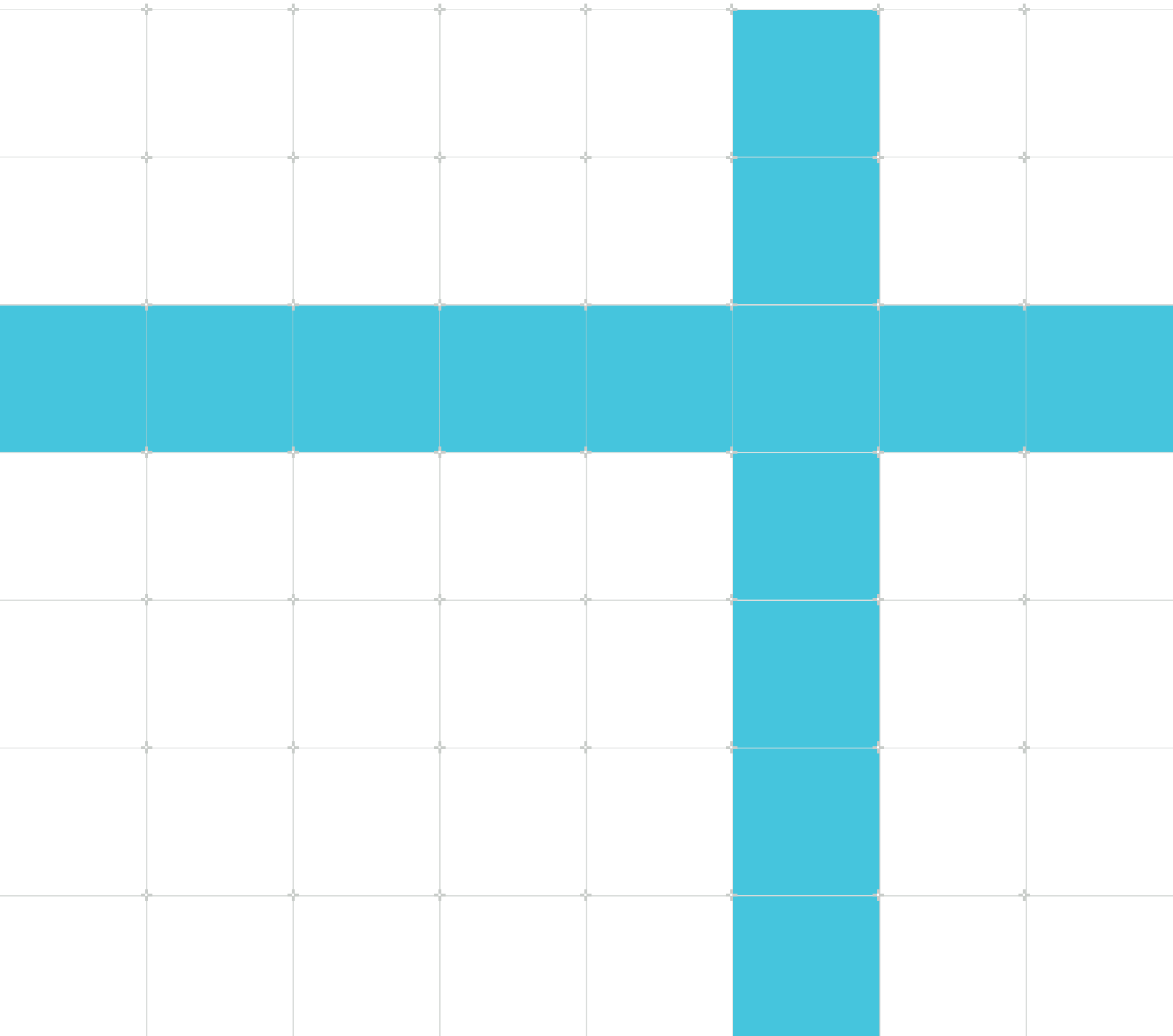
Feature names in A-profile architecture

Version 1.0

Non-Confidential

Issue
109697_2025_03_en



Feature names in A-profile architecture

Release information

Document history

Issue	Date	Confidentiality	Change
2025_03	14 March 2025	Non-Confidential	
2024_12	19 November 2024	Non-Confidential	
2024_09	26 September 2024	Non-Confidential	
0100-02	28 June 2024	Non-Confidential	
0100-01	18 March 2024	Non-Confidential	First release

Proprietary Notice

Arm Limited. Company 02557590 registered in England.

110 Fulbourn Road, Cambridge, England CB1 9NJ.

PRE-1121-V1.0

Confidentiality Status

Unrestricted Access is an Arm internal classification.

Product Status

The information in this document is for an Alpha product, that is a product under development.

Feedback

Arm welcomes feedback on this product and its documentation. To provide feedback on the product, create a ticket on https://support.developer.arm.com

To provide feedback on the document, fill the following survey: https://developer.arm.com/documentation-feedback-survey.

Inclusive language commitment

Arm values inclusive communities. Arm recognizes that we and our industry have used language that can be offensive. Arm strives to lead the industry and create change.

We believe that this document contains no offensive language. To report offensive language in this document, email terms@arm.com.

Contents

1. 2024 Architecture Extensions

Features introduced in 2024.

Feature	Title
FEAT_PMUv3_EXTPMN	Reserving PMU event counters for external agents
FEAT_SPEv1p5	Statistical Profiling Extension version 1.5
FEAT_SPE_EXC	SPE Profiling exception extension
FEAT_SPE_nVM	Statistical Profiling physical addressing mode extension
FEAT_TRBEv1p1	Trace Buffer Extension version 1.1
FEAT_TRBE_EXC	Trace Buffer Profiling exception extension
FEAT_SME_MOP4	Quarter-tile outer product instructions
FEAT_SME_TMOP	Structured sparsity outer product instructions
FEAT_PoPS	Point of Physical Storage
FEAT_SME2p2	Scalable Matrix Extension version 2.2
FEAT_CMPBR	Compare and Branch instructions
FEAT_SSVE_AES	Streaming SVE Mode Advanced Encryption Standard and 128-bit polynomial multiply long instructions
FEAT_RME_GPC3	RME Granule Protection Check 3 Extension
FEAT_RME_GDI	RME Granular Data Isolation extension
FEAT_SVE2p2	Scalable Vector Extensions version 2.2
FEAT_SSVE_BitPerm	Streaming Scalable Vector Bit Permutes instructions
FEAT_SSVE_FEXPA	Streaming FEXPA instruction
FEAT_SVE_AES2	SVE multi-vector Advanced Encryption Standard and 128-bit polynomial multiply long instructions
FEAT_LSFE	Large System Float Extension
FEAT_FPRCVT	Floating-Point to/from Integer in Scalar FP register
FEAT_SVE_F16F32MM	SVE Half-precision floating-point matrix multiply-accumulate to single-precision
FEAT_F8F16MM	8-bit floating-point matrix multiply-accumulate to half-precision
FEAT_F8F32MM	8-bit floating-point matrix multiply-accumulate to single-precision
FEAT_SVE_BFSCALE	BFloat16 Floating-Point Adjust Exponent
FEAT_OCCMO	Outer Cacheable Cache Maintenance Operation
FEAT_LS64WB	LS64 for Write-back cacheable memory
FEAT_MPAM_MSC_DOMAINS	MPAM Domains PARTID translation
FEAT_MPAM_MSC_DCTRL	MPAM Default Resource Control
FEAT_MPAM_PE_BW_CTRL	MPAM PE-side Bandwidth Controls
FEAT_PCDPHINT	Producer-Consumer Data Placement Hints
FEAT_SRMASK	Bitwise System Register Write Masks
FEAT_IDTE3	Trapping ID register accesses to EL3
FEAT_UINJ	Injection of Undefined Instruction exceptions
FEAT_NV2p1	Enhanced nested virtualization support
FEAT_LSUI	Unprivileged Load Store

2. 2023 Architecture Extensions

Features introduced in 2023.

Feature	Title
FEAT_SME_LUTv2	Lookup table instructions with 4-bit indices and 8-bit elements
FEAT_LUT	Lookup table instructions with 2-bit and 4-bit indices
FEAT_FAMINMAX	Floating-point maximum and minimum absolute value instructions
FEAT_FPMR	Floating-point Mode Register
FEAT_FP8	FP8 convert instructions
FEAT_FP8FMA	FP8 multiply-accumulate to half-precision and single-precision instructions
FEAT_SSVE_FP8FMA	SVE2 FP8 multiply-accumulate to half-precision and single-precision instructions in Streaming SVE mode
FEAT_FP8DOT4	FP8 4-way dot product to single-precision instructions
FEAT_SSVE_FP8DOT4	SVE2 FP8 4-way dot product to single-precision instructions in Streaming SVE mode
FEAT_FP8DOT2	FP8 2-way dot product to half-precision instructions
FEAT_SSVE_FP8DOT2	SVE FP8 2-way dot product to half-precision instructions in Streaming SVE mode
FEAT_SME_F8F16	SME2 ZA-targeting FP8 multiply-accumulate, dot product, and outer product to half-precision instructions
FEAT_SME_F8F32	SME2 ZA-targeting FP8 multiply-accumulate, dot product, and outer product to single-precision instructions
FEAT_CPA	Instruction-only Checked Pointer Arithmetic
FEAT_CPA2	Checked Pointer Arithmetic
FEAT_STEP2	Enhanced Software Step Extension
FEAT_BWE2	Breakpoint and watchpoint enhancements 2
FEAT_SPE_FPF	Statistical Profiling floating-point and SIMD flag extension
FEAT_SPE_EFT	Statistical Profiling extended filtering by type
FEAT_PMUv3_TH2	Performance Monitors event counter linking extension
FEAT_SPMU2	System Performance Monitors Extension version 2
FEAT_E3DSE	Delegated SError exception injection
FEAT_PMUv3_SME	Performance Monitors extensions for SME
FEAT_SPE_SME	Statisical Profiling extensions for SME
FEAT_SPE_ALTCLK	Statistical Profiling alternate clock domain extension
FEAT_HDBSS	Hardware Dirty state tracking structure
FEAT_HACDBS	Hardware accelerator for cleaning Dirty state
FEAT_TLBIW	TLBI VMALL for Dirty state
FEAT_ASID2	Support for concurrent use of two ASIDs
FEAT_RME_GPC2	RME Granule Protection Check 2 Extension
FEAT_FGWTE3	Fine-Grained Write Trap EL3
FEAT_PAuth_LR	Pointer authentication instructions that allow signing of LR using SP and PC as diversifiers
FEAT_ETS3	Enhanced Translation Synchronization

3. 2022 Architecture Extensions

Features introduced in 2022.

Feature	Title
FEAT_BWE	Breakpoint and watchpoint enhancements
FEAT_ADERR	Asynchronous Device Error Exceptions
FEAT_ANERR	Asynchronous Normal Error Exceptions
FEAT_DoubleFault2	Double Fault Extension v2
FEAT_PFAR	Physical Fault Address Register Extension
FEAT_RASv2	RAS Extension v2
FEAT_RASSAv2	RAS System Architecture Extension v2
FEAT_CSSC	Common Short Sequence Compression instructions
FEAT_RPRFM	Support for Range Prefetch Memory instruction
FEAT_PRFMSLC	SLC target support for PRFM instructions
FEAT_SPECRES2	Enhanced speculation restriction instructions
FEAT_CLRBHB	Support for Clear Branch History instruction
FEAT_ECBHB	Exploitative control using branch history information
FEAT_SME2p1	Scalable Matrix Extension version 2.1
FEAT_SME_F16F16	Non-widening half-precision FP16 to FP16 arithmetic for SME2
FEAT_SVE_B16B16	Non-widening BFloat16 to BFloat16 arithmetic for SVE2 and SME2
FEAT_SME_B16B16	Non-widening BFloat16 to BFloat16 SME ZA-targeting arithmetic
FEAT_FGT2	Fine-grained traps 2
FEAT_MTE4	Enhanced Memory Tagging Extension
FEAT_MTE_CANONICAL_TAGS	Canonical Tag checking for Untagged memory
FEAT_MTE_NO_ADDRESS_TAGS	Memory tagging with Address tagging disabled
FEAT_MTE_TAGGED_FAR	FAR_ELx on a Tag Check Fault
FEAT_MTE_PERM	Allocation tag access permission
FEAT_MTE_STORE_ONLY	Store-only Tag Checking
FEAT_ITE	Instrumentation Trace Extension
FEAT_TRBE_EXT	Trace Buffer external mode
FEAT_TRBE_MPAM	Trace Buffer MPAM extensions
FEAT_ETEv1p3	Embedded Trace Extension version 1.3
FEAT_GCS	Guarded Control Stack Extension
FEAT_CHK	Check Feature Status
FEAT_SPE_DPFZS	Disable Cycle Counter on SPE Freeze
FEAT_SPE_CRR	Statistical Profiling call return branch records
FEAT_EBEP	Exception-based Event Profiling
FEAT_SEBEP	Synchronous Exception-based Event Profiling
FEAT_PMUv3_SS	PMU Snapshot extension
FEAT_SPMU	System Performance Monitors Extension

Feature	Title
FEAT_PMUv3_ICNTR	Fixed-function instruction counter
FEAT_PMUv3p9	Armv8.9 PMU extensions
FEAT_SVE2p1	Scalable Vector Extensions version 2.1
FEAT_Debugv8p9	Debug v8.9
FEAT_ABLE	Address Breakpoint Linking Extension
FEAT_PCSRv8p9	Armv8.9 PC Sample-based Profiling Extension
FEAT_LRCPC3	Load-Acquire RCpc instructions version 3
FEAT_SPEv1p4	Statistical Profiling Extension version 1.4
FEAT_SPE_FDS	Statistical Profiling data source filtering
FEAT_PMUv3_EDGE	PMU event edge detection
FEAT_D128	128-bit Translation Tables, 56 bit PA
FEAT_AIE	Memory Attribute Index Enhancement
FEAT_S1PIE	Stage 1 permission indirections
FEAT_ATS1A	Address Translation operations that ignore stage 1 permissions
FEAT_S1POE	Stage 1 permission overlays
FEAT_S2PIE	Stage 2 permission indirections
FEAT_S2POE	Stage 2 permission overlays
FEAT_SYSREG128	128-bit System registers
FEAT_SYSINSTR128	128-bit System instructions
FEAT_LSE128	128-bit Atomics
FEAT_HAFT	Hardware managed Access Flag for Table descriptors
FEAT_THE	Translation Hardening Extension
FEAT_LVA3	56-bit VA
FEAT_MTE_ASYNC	Asynchronous reporting of Tag Check Fault
FEAT_EDHSR	Support for EDHSR
FEAT_AMU_EXT64	the 64-bit external Activity Monitors extension

4. 2021 Architecture Extensions

Features introduced in 2021.

Feature	Title
FEAT_SME2	Scalable Matrix Extensions version 2
FEAT_MEC	Memory Encryption Contexts
FEAT_BRBEv1p1	Branch Record Buffer Extension version 1.1
FEAT_CMOW	Control for cache maintenance permission
FEAT_Debugv8p8	Debug v8.8
FEAT_GICv3_NMI	GIC Non-Maskable Interrupts
FEAT_HBC	Hinted conditional branches
FEAT_MOPS	Standardization of memory operations
FEAT_NMI	Non-maskable Interrupts
FEAT_SPEv1p3	Statistical Profiling Extensions version 1.3
FEAT_TIDCP1	EL0 use of IMPLEMENTATION DEFINED functionality
FEAT_PMUv3_TH	Event counting threshold
FEAT_PMUv3p8	Armv8.8 PMU extensions
FEAT_SCTLR2	Extension to SCTLR_ELx
FEAT_TCR2	Support for TCR2_ELx
FEAT_PMUv3_EXT64	64-bit external interface to the Performance Monitors

5. 2020 Architecture Extensions

Features introduced in 2020.

Feature	Title
FEAT_EBF16	AArch64 Extended BFloat16 behaviors
FEAT_ETEv1p1	Embedded Trace Extension
FEAT_HCX	Support for the HCRX_EL2 register
FEAT_PAN3	Support for SCTLR_ELx.EPAN
FEAT_WFxT	WFE and WFI instructions with timeout
FEAT_XS	XS attribute
FEAT_AFP	Alternate floating-point behavior
FEAT_RPRES	Increased precision of FRECPE and FRSQRTE
FEAT_LPA2	Larger physical address for 4KB and 16KB translation granules
FEAT_LS64	Support for 64-byte loads and stores without status
FEAT_LS64_V	Support for 64-byte stores with status
FEAT_LS64_ACCDATA	Support for 64-byte EL0 stores with status
FEAT_MTE3	MTE Asymmetric Fault Handling
FEAT_MTE_ASYM_FAULT	Memory tagging asymmetric faults
FEAT_SPEv1p2	Statistical Profiling Extensions version 1.2
FEAT_SPE_FnE	Statistical Profiling inverse event filter
FEAT_SPE_PBT	Statistical Profiling previous branch target
FEAT_PMUv3p7	Armv8.7 PMU extensions

6. Architecture Extensions before 2020

Features introduced before 2020.

Feature	Title
FEAT_RASSA_GRP	RAS groups
FEAT_RASSA_ACR	Access Control Register
FEAT_TGran4K	Support for 4KB memory translation granule size at stage 1
FEAT_S2TGran4K	Support for 4KB memory translation granule size at stage 2
FEAT_TGran16K	Support for 16KB memory translation granule size at stage 1
FEAT_S2TGran16K	Support for 16KB memory translation granule size at stage 2
FEAT_TGran64K	Support for 64KB memory translation granule size at stage 1
FEAT_S2TGran64K	Support for 64KB memory translation granule size at stage 2
FEAT_ASID16	16 bit ASID
FEAT_MixedEnd	Mixed-endian support
FEAT_MixedEndEL0	Mixed-endian support at EL0
FEAT_SpecSEI	SError interrupt exceptions from speculative reads of memory
FEAT_E2H0	Programming of HCR_EL2.E2H
FEAT_Armv9_Crypto	Armv9 Cryptographic Extension
FEAT_DoPD	Debug over Powerdown
FEAT_ETE	Embedded Trace Extension
FEAT_SVE_AES	Scalable Vector AES instructions
FEAT_SVE_BitPerm	Scalable Vector Bit Permutes instructions
FEAT_SVE_PMULL128	SVE single-vector Advanced Encryption Standard and 128-bit polynomial multiply long instructions
FEAT_SVE_SHA3	Scalable Vector SHA3 instructions
FEAT_SVE_SM4	Scalable Vector SM4 instructions
FEAT_SVE2	Scalable Vector Extension version 2
FEAT_TME	Transactional Memory Extension
FEAT_TRBE	Trace Buffer Extension
FEAT_CP15SDISABLE2	CP15SDISABLE2
FEAT_DGH	Data Gathering Hint
FEAT_ECV	Enhanced Counter Virtualization
FEAT_ECV_POFF	Enhanced Counter Virtualization Physical Offset
FEAT_FGT	Fine Grain Traps
FEAT_PAuth2	Enhancements to pointer authentication
FEAT_AMUv1p1	Activity Monitors Extension version 1.1
FEAT_HPMN0	Setting of MDCR_EL2.HPMN to zero
FEAT_MPAMv0p1	Memory Partitioning and Monitoring extension version 0.1
FEAT_MPAMv1p1	Memory Partitioning and Monitoring extension version 1.1
FEAT_MTPMU	Multi-threaded PMU extensions
FEAT_TWED	Delayed Trapping of WFE

Feature	Title
FEAT_SSBS	Speculative Store Bypass Safe
FEAT_SSBS2	MRS and MSR instructions for SSBS version 2
FEAT_CSV2	Cache Speculation Variant 2
FEAT_CSV3	Cache Speculation Variant 3
FEAT_SB	Speculation Barrier
FEAT_SPECRES	Speculation restriction instructions
FEAT_EVT	Enhanced Virtualization Traps
FEAT_GTG	Guest translation granule size
FEAT_BTI	Branch Target Identification
FEAT_E0PD	Preventing EL0 access to halves of address maps
FEAT_DPB2	DC CVADP instruction
FEAT_FlagM2	Enhancements to flag manipulation instructions
FEAT_ExS	Context synchronization and exception handling
FEAT_FRINTTS	Floating-point to integer instructions
FEAT_MTE	Memory Tagging Extension
FEAT_MTE2	Memory Tagging Extension
FEAT_PMUv3p5	Arm8.5 PMU extensions
FEAT_RNG	Random number generator
FEAT_RNG_TRAP	Trapping support for RNDR/RNDRRS
FEAT_DotProd	Advanced SIMD dot product instructions
FEAT_FHM	Floating-point half-precision to single-precision multiply-add instructions
FEAT_Secure	Support for Secure state
FEAT_SEL2	Secure EL2
FEAT_S2FWB	Stage 2 forced Write-Back
FEAT_DIT	Data Independent Timing instructions
FEAT_IDST	ID space trap handling
FEAT_FlagM	Condition flag manipulation instructions
FEAT_LSE2	Large System Extensions version 2
FEAT_LRCPC2	Load-Acquire RCpc instructions version 2
FEAT_TLBIOS	TLB invalidate instructions in Outer Shareable domain
FEAT_TLBIRANGE	TLB invalidate range instructions
FEAT_TTL	Translation Table Level
FEAT_BBM	Translation table break-before-make levels
FEAT_RASv1p1	RAS extension v1.1
FEAT_RASSAv1p1	RAS version 1.1 System Architecture
FEAT_DoubleFault	Double Fault Extension
FEAT_Debugv8p4	Debug v8.4
FEAT_CNTSC	Generic Counter Scaling
FEAT_NV2	Enhanced nested virtualization support
FEAT_PMUv3p4	Arm8.4 PMU extensions

Feature	Title
FEAT_TRF	Self-hosted Trace extensions
FEAT_TTST	Small translation tables
FEAT_AMUv1	Activity Monitors Extension version 1
FEAT_PAuth	Pointer authentication
FEAT_JSCVT	JavaScript conversion instructions
FEAT_LRCPC	Load-Acquire RCpc instructions
FEAT_FCMA	Floating-point complex number instructions
FEAT_CCIDX	Extended cache index
FEAT_NV	Nested Virtualization
FEAT_SPEv1p1	Statistical Profiling Extension version 1.1
FEAT_PACQARMA3	Pointer authentication - QARMA3 algorithm
FEAT_PACQARMA5	Pointer authentication - QARMA5 algorithm
FEAT_PACIMP	Pointer authentication - IMPLEMENTATION DEFINED algorithm
FEAT_EPAC	Enhanced pointer authentication
FEAT_FPAC	Faulting on AUT* instructions
FEAT_FPACC_SPEC	Speculative behavior of pointer authentication instructions
FEAT_FPACCOMBINE	Faulting on combined pointer authentication instructions
FEAT_CONSTPACFIELD	PAC algorithm enhancement
FEAT_TTCNP	Translation table Common not private translations
FEAT_XNX	Translation table stage 2 Unprivileged Execute-never
FEAT_UAO	Unprivileged Access Override control
FEAT_PAN2	AT S1E1R and AT S1E1W instruction variants affected by PSTATE.PAN
FEAT_DPB	DC CVAP instruction
FEAT_Debugv8p2	Debug v8.2
FEAT_ASMv8p2	Armv8.2 changes to the A64 ISA
FEAT_IESB	Implicit Error Synchronization event
FEAT_RASSA	RAS System Architecture
FEAT_RASSAv1	RAS System Architecture version 1
FEAT_RAS	Reliability, Availability and Serviceability (RAS) Extension
FEAT_AA32HPD	AArch32 Hierarchical permission disables
FEAT_HPDS2	Hierarchical permission disables
FEAT_LSMAOC	AArch32 Load/Store Multiple instruction atomicity and ordering controls
FEAT_FP16	Half-precision floating-point data processing
FEAT_LVA	Large VA support
FEAT_LPA	Large PA and IPA support
FEAT_PCSRv8p2	PC Sample-based Profiling Extension
FEAT_SHA512	Advanced SIMD SHA512 instructions
FEAT_SHA3	Advanced SIMD SHA3 instructions
FEAT_SM3	Advanced SIMD SM3 instructions
FEAT_SM4	Advanced SIMD SM4 instructions

Feature	Title
FEAT_Crypto	Cryptographic Extension
FEAT_F32MM	Single-precision floating-point matrix multiply-accumulate
FEAT_F64MM	Double-precision floating-point matrix multiply-accumulate
FEAT_BF16	AArch64 BFloat16 instructions
FEAT_AA32BF16	AArch32 BFloat16 instructions
FEAT_I8MM	AArch64 Int8 matrix multiply-accumulate
FEAT_AA32I8MM	AArch32 Int8 matrix multiply-accumulate
FEAT_SPE	Statistical Profiling Extension
FEAT_SVE	Scalable Vector Extension
FEAT_MPAM	Memory Partitioning and Monitoring Extension
FEAT_CRC32	CRC32 instructions
FEAT_LSE	Large System Extensions
FEAT_RDM	Advanced SIMD rounding double multiply accumulate instructions
FEAT_HPDS	Hierarchical permission disables in translations tables
FEAT_VHE	Virtualization Host Extensions
FEAT_PAN	Privileged access never
FEAT_LOR	Limited ordering regions
FEAT_Debugv8p1	Debug v8.1
FEAT_HAFDBS	Hardware management of the Access flag and dirty state
FEAT_PMUv3p1	Armv8.1 PMU extensions
FEAT_VMID16	16-bit VMID
FEAT_AdvSIMD	Advanced SIMD Extension
FEAT_FP	Floating Point extensions
FEAT_CSV2_1p1	Cache Speculation Variant 2
FEAT_CSV2_1p2	Cache Speculation Variant 2 version 1.2
FEAT_CSV2_2	Cache Speculation Variant 2 version 2
FEAT_CSV2_3	Cache Speculation Variant 2 version 3
FEAT_ETS2	Enhanced Translation Synchronization
FEAT_SHA1	Advanced SIMD SHA1 instructions
FEAT_SHA256	Advanced SIMD SHA256 instructions
FEAT_AES	Advanced SIMD AES instructions
FEAT_PMULL	Advanced SIMD PMULL instructions
FEAT_PCSRv8	PC Sample-based Profiling extension
FEAT_DoubleLock	Double Lock
FEAT_GICv3	Generic Interrupt Controller
FEAT_GICv3p1	Generic Interrupt Controller version 3.1
FEAT_GICv3_TDIR	Trapping EL1 writes to ICV_DIR_EL1 and ICV_DIR
FEAT_GICv3_LEGACY	Support for GICv2 legacy operations
FEAT_GICv4	Generic Interrupt Controller version 4
FEAT_AMU_EXT32	AArch32 External Activity Monitors

Feature	Title
FEAT_AMU_EXT	External Activity Monitors
FEAT_PMUv3_EXT32	32-bit external interface to the Performance Monitors
FEAT_PMUv3_EXT	External interface to the Performance Monitors
FEAT_GICv4p1	Generic Interrupt Controller version 4.1
FEAT_IVIPT	The IVIPT Extension
FEAT_ETMv4	Embedded Trace Macrocell version 4
FEAT_TRC_SR	Trace System registers
FEAT_TRC_EXT	Trace external registers
FEAT_ETMv4p1	Embedded Trace Macrocell version 4.1
FEAT_ETMv4p2	Embedded Trace Macrocell version 4.2
FEAT_ETMv4p3	Embedded Trace Macrocell version 4.3
FEAT_ETMv4p4	Embedded Trace Macrocell version 4.4
FEAT_ETMv4p5	Embedded Trace Macrocell version 4.5
FEAT_ETMv4p6	Embedded Trace Macrocell version 4.6
FEAT_nTLBPA	Intermediate caching of translation table walks
FEAT_PMUv3	PMU extension version 3
FEAT_EL0	Support for execution at EL0
FEAT_EL1	Support for execution at EL1
FEAT_EL2	Support for execution at EL2
FEAT_EL3	Support for EL3
FEAT_AA32	PE supports AArch32.
FEAT_AA64	PE uses AArch64 after last reboot
FEAT_AA32EL0	Support for AArch32 at EL0
FEAT_AA32EL1	Support for AArch32 at EL1
FEAT_AA32EL2	Support for AArch32 at EL2
FEAT_AA32EL3	Support for AArch32 at EL3
FEAT_AA64EL0	Support for AArch64 at EL0
FEAT_AA64EL1	Support for AArch64 at EL1
FEAT_AA64EL2	Support for AArch64 at EL2
FEAT_AA64EL3	Support for AArch64 at EL3

7. Feature descriptions

Additional information about each feature:

7.1 The Armv8.0 architecture extension

The original Armv8-A architecture is called Armv8.0. It contains mandatory and optional architectural features. Some features must be implemented together. An implementation is Armv8.0 compliant if it includes all of the Armv8.0 architectural features that are mandatory.

An Armv8.0 compliant implementation can additionally include:

- Armv8.0 features that are optional.
- Any arbitrary subset of the architectural features of Armv8.1, subject only to those constraints that require that certain features be implemented together.

FEAT_AA32, PE supports AArch32.

FEAT_AA32 is OPTIONAL.

If FEAT_AA32 is implemented, then FEAT_AA32EL0 is implemented.

FEAT_AA32EL0, Support for AArch32 at EL0

FEAT_AA32EL0 is OPTIONAL from Armv8.0.

If FEAT_AA32EL0 is implemented, then FEAT_EL0 is implemented.

If FEAT_AA32EL0 is implemented, then FEAT_AA32 is implemented.

FEAT_AA32EL1, Support for AArch32 at EL1

FEAT_AA32EL1 is OPTIONAL from Armv8.0.

If Armv9.0 is implemented, then FEAT_AA32EL1 is not implemented.

If FEAT_AA32EL1 is implemented, then FEAT_AA32EL0 is implemented.

If FEAT_AA32EL1 is implemented, then FEAT_EL1 is implemented.

FEAT_AA32EL2, Support for AArch32 at EL2

FEAT_AA32EL2 is OPTIONAL from Armv8.0.

If FEAT_AA32EL2 is implemented, then FEAT_AA32EL1 is implemented.

If FEAT_AA32EL2 is implemented, then FEAT_EL2 is implemented.

FEAT_AA32EL3, Support for AArch32 at EL3

FEAT_AA32EL3 is OPTIONAL from Armv8.0.

If FEAT_AA32EL3 is implemented, then FEAT_AA32EL1 is implemented.

When FEAT_AA32EL3 and FEAT_EL2 are implemented, FEAT_AA32EL2 is implemented.

If FEAT_AA32EL3 is implemented, then FEAT_EL3 is implemented.

FEAT_AA64, PE uses AArch64 after last reboot

FEAT_AA64 is OPTIONAL.

If FEAT_AA64 is implemented, then FEAT_AA64EL0, FEAT_AA64EL1, FEAT_AA64EL2, or FEAT_AA64EL3 is implemented.

When FEAT_AA64EL0, FEAT_AA64EL1, FEAT_AA64EL2, or FEAT_AA64EL3 is implemented, FEAT_AA64 is implemented.

FEAT_AA64EL0, Support for AArch64 at EL0

FEAT_AA64EL0 is OPTIONAL from Armv8.0.

FEAT_AA64EL0 is mandatory from Armv9.0.

If FEAT_AA64EL0 is implemented, then FEAT_AA64EL1 is implemented.

If FEAT_AA64EL0 is implemented, then FEAT_EL0 is implemented.

FEAT_AA64EL1, Support for AArch64 at EL1

FEAT_AA64EL1 is OPTIONAL from Armv8.0.

FEAT_AA64EL1 is mandatory from Armv9.0.

If FEAT_AA64EL1 is implemented, then FEAT_AA64EL0 is implemented.

When FEAT_AA64EL1 and FEAT_EL2 are implemented, FEAT_AA64EL2 is implemented.

When FEAT_AA64EL1 and FEAT_EL3 are implemented, FEAT_AA64EL3 is implemented.

If FEAT_AA64EL1 is implemented, then FEAT_EL1 is implemented.

FEAT_AA64EL2, Support for AArch64 at EL2

FEAT_AA64EL2 is OPTIONAL from Armv8.0.

If FEAT_AA64EL2 is implemented, then FEAT_AA64EL1 is implemented.

When FEAT_AA64EL2 and FEAT_EL3 are implemented, FEAT_AA64EL3 is implemented.

If FEAT_AA64EL2 is implemented, then FEAT_EL2 is implemented.

FEAT_AA64EL3, Support for AArch64 at EL3

FEAT_AA64EL3 is OPTIONAL from Armv8.0.

If FEAT_AA64EL3 is implemented, then FEAT_AA64EL1 is implemented.

When FEAT_AA64EL3 and FEAT_EL2 are implemented, FEAT_AA64EL2 is implemented.

If FEAT_AA64EL3 is implemented, then FEAT_EL3 is implemented.

FEAT_AES, Advanced SIMD AES instructions

FEAT_AES provides the AES* instructions to support AES encryption and decryption, AESD, AESE, AESMC, and AESIMC.

This feature is supported in both AArch64 and AArch32 states.

FEAT_AES is OPTIONAL from Armv8.0.

If FEAT_AES is implemented, then FEAT_Crypto is implemented.

The following field identifies the presence of FEAT_AES:

- ID_AA64ISAR0_EL1.AES.

FEAT_ASID16, 16 bit ASID

FEAT_ASID16 is OPTIONAL from Armv8.0.

The following field identifies the presence of FEAT_ASID16:

- ID_AA64MMFR0_EL1.ASIDBits.

FEAT_AdvSIMD, Advanced SIMD Extension

FEAT_AdvSIMD includes support for the SISD and SIMD operations. All Armv8-A systems that support standard operating systems with rich application environments also provide hardware support for Advanced SIMD instructions.

This feature is supported in both AArch64 and AArch32 states.

FEAT_AdvSIMD is OPTIONAL from Armv8.0.

If FEAT_AdvSIMD is implemented, then FEAT_FP is implemented.

The following fields identify the presence of FEAT_AdvSIMD:

- ID_AA64PFR0_EL1.AdvSIMD.
- EDPFR.AdvSIMD.

For more information, see:

- ‘Supported data types’.
- ‘A64 Advanced SIMD and Floating-point Instruction Descriptions’.

- ‘Advanced SIMD and floating-point instructions’.
- ‘T32 and A32 Advanced SIMD and Floating-point Instruction Descriptions’.

FEAT_CRC32, CRC32 instructions

FEAT_CRC32 introduces the support for the CRC32* instructions that perform cyclic redundancy check calculations.

This feature is supported in both AArch64 and AArch32 states.

FEAT_CRC32 is OPTIONAL from Armv8.0.

FEAT_CRC32 is mandatory from Armv8.1.

