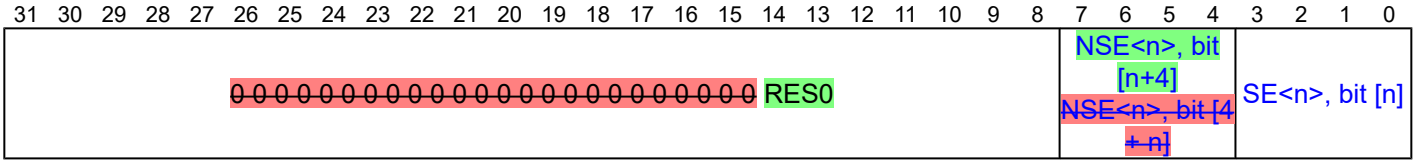
SE<n>	Meaning
0b0	If the corresponding SR<n> bit is 0, then Exception Catch debug events are disabled for Secure Exception level <n>. If the corresponding SR<n> bit is 1, then Exception Catch debug events are enabled for exception returns to Secure Exception level <n>.
0b1	If the corresponding SR<n> bit is 0, then Exception Catch debug events are enabled for exception entry, reset entry and exception return to Secure Exception level <n>. If the corresponding SR<n> bit is 1, then Exception Catch debug events are enabled for exception entry and reset entry to Secure Exception level <n>.

A value of the SE field that enables an Exception Catch debug event for an Exception level that is not implemented is reserved. If the SE field is programmed with a reserved value then:

- The PE behaves as if it is programmed with a defined value, other than for a read of EDECCR.
- The value returned for SE by a read of EDECCR is UNKNOWN.

On a Cold reset, this field resets to 0.

Otherwise:



Bits [31:8]

Reserved, RES0.

NSE<n>, bit [4 + n+4], for n = 0 to 3

Coarse-grained Non-secure exception catch. If EL3 and EL2 are not implemented and the PE behaves as if SCR_EL3.NS is set to 0, this field is reserved, RES0. Otherwise, possible values for this field are:

NSE<n>	Meaning
0b0	Exception Catch debug events are disabled for Non-secure Exception level <n>.
0b1	Exception Catch debug events are enabled for Non-secure Exception level <n>.

A value of the NSE field that enables an Exception Catch debug event for an Exception level that is not implemented is reserved. If the NSE field is programmed with a reserved value then:

- The PE behaves as if it is programmed with a defined value, other than for a read of EDECCR.
- The value returned for NSE by a read of EDECCR is UNKNOWN.

SE<n>, bit [n], for n = 0 to 3

Coarse-grained Secure exception catch. If EL3 is not implemented and the PE behaves as if SCR_EL3.NS is set to 1, this field is reserved, RES0. Otherwise, possible values for this field are:

SE<n>	Meaning
0b0	Exception Catch debug events are disabled for Secure Exception level <n>.
0b1	Exception Catch debug events are enabled for Secure Exception level <n>.

A value of the SE field that enables an Exception Catch debug event for an Exception level that is not implemented is reserved. If the SE field is programmed with a reserved value then:

- The PE behaves as if it is programmed with a defined value, other than for a read of EDECCR.
- The value returned for SE by a read of EDECCR is UNKNOWN.

Accessing the EDECCR

EDECCR can be accessed through the external debug interface:

Component	Offset	Instance
Debug	0x098	EDECCR

This interface is accessible as follows:

- When IsCorePowered(), !DoubleLockStatus(), !OSLockStatus() and SoftwareLockStatus() access to this register is **RO**.
- When IsCorePowered(), !DoubleLockStatus(), !OSLockStatus() and !SoftwareLockStatus() access to this register is **RW**.
- Otherwise access to this register returns an Error.

27~~13~~/03~~12~~/20192~~018~~ 21~~16~~:59~~42~~; e5e4db499bf9867a4b93324c4dbac985d3da93766~~379d01e197f1d40720d32d0f84e419e9187e009~~

(old) htmldiff from- (new)

EDECR, External Debug Execution Control Register

The EDECR characteristics are:

Purpose

Controls Halting debug events.

Configuration

It is IMPLEMENTATION DEFINED whether EDECR is implemented in the Core power domain or in the Debug power domain. Some or all RW fields of this register have defined reset values, and:

- The register is not affected by a Warm reset.
- If the register is implemented in the Core power domain the reset values apply on a Cold reset, and the register is not affected by an External debug reset.
- If the register is implemented in the Debug power domain the reset values apply on an External debug reset, and the register is not affected by a Cold reset.

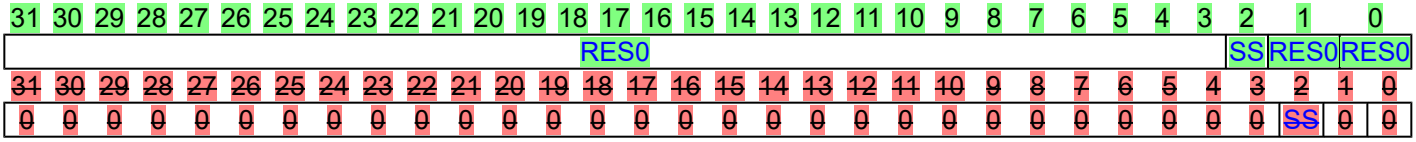
If ARMv8.3-DoPD is implemented, this register is in the Core power domain. If ARMv8.3-DoPD is not implemented, this register is in the Debug power domain.

Attributes

EDECR is a 32-bit register.

Field descriptions

The EDECR bit assignments are:



Bits [31:3]

Reserved, RES0.

SS, bit [2]

Halting step enable. Possible values of this field are:

SS	Meaning
0b0	Halting step debug event disabled.
0b1	Halting step debug event enabled.

If the value of EDECR.SS is changed when the PE is in Non-debug state, behavior is CONSTRAINED UNPREDICTABLE as described in 'Changing the value of EDECR.SS when not in Debug state' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section H3.2.5.

The following resets apply:

- OnIf aArmv8.3-DoPD is implemented, this register is reset by Cold reset and not affected by External debug reset. If Armv8.3-DoPD is not implemented, this fieldregister resetsis toreset :by External debug reset and not affected by Cold reset.
    - if IsFeatureImplemented(ARMv8.3-DoPD), this field resets to 0. .

- On a Debug reset, this field resets to :
  - if IsFeatureImplemented(ARMv8.3-DoPD), this field resets to `0`. .

~~On a reset, this field resets to `0`.~~

Bit [1]

When ARMv8.3-DoPD is implemented:

Reserved, RES0.

Otherwise:

Reset Catch Enable.

RCE	Meaning
`0b0`	Reset Catch debug event disabled.
`0b1`	Reset Catch debug event enabled.

On a External debug reset, this field resets to `0`.

Bit [0]

When ARMv8.3-DoPD is implemented:

Reserved, RES0.

Otherwise:

OS Unlock Catch Enable.

OSUCE	Meaning
`0b0`	OS Unlock Catch debug event disabled.
`0b1`	OS Unlock Catch debug event enabled.

On a External debug reset, this field resets to `0`.

Accessing the EDECR

EDECR can be accessed through the external debug interface:

Component	Offset	Instance
Debug	`0x024`	EDECR

This interface is accessible as follows:

- When (ARMv8.3-DoPD is not implemented or IsCorePowered()) and SoftwareLockStatus() access to this register is **RO**.
- When (ARMv8.3-DoPD is not implemented or IsCorePowered()) and !SoftwareLockStatus() access to this register is **RW**.
- Otherwise access to this register returns an Error.

27~~13~~/03~~12~~/2019~~2018~~ 21~~16~~:59~~42~~; e5e4db499bf9867a4b93324c4dbac985d3da93766~~379d01e197f1d40720d32d0f84e419e9187e009~~

(old) htmldiff from- (new)

EDPFR, External Debug Processor Feature Register

The EDPFR characteristics are:

Purpose

Provides information about implemented PE features.

For general information about the interpretation of the ID registers, see 'Principles of the ID scheme for fields in ID registers' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

Configuration

It is IMPLEMENTATION DEFINED whether EDPFR is implemented in the Core power domain or in the Debug power domain.

Attributes

EDPFR is a 64-bit register.

Field descriptions

The EDPFR bit assignments are:

63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
UNKNOWN				UNKNOWN				~~0~~	~~0~~	~~0~~	~~0~~ RES0					UNKNOWN				AMU				UNKNOWN				SEL2				SVE			
UNKNOWN				GIC				AdvSIMD				FP				EL3				EL2				EL1				EL0			
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Bits~~UNKNOWN, bits~~ [63:60]

From Armv8.5:

Reserved, UNKNOWN.

Otherwise:

Reserved, RES0.

Bits~~UNKNOWN, bits~~ [59:56]

From Armv8.5:

Reserved, UNKNOWN.

Otherwise:

Reserved, RES0.

Bits [55:52]

Reserved, RES0.

Bits~~UNKNOWN, bits~~ [51:48]

From Armv8.4:

Reserved, UNKNOWN.

Otherwise:

Reserved, RES0.

AMU, bits [47:44]

From Armv8.4:

Activity Monitors Extension. Defined values are:

AMU	Meaning
0b0000	Activity Monitors Extension is not implemented.
0b0001	Activity Monitors Extension version 1 is implemented.

All other values are reserved.

ARMv8.4-AMUv1 implements the functionality identified by the value 0b0001.

In Armv8.0, Armv8.1, Armv8.2, and Armv8.3, the only permitted value is 0b0000.

From Armv8.4, the permitted values are 0b0000 and 0b0001.

Otherwise:

Reserved, RES0.

Bits~~UNKNOWN, bits~~ [43:40]

From Armv8.2:

Reserved, UNKNOWN.

Otherwise:

Reserved, RES0.

SEL2, bits [39:36]

From Armv8.4:

Secure EL2. Defined values are:

SEL2	Meaning
0b0000	Secure EL2 is not implemented.
0b0001	Secure EL2 is implemented.

All other values are reserved.

Otherwise:

Reserved, RES0.

SVE, bits [35:32]

From Armv8.2:

Scalable Vector Extension. Defined values are:

SVE	Meaning
0b0000	SVE is not implemented.
0b0001	SVE is implemented.

All other values are reserved.

Otherwise:

Reserved, RES0.

Bits~~UNKNOWN, bits~~ [31:28]

When RAS is implemented:

Reserved, UNKNOWN.

Otherwise:

Reserved, RES0.

GIC, bits [27:24]

System register GIC interface support. Defined values are:

GIC	Meaning
0b0000	No System register interface to the GIC is supported.
0b0001	System register interface to versions 3.0 and 4.0 of the GIC CPU interface is supported.

All other values are reserved.

In an Armv8-A implementation that supports AArch64 state in at least one Exception level, this field returns the value of ID_AA64PFR0_EL1.GIC.

AdvSIMD, bits [23:20]

Advanced SIMD. Defined values are:

AdvSIMD	Meaning
0b0000	Advanced SIMD is implemented, including support for the following SISD and SIMD operations: • Integer byte, halfword, word and doubleword element operations. • Single-precision and double-precision floating-point arithmetic. • Conversions between single-precision and half-precision data types, and double-precision and half-precision data types.
0b0001	As for 0b0000, and also includes support for half-precision floating-point arithmetic.
0b1111	Advanced SIMD is not implemented.

All other values are reserved.

This field must have the same value as the FP field.

The permitted values are:

- 0b0000 in an implementation with Advanced SIMD support, that does not include the ARMv8.2-FP16 extension.
- 0b0001 in an implementation with Advanced SIMD support, that includes the ARMv8.2-FP16 extension.
- 0b1111 in an implementation without Advanced SIMD support.

In an Armv8-A implementation that supports AArch64 state in at least one Exception level, this field returns the value of ID_AA64PFR0_EL1.AdvSIMD.

FP, bits [19:16]

Floating-point. Defined values are:

FP	Meaning
0b0000	Floating-point is implemented, and includes support for: • Single-precision and double-precision floating-point types. • Conversions between single-precision and half-precision data types, and double-precision and half-precision data types.
0b0001	As for 0b0000, and also includes support for half-precision floating-point arithmetic.
0b1111	Floating-point is not implemented.

All other values are reserved.

This field must have the same value as the AdvSIMD field.

The permitted values are:

- 0b0000 in an implementation with floating-point support, that does not include the ARMv8.2-FP16 extension.
- 0b0001 in an implementation with floating-point support, that includes the ARMv8.2-FP16 extension.
- 0b1111 in an implementation without floating-point support.

In an Armv8-A implementation that supports AArch64 state in at least one Exception level, this field returns the value of ID_AA64PFR0_EL1.FP.

EL3, bits [15:12]

AArch64 EL3 Exception level handling. Defined values are:

EL3	Meaning
0b0000	EL3 is not implemented or cannot be executed in AArch64 state.
0b0001	EL3 can be executed in AArch64 state only.
0b0010	EL3 can be executed in either AArch64 or AArch32 state.

