



Arm[®] Architecture Reference Manual Supplement, The Scalable Matrix Extension (SME), for Armv9-A

Document number	DDI0616
Document quality	EAC
Document version	A.a
Document confidentiality	Non-confidential
Document build information	a950072 Monday, 7 February 2022 at 11:07

Copyright © 2022 Arm Limited or its affiliates. All rights reserved.

Arm® Architecture Reference Manual Supplement, The Scalable Matrix Extension (SME), for Armv9-A

Release information

Date	Version	Changes
2022/Feb/07	A.a	• First release.

Non-Confidential Proprietary Notice

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

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

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

This document may include technical inaccuracies or typographical errors.

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

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

This document may be translated into other languages for convenience, and you agree that if there is any conflict between the English version of this document and any translation, the terms of the English version of the Agreement shall prevail.

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

Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.

Arm Limited. Company 02557590 registered in England.

110 Fulbourn Road, Cambridge, England CB1 9NJ.

LES-PRE-20349 version 21.0

Product Status

The information in this document is final; that is, it is for a developed product.

The information in this Manual is at EAC quality, which means that:

- All features of the specification are described in the manual.
- Information can be used for software and hardware development.

Contents

Arm® Architecture Reference Manual Supplement, The Scalable Matrix Extension (SME), for Armv9-A

Arm® Architecture Reference Manual Supplement, The Scalable Matrix Extension (SME), for Armv9-A	ii
Release information	ii
Non-Confidential Proprietary Notice	iii
Product Status	iii

Preface

About this supplement	x
Conventions	xi
Typographical conventions	xi
Numbers	xi
Pseudocode descriptions	xi
Asterisks in instruction mnemonics	xi
Assembler syntax descriptions	xii
Rules-based writing	xiii
Identifiers	xiii
Examples	xiii
Additional reading	xiv
Feedback	xv
Feedback on this book	xv
Progressive terminology commitment	xvi

Part A Introduction

Chapter A1	SME Introduction	
A1.1	About the Scalable Matrix Extension (SME)	18
Chapter A2	Architecture Features and Extensions	
A2.1	Extensions and features defined by SME	19
A2.2	Changes to existing features and extension requirements	20

Part B SME application level programmers' model

Chapter B1	Application processing modes	
B1.1	Overview	22
B1.1.1	Process state	22
Chapter B2	Architectural state	
B2.1	Architectural state summary	26
B2.2	SME ZA storage	28
B2.2.1	ZA array vector access	28
B2.2.2	ZA tile access	28
B2.2.3	Accessing an 8-bit element ZA tile	29
B2.2.4	Accessing a 16-bit element ZA tile	30
B2.2.5	Accessing a 32-bit element ZA tile	31
B2.2.6	Accessing a 64-bit element ZA tile	32

B2.2.7	Accessing a 128-bit element ZA tile	32
B2.3	ZA storage layout	34
B2.3.1	ZA array vector and tile slice mappings	34
B2.3.2	Tile mappings	34
B2.3.3	Horizontal tile slice mappings	36
B2.3.4	Vertical tile slice mappings	36
B2.3.5	Mixed horizontal and vertical tile slice mappings	37

Chapter B3

Floating-point behaviors

B3.1	Overview	39
B3.1.1	Extended BFloat16	39
B3.1.2	BFloat16 behaviors	39
B3.1.3	Floating-point behaviors in Streaming SVE mode	41
B3.1.4	ZA-targeting floating-point behaviors	41

Part C SME system level programmers' model

Chapter C1

Introduction

Chapter C2

System management

C2.1	Overview	45
C2.1.1	Identification	46
C2.1.2	Traps and exceptions	46
C2.1.3	Vector lengths	47
C2.1.4	Streaming execution priority	48
C2.2	Processor behavior	49
C2.2.1	Exception priorities	49
C2.2.2	Synchronous Data Abort	51
C2.2.3	Validity of SME and SVE state	51
C2.2.4	Streaming execution priority for shared implementations	52
C2.2.5	Security considerations	53
C2.3	Changes to existing System registers	54
C2.3.1	ID_AA64PFR1_EL1	54
C2.3.2	ID_AA64ZFR0_EL1	54
C2.3.3	CPACR_EL1	54
C2.3.4	CPTR_EL2	54
C2.3.5	CPTR_EL3	55
C2.3.6	HCR_EL2	55
C2.3.7	HCRX_EL2	55
C2.3.8	SCR_EL3	55
C2.3.9	SCTLR_EL1	55
C2.3.10	SCTLR_EL2	56
C2.3.11	HFGTR_EL2	56
C2.3.12	HFGWTR_EL2	56
C2.3.13	ESR_EL1, ESR_EL2, and ESR_EL3	56
C2.3.14	ZCR_EL1, ZCR_EL2, and ZCR_EL3	57
C2.4	SME-specific System registers	58
C2.4.1	ID_AA64SMFR0_EL1	58
C2.4.2	SMCR_EL1	58
C2.4.3	SMCR_EL2	58
C2.4.4	SMCR_EL3	58
C2.4.5	SVCR	58
C2.4.6	SMPRI_EL1	58
C2.4.7	SMPRMAP_EL2	59

C2.4.8	SMIDR_EL1	59
C2.4.9	TPIDR2_EL0	59

Chapter C3

Interaction with other Armv9-A architectural features

C3.1	Overview	60
C3.2	Other architectural features	61
C3.2.1	Watchpoints	61
C3.2.2	Self-hosted debug	64
C3.2.3	External debug	64
C3.2.4	Memory Tagging Extension (MTE)	65
C3.2.5	Reliability, Availability, and Serviceability (RAS)	65
C3.2.6	Memory Partitioning and Monitoring (MPAM)	65
C3.2.7	Transactional Memory Extension (TME)	65
C3.2.8	Memory consistency model	66

Part D SME instruction set

Chapter D1

SME instructions

D1.1	SME data-processing instructions	69
D1.1.1	ADDHA	69
D1.1.2	ADDSP	71
D1.1.3	ADDVL	72
D1.1.4	ADDVA	73
D1.1.5	BFMOPA	75
D1.1.6	BFMOPS	77
D1.1.7	FMOPA (non-widening)	79
D1.1.8	FMOPA (widening)	81
D1.1.9	FMOPS (non-widening)	83
D1.1.10	FMOPS (widening)	85
D1.1.11	LD1B	87
D1.1.12	LD1D	89
D1.1.13	LD1H	91
D1.1.14	LD1Q	93
D1.1.15	LD1W	95
D1.1.16	LDR	97
D1.1.17	MOV (tile to vector)	98
D1.1.18	MOV (vector to tile)	101
D1.1.19	MOVA (tile to vector)	104
D1.1.20	MOVA (vector to tile)	107
D1.1.21	RDSVL	110
D1.1.22	SMOPA	111
D1.1.23	SMOPS	113
D1.1.24	ST1B	115
D1.1.25	ST1D	117
D1.1.26	ST1H	119
D1.1.27	ST1Q	121
D1.1.28	ST1W	123
D1.1.29	STR	125
D1.1.30	SUMOPA	126
D1.1.31	SUMOPS	128
D1.1.32	UMOPA	130
D1.1.33	UMOPS	132
D1.1.34	USMOPA	134
D1.1.35	USMOPS	136

	D1.1.36	ZERO	138
D1.2		Base A64 instructions	140
	D1.2.1	MSR (immediate)	140
	D1.2.2	SMSTART	144
	D1.2.3	SMSTOP	146
D1.3		SVE2 instructions	148
	D1.3.1	PSEL	148
	D1.3.2	REVD	150
	D1.3.3	SCLAMP	152
	D1.3.4	UCLAMP	154

Part E Appendices

Chapter E1

Instructions affected by SME

E1.1	Illegal instructions in Streaming SVE mode	158
	E1.1.1 Illegal Advanced SIMD instructions	158
	E1.1.2 Illegal SVE instructions	166
E1.2	Unimplemented SVE instructions	170
E1.3	Reduced performance in Streaming SVE mode	171
	E1.3.1 Scalar floating-point instructions	171
	E1.3.2 SVE instructions	171

Chapter E2

SME Shared pseudocode

E2.1	AArch64.CheckFPAdvSIMDEnabled	173
E2.2	BFDotAdd	173
E2.3	CheckFPAdvSIMDEnabled64	174
E2.4	CheckNonStreamingSVEEnabled	174
E2.5	CheckSMEAccess	174
E2.6	CheckSMEAndZEnabled	174
E2.7	CheckSMEEnabled	175
E2.8	CheckStreamingSVEAndZEnabled	175
E2.9	CheckStreamingSVEEnabled	175
E2.10	FPDot	175
E2.11	FPDotAdd_ZA	177
E2.12	FPMulAdd_ZA	177
E2.13	FPPProcessDenorms4	177
E2.14	FPPProcessNaNs4	177
E2.15	HaveEBF16	178
E2.16	HaveSME	178
E2.17	HaveSMEF64F64	178
E2.18	HaveSMEI16I64	178
E2.19	ImplementedSMEVectorLength	178
E2.20	InStreamingMode	179
E2.21	IsFullA64Enabled	179
E2.22	IsMerging	179
E2.23	IsNormalSVEEnabled	179
E2.24	IsStreamingSVEEnabled	180
E2.25	IsSVEEnabled	180
E2.26	MaybeZeroSVEUppers	181
E2.27	NVL	181
E2.28	ResetSMEState	182
E2.29	ResetSVESState	182
E2.30	SetPSTATE_SM	182
E2.31	SetPSTATE_SVCR	182