The following fields identify the presence of FEAT_CRC32:

- ID_AA64ISAR0_EL1.CRC32.
- ID_ISAR5_EL1.CRC32.
- ID_ISAR5.CRC32.

FEAT_CSV2, Cache Speculation Variant 2

FEAT_CSV2 introduces a mechanism to identify if hardware cannot disclose information about whether any prediction resource, such as branch target, data-value, or cache prefetch, trained in one hardware described context can control speculative execution in a different hardware described context.

This feature is supported in both AArch64 and AArch32 states.

FEAT_CSV2 is OPTIONAL from Armv8.0.

FEAT_CSV2 is mandatory from Armv8.5.

The following fields identify the presence of FEAT_CSV2:

- ID_AA64PFR0_EL1.CSV2.
- ID_PFR0_EL1.CSV2.
- ID_PFR0.CSV2.

For more information, see:

- For AArch64, ‘Restrictions on the effects of speculation’.
- ‘AArch32 restrictions on the effects of speculation’.

FEAT_CSV2_1p1, Cache Speculation Variant 2

For FEAT_CSV2_1p1, within a hardware-described context, branch targets trained for branches situated at one address can control speculative execution of branches situated at different addresses only in a hard-to-determine way.

FEAT_CSV2_1p1 does not support the SCXTNUM_ELx registers.

This feature is supported in both AArch64 and AArch32 states.

FEAT_CSV2_1p1 is OPTIONAL from Armv8.0.

If FEAT_CSV2_1p1 is implemented, then FEAT_CSV2 is implemented.

The following fields identify the presence of FEAT_CSV2_1p1:

- ID_AA64PFR1_EL1.CSV2_frac.
- ID_AA64PFR0_EL1.CSV2.
- ID_PFR0_EL1.CSV2.
- ID_PFR0.CSV2.

For more information, see:

- For AArch64, 'Restrictions on the effects of speculation'.
- 'AArch32 restrictions on the effects of speculation'.

FEAT_CSV2_1p2, Cache Speculation Variant 2 version 1.2

For FEAT_CSV2_1p2, within a hardware-described context, branch targets trained for branches situated at one address can control speculative execution of branches situated at different addresses only in a hard-to-determine way.

FEAT_CSV2_1p2 introduces the SCXTNUM_ELx registers, but the contexts do not include the SCXTNUM_ELx register contexts.

This feature is supported in AArch64 state only.

FEAT_CSV2_1p2 is OPTIONAL from Armv8.0.

If FEAT_CSV2_1p2 is implemented, then FEAT_CSV2_1p1 is implemented.

The following fields identify the presence of FEAT_CSV2_1p2:

- ID_AA64PFR1_EL1.CSV2_frac.
- ID_AA64PFR0_EL1.CSV2.

For more information, see:

- For AArch64, 'Restrictions on the effects of speculation'.
- 'AArch32 restrictions on the effects of speculation'.

FEAT_CSV2_2, Cache Speculation Variant 2 version 2

FEAT_CSV2_2 introduces the SCXTNUM_ELx registers, which provide a number that can be used to separate out different context numbers within their respective Exception levels for the purpose of protecting against speculative attacks through prediction resources.

This feature is supported in AArch64 state only.

FEAT_CSV2_2 is OPTIONAL from Armv8.0.

If FEAT_CSV2_2 is implemented, then FEAT_CSV2 is implemented.

If FEAT_CSV2_2 is implemented, then FEAT_CSV2_1p1 is not implemented.

The following field identifies the presence of FEAT_CSV2_2:

- ID_AA64PFR0_EL1.CSV2.

For more information, see:

- For AArch64, 'Restrictions on the effects of speculation'.
- 'AArch32 restrictions on the effects of speculation'.

FEAT_CSV2_3, Cache Speculation Variant 2 version 3

FEAT_CSV2_3 introduces a mechanism to identify if hardware cannot disclose information about whether prediction resources and branch history trained in one hardware described context can control speculative execution in a different hardware described context.

This feature is supported in AArch64 state only.

FEAT_CSV2_3 is OPTIONAL from Armv8.0.

If FEAT_CSV2_3 is implemented, then FEAT_CSV2_2 is implemented.

The following field identifies the presence of FEAT_CSV2_3:

- ID_AA64PFR0_EL1.CSV2.

For more information, see:

- For AArch64, 'Restrictions on the effects of speculation'.
- 'AArch32 restrictions on the effects of speculation'.

FEAT_Crypto, Cryptographic Extension

The Arm Cryptographic Extension provides instructions for the acceleration of encryption and decryption. The presence of the Cryptographic Extension in an implementation is subject to export license controls.

The Cryptographic Extension is an extension of the SIMD support and operates on the vector register file. It also provides multiply instructions that operate on long polynomials.

In an implementation that supports both AArch64 state and AArch32 state, FEAT_AES, FEAT_PMULL, FEAT_SHA1 and FEAT_SHA256 provide the same functionality in both states.

This feature is supported in both AArch64 and AArch32 states.

FEAT_Crypto is OPTIONAL.

If FEAT_Crypto is implemented, then FEAT_AES or FEAT_SHA1 is implemented.

In an Armv8.2 implementation, if FEAT_Crypto is implemented, FEAT_PMULL, FEAT_SHA256, FEAT_SM3, or FEAT_SM4 is implemented.

For more information, see:

- The Cryptographic Extension in AArch64 state.
- The Cryptographic Extension in AArch32 state.

FEAT_DoubleLock, Double Lock

FEAT_DoubleLock is the mnemonic used for the OS Double Lock.

FEAT_DoubleLock is OPTIONAL from Armv8.0.

If Armv9.0 is implemented, then FEAT_DoubleLock is not implemented.

The following fields identify the presence of FEAT_DoubleLock:

- ID_AA64DFR0_EL1.DoubleLock.
- DBGDEVID.DoubleLock.

FEAT_EL0, Support for execution at EL0

FEAT_EL0 is mandatory from Armv8.0.

If FEAT_EL0 is implemented, then FEAT_AA32EL0 or FEAT_AA64EL0 is implemented.

FEAT_EL1, Support for execution at EL1

FEAT_EL1 is mandatory from Armv8.0.

If FEAT_EL1 is implemented, then FEAT_AA32EL1 or FEAT_AA64EL1 is implemented.

FEAT_EL2, Support for execution at EL2

FEAT_EL2 is OPTIONAL from Armv8.0.

If FEAT_EL2 is implemented, then FEAT_AA32EL2 or FEAT_AA64EL2 is implemented.

FEAT_EL3, Support for EL3

FEAT_EL3 is OPTIONAL from Armv8.0.

If FEAT_EL3 is implemented, then FEAT_AA32EL3 or FEAT_AA64EL3 is implemented.

FEAT_ETMv4, Embedded Trace Macrocell version 4

FEAT_ETMv4 indicates support for the Embedded Trace Macrocell architecture ETMv4.

FEAT_ETMv4 is OPTIONAL from Armv8.0.

If Armv9.0 is implemented, then FEAT_ETMv4 is not implemented.

If FEAT_ETMv4 is implemented, then FEAT_TRC_SR or FEAT_TRC_EXT is implemented.

For more information, see the *Arm® Embedded Trace Macrocell Architecture Specification, ETMv4(ARM IHI 0064)*.

FEAT_ETS2, Enhanced Translation Synchronization

FEAT_ETS2 introduces support for enhanced memory access ordering requirements for translation table walks.

This feature is supported in both AArch64 and AArch32 states.

FEAT_ETS2 is OPTIONAL from Armv8.0.

FEAT_ETS2 is mandatory from Armv8.8.

The following fields identify the presence of FEAT_ETS2:

- ID_AA64MMFR1_EL1.ETS.
- ID_MMFR5_EL1.ETS.
- ID_MMFR5.ETS.

For more information, see:

- ‘Ordering requirements defined by the formal concurrency model’.
- ‘Ordering of memory accesses from translation table walks’.
- ‘Ordering of translation table walks’.

FEAT_FP, Floating Point extensions

FEAT_FP includes support for single-precision and double-precision floating-point types. All Armv8-A systems that support standard operating systems with rich application environments also provide hardware support for floating-point instructions.

This feature is supported in both AArch64 and AArch32 states.

FEAT_FP is OPTIONAL from Armv8.0.

If FEAT_FP is implemented, then FEAT_AdvSIMD is implemented.

The following fields identify the presence of FEAT_FP:

- ID_AA64PFR0_EL1.FP.
- EDPFR.FP.

For more information, see:

- ‘Supported data types’.
- ‘Floating-point support’.
- ‘A64 Advanced SIMD and Floating-point Instruction Descriptions’.

- ‘Advanced SIMD and floating-point instructions’.
- ‘T32 and A32 Advanced SIMD and Floating-point Instruction Descriptions’.

FEAT_IVIPT, The IVIPT Extension

FEAT_IVIPT describes any permitted instruction cache implementation. This includes *Virtual Index*, *Physical tag* (VIPT) cache policy and *Physical Index*, *Physical Tag* (PIPT) cache policy.

FEAT_IVIPT is mandatory from Armv8.0.

For more information, see:

- ‘Instruction caches’.
- ‘Instruction caches’.

FEAT_MixedEnd, Mixed-endian support

FEAT_MixedEnd provides support for mixed-endian configuration.

If FEAT_MixedEnd is implemented, then FEAT_MixedEndEL0 is implemented.

FEAT_MixedEnd is OPTIONAL from Armv8.0.

The following field identifies the presence of FEAT_MixedEnd:

- ID_AA64MMFR0_EL1.BigEnd.

For more information, see:

- ‘Mixed-endian support in AArch64’.
- ‘Mixed-endian support in AArch32’.

FEAT_MixedEndEL0, Mixed-endian support at EL0

FEAT_MixedEndEL0 provides support for mixed-endian at EL0.

FEAT_MixedEndEL0 is OPTIONAL from Armv8.0.

The following fields identify the presence of FEAT_MixedEndEL0:

- ID_AA64MMFR0_EL1.BigEndEL0.
- ID_AA64MMFR0_EL1.BigEnd.

For more information, see:

- ‘Mixed-endian support in AArch64’.
- ‘Mixed-endian support in AArch32’.

FEAT_PCSRv8, PC Sample-based Profiling extension

FEAT_PCSRv8 introduces support for PC Sample-based Profiling Extension that provides coarse-grained, non-invasive profiling by an external debugger.

FEAT_PCSRv8 is OPTIONAL from Armv8.0.

If FEAT_PCSRv8 is implemented, then FEAT_PCSRv8p2 is not implemented.

The following fields identify the presence of FEAT_PCSRv8:

- EDDEVID.PCSample.
- DBGDEVID.PCSample.
- EDDEVID1.PCSROffset.
- DBGDEVID1.PCSROffset.

For more information, see 'About the PC Sample-based Profiling Extension'.

FEAT_PMULL, Advanced SIMD PMULL instructions

FEAT_PMULL provides the AES* instructions that support multiplication of 64-bit polynomials, PMULL, PMULL2.

This feature is supported in both AArch64 and AArch32 states.

FEAT_PMULL is OPTIONAL from Armv8.0.

If FEAT_PMULL is implemented, then FEAT_AES is implemented.

The following field identifies the presence of FEAT_PMULL:

- ID_AA64ISAR0_EL1.AES.

FEAT_PMUv3, PMU extension version 3

The Performance Monitors Extension, FEAT_PMUv3, is an optional non-invasive debug component.

This feature is supported in both AArch64 and AArch32 states.

FEAT_PMUv3 is OPTIONAL from Armv8.0.

The following fields identify the presence of FEAT_PMUv3:

- ID_AA64DFR0_EL1.PMUVer.
- ID_DFR0_EL1.PerfMon.
- ID_DFR0.PerfMon.
- EDDFR.PMUVer.

For more information, see 'The Performance Monitors Extension'.

FEAT_PMUv3_EXT, External interface to the Performance Monitors

FEAT_PMUv3_EXT indicates support for external access to the PMUv3 registers.

FEAT_PMUv3_EXT is OPTIONAL from Armv8.0.

If FEAT_PMUv3_EXT is implemented, then FEAT_PMUv3 is implemented.

If FEAT_PMUv3_EXT is implemented, then FEAT_PMUv3_EXT32 or FEAT_PMUv3_EXT64 is implemented.

FEAT_PMUv3_EXT32, 32-bit external interface to the Performance Monitors

FEAT_PMUv3_EXT32 indicates the external Performance Monitors and CoreSight registers are implemented as mostly 32-bit registers.

FEAT_PMUv3_EXT32 is OPTIONAL from Armv8.0.

If FEAT_PMUv3_EXT32 is implemented, then FEAT_PMUv3_EXT is implemented.

If FEAT_PMUv3_EXT32 is implemented, then FEAT_PMUv3_EXT64 is not implemented.

The following field identifies the presence of FEAT_PMUv3_EXT32:

- PMDEVARCH.ARCHPART.

For more information, see 'Recommended External Interface to the Performance Monitors'.

FEAT_SHA1, Advanced SIMD SHA1 instructions

FEAT_SHA1 implements the SHA1* instructions.

This feature is supported in both AArch64 and AArch32 states.

FEAT_SHA1 is OPTIONAL from Armv8.0.

If FEAT_SHA1 is implemented, then FEAT_Crypto is implemented.

The following field identifies the presence of FEAT_SHA1:

- ID_AA64ISAR0_EL1.SHA1.

FEAT_SHA256, Advanced SIMD SHA256 instructions

FEAT_SHA256 implements the SHA256* instructions.

This feature is supported in both AArch64 and AArch32 states.

FEAT_SHA256 is OPTIONAL from Armv8.0.

If FEAT_SHA256 is implemented, then FEAT_SHA1 is implemented.

The following field identifies the presence of FEAT_SHA256:

- ID_AA64ISAR0_EL1.SHA2.

FEAT_Secure, Support for Secure state

FEAT_Secure adds support for Secure state.

This feature is not supported if EL2 is using AArch32.

FEAT_Secure is OPTIONAL from Armv8.0.

If FEAT_RME is not implemented and FEAT_EL3 is implemented, then FEAT_Secure is implemented.

FEAT_SpecSEI, SError interrupt exceptions from speculative reads of memory

Describes whether the PE can generate SError interrupt exceptions from speculative reads of memory, including speculative instruction fetches.

FEAT_SpecSEI is OPTIONAL.

The following field identifies the presence of FEAT_SpecSEI:

- ID_AA64MMFR1_EL1.SpecSEI.

FEAT_TGran16K, Support for 16KB memory translation granule size at stage 1

FEAT_TGran16K is OPTIONAL from Armv8.0.

When FEAT_AA64EL2 and FEAT_TGran16K are implemented, FEAT_S2TGran16K is implemented.

The following field identifies the presence of FEAT_TGran16K:

- ID_AA64MMFR0_EL1.TGran16.

FEAT_TGran4K, Support for 4KB memory translation granule size at stage 1

FEAT_TGran4K is OPTIONAL from Armv8.0.

When FEAT_AA64EL2 and FEAT_TGran4K are implemented, FEAT_S2TGran4K is implemented.

The following field identifies the presence of FEAT_TGran4K:

- ID_AA64MMFR0_EL1.TGran4.

FEAT_TGran64K, Support for 64KB memory translation granule size at stage 1

FEAT_TGran64K is OPTIONAL from Armv8.0.

When FEAT_AA64EL2 and FEAT_TGran64K are implemented, FEAT_S2TGran64K is implemented.

The following field identifies the presence of FEAT_TGran64K:

- ID_AA64MMFR0_EL1.TGran64.

FEAT_TRC_EXT, Trace external registers

FEAT_TRC_EXT indicates support for external access to the ETMv4 or ETE registers.

FEAT_TRC_EXT is OPTIONAL from Armv8.0.

If FEAT_TRC_EXT is implemented, then FEAT_ETMv4 or FEAT_ETE is implemented.

FEAT_TRC_SR, Trace System registers

FEAT_TRC_SR indicates support for System registers for ETMv4 or ETE.

This feature is supported in both AArch64 and AArch32, but only when either of these are supported at EL1.

FEAT_TRC_SR is OPTIONAL from Armv8.0.

If FEAT_TRC_SR is implemented, then FEAT_ETMv4 or FEAT_ETE is implemented.

The following fields identify the presence of FEAT_TRC_SR:

- ID_AA64DFR0_EL1.TraceVer.
- EDDFR.TraceVer.
- ID_DFR0.CopTrc.
- ID_DFR0_EL1.CopTrc.

FEAT_nTLBPA, Intermediate caching of translation table walks

FEAT_nTLBPA introduces a mechanism to identify if the intermediate caching of translation table walks does not include non-coherent caches of previous valid translation table entries since the last completed TLBI applicable to the PE.

This feature is supported in both AArch64 and AArch32 states.

FEAT_nTLBPA is OPTIONAL from Armv8.0.

The following fields identify the presence of FEAT_nTLBPA:

- ID_MMFR5.nTLBPA.
- ID_AA64MMFR1_EL1.nTLBPA.
- ID_MMFR5_EL1.nTLBPA.

For more information, see:

- ‘TLB maintenance’.
- ‘General TLB maintenance requirements’.

Features added to the Armv8.0 extension in later releases

- FEAT_CHK.
- FEAT_CLRBHB.
- FEAT_CP15SDISABLE2.
- FEAT_CSV3.
- FEAT_DGH.
- FEAT_ECBHB.
- FEAT_ETS3.

- FEAT_GTG.
- FEAT_PRFMSLC.
- FEAT_RAS.
- FEAT_RPRFM.
- FEAT_SB.
- FEAT_SCTLR2.
- FEAT_SPECRES2.
- FEAT_SPECRES.
- FEAT_SSBS2.
- FEAT_SSBS.
- FEAT_TCR2.

7.2 The Armv8.1 architecture extension

The Armv8.1 architecture extension is an extension to Armv8.0. It adds mandatory and optional architectural features. Some features must be implemented together. An implementation is Armv8.1 compliant if all of the following apply:

- It is Armv8.0 compliant.
- It includes all of the Armv8.1 architectural features that are mandatory.

An Armv8.1 compliant implementation can additionally include:

- Armv8.1 features that are optional.
- Any arbitrary subset of the architectural features of Armv8.2, subject only to those constraints that require that certain features be implemented together.

FEAT_Debugv8p1, Debug v8.1

FEAT_Debugv8p1 is OPTIONAL from Armv8.0.

The following fields identify the presence of FEAT_Debugv8p1:

- ID_AA64DFR0_EL1.DebugVer.
- ID_DFR0_EL1.CopDbg.
- ID_DFR0.CopDbg.
- EDDEVARCH.ARCHVER.

FEAT_E2H0, Programming of HCR_EL2.E2H

When FEAT_VHE is implemented, FEAT_E2H0 indicates that HCR_EL2.E2H can be programmed to the value 0. The absence of FEAT_E2H0 results in a relaxed behavior where HCR_EL2.E2H is RES1, with EL2 host mode always enabled.

This feature is supported in AArch64 state only.

FEAT_E2H0 is OPTIONAL from Armv8.0.

If FEAT_E2H0 is implemented, then FEAT_VHE is implemented.

The following field identifies the presence of FEAT_E2H0:

- ID_AA64MMFR4_EL1.E2H0.

FEAT_HAFDBS, Hardware management of the Access flag and dirty state

In Armv8.0, all updates to the translation tables are performed by software. From Armv8.1, for the VMSAv8-64 translation regimes only, hardware can perform updates to the translation tables in two contexts:

- Hardware management of the Access flag.
- Hardware management of dirty state, with updates to a dirty state in the translation tables.

The dirty state is introduced in Armv8.1.

Hardware management of dirty state can be enabled only when hardware management of the Access flag is also enabled.

This feature is supported in AArch64 state only.

FEAT_HAFDBS is OPTIONAL from Armv8.0.

The following field identifies the presence of FEAT_HAFDBS:

- ID_AA64MMFR1_EL1.HAFDBS.

For more information, see:

- ‘Hardware management of the Access flag’.
- ‘Hardware management of the dirty state’.

FEAT_HPDS, Hierarchical permission disables in translations tables

FEAT_HPDS introduces the facility to disable the hierarchical attributes, APTable, PXNTable, and UXNTable, in the translation tables. This disable has no effect on the NSTable bit.

FEAT_HPDS is OPTIONAL from Armv8.0.

FEAT_HPDS is mandatory from Armv8.1.

The following field identifies the presence of FEAT_HPDS:

- ID_AA64MMFR1_EL1.HPDS.

This feature is added only to the VMSAv8-64 translation regimes. Armv8.2 extends this to the AArch32 translation regimes, see FEAT_AA32HPD.

FEAT_LOR, Limited ordering regions

Limited ordering regions allow large systems to perform special Load-Acquire and Store-Release instructions that provide order between the memory accesses to a region of the PA map as observed by a limited set of observers.

This feature is supported in AArch64 state only.

FEAT_LOR is OPTIONAL from Armv8.0.

FEAT_LOR is mandatory from Armv8.1.

The following field identifies the presence of FEAT_LOR:

- ID_AA64MMFR1_EL1.LO.

For more information, see 'Limited ordering regions'.

FEAT_LSE, Large System Extensions

FEAT_LSE introduces a set of atomic instructions:

- Compare and Swap instructions, CAS and CASP.
- Atomic memory operation instructions, LD<OP> and ST<OP>, where <OP> is one of ADD, CLR, EOR, SET, SMAX, SMIN, UMAX, and UMIN.
- Swap instruction, SWP.

This feature is supported in AArch64 state only.

FEAT_LSE is OPTIONAL from Armv8.0.

FEAT_LSE is mandatory from Armv8.1.

The following field identifies the presence of FEAT_LSE:

- ID_AA64ISAR0_EL1.Atomic.

For more information, see:

- 'Atomic memory operations'
- 'Swap'.
- 'Compare and Swap'.

FEAT_PAN, Privileged access never

FEAT_PAN introduces a bit to PSTATE. When the value of this PAN state bit is 1, any privileged data access from EL1, or EL2 when the Effective value of HCR_EL2.E2H is 1, to a virtual memory address that is accessible to data accesses at EL0, generates a Permission fault.

This feature is supported in both AArch64 and AArch32 states.

FEAT_PAN is OPTIONAL from Armv8.0.

FEAT_PAN is mandatory from Armv8.1.

The following fields identify the presence of FEAT_PAN:

- ID_AA64MMFR1_EL1.PAN.
- ID_MMFR3_EL1.PAN.
- ID_MMFR3.PAN.

For more information, see:

- 'PSTATE.PAN'.
- 'About the PAN bit'.

FEAT_PMUv3p1, Armv8.1 PMU extensions

FEAT_PMUv3p1 introduces the following:

- The event number space is extended to 16 bits.
- The HPMD bit is added to MDCR_EL2. This bit disables event counting at EL2.
- The STALL_FRONTEND and STALL_BACKEND events are required to be implemented.

This feature is supported in both AArch64 and AArch32 states.

FEAT_PMUv3p1 is OPTIONAL from Armv8.0.

In an Armv8.1 implementation, if FEAT_PMUv3 is implemented, FEAT_PMUv3p1 is implemented.

If FEAT_PMUv3p1 is implemented, then FEAT_PMUv3 is implemented.

The following fields identify the presence of FEAT_PMUv3p1:

- ID_AA64DFR0_EL1.PMUVer.
- ID_DFR0_EL1.PerfMon.
- ID_DFR0.PerfMon.
- EDDFR.PMUVer.

For more information, see 'Required events'.

FEAT_RDM, Advanced SIMD rounding double multiply accumulate instructions

FEAT_RDM introduces Rounding Double Multiply Add/Subtract Advanced SIMD instructions. For more information, see:

For the A64 instruction set

- SQRDMLAH (by element).
- SQRDMLAH (vectors).
- SQRDMLSH (by element).
- SQRDMLSH (vector).

For the T32 and A32 instruction sets

- VQRDMLAH.
- VQRDMLSH.

This feature is supported in both AArch64 and AArch32 states.

FEAT_RDM is OPTIONAL from Armv8.0.

In an Armv8.1 implementation, if FEAT_AdvSIMD is implemented, FEAT_RDM is implemented.

The following fields identify the presence of FEAT_RDM:

- ID_AA64ISAR0_EL1.RDM.
- ID_ISAR5_EL1.RDM.
- ID_ISAR5.RDM.

FEAT_VHE, Virtualization Host Extensions

Armv8.1 introduces the *Virtualization Host Extensions* (VHE) that provide enhanced support for Type 2 hypervisors in Non-secure state.

FEAT_VHE is OPTIONAL from Armv8.0.

In an Armv8.1 implementation, if FEAT_AA64EL2 is implemented, FEAT_VHE is implemented.

If FEAT_VHE is implemented, then FEAT_LSE, FEAT_Debugv8p1, and FEAT_AA64EL2 are implemented.

The following field identifies the presence of FEAT_VHE:

- ID_AA64MMFR1_EL1.VH.

For more information, see 'Virtualization Host Extensions'

FEAT_VMID16, 16-bit VMID

In an Armv8.1 implementation, when EL2 is using AArch64, the *virtual machine identifier* (VMID) size is an IMPLEMENTATION DEFINED choice of 8 bits or 16 bits.

When implemented, this feature is supported only when EL2 is using AArch64.

FEAT_VMID16 is OPTIONAL from Armv8.0.

The following field identifies the presence of FEAT_VMID16:

- ID_AA64MMFR1_EL1.VMIDBits.

For more information, see 'VMID size'.

Features added to the Armv8.1 extension in later releases

- FEAT_DPB2.
- FEAT_DotProd.
- FEAT_FHM.
- FEAT_FlagM.
- FEAT_PAN3.

7.3 The Armv8.2 architecture extension

The Armv8.2 architecture extension is an extension to Armv8.1. It adds mandatory and optional architectural features. Some features must be implemented together. An implementation is Armv8.2 compliant if all of the following apply:

- It is Armv8.1 compliant.
- It includes all of the Armv8.2 architectural features that are mandatory.

An Armv8.2 compliant implementation can additionally include:

- Armv8.2 features that are optional.
- Any arbitrary subset of the architectural features of Armv8.3, subject only to those constraints that require that certain features be implemented together.

FEAT_AA32HPD, AArch32 Hierarchical permission disables

FEAT_HPDS introduced the ability to disable the hierarchical attributes, APTable, PXNTable, and UXNTable, in the VMSAv8-64 translation regimes. FEAT_AA32HPD extends this functionality to the VMSAv8-32 translation regimes when those regimes are using the Long descriptor Translation Table format.

This feature is supported in AArch32 state only.

FEAT_AA32HPD is OPTIONAL from Armv8.1.

The following fields identify the presence of FEAT_AA32HPD:

- ID_MMFR4_EL1.HPDS.
- ID_MMFR4.HPDS.

For more information, see ‘Attribute fields in VMSAv8-32 Long-descriptor translation table format descriptors’

FEAT_AA32I8MM, AArch32 Int8 matrix multiply-accumulate

FEAT_AA32I8MM introduces integer matrix multiply-accumulate instructions and dot product instructions.

This feature is supported in AArch32 state only.

FEAT_AA32I8MM is OPTIONAL from Armv8.1.

If FEAT_AA32I8MM is implemented, then FEAT_I8MM is implemented.

The following fields identify the presence of FEAT_AA32I8MM:

- ID_ISAR6_EL1.I8MM.
- ID_ISAR6.I8MM.

For more information, see:

- 'Advanced SIMD dot product instructions'.
- 'Advanced SIMD matrix multiply-accumulate instructions'.

FEAT_ASMv8p2, Armv8.2 changes to the A64 ISA

FEAT_ASMv8p2 introduces the BFC instruction to the A64 instruction set as an alias of BFM. It also requires that the BFC instruction and the A64 pseudo-instruction REV64 are implemented by assemblers.

In Armv8.0 and Armv8.1, the A64 pseudo-instruction REV64 is OPTIONAL.

FEAT_ASMv8p2 is OPTIONAL from Armv8.1.

FEAT_ASMv8p2 is mandatory from Armv8.2.

For more information, see:

- BFC.
- REV64.

FEAT_DPB, DC CVAP instruction

FEAT_DPB introduces a mechanism to identify and manage persistent memory locations in a shared memory hierarchy, including adding the DC CVAP instruction.

This feature is supported in AArch64 state only.

FEAT_DPB is OPTIONAL from Armv8.1.

FEAT_DPB is mandatory from Armv8.2.

The following field identifies the presence of FEAT_DPB:

- ID_AA64ISAR1_EL1.DPB.

For more information, see 'Memory hierarchy'.

FEAT_Debugv8p2, Debug v8.2

FEAT_Debugv8p2 provides the following changes:

- If the Core power domain is powered up and DoubleLockStatus() == TRUE, EDPRSR.{DLK,SPD,PU} is only permitted to read {UNKNOWN, 0, 0}.
- The definition of Exception Catch debug events is extended to include reset entry.
- All CONSTRAINED UNPREDICTABLE cases that generate Exception Catch debug events are removed.
- Controls are added to EDECCR to control Exception Catch debug event generation on exception return.
- All IMPLEMENTATION DEFINED control of external debug accesses to OSLAR_EL1 is removed.
- ExternalSecureNoninvasiveDebugEnabled() cannot override software controls of counting attributable events in Secure state.

FEAT_Debugv8p2 is OPTIONAL from Armv8.1.

If FEAT_Debugv8p2 is implemented, then FEAT_Debugv8p1 is implemented.

FEAT_Debugv8p2 is mandatory from Armv8.2.

The following fields identify the presence of FEAT_Debugv8p2:

- ID_AA64DFR0_EL1.DebugVer.
- ID_DFR0_EL1.CopDbg.
- DBGDIDR.Version.
- ID_DFR0.CopDbg.
- EDDEVARCH.ARCHVER.

For more information, see:

- ‘Exception Catch debug event’.
- ‘EDPRSR.{DLK, SPD, PU} and the Core power domain’.
- ‘Interaction with EL3’.
- ‘External access disabled’.

FEAT_F32MM, Single-precision floating-point matrix multiply-accumulate

FEAT_F32MM introduces support for the SVE single-precision floating-point matrix multiply-accumulate variant of the FMMLA instruction.

This feature is supported in AArch64 state only.

FEAT_F32MM is OPTIONAL from Armv8.2.

If FEAT_F32MM is implemented, then FEAT_SVE is implemented.

The following field identifies the presence of FEAT_F32MM:

- ID_AA64ZFR0_EL1.F32MM.

FEAT_F64MM, Double-precision floating-point matrix multiply-accumulate

FEAT_F64MM introduces support for the following SVE instructions:

- FMMLA (double-precision floating-point matrix multiply-accumulate variant).
- LD1ROB (scalar plus immediate).
- LD1ROB (scalar plus scalar).
- LD1ROD (scalar plus immediate).
- LD1ROD (scalar plus scalar).
- LD1ROH (scalar plus immediate).
- LD1ROH (scalar plus scalar).
- LD1ROW (scalar plus immediate).
- LD1ROW (scalar plus scalar).
- TRN1, TRN2 (vectors) (128-bit variant).
- UZP1, UZP2 (vectors) (128-bit variant).
- ZIP1, ZIP2 (vectors) (128-bit variant).

This feature is supported in AArch64 state only.

FEAT_F64MM is OPTIONAL from Armv8.2.

If FEAT_F64MM is implemented, then FEAT_SVE is implemented.

The following field identifies the presence of FEAT_F64MM:

- ID_AA64ZFR0_EL1.F64MM.

FEAT_FP16, Half-precision floating-point data processing

FEAT_FP16 supports:

- Half-precision data-processing instructions for Advanced SIMD and floating-point in both AArch64 and AArch32 states.

The following fields enable flushing of denormalized numbers to zero for half-precision data-processing instructions:

- FPCR.FZ16.
- FPSCR.FZ16.

This feature is supported in both AArch64 and AArch32 states.

FEAT_FP16 is OPTIONAL from Armv8.2.

In an Armv8.4 implementation, if FEAT_FHM is not implemented, then FEAT_FP16 is not implemented.

In an Armv8.2 implementation, if FEAT_SVE or FEAT_FHM is implemented, FEAT_FP16 is implemented.

The following fields identify the presence of FEAT_FP16:

- ID_AA64PFR0_EL1.FP.
- ID_AA64PFR0_EL1.AdvSIMD.
- MVFR1_EL1.FPHP.
- MVFR1_EL1.SIMDHP.
- MVFR1.FPHP.
- MVFR1.SIMDHP.

For more information, see:

- ‘Half-precision floating-point formats’.
- ‘Flushing denormalized numbers to zero’.
- ‘Modified immediate constants in A64 floating-point instructions’.

FEAT_HPDS2, Hierarchical permission disables

FEAT_HPDS2 provides a mechanism to allow operating systems or hypervisors to make up to four bits of Translation Table final-level descriptors available for IMPLEMENTATION DEFINED hardware use.

This feature is supported in both AArch64 and AArch32 states.

FEAT_HPDS2 is OPTIONAL from Armv8.1.

When FEAT_AA32EL1 and FEAT_HPDS2 are implemented, FEAT_AA32HPD is implemented.

If FEAT_HPDS2 is implemented, then FEAT_HPDS is implemented.

The following fields identify the presence of FEAT_HPDS2:

- ID_AA64MMFR1_EL1.HPDS.
- ID_MMFR4_EL1.HPDS.
- ID_MMFR4.HPDS.

For more information, see:

- ‘Page Based Hardware attributes’.
- ‘Attribute fields in VMSAv8-32 Long-descriptor translation table format descriptors’.

FEAT_I8MM, AArch64 Int8 matrix multiply-accumulate

FEAT_I8MM introduces integer matrix multiply-accumulate instructions and dot product instructions.

This feature is supported in AArch64 state only.

FEAT_I8MM is OPTIONAL from Armv8.1.

FEAT_I8MM is mandatory from Armv8.6.

The following fields identify the presence of FEAT_I8MM:

- ID_AA64ISAR1_EL1.I8MM.
- ID_AA64ZFR0_EL1.I8MM.

For more information, see:

- 'SIMD integer dot product'.
- 'SIMD integer matrix multiply-accumulate'.
- 'SVE Integer dot product'.
- 'SVE Integer matrix multiply operations'.

FEAT_IESB, Implicit Error Synchronization event

FEAT_IESB introduces control for an implicit error synchronization event at exception entry and return.

The implicit error synchronization events affect the same synchronizable asynchronous events that are synchronized by the ESB instruction. This means that if an implementation has either no sources of SError exceptions, or all SError exceptions are not synchronizable errors, then the controls added by this feature have no effect.

This feature is supported in AArch64 state only.

FEAT_IESB is OPTIONAL from Armv8.1.

If FEAT_IESB is implemented, then FEAT_RAS is implemented.