When the value of EDAA32PFR.EL3 is non-zero, this field must be 0b0000.

All other values are reserved.

In an Armv8-A implementation that supports AArch64 state in at least one Exception level, this field returns the value of ID_AA64PFR0_EL1.EL3.

EL2, bits [11:8]

AArch64 EL2 Exception level handling. Defined values are:

EL2	Meaning
0b0000	EL2 is not implemented or cannot be executed in AArch64 state.
0b0001	EL2 can be executed in AArch64 state only.
0b0010	EL2 can be executed in either AArch64 or AArch32 state.

When the value of EDAA32PFR.EL2 is non-zero, this field must be 0b0000.

All other values are reserved.

In an Armv8-A implementation that supports AArch64 state in at least one Exception level, this field returns the value of ID_AA64PFR0_EL1.EL2.

EL1, bits [7:4]

AArch64 EL1 Exception level handling. Defined values are:

EL1	Meaning
0b0000	EL1 can be executed in AArch32 state only.
0b0001	EL1 can be executed in AArch64 state only.
0b0010	EL1 can be executed in either AArch64 or AArch32 state.

All other values are reserved.

In an Armv8-A implementation that supports AArch64 state in at least one Exception level, this field returns the value of ID_AA64PFR0_EL1.EL1.

EL0, bits [3:0]

AArch64 EL0 Exception level handling. Defined values are:

EL0	Meaning
0b0000	EL0 can be executed in AArch32 state only.
0b0001	EL0 can be executed in AArch64 state only.
0b0010	EL0 can be executed in either AArch64 or AArch32 state.

All other values are reserved.

In an Armv8-A implementation that supports AArch64 state in at least one Exception level, this field returns the value of ID_AA64PFR0_EL1.EL0.

Accessing the EDPFR

EDPFR can be accessed through the external debug interface:

Component	Offset	Instance	Range
Debug	0xD20	EDPFR	31:0

This interface is accessible as follows:

- When IsCorePowered() and !DoubleLockStatus() access to this register is **RO**.
- Otherwise access to this register is **IMPDEF**.

Component	Offset	Instance	Range
Debug	0xD24	EDPFR	63:32

This interface is accessible as follows:

- When IsCorePowered() and !DoubleLockStatus() access to this register is **RO**.
- Otherwise access to this register is **IMPDEF**.

2713/0312/20192018 2116:5942: e5e4db499bf9867a4b93324c4dbac985d3da93766379d01e197f1d40720d32d0f84e419e9187e009

(old) htmldiff from- (new)

EDPRSR, External Debug Processor Status Register

The EDPRSR characteristics are:

Purpose

Holds information about the reset and powerdown state of the PE.

Configuration

EDPRSR contains fields that are in the Core power domain and fields that are in the Debug power domain.

Some of the fields in the Core power domain are in the Cold reset domain and others are in the Warm reset domain. See the field descriptions for more information. However:

- Fields that are in the Cold reset domain are not affected by a warm reset and are not affected by an External debug reset.
- Fields in the Warm reset domain are also reset by a Cold reset but are not affected by an External debug reset.
- Fields in the Debug power domain are not affected by a Warm reset and are not affected by a Cold reset.

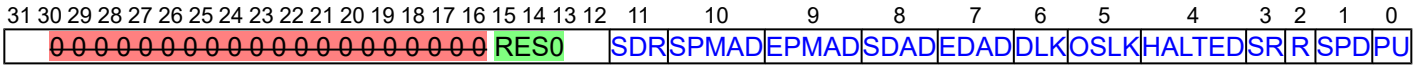
If ARMv8.3-DoPD is implemented then all fields in this register are in the Core power domain.

Attributes

EDPRSR is a 32-bit register.

Field descriptions

The EDPRSR bit assignments are:



Bits [31:12]

Reserved, RES0.

SDR, bit [11]

Sticky debug restart. Set to 1 when the PE exits Debug state.

Permitted values are:

SDR	Meaning
0b0	The PE has not restarted since EDPRSR was last read.
0b1	The PE has restarted since EDPRSR was last read.

Note

If a reset occurs when the PE is in Debug state, the PE exits Debug state. SDR is UNKNOWN on Warm reset, meaning a debugger must also use the SR bit to determine whether the PE has left Debug state.

If The Core power domain is powered up, then following a read of EDPRSR:

- If ~~the OS Double Lock is not implemented or DoubleLockStatus() == FALSE this bit clears to 0.~~ ARMv8.0-DoubleLock is not implemented or DoubleLockStatus() == FALSE this bit clears to 0.
- If ~~the OS Double Lock is implemented and DoubleLockStatus() == TRUE, it is~~ ARMv8.0-DoubleLock is implemented and DoubleLockStatus() == TRUE, it is CONSTRAINED UNPREDICTABLE whether this bit clears to 0 or is unchanged.

This field is in the Core power domain and the Warm reset domain.

This field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- When !IsCorePowered(), or DoubleLockStatus() or EDPRSR.R == 1, access to this field is **UNKNOWN**.
- When SoftwareLockStatus(), access to this field is **RO**.
- Otherwise, access to this field is **RC**.

SPMAD, bit [10]

When ARMv8.4-Debug is implemented:

Sticky EPMAD error. Set to 1 if an external debug interface access to a Performance Monitors register returns an error because AllowExternalPMUAccess() == FALSE.

Permitted values are:

SPMAD	Meaning
0b0	No Non-secure external debug interface accesses to the external Performance Monitors registers have failed because AllowExternalPMUAccess() == FALSE for the access since EDPRSR was last read..
0b1	At least one Non-secure external debug interface access to the external Performance Monitors register has~~have~~ failed and returned an error because AllowExternalPMUAccess() == FALSE for the access since EDPRSR was last read.

If the Core power domain is powered up, then, following a read of EDPRSR:

- If ~~the OS Double Lock is not implemented or DoubleLockStatus() == FALSE, this bit clears to 0.~~ ARMv8.0-DoubleLock is not implemented or DoubleLockStatus() == FALSE, this bit clears to 0.
- If ~~the OS Double Lock is implemented, and DoubleLockStatus() == TRUE, it is~~ ARMv8.0-DoubleLock is implemented, and DoubleLockStatus() == TRUE, it is CONSTRAINED UNPREDICTABLE whether this bit clears to 0 or is unchanged.

This field is in the Core power domain.

On a Cold reset, this field resets to 0.

Accessing this field has the following behavior:

- When !IsCorePowered(), or DoubleLockStatus() or EDPRSR.R == 1, access to this field is **UNKNOWN**.
- When SoftwareLockStatus(), access to this field is **RO**.
- Otherwise, access to this field is **RC**.

Otherwise:

Sticky EPMAD error.

SPMAD	Meaning
0b0	No external debug interface accesses to the ~~external~~ Performance Monitors registers have failed because~~with~~ ~~an~~ AllowExternalPMUAccess() == FALSE ~~error~~ since EDPRSR was last read.
0b1	At least one external debug interface access to the ~~external~~ Performance Monitors registers has failed and returned an error because AllowExternalPMUAccess() == FALSE~~ error~~ since EDPRSR was last read.

If the Core power domain is powered up, then, following a read of EDPRSR:

- If ~~the OS Double Lock is not implemented or DoubleLockStatus() == FALSE, this bit clears to 0.~~ ARMv8.0-DoubleLock is not implemented or DoubleLockStatus() == FALSE, this bit clears to 0.
- If ~~the OS Double Lock is implemented, and DoubleLockStatus() == TRUE, it is~~ ARMv8.0-DoubleLock is implemented, and DoubleLockStatus() == TRUE, it is CONSTRAINED UNPREDICTABLE whether this bit clears to 0 or is unchanged.

This field is in the Core power domain.

On a Cold reset, this field resets to 0.

Accessing this field has the following behavior:

- When !IsCorePowered(), or OSLockStatus(), or DoubleLockStatus() or EDPRSR.R == 1, access to this field is **UNKNOWN**.
- When SoftwareLockStatus(), access to this field is **RO**.
- Otherwise, access to this field is **RC**.

EPMAD, bit [9]

When ARMv8.4-Debug is implemented:

External Performance Monitors Access~~access~~ Disable~~disable~~ status.

EPMAD	Meaning
0b0	External Non-secure Performance Monitors access enabled. AllowExternalPMUAccess() == TRUE.
0b1	External Non-secure Performance Monitors access disabled. AllowExternalPMUAccess() == FALSE.

This field is in the Core power domain.

Accessing this field has the following behavior:

- When !IsCorePowered(), or DoubleLockStatus() or EDPRSR.R == 1, access to this field is **UNKNOWN**.
- Otherwise, access to this field is **RO**.

Otherwise:

External Performance Monitors access disable status.

EPMAD	Meaning
0b0	External Performance Monitors access enabled. AllowExternalPMUAccess() == TRUE.
0b1	External Performance Monitors access disabled. AllowExternalPMUAccess() == FALSE.

This field is in the Core power domain.

Accessing this field has the following behavior:

- When !IsCorePowered(), or OSLockStatus(), or DoubleLockStatus() or EDPRSR.R == 1, access to this field is **UNKNOWN**.
- Otherwise, access to this field is **RO**.

SDAD, bit [8]

When ARMv8.4-Debug is implemented:

Sticky EDAD error. Set to 1 if an external debug interface access to a debug register returns an error because AllowExternalDebugAccess() == FALSE.

SDAD	Meaning
0b0	No Non-secure external debug interface accesses to the ~~external~~ debug registers have failed because~~with~~ AllowExternalDebugAccess() == FALSE for the access since EDPRSR was last read.
0b1	At least one Non-secure external debug interface access to the ~~external~~ debug registers has failed and~~with~~ returned an error because AllowExternalDebugAccess() == FALSE for the access since EDPRSR was last read.

If the Core power domain is powered up, then, following a read of EDPRSR:

- If ~~the OS Double Lock is not implemented or DoubleLockStatus() == FALSE this bit clears to 0.~~ ARMv8.0-DoubleLock is not implemented or DoubleLockStatus() == FALSE this bit clears to 0.

- If ~~the OS Double Lock is implemented and DoubleLockStatus() == TRUE, it is~~ ARMv8.0-DoubleLock is implemented and DoubleLockStatus() == TRUE, it is CONSTRAINED UNPREDICTABLE whether this bit clears to 0 or is unchanged.

This field is in the Core power domain.

On a Cold reset, this field resets to `0`.

Accessing this field has the following behavior:

- When !IsCorePowered(), or DoubleLockStatus() or EDPRSR.R == 1, access to this field is **UNKNOWN**.
- Otherwise, access to this field is **RO**.

Otherwise:

Sticky EDAD error. Set to 1 if an external debug interface access to a debug register returns an error because AllowExternalDebugAccess() == FALSE.

SDAD	Meaning
`0b0`	No external debug interface accesses to the ~~external~~ debug registers have failed because~~with~~ AllowExternalDebugAccess() == FALSE since EDPRSR was last read.
`0b1`	At least one external debug interface access to the ~~external~~ debug registers has failed and returned an error because~~with~~ AllowExternalDebugAccess() == FALSE since EDPRSR was last read.

If the Core power domain is powered up, then, following a read of EDPRSR:

- If ~~the OS Double Lock is not implemented or DoubleLockStatus() == FALSE this bit clears to 0.~~ ARMv8.0-DoubleLock is not implemented or DoubleLockStatus() == FALSE this bit clears to 0.
- If ~~the OS Double Lock is implemented and DoubleLockStatus() == TRUE, it is~~ ARMv8.0-DoubleLock is implemented and DoubleLockStatus() == TRUE, it is CONSTRAINED UNPREDICTABLE whether this bit clears to 0 or is unchanged.

This bit is UNKNOWN on reads if OSLockStatus() == TRUE and external debug writes to OSLAR_EL1 do not return an error when AllowExternalDebugAccess() == FALSE.

This field is in the Core power domain.

On a Cold reset, this field resets to `0`.

Accessing this field has the following behavior:

- When !IsCorePowered(), or DoubleLockStatus() or EDPRSR.R == 1, access to this field is **UNKNOWN**.
- Otherwise, access to this field is **RO**.

EDAD, bit [7]

When ARMv8.4-Debug is implemented:

External debug access disable status.

EDAD	Meaning
`0b0`	External Non-secure access to breakpoint registers, watchpoint registers, and OSLAR_EL1 ~~is~~ enabled. AllowExternalDebugAccess() == TRUE.
`0b1`	External Non-secure access to breakpoint registers, watchpoint registers, and OSLAR_EL1 disabled.~~from~~ AllowExternalDebugAccess()~~an~~ ==~~external~~ FALSE.~~debugger is not permitted.~~

This field is in the Core power domain.

Accessing this field has the following behavior:

- When !IsCorePowered(), or DoubleLockStatus() or EDPRSR.R == 1, access to this field is **UNKNOWN**.
- Otherwise, access to this field is **RO**.