E2.32	SetPSTATE_ZA	182
E2.33	SMEAccessTrap	183
E2.34	SVL	183
E2.35	VL	183
E2.36	ZAhslice	183
E2.37	ZAslice	184
E2.38	ZAtile	184
E2.39	ZAvector	185
E2.40	ZAvslice	185

Chapter E3

System registers affected by SME

E3.1	SME-Specific System registers	187
E3.1.1	ID_AA64SMFR0_EL1, SME Feature ID register 0	188
E3.1.2	MPAMSM_EL1, MPAM Streaming Mode Register	192
E3.1.3	SMCR_EL1, SME Control Register (EL1)	195
E3.1.4	SMCR_EL2, SME Control Register (EL2)	200
E3.1.5	SMCR_EL3, SME Control Register (EL3)	205
E3.1.6	SMIDR_EL1, Streaming Mode Identification Register	208
E3.1.7	SMPRI_EL1, Streaming Mode Priority Register	211
E3.1.8	SMPRMAP_EL2, Streaming Mode Priority Mapping Register	214
E3.1.9	SVCR, Streaming Vector Control Register	219
E3.1.10	TPIDR2_EL0, EL0 Read/Write Software Thread ID Register 2	224
E3.1.11	EDHSR, External Debug Halt Status Register	227
E3.2	Changes to existing System registers	230
E3.2.1	CPACR_EL1, Architectural Feature Access Control Register	231
E3.2.2	CPTR_EL2, Architectural Feature Trap Register (EL2)	237
E3.2.3	CPTR_EL3, Architectural Feature Trap Register (EL3)	249
E3.2.4	FAR_EL1, Fault Address Register (EL1)	254
E3.2.5	FAR_EL2, Fault Address Register (EL2)	259
E3.2.6	FAR_EL3, Fault Address Register (EL3)	263
E3.2.7	FPCR, Floating-point Control Register	266
E3.2.8	HCRX_EL2, Extended Hypervisor Configuration Register	277
E3.2.9	HFGTR_EL2, Hypervisor Fine-Grained Read Trap Register	285
E3.2.10	HFGWTR_EL2, Hypervisor Fine-Grained Write Trap Register	310
E3.2.11	ID_AA64ISAR1_EL1, AArch64 Instruction Set Attribute Register 1	330
E3.2.12	ID_AA64PFR1_EL1, AArch64 Processor Feature Register 1	338
E3.2.13	ID_AA64ZFR0_EL1, SVE Feature ID register 0	344
E3.2.14	MPAM2_EL2, MPAM2 Register (EL2)	349
E3.2.15	SCR_EL3, Secure Configuration Register	356
E3.2.16	SCTLR_EL1, System Control Register (EL1)	375
E3.2.17	SCTLR_EL2, System Control Register (EL2)	405
E3.2.18	EDDEVID1, External Debug Device ID register 1	438

Chapter E4

Glossary terms

Preface

About this supplement

$\mathbb{I}_{\text{RFSSZ}}$ This supplement is the *Arm® Architecture Reference Manual Supplement, The Scalable Matrix Extension (SME), for Armv9-A*.

$\mathbb{I}_{\text{LZCVC}}$ This supplement describes the changes and additions introduced by SME to the Armv9-A architecture.

$\mathbb{I}_{\text{CDMPR}}$ For SME, this supplement is to be read with the following documents:

- *Arm® Architecture Reference Manual for A-profile architecture* [\[1\]](#)
- *Arm® Architecture Registers Armv9, for Armv9-A architecture profile* [\[2\]](#)
- *Arm® A64 Instruction Set Architecture Armv9, for Armv9-A architecture profile* [\[3\]](#)

Together, the supplement and these documents provide a full description of the Armv9-A Scalable Matrix Extension.

This supplement is organized into parts:

- SME Application level programmers' model

Describes how the PE at an application level is altered by the implementation of SME.

- SME System level programmers' model

Describes how the PE at a system level is altered by the implementation of SME.

- SME instruction set

Describes the extensions made for SME to the A64 instruction set.

- Appendices

Provides reference information relating to the SME. This includes summarized information about the instruction set, imported shared pseudocode and System register data, and a glossary that defines terms used in this document that have a specialized meaning.

Conventions

Typographical conventions

The typographical conventions are:

italic

Introduces special terminology, and denotes citations.

bold

Denotes signal names, and is used for terms in descriptive lists, where appropriate.

`monospace`

Used for assembler syntax descriptions, pseudocode, and source code examples.

Also used in the main text for instruction mnemonics and for references to other items appearing in assembler syntax descriptions, pseudocode, and source code examples.

SMALL CAPITALS

Used for some common terms such as IMPLEMENTATION DEFINED.

Used for a few terms that have specific technical meanings, and are included in the Glossary.

Blue text

Indicates a link. This can be

- A cross-reference to another location within the document
- A URL, for example <http://developer.arm.com>

Numbers

Numbers are normally written in decimal. Binary numbers are preceded by `0b`, and hexadecimal numbers by `0x`. In both cases, the prefix and the associated value are written in a monospace font, for example `0xFFFF0000`. To improve readability, long numbers can be written with an underscore separator between every four characters, for example `0xFFFF_0000_0000_0000`. Ignore any underscores when interpreting the value of a number.

Pseudocode descriptions

This book uses a form of pseudocode to provide precise descriptions of the specified functionality. This pseudocode is written in a monospace font. The pseudocode language is described in the Arm Architecture Reference Manual.

Asterisks in instruction mnemonics

Some behavior descriptions in this manual apply to a group of similar instructions that start with the same characters. In these situations, an `*` might be inserted at the end of a series of characters as a wildcard.

Assembler syntax descriptions

This book contains numerous syntax descriptions for assembler instructions and for components of assembler instructions. These are shown in a `monospace` font.

Rules-based writing

This specification consists of a set of individual rules. Each rule is clearly identified by the letter R.

Rules must not be read in isolation, and where more than one rule relating to a particular feature exists, individual rules are grouped into sections and subsections to provide the proper context. Where appropriate, these sections contain a short introduction to aid the reader. An implementation which is compliant with the architecture must conform to all of the rules in this specification.

Some architecture rules are accompanied by rationale statements which explain why the architecture was specified as it was. Rationale statements are identified by the letter X.

Some sections contain additional information and guidance that do not constitute rules. This information and guidance is provided purely as an aid to understanding the architecture. Information statements are clearly identified by the letter I.

Implementation notes are identified by the letter U.

Software usage descriptions are identified by the letter S.

Arm strongly recommends that implementers read *all* chapters and sections of this document to ensure that an implementation is compliant.

Rules, rationale statements, information statements, implementation notes and software usage statements are collectively referred to as *content items*.

Identifiers

Each content item may have an associated identifier which is unique within the context of this specification.

When the document is prior to beta status:

- Content items are assigned numerical identifiers, in ascending order through the document (*0001, 0002, ...*).
- Identifiers are volatile: the identifier for a given content item may change between versions of the document.

After the document reaches beta status:

- Content items are assigned random alphabetical identifiers (*HJQS, PZWL, ...*).
- Identifiers are preserved: a given content item has the same identifier across versions of the document.

Examples

Below are examples showing the appearance of each type of content item.

R_{BNPWG}

This is a rule statement.

I_{RSPLN}

This is an information statement.

D_{DDYRY}

This is a term, syntax, data structure, or encoding description.

G_{PBGWT}

This is a goal statement.

Additional reading

This section lists publications by Arm and by third parties.

See Arm Developer (<http://developer.arm.com>) for access to Arm documentation.

- [1] *Arm® Architecture Reference Manual for A-profile architecture*. (ARM DDI 0487) Arm Ltd.
- [2] *Arm® Architecture Registers Armv9, for Armv9-A architecture profile*. (ARM DDI 0601) Arm Ltd.
- [3] *Arm® A64 Instruction Set Architecture Armv9, for Armv9-A architecture profile*. (ARM DDI 0602) Arm Ltd.
- [4] *Arm® Architecture Reference Manual Supplement Armv9, for A-profile architecture*. (ARM DDI 0608) Arm Ltd.
- [5] *Arm® Reliability, Availability, and Serviceability (RAS) Specification, for A-profile architecture*. (ARM DDI 0587) Arm Ltd.
- [6] *Arm® Architecture Reference Manual Supplement, Memory System Resource Partitioning and Monitoring (MPAM), for A-profile architecture*. (ARM DDI 0598) Arm Ltd.

Feedback

Arm welcomes feedback on its documentation.