The following field identifies the presence of FEAT_IESB:

- ID_AA64MMFR2_EL1.IESB.

For more information, see 'Error synchronization event'.

FEAT_LPA, Large PA and IPA support

FEAT_LPA:

- Allows a larger *physical address* (PA) and *intermediate physical address* (IPA) space of up to 52 bits when using the 64KB translation granule.
- Allows a level 1 block size where the block covers a 4TB address range for the 64KB translation granule if the implementation support 52 bits of PA.

This feature is supported in AArch64 state only.

FEAT_LPA is OPTIONAL from Armv8.1.

The following field identifies the presence of FEAT_LPA:

- ID_AA64MMFR0_EL1.PARange.

For more information about FEAT_LPA, see:

- ‘Implemented physical address size’.
- ‘Output address size configuration’.
- ‘Intermediate physical address size configuration’
- ‘Translation using the 64KB granule’
- ‘Table descriptor format’
- ‘Block descriptor and Page descriptor formats’

FEAT_LSMAOC, AArch32 Load/Store Multiple instruction atomicity and ordering controls

FEAT_LSMAOC introduces controls that disable legacy behavior of AArch32 load multiple and store multiple instructions, and provide a trap of one aspect of this legacy behavior.

This feature is supported in both AArch64 and AArch32 states.

FEAT_LSMAOC is OPTIONAL from Armv8.1.

The following fields identify the presence of FEAT_LSMAOC:

- ID_AA64MMFR2_EL1.LSM.
- ID_MMFR4_EL1.LSM.
- ID_MMFR4.LSM.

For more information, see the register field descriptions and:

- ‘Generation of Alignment faults by load/store multiple accesses to Device memory’.
- ‘Multi-register loads and stores that access Device memory’.
- ‘Taking an interrupt or other exception during a multiple-register load or store’.

FEAT_LVA, Large VA support

FEAT_LVA supports a *virtual address* (VA) space for each translation table base register of up to 52 bits when using any of the following:

- VMSAv9-128 translation format.
- VMSAv8-64 translation format with the 64KB translation granule.

This feature is supported in AArch64 state only.

FEAT_LVA is OPTIONAL from Armv8.1.

The following field identifies the presence of FEAT_LVA:

- ID_AA64MMFR2_EL1.VARange.

For more information about FEAT_LVA, see:

- ‘Supported virtual address ranges’.
- ‘Input address size configuration’.
- ‘Translation using the 64KB granule’.

FEAT_PAN2, AT S1E1R and AT S1E1W instruction variants affected by PSTATE.PAN

FEAT_PAN2 introduces variants of the AArch64 AT S1E1R and AT S1E1W instructions and the AArch32 ATS1CPR and ATS1CPW instructions. These instructions factor in the PSTATE.PAN bit when determining whether or not the location will generate a Permission fault for a privileged access, as is reported in the PAR. For more information, see:

For the AArch64 System instructions:

- AT S1E1RP, Address Translate Stage 1 EL1 Read PAN.
- AT S1E1RP, Address Translate Stage 1 EL1 Write PA.

For the AArch32 System instructions:

- ATS1CPRP, Address Translate Stage 1 Current state PL1 Read PAN.
- ATS1CPRP, Address Translate Stage 1 Current state PL1 write PAN.

This feature is supported in both AArch64 and AArch32 states.

FEAT_PAN2 is OPTIONAL from Armv8.1.

FEAT_PAN2 is mandatory from Armv8.2.

If FEAT_PAN2 is implemented, then FEAT_PAN is implemented.

The following fields identify the presence of FEAT_PAN2:

- ID_AA64MMFR1_EL1.PAN.
- ID_MMFR3_EL1.PAN.
- ID_MMFR3.PAN.

For more information, see:

- Address translation instructions.
- Address translation instructions.
- ATS1C**, Address translation stage 1, current security state.
- Encoding and availability of the address translation instructions.

FEAT_PCSRv8p2, PC Sample-based Profiling Extension

FEAT_PCSRv8p2 moves the control and implementation of the OPTIONAL PC Sample-based Profiling Extension from the ED*SR Debug registers to PM*SR registers in the Performance Monitors address space.

FEAT_PCSRv8p2 is OPTIONAL from Armv8.1.

If FEAT_PCSRv8p2 is implemented, then FEAT_PCSRv8 is not implemented.

The following field identifies the presence of FEAT_PCSRv8p2:

- PMDEVID.PCSample.

For more information, see 'The PC Sample-based Profiling Extension'.

FEAT_RAS, Reliability, Availability and Serviceability (RAS) Extension

FEAT_RAS introduces the following:

- The *Error Synchronization Barrier* (ESB) instruction for the A32, T32, and A64 instruction sets.
- Error records.
- The Error synchronization event.
- Additional syndrome information on External Aborts.

FEAT_RAS is OPTIONAL from Armv8.0.

FEAT_RAS is mandatory from Armv8.2.

The following fields identify the presence of FEAT_RAS:

- ID_AA64PFR0_EL1.RAS.
- ID_PFR0_EL1.RAS.
- ID_PFR0.RAS.

For more information, see:

- 'Reliability, Availability, and Serviceability'.
- 'RAS PE Architecture'.
- *Arm® Reliability Availability and Serviceability (RAS) System Architecture, for A-profile architecture* (ARM IHI 0100).

FEAT_RASSAv1, RAS System Architecture version 1

FEAT_RASSAv1 implements the RAS System Architecture v1.

FEAT_RASSAv1 is OPTIONAL.

If FEAT_RASSAv1 is implemented, then FEAT_RASSA is implemented.

For more information, see *Arm® Reliability Availablity and Servicability (RAS) System Architecture, for A-profile architecture* (ARM IHI 0100).

FEAT_SHA3, Advanced SIMD SHA3 instructions

FEAT_SHA3 introduces Advanced SIMD instructions that support SHA3 functionality.

These instructions are added to the A64 instruction set only.

FEAT_SHA3 is OPTIONAL from Armv8.1.

If FEAT_SHA3 is implemented, then FEAT_Crypto is implemented.

If FEAT_SHA3 is implemented, then FEAT_SHA256 and FEAT_SHA1 are implemented.

The following field identifies the presence of FEAT_SHA3:

- ID_AA64ISAR0_EL1.SHA3.

For more information, see FEAT_SHA3, SHA3 functionality.

FEAT_SHA512, Advanced SIMD SHA512 instructions

FEAT_SHA512 introduces Advanced SIMD instructions that support SHA2-512 functionality.

These instructions are added to the A64 instruction set only.

FEAT_SHA512 is OPTIONAL from Armv8.1.

If FEAT_SHA512 is implemented, then FEAT_Crypto is implemented.

If FEAT_SHA512 is implemented, then FEAT_SHA256 and FEAT_SHA1 are implemented.

The following field identifies the presence of FEAT_SHA512:

- ID_AA64ISAR0_EL1.SHA2.

For more information, see FEAT_SHA512, SHA2-512 functionality.

FEAT_SM3, Advanced SIMD SM3 instructions

FEAT_SM3 introduces Advanced SIMD instructions that support the Chinese cryptography algorithm SM3.

These instructions are added to the A64 instruction set only.

FEAT_SM3 is OPTIONAL from Armv8.1.

If FEAT_SM3 is implemented, then FEAT_Crypto is implemented.

The following field identifies the presence of FEAT_SM3:

- ID_AA64ISAR0_EL1.SM3.

For more information, see FEAT_SM3, SM3 functionality.

FEAT_SM4, Advanced SIMD SM4 instructions

FEAT_SM4 introduces Advanced SIMD instructions that support the Chinese cryptography algorithm SM4.

These instructions are added to the A64 instruction set only.

FEAT_SM4 is OPTIONAL from Armv8.1.

If FEAT_SM4 is implemented, then FEAT_Crypto is implemented.

The following field identifies the presence of FEAT_SM4:

- ID_AA64ISAR0_EL1.SM4.

For more information, see FEAT_SM4, SM4 functionality.

FEAT_SPE, Statistical Profiling Extension

FEAT_SPE provides a non-invasive method of sampling software and hardware using randomized sampling of either architectural instructions, as defined by the instruction set architecture, or by microarchitectural operations.

This feature is supported in AArch64 state only.

FEAT_SPE is OPTIONAL from Armv8.1.

When FEAT_SPE and FEAT_PMUv3 are implemented, FEAT_PMUv3p1 is implemented.

The following field identifies the presence of FEAT_SPE:

- ID_AA64DFR0_EL1.PMSVer.

For more information, see 'The Statistical Profiling Extension'.

FEAT_SVE, Scalable Vector Extension

The Scalable Vector Extension includes the following functionality:

- Configurable vector length with scalable vector lengths from 128 bits up to 2048 bits.
- Predication using scalable predicate registers from 16 bits up to 256 bits.
- Instructions that operate on scalable size vectors and predicates.
- Gather-load and scatter-store.
- Software-managed speculative vectorization.
- System registers and fields to configure the Effective SVE vector length and traps.

The Scalable Vector Extension complements the AArch64 Advanced SIMD and floating-point functionality. SVE does not replace the AArch64 Advanced SIMD and floating-point functionality.

This feature is supported in AArch64 state only.

FEAT_SVE is OPTIONAL from Armv8.2.

If FEAT_SVE is implemented, then FEAT_FCMA and FEAT_FP16 are implemented.

When FEAT_SVE and FEAT_PMUv3 are implemented, FEAT_PMUv3p1 is implemented.

The following field identifies the presence of FEAT_SVE:

- ID_AA64PFR0_EL1.SVE.

FEAT_TTCNP, Translation table Common not private translations

FEAT_TTCNP permits multiple PEs in the same Inner Shareable domain to use the same translation tables for a given stage of address translation.

This facility is available for all VMSAv8-64 translation regimes and for VMSAv8-32 translation stages that use the Long descriptor Translation Table format.

This feature is supported in both AArch64 and AArch32 states.

FEAT_TTCNP is OPTIONAL from Armv8.1.

FEAT_TTCNP is mandatory from Armv8.2.

The following fields identify the presence of FEAT_TTCNP:

- ID_AA64MMFR2_EL1.CnP.
- ID_MMFR4_EL1.CnP.
- ID_MMFR4.CnP.

For more information, see:

- ‘Common not private translations’.
- ‘Common not private translations in VMSAv8-32’.

FEAT_UAO, Unprivileged Access Override control

FEAT_UAO introduces a bit to PSTATE. When the value of this UAO state bit is 1, and when executed at EL1 or at EL2 with the Effective value of HCR_EL2.{E2H, TGE} == {1, 1} the memory accesses made by the load/store unprivileged instructions behave as if they were made by the load/store register instructions.

This feature is supported in AArch64 state only.

FEAT_UAO is OPTIONAL from Armv8.1.

FEAT_UAO is mandatory from Armv8.2.

The following field identifies the presence of FEAT_UAO:

- ID_AA64MMFR2_EL1.UAO.

For more information, see:

- ‘PSTATE.UAO’.
- ‘Load/store unprivileged’.
- ‘Load/store register’.

FEAT_XNX, Translation table stage 2 Unprivileged Execute-never

FEAT_XNX extends the stage 2 translation table access permissions to provide control of whether memory is executable at EL0 independent of whether it is executable at EL1.

This facility is available for stage 2 translation stages in VMSAv8-64 and VMSAv8-32.

This feature is supported in both AArch64 and AArch32 states.

FEAT_XNX is OPTIONAL from Armv8.1.

In an Armv8.2 implementation, if FEAT_EL2 is implemented, FEAT_XNX is implemented.

The following fields identify the presence of FEAT_XNX:

- ID_AA64MMFR1_EL1.XNX.
- ID_MMFR4_EL1.XNX.
- ID_MMFR4.XNX.

For more information, see:

- 'Stage 2 instruction execution using Direct permissions'.
- 'Access permissions for instruction execution'.

Features added to the Armv8.2 extension in later releases

- FEAT_AA32BF16.
- FEAT_BF16.
- FEAT_EBF16.
- FEAT_EVT.
- FEAT_LRCPC2.
- FEAT_LRCPC3.
- FEAT_LSE2.
- FEAT_MPAM.
- FEAT_PAuth2.
- FEAT_RASSAv1p1.
- FEAT_RASv1p1.

7.4 The Armv8.3 architecture extension

The Armv8.3 architecture extension is an extension to Armv8.2. It adds mandatory and optional architectural features. Some features must be implemented together. An implementation is Armv8.3 compliant if all of the following apply:

- It is Armv8.2 compliant.
- It includes all of the Armv8.3 architectural features that are mandatory.

An Armv8.3 compliant implementation can additionally include:

- Armv8.3 features that are optional.
- Any arbitrary subset of the architectural features of Armv8.4, subject only to those constraints that require that certain features be implemented together.

FEAT_CCIDX, Extended cache index

FEAT_CCIDX introduces the following registers to allow caches to be described with greater numbers of sets and greater associativity:

- A 64-bit format of CCSIDR_EL1.
- CCSIDR2_EL1.
- CCSIDR2.

This feature is supported in both AArch64 and AArch32 states.

FEAT_CCIDX is OPTIONAL from Armv8.2.

The following fields identify the presence of FEAT_CCIDX:

- ID_AA64MMFR2_EL1.CCIDX.
- ID_MMFR4_EL1.CCIDX.
- ID_MMFR4.CCIDX.

For more information, see:

- ‘Possible formats of the Cache Size Identification Register, CCSIDR_EL1’.
- ‘Possible formats of the Cache Size Identification Register, CCSIDR and CCSIDR2’.

FEAT_CONSTPACFIELD, PAC algorithm enhancement

FEAT_CONSTPACFIELD introduces functionality that permits an implementation with pointer authentication to use the value of bit[55] in the virtual address to determine the size of the PAC field when adding a PAC to the virtual address, even when the top byte is not being ignored.

This feature is supported in AArch64 state only.

FEAT_CONSTPACFIELD is OPTIONAL from Armv8.2.

If FEAT_CONSTPACFIELD is implemented, then FEAT_PAuth2 is implemented.

The following field identifies the presence of FEAT_CONSTPACFIELD:

- ID_AA64ISAR2_EL1.PAC_frac.

For more information, see 'PAC field'.

FEAT_DoPD, Debug over Powerdown

FEAT_DoPD provides a debug programmers' model where all debug and PMU registers are in the Core power domain and all CTI registers are in the Debug power domain. Power control is provided by an OPTIONAL external powerup request mechanism.

When FEAT_DoPD is implemented:

- The optional Software Lock is not implemented by the architecturally defined debug components in the PE Core power domain.
- If an ETMv4 trace unit is implemented, the ETM must implement:
  - ETMv4.2 or later.
  - The Unified Power Domain Model.
- If FEAT_ETE is implemented, the trace unit always implements a single power domain.

FEAT_DoPD is OPTIONAL from Armv8.2.

If FEAT_DoPD is implemented, then FEAT_DoubleLock is not implemented.

If FEAT_DoPD is implemented, then FEAT_Debugv8p2 is implemented.

The following field identifies the presence of FEAT_DoPD:

- EDDEVID.DebugPower.

For more information, see 'Debug Reset and Powerdown Support'.

FEAT_EPAC, Enhanced pointer authentication

FEAT_EPAC introduces functionality that permits setting the *Pointer Authentication Code* (PAC) field to 0 on performing a PAC operation on a non-canonical address.

This feature is supported in AArch64 state only.

FEAT_EPAC is OPTIONAL from Armv8.2.

If FEAT_EPAC is implemented, then FEAT_PAuth is implemented.

If FEAT_EPAC is implemented, then FEAT_PAuth2 is not implemented.

The following fields identify the presence of FEAT_EPAC:

- ID_AA64ISAR1_EL1.APA.
- ID_AA64ISAR1_EL1.API.
- ID_AA64ISAR2_EL1.APA3.

For more information, see 'Pointer authentication'.

FEAT_FCMA, Floating-point complex number instructions

FEAT_FCMA introduces instructions for floating-point multiplication and addition of complex numbers.

These instructions are added to the A64 and A32/T32 instruction sets.

FEAT_FCMA is OPTIONAL from Armv8.2.

In an Armv8.3 implementation, if FEAT_FP is implemented, FEAT_FCMA is implemented.

If FEAT_FCMA is implemented, then FEAT_FP is implemented.

The following fields identify the presence of FEAT_FCMA:

- ID_AA64ISAR1_EL1.FCMA.
- ID_ISAR5_EL1.VCMA.
- ID_ISAR5.VCMA.

For more information, see:

- AArch64 Advanced SIMD complex number arithmetic instructions.
- AArch32 Advanced SIMD complex number arithmetic instructions.

FEAT_FPAC, Faulting on AUT* instructions

FEAT_FPAC introduces faulting on an AUT* instruction.

FEAT_FPAC is added as a further extension to FEAT_PAuth2.

This feature is supported in AArch64 state only.

FEAT_FPAC is OPTIONAL from Armv8.2.

If FEAT_FPAC is implemented, then FEAT_PAuth2 is implemented.

The following fields identify the presence of FEAT_FPAC:

- ID_AA64ISAR1_EL1.APA.
- ID_AA64ISAR1_EL1.API.
- ID_AA64ISAR2_EL1.APA3.

For more information, see 'Faulting on pointer authentication'.

FEAT_FPACCOMBINE, Faulting on combined pointer authentication instructions

FEAT_FPACCOMBINE introduces faulting on the combined instructions that perform pointer authentication.

FEAT_FPACCOMBINE is added as a further extension to FEAT_FPAC.

FEAT_FPACCOMBINE is OPTIONAL from Armv8.2.

If FEAT_FPACCOMBINE is implemented, then FEAT_FPAC is implemented.

The following fields identify the presence of FEAT_FPACCOMBINE:

- ID_AA64ISAR1_EL1.APA.
- ID_AA64ISAR1_EL1.API.
- ID_AA64ISAR2_EL1.APA3.

For more information, see 'Faulting on pointer authentication'.

FEAT_FPACC_SPEC, Speculative behavior of pointer authentication instructions

FEAT_FPACC_SPEC introduces consistent impact of speculation for instructions that perform authentication.

FEAT_FPACC_SPEC is added as a further extension to FEAT_FPACCOMBINE.

This feature is supported in AArch64 state only.

FEAT_FPACC_SPEC is OPTIONAL from Armv8.2.

If FEAT_FPACC_SPEC is implemented, then FEAT_FPACCOMBINE is implemented.

The following field identifies the presence of FEAT_FPACC_SPEC:

- ID_AA64MMFR3_EL1.Spec_FPACC.

For more information, see 'Faulting on pointer authentication'.

FEAT_JSCVT, JavaScript conversion instructions

FEAT_JSCVT introduces JavaScript convert instructions that truncate a double-precision value to a 32-bit signed integer, setting the condition flags to indicate whether the converted value was in range.

These instructions are added to the A64 and A32/T32 instruction sets.

FEAT_JSCVT is OPTIONAL from Armv8.2.

In an Armv8.3 implementation, if FEAT_FP is implemented, FEAT_JSCVT is implemented.

If FEAT_JSCVT is implemented, then FEAT_FP is implemented.

The following fields identify the presence of FEAT_JSCVT:

- ID_AA64ISAR1_EL1.JSCVT.
- ID_ISAR6_EL1.JSCVT.

- ID_ISAR6.JSCVT.

For more information, see:

- Floating-point conversion.
- About the A64 Advanced SIMD and floating-point instructions.
- Advanced SIMD and floating-point instructions.
- Floating-point data-processing instructions.

FEAT_LRCPC, Load-Acquire RCpc instructions

FEAT_LRCPC introduces instructions that support the weaker *Release Consistency processor consistent* (RCpc) model that enables the reordering of a Store-Release followed by a Load-Acquire to a different address.

These instructions are added to the A64 instruction set only.

FEAT_LRCPC is OPTIONAL from Armv8.2.

FEAT_LRCPC is mandatory from Armv8.3.

The following field identifies the presence of FEAT_LRCPC:

- ID_AA64ISAR1_EL1.LRCPC.

For more information, see:

- 'Load-Acquire, Load-AcquirePC, and Store-Release'.
- 'Load-Acquire/Store-Release'.

FEAT_NV, Nested Virtualization

FEAT_NV provides support for a Guest Hypervisor to run in EL1 and ensures that the Guest Hypervisor is unaware that it is running at that Exception level. A Guest Hypervisor is supported regardless of the Effective value of HCR_EL2.E2H.

This feature is supported in AArch64 state only.

FEAT_NV is OPTIONAL from Armv8.2.

If FEAT_NV is implemented, then FEAT_EL2 is implemented.

The following fields identify the presence of FEAT_NV:

- ID_AA64MMFR4_EL1.NV_frac.
- ID_AA64MMFR2_EL1.NV.

For more information, see 'Nested virtualization'.

FEAT_PACIMP, Pointer authentication - IMPLEMENTATION DEFINED algorithm

FEAT_PACIMP permits an IMPLEMENTATION DEFINED cryptographic algorithm to be used for PAC calculation.

This feature is supported in AArch64 state only.

FEAT_PACIMP is OPTIONAL from Armv8.2.

If FEAT_PACIMP is implemented, then FEAT_PAuth is implemented.

If FEAT_PACIMP is implemented, then FEAT_PACQARMA5 is not implemented.

If FEAT_PACIMP is implemented, then FEAT_PACQARMA3 is not implemented.

The following fields identify the presence of FEAT_PACIMP:

- ID_AA64ISAR1_EL1.GPI.
- ID_AA64ISAR1_EL1.API.

For more information, see 'Pointer authentication'.

FEAT_PACQARMA3, Pointer authentication - QARMA3 algorithm

FEAT_PACQARMA3 introduces the QARMA3 cryptographic algorithm for PAC calculation.

This feature is supported in AArch64 state only.

FEAT_PACQARMA3 is OPTIONAL from Armv8.2.

If FEAT_PACQARMA3 is implemented, then FEAT_PAuth is implemented.

If FEAT_PACQARMA3 is implemented, then FEAT_PACQARMA5 is not implemented.

If FEAT_PACQARMA3 is implemented, then FEAT_PACIMP is not implemented.

The following fields identify the presence of FEAT_PACQARMA3:

- ID_AA64ISAR2_EL1.GPA3.
- ID_AA64ISAR2_EL1.APA3.

For more information, see 'Pointer authentication'.

FEAT_PACQARMA5, Pointer authentication - QARMA5 algorithm

FEAT_PACQARMA5 introduces the QARMA5 cryptographic algorithm for PAC calculation.

This feature is supported in AArch64 state only.

FEAT_PACQARMA5 is OPTIONAL from Armv8.2.

If FEAT_PACQARMA5 is implemented, then FEAT_PAuth is implemented.

If FEAT_PACQARMA5 is implemented, then FEAT_PACQARMA3 is not implemented.

If FEAT_PACQARMA5 is implemented, then FEAT_PACIMP is not implemented.

The following fields identify the presence of FEAT_PACQARMA5:

- ID_AA64ISAR1_EL1.GPA.
- ID_AA64ISAR1_EL1.APA.

For more information, see 'Pointer authentication'.

FEAT_PAuth, Pointer authentication

FEAT_PAuth introduces functionality that supports address authentication of the contents of a register before that register is used as the target of an indirect branch, or as a load.

When FEAT_PAuth is implemented, all of the following must be true:

- Exactly one of the PAC algorithms is implemented.
- The PACGA instruction and other Pointer authentication instructions use the same algorithm.

The PAC algorithm features are:

- FEAT_PACQARMA5.
- FEAT_PACIMP.
- FEAT_PACQARMA3.

This feature is supported in AArch64 state only.

FEAT_PAuth is OPTIONAL from Armv8.2.

FEAT_PAuth is mandatory from Armv8.3.

If FEAT_PAuth is implemented, then FEAT_PACQARMA5, FEAT_PACIMP, or FEAT_PACQARMA3 is implemented.

For more information, see 'Pointer authentication'.

FEAT_SPEv1p1, Statistical Profiling Extension version 1.1

FEAT_SPEv1p1 introduces an Alignment Flag in the Events packet and filtering on this event using PMSEVFR_EL1, together with support for the profiling of Scalable Vector Extension operations.

This feature is supported in AArch64 state only.

FEAT_SPEv1p1 is OPTIONAL from Armv8.2.

In an Armv8.5 implementation, if FEAT_SPE is implemented, FEAT_SPEv1p1 is implemented.

If FEAT_SPEv1p1 is implemented, then FEAT_SPE is implemented.

The following field identifies the presence of FEAT_SPEv1p1:

- ID_AA64DFR0_EL1.PMSVer.

An implementation that includes FEAT_SVE and the Statistical Profiling Extension is strongly recommended to implement FEAT_SPEv1p1 whenever possible.

For more information, see:

- 'The Statistical Profiling Extension'.
- 'Statistical Profiling Extension Sample Record Specification'.

7.5 The Armv8.4 architecture extension

The Armv8.4 architecture extension is an extension to Armv8.3. It adds mandatory and optional architectural features. Some features must be implemented together. An implementation is Armv8.4 compliant if all of the following apply:

- It is Armv8.3 compliant.
- It includes all of the Armv8.4 architectural features that are mandatory.

An Armv8.4 compliant implementation can additionally include:

- Armv8.4 features that are optional.
- Any arbitrary subset of the architectural features of Armv8.5, subject only to those constraints that require that certain features be implemented together.

FEAT_AMU_EXT, External Activity Monitors

FEAT_AMU_EXT indicates support for external access to the Activity Monitors.

FEAT_AMU_EXT is OPTIONAL.

If FEAT_AMU_EXT is implemented, then FEAT_AMU_EXT32 or FEAT_AMU_EXT64 is implemented.

If FEAT_AMU_EXT is implemented, then FEAT_AMUv1 is implemented.

FEAT_AMU_EXT32, AArch32 External Activity Monitors

FEAT_AMU_EXT32 indicates the external AMU registers are implemented as mostly 32-bit registers.

FEAT_AMU_EXT32 is OPTIONAL.

If FEAT_AMU_EXT32 is implemented, then FEAT_AMU_EXT is implemented.

If FEAT_AMU_EXT32 is implemented, then FEAT_AMU_EXT64 is not implemented.

The following field identifies the presence of FEAT_AMU_EXT32:

- AMDEVARCH.ARCHID.

For more information, see ‘Recommended External Interface to the Activity Monitors’.

FEAT_AMUv1, Activity Monitors Extension version 1

FEAT_AMUv1 provides a function similar to a subset of the existing Performance Monitors Extension functionality, intended for system management use rather than debugging and profiling.

This feature is supported in both AArch64 and AArch32 states.

FEAT_AMUv1 is OPTIONAL from Armv8.3.

The following fields identify the presence of FEAT_AMUv1:

- ID_AA64PFR0_EL1.AMU.
- ID_PFR0_EL1.AMU.
- ID_PFR0.AMU.
- EDPFR.AMU.

For more information, see ‘The Activity Monitors Extension’.

FEAT_BBM, Translation table break-before-make levels

FEAT_BBM provides support to identify the requirements of hardware to have break-before-make sequences when changing between block size for a translation.

This feature is supported in AArch64 state only.

FEAT_BBM is OPTIONAL from Armv8.3.

FEAT_BBM is mandatory from Armv8.4.

The following field identifies the presence of FEAT_BBM:

- ID_AA64MMFR2_EL1.BBM.

For more information, see:

- ‘Block descriptor and Page descriptor formats’.
- ‘Block translationentry’.
- ‘Support levels for changing block size’.

FEAT_CNTSC, Generic Counter Scaling

FEAT_CNTSC introduces a scaling register to the memory-mapped counter module that allows the frequency of the counter that is generated to be scaled from the basic frequency reported in the counter ID mechanisms.

This feature is supported in both AArch64 and AArch32 states.

FEAT_CNTSC is OPTIONAL from Armv8.3.

The following field identifies the presence of FEAT_CNTSC:

- CNTID.CNTSC.

For more information, see CNTCR.

FEAT_DIT, Data Independent Timing instructions

FEAT_DIT provides independent timing for data processing instructions.

This feature is supported in both AArch64 and AArch32 states.

FEAT_DIT is OPTIONAL from Armv8.3.

FEAT_DIT is mandatory from Armv8.4.

The following fields identify the presence of FEAT_DIT:

- ID_AA64PFR0_EL1.DIT.
- ID_PFR0_EL1.DIT.
- ID_PFR0.DIT.

For more information, see:

- 'About PSTATE.DIT'.
- 'About the DIT bit'.

FEAT_Debugv8p4, Debug v8.4

FEAT_Debugv8p4 provides the following mandatory changes:

- The fields MDCR_EL3.{EPMAD, EDAD} control Non-secure access to the debug and PMU registers. The bus Requester is responsible for other debug authentication.
- The Software Lock is obsolete.
- Non-invasive Debug controls are relaxed.
- Secure and Non-secure views of the debug registers are enabled.

FEAT_Debugv8p4 is OPTIONAL from Armv8.3.

FEAT_Debugv8p4 is mandatory from Armv8.4.

If FEAT_Debugv8p4 is implemented, then FEAT_Debugv8p2 is implemented.

If FEAT_SEL2 is implemented, then FEAT_Debugv8p4 is implemented.

The following fields identify the presence of FEAT_Debugv8p4:

- ID_DFR0_EL1.CopDbg.
- ID_AA64DFR0_EL1.DebugVer.

- DBGDIDR.Version.
- ID_DFR0.CopDbg.
- EDDEVARCH.ARCHVER.

For more information, see:

- Definition and constraints of a debugger in the context of external debug.
- External debug interface register access permissions.

FEAT_DotProd, Advanced SIMD dot product instructions

FEAT_DotProd provides instructions to perform a four-way vector dot product of 8-bit integers, accumulating each sum of four products into a 32-bit integer. Each 8-bit input can be treated as a signed or unsigned value.

These instructions are added to the A64 and A32/T32 instruction sets.

FEAT_DotProd is OPTIONAL from Armv8.1.

In an Armv8.4 implementation, if FEAT_AdvSIMD is implemented, FEAT_DotProd is implemented.

The following fields identify the presence of FEAT_DotProd:

- ID_AA64ISAR0_EL1.DP.
- ID_ISAR6_EL1.DP.
- ID_ISAR6.DP.

For more information, see:

- 'SIMD integer dot product'.
- 'Advanced SIMD dot product instructions'.

FEAT_DoubleFault, Double Fault Extension

FEAT_DoubleFault provides controls for routing and masking error exceptions.

This feature is supported in AArch64 state only.

FEAT_DoubleFault is OPTIONAL from Armv8.3.

In an Armv8.4 implementation, if FEAT_AA64EL3 is implemented, FEAT_DoubleFault is implemented.

If FEAT_DoubleFault is implemented, then FEAT_DoubleFault2 or FEAT_AA64EL3 is implemented.

The following field identifies the presence of FEAT_DoubleFault:

- ID_AA64PFR0_EL1.RAS.

For more information, see:

- ‘Taking error exceptions’.
- ‘Error synchronization event’.

FEAT_FHM, Floating-point half-precision to single-precision multiply-add instructions

FEAT_FHM introduces half-precision to single-precision fused multiply-add instructions.

These instructions are added to the A64 and A32/T32 instruction sets.

FEAT_FHM is OPTIONAL from Armv8.1.

In an Armv8.4 implementation, if FEAT_FP16 is implemented, FEAT_FHM is implemented.

In an Armv8.2 implementation, if FEAT_FHM is implemented, FEAT_FP16 is implemented.

The following fields identify the presence of FEAT_FHM:

- ID_AA64ISAR0_EL1.FHM.
- ID_ISAR6_EL1.FHM.
- ID_ISAR6.FHM.

For more information, see:

- ‘SIMD arithmetic’.
- ‘SIMD by element arithmetic’.
- ‘Advanced SIMD multiply instructions’.

FEAT_FlagM, Condition flag manipulation instructions

FEAT_FlagM provides instructions that manipulate the PSTATE.{N,Z,C,V} flags.

These instructions are added to the A64 instruction set only.

FEAT_FlagM is OPTIONAL from Armv8.1.

FEAT_FlagM is mandatory from Armv8.4.

The following field identifies the presence of FEAT_FlagM:

- ID_AA64ISAR0_EL1.TS.

For more information, see ‘Flag manipulation instructions’.

FEAT_IDST, ID space trap handling

FEAT_IDST causes all AArch64 read accesses to the feature ID space when exceptions are generated to be reported in ESR_ELx using the EC code `0x18`.

This feature is supported in AArch64 state only.

FEAT_IDST is OPTIONAL from Armv8.3.

FEAT_IDST is mandatory from Armv8.4.

The following field identifies the presence of FEAT_IDST:

- ID_AA64MMFR2_EL1.IDS.

FEAT_LRCPC2, Load-Acquire RCpc instructions version 2

FEAT_LRCPC2 provides versions of `LDAPR` and `STLR` with a 9-bit unscaled signed immediate offset.

These instructions are added to the A64 instruction set only.

FEAT_LRCPC2 is OPTIONAL from Armv8.2.

FEAT_LRCPC2 is mandatory from Armv8.4.

If FEAT_LRCPC2 is implemented, then FEAT_LRCPC is implemented.

The following field identifies the presence of FEAT_LRCPC2:

- ID_AA64ISAR1_EL1.LRCPC.

For more information, see:

- ‘Changes to single-copy atomicity in Armv8.4’.
- ‘Load-Acquire/Store-Release’.
- ‘A64 instructions that are changed in Debug state’.

FEAT_LSE2, Large System Extensions version 2

FEAT_LSE2 introduces changes to single-copy atomicity requirements for loads and stores, and changes to alignment requirements for loads and stores.

This feature is supported in AArch64 state only.

FEAT_LSE2 is OPTIONAL from Armv8.2.

FEAT_LSE2 is mandatory from Armv8.4.

The following field identifies the presence of FEAT_LSE2:

- ID_AA64MMFR2_EL1.AT.

For more information, see:

- ‘Requirements for single-copy atomicity’.
- ‘Alignment of data accesses’.

FEAT_MPAM, Memory Partitioning and Monitoring Extension

The MPAM Extension provides a framework for memory-system component controls that partition one or more of the performance resources of the component.

This feature is supported in AArch64 state only.

FEAT_MPAM is OPTIONAL from Armv8.2.

The following field identifies the presence of FEAT_MPAM:

- ID_AA64PFR0_EL1.MPAM.

For more information, see 'MPAM PE Architecture'.

FEAT_NV2, Enhanced nested virtualization support

FEAT_NV2 supports nested virtualization by redirecting register accesses that would be trapped to EL1 and EL2 to access memory instead. The address of the memory access depends on information held in VNCR_EL2.

This feature is supported in AArch64 state only.

FEAT_NV2 is OPTIONAL from Armv8.3.