When ARMv8.2-Debug is implemented:

External debug access disable status.

EDAD	Meaning
0b0	ExternalAccess accessfrom an external debugger to breakpoint registers, watchpoint registers, and OSLAR_EL1 is enabled. AllowExternalDebugAccess() == TRUE.
0b1	ExternalAccess accessfrom an external debugger to breakpoint registers, watchpoint registers, and OSLAR_EL1 disabled.is AllowExternalDebugAccess()not == FALSE.permitted.

This bit is not valid and reads UNKNOWN if OSLockStatus() == TRUE and external debug writes to OSLAR_EL1 do not return an error when AllowExternalDebugAccess() == FALSE.

This field is in the Core power domain.

Accessing this field has the following behavior:

- When !IsCorePowered(), or DoubleLockStatus() or EDPRSR.R == 1, access to this field is **UNKNOWN**.
- Otherwise, access to this field is **RO**.

Otherwise:

External debug access disable status.

EDAD	Meaning
0b0	ExternalAccess accessfrom an external debugger to breakpoint registers, watchpoint registers, andis enabled. OSLAR_EL1 enabled. AllowExternalDebugAccess() == TRUE.
0b1	ExternalAccess accessfrom an external debugger to breakpoint registers, watchpoint registers disabled.is Itnot permitted and it is IMPLEMENTATAION DEFINED whether accesses to IMPLEMENTATION DEFINED whether accesses to OSLAR_EL1 are enabled or disabled. AllowExternalDebugAccess() == FALSE.permitted.

This field is in the Core power domain.

Accessing this field has the following behavior:

- When !IsCorePowered(), or DoubleLockStatus() or EDPRSR.R == 1, access to this field is **UNKNOWN**.
- Otherwise, access to this field is **RO**.

DLK, bit [6]

When ARMv8.4-Debug is implemented:

This field is RES0.

When ARMv8.2-Debug is implemented and ARMv8.0-DoubleLock is implemented:

From Armv8.2, this field is deprecated.

This field is in the Core power domain.

Accessing this field has the following behavior:

- When IsCorePowered() and !DoubleLockStatus(), access to this field is **RAZ**.
- Otherwise, access to this field is **UNKNOWN**.

When ARMv8.0-DoubleLock is implementedOtherwise:

This field returns the result of the pseudocode function DoubleLockStatus().

If the Core power domain is powered up and the OS Double Lock is implemented and DoubleLockStatus() == TRUE, it is IMPLEMENTATION DEFINED whether:

- EDPRSR.PU reads as 1, EDPRSR.DLK reads as 1, and EDPRSR.SPD is UNKNOWN.
- EDPRSR.PU reads as 0, EDPRSR.DLK is UNKNOWN, and EDPRSR.SPD reads as 0.

This field is in the Core power domain.

DLK	Meaning
0b0	DoubleLockStatus() returns FALSE.
0b1	DoubleLockStatus() returns TRUE and the Core power domain is powered up.

Accessing this field has the following behavior:

- When !IsCorePowered(), access to this field is **UNKNOWN**.
- Otherwise, access to this field is **RO**.

Otherwise:

Reserved, RES0.

OSLK, bit [5]

OS lock status bit.

A read of this bit returns the value of OSLSR_EL1.OSLK.

This field is in the Core power domain.

Accessing this field has the following behavior:

- When !IsCorePowered(), DoubleLockStatus() and EDPRSR.R == 1, access to this field is **UNKNOWN**.
- Otherwise, access to this field is **RO**.

HALTED, bit [4]

Halted status bit.

~~This bit is UNKNOWN on reads if EDPRSR.PU is 0.~~

~~Otherwise permitted values are:~~

HALTED	Meaning
0b0	PE is in Non-debug state.
0b1	PE is in Debug state.

~~Because the OS Double Lock is never set when the PE is in Debug state, this bit is always RAZ when DoubleLockStatus() == TRUE.~~

This field is in the Core power domain.

Accessing this field has the following behavior:

- When !IsCorePowered(), access to this field is **UNKNOWN**.
- Otherwise, access to this field is **RO**.

SR, bit [3]

Sticky core reset status bit.

Permitted values are:

SR	Meaning
0b0	The non-debug logic of the PE is not in reset state and has not been reset since the last time EDPRSR was read.
0b1	The non-debug logic of the PE is in reset state or has been reset since the last time EDPRSR was read.

If EDPRSR.PU reads as 1 and EDPRSR.R reads as 0, which means that the Core power domain is in a powerup state and that the non-debug logic of the PE is not in reset state, then following a read of EDPRSR:

- If ~~the OS Double Lock is not implemented or DoubleLockStatus() == FALSE this bit clears to 0.~~ ARMv8.0-DoubleLock is not implemented or DoubleLockStatus() == FALSE this bit clears to 0.

- If ~~the OS Double Lock is implemented and DoubleLockStatus() == TRUE, it is~~ ARMv8.0-DoubleLock is implemented and DoubleLockStatus() == TRUE, it is UNPREDICTABLE whether this bit clears to 0 or is unchanged.

This field is in the Core power domain and the Warm reset domain.

This field resets to `1`.

Accessing this field has the following behavior:

- When !IsCorePowered() or DoubleLockStatus(), access to this field is **UNKNOWN**.
- When SoftwareLockStatus(), access to this field is **RO**.
- Otherwise, access to this field is **RC**.

R, bit [2]

PE reset status bit.

Permitted values are:

R	Meaning
`0b0`	The non-debug logic of the PE is not in reset state.
`0b1`	The non-debug logic of the PE is in reset state.

If ~~the OS Double Lock is implemented, the PE is in reset state, and the PE entered reset state with the OS Double Lock locked this bit has a~~ ARMv8.0-DoubleLock is implemented, the PE is in reset state, and the PE entered reset state with the OS Double Lock locked this bit has a CONSTRAINED UNPREDICTABLE value. For more information see 'EDPRSR.{DLK, R} and reset state' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section H6 (Debug Reset and Powerdown Support)

This field is in the Core power domain.

Accessing this field has the following behavior:

- When !IsCorePowered() or DoubleLockStatus(), access to this field is **UNKNOWN**.
- Otherwise, access to this field is **RO**.

SPD, bit [1]

Sticky Core powerdown status bit.

~~This bit is UNKNOWN on reads if EDPRSR.PU is 1, the OS Double Lock is implemented and DoubleLockStatus() == TRUE .~~

~~Otherwise, permitted values are:~~

SPD	Meaning
`0b0`	If EDPRSR.PU is 0, it is not known whether the state of the debug registers in the Core power domain is lost. If EDPRSR.PU is 1, the state of the debug registers in the Core power domain has not been lost.
`0b1`	The state of the debug registers in the Core power domain has been lost.

If the Core power domain is powered up, then, following a read of EDPRSR:

- If ~~the OS Double Lock is not implemented or DoubleLockStatus() == FALSE this bit clears to 0.~~ ARMv8.0-DoubleLock is not implemented or DoubleLockStatus() == FALSE this bit clears to 0.
- If ~~the OS Double Lock is implemented and DoubleLockStatus() == TRUE, it is~~ ARMv8.0-DoubleLock is implemented and DoubleLockStatus() == TRUE, it is CONSTRAINED UNPREDICTABLE whether this bit clears to 0 or is unchanged.

When the value of EDPRSR.PU is 0 indicating that the Core power domain is in either retention or powerdown state, EDPRSR.SPD reads as 0. For more information, see 'EDPRSR.SPD when the Core domain is in either retention or powerdown state' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section H6 (Debug Reset and Powerdown Support).

EDPRSR.{DLK, SPD, PU} describe whether registers in the Core power domain can be accessed, and whether their state has been lost since the last time the register was read. For more information, see 'EDPRSR.{DLK, SPD, PU} bits record accessibility and lost of state in Core power domain' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section H6 (Debug Reset and Powerdown Support).

This field is in the Core power domain and the Cold reset domain.

On a Cold reset, this field resets to `1`.

Accessing this field has the following behavior:

- When !IsCorePowered(), access to this field is **RAZ**.
- When IsCorePowered() and DoubleLockStatus(), access to this field is **UNKNOWN**.
- Otherwise, access to this field is **RO**.

PU, bit [0]

When ARMv8.3-DoPD is implemented:

Core powerup status bit.

Access to this field is **RAO**.

When ARMv8.2-Debug is implemented:

Core powerup status bit. Indicates whether the debug registers in the Core power domain can be accessed.

PU	Meaning
0b0	Either the Core power domain is in a low-power or powerdown state, or ~~the OS Double Lock is implemented and DoubleLockStatus() == TRUE, meaning the debug registers in the Core power domain cannot be accessed.~~ ARMv8.0-DoubleLock is implemented and DoubleLockStatus() == TRUE, meaning the debug registers in the Core power domain cannot be accessed.
0b1	The Core power domain is in a powerup state, and either ~~the OS Double Lock is not implemented or DoubleLockStatus() == FALSE, meaning the debug registers in the Core power domain can be accessed.~~ ARMv8.0-DoubleLock is not implemented or DoubleLockStatus() == FALSE, meaning the debug registers in the Core power domain can be accessed.

If ~~the OS Double Lock is implemented, the PE is in reset state, and the PE entered reset state with the OS Double Lock locked this bit has a~~ ARMv8.0-DoubleLock is implemented, the PE is in reset state, and the PE entered reset state with the OS Double Lock locked this bit has a CONSTRAINED UNPREDICTABLE value. For more information see 'EDPRSR.{DLK, R} and reset state' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section H6 (Debug Reset and Powerdown Support)

EDPRSR.{DLK, SPD, PU} describe whether registers in the Core power domain can be accessed, and whether their state has been lost since the last time the register was read. For more information, see 'EDPRSR.{DLK, SPD, PU} bits record accessibility and lost of state in Core power domain' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section H6 (Debug Reset and Powerdown Support)

Access to this field is **RO**.

Otherwise:

Core powerup status bit. Indicates whether the debug registers in the Core power domain can be accessed.

When the Core power domain is powered-up and DoubleLockStatus() == TRUE, then the value of EDPRSR.PU is IMPLEMENTATION DEFINED. See the description of the DLK bit for more information.

Otherwise, permitted values are:

PU	Meaning
0b0	Core power domain is in a low-power or powerdown state where the debug registers in the Core power domain cannot be accessed.
0b1	Core power domain is in a powerup state where the debug registers in the Core power domain can be accessed.

If ~~the Core power domain is powered up and DoubleLockStatus() == TRUE, it is~~ ARMv8.0-DoubleLock is implemented, the Core power domain is powered up, and DoubleLockStatus() == TRUE, it is IMPLEMENTATION DEFINED whether this bit reads as 0 or 1.

If ~~the PE is in reset state and entered reset state with the OS Double Lock locked this bit has a~~ ARMv8.0-DoubleLock is implemented, the PE is in reset state, and the PE entered reset state with the OS Double Lock locked this bit has a CONSTRAINED UNPREDICTABLE value. For more information see 'EDPRSR.{DLK, R} and reset state' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section H6 (Debug Reset and Powerdown Support)

EDPRSR.{DLK, SPD, PU} describe whether registers in the Core power domain can be accessed, and whether their state has been lost since the last time the register was read. For more information, see 'EDPRSR.{DLK, SPD, PU} bits record accessibility and lost of state in Core power domain' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section H6 (Debug Reset and Powerdown Support)

Access to this field is **RO**.

Accessing the EDPRSR

On permitted accesses to the register, other access controls affect the behavior of some fields. See the field descriptions for more information.

If the Core power domain is powered up (EDPRSR.PU == 1), then following a read of EDPRSR:

- If ~~the OS Double Lock is not implemented or DoubleLockStatus() == FALSE, then:~~ ARMv8.0-DoubleLock is not implemented or DoubleLockStatus() == FALSE, then:
    - EDPRSR.{SDR, SPMAD, SDAD, SPD} are cleared to 0.
    - EDPRSR.SR is cleared to 0 if the non-debug logic of the PE is not in reset state (EDPRSR.R == 0).
- Otherwise it is CONSTRAINED UNPREDICTABLE whether or not this clearing occurs.

If the Core power domain is powered down (EDPRSR.PU == 0), then:

- EDPRSR.{SDR, SPMAD, SDAD, SR} are all UNKNOWN, and are either reset or restored on being powered up.
- EDPRSR.SPD is not cleared following a read of EDPRSR. See the SPD bit description for more information.

The clearing of bits is an indirect write to EDPRSR.

EDPRSR can be accessed through the external debug interface:

Component	Offset	Instance
Debug	0x314	EDPRSR

This interface is accessible as follows:

- When ARMv8.3-DoPD is not implemented or IsCorePowered() access to this register is **RO**.
- Otherwise access to this register returns an Error.