Feedback on this book

If you have comments on the content of this book, send an e-mail to errata@arm.com. Give:

- The title (Arm® Architecture Reference Manual Supplement, The Scalable Matrix Extension (SME), for Armv9-A).
- The number (DDI0616 A.a).
- The page numbers to which your comments apply.
- The rule identifiers to which your comments apply, if applicable.
- A concise explanation of your comments.

Arm also welcomes general suggestions for additions and improvements.

Note

Arm tests PDFs only in Adobe Acrobat and Acrobat Reader, and cannot guarantee the appearance or behavior of any document when viewed with any other PDF reader.

Progressive terminology commitment

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

We believe that this document contains no offensive terms. If you find offensive terms in this document, please contact terms@arm.com.

Part A

Introduction

Chapter A1

SME Introduction

A1.1 About the Scalable Matrix Extension (SME)

I_{YVCPK}

The Scalable Matrix Extension (SME) defines architectural state capable of holding two-dimensional matrix tiles, and a Streaming SVE mode which supports execution of SVE2 instructions with a vector length that matches the tile width, along with instructions that accumulate the outer product of two vectors into a tile, as well as load, store, and move instructions that transfer a vector to or from a tile row or column. The extension also defines System registers and fields that identify the presence and capabilities of SME, and enable and control its behavior at each Exception level.

I_{SQCB}

Unless otherwise specified by this document, the behaviors of instructions and architectural state when the PE is in *Streaming SVE mode* are as described in *Arm® Architecture Reference Manual for A-profile architecture* [1].

Chapter A2

Architecture Features and Extensions

A2.1 Extensions and features defined by SME

The Scalable Matrix Extension (SME) inherits the rules for architectural features and extensions from Armv9-A *Arm® Architecture Reference Manual for A-profile architecture* [1]. This specification describes changes to those rules, and defines any features added by SME.

R_{PDXHJ} An architecture extension, the Scalable Matrix Extension (SME), is introduced. SME is represented by the feature FEAT_SME.

R_{QFSVK} SME is an OPTIONAL extension from Armv9.2-A.

I_{VQCZZ} The following list summarizes the OPTIONAL SME features:

- FEAT_SME_FA64: Support the full A64 instruction set in *Streaming SVE mode*.
- FEAT_SME_F64F64: Double-precision floating-point outer product instructions.
- FEAT_SME_I16I64: 16-bit to 64-bit integer widening outer product instructions.
- FEAT_EBF16: Support for Extended BFloat16 mode.

A2.2 Changes to existing features and extension requirements

R_{DSHWS}

If SME is implemented, the following features that are not already mandatory in Armv9.1 are also implemented:

- FEAT_HCX.

Part B
SME application level programmers' model

Chapter B1

Application processing modes

B1.1 Overview

SME extends the AArch64 application level programmers' model with added processing modes and related instructions, architectural state, and registers:

- The `PSTATE.SM` control to enable an execution mode, known as *Streaming SVE mode*.
- The `PSTATE.ZA` control to enable access to SME ZA storage.
- The Special-purpose register, `SVCR`, which provides read/write access to `PSTATE.SM` and `PSTATE.ZA` from any Exception level.
- The `SMSTART` and `SMSTOP` instructions, aliases of `MSR` (immediate) instructions, that can set or clear `PSTATE.SM`, `PSTATE.ZA`, or both `PSTATE.SM` and `PSTATE.ZA` from any Exception level.

B1.1.1 Process state

<code>D_{XDPXS}</code>	A PE that implements SME has a <i>Streaming SVE mode</i> .
<code>D_{JYVLM}</code>	Streaming SVE register state is the vector registers <i>Z0-Z31</i> and predicate registers <i>P0-P15</i> that may be accessed by SME, SVE, Advanced SIMD, and floating-point instructions when the PE is in <i>Streaming SVE mode</i> .
<code>D_{DMZFR}</code>	Streaming SVE register state includes the SVE <i>FFR</i> predicate register if <code>FEAT_SME_FA64</code> is implemented and enabled at the current Exception level.
<code>I_{XXKGV}</code>	If SME is implemented, a PE has the following additional architectural state: <ul style="list-style-type: none">• Streaming SVE vector and predicate register state.• SME ZA storage.

I_ZTTNW	A PE enters <i>Streaming SVE mode</i> to access Streaming SVE vector and predicate register state.
I_NXQFB	If SME is implemented, this does not imply that FEAT_SVE and FEAT_SVE2 are implemented by the PE when it is not in <i>Streaming SVE mode</i> .
I_RNTPX	When the PE is in <i>Streaming SVE mode</i> , a different set of vector lengths might be available for SVE instructions, as specified in C2.1.3 Vector lengths .
I_TDSPN	When the PE is in <i>Streaming SVE mode</i> , the performance characteristics of some instructions may be significantly reduced, as specified in E1.3 Reduced performance in Streaming SVE mode .
I_NWFQX	SME extends a PE's <i>Process state</i> or <code>PSTATE</code> with the <code>SM</code> and <code>ZA</code> fields, which control the execution mode, <i>Streaming SVE mode</i> , and access to SME ZA storage, respectively. These <code>PSTATE</code> fields can be modified by the <code>SMSTART</code> and <code>SMSTOP</code> instructions, and can also be read and written using the <code>SVCR</code> register.
I_DVDDL	The <code>SMSTART</code> instruction is used to enter <i>Streaming SVE mode</i> , or to enable the SME ZA storage, or both simultaneously.
I_QQZTL	The <code>SMSTOP</code> instruction is used to exit <i>Streaming SVE mode</i> , or to disable the SME ZA storage, or both simultaneously.
I_NKJKL	After entering <i>Streaming SVE mode</i> , subsequent <code>SMSTART</code> and <code>SMSTOP</code> instructions might be used to enable and disable the ZA storage for different phases of execution within <i>Streaming SVE mode</i> , before using a final <code>SMSTOP</code> instruction to exit <i>Streaming SVE mode</i> .
D_RSYPB	SME instructions are the instructions defined by the SME architecture in Chapter D1 SME instructions .
D_LSJBN	SME data-processing instructions are the instructions defined in D1.1 SME data-processing instructions .
D_NHNF	A <i>legal</i> instruction is an implemented instruction which can be executed by a PE when <code>PSTATE.SM</code> and <code>PSTATE.ZA</code> are in the required state, unless its execution at this Exception level is prevented by a configurable trap or enable.
D_HZFSG	An <i>illegal</i> instruction is an implemented instruction whose attempted execution by a PE when <code>PSTATE.SM</code> and <code>PSTATE.ZA</code> are not in the required state will cause an SME illegal instruction exception to be taken, unless its execution at this Exception level is prevented by a higher-priority configurable trap or enable.
I_PSNCC	When the PE is in <i>Streaming SVE mode</i> : <ul style="list-style-type: none"> • SME data-processing instructions that do not access the SME ZA storage are <i>legal</i>. • SME data-processing instructions that access SME ZA storage are <i>legal</i>, if SME ZA storage is enabled.
I_CKSBS	When the PE is in <i>Streaming SVE mode</i> and FEAT_SME_FA64 is not implemented or not enabled at the current Exception level: <ul style="list-style-type: none"> • Most Advanced SIMD instructions become <i>illegal</i>, as described in E1.1.1 Illegal Advanced SIMD instructions. • Some SVE and SVE2 instructions become <i>illegal</i>, as described in E1.1.2 Illegal SVE instructions. • Most other instructions implemented by the PE, including scalar floating-point instructions, remain <i>legal</i>.
I_DGNQM	When the PE is not in <i>Streaming SVE mode</i> : <ul style="list-style-type: none"> • SME data-processing instructions that access SVE vector registers <code>Z0-Z31</code> and predicate registers <code>P0-P15</code> are <i>illegal</i>. • SME <code>LDR</code>, <code>STR</code>, and <code>ZERO</code> instructions that access the SME ZA storage are <i>legal</i> if ZA is enabled. • The <code>MSR</code> and <code>MRS</code> instructions that directly access the SME <code>SVCR</code> register are <i>legal</i>. • All other instructions implemented by the PE are <i>legal</i>.
I_HDWMG	When the SME ZA storage is not enabled: <ul style="list-style-type: none"> • SME data-processing instructions that access SME ZA storage are <i>illegal</i>. • There is no effect on other instructions implemented by the PE. <p>See also:</p> <ul style="list-style-type: none"> • SVCR • ESR_EL1, ESR_EL2, and ESR_EL3

- [MSR \(immediate\)](#)
- [SMSTART](#)
- [SMSTOP](#)
- [C2.1.3 Vector lengths](#)
- [C2.2.3 Validity of SME and SVE state](#)
- [Chapter E1 Instructions affected by SME](#)