If FEAT_NV2 is implemented, then FEAT_NV is implemented.

The following fields identify the presence of FEAT_NV2:

- ID_AA64MMFR4_EL1.NV_frac.
- ID_AA64MMFR2_EL1.NV.

For more information, see 'Enhanced support for nested virtualization'.

FEAT_PMUv3p4, Arm8.4 PMU extensions

FEAT_PMUv3p4 introduces:

- PMMIR_EL1
- PMMIR registers.

This feature is supported in both AArch64 and AArch32 states.

FEAT_PMUv3p4 is OPTIONAL from Armv8.3.

In an Armv8.4 implementation, if FEAT_PMUv3 is implemented, FEAT_PMUv3p4 is implemented.

If FEAT_PMUv3p4 is implemented, then FEAT_PMUv3p1 is implemented.

The following fields identify the presence of FEAT_PMUv3p4:

- ID_AA64DFR0_EL1.PMUVer.
- ID_DFR0_EL1.PerfMon.
- ID_DFR0.PerfMon.
- EDDFR.PMUVer.

For more information, see PMU Event Descriptions.

FEAT_RASSA_ACR, Access Control Register

FEAT_RASSA_ACR is OPTIONAL.

If FEAT_RASSA_ACR is implemented, then FEAT_RASSAv1p1 and FEAT_RASSA_GRP are implemented.

FEAT_RASSA_GRP, RAS groups

FEAT_RASSA_GRP is OPTIONAL.

If FEAT_RASSA_GRP is implemented, then FEAT_RASSA is implemented.

FEAT_RASSAv1p1, RAS version 1.1 System Architecture

FEAT_RASSAv1p1 implements the RAS System Architecture v1.1, including support for the following:

- Additional ERR<n>MISC<m> registers.
- The optional RAS Common Fault Injection Model Extension.

FEAT_RASSAv1p1 is OPTIONAL from Armv8.2.

In an Armv8.4 implementation, if FEAT_RAS is implemented, FEAT_RASSAv1p1 is implemented.

If FEAT_RASv1p1 is implemented, then FEAT_RASSAv1p1 is implemented.

FEAT_RASv1p1, RAS extension v1.1

FEAT_RASv1p1 introduces support for System register access to the following RAS System Architecture v1.1 features:

- Additional ERR<n>MISC<m> registers.
- The optional RAS Common Fault Injection Model Extension.

This feature is supported in both AArch64 and AArch32 states.

FEAT_RASv1p1 is OPTIONAL from Armv8.2.

In an Armv8.4 implementation, if FEAT_RAS is implemented, FEAT_RASv1p1 is implemented.

If FEAT_RASv1p1 is implemented, then FEAT_RAS is implemented.

If FEAT_RASv1p1 is implemented, then FEAT_RASSAv1p1 is implemented.

The following fields identify the presence of FEAT_RASv1p1:

- ID_AA64PFR0_EL1.RAS.
- ID_AA64PFR1_EL1.RAS_frac.
- ID_PFR0_EL1.RAS.

- ID_PFR2_EL1.RAS_frac.
- ID_PFR0.RAS.
- ID_PFR2.RAS_frac.

For more information, see *Arm® Reliability Availability and Serviceability (RAS) System Architecture, for A-profile architecture* (ARM IHI 0100).

FEAT_S2FWB, Stage 2 forced Write-Back

FEAT_S2FWB reduces the requirement of additional cache maintenance instructions in systems where the data Cacheability attributes used by the Guest operating system are different from those expected by the Hypervisor. If this feature is implemented, there is no meaningful distinction between the Inner and Outer Shareability domains for accesses to Normal Cacheable memory.

This feature is supported in AArch64 state only.

FEAT_S2FWB is OPTIONAL from Armv8.3.

In an Armv8.4 implementation, if FEAT_EL2 is implemented, FEAT_S2FWB is implemented.

The following field identifies the presence of FEAT_S2FWB:

- ID_AA64MMFR2_EL1.FWB.

For more information, see:

- 'Block descriptor and Page descriptor formats'.
- 'Stage 2 memory type and Cacheability attributes when FEAT_S2FWB is enabled'.

FEAT_SEL2, Secure EL2

FEAT_SEL2 permits EL2 to be implemented in Secure state. When Secure EL2 is enabled, a translation regime is introduced that follows the same format as the other Secure translation regimes.

This feature is not supported if EL2 is using AArch32.

FEAT_SEL2 is OPTIONAL from Armv8.3.

In an Armv8.4 implementation, if FEAT_AA64EL2 and FEAT_Secure are implemented, FEAT_SEL2 is implemented.

If FEAT_SEL2 is implemented, then FEAT_TTST is implemented.

If FEAT_SEL2 is implemented, then FEAT_PCSRv8 is not implemented.

If FEAT_SEL2 is implemented, then FEAT_EL2 is implemented.

If FEAT_SEL2 is implemented, then FEAT_Secure is implemented.

The following field identifies the presence of FEAT_SEL2:

- ID_AA64PFR0_EL1.SEL2.

For more information, see:

- ‘Security states’
- ‘Translation regimes’.

FEAT_TLBIOS, TLB invalidate instructions in Outer Shareable domain

FEAT_TLBIOS provides TLBI maintenance instructions that extend to the Outer Shareable domain.

This feature is supported in AArch64 state only.

FEAT_TLBIOS is OPTIONAL from Armv8.3.

FEAT_TLBIOS is mandatory from Armv8.4.

The following field identifies the presence of FEAT_TLBIOS:

- ID_AA64ISAR0_EL1.TLB.

For more information, see ‘TLB maintenance instructions’.

FEAT_TLBIRANGE, TLB invalidate range instructions

FEAT_TLBIRANGE provides TLBI maintenance instructions that apply to a range of input addresses.

This feature is supported in AArch64 state only.

FEAT_TLBIRANGE is OPTIONAL from Armv8.3.

FEAT_TLBIRANGE is mandatory from Armv8.4.

If FEAT_TLBIRANGE is implemented, then FEAT_TLBIOS is implemented.

The following field identifies the presence of FEAT_TLBIRANGE:

- ID_AA64ISAR0_EL1.TLB.

For more information, see:

- ‘TLB maintenance instructions’.
- ‘TLB maintenance instructions that do not apply to a range of addresses’.

FEAT_TRF, Self-hosted Trace extensions

FEAT_TRF introduces control of trace in a self-hosted system through System registers.

The feature provides:

- Control of Exception levels and Security states where trace generation is prohibited.
- Control of whether an offset is used for the timestamp recorded with trace information.

- A context synchronization instruction, TSB CSYNC, that can be used to prevent reordering of trace operation accesses with respect to other accesses of the same System registers.

This feature is supported in both AArch64 and AArch32 states.

FEAT_TRF is OPTIONAL from Armv8.3.

In an Armv8.4 implementation, if FEAT_ETMv4 is implemented, FEAT_TRF is implemented.

If FEAT_TRF is implemented, then FEAT_TRC_SR is implemented.

The following fields identify the presence of FEAT_TRF:

- ID_DFR0_EL1.TraceFilt.
- ID_DFR0.TraceFilt.
- ID_AA64DFR0_EL1.TraceFilt.
- EDDFR.TraceFilt.

For more information on FEAT_TRF, see:

- 'AArch64 Self-hosted Trace'.
- 'AArch32 Self-hosted Trace'.

FEAT_TTL, Translation Table Level

FEAT_TTL provides the TTL field to indicate the level of translation table walk holding the leaf entry for the address that is being invalidated. This field is provided in all TLB maintenance instructions that take a VA or an IPA argument.

This feature is supported in AArch64 state only.

FEAT_TTL is OPTIONAL from Armv8.3.

FEAT_TTL is mandatory from Armv8.4.

The following field identifies the presence of FEAT_TTL:

- ID_AA64MMFR2_EL1.TTL.

For more information, see:

- 'TLB maintenance instructions'.
- 'TLB maintenance instructions that do not apply to a range of addresses'.

FEAT_TTST, Small translation tables

FEAT_TTST relaxes the lower limit on the size of translation tables, by increasing the maximum permitted value of the T1SZ and T0SZ fields in TCR_EL1, TCR_EL2, TCR_EL3, VTCR_EL2 and VSTCR_EL2.

This feature is supported in AArch64 state only.

FEAT_TTST is OPTIONAL from Armv8.3.

If FEAT_SEL2 is implemented, then FEAT_TTST is implemented.

The following field identifies the presence of FEAT_TTST:

- ID_AA64MMFR2_EL1.ST.

For more information, see:

- 'Input address size configuration'.
- 'Translation using the 4KB granule'.
- 'Translation using the 16KB granule'.
- 'Translation using the 64KB granule'.

7.6 The Armv8.5 architecture extension

The Armv8.5 architecture extension is an extension to Armv8.4. It adds mandatory and optional architectural features. Some features must be implemented together. An implementation is Armv8.5 compliant if all of the following apply:

- It is Armv8.4 compliant.
- It includes all of the Armv8.5 architectural features that are mandatory.

An Armv8.5 compliant implementation can additionally include:

- Armv8.5 features that are optional.
- Any arbitrary subset of the architectural features of Armv8.6, subject only to those constraints that require that certain features be implemented together.

FEAT_BTI, Branch Target Identification

FEAT_BTI allows memory pages to be guarded against the execution of instructions that are not the intended target of a branch. To do this, it introduces:

- The GP field, which denotes the blocks and pages in stage 1 translation tables that are guarded pages.
- The PSTATE.BTYPE field, which is used to determine whether an access to a guarded memory region will generate a Branch Target exception.
- The BTI instruction, which is used to guard against the execution of instructions that are not the intended target of a branch.

This feature is supported in AArch64 state only.

FEAT_BTI is OPTIONAL from Armv8.4.

FEAT_BTI is mandatory from Armv8.5.

The following field identifies the presence of FEAT_BTI:

- ID_AA64PFR1_EL1.BT.

For more information, see:

- ‘Table descriptor format’.
- ‘PSTATE.BTYPE’.
- ‘Effect of entering Debug state on PSTATE’.

FEAT_CSV3, Cache Speculation Variant 3

FEAT_CSV3 introduces a mechanism to identify whether data loaded or read from a register under speculation where the data load or register read would not be permitted architecturally, can be used by instructions newer than the load or register read in a manner that allows the value of the inaccessible data to be recovered by code architecturally executed.

This feature is supported in both AArch64 and AArch32 states.

FEAT_CSV3 is OPTIONAL from Armv8.0.

FEAT_CSV3 is mandatory from Armv8.5.

The following field identifies the presence of FEAT_CSV3:

- ID_AA64PFR0_EL1.CSV3.

FEAT_DPB2, DC CVADP instruction

FEAT_DPB2 allows two levels of cache clean to the ‘Point of Persistence’ by:

- Redefining ‘Point of Persistence’, which changes the scope of DC CVAP.
- Defining a ‘Point of Deep Persistence’.
- Adding the DC CVADP System instruction.

This feature is supported in AArch64 state only.

FEAT_DPB2 is OPTIONAL from Armv8.1.

FEAT_DPB2 is mandatory from Armv8.5.

If FEAT_DPB2 is implemented, then FEAT_DPB is implemented.

The following field identifies the presence of FEAT_DPB2:

- ID_AA64ISAR1_EL1.DPB.

For more information, see ‘Terminology for Clean, Invalidate, and Clean and Invalidate instruction’.

FEAT_E0PD, Preventing EL0 access to halves of address maps

FEAT_E0PD prevents access at EL0 to half of the addresses in the memory map.

This feature is supported in AArch64 state only. When EL1 is using AArch64 state, this feature affects access to EL0, in either Execution state.

FEAT_E0PD is OPTIONAL from Armv8.4.

FEAT_E0PD is mandatory from Armv8.5.

If FEAT_E0PD is implemented, then FEAT_CSV3 is implemented.

The following field identifies the presence of FEAT_E0PD:

- ID_AA64MMFR2_EL1.E0PD.

For more information, see 'Preventing EL0 access to halves of the address map'.

FEAT_EVT, Enhanced Virtualization Traps

FEAT_EVT introduces additional traps for EL1 and EL0 Cache controls in:

- HCR_EL2
- HCR2.

These traps are independent of existing controls.

This feature is supported in both AArch64 and AArch32 states.

FEAT_EVT is OPTIONAL from Armv8.2.

In an Armv8.5 implementation, if FEAT_EL2 is implemented, FEAT_EVT is implemented.

The following fields identify the presence of FEAT_EVT:

- ID_AA64MMFR2_EL1.EVT.
- ID_MMFR4_EL1.EVT.
- ID_MMFR4.EVT.

FEAT_ExS, Context synchronization and exception handling

FEAT_ExS provides a mechanism to control whether exception entry and exception return are Context Synchronization events.

Fields in the SCTLR_ELx registers enable and disable context synchronization at exception entry and return at an Exception level.

This feature is supported in AArch64 state only.

FEAT_ExS is OPTIONAL from Armv8.4.

The following field identifies the presence of FEAT_ExS:

- ID_AA64MMFR0_EL1.ExS.

For more information, see ‘Context Synchronization event’.

FEAT_FRINTTS, Floating-point to integer instructions

FEAT_FRINTTS provides instructions that round a floating-point number to an integral valued floating-point number that fits in a 32-bit or 64-bit integer number range.

These instructions are added to the A64 instruction set only.

FEAT_FRINTTS is OPTIONAL from Armv8.4.

In an Armv8.5 implementation, if FEAT_FP is implemented, FEAT_FRINTTS is implemented.

If FEAT_FRINTTS is implemented, then FEAT_FP and FEAT_AdvSIMD are implemented.

The following field identifies the presence of FEAT_FRINTTS:

- ID_AA64ISAR1_EL1.FRINTTS.

For more information, see ‘Floating-point round to integral value’.

FEAT_FlagM2, Enhancements to flag manipulation instructions

FEAT_FlagM2 provides instructions that convert between the PSTATE condition flag format used by the FCMP instruction and an alternative format described in ‘Condition flags and related instructions’.

These instructions are added to the A64 instruction set only.

FEAT_FlagM2 is OPTIONAL from Armv8.4.

If FEAT_FlagM2 is implemented, then FEAT_FlagM is implemented.

The following field identifies the presence of FEAT_FlagM2:

- ID_AA64ISAR0_EL1.TS.

For more information, see ‘Flag manipulation instructions’.

FEAT_GTG, Guest translation granule size

FEAT_GTG allows a hypervisor to support different granule sizes for stage 2 and stage 1 translation, and allows a nested hypervisor to determine what stage 2 granule sizes are available.

This feature is supported in AArch64 state only.

FEAT_GTG is OPTIONAL from Armv8.0.

In an Armv8.5 implementation, if FEAT_AA64EL2 is implemented, FEAT_GTG is implemented.

The following fields identify the presence of FEAT_GTG:

- ID_AA64MMFR0_EL1.TGran4_2.

- ID_AA64MMFR0_EL1.TGran16_2.
- ID_AA64MMFR0_EL1.TGran64_2.

For more information, see 'Translation granules'.

FEAT_MTE, Memory Tagging Extension

FEAT_MTE provides architectural support for runtime, always-on detection of various classes of memory error to aid with software debugging to eliminate vulnerabilities arising from memory-unsafe languages.

These features are supported in AArch64 state only.

FEAT_MTE is OPTIONAL from Armv8.4.

The following field identifies the presence of FEAT_MTE:

- ID_AA64PFR1_EL1.MTE.

For more information, see:

- 'The Memory Tagging Extension'.
- 'The AArch64 Application Level Memory Model'.
- PMU Event Descriptions.
- 'The Statistical Profiling Extension'.
- 'Debug State'.

FEAT_MTE2, Memory Tagging Extension

FEAT_MTE2 provides architectural support for runtime, always-on detection of various classes of memory error to aid with software debugging to eliminate vulnerabilities arising from memory-unsafe languages.

These features are supported in AArch64 state only.

FEAT_MTE2 is OPTIONAL from Armv8.4.

If FEAT_MTE2 is implemented, then FEAT_MTE is implemented.

The following field identifies the presence of FEAT_MTE2:

- ID_AA64PFR1_EL1.MTE.

For more information, see:

- 'The Memory Tagging Extension'.
- 'The AArch64 Application Level Memory Model'.
- PMU Event Descriptions.
- 'The Statistical Profiling Extension'.

- ‘Debug State’.

FEAT_PMUv3p5, Arm8.5 PMU extensions

FEAT_PMUv3p5 extends event counters to 64-bit event counters, and introduces mechanisms to disable the cycle counter in Secure state, in EL3, and in EL2.

FEAT_PMUv3p5 relaxes the behavior of PMCR.{IMP, IDCODE}, and deprecates use of these fields.

This feature is supported in both AArch64 and AArch32 states.

FEAT_PMUv3p5 is OPTIONAL from Armv8.4.

In an Armv8.5 implementation, if FEAT_PMUv3 is implemented, FEAT_PMUv3p5 is implemented.

If FEAT_PMUv3p5 is implemented, then FEAT_PMUv3p4 is implemented.

The following fields identify the presence of FEAT_PMUv3p5:

- ID_AA64DFR0_EL1.PMUVer.
- ID_DFR0_EL1.PerfMon.
- ID_DFR0.PerfMon.
- EDDFR.PMUVer.

For more information, see:

- ‘Behavior on overflow’.
- ‘Controlling the PMU counters’.
- PMU Event Descriptions.

FEAT_RNG, Random number generator

FEAT_RNG introduces the RNDR and RNDRRS registers. Reads to these registers return a 64-bit random number. A read to RNDRRS will cause a reseeding of the random number before the generation of the random number that is returned.

This feature is supported in AArch64 state only.

FEAT_RNG is OPTIONAL from Armv8.4.

The following field identifies the presence of FEAT_RNG:

- ID_AA64ISAR0_EL1.RNDR.

For more information, see:

- ‘Effect of random number generation instructions on Condition flags’
- ‘Appendix K14 Random Number Generation’

FEAT_RNG_TRAP, Trapping support for RNDR/RNDRRS

FEAT_RNG_TRAP introduces support for EL3 trapping of reads of the RNDR and RNDRRS registers.

This feature is supported in AArch64 state only.

FEAT_RNG_TRAP is OPTIONAL from Armv8.4.

The following field identifies the presence of FEAT_RNG_TRAP:

- ID_AA64PFR1_EL1.RNDR_trap.

FEAT_S2TGran16K, Support for 16KB memory translation granule size at stage 2

FEAT_S2TGran16K is OPTIONAL.

If FEAT_S2TGran16K is implemented, then FEAT_AA64EL2 and FEAT_TGran16K are implemented.

The following field identifies the presence of FEAT_S2TGran16K:

- ID_AA64MMFR0_EL1.TGran16_2.

FEAT_S2TGran4K, Support for 4KB memory translation granule size at stage 2

FEAT_S2TGran4K is OPTIONAL.

If FEAT_S2TGran4K is implemented, then FEAT_AA64EL2 and FEAT_TGran4K are implemented.

The following field identifies the presence of FEAT_S2TGran4K:

- ID_AA64MMFR0_EL1.TGran4_2.

FEAT_S2TGran64K, Support for 64KB memory translation granule size at stage 2

FEAT_S2TGran64K is OPTIONAL.

If FEAT_S2TGran64K is implemented, then FEAT_AA64EL2 and FEAT_TGran64K are implemented.

The following field identifies the presence of FEAT_S2TGran64K:

- ID_AA64MMFR0_EL1.TGran64_2.

FEAT_SB, Speculation Barrier

Speculation Barrier FEAT_SB introduces a barrier to control speculation.

This feature is supported in both AArch64 and AArch32 states.

FEAT_SB is OPTIONAL from Armv8.0.

FEAT_SB is mandatory from Armv8.5.

The following fields identify the presence of FEAT_SB:

- ID_AA64ISAR1_EL1.SB.
- ID_ISAR6_EL1.SB.
- ID_ISAR6.SB.

For more information, see:

- ‘Speculation Barrier (SB)’.
- ‘Barriers and CLREX instructions’.
- ‘Speculation Barrier (SB)’.
- ‘Miscellaneous instructions’.

FEAT_SPECRES, Speculation restriction instructions

FEAT_SPECRES introduces System instructions that prevent predictions based on information gathered from earlier execution within a particular execution context from affecting the later speculative execution within that context, to the extent that the speculative execution is observable through side channels.

This feature is supported in both AArch64 and AArch32 states.

FEAT_SPECRES is OPTIONAL from Armv8.0.

FEAT_SPECRES is mandatory from Armv8.5.

The following fields identify the presence of FEAT_SPECRES:

- ID_AA64ISAR1_EL1.SPECRES.
- ID_ISAR6_EL1.SPECRES.
- ID_ISAR6.SPECRES.

For more information, see:

- ‘Prediction restriction instructions’.
- ‘Execution, data prediction and prefetching restriction System instructions’.
- ‘Execution and data prediction restriction System instructions’.

FEAT_SSBS, Speculative Store Bypass Safe

FEAT_SSBS allows software to indicate whether hardware is permitted to load or store speculatively in a manner that could give rise to a cache timing side channel, which in turn could be used to derive an address from values loaded to a register from memory.

This feature is supported in both AArch64 and AArch32 states.

FEAT_SSBS is OPTIONAL from Armv8.0.

The following fields identify the presence of FEAT_SSBS:

- ID_AA64PFR1_EL1.SSBS.
- ID_PFR2_EL1.SSBS.
- ID_PFR2.SSBS.

For more information, see:

- AArch64 Speculative Store Bypass Safe.
- AArch32 Speculative Store Bypass Safe.

FEAT_SSBS2, MRS and MSR instructions for SSBS version 2

FEAT_SSBS2 provides controls for the MSR and MRS instructions to read and write the PSTATE.SSBS field.

This feature is supported in AArch64 state only.

FEAT_SSBS2 is OPTIONAL from Armv8.0.

If FEAT_SSBS2 is implemented, then FEAT_SSBS is implemented.

The following field identifies the presence of FEAT_SSBS2:

- ID_AA64PFR1_EL1.SSBS.

For more information, see:

- AArch64 Speculative Store Bypass Safe.
- AArch32 Speculative Store Bypass Safe.

Features added to the Armv8.5 extension in later releases

- FEAT_MTE3.
- FEAT_MTE_ASYNC.

7.7 The Armv8.6 architecture extension

The Armv8.6 architecture extension is an extension to Armv8.5. It adds mandatory and optional architectural features. Some features must be implemented together. An implementation is Armv8.6 compliant if all of the following apply:

- It is Armv8.5 compliant.
- It includes all of the Armv8.6 architectural features that are mandatory.

An Armv8.6 compliant implementation can additionally include:

- Armv8.6 features that are optional.
- Any arbitrary subset of the architectural features of Armv8.7, subject only to those constraints that require that certain features be implemented together.

FEAT_AA32BF16, AArch32 BFloat16 instructions

FEAT_AA32BF16 supports the BFloat16, or BF16, 16-bit floating-point storage format in AArch32 state. This format supports:

- Arithmetic instructions to multiply BFloat16 values and accumulate into single-precision results.
- Arithmetic instructions to accelerate dot product and matrix-multiply accumulate of BFloat16 values into single-precision results.
- Instructions to convert single-precision floating-point values to BFloat16 format.

This feature is supported in AArch32 state only.

FEAT_AA32BF16 is OPTIONAL from Armv8.2.

If FEAT_AA32BF16 is implemented, then FEAT_BF16 is implemented.

The following fields identify the presence of FEAT_AA32BF16:

- ID_ISAR6_EL1.BF16.
- ID_ISAR6.BF16.

For more information, see:

- 'BFloat16 floating-point format'.
- 'Advanced SIMD BFloat16 instructions'.
- 'Floating-point data-processing'.

FEAT_AMUv1p1, Activity Monitors Extension version 1.1

FEAT_AMUv1p1 introduces support for virtualization of Activity Monitors event counters, and introduces controls to disable access to auxiliary event counters below the highest Exception level.

This feature is supported in AArch32 state and AArch64 state, if the hypervisor is using AArch64.

FEAT_AMUv1p1 is OPTIONAL from Armv8.5.

If FEAT_AMUv1p1 is implemented, then FEAT_AMUv1 is implemented.

The following fields identify the presence of FEAT_AMUv1p1:

- ID_AA64PFR0_EL1.AMU.
- ID_PFR0_EL1.AMU.
- ID_PFR0.AMU.
- EDPFR.AMU.

For more information, see 'The Activity Monitors Extension'.

FEAT_BF16, AArch64 BFloat16 instructions

FEAT_BF16 supports the BFloat16, or BF16, 16-bit floating-point storage format in AArch64 state. This format supports:

- Arithmetic instructions to multiply BFloat16 values and accumulate into single-precision results.
- Arithmetic instructions to accelerate dot product and matrix-multiply accumulate of BFloat16 values into single-precision results.
- Instructions to convert single-precision floating-point values to BFloat16 format.

This feature is supported in AArch64 state only.

FEAT_BF16 is OPTIONAL from Armv8.2.

In an Armv8.6 implementation, if FEAT_FP is implemented, FEAT_BF16 is implemented.

The following fields identify the presence of FEAT_BF16:

- ID_AA64ISAR1_EL1.BF16.
- ID_AA64ZFR0_EL1.BF16.

For more information, see:

- 'BFloat16 floating-point format'.
- 'Summary of BFloat16 instruction behaviors'.

FEAT_CP15SDISABLE2, CP15SDISABLE2

FEAT_CP15SDISABLE2 provides an implementation-defined mechanism, the **CP15SDISABLE2** signal, which when asserted HIGH prevents writes to a set of Secure CP15 registers. This signal is analogous to the existing **CP15SDISABLE** signal.

This feature is supported only when EL3 is executing in AArch32 state.

FEAT_CP15SDISABLE2 is OPTIONAL from Armv8.0.

When FEAT_AA32 and FEAT_CP15SDISABLE2 are implemented, FEAT_AA32EL3 is implemented.

For more information, see 'The CP15SDISABLE and CP15SDISABLE2 input signals'.

FEAT_DGH, Data Gathering Hint

FEAT_DGH introduces the Data Gathering Hint instruction to the hint space.

This instruction is added to the A64 instruction set only.

FEAT_DGH is OPTIONAL from Armv8.0.

The following field identifies the presence of FEAT_DGH:

- ID_AA64ISAR1_EL1.DGH.

For more information, see 'Hint instructions'.

FEAT_ECV, Enhanced Counter Virtualization

FEAT_ECV enhances the Generic Timer architecture. FEAT_ECV provides:

- Self-synchronizing views of the virtual and physical timers in AArch64 state and AArch32 state.
- The ability to scale the generation of the event stream when executing in AArch64 state or AArch32 state.
- When EL2 is using AArch64 state, traps to EL0 and EL1 access to the virtual counter or timer registers, and the physical timer registers when accessed using an EL02 mnemonic. The traps are configured in CNTHCTL_EL2, and apply to EL1 and EL0 accesses, whether EL1 and EL0 are in AArch64 state or AArch32 state.

For more information on the offset to views of physical time, see FEAT_ECV_POFF.

This feature is supported in both AArch64 and AArch32 states.

FEAT_ECV is OPTIONAL from Armv8.5.

FEAT_ECV is mandatory from Armv8.6.

The following field identifies the presence of FEAT_ECV:

- ID_AA64MMFR0_EL1.ECV.

For more information, see:

- 'Self-hosted trace timestamps'.
- 'The profiling data'.
- 'The AArch64 view of the Generic Timer'.
- 'The AArch32 view of the Generic Timer'.

FEAT_ECV_POFF, Enhanced Counter Virtualization Physical Offset

FEAT_ECV_POFF provides an offset between the EL1 or EL0 view of physical time, and the EL2 or EL3 view of physical time.

The offset to views of physical time at EL1 and EL0 apply in AArch64 state and AArch32 state.

This feature is supported in both AArch64 and AArch32 states.

FEAT_ECV_POFF is OPTIONAL from Armv8.5.

If FEAT_ECV_POFF is implemented, then FEAT_ECV is implemented.

When FEAT_AA64 and FEAT_ECV_POFF are implemented, FEAT_AA64EL2 is implemented.

If FEAT_RME is implemented, then FEAT_ECV_POFF is implemented.

The following field identifies the presence of FEAT_ECV_POFF:

- ID_AA64MMFR0_EL1.ECV.

FEAT_FGT, Fine Grain Traps

FEAT_FGT introduces additional traps to EL2 of EL1 and EL0 access to individual or small groups of System registers and instructions, and traps to EL3 and EL2 of the Debug Communications Channel registers. The traps are independent of existing controls.

This feature is supported in AArch64, and when EL1 is using AArch64, EL0 accesses using AArch32 are also trapped.

FEAT_FGT is OPTIONAL from Armv8.5.

In an Armv8.6 implementation, if FEAT_AA64EL2 or FEAT_AA64EL3 is implemented, FEAT_FGT is implemented.

The following field identifies the presence of FEAT_FGT:

- ID_AA64MMFR0_EL1.FGT.

For more information, see 'Configurable instruction controls'.

FEAT_HPMN0, Setting of MDCR_EL2.HPMN to zero

FEAT_HPMN0 permits a hypervisor to provide zero PMU event counters for a guest operating system by setting MDCR_EL2.HPMN to zero.

This feature is supported in both AArch64 and AArch32 states.

FEAT_HPMN0 is OPTIONAL from Armv8.5.

In an Armv8.8 implementation, if FEAT_PMUv3 and FEAT_EL2 are implemented, FEAT_HPMN0 is implemented.

If FEAT_HPMN0 is implemented, then FEAT_EL2 is implemented.

If FEAT_HPMN0 is implemented, then FEAT_PMUv3 and FEAT_FGT are implemented.

The following fields identify the presence of FEAT_HPMN0:

- ID_AA64DFR0_EL1.HPMN0.
- ID_DFR1_EL1.HPMN0.
- ID_DFR1.HPMN0.

For more information, see:

- 'Interaction with EL2'.
- 'Controlling the PMU counters'.
- 'The Performance Monitors Extension'.
- 'The Performance Monitors Extension'.

FEAT_MPAMv0p1, Memory Partitioning and Monitoring extension version 0.1

FEAT_MPAMv0p1 introduces support for version 0.1 of the MPAM extension.

This feature is supported in AArch64 state only.

FEAT_MPAMv0p1 is OPTIONAL from Armv8.5.

The following fields identify the presence of FEAT_MPAMv0p1:

- ID_AA64PFR0_EL1.MPAM.
- ID_AA64PFR1_EL1.MPAM_frac.

For more information, see 'MPAM PE Architecture'.

FEAT_MPAMv1p1, Memory Partitioning and Monitoring extension version 1.1

FEAT_MPAMv1p1 introduces support for version 1.1 of the MPAM extension.

This feature is supported in AArch64 state only.

FEAT_MPAMv1p1 is OPTIONAL from Armv8.5.

If FEAT_MPAMv1p1 is implemented, then FEAT_MPAM is implemented and FEAT_MPAMv0p1 is not implemented.

The following fields identify the presence of FEAT_MPAMv1p1:

- ID_AA64PFR0_EL1.MPAM.
- ID_AA64PFR1_EL1.MPAM_frac.

For more information, see 'MPAM PE Architecture'.

FEAT_MTPMU, Multi-threaded PMU extensions

FEAT_MTPMU introduces controls to disable PMEVTYPER<n>_EL0.MT.

From Armv8.6, when FEAT_PMUv3 is implemented, multithreaded event counting is only supported in multithreaded implementations that also include FEAT_MTPMU.

This feature is supported in both AArch64 and AArch32 states.

FEAT_MTPMU is OPTIONAL from Armv8.5.

If FEAT_MTPMU is implemented, then FEAT_PMUv3 is implemented.

If FEAT_MTPMU is implemented, then FEAT_EL2 or FEAT_EL3 is implemented.

The following fields identify the presence of FEAT_MTPMU:

- ID_AA64DFR0_EL1.MTPMU.
- ID_DFR1.MTPMU.

- ID_DFR1_EL1.MTPMU.

For more information, see:

- ‘Multithreaded implementations’.
- MDCR_EL3.MTPME
- SDCR.MTPME
- MDCR_EL2.MTPME
- HDCR.MTPME.

FEAT_PAuth2, Enhancements to pointer authentication

FEAT_PAuth2 introduces enhanced pointer authentication functionality that changes the mechanism by which a PAC is added to the pointer.

This feature is supported in AArch64 state only.

FEAT_PAuth2 is OPTIONAL from Armv8.2.

FEAT_PAuth2 is mandatory from Armv8.6.

If FEAT_PAuth2 is implemented, then FEAT_PAuth is implemented.

If FEAT_PAuth2 is implemented, then FEAT_EPAC is not implemented.

The following fields identify the presence of FEAT_PAuth2:

- ID_AA64ISAR1_EL1.APA.
- ID_AA64ISAR1_EL1.API.
- ID_AA64ISAR2_EL1.APA3.

For more information, see ‘Pointer authentication’.

FEAT_TWED, Delayed Trapping of WFE

FEAT_TWED introduces support for configurable delayed trapping of the WFE instruction.

FEAT_TWED is OPTIONAL from Armv8.5.

The following field identifies the presence of FEAT_TWED:

- ID_AA64MMFR1_EL1.TWED.

For more information, see ‘The Wait for Event and Wait for Event with Timeout instructions’.

Features added to the Armv8.6 extension in later releases

- FEAT_SPE_DPFZS.

7.8 The Armv8.7 architecture extension

The Armv8.7 architecture extension is an extension to Armv8.6. It adds mandatory and optional architectural features. Some features must be implemented together. An implementation is Armv8.7 compliant if all of the following apply:

- It is Armv8.6 compliant.
- It includes all of the Armv8.7 architectural features that are mandatory.

An Armv8.7 compliant implementation can additionally include:

- Armv8.7 features that are optional.
- Any arbitrary subset of the architectural features of Armv8.8, subject only to those constraints that require that certain features be implemented together.

FEAT_AFP, Alternate floating-point behavior

FEAT_AFP allows alternate behavior for specified floating-point instructions including:

- Flushing of denormalized numbers to zero can be controlled separately on inputs and outputs.
- Alternate NaN propagation rules and Default NaN values can apply.
- Certain scalar SIMD and floating-point instructions can be configured to preserve higher numbered SIMD vector elements.
- Changes to floating-point exception generation.

This feature is supported in AArch64 state only.

FEAT_AFP is OPTIONAL from Armv8.6.

If FEAT_AFP is implemented, then FEAT_FP is implemented.

In an Armv8.7 implementation, if FEAT_FP is implemented, FEAT_AFP is implemented.

The following field identifies the presence of FEAT_AFP:

- ID_AA64MMFR1_EL1.AFP.