2713/0312/20192018 2116:5942: e5e4db499bf9867a4b93324c4dbac985d3da93766379d01e197f1d40720d32d0f84e419e9187e009

(old) htmldiff from- (new)

EDSCR, External Debug Status and Control Register

The EDSCR characteristics are:

Purpose

Main control register for the debug implementation.

Configuration

External register EDSCR bits [30:29] are architecturally mapped to AArch64 System register MDCCSR_EL0[30:29] .

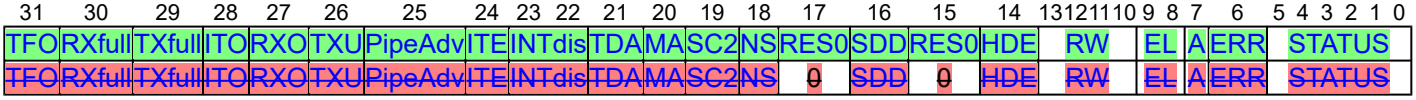
EDSCR is in the Core power domain. Some or all RW fields of this register have defined reset values. These apply only on a Cold reset. The register is not affected by a Warm reset and is not affected by an External debug reset.

Attributes

EDSCR is a 32-bit register.

Field descriptions

The EDSCR bit assignments are:



TFO, bit [31]

When ARMv8.4-Trace is implemented:

Trace Filter Override. Overrides the Trace Filter controls allowing the external debugger to trace any visible Exception level.

TFO	Meaning
0b0	Trace Filter controls are not affected.
0b1	Trace Filter controls in TRFCR_EL1, TRFCR_EL2, TRFCR, and HTRFCR are ignored.

When OSLSR_EL1.OSLK == 1, this bit can be indirectly read and written through the MDSCR_EL1 and DBGDSCRext System registers.

This bit is ignored by the PE when ExternalSecureNoninvasiveDebugEnabled() == FALSE and the Effective value of MDCR_EL3.STE == 1.

On a Cold reset, this field resets to 0.

Otherwise:

Reserved, RES0.

RXfull, bit [30]

DTRRX full. This bit is RO.

On a Cold reset, this field resets to 0.

TXfull, bit [29]

DTRTX full. This bit is RO.

On a Cold reset, this field resets to 0.

ITO, bit [28]

ITR overrun. This bit is RO.

If the PE is in Non-debug state, this bit is UNKNOWN. ITO is set to 0 on entry to Debug state.

RXO, bit [27]

DTRRX overrun. This bit is RO.

On a Cold reset, this field resets to 0.

TXU, bit [26]

DTRTX underrun. This bit is RO.

On a Cold reset, this field resets to 0.

PipeAdv, bit [25]

Pipeline advance. This bit is RO. Set to 1 every time the PE pipeline retires one or more instructions. Cleared to 0 by a write to EDRCR.CSPA.

The architecture does not define precisely when this bit is set to 1. It requires only that this happen periodically in Non-debug state to indicate that software execution is progressing.

ITE, bit [24]

ITR empty. This bit is RO.

If the PE is in Non-debug state, this bit is UNKNOWN. It is always valid in Debug state.

INTdis, bits [23:22]

When ARMv8.4-Debug is implemented:

Interrupt disable. Disables taking interrupts in Non-Debug state.

INTdis	Meaning
0b0	Masking of interrupts is controlled by PSTATE and interrupt routing controls.
0b1	If ExternalSecureDebugEnabled() == TRUE, then all interrupts, including virtual and SError interrupts, are masked. If ExternalSecureDebugEnabled() == FALSE, then all interrupts targetting Non-secure state are masked.

When OSLSR_EL1.OSLK == 1, this field can be indirectly read and written through the MDSCR_EL1 and DBGDSCRext System registers.

This field is ignored by the PE and treated as zero when ExternalDebugEnabled() == FALSE.

On a Cold reset, this field resets to 0.

Otherwise:

Interrupt disable.

When OSLSR_EL1.OSLK == 1, this field can be indirectly read and written through the MDSCR_EL1 and DBGDSCRext System registers.

INTdis	Meaning
0b00	Do not disable interrupts.
0b01	Disable interrupts taken to Non-secure EL1.
0b10	Disable interrupts taken only to Non-secure EL1 and Non-secure EL2. If ExternalSecureInvasiveDebugEnabled() == TRUE, also disable interrupts taken to Secure EL1.
0b11	Disable interrupts taken only to Non-secure EL1 and Non-secure EL2. If ExternalSecureInvasiveDebugEnabled() == TRUE, also disable all other interrupts.

On a Cold reset, this field resets to 0.

TDA, bit [21]

Traps accesses to the following debug System registers:

- AArch64: DBGBCR<n>_EL1, DBGBVR<n>_EL1, DBGWCR<n>_EL1, DBGWVR<n>_EL1.
- AArch32: DBGBCR<n>, DBGBVR<n>, DBGBXVR<n>, DBGWCR<n>, DBGWVR<n>.

The possible values of this field are:

TDA	Meaning
0b0	Accesses to debug System registers do not generate a Software Access Debug event.
0b1	Accesses to debug System registers generate a Software Access Debug event, if OSLSR_EL1.OSLK is 0 and if halting is allowed.

On a Cold reset, this field resets to 0.

MA, bit [20]

Memory access mode. Controls the use of memory-access mode for accessing ITR and the DCC. This bit is ignored if in Non-debug state and set to zero on entry to Debug state.

Possible values of this field are:

MA	Meaning
0b0	Normal access mode.
0b1	Memory access mode.

On a Cold reset, this field resets to 0.

SC2, bit [19]

When ARMv8.0-PCSample is implemented, ARMv8.1-VHE is implemented and ARMv8.2-PCSampleARMv8.1-VHE is not implemented:

Sample CONTEXTIDR_EL2. ControlsIf ARMv8.2-PCSample is not implemented, controls whether the PC Sample-based Profiling Extension samples CONTEXTIDR_EL2 or VTTBR_EL2.VMID.

SC2	Meaning
0b0	Sample VTTBR_EL2.VMID.
0b1	Sample CONTEXTIDR_EL2.

On a Cold reset, this field resets to 0.

Otherwise:

Reserved, RES0.

NS, bit [18]

Non-secure status. Read-only. When in Debug state, gives the current Security state:

NS	Meaning
0b0	Secure state, IsSecure() == TRUE.
0b1	Non-secure state, IsSecure() == FALSE.

In Non-debug state, this bit is UNKNOWN.

Bit [17]

Reserved, RES0.

SDD, bit [16]

Secure debug disabled. This bit is RO.

On entry to Debug state:

- If entering in Secure state, SDD is set to 0.
- If entering in Non-secure state, SDD is set to the inverse of ExternalSecureInvasiveDebugEnabled().

In Debug state, the value of the SDD bit does not change, even if ExternalSecureInvasiveDebugEnabled() changes.

In Non-debug state:

- SDD returns the inverse of ExternalSecureInvasiveDebugEnabled(). If the authentication signals that control ExternalSecureInvasiveDebugEnabled() change, a context synchronization event is required to guarantee their effect.
- This bit is unaffected by the Security state of the PE.

If EL3 is not implemented and the implementation is Non-secure, this bit is RES1.

Bit [15]

Reserved, RES0.

HDE, bit [14]

Halting debug enable. The possible values of this field are:

HDE	Meaning
0b0	Halting disabled for Breakpoint, Watchpoint and Halt Instruction debug events.
0b1	Halting enabled for Breakpoint, Watchpoint and Halt Instruction debug events.

On a Cold reset, this field resets to 0.

RW, bits [13:10]

Exception level Execution state status. Read-only. In Debug state, each bit gives the current Execution state of each Exception level:

RW	Meaning
0b1111	All Exception levels are using AArch64 or the PE is in Non-debug state.
0b1110	The PE is in Debug state. EL0 is using AArch32. All other Exception levels are using AArch64. Only permitted if the PE is executing at EL0.
0b110x	The PE is in Debug state. EL0 and EL1 are using AArch32. EL2 and EL3 are using AArch64. Only permitted if EL2 is implemented and enabled in the current Security state.
0b10xx	The PE is in Debug state. EL0, EL1, and, if implemented in the current Security state, EL2 are using AArch32. EL3 is using AArch64. Only permitted if EL3 is implemented.
0b0xxx	The PE is in Debug state. All Exception levels are using AArch32.

In Non-debug state, this field is RAO.

EL, bits [9:8]

Exception level. Read-only. In Debug state, this gives the current EL of the PE.

In Non-debug state, this field is RAZ.

A, bit [7]

SError interrupt pending. Read-only. In Debug state, indicates whether ana SError interrupt is pending:

- If HCR_EL2.{AMO, TGE} = {1, 0}, EL2 is enabled in the current Security state, and the PE is executing at EL0 or EL1, a virtual SError interrupt.
- Otherwise, a physical SError interrupt.

A	Meaning
0b0	No SError interrupt pending.
0b1	SError interrupt pending.

A debugger can read EDSCR to check whether an SError interrupt is pending without having to execute further instructions. A pending SError might indicate data from target memory is corrupted.

UNKNOWN in Non-debug state.

ERR, bit [6]

Cumulative error flag. This field is RO. It is set to 1 following exceptions in Debug state and on any signaled overrun or underrun on the DTR or EDITR.

On a Cold reset, this field resets to 0.

STATUS, bits [5:0]

Debug status flags. This field is RO.

The possible values of this field are:

STATUS	Meaning
0b000001	PE is restarting, exiting Debug state.
0b000010	PE is in Non-debug state.
0b000111	Breakpoint.
0b010011	External debug request.
0b011011	Halting step, normal.
0b011111	Halting step, exclusive.
0b100011	OS Unlock Catch.
0b100111	Reset Catch.
0b101011	Watchpoint.
0b101111	HLT instruction.
0b110011	Software access to debug register.
0b110111	Exception Catch.
0b111011	Halting step, no syndrome.

All other values of STATUS are reserved.

Accessing the EDSCR

EDSCR can be accessed through the external debug interface:

Component	Offset	Instance
Debug	0x088	EDSCR

This interface is accessible as follows:

- When IsCorePowered(), !DoubleLockStatus(), !OSLockStatus() and SoftwareLockStatus() access to this register is **RO**.
- When IsCorePowered(), !DoubleLockStatus(), !OSLockStatus() and !SoftwareLockStatus() access to this register is **RW**.
- Otherwise access to this register returns an Error.

27~~13~~/03~~12~~/2019~~2018~~ 21~~16~~:59~~42~~: e5e4db499bf9867a4b93324c4dbac985d3da93766~~379d01e197f1d40720d32d0f84e419e9187e009~~

(old) htmldiff from- (new)

(old) htmldiff from- (new)

MIDR_EL1, Main ID Register

The MIDR_EL1 characteristics are:

Purpose

Provides identification information for the PE, including an implementer code for the device and a device ID number.

Configuration

External register MIDR_EL1 bits [31:0] are architecturally mapped to AArch64 System register MIDR_EL1[31:0] .

External register MIDR_EL1 bits [31:0] are architecturally mapped to AArch32 System register MIDR[31:0] .

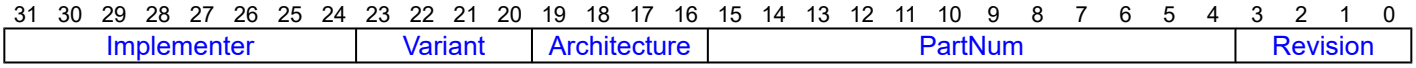
It is IMPLEMENTATION DEFINED whether MIDR_EL1 is implemented in the Core power domain or in the Debug power domain.

Attributes

MIDR_EL1 is a 32-bit register.

Field descriptions

The MIDR_EL1 bit assignments are:



Implementer, bits [31:24]

The Implementer code. This field must hold an implementer code that has been assigned by Arm. Assigned codes include the following:

Hex representation	Implementer
0x000x0	Reserved for software use
0xC0	Ampere Computing
0x41	Arm Limited
0x42	Broadcom Corporation
0x43	Cavium Inc.
0x44	Digital Equipment Corporation
0x49	Infineon Technologies AG
0x4D	Motorola or Freescale Semiconductor Inc.
0x4E	NVIDIA Corporation
0x50	Applied Micro Circuits Corporation
0x51	Qualcomm Inc.
0x56	Marvell International Ltd.
0x69	Intel Corporation

Arm can assign codes that are not published in this manual. All values not assigned by Arm are reserved and must not be used.

Variant, bits [23:20]

An IMPLEMENTATION DEFINED variant number. Typically, this field is used to distinguish between different product variants, or major revisions of a product.

Architecture, bits [19:16]

The permitted values of this field are:

Architecture	Meaning
0b0001	Armv4.
0b0010	Armv4T.
0b0011	Armv5 (obsolete).
0b0100	Armv5T.
0b0101	Armv5TE.
0b0110	Armv5TEJ.
0b0111	Armv6.
0b1111	Architectural features are individually identified in the ID_* registers, see ID registers in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile

All other values are reserved.