B1.1.1.1 PSTATE.SM

I _{YYQJK}	The value of <code>PSTATE.SM</code> can be changed by executing the <code>MSR</code> instructions that access the <code>SVCR</code> . For more information, see B1.1.1.3 Changing PSTATE.SM and PSTATE.ZA .
D _{PRGHY}	The PE is in <i>Streaming SVE mode</i> when the Effective value of <code>PSTATE.SM</code> is 1.
R _{QXCF}	When the PE is in <i>Streaming SVE mode</i> : <ul style="list-style-type: none"> • Streaming SVE register state is valid. • SME data-processing instructions which access any of the SVE registers <i>Z0-Z31</i> and <i>P0-P15</i> are <i>legal</i>. • <i>Legal</i> instructions which access SVE or SIMD&FP registers will access Streaming SVE register state.
I _{YDRPH}	The SVE <i>FFR</i> predicate register is not architecturally visible when the PE is in <i>Streaming SVE mode</i> if <code>FEAT_SME_FA64</code> is not implemented or not enabled at the current Exception level.
R _{GBNWK}	When the PE is in <i>Streaming SVE mode</i> and <code>FEAT_SME_FA64</code> is not implemented or not enabled at the current Exception level: <ul style="list-style-type: none"> • Most Advanced SIMD instructions are <i>illegal</i>, as described in E1.1.1 Illegal Advanced SIMD instructions. • Some SVE and SVE2 instructions are <i>illegal</i>, as described in E1.1.2 Illegal SVE instructions.
D _{DVDTY}	The PE is not in <i>Streaming SVE mode</i> when the Effective value of <code>PSTATE.SM</code> is 0.
R _{CSSWX}	When the PE is not in <i>Streaming SVE mode</i> : <ul style="list-style-type: none"> • Streaming SVE register state is not valid. • Instructions which access SVE or SIMD&FP registers will access the Non-streaming SVE or SIMD&FP register state. • SME data-processing instructions which access any of the SVE registers <i>Z0-Z31</i> and <i>P0-P15</i> are <i>illegal</i>.
R _{RSWFQ}	When the Effective value of <code>PSTATE.SM</code> is changed by any means from 0 to 1, an entry to <i>Streaming SVE mode</i> is performed, and each implemented bit of SVE registers <i>Z0-Z31</i> , <i>P0-P15</i> , and <i>FFR</i> in the new mode is set to zero.
R _{KFRQZ}	When the Effective value of <code>PSTATE.SM</code> is changed by any means from 1 to 0, an exit from <i>Streaming SVE mode</i> is performed, and each implemented bit of SVE registers <i>Z0-Z31</i> , <i>P0-P15</i> , and <i>FFR</i> in the new mode is set to zero.
R _{MHTLZ}	When the Effective value of <code>PSTATE.SM</code> is changed by any means from 0 to 1, or from 1 to 0, the <code>FPSR</code> is set to the value <code>0x0000_0000_0800_009f</code> , in which all of the cumulative status bits are set to 1.
I _{YTZVD}	Statements which refer to the value of the SVE vector registers, <i>Z0-Z31</i> , implicitly also refer to the lower bits of those registers accessed by the SIMD&FP register names <i>V0-V31</i> , <i>Q0-Q31</i> , <i>D0-D31</i> , <i>S0-S31</i> , <i>H0-H31</i> , and <i>B0-B31</i> . See also: <ul style="list-style-type: none"> • SVCR • C2.1.2 Traps and exceptions

B1.1.1.2 PSTATE.ZA

I _{GJZLD}	The value of <code>PSTATE.ZA</code> can be changed by executing the <code>MSR</code> instructions that access the <code>SVCR</code> . For more information, see B1.1.1.3 Changing PSTATE.SM and PSTATE.ZA .
D _{HBFWD}	The SME ZA storage is enabled when <code>PSTATE.ZA</code> is 1.

R _{SFWMY}	When ZA storage is enabled: <ul style="list-style-type: none"> The contents of ZA storage is valid and will be retained by hardware irrespective of whether the PE is in <i>Streaming SVE mode</i>. SME data-processing instructions which access the ZA storage are <i>legal</i> and may be executed, unless execution is prevented by some other trap or exception.
D _{VLMFC}	The SME ZA storage is disabled when PSTATE.ZA is 0.
R _{JHMYL}	When ZA storage is disabled: <ul style="list-style-type: none"> The contents of ZA storage is not valid. SME data-processing instructions which access the ZA storage are <i>illegal</i>.
R _{YRZRM}	When PSTATE.ZA is changed by any means from 0 to 1, all implemented bits of the SME ZA storage are set to zero.
I _{LRDZR}	There is no architecturally defined effect on the SME ZA storage when PSTATE.ZA is changed from 1 to 0, because the contents of ZA storage cannot be observed when PSTATE.ZA is 0.
I _{QWCJS}	When PSTATE.ZA is changed from 0 to 1, or 1 to 0, there is no effect on the SVE vector and predicate registers and the FPSR if PSTATE.SM is not changed.
	See also: <ul style="list-style-type: none"> SVCR C2.1.2 Traps and exceptions

B1.1.1.3 Changing PSTATE.SM and PSTATE.ZA

D _{QRSXV}	The MSR (immediate) instructions, MSR SVCRSM, #<imm1>, MSR SVCRZA, #<imm1>, and MSR SVCRSMZA, #<imm1>, are provided to independently set or clear PSTATE.SM, PSTATE.ZA, or both PSTATE.SM and PSTATE.ZA respectively.
R _{MPQWY}	MSR SVCRSM, MSR SVCRZA, and MSR SVCRSMZA instructions are permitted to be executed from any Exception level.
I _{SYZBC}	Access to SVCR through the MRS and MSR (register) instructions might be used where a calling convention or ABI requires save/restore of current state, and are permitted to be executed from any Exception level. However, the MSR (immediate) instructions might be higher performance than the MSR (register) instruction, so the MSR (immediate) instructions should be preferred for explicit changes to PSTATE.SM and PSTATE.ZA.
D _{YGDXX}	The SMSTART instruction is the preferred alias of the MSR SVCRSM, #1, MSR SVCRZA, #1, and MSR SVCRSMZA, #1 instructions.
D _{DZTDH}	The SMSTOP instruction is the preferred alias of the MSR SVCRSM, #0, MSR SVCRZA, #0, and MSR SVCRSMZA, #0 instructions.
I _{HNNJR}	The PE might consume less power when PSTATE.SM is 0 and PSTATE.ZA is 0.
	See also: <ul style="list-style-type: none"> SVCR MSR (immediate) SMSTART SMSTOP

Chapter B2

Architectural state

B2.1 Architectural state summary

D_{XJCGQ}	The Effective Streaming SVE vector length, SVL, is a power of two in the range 128 to 2048 bits inclusive.
I_{NBPPM}	When the PE is in <i>Streaming SVE mode</i> , the Effective SVE vector length, VL, is equal to SVL. This might be different from the value of VL when the PE is not in <i>Streaming SVE mode</i> , as described in C2.1.3 Vector lengths .
D_{JBVYJ}	In a vector of SVL bits: <ul style="list-style-type: none">• SVL_B is the number of 8-bit elements.• SVL_H is the number of 16-bit elements.• SVL_S is the number of 32-bit elements.• SVL_D is the number of 64-bit elements.• SVL_Q is the number of 128-bit elements.

SVL [bits]	SVL_B	SVL_H	SVL_S	SVL_D	SVL_Q
128	16	8	4	2	1
256	32	16	8	4	2
512	64	32	16	8	4
1024	128	64	32	16	8
2048	256	128	64	32	16

See also:

- [Chapter B1 *Application processing modes*](#)
- [C2.1.3 *Vector lengths*](#).

B2.2 SME ZA storage

D_{SSXP}L The ZA storage is architectural register state consisting of a two-dimensional ZA array of $[SVL_B \times SVL_B]$ bytes.

B2.2.1 ZA array vector access

R_{FFWN}B The ZA array can be accessed as vectors of SVL bits.

D_{PPPC}M An untyped vector access to the ZA array is represented by $ZA[N]$, where N is in the range 0 to SVL_B-1 inclusive.

D_{DTVZ}N In SME **LDR** and **STR** instructions an untyped ZA array vector is selected by the sum of a 32-bit general-purpose register (vector select register W_v) and an immediate, modulo SVL_B .

D_{YXH}F The preferred disassembly for an untyped ZA array vector is $ZA[W_v, imm]$, where imm is in the range 0 to 15 inclusive.

D_{CRJ}P The ZA array can be accessed as vectors of 8-bit, 16-bit, 32-bit, 64-bit, or 128-bit elements.

D_{WMV}T An element-wise vector access to the ZA array is indicated by appending a vector index “[N]” to the ZA array name and element size qualifier, where N is in the range 0 to SVL_B-1 inclusive, as follows:

- An 8-bit element vector access to the ZA array is represented by $ZA.B[N]$.
- A 16-bit element vector access to the ZA array is represented by $ZA.H[N]$.
- A 32-bit element vector access to the ZA array is represented by $ZA.S[N]$.
- A 64-bit element vector access to the ZA array is represented by $ZA.D[N]$.
- A 128-bit element vector access to the ZA array is represented by $ZA.Q[N]$.

B2.2.2 ZA tile access

D_{VSM}X A ZA tile is a square, two-dimensional sub-array of elements within the ZA array.

I_{WLR}T Depending on the element size with which it is accessed, the ZA array is treated as containing one or more ZA tiles, as described in the following sections.

D_{DWM}Y A ZA tile is indicated by appending the tile number to the ZA name.