For more information, see:

- ‘Flushing denormalized numbers to zero’.
- ‘NaN handling and the Default NaN’.
- ‘Rounding’.
- ‘Floating-point exceptions and exception traps’.

FEAT_EBF16, AArch64 Extended BFloat16 behaviors

FEAT_EBF16 supports the Extended BFloat16 behaviors.

This feature is supported in AArch64 state only.

FEAT_EBF16 is OPTIONAL from Armv8.2.

If FEAT_EBF16 is implemented, then FEAT_BF16 is implemented.

If FEAT_EBF16 is implemented, then FEAT_AdvSIMD, FEAT_FP, FEAT_SVE, or FEAT_SME is implemented.

The following fields identify the presence of FEAT_EBF16:

- ID_AA64ISAR1_EL1.BF16.
- ID_AA64ZFR0_EL1.BF16.

For more information, see:

- 'BFloat16 floating-point format'.
- 'Floating-point behaviors for instructions that compute sum-of-products'.

FEAT_HCX, Support for the HCRX_EL2 register

FEAT_HCX introduces the Extended Hypervisor Configuration Register, HCRX_EL2, that provides configuration controls for virtualization in addition to those provided by HCR_EL2, including defining whether various operations are trapped to EL2.

This feature is supported in AArch64 state only.

FEAT_HCX is OPTIONAL from Armv8.6.

In an Armv8.7 implementation, if FEAT_AA64EL2 is implemented, FEAT_HCX is implemented.

When FEAT_AA64 and FEAT_HCX are implemented, FEAT_AA64EL2 is implemented.

The following field identifies the presence of FEAT_HCX:

- ID_AA64MMFR1_EL1.HCX.

For more information, see 'Configurable instruction controls'.

FEAT_LPA2, Larger physical address for 4KB and 16KB translation granules

FEAT_LPA2:

- Allows a larger VA space for each translation table base register of up to 52 bits when using the 4KB or 16KB translation granules.
- Allows a larger intermediate physical address (IPA) and PA space of up to 52 bits when using the 4KB or 16KB translation granules.
- Allows a level 0 block size where the block covers a 512GB address range for the 4KB translation granule if the implementation supports 52 bits of PA.
- Allows a level 1 block size where the block covers a 64GB address range for the 16KB translation granule if the implementation supports 52 bits of PA.

This feature is supported in AArch64 state only.

FEAT_LPA2 is OPTIONAL from Armv8.6.

If FEAT_LPA2 is implemented, then FEAT_LVA is implemented.

The following fields identify the presence of FEAT_LPA2:

- ID_AA64MMFR0_EL1.TGran4_2.
- ID_AA64MMFR0_EL1.TGran16_2.
- ID_AA64MMFR0_EL1.TGran4.
- ID_AA64MMFR0_EL1.TGran16.

For more information, see:

- 'Implemented physical address size'.
- 'Output address size configuration'.
- 'Supported virtual address ranges'.
- 'Input address size configuration'.
- 'Intermediate physical address size configuration'.
- 'Translation using the 4KB granule'.
- 'Translation using the 16KB granule'.
- 'Table descriptor format'.
- 'Block descriptor and Page descriptor formats'.

FEAT_LS64, Support for 64-byte loads and stores without status

FEAT_LS64 introduces support for single-copy atomic 64-byte loads and stores without status result. For more information, see:

- LD64B.
- ST64B.

This feature is supported in AArch64 state only.

FEAT_LS64 is OPTIONAL from Armv8.6.

The following field identifies the presence of FEAT_LS64:

- ID_AA64ISAR1_EL1.LS64.

For more information, see 'Single-copy atomic 64-byte load/store'.

FEAT_LS64_ACCDATA, Support for 64-byte EL0 stores with status

FEAT_LS64_ACCDATA introduces support for single-copy atomic 64-byte EL0 stores with status result. For more information, see:

- ST64BV0.
- ACCDATA_EL1.


Note

The meaning of any status being returned by the ST64BV0 instruction is defined by the peripheral providing the response.

This feature is supported in AArch64 state only.

FEAT_LS64_ACCDATA is OPTIONAL from Armv8.6.

If FEAT_LS64_ACCDATA is implemented, then FEAT_LS64_V is implemented.

The following field identifies the presence of FEAT_LS64_ACCDATA:

- ID_AA64ISAR1_EL1.LS64.

For more information, see 'Single-copy atomic 64-byte load/store'.

FEAT_LS64_V, Support for 64-byte stores with status

FEAT_LS64_V introduces support for single-copy atomic 64-byte stores with status result. For more information, see:

- ST64BV.


Note

The meaning of any status being returned by the ST64BV instruction is defined by the peripheral providing the response.

This feature is supported in AArch64 state only.

FEAT_LS64_V is OPTIONAL from Armv8.6.

If FEAT_LS64_V is implemented, then FEAT_LS64 is implemented.

The following field identifies the presence of FEAT_LS64_V:

- ID_AA64ISAR1_EL1.LS64.

For more information, see 'Single-copy atomic 64-byte load/store'.

FEAT_MTE3, MTE Asymmetric Fault Handling

FEAT_MTE3 introduces support for asymmetric Tag Check Fault handling.

This feature is supported in AArch64 state only.

FEAT_MTE3 is OPTIONAL from Armv8.5.

In an Armv8.7 implementation, if FEAT_MTE_ASYNC is implemented, FEAT_MTE3 is implemented.

If FEAT_MTE3 is implemented, then FEAT_MTE2 is implemented.

The following field identifies the presence of FEAT_MTE3:

- ID_AA64PFR1_EL1.MTE.

For more information, see 'The Memory Tagging Extension'.

FEAT_MTE_ASYM_FAULT, Memory tagging asymmetric faults

FEAT_MTE_ASYM_FAULT introduces support for asymmetric MTE Tag Check fault handling.

This feature is supported in AArch64 state only.

FEAT_MTE_ASYM_FAULT is OPTIONAL.

If FEAT_MTE3 is implemented, then FEAT_MTE_ASYM_FAULT is implemented.

If FEAT_MTE_ASYM_FAULT is implemented, then FEAT_MTE3 is implemented.

If FEAT_MTE_ASYM_FAULT is implemented, then FEAT_MTE_ASYNC is implemented.

The following field identifies the presence of FEAT_MTE_ASYM_FAULT:

- ID_AA64PFR1_EL1.MTE.

FEAT_PAN3, Support for SCTLR_ELx.EPAN

FEAT_PAN3 introduces a bit to SCTLR_EL1 and SCTLR_EL2, EPAN, to support using Privileged Access Never with instruction accesses for stage 1 translation regimes.

This feature is supported in AArch64 state only.

FEAT_PAN3 is OPTIONAL from Armv8.1.

FEAT_PAN3 is mandatory from Armv8.7.

If FEAT_PAN3 is implemented, then FEAT_PAN2 is implemented.

The following field identifies the presence of FEAT_PAN3:

- ID_AA64MMFR1_EL1.PAN.

For more information, see PSTATE PAN.

FEAT_PMUv3p7, Armv8.7 PMU extensions

FEAT_PMUv3p7 introduces the following to the Performance Monitors Extension:

- PMU counters can be frozen when an event counter has an unsigned overflow.
- Event counters can be prohibited from counting events at EL3 without affecting other Exception levels in Secure state.

- The cycle counter can be prohibited from counting cycles at EL3 without affecting other Exception levels in Secure state.

This feature is supported in both AArch64 and AArch32 states.

FEAT_PMUv3p7 is OPTIONAL from Armv8.6.

In an Armv8.7 implementation, if FEAT_PMUv3 is implemented, FEAT_PMUv3p7 is implemented.

If FEAT_PMUv3p7 is implemented, then FEAT_PMUv3p5 is implemented.

The following fields identify the presence of FEAT_PMUv3p7:

- ID_AA64DFR0_EL1.PMUVer.
- ID_DFR0_EL1.PerfMon.
- ID_DFR0.PerfMon.
- EDDFR.PMUVer.
- PMCFGR.FZO.

For more information, see:

- 'Controlling the PMU counters'.
- 'Freezing event counters'.

FEAT_RPRES, Increased precision of FRECPE and FRSQRTE

FEAT_RPRES allows an increase in the precision of the single-precision floating-point reciprocal estimate and reciprocal square root estimate from an 8-bit mantissa to a 12-bit mantissa.

This feature is supported in AArch64 state only.

FEAT_RPRES is OPTIONAL from Armv8.6.

If FEAT_RPRES is implemented, then FEAT_AFP is implemented.

The following field identifies the presence of FEAT_RPRES:

- ID_AA64ISAR2_EL1.RPRES.

For more information, see `RecipEstimate()` and `RecipSqrtEstimate()`.

FEAT_SPE_FnE, Statistical Profiling inverse event filter

FEAT_SPE_FnE provides the capability to filter sample records by the inverse value of bits in the sampled 'Events packet'.

This feature is supported in AArch64 state only.

FEAT_SPE_FnE is OPTIONAL.

If FEAT_SPE_FnE is implemented, then FEAT_SPEv1p2 is implemented.

If FEAT_SPEv1p2 is implemented, then FEAT_SPE_FnE is implemented.

The following field identifies the presence of FEAT_SPE_FnE:

- PMSIDR_EL1.FnE.

For more information, see 'Filtering sample records'.

FEAT_SPE_PBT, Statistical Profiling previous branch target

FEAT_SPE_PBT provides support for generating a packet that provides the target address for the previous taken branch.

This feature is supported in AArch64 state only.

FEAT_SPE_PBT is OPTIONAL.

If FEAT_SPE_PBT is implemented, then FEAT_SPEv1p2 is implemented.

The following field identifies the presence of FEAT_SPE_PBT:

- PMSIDR_EL1.PBT.

For more information, see:

- 'Previous branch target'.
- 'Address packet'.

FEAT_SPEv1p2, Statistical Profiling Extensions version 1.2

FEAT_SPEv1p2 introduces the following to the Statistical Profiling Extension:

- Controls to freeze the PMU event counters after an SPE buffer management event occurs.
- A discard mode that allows all SPE data to be discarded rather than written to memory.

This feature is supported in AArch64 state only.

FEAT_SPEv1p2 is OPTIONAL from Armv8.6.

If FEAT_SPEv1p2 is implemented, then FEAT_SPEv1p1 is implemented.

In an Armv8.7 implementation, if FEAT_SPE is implemented, FEAT_SPEv1p2 is implemented.

The following field identifies the presence of FEAT_SPEv1p2:

- ID_AA64DFR0_EL1.PMSVer.

For more information, see:

- 'Freezing event counters'.
- 'Discard mode'.

FEAT_WFxT, WFE and WFI instructions with timeout

FEAT_WFxT introduces WFET and WFIT. These instructions support the generation of a local timeout event to act as a wake-up event for the PE when the virtual count in CNTVCT_EL0 equals or exceeds the value supplied by the instruction for the first time.

These instructions are added to the A64 instruction set only.

FEAT_WFxT is OPTIONAL from Armv8.6.

FEAT_WFxT is mandatory from Armv8.7.

The following field identifies the presence of FEAT_WFxT:

- ID_AA64ISAR2_EL1.WFxT.

For more information, see:

- Wait for Event.
- Wait for Interrupt mechanism.

FEAT_XS, XS attribute

FEAT_XS introduces the XS attribute for memory to indicate that an access could take a long time to complete. This feature provides variants of DSB instructions and TLB maintenance instructions, the completion of which does not depend on the completion of memory accesses with the XS attribute.

FEAT_XS adds:

- A mechanism to define the XS attribute for memory.
- An optional nXS variant to the AArch64 DSB instruction and OPTIONAL nXS qualifier to each AArch64 TLBI instruction to handle memory accesses with the XS attribute.
- The HCRX_EL2.FGTnXS bit to determine the behavior of fine-grained traps in HFGITR_EL2 for TLB maintenance instructions with the nXS qualifier.
- The HCRX_EL2.FnXS bit to determine the behavior of pre-existing TLB maintenance instructions in relation to the XS attribute.

This feature is supported in AArch64 state only, but the XS attribute also impacts AArch32 state execution.

FEAT_XS is OPTIONAL from Armv8.6.

FEAT_XS is mandatory from Armv8.7.

The following field identifies the presence of FEAT_XS:

- ID_AA64ISAR1_EL1.XS.

For more information, see:

- ‘Data Synchronization Barrier (DSB)’.

- ‘Behavior when stage 1 address translation is disabled’.
- ‘Block descriptor and Page descriptor formats’.
- ‘Stage 1 memory type and Cacheability attributes’.
- ‘XS attribute modifier’.
- ‘Overview of memory region attributes for stage 1 translations’.
- ‘Ordering and completion of TLB maintenance instructions’.

Features added to the Armv8.7 extension in later releases

- FEAT_CSSC.
- FEAT_HAFT.
- FEAT_MTE4.
- FEAT_MTE_PERM.

7.9 The Armv8.8 architecture extension

The Armv8.8 architecture extension is an extension to Armv8.7. It adds mandatory and optional architectural features. Some features must be implemented together. An implementation is Armv8.8 compliant if all of the following apply:

- It is Armv8.7 compliant.
- It includes all of the Armv8.8 architectural features that are mandatory.

An Armv8.8 compliant implementation can additionally include:

- Armv8.8 features that are optional.
- Any arbitrary subset of the architectural features of Armv8.9, subject only to those constraints that require that certain features be implemented together.

FEAT_CMOW, Control for cache maintenance permission

FEAT_CMOW introduces support for controlling the required permissions for some cache maintenance instructions that operate by VA and perform translation such that:

- Stage 1 can be configured to generate a Permission fault if write permission is not present for cache maintenance instructions executed at EL0.
- Stage 2 can be configured to generate a Permission fault if write permission is not present for cache maintenance instructions executed at EL1 or EL0.

This feature is supported in AArch64 state only, but also impacts AArch32 instructions.

FEAT_CMOW is OPTIONAL from Armv8.7.

FEAT_CMOW is mandatory from Armv8.8.

The following field identifies the presence of FEAT_CMOW:

- ID_AA64MMFR1_EL1.CMOW.

For more information, see:

- ‘A64 Cache maintenance instructions’.
- ‘Permission fault’.

FEAT_Debugv8p8, Debug v8.8

FEAT_Debugv8p8 introduces support to allow an asynchronous exception to be taken after an exception generates an Exception Catch debug event, but before the PE halts.

This feature is supported in both AArch64 and AArch32 states.

FEAT_Debugv8p8 is OPTIONAL from Armv8.7.

FEAT_Debugv8p8 is mandatory from Armv8.8.

If FEAT_Debugv8p8 is implemented, then FEAT_Debugv8p4 is implemented.

The following fields identify the presence of FEAT_Debugv8p8:

- ID_AA64DFR0_EL1.DebugVer.
- ID_DFR0_EL1.CopDbg.
- DBGDIDR.Version.
- ID_DFR0.CopDbg.
- EDDEVARCH.ARCHVER.

For more information, see ‘Exception Catch debug event’.

FEAT_HBC, Hinted conditional branches

FEAT_HBC provides the BC.cond instruction to give a conditional branch with a hint to branch prediction logic that this branch will behave consistently and is highly unlikely to change direction.

This feature is supported in AArch64 state only.

FEAT_HBC is OPTIONAL from Armv8.7.

FEAT_HBC is mandatory from Armv8.8.

The following field identifies the presence of FEAT_HBC:

- ID_AA64ISAR2_EL1.BC.

For more information, see ‘Conditional branch’.

FEAT_MOPS, Standardization of memory operations

FEAT_MOPS provides instructions that perform a memory copy or memory set, and introduces Memory Copy and Memory Set exceptions.

FEAT_MOPS also introduces the HCRX_EL2.{MSCEn, MCE2}, SCTLR_EL1.MSCEn, and SCTLR_EL2.MSCEn control bits.

This feature is supported in AArch64 state only.

FEAT_MOPS is OPTIONAL from Armv8.7.

FEAT_MOPS is mandatory from Armv8.8.

The following field identifies the presence of FEAT_MOPS:

- ID_AA64ISAR2_EL1.MOPS.

For more information, see:

- 'Memory Copy and Memory Set instructions'.
- 'Memory Copy and Memory Set exceptions'.

FEAT_NMI, Non-maskable Interrupts

FEAT_NMI provides a mechanism to support *non-maskable interrupts* (NMI) and *less-masked interrupts* (LMI). In addition to legacy behavior, the feature includes the following:

- A mode for supporting an LMI interrupt mask that is distinct from PSTATE.{I, F}.
- A mode for supporting a limited NMI, where the value when PSTATE.SP is 1 is taken as an interrupt mask for all interrupts targeting that Exception level, and where the LMI interrupt mask can also be used

FEAT_NMI adds:

- The AllIntMask variable.
- An Optional Superpriority attribute to denote virtual and physical IRQ and FIQ interrupts as non-maskable.
- The SCTLR_ELx.{NMI, SPINTMASK} control bits.
- The PSTATE.ALLINT bit and associated instructions.
- The HCRX_EL2.TALLINT bit to enable trapping of ALLINT instructions at EL1.

This feature is supported in AArch64 state only.

FEAT_NMI is OPTIONAL from Armv8.7.

FEAT_NMI is mandatory from Armv8.8.

The following field identifies the presence of FEAT_NMI:

- ID_AA64PFR1_EL1.NMI.

For more information, see:

- ‘Asynchronous exception types’.
- ‘Virtual interrupts’.
- ‘PSTATE fields that are meaningful in AArch64 state’.
- ‘WFE wake-up events’.

FEAT_PMUv3_EXT64, 64-bit external interface to the Performance Monitors

FEAT_PMUv3_EXT64 indicates the external Performance Monitors registers are implemented as 64-bit registers. The 32-bit CoreSight management registers remain 32-bit registers.

FEAT_PMUv3_EXT64 is OPTIONAL from Armv8.8.

If FEAT_PMUv3_EXT64 is implemented, then FEAT_PMUv3_EXT is implemented.

If FEAT_PMUv3_EXT64 is implemented, then FEAT_PMUv3_EXT32 is not implemented.

The following field identifies the presence of FEAT_PMUv3_EXT64:

- PMDEVARCH.ARCHPART.

For more information, see ‘Recommended External Interface to the Performance Monitors’.

FEAT_PMUv3_TH, Event counting threshold

FEAT_PMUv3_TH introduces threshold condition controls to each PMEVTYPER<n>_EL0 register. This feature permits the counter to count only when PMEVTYPER<n>.{MT, evtCount} describe an event whose count meets a specified threshold condition.

This feature is supported in both AArch64 and AArch32 states. The threshold condition controls are accessible only in AArch64 state. However, threshold conditions still apply in AArch32 state.

FEAT_PMUv3_TH is OPTIONAL from Armv8.7.

If FEAT_PMUv3_TH is implemented, then FEAT_PMUv3 is implemented.

The following fields identify the presence of FEAT_PMUv3_TH:

- PMMIR_EL1.THWIDTH.
- PMMIR.THWIDTH.
- PMMIR.THWIDTH.

For more information, see ‘Event threshold and edge counting’.

FEAT_PMUv3p8, Armv8.8 PMU extensions

FEAT_PMUv3p8 introduces the following to the Performance Monitors Extension:

- The Common event number space is extended to include the ranges `0x0040-0x00BF` and `0x4040-0x40BF`.

- For an event counter n, if any reserved or unimplemented PMU event number is written to PMEVTYPER<n>.evtCount, then event counter n does not count, and a read of PMEVTYPER<n>.evtCount returns the value written.

This feature is supported in both AArch64 and AArch32 states.

FEAT_PMUv3p8 is OPTIONAL from Armv8.7.

In an Armv8.8 implementation, if FEAT_PMUv3 is implemented, FEAT_PMUv3p8 is implemented.

If FEAT_PMUv3p8 is implemented, then FEAT_PMUv3p7 is implemented.

The following fields identify the presence of FEAT_PMUv3p8:

- ID_AA64DFR0_EL1.PMUVer.
- ID_DFR0_EL1.PerfMon.
- ID_DFR0.PerfMon.
- EDDFR.PMUVer.

For more information, see PMU Event Descriptions.

FEAT_SCTLR2, Extension to SCTLR_ELx

FEAT_SCTLR2 introduces the SCTLR2_ELx registers, which provide top-level control of the system, including its memory system. These registers are extensions of the corresponding SCTLR_ELx registers.

This feature is supported in AArch64 state only.

FEAT_SCTLR2 is OPTIONAL from Armv8.0.

FEAT_SCTLR2 is mandatory from Armv8.9.

When FEAT_SCTLR2 and FEAT_AA64EL2 are implemented, FEAT_HCX is implemented.

The following field identifies the presence of FEAT_SCTLR2:

- ID_AA64MMFR3_EL1.SCTLRX.

FEAT_SPEv1p3, Statistical Profiling Extensions version 1.3

FEAT_SPEv1p3 introduces the following to the Statistical Profiling Extension:

- Support for sampling Tag operations.
- Support for sampling Memory Copy and Set operations.

This feature is supported in AArch64 state only.

FEAT_SPEv1p3 is OPTIONAL from Armv8.7.

If FEAT_SPEv1p3 is implemented, then FEAT_SPEv1p2 is implemented.

In an Armv8.8 implementation, if FEAT_SPE is implemented, FEAT_SPEv1p3 is implemented.

The following field identifies the presence of FEAT_SPEv1p3:

- ID_AA64DFR0_EL1.PMSVer.

For more information, see:

- 'Additional information for each profiled memory access operation'.
- 'About the Statistical Profiling Extension sample records'.
- 'Address packet'.

FEAT_TCR2, Support for TCR2_ELx

FEAT_TCR2 introduces the TCR2_ELx registers which provide top-level control of the EL1&0 and EL2&0 translation regimes respectively. These registers are extensions of the corresponding TCR_ELx registers.

This feature is supported in AArch64 state only.

FEAT_TCR2 is OPTIONAL from Armv8.0.

FEAT_TCR2 is mandatory from Armv8.9.

When FEAT_TCR2 and FEAT_AA64EL2 are implemented, FEAT_HCX is implemented.

The following field identifies the presence of FEAT_TCR2:

- ID_AA64MMFR3_EL1.TCRX.

FEAT_TIDCP1, EL0 use of IMPLEMENTATION DEFINED functionality

FEAT_TIDCP1 introduces a control at EL1 and EL2 to enable trapping of EL0 accesses to registers that might control IMPLEMENTATION DEFINED functions.

This feature introduces controls only in AArch64 state, and controls IMPLEMENTATION DEFINED execution at EL0 in both AArch32 and AArch64 states.

FEAT_TIDCP1 is OPTIONAL from Armv8.7.

FEAT_TIDCP1 is mandatory from Armv8.8.

The following field identifies the presence of FEAT_TIDCP1:

- ID_AA64MMFR1_EL1.TIDCP1.

For more information, see 'Prioritization of Synchronous exceptions taken to AArch64 state'

7.10 The Armv8.9 architecture extension

The Armv8.9 architecture extension is an extension to Armv8.8. It adds mandatory and optional architectural features. Some features must be implemented together. An implementation is Armv8.9 compliant if all of the following apply:

- It is Armv8.8 compliant.
- It includes all of the Armv8.9 architectural features that are mandatory.

An Armv8.9 compliant implementation can additionally include:

- Armv8.9 features that are optional.

FEAT_ADERR, Asynchronous Device Error Exceptions

FEAT_ADERR introduces controls for whether an error signaled on a load from Device memory is handled precisely and synchronously.

This feature is supported in both AArch64 and AArch32 states.

FEAT_ADERR is OPTIONAL from Armv8.8.

If FEAT_ADERR is implemented, then FEAT_RASv2 is implemented.

If FEAT_ADERR is implemented, then FEAT_SCTLR2 is implemented.

When FEAT_AA64 and FEAT_ADERR are implemented, FEAT_AA64EL1 is implemented.

When FEAT_ADERR and FEAT_AA64EL2 are implemented, FEAT_HCX is implemented.

The following fields identify the presence of FEAT_ADERR:

- ID_AA64MMFR3_EL1.ADERR.
- ID_AA64MMFR3_EL1.SDERR.

For more information, see:

- ‘About the RAS Extension’.
- ‘Taking error exceptions’.

FEAT_AIE, Memory Attribute Index Enhancement

FEAT_AIE increases the stage 1 descriptor attribute index bit width from 3 to 4, allowing use of up to 16 memory attributes.

This feature is supported in AArch64 state only.

FEAT_AIE is OPTIONAL from Armv8.8.

If FEAT_AIE is implemented, then FEAT_TCR2 is implemented.

If FEAT_AIE is implemented, then FEAT_HPDS is implemented.

The following field identifies the presence of FEAT_AIE:

- ID_AA64MMFR3_EL1.AIE.

For more information, see 'Stage 1 memory type and Cacheability attributes'.

FEAT_AMU_EXT64, the 64-bit external Activity Monitors extension

FEAT_AMU_EXT64 indicates the external AMU registers are implemented as 64-bit registers.

FEAT_AMU_EXT64 is OPTIONAL.

If FEAT_AMU_EXT64 is implemented, then FEAT_AMU_EXT is implemented.

If FEAT_AMU_EXT64 is implemented, then FEAT_AMU_EXT32 is not implemented.

The following field identifies the presence of FEAT_AMU_EXT64:

- AMDEVARCH.ARCHID.

For more information, see 'Recommended External Interface to the Activity Monitors'.

FEAT_ANERR, Asynchronous Normal Error Exceptions

FEAT_ANERR introduces controls for whether an error signaled on a load from Normal memory is handled precisely and synchronously.

This feature is supported in both AArch64 and AArch32 states.

FEAT_ANERR is OPTIONAL from Armv8.8.

If FEAT_ANERR is implemented, then FEAT_RASv2 is implemented.

If FEAT_ANERR is implemented, then FEAT_SCTLR2 is implemented.

When FEAT_AA64 and FEAT_ANERR are implemented, FEAT_AA64EL1 is implemented.

When FEAT_ANERR and FEAT_AA64EL2 are implemented, FEAT_HCX is implemented.

The following fields identify the presence of FEAT_ANERR:

- ID_AA64MMFR3_EL1.ANERR.
- ID_AA64MMFR3_EL1.SNERR.

For more information, see:

- 'About the RAS Extension'.
- 'Taking error exceptions'.

FEAT_ATS1A, Address Translation operations that ignore stage 1 permissions

FEAT_ATS1A introduces instructions that provide the output address and attributes of a valid translation without checking for stage 1 permissions.

These instructions are added to the A64 instruction set only.

FEAT_ATS1A is OPTIONAL from Armv8.8.

The following field identifies the presence of FEAT_ATS1A:

- ID_AA64ISAR2_EL1.ATS1A.

FEAT_CLRBHB, Support for Clear Branch History instruction

FEAT_CLRBHB provides a CLRBHB instruction, which clears the branch history for the current context to the extent that branch history information created before the CLRBHB instruction cannot be used by code before the CLRBHB instruction to exploitatively control the execution of any indirect branches in code in the current context that appear in program order after the instruction.

This feature is supported in both AArch64 and AArch32 states.

FEAT_CLRBHB is OPTIONAL from Armv8.0.

FEAT_CLRBHB is mandatory from Armv8.9.

The following fields identify the presence of FEAT_CLRBHB:

- ID_AA64ISAR2_EL1.CLRBHB.
- ID_ISAR6_EL1.CLRBHB.
- ID_ISAR6.CLRBHB.

For more information, see:

- ‘Branch prediction’.
- ‘AArch32 cache and branch predictor maintenance instructions’.

FEAT_CSSC, Common Short Sequence Compression instructions

FEAT_CSSC introduces a set of instructions for optimization of short instruction sequences using general-purpose registers.

This feature is supported in AArch64 state only.

FEAT_CSSC is OPTIONAL from Armv8.7.

FEAT_CSSC is mandatory from Armv8.9.

The following field identifies the presence of FEAT_CSSC:

- ID_AA64ISAR2_EL1.CSSC.

For more information, see:

- ‘Integer minimum and maximum (immediate)’.
- ‘Integer maximum and minimum (register)’.
- ‘Absolute value’.
- ‘Bit operation’.

FEAT_Debugv8p9, Debug v8.9

FEAT_Debugv8p9 introduces all of the following:

- The ability to implement more than 16 breakpoints.
- The ability to implement more than 16 watchpoints.
- DBGBCR<n>_EL1 and DBGWCR<n>_EL1 are 64-bit registers in the external debug interface.
- DSPSR2 is added to extend DSPSR for holding the saved process state for Debug state.

This feature is supported in both AArch64 and AArch32 states.

FEAT_Debugv8p9 is OPTIONAL from Armv8.8.

FEAT_Debugv8p9 is mandatory from Armv8.9.

If FEAT_Debugv8p9 is implemented, then FEAT_Debugv8p8 is implemented.

When FEAT_Debugv8p9 and FEAT_AA64EL2 are implemented, FEAT_FGT2 is implemented.

The following fields identify the presence of FEAT_Debugv8p9:

- ID_AA64DFR0_EL1.DebugVer.
- ID_DFR0_EL1.CopDbg.
- ID_DFR0.CopDbg.
- DBGDIDR.Version.
- EDDEVARCH.ARCHVER.
- EDDEVID1.HSR.

For more information, see:

- ‘About Breakpoint exceptions’.
- ‘About Watchpoint exceptions’.

FEAT_DoubleFault2, Double Fault Extension v2

FEAT_DoubleFault2 provides additional controls for routing and masking error exceptions.

This feature is supported in AArch64 state only.

FEAT_DoubleFault2 is OPTIONAL from Armv8.8.

If FEAT_DoubleFault2 is implemented, then FEAT_SCTLR2 is implemented.

When FEAT_AA64 and FEAT_DoubleFault2 are implemented, FEAT_AA64EL1 is implemented.

When FEAT_DoubleFault2 and FEAT_AA64EL2 are implemented, FEAT_HCX is implemented.

If FEAT_DoubleFault2 is implemented, then FEAT_DoubleFault is implemented.

The following field identifies the presence of FEAT_DoubleFault2:

- ID_AA64PFR1_EL1.DF2.

For more information, see:

- 'Synchronous exception types'.
- 'Asynchronous exception types'.
- 'Error synchronization event'.

FEAT_ECBHB, Exploitative control using branch history information

FEAT_ECBHB imposes restrictions on branch history speculation around exceptions.

This feature is supported in AArch64 state only.

FEAT_ECBHB is OPTIONAL from Armv8.0.

FEAT_ECBHB is mandatory from Armv8.9.

The following field identifies the presence of FEAT_ECBHB:

- ID_AA64MMFR1_EL1.ECBHB.

For more information, see 'Branch prediction'.

FEAT_EDHSR, Support for EDHSR

FEAT_EDHSR introduces the EDHSR, which holds syndrome information of a Debug event.

FEAT_EDHSR is OPTIONAL.

If FEAT_Debugv8p9 is implemented, then FEAT_EDHSR is implemented.

If FEAT_EDHSR is implemented, then FEAT_Debugv8p2 is implemented.

The following field identifies the presence of FEAT_EDHSR:

- EDDEVID1.HSR.

FEAT_FGT2, Fine-grained traps 2

FEAT_FGT2 introduces the hypervisor registers HFGITR2_EL2, HFGRTR2_EL2, HFGWTR_EL2, HDFGRTR2_EL2, and HDFGWTR2_EL2. These registers are extensions of the corresponding FGT registers.

This feature is supported in AArch64, and in AArch32 at EL0 when EL1 is using AArch64.

FEAT_FGT2 is OPTIONAL from Armv8.8.

In an Armv8.9 implementation, if FEAT_AA64EL2 is implemented, FEAT_FGT2 is implemented.

If FEAT_FGT2 is implemented, then FEAT_FGT is implemented.

The following field identifies the presence of FEAT_FGT2:

- ID_AA64MMFR0_EL1.FGT.

FEAT_HAFT, Hardware managed Access Flag for Table descriptors

FEAT_HAFT introduces the support for hardware management of the Table descriptor Access flag.

This feature is supported in AArch64 state only.

FEAT_HAFT is OPTIONAL from Armv8.7.

If FEAT_HAFT is implemented, then FEAT_HAFDBS is implemented.

If FEAT_HAFT is implemented, then FEAT_TCR2 is implemented.

The following field identifies the presence of FEAT_HAFT:

- ID_AA64MMFR1_EL1.HAFDBS.

For more information, see 'Hardware management of the Table descriptor Access Flag'.

FEAT_LRCPC3, Load-Acquire RCpc instructions version 3

FEAT_LRCPC3 introduces variants of load/store pair and load/store single register instructions, with release consistency, to optimize additional use cases where ordering is required.
FEAT_LRCPC3 also introduces a set of additional load/store instructions with release consistency ordering in the Advanced SIMD and floating-point instruction set.

This feature is supported in AArch64 state only.

FEAT_LRCPC3 is OPTIONAL from Armv8.2.

If FEAT_LRCPC3 is implemented, then FEAT_LRCPC2 is implemented.

The following field identifies the presence of FEAT_LRCPC3:

- ID_AA64ISAR1_EL1.LRCPC.

For more information, see:

- 'Changes to single-copy atomicity in Armv8.4'.
- 'Load-Acquire/Store-Release'.
- 'A64 instructions that are changed in Debug state'.

FEAT_MTE4, Enhanced Memory Tagging Extension

FEAT_MTE4 introduces support for the following sub-features:

- Canonical tag checking, identified as FEAT_MTE_CANONICAL_TAGS.
- Reporting of all non-address bits on a fault, identified as FEAT_MTE_TAGGED_FAR.
- Store-only Tag checking, identified as FEAT_MTE_STORE_ONLY.
- Memory tagging with Address tagging disabled, identified as FEAT_MTE_NO_ADDRESS_TAGS.

This feature is supported in AArch64 state only.

FEAT_MTE4 is OPTIONAL from Armv8.7.

In an Armv8.9 implementation, if FEAT_MTE2 is implemented, FEAT_MTE4 is implemented.

If FEAT_MTE4 is implemented, then FEAT_MTE_PERM is implemented.

If FEAT_MTE4 is implemented, then FEAT_MTE_CANONICAL_TAGS, FEAT_MTE_NO_ADDRESS_TAGS, FEAT_MTE_TAGGED_FAR, and FEAT_MTE_STORE_ONLY are implemented.

The following fields identify the presence of FEAT_MTE4:

- ID_AA64PFR1_EL1.MTEX.
- ID_AA64PFR2_EL1.MTESTOREONLY.
- ID_AA64PFR2_EL1.MTEFAR.

FEAT_MTE_ASYNC, Asynchronous reporting of Tag Check Fault

FEAT_MTE_ASYNC provides support for asynchronously accumulating Tag Check Faults into the TFSRE0_EL1 or TFSR_ELx registers. A PE that is compliant with FEAT_MTE2 is compliant with the behavior defined for this feature.