PartNum, bits [15:4]

An IMPLEMENTATION DEFINED primary part number for the device.

On processors implemented by Arm, if the top four bits of the primary part number are 0x0 or 0x7, the variant and architecture are encoded differently.

Revision, bits [3:0]

An IMPLEMENTATION DEFINED revision number for the device.

Accessing the MIDR_EL1

MIDR_EL1 can be accessed through the external debug interface:

Component	Offset	Instance
Debug	0xD00	MIDR_EL1

This interface is accessible as follows:

- When IsCorePowered() and !DoubleLockStatus() access to this register is **RO**.
- Otherwise access to this register is **IMPDEF**.

2713/0312/20192018 2116:5942; e5e4db499bf9867a4b93324c4dbac985d3da93766379d01e197f1d40720d32d0f84e419e9187e009

(old) htmldiff from- (new)

MPAMF_IDR, MPAM Features Identification Register

The MPAMF_IDR characteristics are:

Purpose

The MPAMF_IDR is a 32-bit read-only register that indicates which memory partitioning and monitoring features are present on this MSC.

Configuration

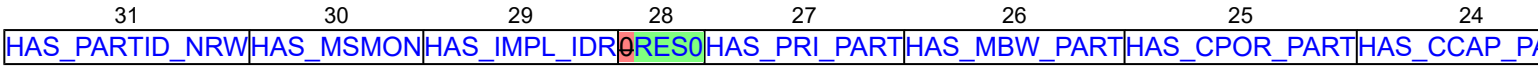
The power domain of MPAMF_IDR is IMPLEMENTATION DEFINED.

Attributes

MPAMF_IDR is a 32-bit register.

Field descriptions

The MPAMF_IDR bit assignments are:



HAS_PARTID_NRW, bit [31]

Has PARTID narrowing.

HAS_PARTID_NRW	Meaning
0b0	Does not have MPAMF_PARTID_NRW_IDR, MPAMCFG_INTPARTID or intPARTID mapping support.
0b1	Supports the MPAMF_PARTID_NRW_IDR, MPAMCFG_INTPARTID registers.

HAS_MSMON, bit [30]

Has resource monitors. Indicates whether this MSC has MPAM resource monitors.

HAS_MSMON	Meaning
0b0	Does not support MPAM resource monitoring by groups or MPAMF_MSMON_IDR.
0b1	Supports resource monitoring by matching a combination of PARTID and PMG. See MPAMF_MSMON_IDR.

HAS_IMPL_IDR, bit [29]

Has MPAMF_IMPL_IDR. Indicates whether this MSC has the implementation-specific MPAM features register, MPAMF_IMPL_IDR.

HAS_IMPL_IDR	Meaning
0b0	Does not have MPAMF_IMPL_IDR.
0b1	Has MPAMF_IMPL_IDR.

Bit [28]

Reserved, RES0.

HAS_PRI_PART, bit [27]

Has priority partitioning. Indicates whether this MSC implements MPAM priority partitioning and MPAMF_PRI_IDR.

HAS_PRI_PART	Meaning
0b0	Does not support priority partitioning or have MPAMF_PRI_IDR.
0b1	Has MPAMF_PRI_IDR.

HAS_MBW_PART, bit [26]

Has memory bandwidth partitioning. Indicates whether this MSC implements MPAM memory bandwidth~~bandwdith~~ partitioning and MPAMF_MBW_IDR.

HAS_MBW_PART	Meaning
0b0	Does not support memory bandwidth partitioning or have MPAMF_MBW_IDR register.
0b1	Has MPAMF_MBW_IDR register.

HAS_CPOR_PART, bit [25]

Has cache portion partitioning. Indicates whether this MSC implements MPAM cache portion partitioning and MPAMF_CPOR_IDR.

HAS_CPOR_PART	Meaning
0b0	Does not support cache portion partitioning or have MPAMF_CPOR_IDR or MPAMCFG_CPBM registers.
0b1	Has MPAMF_CPOR_IDR and MPAMCFG_CPBM registers.

HAS_CCAP_PART, bit [24]

Has cache capacity partitioning. Indicates whether this MSC implements MPAM cache capacity partitioning and the MPAMF_CCAP_IDR and MPAMCFG_CMAX registers.

HAS_CCAP_PART	Meaning
0b0	Does not support cache capacity partitioning or have MPAMF_CCAP_IDR and MPAMCFG_CMAX registers.
0b1	Has MPAMF_CCAP_IDR and MPAMCFG_CMAX registers.

PMG_MAX, bits [23:16]

Maximum value of Non-secure PMG supported by this component.

PARTID_MAX, bits [15:0]

Maximum value of Non-secure PARTID supported by this component.

Accessing the MPAMF_IDR

This register is part of the MPAMF_BASE memory frame. In a system that supports Secure and Non-secure memory maps, the MPAMF_BASE frame must be accessible in both Secure and Non-secure memory address maps.

MPAMF_IDR must be accessible from the Non-secure and Secure address maps.

MPAMF_IDR is permitted to be shared between the Secure and Non-secure address maps unless the register contents is different for Secure and Non-secure partitions, when the register must be banked.

MPAMF_IDR can be accessed through the memory-mapped interfaces:

Component	Frame	Offset	Instance
MPAM.any	MPAMF_BASE_s	0x0000	MPAMF_IDR_s

Access on this interface is **RO**.

Component	Frame	Offset	Instance
MPAM.any	MPAMF_BASE_ns	0x0000	MPAMF_IDR_ns

Access on this interface is **RO**.

(old) htmldiff from- (new)

(old) htmldiff from- (new)

MPAMF_MSMON_IDR, MPAM Resource Monitoring Identification Register

The MPAMF_MSMON_IDR characteristics are:

Purpose

The MPAMF_MSMON_IDR is a 32-bit read-only register that indicates which MPAM monitoring features are present on this MSC.

Configuration

The power domain of MPAMF_MSMON_IDR is IMPLEMENTATION DEFINED.

This register is present only when MPAMF_IDR.HAS_MSMON == 1. Otherwise, direct accesses to MPAMF_MSMON_IDR are IMPLEMENTATION DEFINED.

Attributes

MPAMF_MSMON_IDR is a 32-bit register.

Field descriptions

The MPAMF_MSMON_IDR bit assignments are:

31	30292827262524232221201918	17	16	15141312111098765432 10
HAS_LOCAL_CAPT_EVNT	RES0	MSMON_MBWU	MSMON_CSU	RES0
~~31~~	~~30292827262524232221201918~~	~~17~~	~~16~~	~~15141312111098765432 10~~
~~HAS_LOCAL_CAPT_EVNT~~	~~0 0 0 0 0 0 0 0 0 0 0 0 0 0~~	~~MSMON_MBWU~~	~~MSMON_CSU~~	~~0 0 0 0 0 0 0000000000~~

HAS_LOCAL_CAPT_EVNT, bit [31]

Has local capture event generator. Indicates whether this MSC has the MPAM local capture event generator and the MSMON_CAPT_EVNT register.

HAS_LOCAL_CAPT_EVNT	Meaning
0b0	Does not support MPAM local capture event generator or MSMON_CAPT_EVNT.
0b1	Supports the MPAM local capture event generator and the MSMON_CAPT_EVNT register.

Bits [30:18]

Reserved, RES0.

MSMON_MBWU, bit [17]

Memory bandwidth usage monitoring. Indicates whether this MSC has MPAM monitoring for Memory Bandwidth Usage by PARTID and PMG.

MSMON_MBWU	Meaning
0b0	Does not have monitoring for memory bandwidth usage or the MPAMF_MBWUMON_IDR, MSMON_CFG_MBWU_CTL, MSMON_CFG_MBWU_FLT, MSMON_MBWU or MSMON_MBWU_CAPTURE registers.
0b1	Has monitoring of memory bandwdith usage and the MPAMF_MBWUMON_IDR, MSMON_CFG_MBWU_CTL, MSMON_CFG_MBWU_FLT, MSMON_MBWU and optional MSMON_MBWU_CAPTURE registers.

MSMON_CSU, bit [16]

Cache storage usage monitoring. Indicates whether this MSC has MPAM monitoring of cache storage usage by PARTID and PMG.

MSMON_CSU	Meaning
0b0	Does not have monitoring for cache storage usage or the MPAMF_CSUMON_IDR, MSMON_CFG_CSU_CTL, MSMON_CFG_CSU_FLT, MSMON_CSU or MSMON_CSU_CAPTURE registers.
0b1	Has monitoring of cache storage usage and the MPAMF_CSUMON_IDR, MSMON_CFG_CSU_CTL, MSMON_CFG_CSU_FLT, MSMON_CSU and optional MSMON_CSU_CAPTURE registers.

Bits [15:0]

Reserved, RES0.

Accessing the MPAMF_MSMON_IDR

This register is part of the MPAMF_BASE memory frame. In a system that supports Secure and Non-secure memory maps, the MPAMF_BASE frame must be accessible in both Secure and Non-secure memory address maps.

MPAMF_MSMON_IDR must be accessible from the Non-secure and Secure address maps.

MPAMF_MSMON_IDR is permitted to be shared between the Secure and Non-secure address maps unless the register contents is different for Secure and Non-secure partitions, when the register must be banked.

MPAMF_MSMON_IDR can be accessed through the memory-mapped interfaces:

Component	Frame	Offset	Instance
MPAM.any	MPAMF_BASE_s	0x0080	MPAMF_MSMON_IDR_s

Access on this interface is **RO**.

Component	Frame	Offset	Instance
MPAM.any	MPAMF_BASE_ns	0x0080	MPAMF_MSMON_IDR_ns

Access on this interface is **RO**.

2713/0312/20192018 2116:5942; e5e4db499bf9867a4b93324c4dbac985d3da93766379d01e197f1d40720d32d0f84e419e9187e009

(old) htmldiff from- (new)

MSMON_CFG_CSU_CTL, MPAM Memory System Monitor Configure Cache Storage Usage Monitor Control Register

The MSMON_CFG_CSU_CTL characteristics are:

Purpose

MSMON_CFG_CSU_CTL is a 32-bit read-write register that controls the CSU monitor selected by MSMON_CFG_MON_SEL.

Configuration

The power domain of MSMON_CFG_CSU_CTL is IMPLEMENTATION DEFINED.

This register is present only when MPAMF_IDR.HAS_MSMON == 1 and MPAMF_MSMON_IDR.MSMON_CSU == 1. Otherwise, direct accesses to MSMON_CFG_CSU_CTL are IMPLEMENTATION DEFINED.

Attributes

MSMON_CFG_CSU_CTL is a 32-bit register.

Field descriptions

The MSMON_CFG_CSU_CTL bit assignments are:

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15
EN	CAPT_EVNT			CAPT_RESET	OFLOW_STATUS	OFLOW_INTR	OFLOW_FRZ	SUBTYPE				RES0		MATCH_PMG	MATCH_PARTID	
~~31~~	~~30~~	~~29~~	~~28~~	~~27~~	~~26~~	~~25~~	~~24~~	~~23~~	~~22~~	~~21~~	~~20~~	~~19~~	~~18~~	~~17~~	~~16~~	~~15~~
EN	CAPT_EVNT			CAPT_RESET	OFLOW_STATUS	OFLOW_INTR	OFLOW_FRZ	SUBTYPE				~~0~~	~~0~~	MATCH_PMG	MATCH_PARTID	~~0~~

EN, bit [31]

Enabled.

EN	Meaning
0b0	The monitor is disabled and must not collect any information.
0b1	The monitor is enabled to collect information according to its configuration.

CAPT_EVNT, bits [30:28]

Capture event selector.

Select the event that triggers capture from the following:

CAPT_EVNT	Meaning
0b000	No capture event is triggered.
0b001	External capture event 1 (optional but recommended)
0b010	External capture event 2 (optional)
0b011	External capture event 3 (optional)
0b100	External capture event 4 (optional)
0b101	External capture event 5 (optional)
0b110	External capture event 6 (optional)
0b111	Capture occurs when a MSMON_CAPT_EVNT register in this MSC is written and causes a capture event for the security state of this monitor. (optional)

The values marked as optional indicate capture event sources that can be omitted in an implementation. Those values representing non-implemented event sources should not trigger a capture event.

If capture is not implemented for the CSU monitor type as indicated by MPAMF_CSUMON_IDR.HAS_CAPTURE = 0, this field is RAZ/WI.

CAPT_RESET, bit [27]

Reset after capture.

Controls whether the value of MSMON_CSU is reset to zero immediately after being copied to MSMON_CSU_CAPTURE.

CAPT_RESET	Meaning
0b0	Monitor is not reset on capture.
0b1	Monitor is reset on capture.

If capture is not implemented for the CSU monitor type as indicated by MPAMF_CSUMON_IDR.HAS_CAPTURE = 0, this field is RAZ/WI.