D_{ZGB}H A ZA tile slice is a one-dimensional set of horizontally or vertically contiguous elements within a ZA tile.

R_{PZN}W A vector access to a tile reads or writes a ZA tile slice.

I_{NFX}H A ZA tile can be accessed as vectors of 8-bit, 16-bit, 32-bit, 64-bit, or 128-bit elements.

I_{YZD}B A ZA tile can be accessed as horizontal slices of SVL bits.

R_{GPV}S A ZA tile is accessed as horizontal slices if the V field in the accessing instruction opcode is 0.

D_{TRH}T An access to horizontal tile slices is indicated by an “H” suffix on the ZA tile name.

I_{HBY}T A ZA tile can be accessed as vertical slices of SVL bits.

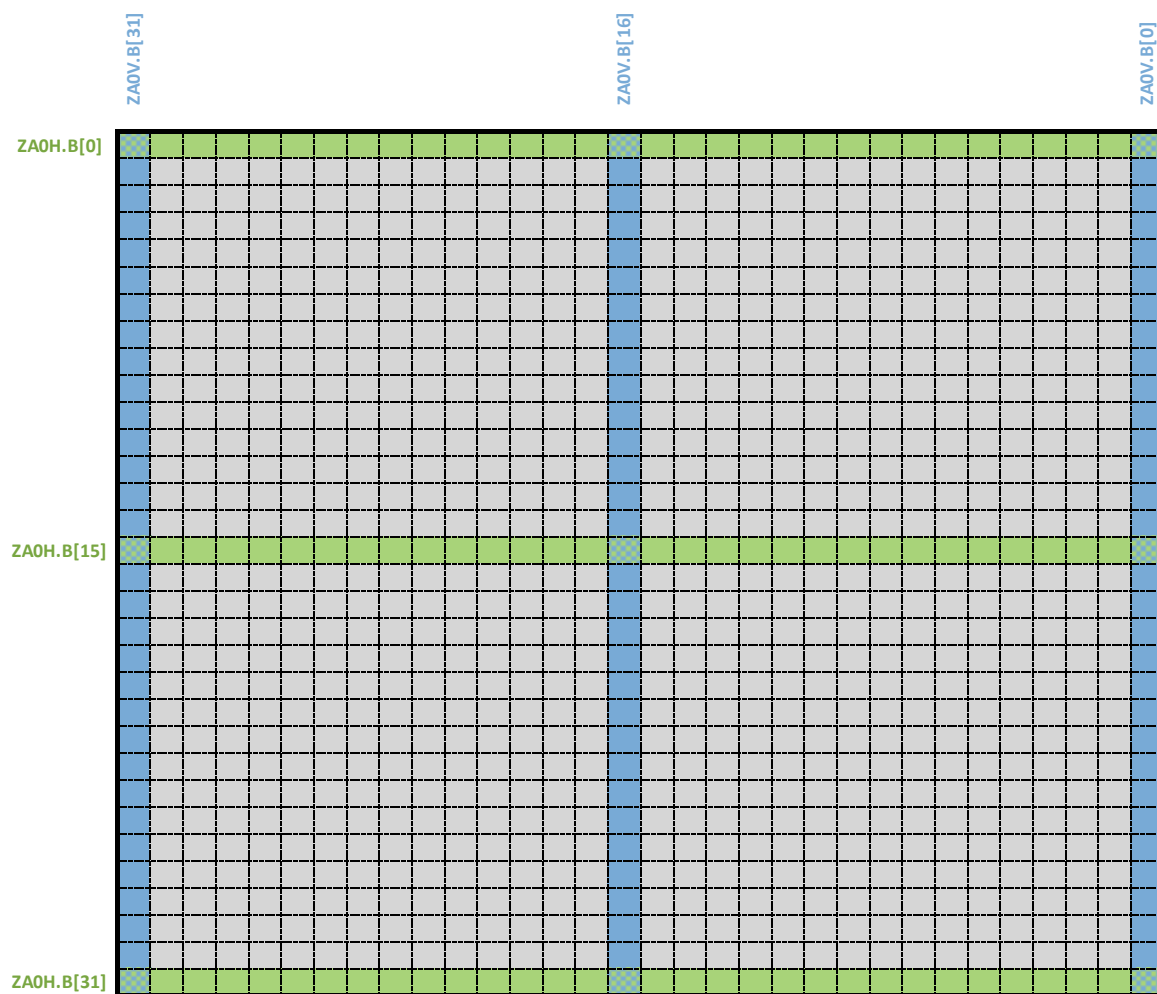
R_{GPP}K A ZA tile is accessed as vertical slices if the V field in the accessing instruction opcode is 1.

D_{WSB}V An access to vertical tile slices is indicated by a “V” suffix on the ZA tile name.

R_{TWW}T In SME instructions the tile slice is selected by the sum of a 32-bit general-purpose register (slice index register W_s) and an immediate, modulo the number of slices in the named tile.

B2.2.3 Accessing an 8-bit element ZA tile

D_{HMSNH}	An 8-bit element ZA tile is indicated by a “.B” qualifier following the tile name.
D_{NLCNH}	There is a single tile named ZA0.B which consists of $[SVL_B \times SVL_B]$ 8-bit elements and occupies all of the ZA storage.
R_{NBSMJ}	An access to a horizontal or vertical 8-bit element ZA tile slice reads or writes SVL_B 8-bit elements.
D_{NMHLM}	An access to a horizontal or vertical 8-bit element ZA tile slice is indicated by appending a slice index “[N]” to the tile name, direction suffix, and qualifier, ZA0H.B[N] or ZA0V.B[N], where N is in the range 0 to SVL_B-1 inclusive.
I_{JVTNY}	Horizontal and vertical ZA0.B slice accesses are illustrated in the following diagram for SVL of 256 bits:

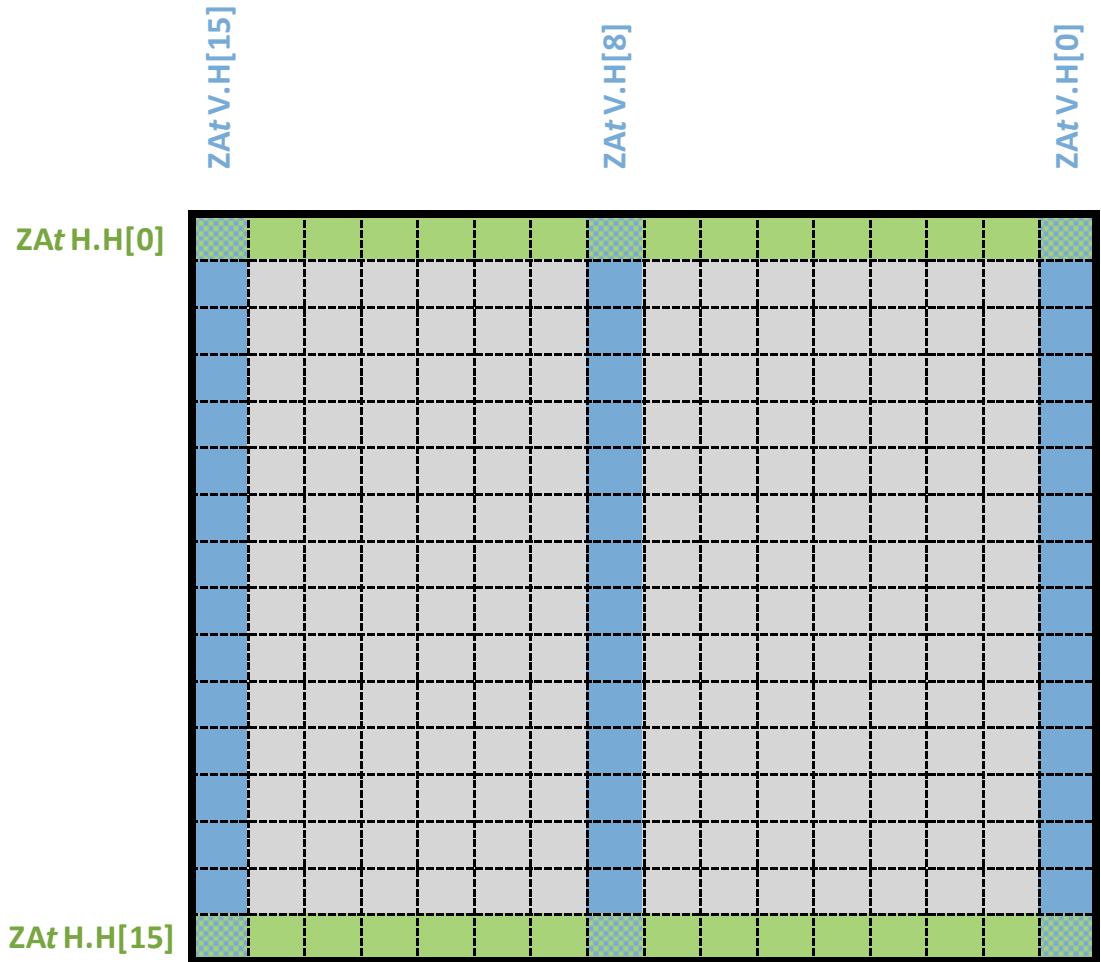


R_{DCSDX}	An access to the horizontal slice ZA0H.B[N] reads or writes the SVL_B bytes in ZA array vector ZA.B[N].
R_{FHYSQ}	An access to the vertical slice ZA0V.B[N] reads or writes the 8-bit element [N] within each horizontal slice of ZA0.B.
D_{CDDVV}	The preferred disassembly is: <ul style="list-style-type: none"> • ZA0H.B[Ws, imm], for a horizontal 8-bit element ZA tile slice selection. • ZA0V.B[Ws, imm], for a vertical 8-bit element ZA tile slice selection.