This feature is supported in AArch64 state only.

FEAT_MTE_ASYNC is OPTIONAL from Armv8.5.

If FEAT_MTE_ASYNC is implemented, then FEAT_MTE2 is implemented.

The following field identifies the presence of FEAT_MTE_ASYNC:

- ID_AA64PFR1_EL1.MTE_frac.

For more information, see:

- ‘The Memory Tagging Extension’.
- ‘The AArch64 Application Level Memory Model’.
- PMU Event Descriptions.
- ‘The Statistical Profiling Extension’.

- ‘Debug State’.

FEAT_MTE_CANONICAL_TAGS, Canonical Tag checking for Untagged memory

FEAT_MTE_CANONICAL_TAGS introduces support for MTE canonical tag checking.

This feature is supported in AArch64 state only.

FEAT_MTE_CANONICAL_TAGS is OPTIONAL.

If FEAT_MTE_CANONICAL_TAGS is implemented, then FEAT_MTE4 is implemented.

The following field identifies the presence of FEAT_MTE_CANONICAL_TAGS:

- ID_AA64PFR1_EL1.MTEX.

FEAT_MTE_NO_ADDRESS_TAGS, Memory tagging with Address tagging disabled

FEAT_MTE_NO_ADDRESS_TAG introduces support for MTE tagging with Address tagging disabled.

This feature is supported in AArch64 state only.

FEAT_MTE_NO_ADDRESS_TAGS is OPTIONAL.

If FEAT_MTE_NO_ADDRESS_TAGS is implemented, then FEAT_MTE4 is implemented.

The following field identifies the presence of FEAT_MTE_NO_ADDRESS_TAGS:

- ID_AA64PFR1_EL1.MTEX.

FEAT_MTE_PERM, Allocation tag access permission

FEAT_MTE_PERM introduces support for the Stage 2 NoTagAccess memory attribute.

This feature is supported in AArch64 state only.

FEAT_MTE_PERM is OPTIONAL from Armv8.7.

In an Armv8.9 implementation, if FEAT_MTE2 is implemented, FEAT_MTE_PERM is implemented.

If FEAT_MTE_PERM is implemented, then FEAT_MTE2 is implemented.

The following field identifies the presence of FEAT_MTE_PERM:

- ID_AA64PFR2_EL1.MTEPERM.

FEAT_MTE_STORE_ONLY, Store-only Tag Checking

This feature is supported in AArch64 state only.

FEAT_MTE_STORE_ONLY is OPTIONAL.

If FEAT_MTE_STORE_ONLY is implemented, then FEAT_MTE4 is implemented.

The following field identifies the presence of FEAT_MTE_STORE_ONLY:

- ID_AA64PFR2_EL1.MTESTOREONLY.

FEAT_MTE_TAGGED_FAR, FAR_ELx on a Tag Check Fault

FEAT_MTE_TAGGED_FAR introduces support for reporting all non-address bits on a synchronous MTE tag check fault.

This feature is supported in AArch64 state only.

FEAT_MTE_TAGGED_FAR is OPTIONAL.

If FEAT_MTE_TAGGED_FAR is implemented, then FEAT_MTE4 is implemented.

The following field identifies the presence of FEAT_MTE_TAGGED_FAR:

- ID_AA64PFR2_EL1.MTEFAR.

FEAT_PCSRv8p9, Armv8.9 PC Sample-based Profiling Extension

FEAT_PCSRv8p9 introduces a mechanism to suspend PC Sample-based Profiling.

This feature is supported in both AArch64 and AArch32 states.

FEAT_PCSRv8p9 is OPTIONAL from Armv8.8.

If FEAT_PCSRv8p9 is implemented, then FEAT_PCSRv8p2 is implemented.

The following field identifies the presence of FEAT_PCSRv8p9:

- PMDEVID.PCSample.

For more information, see ‘Suspending and activating PC Sample-based Profiling’.

FEAT_PFAR, Physical Fault Address Register Extension

FEAT_PFAR introduces the Physical Fault Address Registers, PFAR_ELx, that record the faulting physical address for a synchronous External abort or SError exception.

This feature is supported in both AArch64 and AArch32 states.

FEAT_PFAR is OPTIONAL from Armv8.8.

When FEAT_AA64 and FEAT_PFAR are implemented, FEAT_AA64EL1 is implemented.

When FEAT_PFAR and FEAT_AA64EL2 are implemented, FEAT_FGT2 is implemented.

The following field identifies the presence of FEAT_PFAR:

- ID_AA64PFR1_EL1.PFAR.

FEAT_PMUv3_EDGE, PMU event edge detection

FEAT_PMUv3_EDGE introduces edge-detection logic to support counting threshold crossing events.

This feature is supported in both AArch64 and AArch32 states.

FEAT_PMUv3_EDGE is OPTIONAL from Armv8.8.

If FEAT_PMUv3_EDGE is implemented, then FEAT_PMUv3_TH is implemented.

The following fields identify the presence of FEAT_PMUv3_EDGE:

- PMMIR_EL1.EDGE.
- PMMIR.EDGE.
- PMMIR.EDGE.

For more information, see 'Event threshold and edge counting'.

FEAT_PMUv3_ICNTR, Fixed-function instruction counter

FEAT_PMUv3_ICNTR introduces a fixed-function instruction counter to the PMU.

This feature is supported in both AArch64 and AArch32 states. The counter is not accessible from AArch32 state.

FEAT_PMUv3_ICNTR is OPTIONAL from Armv8.8.

If FEAT_PMUv3_ICNTR is implemented, then FEAT_PMUv3p9 is implemented.

When FEAT_PMUv3_ICNTR and FEAT_AA64EL2 are implemented, FEAT_FGT2 is implemented.

The following fields identify the presence of FEAT_PMUv3_ICNTR:

- ID_AA64DFR1_EL1.PMICNTR.
- PMCFGR.NCG.

For more information, see About the Performance Monitors.

FEAT_PMUv3_SS, PMU Snapshot extension

FEAT_PMUv3_SS defines an IMPLEMENTATION DEFINED Snapshot Extension, compatible with the CoreSight PMU Snapshot Extension.

This feature is supported in both AArch64 and AArch32 states. The PMU snapshot registers are not accessible from AArch32 state.

FEAT_PMUv3_SS is OPTIONAL from Armv8.8.

If FEAT_PMUv3_SS is implemented, then FEAT_PMUv3p9 is implemented.

When FEAT_PMUv3_SS and FEAT_AA64EL2 are implemented, FEAT_FGT2 is implemented.

If FEAT_PMUv3_SS is implemented, then FEAT_AA32EL1 is not implemented.

The following fields identify the presence of FEAT_PMUv3_SS:

- ID_AA64DFR0_EL1.PMSS.
- PMDEVID.PMSS.

For more information, see 'PMU snapshots'.

FEAT_PMUv3p9, Armv8.9 PMU extensions

FEAT_PMUv3p9 introduces the following to the Performance Monitors Extension:

- Provides finer-grained control over allocation of PMU event counters to an EL0 process.
- Allows an arbitrary combination of event counters and fixed-function counters to be zeroed.
- Provides controls to configure the PMU to directly request the PE enters Debug state without using the CTI.
- Updates PMU event definitions.

This feature is supported in both AArch64 and AArch32 states.

FEAT_PMUv3p9 is OPTIONAL from Armv8.8.

In an Armv8.9 implementation, if FEAT_PMUv3 is implemented, FEAT_PMUv3p9 is implemented.

If FEAT_PMUv3p9 is implemented, then FEAT_PMUv3p8 is implemented.

When FEAT_PMUv3p9 and FEAT_AA64EL2 are implemented, FEAT_FGT2 is implemented.

The following fields identify the presence of FEAT_PMUv3p9:

- ID_AA64DFR0_EL1.PMUVer.
- ID_DFR0_EL1.PerfMon.
- ID_DFR0.PerfMon.
- EDDFR.PMUVer.

For more information, see:

- PMU Overflow external debug request.
- EL0 access controls.
- PMU Event Descriptions.
- Resetting counters.

FEAT_PRFMSLC, SLC target support for PRFM instructions

FEAT_PRFMSLC introduces a system level cache option for the PRFM instructions.

This feature is supported in AArch64 state only.

FEAT_PRFMSLC is OPTIONAL from Armv8.0.

The following field identifies the presence of FEAT_PRFMSLC:

- ID_AA64ISAR2_EL1.PRFMSLC.

FEAT_RASSAv2, RAS System Architecture Extension v2

FEAT_RASSAv2 implements RAS System Architecture v2, including support for the following:

- System RAS Agents.
- A control for disabling Fault Handling Interrupts for Deferred errors.
- Layouts for 16KB and 64KB error record groups.
- A status flag set on an Error Recovery reset.
- A control for disabling error counting on corrected error Events.
- The OPTIONAL Access Control Register feature.
- Fault Injection Groups.
- Interrupt Control registers for error record groups.
- Additional error record types.

FEAT_RASSAv2 is OPTIONAL from Armv8.8.

In an Armv8.9 implementation, if FEAT_RAS is implemented, FEAT_RASSAv2 is implemented.

For more information, see: *Arm® Reliability Availability and Serviceability (RAS) System Architecture, for A-profile architecture* (ARM IHI 0100).

FEAT_RASv2, RAS Extension v2

FEAT_RASv2 introduces the following to the Reliability, Availability, and Serviceability Extension:

- The features defined by FEAT_RASSAv2 in the System register error records.
- The ERXGSR_EL1 register.
- A Trap exception to EL3 for writes to RAS System registers.
- An additional syndrome to ESR_ELx on an error exception, to give information on whether a location being accessed has been updated.

This feature is supported in both AArch64 and AArch32 states.

FEAT_RASv2 is OPTIONAL from Armv8.8.

In an Armv8.9 implementation, if FEAT_RAS is implemented, FEAT_RASv2 is implemented.

When FEAT_RASv2 and FEAT_AA64EL2 are implemented, FEAT_FGT2 is implemented.

If FEAT_RASv2 is implemented, then FEAT_RASv1p1 is implemented.

If FEAT_RASv2 is implemented, then FEAT_RASSAv2 is implemented.

The following fields identify the presence of FEAT_RASv2:

- ID_AA64PFR0_EL1.RAS.
- ID_PFR0_EL1.RAS.
- ID_PFR0.RAS.

For more information, see:

- 'RAS PE architecture'.
- *Arm® Reliability Availability and Serviceability (RAS) System Architecture, for A-profile architecture* (ARM IHI 0100).

FEAT_RPRFM, Support for Range Prefetch Memory instruction

FEAT_RPRFM introduces the Range Prefetch Memory hint instruction, RPRFM which specifies the range of addresses that are likely to be accessed in the near future and their expected reuse.

This feature is supported in AArch64 state only.

FEAT_RPRFM is OPTIONAL from Armv8.0.

The following field identifies the presence of FEAT_RPRFM:

- ID_AA64ISAR2_EL1.RPRFM.

FEAT_S1PIE, Stage 1 permission indirections

FEAT_S1PIE introduces a way to set stage 1 permissions that allows more efficient use of the permission bits in translation table descriptors and provides the ability to introduce additional permission types.

This feature is supported in AArch64 state only.

FEAT_S1PIE is OPTIONAL from Armv8.8.

If FEAT_S1PIE is implemented, then FEAT_ATS1A is implemented.

If FEAT_S1PIE is implemented, then FEAT_TCR2 is implemented.

The following field identifies the presence of FEAT_S1PIE:

- ID_AA64MMFR3_EL1.S1PIE.

For more information, see 'Stage 1 Indirect permissions'.

FEAT_S1POE, Stage 1 permission overlays

FEAT_S1POE allows stage 1 permissions to be progressively restricted by processes running at EL0 without requiring TLB maintenance, and reduces the number of calls to privileged software.

This feature is supported in AArch64 state only.

FEAT_S1POE is OPTIONAL from Armv8.8.

If FEAT_S1POE is implemented, then FEAT_TCR2 is implemented.

If FEAT_S1POE is implemented, then FEAT_ATS1A is implemented.

If FEAT_S1POE is implemented, then FEAT_HPDS is implemented.

The following field identifies the presence of FEAT_S1POE:

- ID_AA64MMFR3_EL1.S1POE.

For more information, see 'Stage 1 Overlay permissions'.

FEAT_S2PIE, Stage 2 permission indirections

FEAT_S2PIE introduces all of the following:

- A method to set stage 2 permissions that allows more efficient use of the permission bits in translation table descriptors and provides the ability to introduce additional permission types.
- The Mostly Read Only (MRO) permission in stage 2 translations.

This feature is supported in AArch64 state only.

FEAT_S2PIE is OPTIONAL from Armv8.8.

If FEAT_S2PIE is implemented, then FEAT_EL2 is implemented.

The following field identifies the presence of FEAT_S2PIE:

- ID_AA64MMFR3_EL1.S2PIE.

For more information, see 'Stage 2 Indirect permissions'.

FEAT_S2POE, Stage 2 permission overlays

FEAT_S2POE provides a mechanism for stage 2 permissions to be progressively restricted, such that different EL1&0 contexts that are using the same stage 2 translation tables can be provided with different sets of permissions.

This feature is supported in AArch64 state only.

FEAT_S2POE is OPTIONAL from Armv8.8.

If FEAT_S2POE is implemented, then FEAT_EL2 is implemented.

If FEAT_S2POE is implemented, then FEAT_S2PIE is implemented.

The following field identifies the presence of FEAT_S2POE:

- ID_AA64MMFR3_EL1.S2POE.

For more information, see 'Stage 2 Overlay permissions'.

FEAT_SPECRES2, Enhanced speculation restriction instructions

FEAT_SPECRES2 introduces a speculation restriction instruction, Clear Other Speculative Prediction Restriction by Context (COSP), to the instructions that are part of FEAT_SPECRES.

This feature is supported in both AArch64 and AArch32 states.

FEAT_SPECRES2 is OPTIONAL from Armv8.0.

FEAT_SPECRES2 is mandatory from Armv8.9.

The following fields identify the presence of FEAT_SPECRES2:

- ID_AA64ISAR1_EL1.SPECRES.
- ID_ISAR6_EL1.SPECRES.
- ID_ISAR6.SPECRES.

For more information, see:

- 'Prediction restriction instructions'.
- 'Execution, data prediction and prefetching restriction System instructions'.
- 'Execution and data prediction restriction System instructions'.

FEAT_SPE_CRR, Statistical Profiling call return branch records

FEAT_SPE_CRR extends the 'Operation Type packet' to provide more information whether the branch is a procedure call or a procedure return.

This feature is supported in AArch64 state only.

FEAT_SPE_CRR is OPTIONAL.

In an Armv8.9 implementation, if FEAT_SPE is implemented, FEAT_SPE_CRR is implemented.

If FEAT_SPE_CRR is implemented, then FEAT_SPEv1p4 is implemented.

If FEAT_SPE and FEAT_GCS are implemented, then FEAT_SPE_CRR is implemented.

The following field identifies the presence of FEAT_SPE_CRR:

- PMSIDR_EL1.CRR.

FEAT_SPE_DPFZS, Disable Cycle Counter on SPE Freeze

FEAT_SPE_DPFZS introduces controls to disable cycle counting when event counting is frozen on a Statistical Profiling Buffer Management event.

This feature is supported in AArch64 state only.

FEAT_SPE_DPFZS is OPTIONAL from Armv8.6.

If FEAT_PMUv3p9 and FEAT_SPEv1p4 are implemented, then FEAT_SPE_DPFZS is implemented.

If FEAT_SPE_DPFZS is implemented, then FEAT_PMUv3p7 and FEAT_SPEv1p2 are implemented.

The following field identifies the presence of FEAT_SPE_DPFZS:

- ID_AA64DFR1_EL1.DPFZS.

FEAT_SPE_FDS, Statistical Profiling data source filtering

FEAT_SPE_FDS provides the capability to filter sample records by all or part of the sampled 'Data Source packet'.

This feature is supported in AArch64 state only.

FEAT_SPE_FDS is OPTIONAL from Armv8.8.

In an Armv8.9 implementation, if FEAT_SPE is implemented, FEAT_SPE_FDS is implemented.

If FEAT_SPE_FDS is implemented, then FEAT_SPEv1p4 is implemented.

When FEAT_SPE_FDS and FEAT_AA64EL2 are implemented, FEAT_FGT2 is implemented.

The following field identifies the presence of FEAT_SPE_FDS:

- PMSIDR_EL1.FDS.

For more information, see 'Filtering sample records'.

FEAT_SPEv1p4, Statistical Profiling Extension version 1.4

FEAT_SPEv1p4 introduces the following to the Statistical Profiling Extension:

- Additions to the 'Events packet' to provide more information about the data source.
- Additional event-based record filtering control bits in PMSEVFR_EL1 and PMSNEVFR_EL1 that add the ability to filter by additional existing bits in the Events packet.

This feature is supported in AArch64 state only.

FEAT_SPEv1p4 is OPTIONAL from Armv8.8.

In an Armv8.9 implementation, if FEAT_SPE is implemented, FEAT_SPEv1p4 is implemented.

If FEAT_SPEv1p4 is implemented, then FEAT_SPEv1p3 is implemented.

The following field identifies the presence of FEAT_SPEv1p4:

- ID_AA64DFR0_EL1.PMSVer.

FEAT_SPMU, System Performance Monitors Extension

FEAT_SPMU provides a framework of architectural System registers and behaviors for System PMUs, which are PMUs other than the Performance Monitors Extension PMU, that are accessible by the PE.

This feature is supported in both AArch64 and AArch32 states.

FEAT_SPMU is OPTIONAL from Armv8.8.

If FEAT_SPMU is implemented, then FEAT_PMUv3p9 is implemented.

When FEAT_SPMU and FEAT_AA64EL2 are implemented, FEAT_FGT2 is implemented.

The following field identifies the presence of FEAT_SPMU:

- ID_AA64DFR1_EL1.SPMU.

For more information, see 'System Performance Monitors Extension'.

FEAT_THE, Translation Hardening Extension

FEAT_THE provides a mechanism to prevent modification of an arbitrary subset of translation table entries from within the exception level that owns the translation tables, and introduces the following:

- The stage 1 Assured Translation property and the stage 2 AssuredOnly property.
- Stage 2 TopLevel checks.
- Read-Check-Write instructions, RCW* and RCWS*.
- Read-Check-Write mask registers, RCWMASK_EL1 and RCWSMASK_EL1.
- The Protected attribute in stage 1 descriptors.

This feature is supported in AArch64 state only.

FEAT_THE is OPTIONAL from Armv8.8.

When FEAT_THE and FEAT_AA64EL2 are implemented, FEAT_S2PIE is implemented.

When FEAT_THE and FEAT_AA64EL2 are implemented, FEAT_FGT2 is implemented.

If FEAT_THE is implemented, then FEAT_TCR2 is implemented.

The following field identifies the presence of FEAT_THE:

- ID_AA64PFR1_EL1.THE.

For more information, see:

- 'Translation Hardening Extension'.
- 'Read-Check-Write'.

7.11 The Armv9.0 architecture extension

The Armv9.0 architecture extension adds mandatory and optional architectural features. Some features must be implemented together. An implementation is Armv9.0 compliant if all of the following apply:

- It is Armv8.5 compliant.
- It includes all of the Armv9.0 architectural features that are mandatory.

An Armv9.0 compliant implementation can additionally include:

- Armv9.0 features that are optional.
- Any arbitrary subset of the architectural features of Armv9.1, subject only to those constraints that require that certain features be implemented together.

In an Armv9.0.implementation, the AArch32 state might optionally be implemented at EL0. The AArch32 state is not implemented at EL1, EL2, and EL3.

An implementation of Armv9.0. cannot implement FEAT_DoubleLock.

An implementation of the Armv9.0. architecture cannot include an ETM.

In an Armv9.0. implementation, if FEAT_FP and FEAT_AdvSIMD are implemented, the following features are implemented:

- FEAT_RDM.
- FEAT_FP16.
- FEAT_DotProd.
- FEAT_FHM.
- FEAT_FCMA.
- FEAT_JSCVT.
- FEAT_FlagM2.
- FEAT_FRINTTS.

FEAT_Armv9_Crypto, Armv9 Cryptographic Extension

The Armv9 Cryptographic Extension provides instructions for the acceleration of encryption and decryption. The presence of the Cryptographic Extension in an implementation is subject to export license controls.

This feature is supported in AArch64 state only.

FEAT_Armv9_Crypto is OPTIONAL.

If FEAT_Armv9_Crypto is implemented, then FEAT_Crypto is implemented.

If FEAT_Armv9_Crypto is implemented, then FEAT_PMULL, FEAT_AES, FEAT_SHA1, FEAT_SHA3, FEAT_SHA256, and FEAT_SHA512 are implemented.

When FEAT_Armv9_Crypto, FEAT_SVE and FEAT_SVE_SHA3 are implemented, FEAT_SVE_AES, and FEAT_SVE_PMULL128 are implemented.

When FEAT_Armv9_Crypto, FEAT_SVE and FEAT_SVE_AES are implemented, FEAT_SVE_PMULL128, and FEAT_SVE_SHA3 are implemented.

When FEAT_Armv9_Crypto, FEAT_SVE and FEAT_SVE_PMULL128 are implemented, FEAT_SVE_AES, and FEAT_SVE_SHA3 are implemented.

For more information, see:

- The Cryptographic Extension in AArch64 state.
- The Cryptographic Extension in AArch32 state.

FEAT_ETE, Embedded Trace Extension

FEAT_ETE provides a trace unit that records details about software control flow running on a PE, which can be used to aid debugging or optimizing. The trace unit provides filtering functionality to allow the targeting of the information to specific code regions or periods of operation.

This feature is supported in AArch64 state, and performs trace in both AArch64 and AArch32 states.

FEAT_ETE is OPTIONAL from Armv9.0.

If FEAT_ETE is implemented, then FEAT_TRBE is implemented.

If FEAT_ETE is implemented, then FEAT_TRF is implemented.

If FEAT_ETE is implemented, then FEAT_TRC_SR is implemented.

If FEAT_ETE is implemented, then FEAT_ETMv4 is not implemented.

When FEAT_ETE and FEAT_PMUv3 are implemented, FEAT_PMUv3p1 is implemented.

For more information, see 'The Embedded Trace Extension'.

FEAT_SVE2, Scalable Vector Extension version 2

The *Scalable Vector Extension version 2* (SVE2) is a superset of SVE that incorporates functionality similar to Advanced SIMD, and other enhancements. In this Manual, unless stated otherwise, when SVE is used, the behavior also applies to SVE2. All Armv9-A systems that support standard operating systems with rich application environments also provide hardware support for SVE2 instructions.

This feature is supported in AArch64 state only.

FEAT_SVE2 is OPTIONAL from Armv9.0.

If FEAT_SVE2 is implemented, then FEAT_SVE is implemented.

The following field identifies the presence of FEAT_SVE2:

- ID_AA64ZFR0_EL1.SVEver.

For more information, see:

- FEAT_SVE.
- 'Data processing - SVE2'.

FEAT_SVE_AES, Scalable Vector AES instructions

FEAT_SVE_AES provides the following scalable vector AES cryptographic instructions:

- AESD.
- AESE.
- AESIMC.
- AESMC.
- PMULLB.
- PMULLT.

This feature is supported in AArch64 state only.

FEAT_SVE_AES is OPTIONAL from Armv9.0.

If FEAT_SVE_AES is implemented, then FEAT_SVE2 or FEAT_SSVE_AES is implemented.

If FEAT_SVE_AES is implemented, then FEAT_SVE_PMULL128 is implemented.

If FEAT_SVE_PMULL128 is implemented, then FEAT_SVE_AES is implemented.

If FEAT_SVE_AES is implemented, then FEAT_Armv9_Crypto is implemented.

The following field identifies the presence of FEAT_SVE_AES:

- ID_AA64ZFR0_EL1.AES.

FEAT_SVE_BitPerm, Scalable Vector Bit Permutes instructions

FEAT_SVE_BitPerm provides the following scalable vector bit permute instructions:

- BEXT.
- BDEP.
- BGRP.

This feature is supported in AArch64 state only.

FEAT_SVE_BitPerm is OPTIONAL from Armv9.0.

If FEAT_SVE_BitPerm is implemented, then FEAT_SVE2 or FEAT_SSVE_BitPerm is implemented.

The following field identifies the presence of FEAT_SVE_BitPerm:

- ID_AA64ZFR0_EL1.BitPerm.

FEAT_SVE_PMULL128, SVE single-vector Advanced Encryption Standard and 128-bit polynomial multiply long instructions

FEAT_SVE_PMULL128 implements the SVE single-vector Advanced Encryption Standard and 128-bit destination element variants of PMULLB and PMULLT instructions, when the PE is not in Streaming SVE mode.

This feature is supported in AArch64 state only.

FEAT_SVE_PMULL128 is OPTIONAL from Armv9.0.

If FEAT_SVE_PMULL128 is implemented, then FEAT_SVE2 or FEAT_SSVE_AES is implemented.

If FEAT_SVE_PMULL128 is implemented, then FEAT_SVE_AES is implemented.

The following field identifies the presence of FEAT_SVE_PMULL128:

- ID_AA64ZFR0_EL1.AES.

FEAT_SVE_SHA3, Scalable Vector SHA3 instructions

FEAT_SVE_SHA3 provides the following scalable vector SHA3 instruction:

- RAX1.

This feature is supported in AArch64 state only.

FEAT_SVE_SHA3 is OPTIONAL from Armv9.0.

If FEAT_SVE_SHA3 is implemented, then FEAT_SVE2 or FEAT_SME2p1 is implemented.

If FEAT_SVE_SHA3 is implemented, then FEAT_Armv9_Crypto is implemented.

The following field identifies the presence of FEAT_SVE_SHA3:

- ID_AA64ZFR0_EL1.SHA3.

FEAT_SVE_SM4, Scalable Vector SM4 instructions

FEAT_SVE_SM4 provides the following scalable vector SM4 instructions:

- SM4E.
- SM4EKEY.

This feature is supported in AArch64 state only.

FEAT_SVE_SM4 is OPTIONAL from Armv9.0.

If FEAT_SVE_SM4 is implemented, then FEAT_SVE2 is implemented.

If FEAT_SVE_SM4 is implemented, then FEAT_SM4 is implemented.

The following field identifies the presence of FEAT_SVE_SM4:

- ID_AA64ZFR0_EL1.SM4.

FEAT_TME, Transactional Memory Extension

FEAT_TME introduces a set of instructions to support hardware transaction memory, which means a group of instructions can appear to be collectively executed as a single atomic operation. For more information on these instructions, see:

- TCANCEL.
- TCOMMIT.
- TSTART.
- TTEST.

This feature is supported in AArch64 state only.

FEAT_TME is OPTIONAL from Armv9.0.

The following field identifies the presence of FEAT_TME:

- ID_AA64ISAR0_EL1.TME.

For more information on FEAT_TME, see *Arm® Architecture Reference Manual Supplement, Transactional Memory Extension (TME), for A-profile architecture(ARM DDI 0617)*.

FEAT_TRBE, Trace Buffer Extension

FEAT_TRBE enables support for a Trace Buffer Unit within a PE. When the Trace Buffer Unit is enabled, program-flow trace generated by a trace unit is written directly to memory by the Trace Buffer Unit, rather than routing trace data to a trace sink.

This feature is supported in AArch64 state, and collects trace in both AArch64 and AArch32 states.

FEAT_TRBE is OPTIONAL from Armv9.0.

If FEAT_TRBE is implemented, then FEAT_TRF is implemented.

When FEAT_TRBE and FEAT_PMUv3 are implemented, FEAT_PMUv3p1 is implemented.

The following field identifies the presence of FEAT_TRBE:

- ID_AA64DFR0_EL1.TraceBuffer.

For more information, see 'The Trace Buffer Extension'.

Features added to the Armv9.0 extension in later releases

- FEAT_IDTE3.
- FEAT_PCDPHINT.
- FEAT_UINJ.

7.12 The Armv9.1 architecture extension

The Armv9.1 architecture extension is an extension to Armv9.0. It adds mandatory and optional architectural features. Some features must be implemented together. An implementation is Armv9.1 compliant if all of the following apply:

- It is Armv8.6 compliant.
- It is Armv9.0 compliant.
- It includes all of the Armv9.1 architectural features that are mandatory.

An Armv9.1 compliant implementation can additionally include:

- Armv9.1 features that are optional.
- Any arbitrary subset of the architectural features of Armv9.2, subject only to those constraints that require that certain features be implemented together.

FEAT_ETEv1p1, Embedded Trace Extension

FEAT_ETEv1p1 extends FEAT_ETE to provide more flexibility for tracing Timestamp values.

This feature is supported in AArch64 state, and performs trace in both AArch64 and AArch32 states.

FEAT_ETEv1p1 is OPTIONAL from Armv9.0.

If FEAT_ETEv1p1 is implemented, then FEAT_ETE is implemented.

The following fields identify the presence of FEAT_ETEv1p1:

- TRCDEVARCH.REVISION.
- TRCDEVARCH.REVISION.

For more information, see ‘The Embedded Trace Extension.’.

7.13 The Armv9.2 architecture extension

The Armv9.2 architecture extension is an extension to Armv9.1. It adds mandatory and optional architectural features. Some features must be implemented together. An implementation is Armv9.2 compliant if all of the following apply:

- It is Armv8.7 compliant.
- It is Armv9.1 compliant.
- It includes all of the Armv9.2 architectural features that are mandatory.

An Armv9.2 compliant implementation can additionally include:

- Armv9.2 features that are optional.
- Any arbitrary subset of the architectural features of Armv9.3, subject only to those constraints that require that certain features be implemented together.

FEAT_BRBE, Branch Record Buffer Extension

FEAT_BRBE provides a Branch record buffer for capturing control path history.

This feature is supported in AArch64 state only.

FEAT_BRBE is OPTIONAL from Armv9.1.

The following field identifies the presence of FEAT_BRBE:

- ID_AA64DFR0_EL1.BRBE.

For more information, see 'The Branch Record Buffer Extension'.

FEAT_ETEv1p2, Embedded Trace Extension

FEAT_ETEv1p2 extends FEAT_ETE to support FEAT_RME.

This feature is supported in AArch64 state, and performs trace in both AArch64 and AArch32 states.

FEAT_ETEv1p2 is OPTIONAL from Armv9.1.

If FEAT_ETEv1p2 is implemented, then FEAT_ETEv1p1 is implemented.

If FEAT_ETE and FEAT_RME are implemented, then FEAT_ETEv1p2 is implemented.

The following fields identify the presence of FEAT_ETEv1p2:

- TRCDEVARCH.REVISION.
- TRCDEVARCH.REVISION.

For more information, see 'The Embedded Trace Extension.'.

FEAT_RME, Realm Management Extension

The Realm Management Extension (RME) is an extension to the Armv9 A-profile architecture.

RME introduces all of the following features:

- Two additional Security states, Root and Realm.
- Two additional physical address spaces, Root and Realm.
- The ability to dynamically transition memory granules between physical address spaces.
- Granule Protection Check mechanism.

FEAT_RME is one component of the Arm Confidential Compute Architecture (Arm CCA). Together with the other components of the Arm CCA, RME enables support for dynamic, attestable, and trusted execution environments (Realms) to be run on an Arm PE.

This feature is supported in AArch64 state only.

FEAT_RME is OPTIONAL from Armv9.1.

If FEAT_RME is implemented, then FEAT_AA64EL3, FEAT_AA64EL2, and FEAT_RNG or FEAT_RNG_TRAP are implemented.

When FEAT_RME and FEAT_AES or FEAT_SHA1 are implemented, FEAT_PMULL, FEAT_AES, FEAT_SHA3, FEAT_SHA256, and FEAT_SHA512 are implemented.

When FEAT_RME, FEAT_SVE and FEAT_AES or FEAT_SHA1 are implemented, FEAT_SVE_PMULL128, and FEAT_SVE_SHA3 are implemented.

When FEAT_ETE and FEAT_RME are implemented, FEAT_ETEv1p2 is implemented.

When FEAT_PMUv3 and FEAT_RME are implemented, FEAT_PMUv3p7 is implemented.

When FEAT_SPE and FEAT_RME are implemented, FEAT_SPEv1p2 is implemented.

When FEAT_MPAM and FEAT_RME are implemented, FEAT_MPAMv1p1 is implemented.

If FEAT_RME is implemented, then FEAT_PCSRv8 is not implemented.

The following field identifies the presence of FEAT_RME:

- ID_AA64PFR0_EL1.RME.

If the Activity Monitors Extension is implemented, then Arm strongly recommends that a PE that implements FEAT_RME also implements FEAT_AMUv1p1.

Arm recommends that a PE that implements FEAT_RME also implements all of the following:

- FEAT_VMID16.
- FEAT_HAFDBS.

For more information, see:

- 'The AArch64 System Level Programmers´ Model'.
- 'The AArch64 Virtual Memory System Architecture'.
- 'The Granule Protection Check Mechanism'.

FEAT_SME, Scalable Matrix Extension

FEAT_SME introduces two AArch64 execution modes that can be enabled and disabled by application software:

- In ZA storage enabled mode, scalable, two-dimensional, architectural ZA tile storage becomes available and instructions are defined to load, store, extract, insert, and clear rows and columns of the ZA tiles.
- In Streaming SVE mode, the Effective SVE vector length changes to match the Effective ZA tile width, support for a substantial subset of the SVE2 instruction set is available, and, when ZA mode is also enabled, instructions are defined that accumulate the matrix outer product of two SVE vectors into a ZA tile.

This feature is supported in AArch64 state only.

FEAT_SME is OPTIONAL from Armv9.2.

If FEAT_SME is implemented, then FEAT_FCMA, FEAT_FP16, FEAT_BF16, and FEAT_FHM are implemented.

When FEAT_SME and FEAT_EL2 are implemented, FEAT_FGT and FEAT_HCX are implemented.

When FEAT_SME and FEAT_PMUv3 are implemented, FEAT_PMUv3p1 is implemented.

The following field identifies the presence of FEAT_SME:

- ID_AA64PFR1_EL1.SME.

If FEAT_SME is implemented, this does not imply that FEAT_SVE and FEAT_SVE2 are implemented when the PE is not in Streaming SVE mode.

For more information, see 'The Scalable Matrix Extension'.

FEAT_SME_F64F64, Double-precision floating-point outer product instructions

FEAT_SME_F64F64 indicates SME support for instructions that accumulate into double-precision floating-point elements in the ZA array.

This feature is supported in AArch64 state only.

FEAT_SME_F64F64 is OPTIONAL from Armv9.2.