Because the CSU monitor type produces a measurement rather than a count, it might not make sense to ever reset the value after a capture. If there is no reason to ever reset a CSU monitor, this field is RAZ/WI.

OFLOW_STATUS, bit [26]

Overflow status.

Indicates whether the value of MSMON_CSU has overflowed.

OFLOW_STATUS	Meaning
0b0	No overflow has occurred.
0b1	At least one overflow has occurred since this bit was last written to zero.

If overflow is not possible for a CSU monitor in the implementation, this field is RAZ/WI.

OFLOW_INTR, bit [25]

Overflow Interrupt.

Indicates whether the value of MSMON_CSU has overflowed.

OFLOW_INTR	Meaning
0b0	No interrupt is signaled on an overflow of MSMON_CSU.
0b1	On overflow, an implementation-specific interrupt is signaled.

If OFLOW_INTR is not supported by the implementation, this field is RAZ/WI.

OFLOW_FRZ, bit [24]

Freeze Monitor on Overflow.

Controls whether the value of MSMON_CSU freezes on an overflow.

OFLOW_FRZ	Meaning
0b0	Monitor count wraps on overflow.
0b1	Monitor count freezes on overflow. The frozen value might be 0 or another value if the monitor overflowed with an increment larger than 1.

If overflow is not possible for a CSU monitor in the implementation, this field is RAZ/WI.

SUBTYPE, bits [23:20]

Subtype.

A monitor can have other event matching criteria.

This field is not currently used for CSU monitors, but reserved for future use.

This field is RAZ/WI.

Bits [19:18]

Reserved, RES0.

MATCH_PMG, bit [17]

Match PMG.

Controls whether the monitor measures only storage used with PMG matching MSMON_CFG_CSU_FLT.PMG.

MATCH_PMG	Meaning
0b0	The monitor measures storage used with any PMG value.
0b1	The monitor only measures storage used with the PMG value matching MSMON_CFG_CSU_FLT.PMG.

If MATCH_PMG == 1 and MATCH_PARTID == 0, it is CONSTRAINED UNPREDICTABLE whether the monitor instance:

- Measures the storage used with matching PMG and with any PARTID.
- Measures no storage usage, that is, the monitor's VALUE field is zero.
- Measures the storage used with matching PMG and PARTID, that is, treats MATCH_PARTID as == 1.

MATCH_PARTID, bit [16]

Match PARTID.

Controls whether the monitor measures only storage used with PARTID matching MSMON_CFG_CSU_FLT.PARTID.

MATCH_PARTID	Meaning
0b0	The monitor measures storage used with any PARTID value.
0b1	The monitor only measures storage used with the PARTID value matching MSMON_CFG_CSU_FLT.PARTID.

Bits [15:8]

Reserved, RES0.

TYPE, bits [7:0]

Monitor Type Code.

Constant type indicating the type of the monitor.

Read-only.

CSU monitor is TYPE = 0x43.

Accessing the MSMON_CFG_CSU_CTL

This register is part of the MPAMF_BASE memory frame. In a system that supports Secure and Non-secure memory maps, the MPAMF_BASE frame must be accessible in both Secure and Non-secure memory address maps.

MSMON_CFG_CSU_CTL must be accessible from the Non-secure and Secure address maps.

MSMON_CFG_CSU_CTL must be banked for the Secure and Non-secure address maps. The Secure instance accesses the cache storage usage monitor controls used for Secure PARTIDs, and the Non-secure instance accesses the cache storage usage monitor controls used for Non-secure PARTIDs.

MSMON_CFG_CSU_CTL can be accessed through the memory-mapped interfaces:

Component	Frame	Offset	Instance

MPAM.any	MPAMF_BASE_s	0x0818	MSMON_CFG_CSU_CTL_s

Access on this interface is **RW**.

Component	Frame	Offset	Instance
MPAM.any	MPAMF_BASE_ns	0x0818	MSMON_CFG_CSU_CTL_ns

Access on this interface is **RW**.

(old) htmldiff from- (new)

MSMON_CFG_CSU_FLT, MPAM Memory System Monitor Configure Cache Storage Usage Monitor Filter Register

The MSMON_CFG_CSU_FLT characteristics are:

Purpose

MSMON_CFG_CSU_FLT is a 32-bit read-write register that sets PARTID and PMG to measure or count in the CSU monitor selected by MSMON_CFG_MON_SEL.

Configuration

The power domain of MSMON_CFG_CSU_FLT is IMPLEMENTATION DEFINED.

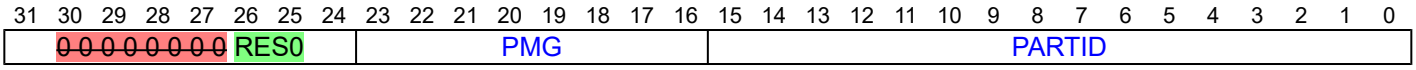
This register is present only when MPAMF_IDR.HAS_MSMON == 1 and MPAMF_MSMON_IDR.MSMON_CSU == 1. Otherwise, direct accesses to MSMON_CFG_CSU_FLT are IMPLEMENTATION DEFINED.

Attributes

MSMON_CFG_CSU_FLT is a 32-bit register.

Field descriptions

The MSMON_CFG_CSU_FLT bit assignments are:



Bits [31:24]

Reserved, RES0.

PMG, bits [23:16]

Performance monitoring group to filter cache storage usage monitoring.

If MSMON_CFG_CSU_CTL.MATCH_PMG == 0, this field is not used to match cache storage to a PMG and the contents of this field is ignored.

If MSMON_CFG_CSU_CTL.MATCH_PMG == 1, andthe monitor selected by MSMON_CFG_CSU_CTL.MATCH_PARTID == 1, the monitor instance selected by MSMON_CFG_MON_SEL measures or counts cache storage labeled with PMG equal to this field and PARTID equal to the PARTID field.

If MSMON_CFG_CSU_CTL.MATCH_PMG == 1 and MSMON_CFG_CSU_CTL.MATCH_PARTID == 0, the behavior of the monitor instance selected by MSMON_CFG_MON_SEL is CONSTRAINED UNPREDICTABLE. See MSMON_CFG_CSU_CTL.MATCH_PMG for more information.

PARTID, bits [15:0]

Partition ID to filter cache storage usage monitoring.

If MSMON_CFG_CSU_CTL.MATCH_PARTID == 0, this field is not used to match cache storage to a PARTID and the contents of this field is ignored. MSMON_CFG_CSU_CTL.MATCH_PMG == 0, the monitor measures all allocated cache storage.

If MSMON_CFG_CSU_CTL.MATCH_PARTID == 01, andthe monitor selected by MSMON_CFG_CSU_CTLMSMON_CFG_MON_SEL.MATCH_PMG == 1, the monitor's behavior is CONSTRAINED UNPREDICTABLE. See the description of MSMON_CFG_CSU_CTL.MATCH_PMG.measures or counts cache storage labeled with PARTID equal to this field.

If MSMON_CFG_CSU_CTL.MATCH_PARTID == 1 and MSMON_CFG_CSU_CTL.MATCH_PMG == 0, the monitor selected by MSMON_CFG_MON_SEL measures or counts cache storage labeled with PARTID equal to this field.

If MSMON_CFG_CSU_CTL.MATCH_PARTID == 1 and MSMON_CFG_CSU_CTL.MATCH_PMG == 1, the monitor selected by MSMON_CFG_MON_SEL measures or counts cache storage labeled with PARTID equal to this field and PMG equal to the PMG field.

Accessing the MSMON_CFG_CSU_FLT

This register is part of the MPAMF_BASE memory frame. In a system that supports Secure and Non-secure memory maps, the MPAMF_BASE frame must be accessible in both Secure and Non-secure memory address maps.

MSMON_CFG_CSU_FLT must be accessible from the Non-secure and Secure address maps.

MSMON_CFG_CSU_FLT must be banked for the Secure and Non-secure address maps. The Secure instance accesses the PARTID and PMG matching for a cache storage usage monitor used for Secure PARTIDs, and the Non-secure instance accesses the PARTID and PMG matching for a cache storage usage monitor used for Non-secure PARTIDs.

MSMON_CFG_CSU_FLT can be accessed through the memory-mapped interfaces:

Component	Frame	Offset	Instance
MPAM.any	MPAMF_BASE_s	0x0810	MSMON_CFG_CSU_FLT_s

Access on this interface is **RW**.

Component	Frame	Offset	Instance
MPAM.any	MPAMF_BASE_ns	0x0810	MSMON_CFG_CSU_FLT_ns

Access on this interface is **RW**.

27~~13~~/03~~12~~/2019~~2018~~ 21~~16~~:59~~42~~; e5e4db499bf9867a4b93324c4dbac985d3da93766379d01e197f1d40720d32d0f84e419e9187e009

(old) htmldiff from- (new)

MSMON_CFG_MBWU_CTL, MPAM Memory System Monitor Configure Memory Bandwidth Usage Monitor Control Register

The MSMON_CFG_MBWU_CTL characteristics are:

Purpose

MSMON_CFG_MBWU_CTL is a 32-bit read-write register that controls the MBWU monitor selected by MSMON_CFG_MON_SEL.

Configuration

The power domain of MSMON_CFG_MBWU_CTL is IMPLEMENTATION DEFINED.

This register is present only when MPAMF_IDR.HAS_MSMON == 1 and MPAMF_MSMON_IDR.MSMON_MBWU == 1. Otherwise, direct accesses to MSMON_CFG_MBWU_CTL are IMPLEMENTATION DEFINED.

Attributes

MSMON_CFG_MBWU_CTL is a 32-bit register.

Field descriptions

The MSMON_CFG_MBWU_CTL bit assignments are:

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15
EN	CAPT_EVNT			CAPT_RESET	OFLOW_STATUS	OFLOW_INTR	OFLOW_FRZ	SUBTYPE				RES0		MATCH_PMG	MATCH_PARTID	
~~31~~	~~30~~	~~29~~	~~28~~	~~27~~	~~26~~	~~25~~	~~24~~	~~23~~	~~22~~	~~21~~	~~20~~	~~19~~	~~18~~	~~17~~	~~16~~	~~15~~
EN	CAPT_EVNT			CAPT_RESET	OFLOW_STATUS	OFLOW_INTR	OFLOW_FRZ	SUBTYPE				~~0~~	~~0~~	MATCH_PMG	MATCH_PARTID	~~0~~

EN, bit [31]

Enabled.

EN	Meaning
0b0	The monitor is disabled and must not collect any information.
0b1	The monitor is enabled to collect information according to its configuration.

CAPT_EVNT, bits [30:28]

Capture event selector.

Select the event that triggers capture from the following:

CAPT_EVNT	Meaning
0b000	No capture event is triggered.
0b001	External capture event 1 (optional but recommended)
0b010	External capture event 2 (optional)
0b011	External capture event 3 (optional)
0b100	External capture event 4 (optional)
0b101	External capture event 5 (optional)
0b110	External capture event 6 (optional)
0b111	Capture occurs when a MSMON_CAPT_EVNT register in this MSC is written and causes a capture event for the security state of this monitor. (optional)

The values marked as optional indicate capture event sources that can be omitted in an implementation. Those values representing non-implemented event sources should not trigger a capture event.

If capture is not implemented for the MBWU monitor type as indicated by MPAMF_MBWUMON_IDR.HAS_CAPTURE = 0, this field is RAZ/WI.

CAPT_RESET, bit [27]

Reset after capture.

Controls whether the value of MSMON_MBWU is reset to zero immediately after being copied to MSMON_MBWU_CAPTURE.

CAPT_RESET	Meaning
0b0	Monitor is not reset on capture.
0b1	Monitor is reset on capture.

If capture is not implemented for the MBWU monitor type as indicated by MPAMF_MBWUMON_IDR.HAS_CAPTURE = 0, this field is RAZ/WI.

OFLOW_STATUS, bit [26]

Overflow status.

Indicates whether the value of MSMON_MBWU has overflowed.

OFLOW_STATUS	Meaning
0b0	No overflow has occurred.
0b1	At least one overflow has occurred since this bit was last written to zero.

If overflow is not possible for a MBWU monitor in the implementation, this field is RAZ/WI.

OFLOW_INTR, bit [25]

Overflow Interrupt.

Indicates whether the value of MSMON_MBWU has overflowed.

OFLOW_INTR	Meaning
0b0	No interrupt is signaled on an overflow of MSMON_MBWU.
0b1	On overflow, an implementation-specific interrupt is signaled.

If OFLOW_INTR is not supported by the implementation, this field is RAZ/WI.

OFLOW_FRZ, bit [24]

Freeze Monitor on Overflow.

Controls whether the value of MSMON_MBWU freezes on an overflow.

OFLOW_FRZ	Meaning
0b0	Monitor count wraps on overflow.
0b1	Monitor count freezes on overflow. The frozen value might be 0 or another value if the monitor overflowed with an increment larger than 1.