Where *imm* is in the range 0 to 15 inclusive.

B2.2.4 Accessing a 16-bit element ZA tile

D_{LNXPD}	A 16-bit element ZA tile is indicated by a “.H” qualifier following the tile name.
D_{GWZDM}	There are two tiles named ZA0.H and ZA1.H which each consists of $[SVL_H \times SVL_H]$ 16-bit elements. Each tile occupies half of the ZA storage.
R_{NMGXG}	An access to a horizontal or vertical 16-bit element ZA tile slice reads or writes SVL_H 16-bit elements.
D_{DHKMC}	An access to a horizontal or vertical 16-bit element ZA tile slice is indicated by appending a slice index “[<i>N</i>]” to the tile name, direction suffix, and qualifier, $ZA_tH.H[N]$ or $ZA_tV.H[N]$, where <i>t</i> is 0 or 1 and <i>N</i> is in the range 0 to SVL_H-1 inclusive.
I_{ZSWJW}	Horizontal and vertical ZA_tH slice accesses, where <i>t</i> is 0 or 1, are illustrated in the following diagram for SVL of 256 bits:



R_{BTLQC}	An access to the horizontal slice $ZA_tH.H[N]$ reads or writes the SVL_H 16-bit elements in ZA array vector $ZA.H[t + 2 * N]$.
R_{NGJBJ}	

An access to the vertical slice $ZAtV.H[N]$ reads or writes the 16-bit element $[N]$ within each horizontal slice of $ZAt.H$.

D_{RHQJT}

The preferred disassembly is:

- $ZAtH.H[Ws, imm]$, for a horizontal 16-bit element ZA tile slice selection.
- $ZAtV.H[Ws, imm]$, for a vertical 16-bit element ZA tile slice selection.

Where t is 0 or 1, and imm is in the range 0 to 7 inclusive.

B2.2.5 Accessing a 32-bit element ZA tile

D_{HBKZV}

A 32-bit element ZA tile is indicated by a “.S” qualifier following the tile name.

D_{RDRRT}

There are four tiles named $ZA0.S$, $ZA1.S$, $ZA2.S$, and $ZA3.S$. Each tile consists of $[SVL_S \times SVL_S]$ 32-bit elements. Each tile occupies quarter of the ZA storage.

R_{XFPPPL}

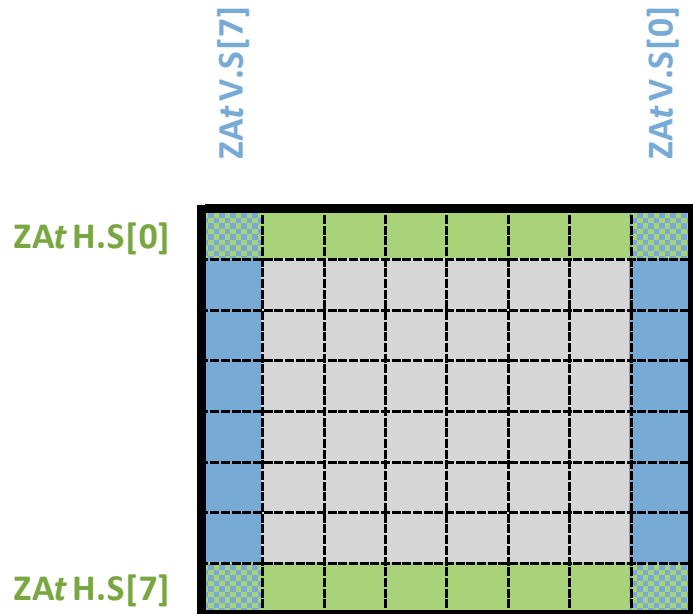
An access to a horizontal or vertical 32-bit element ZA tile slice reads or writes SVL_S 32-bit elements.

D_{JFPSJ}

An access to a horizontal or vertical 32-bit element ZA tile slice is indicated by appending a slice index “[N]” to the tile name, direction suffix, and qualifier, $ZAtH.S[N]$ or $ZAtV.S[N]$, where t is 0, 1, 2, or 3 and N is in the range 0 to SVL_S-1 inclusive.

I_{SZXZR}

Horizontal and vertical $ZAt.S$ slice accesses, where t is 0, 1, 2, or 3, are illustrated in the following diagram for SVL of 256 bits:



R_{JBJZY}

An access to the horizontal slice $ZAtH.S[N]$ reads or writes the SVL_S 32-bit elements in ZA array vector $ZA.S[t + 4 * N]$.

R_{GBYSJ}

An access to the vertical slice $ZAtV.S[N]$ reads or writes the 32-bit element $[N]$ within each horizontal slice of $ZAt.S$.

D_{LQLJH}

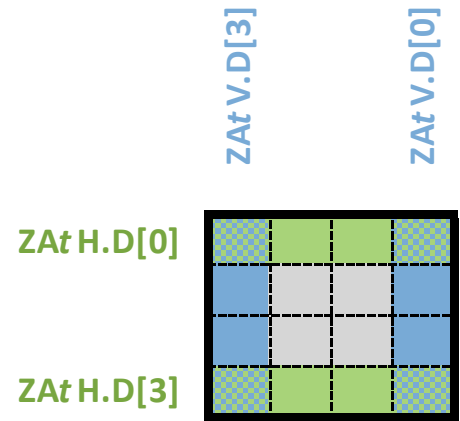
The preferred disassembly is:

- $ZAtH.S[Ws, imm]$, for a horizontal 32-bit element ZA tile slice selection.
- $ZAtV.S[Ws, imm]$, for a vertical 32-bit element ZA tile slice selection.

Where t is 0, 1, 2, or 3, and imm is 0, 1, 2, or 3.

B2.2.6 Accessing a 64-bit element ZA tile

D_{TWMM}	A 64-bit element ZA tile is indicated by a “.D” qualifier following the tile name.
D_{THPSD}	There are eight tiles named ZA0.D, ZA1.D, ZA2.D, ZA3.D, ZA4.D, ZA5.D, ZA6.D, and ZA7.D. Each tile consists of $[SVL_D \times SVL_D]$ 64-bit elements. Each tile occupies an eighth of the ZA storage.
R_{ZXYBQ}	An access to a horizontal or vertical 64-bit element ZA tile slice reads or writes SVL_D 64-bit elements.
D_{DCXSX}	An access to a horizontal or vertical 64-bit element ZA tile slice is indicated by appending a slice index “[N]” to the tile name, direction suffix, and qualifier, $ZA_tH.D[N]$ or $ZA_tV.D[N]$, where t is in the range 0 to 7 inclusive and N is in the range 0 to SVL_D-1 inclusive.
I_{LGJZC}	Horizontal and vertical $ZA_t.D$ slice accesses, where t is in the range 0 to 7 inclusive, are illustrated in the following diagram for SVL of 256 bits:



R_{CVVJK}	An access to the horizontal slice $ZA_tH.D[N]$ reads or writes the SVL_D 64-bit elements in ZA array vector $ZA.D[t + 8 * N]$.
R_{JYQKK}	An access to the vertical slice $ZA_tV.D[N]$ reads or writes the 64-bit element $[N]$ within each horizontal slice of $ZA_t.D$.
D_{MQQPX}	The preferred disassembly is: <ul style="list-style-type: none"> • $ZA_tH.D[Ws, imm]$, for a horizontal 64-bit element ZA tile slice selection. • $ZA_tV.D[Ws, imm]$, for a vertical 64-bit element ZA tile slice selection.

Where t is in the range 0 to 7 inclusive, and imm is 0 or 1.

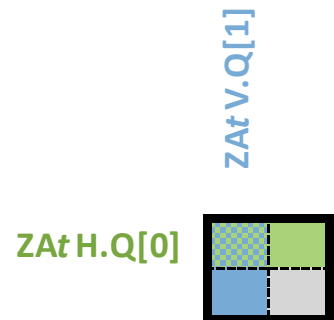
B2.2.7 Accessing a 128-bit element ZA tile

D_{GZDSH}	A 128-bit element ZA tile is indicated by a “.Q” qualifier following the tile name.
D_{RPMJL}	There are sixteen tiles named ZA0.Q, ZA1.Q, ZA2.Q, ZA3.Q, ZA4.Q, ZA5.Q, ZA6.Q, ZA7.Q, ZA8.Q, ZA9.Q, ZA10.Q, ZA11.Q, ZA12.Q, ZA13.Q, ZA14.Q, and ZA15.Q. Each tile consists of $[SVL_Q \times SVL_Q]$ 128-bit elements. Each tile occupies a sixteenth of the ZA storage.
R_{QGHPF}	An access to a horizontal or vertical 128-bit element ZA tile slice reads or writes SVL_Q 128-bit elements.
D_{RLQKW}	

An access to a horizontal or vertical 128-bit element ZA tile slice is indicated by appending a slice index “[N]” to the tile name, direction suffix, and qualifier, $ZAtH.Q[N]$ or $ZAtV.Q[N]$, where t is in the range 0 to 15 inclusive and N is in the range 0 to SVL_Q-1 inclusive.