If FEAT_SME_F64F64 is implemented, then FEAT_SME is implemented.

The following field identifies the presence of FEAT_SME_F64F64:

- ID_AA64SMFR0_EL1.F64F64.

For more information, see 'The Scalable Matrix Extension'.

FEAT_SME_FA64, Full A64 instruction set support in Streaming SVE mode

FEAT_SME_FA64 indicates support for execution of the full A64 instruction set in Streaming SVE mode.

This feature is supported in AArch64 state only.

FEAT_SME_FA64 is OPTIONAL from Armv9.2.

If FEAT_SME_FA64 is implemented, then FEAT_SME is implemented.

If FEAT_SME_FA64 is implemented, then FEAT_SVE2 is implemented.

The following field identifies the presence of FEAT_SME_FA64:

- ID_AA64SMFR0_EL1.FA64.

For more information, see 'The Scalable Matrix Extension'.

FEAT_SME_I16I64, 16-bit to 64-bit integer widening outer product instructions

FEAT_SME_I16I64 indicates SME support for instructions that accumulate into 64-bit integer elements in the ZA array.

This feature is supported in AArch64 state only.

FEAT_SME_I16I64 is OPTIONAL from Armv9.2.

If FEAT_SME_I16I64 is implemented, then FEAT_SME is implemented.

The following field identifies the presence of FEAT_SME_I16I64:

- ID_AA64SMFR0_EL1.I16I64.

For more information, see 'The Scalable Matrix Extension'.

Features added to the Armv9.2 extension in later releases

- FEAT_F8F16MM.
- FEAT_F8F32MM.
- FEAT_FAMINMAX.
- FEAT_FP8DOT2.
- FEAT_FP8DOT4.
- FEAT_FP8FMA.
- FEAT_FP8.
- FEAT_FPMR.

- FEAT_LS64WB.
- FEAT_LUT.
- FEAT_SME2p1.
- FEAT_SME_B16B16.
- FEAT_SME_F16F16.
- FEAT_SME_F8F16.
- FEAT_SME_F8F32.
- FEAT_SME_LUTv2.
- FEAT_SPE_SME.
- FEAT_SSVE_FP8DOT2.
- FEAT_SSVE_FP8DOT4.
- FEAT_SSVE_FP8FMA.
- FEAT_SVE2p1.
- FEAT_SVE_B16B16.
- FEAT_SVE_BFSCALE.
- FEAT_SVE_F16F32MM.

7.14 The Armv9.3 architecture extension

The Armv9.3 architecture extension is an extension to Armv9.2. It adds mandatory and optional architectural features. Some features must be implemented together. An implementation is Armv9.3 compliant if all of the following apply:

- It is Armv8.8 compliant.
- It is Armv9.2 compliant.
- It includes all of the Armv9.3 architectural features that are mandatory.

An Armv9.3 compliant implementation can additionally include:

- Armv9.3 features that are optional.
- Any arbitrary subset of the architectural features of Armv9.4, subject only to those constraints that require that certain features be implemented together.

FEAT_BRBEv1p1, Branch Record Buffer Extension version 1.1

FEAT_BRBEv1p1 extends FEAT_BRBE to enable branch recording at EL3.

This feature is supported in AArch64 state only.

FEAT_BRBEv1p1 is OPTIONAL from Armv9.2.

In an Armv9.3 implementation, if FEAT_BRBE is implemented, FEAT_BRBEv1p1 is implemented.

If FEAT_BRBEv1p1 is implemented, then FEAT_BRBE is implemented.

The following field identifies the presence of FEAT_BRBEv1p1:

- ID_AA64DFR0_EL1.BRBE.

For more information, see 'The Branch Record Buffer Extension'.

FEAT_MEC, Memory Encryption Contexts

Memory Encryption Contexts (MEC) is an extension to the RME.

An existing RME enabled system uses a combination of isolation and external memory protection to guarantee privacy of the Realm Security state. Isolation between Realms is enforced by the Realm stage 2 translation tables.

FEAT_MEC introduces all of the following features:

- Memory encryption contexts are provided to all physical address spaces.
- Multiple memory encryption contexts are provided to the Realm physical address space for assignment to Realm virtual machines, with policy controlled by Realm EL2.
- The Non-secure, Secure, and Root, physical address spaces each have one encryption context.

This feature is supported in AArch64 state only.

FEAT_MEC is OPTIONAL from Armv9.2.

If FEAT_MEC is implemented, then FEAT_RME is implemented.

If FEAT_MEC is implemented, then FEAT_SCTLR2 is implemented.

If FEAT_MEC is implemented, then FEAT_TCR2 is implemented.

The following field identifies the presence of FEAT_MEC:

- ID_AA64MMFR3_EL1.MEC.

FEAT_SME2, Scalable Matrix Extensions version 2

FEAT_SME2 is a superset of FEAT_SME that introduces the following:

- The ability to treat the SME ZA array as containing addressable groups of one-dimensional ZA array vectors, instead of two-dimensional ZA tiles.
- Multi-vector instructions that operate on groups of Z vector registers and ZA array vectors.
- A multi-vector predication mechanism for multi-vector load and store.
- A dedicated 512-bit lookup table register, ZT0, for data decompression.

This feature is supported in AArch64 state only.

FEAT_SME2 is OPTIONAL from Armv9.2.

If FEAT_SME2 is implemented, then FEAT_SME is implemented.

The following fields identify the presence of FEAT_SME2:

- ID_AA64SMFR0_EL1.SMEver.
- ID_AA64PFR1_EL1.SME.

For more information, see 'The Scalable Matrix Extension'.

Features added to the Armv9.3 extension in later releases

- FEAT_LSFE.
- FEAT_MPAM_PE_BW_CTRL.

7.15 The Armv9.4 architecture extension

The Armv9.4 architecture extension is an extension to Armv9.3. It adds mandatory and optional architectural features. Some features must be implemented together. An implementation is Armv9.4 compliant if all of the following apply:

- It is Armv8.9 compliant.
- It is Armv9.3 compliant.
- It includes all of the Armv9.4 architectural features that are mandatory.

An Armv9.4 compliant implementation can additionally include:

- Armv9.4 features that are optional.

FEAT_ABLE, Address Breakpoint Linking Extension

FEAT_ABLE introduces the capability to link a watchpoint to an address matching breakpoint.

This feature is supported in both AArch64 and AArch32 states.

FEAT_ABLE is OPTIONAL from Armv9.3.

If FEAT_ABLE is implemented, then FEAT_BWE is implemented.

If FEAT_ABLE is implemented, then FEAT_Debugv8p9 is implemented.

The following fields identify the presence of FEAT_ABLE:

- ID_AA64DFR1_EL1.ABLE.
- EDDFR1.ABLE.

For more information, see 'About Breakpoint exceptions'.

FEAT_BWE, Breakpoint and watchpoint enhancements

FEAT_BWE introduces the capability to define an included-range-based breakpoint and an excluded-range-based breakpoint.

This feature is supported in AArch64 state only.

FEAT_BWE is OPTIONAL from Armv9.3.

The following fields identify the presence of FEAT_BWE:

- ID_AA64DFR1_EL1.ABLE.
- EDDFR1.ABLE.
- EDDFR2.BWE.
- ID_AA64DFR2_EL1.BWE.

For more information, see 'Other usage constraints for Address breakpoints'.

FEAT_CHK, Check Feature Status

FEAT_CHK introduces the CHKFEAT instruction, which allows software to detect when certain features are enabled.

A PE that is compliant with architectures from Armv8.0 to Armv9.3 is compliant with the behavior defined for this feature.

This feature is supported in AArch64 state only.

FEAT_CHK is OPTIONAL from Armv8.0.

FEAT_CHK is mandatory from Armv9.4.

For more information, see 'Detecting when FEAT_GCS is enabled'.

FEAT_D128, 128-bit Translation Tables, 56 bit PA

FEAT_D128 introduces support for the VMSAv9-128 translation system, comprising the following:

- 128-bit translation table descriptors.
- Support for encoding up to 56-bit physical addresses in translation table descriptors.
- If FEAT_LVA or FEAT_LVA3 are implemented, support for translating up to 56-bit virtual addresses.
- TLBIP VA, *TLBIP RVA*, TLBIP IPA, *TLBIP RIPA* instructions that can take 128-bit inputs.

This feature is supported in AArch64 state only.

FEAT_D128 is OPTIONAL from Armv9.3.

If FEAT_D128 is implemented, then FEAT_SYSREG128 is implemented.

If FEAT_D128 is implemented, then FEAT_SYSINSTR128 is implemented.

If FEAT_D128 is implemented, then FEAT_LSE128 is implemented.

If FEAT_D128 is implemented, then FEAT_S1PIE is implemented.

When FEAT_D128 and FEAT_EL2 are implemented, FEAT_S2PIE is implemented.

If FEAT_D128 is implemented, then FEAT_AIE is implemented.

If FEAT_D128 is implemented, then FEAT_TCR2 is implemented.

The following field identifies the presence of FEAT_D128:

- ID_AA64MMFR3_EL1.D128.

For more information, see 'The AArch64 Virtual Memory System Architecture'.

FEAT_EBEP, Exception-based Event Profiling

FEAT_EBEP provides support for reporting PMU counter overflows as PMU Profiling exceptions. This allows generation of higher quality profiles by eliminating the interrupt latency and jitter incurred outside the PE.

This feature is supported in both AArch64 and AArch32 states.

FEAT_EBEP is OPTIONAL from Armv9.3.

In an Armv9.3 implementation, if FEAT_PMUv3p9 is implemented, FEAT_EBEP is implemented.

When FEAT_EBEP and FEAT_AA64EL2 are implemented, FEAT_FGT2 is implemented.

When FEAT_EBEP and FEAT_AA32EL0 are implemented, FEAT_Debugv8p9 is implemented.

The following field identifies the presence of FEAT_EBEP:

- ID_AA64DFR1_EL1.EBEP.

For more information, see 'Exception-based event profiling'.

FEAT_ETEv1p3, Embedded Trace Extension version 1.3

FEAT_ETEv1p3 extends FEAT_ETE to support all of the following:

- The ETE External Debug Request, when FEAT_Debugv8p9 is implemented.
- The ETE Trace Output Enable, which is mandatory for FEAT_ETEv1p3 and OPTIONAL for FEAT_ETE.

This feature is supported in AArch64 state, and performs trace in both AArch64 and AArch32 states.

FEAT_ETEv1p3 is OPTIONAL from Armv9.3.

If FEAT_ETEv1p3 is implemented, then FEAT_ETEv1p2 is implemented.

The following fields identify the presence of FEAT_ETEv1p3:

- TRCDEVARCH.REVISION.
- TRCDEVARCH.REVISION.

For more information, see 'The Embedded Trace Extension'.

FEAT_GCS, Guarded Control Stack Extension

FEAT_GCS introduces support for a Guarded Control Stack, an area of memory in which procedure return addresses and exception return addresses are stored and protected from modification.

This feature is supported in AArch64 state only.

FEAT_GCS is OPTIONAL from Armv9.3.

If FEAT_GCS is implemented, then FEAT_CHK is implemented.

If FEAT_GCS is implemented, then FEAT_S1PIE is implemented.

The following field identifies the presence of FEAT_GCS:

- ID_AA64PFR1_EL1.GCS.

For more information, see 'Guarded Control Stack'.

FEAT_ITE, Instrumentation Trace Extension

FEAT_ITE provides all of the following to allow software to inject instrumentation information into the ETE trace stream:

- The TRCIT instruction, that injects the value of a general purpose register into the ETE trace stream.
- 'Instrumentation Packet' that contains the value written by the TRCIT instruction.
- Controls that define the behavior of the TRCIT instruction.

This feature is supported in both AArch64 and AArch32 states.

FEAT_ITE is OPTIONAL from Armv9.3.

If FEAT_ITE is implemented, then FEAT_TRF is implemented.

If FEAT_ITE is implemented, then FEAT_ETE is implemented.

If FEAT_ITE is implemented, then FEAT_TRBE is implemented.

When FEAT_ITE and FEAT_AA64EL2 are implemented, FEAT_FGT2 is implemented.

The following fields identify the presence of FEAT_ITE:

- ID_AA64DFR1_EL1.ITE.
- TRCIDR0.ITE.
- TRCIDR0.ITE.

For more information, see:

- ‘Instrumentation extension’.
- ‘Instrumentation element’.

FEAT_LSE128, 128-bit Atomics

FEAT_LSE128 introduces support for 128-bit atomic instructions.

This feature is supported in AArch64 state only.

FEAT_LSE128 is OPTIONAL from Armv9.3.

If FEAT_LSE128 is implemented, then FEAT_LSE is implemented.

The following field identifies the presence of FEAT_LSE128:

- ID_AA64ISAR0_EL1.Atomic.

For more information, see:

- Atomic memory operations.
- Swap.

FEAT_LVA3, 56-bit VA

FEAT_LVA3 introduces support for 56-bit virtual addresses.

This feature is supported in AArch64 state only.

FEAT_LVA3 is OPTIONAL from Armv9.3.

If FEAT_LVA3 is implemented, then FEAT_D128 is implemented.

If FEAT_LVA3 is implemented, then FEAT_LVA is implemented.

The following field identifies the presence of FEAT_LVA3:

- ID_AA64MMFR2_EL1.VARange.

For more information, see ‘Supported virtual address ranges’.

FEAT_SEBEP, Synchronous Exception-based Event Profiling

FEAT_SEBEP creates configurations to generate synchronous and precise PMU Profiling exceptions.

This feature is supported in both AArch64 and AArch32 states.

FEAT_SEBEP is OPTIONAL from Armv9.3.

If FEAT_SEBEP is implemented, then FEAT_EBEP is implemented.

When FEAT_SEBEP and FEAT_AA64EL2 are implemented, FEAT_FGT2 is implemented.

When FEAT_SEBEP and FEAT_AA32EL0 are implemented, FEAT_Debugv8p9 is implemented.

The following field identifies the presence of FEAT_SEBEP:

- ID_AA64DFR0_EL1.SEBEP.

For more information, see:

- 'Synchronous exception-based event profiling'.
- 'Synchronous events'.

FEAT_SME2p1, Scalable Matrix Extension version 2.1

The Scalable Matrix Extension version 2.1 (SME2.1) is a superset of SME2 that adds:

- Support for SVE RAX1 instructions in Streaming SVE mode if FEAT_SVE_SHA3 is implemented.
- SME MOVAZ instructions that move and zero ZA array vectors or ZA tile slices.
- SME ZERO instructions that zero ZA array vectors.
- SME LUTI2 and LUTI4 lookup table instructions to Z vectors with strided numbering.

In this Manual, unless stated otherwise, when SME is used, the behavior also applies to SME2.1.

This feature is supported in AArch64 state only.

FEAT_SME2p1 is OPTIONAL from Armv9.2.

In an Armv9.4 implementation, if FEAT_SME2 is implemented, FEAT_SME2p1 is implemented.

If FEAT_SME2p1 is implemented, then FEAT_SME2 is implemented.

If FEAT_SME and FEAT_SVE2p1 are implemented, then FEAT_SME2p1 is implemented.

The following field identifies the presence of FEAT_SME2p1:

- ID_AA64SMFR0_EL1.SMEver.

For more information, see:

- 'Data processing - SME, SME2'.
- 'The Scalable Matrix Extension'.

FEAT_SME_B16B16, Non-widening BFloat16 to BFloat16 SME ZA-targeting arithmetic

FEAT_SME_B16B16 introduces SME ZA-targeting non-widening BFloat16 floating-point instructions to SME.

This feature is supported in AArch64 state only.

FEAT_SME_B16B16 is OPTIONAL from Armv9.2.

If FEAT_SME_B16B16 is implemented, then FEAT_SME2 is implemented.

If FEAT_SME_B16B16 is implemented, then FEAT_SVE_B16B16 is implemented.

The following field identifies the presence of FEAT_SME_B16B16:

- ID_AA64SMFR0_EL1.B16B16.

FEAT_SME_F16F16, Non-widening half-precision FP16 to FP16 arithmetic for SME2

FEAT_SME_F16F16 introduces the SME2 half-precision to single-precision convert instructions and non-widening half-precision floating-point instructions.

This feature is supported in AArch64 state only.

FEAT_SME_F16F16 is OPTIONAL from Armv9.2.

If FEAT_SME_F16F16 is implemented, then FEAT_SME2 is implemented.

The following field identifies the presence of FEAT_SME_F16F16:

- ID_AA64SMFR0_EL1.F16F16.

For more information, see:

- 'Data processing - SME, SME2'.
- 'The Scalable Matrix Extension'.

FEAT_SVE2p1, Scalable Vector Extensions version 2.1

The Scalable Vector Extension version 2.1 (SVE2.1) is a superset of SVE2 that adds:

- SVE instructions in Non-streaming SVE mode, which were previously added by SME in Streaming SVE mode. These are:
  - Contiguous multi-vector load and store instructions.
  - Predicate-as-counter instructions.
  - General-purpose SVE instructions.
- Other relaxations and enhancements.

In this Manual, unless stated otherwise, when SVE is used, the behavior also applies to SVE2.1.

This feature is supported in AArch64 state only.

FEAT_SVE2p1 is OPTIONAL from Armv9.2.

In an Armv9.4 implementation, if FEAT_SVE2 is implemented, FEAT_SVE2p1 is implemented.

If FEAT_SVE2p1 is implemented, then FEAT_SVE2 is implemented.

If FEAT_SVE2 and FEAT_SME2p1 are implemented, then FEAT_SVE2p1 is implemented.

The following field identifies the presence of FEAT_SVE2p1:

- ID_AA64ZFR0_EL1.SVEver.

For more information, see:

- FEAT_SVE.
- 'Loads and stores - SME, SME2, SVE2p1'.
- 'Data processing - SVE2'.

FEAT_SVE_B16B16, Non-widening BFloat16 to BFloat16 arithmetic for SVE2 and SME2

FEAT_SVE_B16B16 introduces the following:

- Advanced SIMD: Non-widening BFloat16 floating-point instructions to SVE, and multi-vector Z-targeting non-widening BFloat16 floating-point instructions to SME.
- SVE: SVE non-widening BFloat16 instructions when the PE is not in Streaming SVE mode.
- SME: The SVE non-widening BFloat16 instructions when the PE is in Streaming SVE mode.
- SME: The SME Z-targeting multi-vector non-widening BFloat16 instructions.

This feature is supported in AArch64 state only.

FEAT_SVE_B16B16 is OPTIONAL from Armv9.2.

If FEAT_SVE_B16B16 is implemented, then FEAT_SME2 or FEAT_SVE2 is implemented.

The following field identifies the presence of FEAT_SVE_B16B16:

- ID_AA64ZFR0_EL1.B16B16.

For more informations, see:

- 'BFloat16 arithmetic'.
- 'BFloat16 minimum/maximum'.
- 'Clamp to minimum/maximum'.

FEAT_SYSINSTR128, 128-bit System instructions

FEAT_SYSINSTR128 introduces support for IMPLEMENTATION DEFINED System instructions that can take 128-bit inputs.

This feature is supported in AArch64 state only.

FEAT_SYSINSTR128 is OPTIONAL from Armv9.3.

If FEAT_SYSINSTR128 is implemented, then FEAT_SCTLR2 is implemented.

If FEAT_SYSINSTR128 is implemented, then FEAT_D128 is implemented.

The following field identifies the presence of FEAT_SYSINSTR128:

- ID_AA64ISAR2_EL1.SYSINSTR_128.

For more information, see The A64 System Instruction Class.

FEAT_SYSREG128, 128-bit System registers

FEAT_SYSREG128 introduces the following support for 128-bit System registers:

- The MRRS instruction to move a 128-bit System register into a pair of 64-bit general-purpose registers.
- The MSRR instruction to move a pair of 64-bit general-purpose registers to a 128-bit System register.
- 128-bit formats of the following system registers:
- The Physical Address Register, PAR_EL1.
- The Read-Check-Write mask registers, RCWMASK_EL1 and RCWSMASK_EL1.
- The following translation table base address registers, TTBR0_EL1, TTBR0_EL2, TTBR1_EL1, TTBR1_EL2, VTTBR_EL2.

This feature is supported in AArch64 state only.

FEAT_SYSREG128 is OPTIONAL from Armv9.3.

If FEAT_SYSREG128 is implemented, then FEAT_SCTLR2 is implemented.

If FEAT_SYSREG128 is implemented, then FEAT_D128 is implemented.

The following field identifies the presence of FEAT_SYSREG128:

- ID_AA64ISAR2_EL1.SYSREG_128.

FEAT_TRBE_EXT, Trace Buffer external mode

FEAT_TRBE_EXT allows an external debugger, as well as a self-hosted debugger, to use the Trace Buffer Unit. All of the following registers are introduced to determine the parameters of the implementation:

- TRBDEVARCH.
- TRBDEVID.
- TRBDEVID1.

This feature is supported in both AArch64 and AArch32 states.

FEAT_TRBE_EXT is OPTIONAL from Armv9.3.

If FEAT_TRBE_EXT is implemented, then FEAT_TRBE is implemented.

The following fields identify the presence of FEAT_TRBE_EXT:

- ID_AA64DFR0_EL1.ExtTrcBuff.
- EDDFR.TraceBuffer.
- EDDFR.ExtTrcBuff.

For more information, see Trace buffer External mode.

FEAT_TRBE_MPAM, Trace Buffer MPAM extensions

FEAT_TRBE_MPAM allows software to program the MPAM PARTID and PMG to use different MPAM values for trace data. TRBDEVID1.{PMG_MAX, PARTID_MAX} are used to determine the parameters of the MPAM implementation.

This feature is supported in both AArch64 and AArch32 states.

FEAT_TRBE_MPAM is OPTIONAL from Armv9.3.

If FEAT_TRBE_MPAM is implemented, then FEAT_TRBE_EXT is implemented.

If FEAT_TRBE_MPAM is implemented, then FEAT_MPAM is implemented.

When FEAT_TRBE_MPAM and FEAT_AA64EL2 are implemented, FEAT_FGT2 is implemented.

The following fields identify the presence of FEAT_TRBE_MPAM:

- TRBIDR_EL1.MPAM.
- TRBIDR_EL1.MPAM.

For more information, see 'External mode and FEAT_MPAM'.

Features added to the Armv9.4 extension in later releases

- FEAT_RME_GDI.
- FEAT_SME_MOP4.
- FEAT_SME_TMOP.
- FEAT_SSVE_BitPerm.
- FEAT_SSVE_FEXPA.

7.16 The Armv9.5 architecture extension

The Armv9.5 architecture extension is an extension to Armv9.4. It adds mandatory and optional architectural features. Some features must be implemented together. An implementation is Armv9.5 compliant if all of the following apply:

- It is Armv9.4 compliant.

- It includes all of the Armv9.5 architectural features that are mandatory.

An Armv9.5 compliant implementation can additionally include:

- Armv9.5 features that are optional.

FEAT_ASID2, Support for concurrent use of two ASIDs

FEAT_ASID2 introduces support for use of one ASID for each TTBR_ELx.

This feature is supported in AArch64 state only.

FEAT_ASID2 is OPTIONAL from Armv9.4.

FEAT_ASID2 is mandatory from Armv9.5.

If FEAT_ASID2 is implemented, then FEAT_TCR2 is implemented.

The following field identifies the presence of FEAT_ASID2:

- ID_AA64MMFR4_EL1.ASID2.

For more information, see Use of ASIDs and VMIDs to reduce TLB maintenance requirements.

FEAT_BWE2, Breakpoint and watchpoint enhancements 2

FEAT_BWE2 provides the capability to generate a Watchpoint exception or debug event when software accesses an address outside of one or more user-supplied address regions.

This feature is supported in both AArch64 and AArch32 states.

FEAT_BWE2 is OPTIONAL from Armv9.4.

If FEAT_BWE2 is implemented, then FEAT_BWE is implemented.

In an Armv9.5 implementation, if FEAT_BWE is implemented, FEAT_BWE2 is implemented.

The following fields identify the presence of FEAT_BWE2:

- ID_AA64DFR2_EL1.BWE.
- EDDFR2.BWE.

For more information, see:

- About Watchpoint exceptions.
- Exception syndrome information, fault address information, and preferred return address.
- Watchpoint data address comparisons.

FEAT_CPA, Instruction-only Checked Pointer Arithmetic

FEAT_CPA introduces support for Checked Pointer Arithmetic instructions.

This feature is supported in AArch64 state only.

FEAT_CPA is OPTIONAL from Armv9.4.

FEAT_CPA is mandatory from Armv9.5.

The following field identifies the presence of FEAT_CPA:

- ID_AA64ISAR3_EL1.CPA.

For more information, see Checked Pointer Arithmetic.

FEAT_CPA2, Checked Pointer Arithmetic

FEAT_CPA2 introduces support for enabling Checked Pointer Arithmetic. Checked Pointer Arithmetic detects and reports corruption of the following:

- The top 8 bits of a pointer.
- Multiplication overflows that lead to invalid array indices being computed.

This feature is supported in AArch64 state only.

FEAT_CPA2 is OPTIONAL from Armv9.4.

If FEAT_CPA2 is implemented, then FEAT_CPA is implemented.

If FEAT_CPA2 is implemented, then FEAT_SCTLR2 is implemented.

The following field identifies the presence of FEAT_CPA2:

- ID_AA64ISAR3_EL1.CPA.

For more information, see Checked Pointer Arithmetic.

FEAT_E3DSE, Delegated SError exception injection

FEAT_E3DSE provides an SError injection mechanism for EL3 called delegated SErrors.

This feature is supported in AArch64 state only.

FEAT_E3DSE is OPTIONAL from Armv9.4.

In an Armv9.5 implementation, if FEAT_AA64EL3 is implemented, FEAT_E3DSE is implemented.

The following field identifies the presence of FEAT_E3DSE:

- ID_AA64MMFR4_EL1.E3DSE.

For more information, see Asynchronous exception types.

FEAT_ETS3, Enhanced Translation Synchronization

FEAT_ETS3 introduces support for enhanced memory access ordering requirements for translation table walks.

This feature is supported in both AArch64 and AArch32 states.

FEAT_ETS3 is OPTIONAL from Armv8.0.

FEAT_ETS3 is mandatory from Armv9.5.

The following fields identify the presence of FEAT_ETS3:

- ID_AA64MMFR1_EL1.ETS.
- ID_MMFR5_EL1.ETS.
- ID_MMFR5.ETS.

For more information, see:

- 'Ordering requirements defined by the formal concurrency model'.
- 'Ordering of memory accesses from translation table walks'.
- 'Ordering of translation table walks'.

FEAT_FAMINMAX, Floating-point maximum and minimum absolute value instructions

FEAT_FAMINMAX introduces the Advanced SIMD, SVE and SME `FAMAX` and `FAMIN` instructions that compute floating-point maximum and minimum absolute value.

FEAT_FAMINMAX is OPTIONAL from Armv9.2.

In an Armv9.5 implementation, if FEAT_FP is implemented, FEAT_FAMINMAX is implemented.

If FEAT_FAMINMAX is implemented, then FEAT_AdvSIMD, FEAT_SVE2, or FEAT_SME2 is implemented.

The following field identifies the presence of FEAT_FAMINMAX:

- ID_AA64ISAR3_EL1.FAMINMAX.

For more information, see:

- SIMD arithmetic.
- Floating-point minimum/maximum absolute value.
- Multi-vector minimum/maximum.

FEAT_FGWTE3, Fine-Grained Write Trap EL3

FEAT_FGWTE3 introduces traps for write accesses at EL3 to individual *_EL3 System registers. The traps are independent of existing controls. The bits in the control register FGWTE3_EL3 are sticky, and writes of 0 are ignored.

This feature is supported in AArch64 state only.

FEAT_FGWTE3 is OPTIONAL from Armv9.4.

If FEAT_FGWTE3 is implemented, then FEAT_EL3 is implemented.

The following field identifies the presence of FEAT_FGWTE3:

- ID_AA64MMFR4_EL1.FGWTE3.

For more information, see EL3 configurable instruction controls.

FEAT_FP8, FP8 convert instructions

FEAT_FP8 introduces the following:

- OFP8 formats E5M2 and E4M3 for FP8 instructions.
- Advanced SIMD: FP8 convert instructions and a floating point scaling `FSCALE` instruction.
- SME: FP8 convert instructions and floating point scaling `FSCALE` instructions.
- SVE: FP8 convert instructions.

FEAT_FP8 is OPTIONAL from Armv9.2.

If FEAT_FP8 is implemented, then FEAT_FPMR is implemented.

If FEAT_FP8 is implemented, then FEAT_FAMINMAX is implemented.

If FEAT_FP8 is implemented, then FEAT_LUT is implemented.

If FEAT_FP8 is implemented, then FEAT_BF16 is implemented.

If FEAT_FP8 is implemented, then FEAT_AdvSIMD, FEAT_SVE2, or FEAT_SME2 is implemented.

The following fields identify the presence of FEAT_FP8:

- ID_AA64FPFR0_EL1.F8CVT.
- ID_AA64FPFR0_EL1.F8E4M3.
- ID_AA64FPFR0_EL1.F8E5M2.

For more information, see:

- FP8 floating-point formats.
- Summary of FP8 instruction behaviors.
- For Advanced SIMD, FP8 floating-point conversion.
- For SVE, FP8 floating-point conversions.
- For SME, FP8 floating-point conversions.
- The *OCP 8-bit Floating Point Specification (OFP8)*.

FEAT_FP8DOT2, FP8 2-way dot product to half-precision instructions

FEAT_FP8DOT2 introduces the Advanced SIMD FP8 to half-precision 2-way dot product instructions.

If FEAT_SVE2 is implemented, FEAT_FP8DOT2 also implements the SVE versions of these instructions.

Note

This feature does not enable execution of the SVE instructions in Streaming SVE mode.

FEAT_FP8DOT2 is OPTIONAL from Armv9.2.

If FEAT_FP8DOT2 is implemented, then FEAT_FP8DOT4 is implemented.

If FEAT_FP8DOT2 is implemented, then FEAT_AdvSIMD or FEAT_SVE2 is implemented.

The following field identifies the presence of FEAT_FP8DOT2:

- ID_AA64FPFR0_EL1.F8DP2.

For more information, see:

- Summary of FP8 instruction behaviors.
- For Advanced SIMD, FP8 floating-point dot product.
- For SVE, FP8 floating-point dot product.

FEAT_FP8DOT4, FP8 4-way dot product to single-precision instructions

FEAT_FP8DOT4 introduces the following:

- Advanced SIMD: FP8 to single-precision 4-way dot product instructions.
- SVE: SVE FP8 to single-precision 4-way dot product instructions.

Note

This feature does not enable execution of the SVE instructions in Streaming SVE mode.

FEAT_FP8DOT4 is OPTIONAL from Armv9.2.

If FEAT_FP8DOT4 is implemented, then FEAT_FP8FMA is implemented.

If FEAT_FP8DOT4 is implemented, then FEAT_AdvSIMD or FEAT_SVE2 is implemented.

The following field identifies the presence of FEAT_FP8DOT4:

- ID_AA64FPFR0_EL1.F8DP4.

For more information, see:

- Summary of FP8 instruction behaviors.
- For Advanced SIMD, FP8 floating-point dot product.
- For SVE, FP8 floating-point dot product.

FEAT_FP8FMA, FP8 multiply-accumulate to half-precision and single-precision instructions

FEAT_FP8FMA introduces the following:

- Advanced SIMD: FP8 to half-precision and single-precision multiply-accumulate instructions.
- SVE: FP8 to half-precision and single-precision multiply-accumulate instructions.


Note

This feature does not enable execution of the SVE instructions in Streaming SVE mode.

FEAT_FP8FMA is OPTIONAL from Armv9.2.

If FEAT_FP8FMA is implemented, then FEAT_FP8 is implemented.

If FEAT_FP8FMA is implemented, then FEAT_AdvSIMD or FEAT_SVE2 is implemented.

The following field identifies the presence of FEAT_FP8FMA:

- ID_AA64FPFR0_EL1.F8FMA.

For more information, see:

- Summary of FP8 instruction behaviors.
- For Advanced SIMD, FP8 floating-point widening multiply-accumulate.
- For SVE, FP8 floating-point widening multiply-accumulate.

FEAT_FPMR, Floating-point Mode Register

FEAT_FPMR introduces the Special-purpose register FPMR.

FEAT_FPMR is OPTIONAL from Armv9.2.

If FEAT_FPMR is implemented, then FEAT_AdvSIMD is implemented.

When FEAT_FPMR and FEAT_EL2 are implemented, FEAT_FGT is implemented.

When FEAT_FPMR and FEAT_EL2 are implemented, FEAT_HCX is implemented.

The following field identifies the presence of FEAT_FPMR:

- ID_AA64PFR2_EL1.FPMR.

For more information, see Summary of FP8 instruction behaviors.

FEAT_HACDBS, Hardware accelerator for cleaning Dirty state

FEAT_HACDBS introduces the Hardware accelerator for cleaning Dirty state, which enhances the process of updating stage 2 descriptors from writable-dirty to writable-clean.

This feature is supported in AArch64 state only.

FEAT_HACDBS is OPTIONAL from Armv9.4.

If FEAT_HACDBS is implemented, then FEAT_HDBSS is implemented.

The following field identifies the presence of FEAT_HACDBS:

- ID_AA64MMFR4_EL1.HACDBS.

For more information, see Hardware accelerator for cleaning dirty state.

FEAT_HDBSS, Hardware Dirty state tracking structure

FEAT_HDBSS introduces the Hardware Dirty state tracking structure, which enhances tracking the dirty state of stage 2 translation table descriptors.

This feature is supported in AArch64 state only.

FEAT_HDBSS is OPTIONAL from Armv9.4.

If FEAT_HDBSS is implemented, then FEAT_HAFDBS is implemented.

If FEAT_HDBSS is implemented, then FEAT_AA64EL2 is implemented.

The following field identifies the presence of FEAT_HDBSS:

- ID_AA64MMFR1_EL1.HAFDBS.

For more information, see Hardware dirty state tracking structure.

FEAT_LUT, Lookup table instructions with 2-bit and 4-bit indices

FEAT_LUT introduces the Advanced SIMD and SVE `LUTI2` and `LUTI4` lookup table instructions with 2-bit and 4-bit indices.

FEAT_LUT is OPTIONAL from Armv9.2.

In an Armv9.5 implementation, if FEAT_AdvSIMD is implemented, FEAT_LUT is implemented.

If FEAT_LUT is implemented, then FEAT_AdvSIMD, FEAT_SVE2, or FEAT_SME2 is implemented.

The following field identifies the presence of FEAT_LUT:

- ID_AA64ISAR2_EL1.LUT.