If overflow is not possible for a MBWU monitor in the implementation, this field is RAZ/WI.

SUBTYPE, bits [23:20]

Subtype.

A monitor can have other event matching criteria.

This field is not currently used for MBWU monitors, but reserved for future use.

This field is RAZ/WI.

Bits [19:18]

Reserved, RES0.

MATCH_PMG, bit [17]

Match PMG.

Controls whether the monitor measures only datastorage transferredused with PMG matching MSMON_CFG_MBWU_FLT.PMG.

MATCH_PMG	Meaning
0b0	The monitor measures datastorage transferredused with any PMG value.
0b1	The monitor only measures datastorage transferredused with the PMG value matching MSMON_CFG_MBWU_FLT.PMG.

MATCH_PARTID, bit [16]

Match PARTID.

Controls whether the monitor measures only datastorage transferredused with PARTID matching MSMON_CFG_MBWU_FLT.PARTID.

MATCH_PARTID	Meaning
0b0	The monitor measures datastorage transferredused with any PARTID value.
0b1	The monitor only measures datastorage transferredused with the PARTID value matching MSMON_CFG_MBWU_FLT.PARTID.

Bits [15:8]

Reserved, RES0.

TYPE, bits [7:0]

Monitor Type Code.

Constant type indicating the type of the monitor.

Read-only.

MBWU monitor is TYPE = 0x42.

Accessing the MSMON_CFG_MBWU_CTL

This register is part of the MPAMF_BASE memory frame. In a system that supports Secure and Non-secure memory maps, the MPAMF_BASE frame must be accessible in both Secure and Non-secure memory address maps.

MSMON_CFG_MBWU_CTL must be accessible from the Non-secure and Secure address maps.

MSMON_CFG_MBWU_CTL must be banked for the Secure and Non-secure address maps. The Secure instance accesses the memory bandwidth usage monitor controls used for Secure PARTIDs, and the Non-secure instance accesses the memory bandwidth usage monitor controls used for Non-secure PARTIDs.

MSMON_CFG_MBWU_CTL can be accessed through the memory-mapped interfaces:

Component	Frame	Offset	Instance
MPAM.any	MPAMF_BASE_s	0x0828	MSMON_CFG_MBWU_CTL_s

Access on this interface is **RW**.

Component	Frame	Offset	Instance
MPAM.any	MPAMF_BASE_ns	0x0828	MSMON_CFG_MBWU_CTL_ns

Access on this interface is **RW**.

(old) htmldiff from- (new)

(old) htmldiff from- (new)

MSMON_MBWU, MPAM Memory Bandwdith Usage Monitor Register

The MSMON_MBWU characteristics are:

Purpose

MSMON_MBWU is a 32-bit read-write register that accesses the monitor instance selected by MSMON_CFG_MON_SEL.

Configuration

The power domain of MSMON_MBWU is IMPLEMENTATION DEFINED.

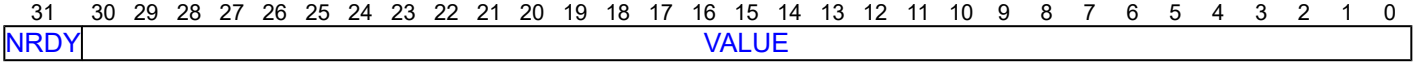
This register is present only when MPAMF_IDR.HAS_MSMON == 1 and MPAMF_MSMON_IDR.MSMON_MBWU == 1. Otherwise, direct accesses to MSMON_MBWU are IMPLEMENTATION DEFINED.

Attributes

MSMON_MBWU is a 32-bit register.

Field descriptions

The MSMON_MBWU bit assignments are:



NRDY, bit [31]

Not Ready. Indicates whether the monitor has possibly inaccurate data.

NRDY	Meaning
0b0	The monitor is not ready and the contents of the VALUE field might be inaccurate or otherwise not represent the actual memory bandwidth usage.
0b1	The monitor is ready and the VALUE fields is accurate.

VALUE, bits [30:0]

Memory bandwidth usage counter value if NRDY == 0. Invalid if NRDY == 1.

VALUE is the scaledmemory countbandwidth ofusage in bytes transferredper sincesecond the monitor was last reset that meetmeeting the criteria set in MSMON_CFG_MBWU_FLT and MSMON_CFG_MBWU_CTL for the monitor instance selected by MSMON_CFG_MON_SEL.

Accessing the MSMON_MBWU

This register is part of the MPAMF_BASE memory frame. In a system that supports Secure and Non-secure memory maps, the MPAMF_BASE frame must be accessible in both Secure and Non-secure memory address maps.

MSMON_MBWU must be accessible from the Non-secure and Secure address maps.

MSMON_MBWU must be banked for the Secure and Non-secure address maps. The Secure instance accesses the memory bandwidth usage monitor used for Secure PARTIDs, and the Non-secure instance accesses the memory bandwidth usage monitor used for Non-secure PARTIDs.

MSMON_MBWU can be accessed through the memory-mapped interfaces:

Component	Frame	Offset	Instance
MPAM.any	MPAMF_BASE_s	0x0860	MSMON_MBWU_s

Access on this interface is **RW**.

Component	Frame	Offset	Instance
MPAM.any	MPAMF_BASE_ns	0x0860	MSMON_MBWU_ns

Access on this interface is **RW**.

(old) htmldiff from- (new)

MSMON_MBWU_CAPTURE, MPAM Memory Bandwidth Usage Monitor Capture Register

The MSMON_MBWU_CAPTURE characteristics are:

Purpose

MSMON_MBWU_CAPTURE is a 32-bit read-write register that accesses the captured MSMON_MBWU monitor instance selected by MSMON_CFG_MON_SEL.

Configuration

The power domain of MSMON_MBWU_CAPTURE is IMPLEMENTATION DEFINED.

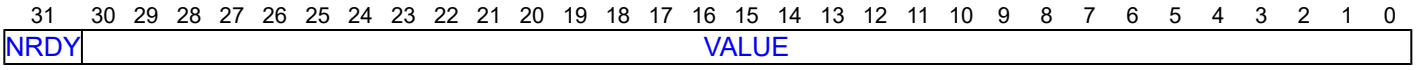
This register is present only when MPAMF_IDR.HAS_MSMON == 1, MPAMF_MSMON_IDR.MSMON_MBWU == 1 and MPAMF_MBWUMON_IDR.HAS_CAPTURE == 1. Otherwise, direct accesses to MSMON_MBWU_CAPTURE are IMPLEMENTATION DEFINED.

Attributes

MSMON_MBWU_CAPTURE is a 32-bit register.

Field descriptions

The MSMON_MBWU_CAPTURE bit assignments are:



NRDY, bit [31]

Not Ready. The captured NRDY bit from the corresponding instance of MSMON_MBWU. This bit indicatesIndicates whether the captured monitor value has possibly inaccurate data.

NRDY	Meaning
0b0	The captured monitor was not ready and the contents of the VALUE field might be inaccurate or otherwise not represent the actual memory bandwidth usage.
0b1	The captured monitor was ready and the VALUE fields is accurate.

VALUE, bits [30:0]

Captured memory bandwidth usage counter value if NRDY == 0. Invalid if NRDY == 1.

VALUE is the captured VALUEmemory fieldbandwidth fromusage thein corresponding instance of MSMON_MBWU, the scaled count of bytes transferred since the monitor was last reset that meetmeeting the criteria set in MSMON_CFG_MBWU_FLT and MSMON_CFG_MBWU_CTL for the monitor instance selected by MSMON_MBWU monitor instance selected by MSMON_CFG_MON_SEL.

Accessing the MSMON_MBWU_CAPTURE

This register is part of the MPAMF_BASE memory frame. In a system that supports Secure and Non-secure memory maps, the MPAMF_BASE frame must be accessible in both Secure and Non-secure memory address maps.

MSMON_MBWU_CAPTURE must be accessible from the Non-secure and Secure address maps.

MSMON_MBWU_CAPTURE must be banked for the Secure and Non-secure address maps. The Secure instance accesses the captured memory bandwidth usage monitor used for Secure PARTIDs, and the Non-secure instance accesses the captured memory bandwidth usage monitor used for Non-secure PARTIDs.

MSMON_MBWU_CAPTURE can be accessed through the memory-mapped interfaces:

Component	Frame	Offset	Instance
MPAM.any	MPAMF_BASE_s	0x0868	MSMON_MBWU_CAPTURE_s

Access on this interface is **RW**.

Component	Frame	Offset	Instance
MPAM.any	MPAMF_BASE_ns	0x0868	MSMON_MBWU_CAPTURE_ns

Access on this interface is **RW**.

2713/0312/20192018 2116:5942; e5e4db499bf9867a4b93324c4dbac985d3da93766379d01e197f1d40720d32d0f84e419e9187e009

(old) htmldiff from- (new)

PMCCNTR_EL0, Performance Monitors Cycle Counter

The PMCCNTR_EL0 characteristics are:

Purpose

Holds the value of the processor Cycle Counter, CCNT, that counts processor clock cycles. See 'Time as measured by the Performance Monitors cycle counter' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile~~, section D5~~ for more information.

PMCCFILTR_EL0 determines the modes and states in which the PMCCNTR_EL0 can increment.

Configuration

External register PMCCNTR_EL0 bits [63:0] are architecturally mapped to AArch64 System register PMCCNTR_EL0[63:0] .

External register PMCCNTR_EL0 bits [63:0] are architecturally mapped to AArch32 System register PMCCNTR[63:0] .

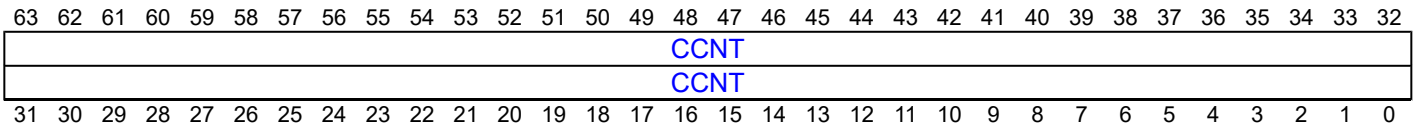
PMCCNTR_EL0 is in the Core power domain. RW fields in this register reset to architecturally UNKNOWN values. These apply on a Warm or Cold reset. The register is not affected by an External debug reset.

Attributes

PMCCNTR_EL0 is a 64-bit register.

Field descriptions

The PMCCNTR_EL0 bit assignments are:



CCNT, bits [63:0]

Cycle count. Depending on the values of PMCR_EL0.{LC,D}, the cycle count increments in one of the following ways:

- Every processor clock cycle.
- Every 64th processor clock cycle.

Writing 1 to PMCR_EL0.C sets this field to 0.

On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing the PMCCNTR_EL0

Note

SoftwareLockStatus() depends on the type of access attempted and AllowExternalPMUAccess() has a new definition from Armv8.4. Refer to the Pseudocode definitions for more information.

PMCCNTR_EL0 can be accessed through the external debug interface:

Component	Offset	Instance	Range

PMU	0x0F8	PMCCNTR_EL0	31:0

This interface is accessible as follows:

- When IsCorePowered(), !DoubleLockStatus(), !OSLockStatus(), AllowExternalPMUAccess() and SoftwareLockStatus() access to this register is **RO**.
- When IsCorePowered(), !DoubleLockStatus(), !OSLockStatus(), AllowExternalPMUAccess() and !SoftwareLockStatus() access to this register is **RW**.
- Otherwise access to this register returns an Error.

Component	Offset	Instance	Range
PMU	0x0FC	PMCCNTR_EL0	63:32

This interface is accessible as follows:

- When IsCorePowered(), !DoubleLockStatus(), !OSLockStatus(), AllowExternalPMUAccess() and SoftwareLockStatus() access to this register is **RO**.
- When IsCorePowered(), !DoubleLockStatus(), !OSLockStatus(), AllowExternalPMUAccess() and !SoftwareLockStatus() access to this register is **RW**.
- Otherwise access to this register returns an Error.

27~~13~~/03~~12~~/2019~~2018~~ 21~~16~~:59~~42~~; e5e4db499bf9867a4b93324c4dbac985d3da93766~~379d01e197f1d40720d32d0f84e419e9187e009~~

(old) htmldiff from- (new)

PMCR_EL0, Performance Monitors Control Register

The PMCR_EL0 characteristics are:

Purpose

Provides details of the Performance Monitors implementation, including the number of counters implemented, and configures and controls the counters.

Configuration

External register PMCR_EL0 bits [76:0] are architecturally mapped to AArch32 System register PMCR[76:0] .

External register PMCR_EL0 bits [76:0] are architecturally mapped to AArch64 System register PMCR_EL0[76:0] .

PMCR_EL0 is in the Core power domain. Some or all RW fields of this register have defined reset values. The field descriptions identify when the reset values apply.