I_{YQPWS}

Horizontal and vertical $ZAt.Q$ slice accesses, where t is in the range 0 to 15 inclusive, are illustrated in the following diagram for SVL of 256 bits:



R_{PJTQJ}

An access to the horizontal slice $ZAtH.Q[N]$ reads or writes the SVL_Q 128-bit elements in ZA array vector $ZA.Q[t + 16 * N]$.

R_{TRJFZ}

An access to the vertical slice $ZAtV.Q[N]$ reads or writes the 128-bit element $[N]$ within each horizontal slice of $ZAt.Q$.

D_{VCLJP}

The preferred disassembly is:

- $ZAtH.Q[Ws, 0]$, for a horizontal 128-bit element ZA tile slice selection.
- $ZAtV.Q[Ws, 0]$, for a vertical 128-bit element ZA tile slice selection.

Where t is in the range 0 to 15 inclusive, and the immediate offset is always zero.

B2.3 ZA storage layout

B2.3.1 ZA array vector and tile slice mappings

I_{PYTLW}

Each horizontal tile slice corresponds to one ZA array vector.

The horizontal slice mappings for all tile sizes are illustrated by this table:

ZA Array Vector	8-bit element Tile Horizontal Slice	16-bit element Tile Horizontal Slice	32-bit element Tile Horizontal Slice	64-bit element Tile Horizontal Slice	128-bit element Tile Horizontal Slice
ZA[0]	ZA0H.B[0]	ZA0H.H[0]	ZA0H.S[0]	ZA0H.D[0]	ZA0H.Q[0]
ZA[1]	ZA0H.B[1]	ZA1H.H[0]	ZA1H.S[0]	ZA1H.D[0]	ZA1H.Q[0]
ZA[2]	ZA0H.B[2]	ZA0H.H[1]	ZA2H.S[0]	ZA2H.D[0]	ZA2H.Q[0]
ZA[3]	ZA0H.B[3]	ZA1H.H[1]	ZA3H.S[0]	ZA3H.D[0]	ZA3H.Q[0]
ZA[4]	ZA0H.B[4]	ZA0H.H[2]	ZA0H.S[1]	ZA4H.D[0]	ZA4H.Q[0]
ZA[5]	ZA0H.B[5]	ZA1H.H[2]	ZA1H.S[1]	ZA5H.D[0]	ZA5H.Q[0]
ZA[6]	ZA0H.B[6]	ZA0H.H[3]	ZA2H.S[1]	ZA6H.D[0]	ZA6H.Q[0]
ZA[7]	ZA0H.B[7]	ZA1H.H[3]	ZA3H.S[1]	ZA7H.D[0]	ZA7H.Q[0]
ZA[8]	ZA0H.B[8]	ZA0H.H[4]	ZA0H.S[2]	ZA0H.D[1]	ZA8H.Q[0]
ZA[9]	ZA0H.B[9]	ZA1H.H[4]	ZA1H.S[2]	ZA1H.D[1]	ZA9H.Q[0]
ZA[10]	ZA0H.B[10]	ZA0H.H[5]	ZA2H.S[2]	ZA2H.D[1]	ZA10H.Q[0]
ZA[11]	ZA0H.B[11]	ZA1H.H[5]	ZA3H.S[2]	ZA3H.D[1]	ZA11H.Q[0]
ZA[12]	ZA0H.B[12]	ZA0H.H[6]	ZA0H.S[3]	ZA4H.D[1]	ZA12H.Q[0]
ZA[13]	ZA0H.B[13]	ZA1H.H[6]	ZA1H.S[3]	ZA5H.D[1]	ZA13H.Q[0]
ZA[14]	ZA0H.B[14]	ZA0H.H[7]	ZA2H.S[3]	ZA6H.D[1]	ZA14H.Q[0]
ZA[15]	ZA0H.B[15]	ZA1H.H[7]	ZA3H.S[3]	ZA7H.D[1]	ZA15H.Q[0]
if applicable ZA[16] to ZA[SVL _B -1]

B2.3.2 Tile mappings

I_{YVYJP}

The smallest ZA tile granule is the 128-bit element tile. When the ZA storage is viewed as an array of tiles, the larger 64-bit, 32-bit, 16-bit, and 8-bit element tiles overlap multiple 128-bit element tiles as follows:

Tile	Overlaps
ZA0.B	ZA0.Q, ZA1.Q, ZA2.Q, ZA3.Q, ZA4.Q, ZA5.Q, ZA6.Q, ZA7.Q, ZA8.Q, ZA9.Q, ZA10.Q, ZA11.Q, ZA12.Q, ZA13.Q, ZA14.Q, ZA15.Q
ZA0.H	ZA0.Q, ZA2.Q, ZA4.Q, ZA6.Q, ZA8.Q, ZA10.Q, ZA12.Q, ZA14.Q
ZA1.H	ZA1.Q, ZA3.Q, ZA5.Q, ZA7.Q, ZA9.Q, ZA11.Q, ZA13.Q, ZA15.Q
ZA0.S	ZA0.Q, ZA4.Q, ZA8.Q, ZA12.Q

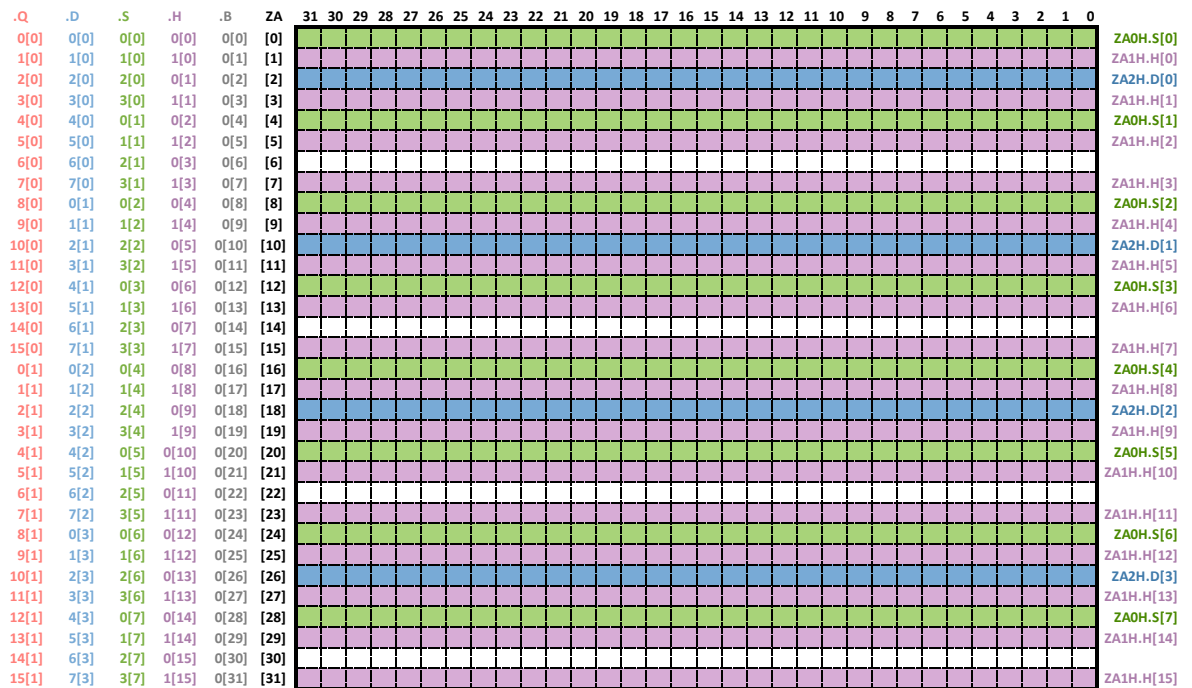
Tile	Overlaps
ZA1.S	ZA1.Q, ZA5.Q, ZA9.Q, ZA13.Q
ZA2.S	ZA2.Q, ZA6.Q, ZA10.Q, ZA14.Q
ZA3.S	ZA3.Q, ZA7.Q, ZA11.Q, ZA15.Q
ZA0.D	ZA0.Q, ZA8.Q
ZA1.D	ZA1.Q, ZA9.Q
ZA2.D	ZA2.Q, ZA10.Q
ZA3.D	ZA3.Q, ZA11.Q
ZA4.D	ZA4.Q, ZA12.Q
ZA5.D	ZA5.Q, ZA13.Q
ZA6.D	ZA6.Q, ZA14.Q
ZA7.D	ZA7.Q, ZA15.Q

I_{WGZBT}

The architecture permits concurrent use of different element size tiles.