For more information, see:

- For Advanced SIMD, Lookup table read.
- For SVE, Lookup table read.

FEAT_PAuth_LR, Pointer authentication instructions that allow signing of LR using SP and PC as diversifiers

FEAT_PAuth_LR introduces instructions that allow signing of the 64-bit value in LR against the value of SP, and a PC-relative instruction address.

This feature is supported in AArch64 state only.

FEAT_PAuth_LR is OPTIONAL from Armv9.4.

If FEAT_PAuth_LR is implemented, then FEAT_FPACCOMBINE is implemented.

If FEAT_PAuth_LR is implemented, then FEAT_SCTLR2 is implemented.

When FEAT_PAuth_LR and FEAT_AA64EL2 are implemented, FEAT_HCX is implemented.

The following fields identify the presence of FEAT_PAuth_LR:

- ID_AA64ISAR1_EL1.APA.
- ID_AA64ISAR1_EL1.API.
- ID_AA64ISAR2_EL1.APA3.

FEAT_PMUv3_SME, Performance Monitors extensions for SME

FEAT_PMUv3_SME supports filtering to control counting events in Streaming and Non-streaming SVE modes.

This feature is supported in AArch64 state only.

FEAT_PMUv3_SME is OPTIONAL from Armv9.4.

In an Armv9.5 implementation, if FEAT_PMUv3 and FEAT_SME are implemented, FEAT_PMUv3_SME is implemented.

If FEAT_PMUv3_SME is implemented, then FEAT_PMUv3 is implemented.

If FEAT_PMUv3_SME is implemented, then FEAT_SME is implemented.

The following field identifies the presence of FEAT_PMUv3_SME:

- PMMIR_EL1.SME.

For more information, see Filtering by SVE Streaming mode.

FEAT_PMUv3_TH2, Performance Monitors event counter linking extension

FEAT_PMUv3_TH2 extends the Performance Monitors Extension to add the ability to program an event counter to count the logical combination of two events.

This feature is supported in both AArch64 and AArch32 states. The threshold condition controls are accessible only in AArch64 state. However, threshold conditions still apply in AArch32 state.

FEAT_PMUv3_TH2 is OPTIONAL from Armv9.4.

If FEAT_PMUv3_TH2 is implemented, then FEAT_PMUv3_TH is implemented.

If FEAT_PMUv3_TH2 is implemented, then FEAT_PMUv3_EDGE is implemented.

The following fields identify the presence of FEAT_PMUv3_TH2:

- PMMIR_EL1.EDGE.
- PMMIR.EDGE.

For more information, see Event threshold and edge counting.

FEAT_RME_GPC2, RME Granule Protection Check 2 Extension

FEAT_RME_GPC2 introduces two mechanisms to the GPC that enable memory isolation models:

- Non-secure Only GPI encoding.
- Physical Address Space Disable.

This feature is supported in AArch64 state only.

FEAT_RME_GPC2 is OPTIONAL from Armv9.4.

If FEAT_RME_GPC2 is implemented, then FEAT_RME is implemented.

The following field identifies the presence of FEAT_RME_GPC2:

- ID_AA64PFR0_EL1.RME.

For more information, see:

- GPC faults.
- GPI field encoding in GPT descriptors.

FEAT_SME_F8F16, SME2 ZA-targeting FP8 multiply-accumulate, dot product, and outer product to half-precision instructions

FEAT_SME_F8F16 introduces the following SME instructions:

- ZA-targeting FP8 to half-precision multiply-accumulate, dot product, and outer product instructions.
- ZA-targeting multi-vector non-widening half-precision FADD and FSUB instructions.

FEAT_SME_F8F16 is OPTIONAL from Armv9.2.

If FEAT_SME_F8F16 is implemented, then FEAT_SME_F8F32 is implemented.

The following field identifies the presence of FEAT_SME_F8F16:

- ID_AA64SMFR0_EL1.F8F16.

For more information, see:

- Summary of FP8 instruction behaviors.
- FP8 floating-point widening multiply-accumulate.
- FP8 floating-point dot product.
- FP8 floating-point outer product.

FEAT_SME_F8F32, SME2 ZA-targeting FP8 multiply-accumulate, dot product, and outer product to single-precision instructions

FEAT_SME_F8F32 introduces the SME ZA-targeting FP8 to single-precision multiply-accumulate, dot product, and outer product instructions.

FEAT_SME_F8F32 is OPTIONAL from Armv9.2.

If FEAT_SME_F8F32 is implemented, then FEAT_SME2 is implemented.

If FEAT_SME_F8F32 is implemented, then FEAT_FP8 is implemented.

The following field identifies the presence of FEAT_SME_F8F32:

- ID_AA64SMFR0_EL1.F8F32.

For more information, see:

- Summary of FP8 instruction behaviors.
- FP8 floating-point widening multiply-accumulate.
- FP8 floating-point dot product.
- FP8 floating-point outer product.

FEAT_SME_LUTv2, Lookup table instructions with 4-bit indices and 8-bit elements

FEAT_SME_LUTv2 introduces the following:

- The LUTI4 (four registers, 8-bit) instruction.
- The MOVT (vector to table) instruction.

FEAT_SME_LUTv2 is OPTIONAL from Armv9.2.

If FEAT_SME_LUTv2 is implemented, then FEAT_SME2 is implemented.

The following field identifies the presence of FEAT_SME_LUTv2:

- ID_AA64SMFR0_EL1.LUTv2.

FEAT_SPE_ALTCLK, Statistical Profiling alternate clock domain extension

FEAT_SPE_ALTCLK supports collection of timing data using the Statistical Profiling Extension when profiling a PE that includes asynchronous accelerators, such as a Streaming Mode Compute Unit (SMCU).

This feature is supported in AArch64 state only.

FEAT_SPE_ALTCLK is OPTIONAL from Armv9.4.

If FEAT_SPE_ALTCLK is implemented, then FEAT_SPE is implemented.

The following field identifies the presence of FEAT_SPE_ALTCLK:

- PMSIDR_EL1.ALTCLK.

For more information, see Asynchronous operation.

FEAT_SPE_EFT, Statistical Profiling extended filtering by type

FEAT_SPE_EFT extends SPE filtering to support targeted profiling of floating-point and SIMD operations.

This feature is supported in AArch64 state only.

FEAT_SPE_EFT is OPTIONAL from Armv9.4.

In an Armv9.5 implementation, if FEAT_SPE is implemented, FEAT_SPE_EFT is implemented.

If FEAT_SPE_EFT is implemented, then FEAT_SPE is implemented.

If FEAT_SPE_EFT is implemented, then FEAT_SPE_FPF is implemented.

The following field identifies the presence of FEAT_SPE_EFT:

- PMSIDR_EL1.EFT.

For more information, see Filtering by operation type.

FEAT_SPE_FPF, Statistical Profiling floating-point and SIMD flag extension

FEAT_SPE_FPF extends the profiling information recorded by the Statisical Profiling Extension to identify sampled scalar floating-point and Advanced SIMD instructions.

This feature is supported in AArch64 state only.

FEAT_SPE_FPF is OPTIONAL from Armv9.4.

In an Armv9.5 implementation, if FEAT_SPE is implemented, FEAT_SPE_FPF is implemented.

If FEAT_SPE_FPF is implemented, then FEAT_SPE is implemented.

If FEAT_SPE_FPF is implemented, then FEAT_SPE_EFT is implemented.

The following field identifies the presence of FEAT_SPE_FPF:

- PMSIDR_EL1.FPF.

For more information, see Additional information for other operations.

FEAT_SPE_SME, Statisical Profiling extensions for SME

FEAT_SPE_SME provides support for profiling of software that uses SME instructions using the Statistical Profiling Extension.

This feature is supported in AArch64 state only.

FEAT_SPE_SME is OPTIONAL from Armv9.2.

In an Armv9.5 implementation, if FEAT_SPEv1p1 and FEAT_SME are implemented, FEAT_SPE_SME is implemented.

If FEAT_SPE_SME is implemented, then FEAT_SPEv1p1 is implemented.

If FEAT_SPE_SME is implemented, then FEAT_SME is implemented.

The following field identifies the presence of FEAT_SPE_SME:

- PMSIDR_EL1.SME.

For more information, see Additional information for each profiled SVE and SME operation.

FEAT_SPMU2, System Performance Monitors Extension version 2

FEAT_SPMU2 provides the functionality to set multiple event counters in a System PMU to zero in a single operation.

This feature is supported in both AArch64 and AArch32 states.

FEAT_SPMU2 is OPTIONAL from Armv9.4.

If FEAT_SPMU and v9Ap5 are implemented, then FEAT_SPMU2 is implemented.

If FEAT_SPMU2 is implemented, then FEAT_SPMU is implemented.

The following field identifies the presence of FEAT_SPMU2:

- ID_AA64DFR1_EL1.SPMU.

For more information, see Zeroing System PMU counters.

FEAT_SSVE_FP8DOT2, SVE FP8 2-way dot product to half-precision instructions in Streaming SVE mode

FEAT_SSVE_FP8DOT2 enables execution of SVE FP8 to half-precision 2-way dot product instructions in Streaming SVE mode.

FEAT_SSVE_FP8DOT2 is OPTIONAL from Armv9.2.

If FEAT_SSVE_FP8DOT2 is implemented, then FEAT_SSVE_FP8DOT4 is implemented.

If FEAT_SME2 and FEAT_FP8DOT2 are implemented, then FEAT_SSVE_FP8DOT2 is implemented.

The following field identifies the presence of FEAT_SSVE_FP8DOT2:

- ID_AA64SMFR0_EL1.SF8DP2.

For more information, see:

- Summary of FP8 instruction behaviors.
- FP8 floating-point dot product.

FEAT_SSVE_FP8DOT4, SVE2 FP8 4-way dot product to single-precision instructions in Streaming SVE mode

FEAT_SSVE_FP8DOT4 enables execution of SVE FP8 to single-precision 4-way dot product instructions in Streaming SVE mode.

FEAT_SSVE_FP8DOT4 is OPTIONAL from Armv9.2.

If FEAT_SSVE_FP8DOT4 is implemented, then FEAT_SSVE_FP8FMA is implemented.

If FEAT_SME2 and FEAT_FP8DOT4 are implemented, then FEAT_SSVE_FP8DOT4 is implemented.

The following field identifies the presence of FEAT_SSVE_FP8DOT4:

- ID_AA64SMFR0_EL1.SF8DP4.

For more information, see:

- Summary of FP8 instruction behaviors.
- FP8 floating-point dot product.

FEAT_SSVE_FP8FMA, SVE2 FP8 multiply-accumulate to half-precision and single-precision instructions in Streaming SVE mode

FEAT_SSVE_FP8FMA enables execution of SVE FP8 to half-precision and single-precision multiply-accumulate instructions in Streaming SVE mode.

FEAT_SSVE_FP8FMA is OPTIONAL from Armv9.2.

If FEAT_SSVE_FP8FMA is implemented, then FEAT_FP8 is implemented.

If FEAT_SSVE_FP8FMA is implemented, then FEAT_SME2 is implemented.

If FEAT_SME2 and FEAT_FP8FMA are implemented, then FEAT_SSVE_FP8FMA is implemented.

The following field identifies the presence of FEAT_SSVE_FP8FMA:

- ID_AA64SMFR0_EL1.SF8FMA.

For more information, see:

- Summary of FP8 instruction behaviors.
- FP8 floating-point widening multiply-accumulate.

FEAT_STEP2, Enhanced Software Step Extension

FEAT_STEP2 introduces support for a debugger to control the instruction executed by the PE when single-stepping, without modifying the instruction in memory.

This feature is supported in both AArch64 and AArch32 states.

FEAT_STEP2 is OPTIONAL from Armv9.4.

FEAT_STEP2 is mandatory from Armv9.5.

When FEAT_STEP2 and FEAT_EL2 are implemented, FEAT_FGT2 is implemented.

The following field identifies the presence of FEAT_STEP2:

- ID_AA64DFR2_EL1.STEP.

For more information, see Behavior in the active-not-pending state.

FEAT_TLBIW, TLBI VMALL for Dirty state

FEAT_TLBIW provides the following TLBI instructions that remove the stage 2 dirty state from TLB entries:

- TLBI VMALLWS2E1, TLBI VMALLWS2E1NXS.
- TLBI VMALLWS2E1IS, TLBI VMALLWS2E1ISNXS.
- TLBI VMALLWS2E1OS, TLBI VMALLWS2E1OSNXS.

This feature is supported in AArch64 state only.

FEAT_TLBIW is OPTIONAL from Armv9.4.

If FEAT_TLBIW is implemented, then FEAT_AA64EL2 is implemented.

The following field identifies the presence of FEAT_TLBIW:

- ID_AA64ISAR3_EL1.TLBIW.

7.17 The Armv9.6 architecture extension

The Armv9.6 architecture extension is an extension to Armv9.5. It adds mandatory and optional architectural features. Some features must be implemented together. An implementation is Armv9.6 compliant if all of the following apply:

- It is Armv9.5 compliant.
- It includes all of the Armv9.6 architectural features that are mandatory.

An Armv9.6 compliant implementation can additionally include:

- Armv9.6 features that are optional.

FEAT_CMPBR, Compare and Branch instructions

FEAT_CMPBR introduces the A64 base compare and branch instructions.

This feature is supported in AArch64 state only.

FEAT_CMPBR is OPTIONAL from Armv9.5.

FEAT_CMPBR is mandatory from Armv9.6.

The following field identifies the presence of FEAT_CMPBR:

- ID_AA64ISAR2_EL1.CSSC.

FEAT_F8F16MM, 8-bit floating-point matrix multiply-accumulate to half-precision

FEAT_F8F16MM introduces the Advanced SIMD 8-bit floating-point matrix multiply-accumulate to half-precision instruction.

If FEAT_SVE2 is implemented, FEAT_F8F16MM implements the SVE 8-bit floating-point matrix multiply-accumulate to half-precision instruction, when the PE is not in Streaming SVE mode.

This feature is supported in AArch64 state only.

FEAT_F8F16MM is OPTIONAL from Armv9.2.

If FEAT_F8F16MM is implemented, then FEAT_F8F32MM is implemented.

If FEAT_F8F16MM is implemented, then FEAT_FP8DOT2 is implemented.

FEAT_F8F32MM, 8-bit floating-point matrix multiply-accumulate to single-precision

FEAT_F8F32MM introduces the Advanced SIMD 8-bit floating-point matrix multiply-accumulate to single-precision instruction.

If FEAT_SVE2 is implemented, FEAT_F8F32MM implements the SVE 8-bit floating-point matrix multiply-accumulate to single-precision instruction, when the PE is not in Streaming SVE mode.

This feature is supported in AArch64 state only.

FEAT_F8F32MM is OPTIONAL from Armv9.2.

If FEAT_F8F32MM is implemented, then FEAT_FP8DOT4 is implemented.

FEAT_FPRCVT, Floating-Point to/from Integer in Scalar FP register

FEAT_FPRCVT introduces A64 base floating-point to integer and integer to floating-point convert instructions with only scalar SIMD&FP register operands and results, and with different input and output register sizes.

If FEAT_SME2p1 is implemented, FEAT_FPRCVT indicates support for conversions between floating-point and integer instructions with only scalar SIMD&FP register operands and results, and with the same input and output register sizes when the PE is in Streaming SVE mode.

This feature is supported in AArch64 state only.

FEAT_FPRCVT is OPTIONAL from Armv9.5.

If FEAT_FP and v9Ap6 are implemented, then FEAT_FPRCVT is implemented.

If FEAT_FPRCVT is implemented, then FEAT_FP is implemented.

The following field identifies the presence of FEAT_FPRCVT:

- ID_AA64ISAR3_EL1.FPRCVT.

FEAT_IDTE3, Trapping ID register accesses to EL3

FEAT_IDTE3 introduces support for trapping ID register accesses to EL3.

This feature is supported in AArch64 state only.

In an Armv9.6 implementation, if FEAT_EL3 is implemented, FEAT_IDTE3 is implemented.

FEAT_IDTE3 is OPTIONAL from Armv9.0.

The following field identifies the presence of FEAT_IDTE3:

- ID_AA64MMFR2_EL1.IDS.

FEAT_LS64WB, LS64 for Write-back cacheable memory

FEAT_LS64WB provides support for large single-copy atomic load and store instructions for accesses to Conventional memory to enable scalability of atomic message passing without relying on separate flag and data accesses.

This feature is supported in AArch64 state only.

FEAT_LS64WB is OPTIONAL from Armv9.2.

If FEAT_LS64WB is implemented, then FEAT_LS64_ACCDATA is implemented.

The following field identifies the presence of FEAT_LS64WB:

- ID_AA64ISAR1_EL1.LS64.

FEAT_LSFE, Large System Float Extension

FEAT_LSFE implements A64 base Atomic floating-point in-memory instructions.

This feature is supported in AArch64 state only.

FEAT_LSFE is OPTIONAL from Armv9.3.

If FEAT_LSFE is implemented, then FEAT_FP is implemented.

The following field identifies the presence of FEAT_LSFE:

- ID_AA64ISAR3_EL1.LSFE.

FEAT_LSUI, Unprivileged Load Store

FEAT_LSUI introduces unprivileged variants of load and store instructions so that clearing PSTATE.PAN is never required in privileged software.

This feature is supported in AArch64 state only.

FEAT_LSUI is OPTIONAL from Armv9.5.

FEAT_LSUI is mandatory from Armv9.6.

The following field identifies the presence of FEAT_LSUI:

- ID_AA64ISAR3_EL1.LSUI.

FEAT_MPAM_MSC_DCTRL, MPAM Default Resource Control

FEAT_MPAM_MSC_DCTRL introduces support in an MSC for a default resource control configuration, which is applied to requests with PARTIDs outside the MSC's supported range, or with PARTIDs for which there is no configuration set.

This feature is supported in AArch64 state only.

FEAT_MPAM_MSC_DCTRL is OPTIONAL from Armv9.5.

If FEAT_MPAM_MSC_DCTRL is implemented, then FEAT_MPAMv1p1 or FEAT_MPAMv0p1 is implemented.

The following field identifies the presence of FEAT_MPAM_MSC_DCTRL:

- MPAMF_IDR.HAS_DEFAULT_PARTID.

FEAT_MPAM_MSC_DOMAINS, MPAM Domains PARTID translation

FEAT_MPAM_MSC_DOMAINS introduces support for MSCs to manipulate the MPAM information bundle by applying PARTID ingress translation, egress translation, or both.

This feature is supported in AArch64 state only.

FEAT_MPAM_MSC_DOMAINS is OPTIONAL from Armv9.5.

If FEAT_MPAM_MSC_DOMAINS is implemented, then FEAT_MPAMv1p1 or FEAT_MPAMv0p1 is implemented.

The following fields identify the presence of FEAT_MPAM_MSC_DOMAINS:

- MPAMF_IDR.HAS_IN_TL.
- MPAMF_IDR.HAS_OUT_TL.

FEAT_MPAM_PE_BW_CTRL, MPAM PE-side Bandwidth Controls

FEAT_MPAM_PE_BW_CTRL introduces support for PE-side bandwidth controls.

This feature is supported in AArch64 state only.

FEAT_MPAM_PE_BW_CTRL is OPTIONAL from Armv9.3.

If FEAT_MPAM_PE_BW_CTRL is implemented, then FEAT_MPAMv1p1 or FEAT_MPAMv0p1 is implemented.

The following field identifies the presence of FEAT_MPAM_PE_BW_CTRL:

- MPAMIDR_EL1.HAS_BW_CTRL.

FEAT_NV2p1, Enhanced nested virtualization support

FEAT NV2p1 further supports nested virtualization by ensuring that control bits in EL1 registers are stateful if the corresponding bits in the corresponding EL2 registers are stateful. This is to prevent loss of the Guest hypervisor's state, and to permit the Host hypervisor to correctly emulate the EL2 environment.

This feature is supported in AArch64 state only.

FEAT_NV2p1 is OPTIONAL from Armv9.5.

In an Armv9.6 implementation, if FEAT_NV is implemented, FEAT_NV2p1 is implemented.

If FEAT_NV2p1 is implemented, then FEAT_NV2 is implemented.

The following field identifies the presence of FEAT_NV2p1:

- ID_AA64MMFR4_EL1.NV_frac.

FEAT_OCCMO, Outer Cacheable Cache Maintenance Operation

FEAT_OCCMO introduces the DC CIVAOC system instruction that provides software with a mechanism to publish writes to the Outer cache level.

This feature is supported in AArch64 state only.

FEAT_OCCMO is OPTIONAL from Armv9.5.

FEAT_OCCMO is mandatory from Armv9.6.

The following field identifies the presence of FEAT_OCCMO:

- ID_AA64ISAR3_EL1.OCCMO.

FEAT_PCDPHINT, Producer-Consumer Data Placement Hints

FEAT_PCDPHINT provides hint instructions to indicate each of the following:

- A store in the current execution thread is generating data at a specific location, which a thread of execution on one or more other observers is waiting on.
- The thread of execution on the current PE will read a location that may not yet have been written with the value to be consumed.

This feature is supported in AArch64 state only.

FEAT_PCDPHINT is OPTIONAL from Armv9.0.

The following field identifies the presence of FEAT_PCDPHINT:

- ID_AA64ISAR2_EL1.PCDPHINT.

FEAT_PMUv3_EXTPMN, Reserving PMU event counters for external agents

FEAT_PMUv3_EXTPMN provides external agents with a mechanism to reserve event counters for external use.

This feature is supported in both AArch64 and AArch32 states.

FEAT_PMUv3_EXTPMN is OPTIONAL from Armv9.5.

If FEAT_PMUv3_EXTPMN is implemented, then FEAT_PMUv3_EXT is implemented.

If FEAT_PMUv3_EXTPMN is implemented, then FEAT_FGT is implemented.

When FEAT_PMUv3_EXTPMN and FEAT_EL2 are implemented, FEAT_HPMN0 is implemented.

The following field identifies the presence of FEAT_PMUv3_EXTPMN:

- PMDEVID.EXTPMN.

FEAT_PoPS, Point of Physical Storage

FEAT_PoPS defines the Point of Physical Storage (PoPS) and introduces System instructions to clean and invalidate to PoPS.

This feature is supported in AArch64 state only.

FEAT_PoPS is OPTIONAL from Armv9.5.

FEAT_RME_GDI, RME Granular Data Isolation extension

FEAT_RME_GDI introduces additional GPT GPI encodings to enable memory isolation of non-PE data flows from the PEs, within an RME system.

This feature is supported in AArch64 state only.

FEAT_RME_GDI is OPTIONAL from Armv9.4.

If FEAT_RME_GDI is implemented, then FEAT_RME is implemented.

If FEAT_RME_GDI is implemented, then FEAT_RME_GPC2 is implemented.

The following field identifies the presence of FEAT_RME_GDI:

- ID_AA64MMFR4_EL1.RMEGDI.

FEAT_RME_GPC3, RME Granule Protection Check 3 Extension

FEAT_RME_GPC3 introduces a method for defining a set of windows in the memory map for which Granule Protection Checks are skipped, and instead applies a set of default settings associated with the window.

This feature is supported in AArch64 state only.

FEAT_RME_GPC3 is OPTIONAL from Armv9.5.

If FEAT_RME_GPC3 is implemented, then FEAT_RME is implemented.

If FEAT_RME_GPC3 is implemented, then FEAT_RME_GPC2 is implemented.

The following field identifies the presence of FEAT_RME_GPC3:

- ID_AA64PFR0_EL1.RME.

FEAT_SME2p2, Scalable Matrix Extension version 2.2

The Scalable Matrix Extension version 2.2 (SME2.2) is a superset of SME2.1 that adds:

- SME multi-vector Z-targeting non-widening floating-point FMUL instruction.
- Quarter-tile outer product instructions.
- Structured sparsity outer product instructions.
- Support for SVE2.1 instructions in Streaming SVE mode.

This feature is supported in AArch64 state only.

FEAT_SME2p2 is OPTIONAL from Armv9.5.

In an Armv9.6 implementation, if FEAT_SME is implemented, FEAT_SME2p2 is implemented.

If FEAT_SME2p2 is implemented, then FEAT_SME2p1 is implemented.

If FEAT_SME2p2 is implemented, then FEAT_SME_MOP4 is implemented.

If FEAT_SME2p2 is implemented, then FEAT_SME_TMOP is implemented.

If FEAT_SME2p2 is implemented, then FEAT_SSVE_BitPerm is implemented.

If FEAT_SME2p2 is implemented, then FEAT_SSVE_FEXPA is implemented.

If FEAT_SME and FEAT_SVE2p2 are implemented, then FEAT_SME2p2 is implemented.

FEAT_SME_MOP4, Quarter-tile outer product instructions

FEAT_SME_MOP4 introduces the SME Quarter-tile outer product instructions

FEAT_SME_MOP4 is OPTIONAL from Armv9.4.

If FEAT_SME_MOP4 is implemented, then FEAT_SME2p1 is implemented.

The following field identifies the presence of FEAT_SME_MOP4:

- ID_AA64SMFR0_EL1.SMOP4.

FEAT_SME_TMOP, Structured sparsity outer product instructions

FEAT_SME_TMOP introduces the SME Structured sparsity outer product instructions.

FEAT_SME_TMOP is OPTIONAL from Armv9.4.

If FEAT_SME_TMOP is implemented, then FEAT_SME2p1 is implemented.

The following field identifies the presence of FEAT_SME_TMOP:

- ID_AA64SMFR0_EL1.STMOP.

FEAT_SPE_EXC, SPE Profiling exception extension

FEAT_SPE_EXC provides support for reporting Profiling Buffer management events as SPE Profiling exceptions.

This feature is supported in AArch64 state only.

FEAT_SPE_EXC is OPTIONAL from Armv9.5.

If FEAT_SPE_EXC is implemented, then FEAT_SPEv1p5 is implemented.

The following field identifies the presence of FEAT_SPE_EXC:

- ID_AA64DFR2_EL1.SPE_EXC.

FEAT_SPE_nVM, Statistical Profiling physical addressing mode extension

FEAT_SPE_nVM provides support for defining the Profiling Buffer using physical addresses or intermediate physical addresses.

This feature is supported in AArch64 state only.

FEAT_SPE_nVM is OPTIONAL from Armv9.5.

If FEAT_SPE_nVM is implemented, then FEAT_SPE is implemented.

When FEAT_SPE_nVM and FEAT_EL2 are implemented, FEAT_FGT2 is implemented.

The following field identifies the presence of FEAT_SPE_nVM:

- ID_AA64DFR2_EL1.SPE_nVM.

FEAT_SPEv1p5, Statistical Profiling Extension version 1.5

FEAT_SPEv1p5 introduces the following to the Statistical Profiling Extension:

- If EL2 and FEAT_FGT are implemented, a fine-grained trap on the `PSB CSYNC` instruction.
- The SPE Profiling exception extension, FEAT_SPE_EXC.

- The Statistical Profiling physical addressing mode extension, FEAT_SPE_nVM.

This feature is supported in AArch64 state only.

FEAT_SPEv1p5 is OPTIONAL from Armv9.5.

In an Armv9.6 implementation, if FEAT_SPE is implemented, FEAT_SPEv1p5 is implemented.

If FEAT_SPEv1p5 is implemented, then FEAT_SPEv1p4 is implemented.

If FEAT_SPEv1p5 is implemented, then FEAT_SPE_CRR is implemented.

If FEAT_SPEv1p5 is implemented, then FEAT_SPE_EXC is implemented.

If FEAT_SPEv1p5 is implemented, then FEAT_SPE_nVM is implemented.

The following field identifies the presence of FEAT_SPEv1p5:

- ID_AA64DFR0_EL1.PMSVer.

FEAT_SRMASK, Bitwise System Register Write Masks

FEAT_SRMASK introduces aliases and bitwise write masks for EL1 control registers to reduce trapping of EL1 System register accesses to EL2. The equivalent bitwise write masks are also present for EL2.

This feature is supported in AArch64 state only.

FEAT_SRMASK is OPTIONAL from Armv9.5.

FEAT_SRMASK is mandatory from Armv9.6.

If FEAT_SRMASK is implemented, then FEAT_E2H0 is not implemented.

FEAT_SSVE_AES, Streaming SVE Mode Advanced Encryption Standard and 128-bit polynomial multiply long instructions

FEAT_SSVE_AES implements the SVE AES and 128-bit polynomial multiply long instructions, identified as implemented by ID_AA64ZFR0_EL1.AES field, when the PE is in Streaming SVE mode.

This feature is supported in AArch64 state only.

FEAT_SSVE_AES is OPTIONAL from Armv9.5.

If FEAT_SSVE_AES is implemented, then FEAT_SME2p1 is implemented.

The following field identifies the presence of FEAT_SSVE_AES:

- ID_AA64SMFR0_EL1.AES.

FEAT_SSVE_BitPerm, Streaming Scalable Vector Bit Permutes instructions

FEAT_SSVE_BitPerm implements the SVE2 bit permute instructions, identified as implemented by ID_AA64ZFR0_EL1.BitPerm, when the PE is in Streaming SVE mode.

This feature is supported in AArch64 state only.

FEAT_SSVE_BitPerm is OPTIONAL from Armv9.4.

If FEAT_SSVE_BitPerm is implemented, then FEAT_SME2p1 is implemented.

The following field identifies the presence of FEAT_SSVE_BitPerm:

- ID_AA64SMFR0_EL1.SBitPerm.

FEAT_SSVE_FEXPA, Streaming FEXPA instruction

FEAT_SSVE_FEXPA implements the SVE FEXPA instruction when the PE is in Streaming SVE mode.

This feature is supported in AArch64 state only.

FEAT_SSVE_FEXPA is OPTIONAL from Armv9.4.

If FEAT_SSVE_FEXPA is implemented, then FEAT_SME2p1 is implemented.

The following field identifies the presence of FEAT_SSVE_FEXPA:

- ID_AA64SMFR0_EL1.SFEXPA.

FEAT_SVE2p2, Scalable Vector Extensions version 2.2

The Scalable Vector Extension version 2.2 (SVE2.2) is a superset of SVE2.1 that adds:

- Floating-point round to integral floating-point value instructions.
- Scalar index of first/last true predicate element.
- Zeroing-predication variant to all SVE unary istructions.
- Additional variants of `COMPACT` and `EXPAND` instructions.

This feature is supported in AArch64 state only.

FEAT_SVE2p2 is OPTIONAL from Armv9.5.

In an Armv9.6 implementation, if FEAT_SVE2 is implemented, FEAT_SVE2p2 is implemented.

If FEAT_SVE2p2 is implemented, then FEAT_SVE2p1 is implemented.

If FEAT_SVE2 and FEAT_SME2p2 are implemented, then FEAT_SVE2p2 is implemented.

The following field identifies the presence of FEAT_SVE2p2:

- ID_AA64ZFR0_EL1.SVEver.

FEAT_SVE_AES2, SVE multi-vector Advanced Encryption Standard and 128-bit polynomial multiply long instructions

FEAT_SVE_AES2 implements the SVE multi-vector Advanced Encryption Standard and 128-bit destination element polynomial multiply long instructions, when the PE is not in Streaming SVE mode.

This feature is supported in AArch64 state only.

FEAT_SVE_AES2 is OPTIONAL from Armv9.5.

If FEAT_SVE_AES2 is implemented, then FEAT_SVE_PMULL128 is implemented.

If FEAT_SVE_AES2 is implemented, then FEAT_SVE2p1 or FEAT_SSVE_AES is implemented.

The following field identifies the presence of FEAT_SVE_AES2:

- ID_AA64ZFR0_EL1.AES.

FEAT_SVE_BFSCALE, BFloat16 Floating-Point Adjust Exponent

FEAT_SVE_BFSCALE introduces the SVE `BFSCALE` instruction, when the PE is not in Streaming SVE mode.

If FEAT_SME2 is implemented, FEAT_SVE_BFSCALE also introduces SME multi-vector Z-targeting BFloat16 scaling instructions, `BFSCALE` and `BFMUL`.

This feature is supported in AArch64 state only.

FEAT_SVE_BFSCALE is OPTIONAL from Armv9.2.

If FEAT_SVE_BFSCALE is implemented, then FEAT_SVE_B16B16 is implemented.

The following field identifies the presence of FEAT_SVE_BFSCALE:

- ID_AA64ZFR0_EL1.B16B16.

FEAT_SVE_F16F32MM, SVE Half-precision floating-point matrix multiply-accumulate to single-precision

FEAT_SVE_F16F32MM introduces the SVE half-precision floating-point matrix multiply-accumulate to single-precision instruction.

This feature is supported in AArch64 state only.

FEAT_SVE_F16F32MM is OPTIONAL from Armv9.2.

If FEAT_SVE_F16F32MM is implemented, then FEAT_SVE2p1 is implemented.

FEAT_TRBE_EXC, Trace Buffer Profiling exception extension

FEAT_TRBE_EXC provides support for reporting trace buffer management events as TRBE Profiling exceptions.

This feature is supported in AArch64 state only.

FEAT_TRBE_EXC is OPTIONAL from Armv9.5.

If FEAT_TRBE_EXC is implemented, then FEAT_TRBEv1p1 is implemented.

The following field identifies the presence of FEAT_TRBE_EXC:

- ID_AA64DFR2_EL1.TRBE_EXC.

FEAT_TRBEv1p1, Trace Buffer Extension version 1.1

FEAT_TRBEv1p1 introduces the following to the Trace Buffer Extension:

- If EL2 and FEAT_FGT are implemented, a fine-grained trap on the `TSB CSYNC` instruction.
- If EL2 is implemented, an EL2 control to override TRBLIMITR_EL1.nVM.
- Support for the TRBE Profiling exception extension, FEAT_TRBE_EXC.

This feature is supported in AArch64 state only.

FEAT_TRBEv1p1 is OPTIONAL from Armv9.5.

In an Armv9.6 implementation, if FEAT_TRBE is implemented, FEAT_TRBEv1p1 is implemented.

If FEAT_TRBEv1p1 is implemented, then FEAT_TRBE is implemented.

If FEAT_TRBEv1p1 is implemented, then FEAT_TRBE_EXC is implemented.

The following fields identify the presence of FEAT_TRBEv1p1:

- ID_AA64DFR0_EL1.TraceBuffer.
- TRBDEVARCH.REVISION.

FEAT_UINJ, Injection of Undefined Instruction exceptions

FEAT_UINJ introduces support for software injection of Undefined Instruction exceptions.

This feature is supported in AArch64 state only.

FEAT_UINJ is OPTIONAL from Armv9.0.

FEAT_UINJ is mandatory from Armv9.6.

The following field identifies the presence of FEAT_UINJ:

- ID_AA64PFR2_EL1.UINJ.