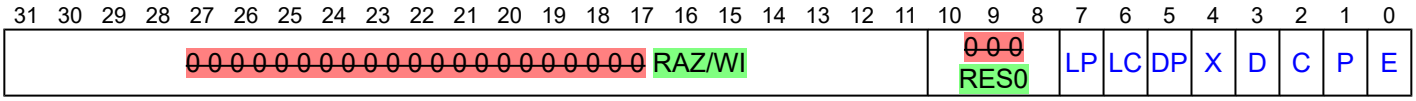
This register is only partially mapped to the internal PMCR System register. An external agent must use other means to discover the information held in PMCR[31:11], such as accessing PMCFGR and the ID registers.

Attributes

PMCR_EL0 is a 32-bit register.

Field descriptions

The PMCR_EL0 bit assignments are:



Bits [31:11]

Reserved~~RAZ/WI. Hardware must implement this field as RAZ/WI. Software must not rely on the register reading as zero, and must use a read-modify-write sequence to write to the register.~~ RAZ/WI.

Hardware must implement this field as RAZ/WI. Software must not rely on the register reading as zero, and must use a read-modify-write sequence to write to the register.

Bits [10:8]

Reserved, RES0.

LP, bit [7]

When ARMv8.5-PMU is implemented:

Long event counter enable. Determines when unsigned overflow is recorded by a counter overflow bit.

LP	Meaning
0b0	Event counter overflow on increment that causes unsigned overflow of PMEVCNTR<n>_EL0[31:0].
0b1	Event counter overflow on increment that causes unsigned overflow of PMEVCNTR<n>_EL0[63:0].

If EL2 is implemented and MDCR_EL2.HPMN or HDCR.HPMN is less than PMCR_EL0.N, this bit does not affect the operation of event counters in the range [HDCR.HPMN:(PMCR_EL0.N-1)] or [MDCR_EL2.HPMN:(PMCR_EL0.N-1)].

Note

The effect of MDCR_EL2.HPMN or HDCR.HPMN on the operation of this bit applies if EL2 is implemented regardless of whether EL2 is enabled in the current Security state. For more information, see the description of MDCR_EL2.HPMN or HDCR.HPMN.

If the highest implemented Exception level is using AArch32, it is IMPLEMENTATION DEFINED whether this bit is RW or RAZ/WI.

Otherwise:

Reserved, RES0.

LC, bit [6]

Long cycle counter enable. Determines when unsigned overflow is recorded by the cycle counter overflow bit.

LC	Meaning
0b0	Cycle counter overflow on increment that causes unsigned overflow of PMCCNTR_EL0[31:0].
0b1	Cycle counter overflow on increment that causes unsigned overflow of PMCCNTR_EL0[63:0].

Arm deprecates use of PMCR_EL0.LC = 0.

In an AArch64 only implementation, this field is RES1.

On a Warm reset, this field resets to an architecturally UNKNOWN value.

DP, bit [5]

Disable cycle counter when event counting is prohibited. The possible values of this bit are:

DP	Meaning
0b0	Cycle counting by PMCCNTR_EL0 is~~,~~ not~~if~~ affected~~enabled,~~ by~~counts~~ this~~when~~ bit.~~event counting is prohibited.~~
0b1	When event counting for counters in the range [0..(MDCR_EL2.HPMN-1)] is prohibited, cycle counting by PMCCNTR_EL0 ~~does not count when event counting~~ is disabled.~~prohibited.~~

~~Counting events is never prohibited in Non-secure state. However, there are some restrictions on counting events in Secure state.~~For more information about the interaction between the Performance Monitors and EL3, see 'Effect~~Interaction~~ of~~with~~ EL3 and EL2' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile~~, section D5.5.1~~.

When EL3 is not implemented, this field is RES0:

- When ARMv8.1-PMU is not implemented.
- When ARMv8.1-PMU is implemented, only if EL2 is not implemented.

Otherwise this field is RW.

When this register has an architecturally-defined reset value, if this field is implemented as an RW field it resets to:

- A value that is architecturally UNKNOWN if the reset is into an Exception level that is using AArch64.
- 0 if the reset is into an Exception level that is using AArch32.

X, bit [4]

Enable export of events in an IMPLEMENTATION DEFINED event stream. The possible values of this bit are:

X	Meaning
0b0	Do not export events.
0b1	Export events where not prohibited.

This field enables the exporting of events over an event bus to another device, for example to an OPTIONAL PE trace unit. If the implementation does not include such an event bus then this field is RAZ/WI, otherwise it is an RW field.

In an implementation that includes an event bus, no events are exported when counting is prohibited.

This field does not affect the generation of Performance Monitors overflow interrupt requests or signaling to a cross-trigger interface (CTI) that can be implemented as signals exported from the PE.

When this register has an architecturally-defined reset value, if this field is implemented as an RW field it resets to:

- A value that is architecturally UNKNOWN if the reset is into an Exception level that is using AArch64.
- 0 if the reset is into an Exception level that is using AArch32.

D, bit [3]

Clock divider. The possible values of this bit are:

D	Meaning
0b0	When enabled, PMCCNTR_EL0 counts every clock cycle.
0b1	When enabled, PMCCNTR_EL0 counts once every 64 clock cycles.

In an AArch64 only implementation this field is RES0, otherwise it is an RW field. If PMCR_EL0.LC == 1, this bit is ignored and the cycle counter counts every clock cycle.

Arm deprecates use of PMCR_EL0.D = 1.

When this register has an architecturally-defined reset value, if this field is implemented as an RW field it resets to:

- A value that is architecturally UNKNOWN if the reset is into an Exception level that is using AArch64.
- 0 if the reset is into an Exception level that is using AArch32.

C, bit [2]

Cycle counter reset. This bit is WO. The effects of writing to this bit are:

C	Meaning
0b0	No action.
0b1	Reset PMCCNTR_EL0 to zero.

This bit is always RAZ.

Note

Resetting PMCCNTR_EL0 does not change the cycle counter overflow bit.

P, bit [1]

Event counter reset. This bit is WO. The effects of writing to this bit are:

P	Meaning
0b0	No action.
0b1	Reset all event counters, not including PMCCNTR_EL0, to zero.

This bit is always RAZ.

Note

Resetting the event counters does not change the event counter overflow bits.

If ARMv8.5-PMU is implemented, the value of MDCR_EL2.HLP, or PMCR_EL0.LP is ignored and bits [63:0] of all event counters are reset.

E, bit [0]

Enable.

E	Meaning
0b0	All event counters in the range [0:(PMN-1)] and PMCCNTR_EL0, are disabled.
0b1	All event counters in the range [0:(PMN-1)] and PMCCNTR_EL0, are enabled by PMCNTENSET_EL0.

This bit is RW.

If EL2 is implemented then:

- If EL2 is using AArch32, PMN is HDCR.HPMN.
- If EL2 is using AArch64, PMN is MDCR_EL2.HPMN.
- If PMN is less than PMCR_EL0.N, this bit does not affect the operation of event counters in the range [PMN:(PMCR_EL0.N-1)].

If EL2 is not implemented, PMN is PMCR_EL0.N.

Note

The effect of MDCR_EL2.HPMN or HDCR.HPMN on the operation of this bit applies if EL2 is implemented regardless of whether EL2 is enabled in the current Security state. For more information, see the description of MDCR_EL2.HPMN or HDCR.HPMN.

On a Warm reset, this field resets to 0.

Accessing the PMCR_EL0

Note

SoftwareLockStatus() depends on the type of access attempted and AllowExternalPMUAccess() has a new definition from Armv8.4. Refer to the Pseudocode definitions for more information.

PMCR_EL0 can be accessed through the external debug interface:

Component	Offset	Instance
PMU	0xE04	PMCR_EL0

This interface is accessible as follows:

- When IsCorePowered(), !DoubleLockStatus(), !OSLockStatus(), AllowExternalPMUAccess() and SoftwareLockStatus() access to this register is **RO**.
- When IsCorePowered(), !DoubleLockStatus(), !OSLockStatus(), AllowExternalPMUAccess() and !SoftwareLockStatus() access to this register is **RW**.
- Otherwise access to this register returns an Error.

27~~13~~/03~~12~~/2019~~2018~~ 21~~16~~:59~~42~~; e5e4db499bf9867a4b93324c4dbac985d3da9376~~6379d01e197f1d40720d32d0f84e419e9187e009~~

(old) htmldiff from- (new)

PMEVCNTR<n>_EL0, Performance Monitors Event Count Registers, n = 0 - 30

The PMEVCNTR<n>_EL0 characteristics are:

Purpose

Holds event counter n, which counts events, where n is 0 to 30.

Configuration

External register PMEVCNTR<n>_EL0 bits [31:0] are architecturally mapped to AArch64 System register PMEVCNTR<n>_EL0[31:0] .

External register PMEVCNTR<n>_EL0 bits [31:0] are architecturally mapped to AArch32 System register PMEVCNTR<n>[31:0] .

PMEVCNTR<n>_EL0 is in the Core power domain. RW fields in this register reset to architecturally UNKNOWN values. The field descriptions identify when the reset values apply.

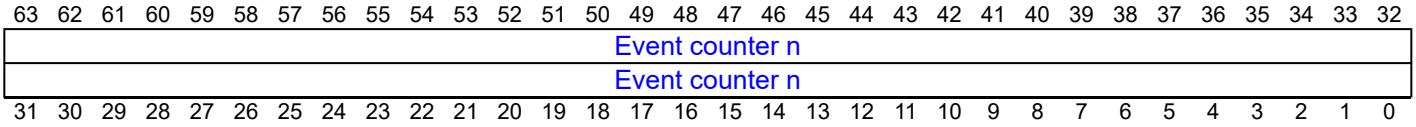
Attributes

PMEVCNTR<n>_EL0 is a 64-bit register.

Field descriptions

The PMEVCNTR<n>_EL0 bit assignments are:

When ARMv8.5-PMU is implemented:



Bits [63:0]

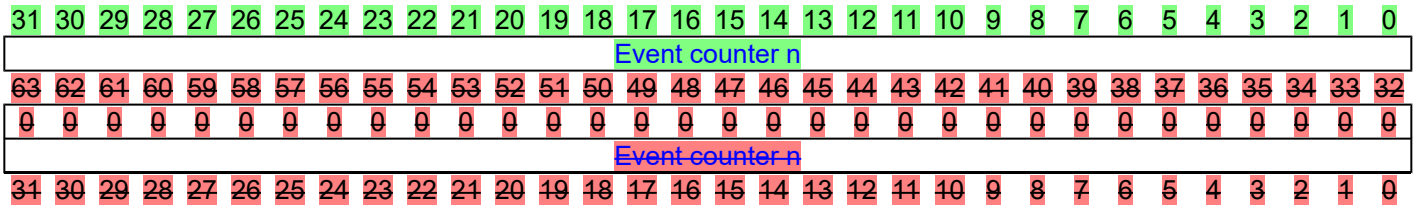
Event counter n. Value of event counter n, where n is the number of this register and is a number from 0 to 30.

If the highest implemented Exception level is using AArch32, the optional external interface to the performance monitors is implemented, and the PMCR.LP and HDCR.HLP bits are RAZ/WI, then locations in the external interface to the performance monitors that map to PMEVCNTR<n>_EL0[63:32] return UNKNOWN values on reads.

If the implementation does not support AArch64 at any Exception level, bits [63:32] of the event counters are not required to be implemented.

This field resets to an architecturally UNKNOWN value.

Otherwise:



Bits [63:32]

Reserved, RES0.

Bits [31:0]

Event counter n. Value of event counter n, where n is the number of this register and is a number from 0 to 30.

This field resets to an architecturally UNKNOWN value.

Accessing the PMEVCNTR<n>_EL0

External accesses to the performance monitors ignore PMUSERENR_EL0 and, if implemented, MDCR_EL2.{TPM, TPMCR, HPMN} and MDCR_EL3.TPM. This means that all counters are accessible regardless of the current Exception level or privilege of the access.

Note

SoftwareLockStatus() depends on the type of access attempted and AllowExternalPMUAccess() has a new definition from Armv8.4. Refer to the Pseudocode definitions for more information.

PMEVCNTR<n>_EL0 can be accessed through the external debug interface:

Component	Offset	Instance
PMU	0x000 + 8n	PMEVCNTR<n>_EL0

This interface is accessible as follows:

- When IsCorePowered(), !DoubleLockStatus(), !OSLockStatus(), AllowExternalPMUAccess() and SoftwareLockStatus() access to this register is **RO**.
- When IsCorePowered(), !DoubleLockStatus(), !OSLockStatus(), AllowExternalPMUAccess() and !SoftwareLockStatus() access to this register is **RW**.
- Otherwise access to this register returns an Error.

2713/0312/20192018 2116:5942: e5e4db499bf9867a4b93324c4dbac985d3da93766379d01e197f1d40720d32d0f84e419e9187e009