I_{HVFMB}

It is possible to simultaneously use non-overlapping ZA array vectors within tiles of differing element sizes. For example, tiles ZA1.H, ZA0.S, and ZA2.D have no ZA array vectors in common, as illustrated in the following diagram for SVL of 256 bits:



I_{WDMCK}

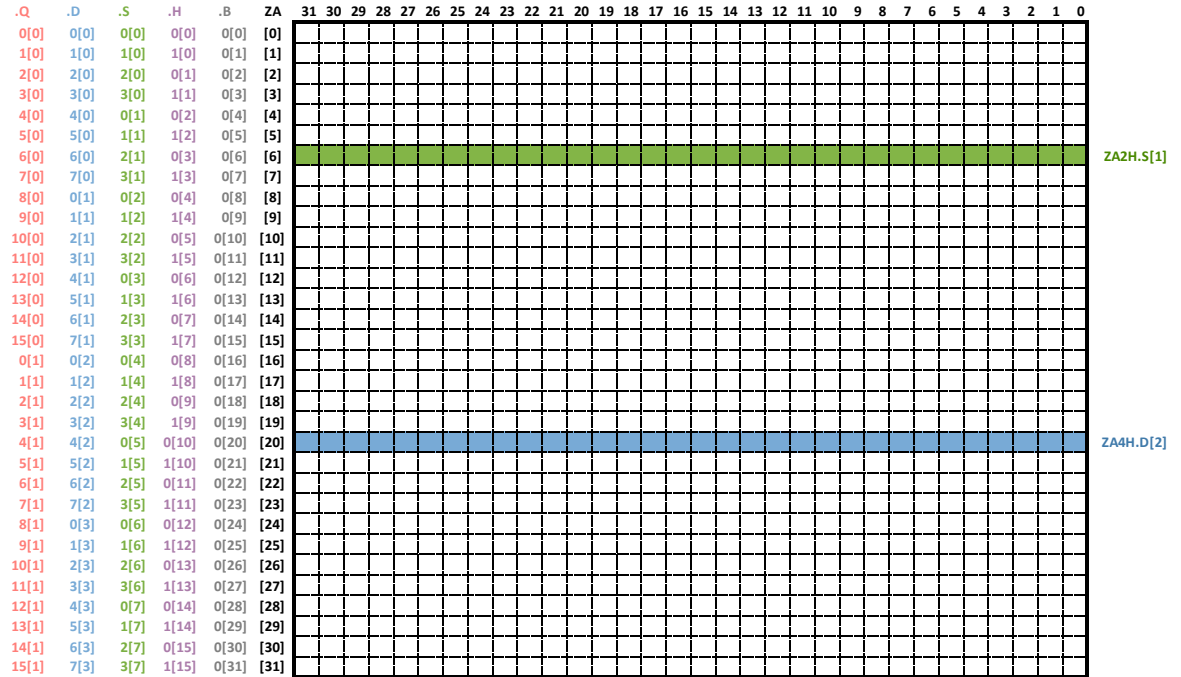
It is possible to access overlapping ZA array vectors within tiles of differing element sizes. For example, tiles ZA0.H, ZA2.S, and ZA6.D have common ZA array vectors.

B2.3.3 Horizontal tile slice mappings

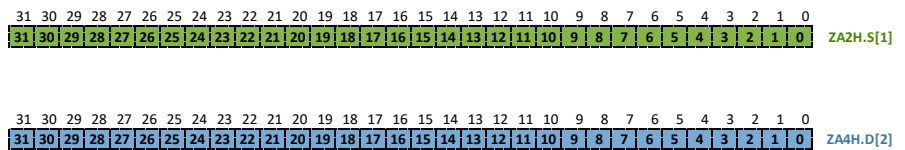
I_{NJJXW}

The following diagram illustrates the ZA storage mapping, for SVL of 256 bits, for a 32-bit element and 64-bit element horizontal tile slice.

Each small numbered square represents 8 bits.



An SME vector load, store, or move instruction which accesses horizontal tile slices ZA2H.S[1] or ZA4H.D[2] treats the slices as vectors with the following layout:

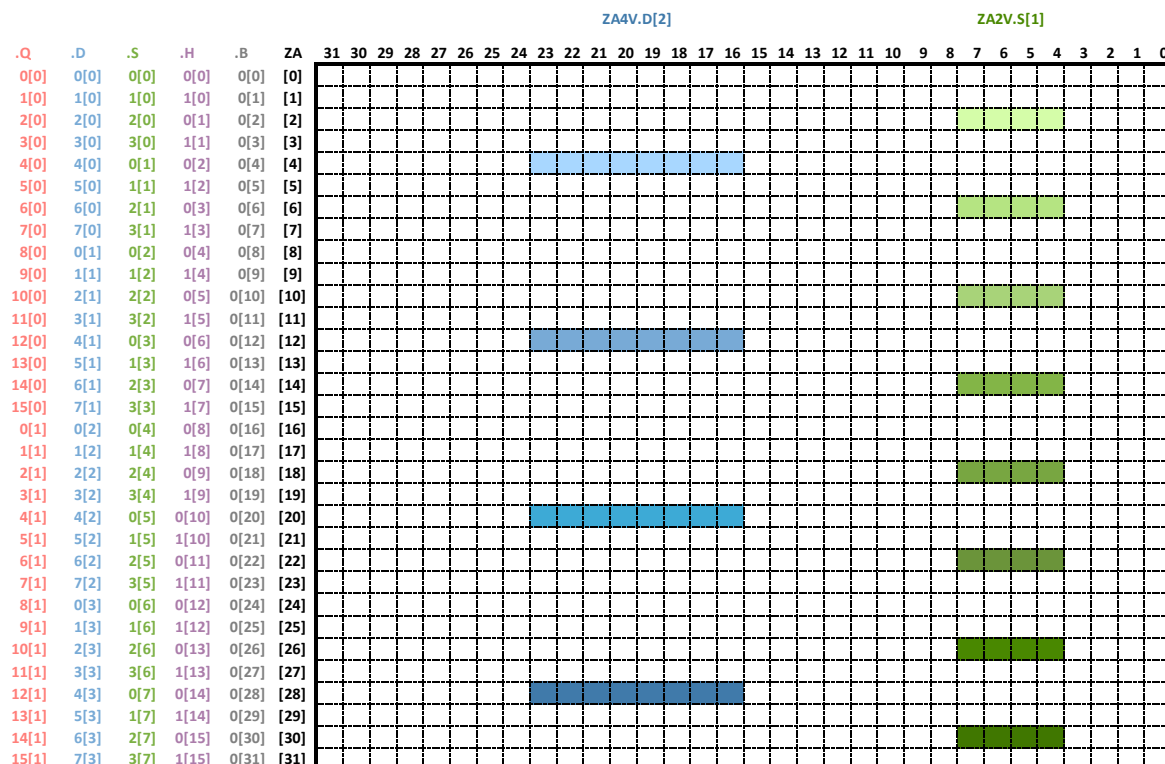


B2.3.4 Vertical tile slice mappings

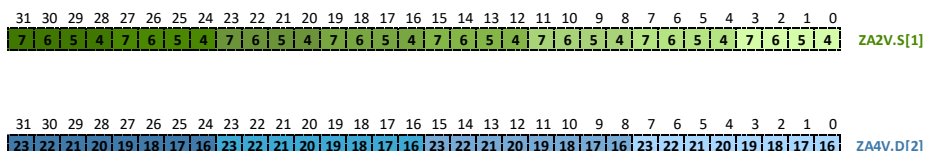
I_{TNCCV}

The following diagram illustrates the ZA storage mapping, for SVL of 256 bits, for a 32-bit element and 64-bit element vertical tile slice.

Each small numbered square represents 8 bits.



An SME vector load, store, or move instruction which accesses vertical tile slices ZA2V.S[1] or ZA4V.D[2] treats the slices as vectors with the following layout:



B2.3.5 Mixed horizontal and vertical tile slice mappings

I_{CGXPJ}

The ZA storage mapping, for SVL of 256 bits, for various element size tiles, horizontal tile slices, and vertical tile slices, is illustrated in the following diagram.

Each small square represents 8 bits.

Chapter B2. Architectural state
B2.3. ZA storage layout

					ZA7V.D[3]				ZA0V.B[22]				ZA3V.S[4]				ZA8V.Q[0]				ZA1V.H[1]																
.Q	.D	.S	.H	.B	ZA	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0[0]	0[0]	0[0]	0[0]	0[0]	0[0]																																
1[0]	1[0]	1[0]	1[0]	1[0]	0[1]																																
2[0]	2[0]	2[0]	0[1]	0[2]	0[2]																																
3[0]	3[0]	3[0]	1[1]	0[3]	0[3]																																
4[0]	4[0]	0[1]	0[2]	0[4]	0[4]																																
5[0]	5[0]	1[1]	1[2]	0[5]	0[5]																																
6[0]	6[0]	2[1]	0[3]	0[6]	0[6]																																
7[0]	7[0]	3[1]	1[3]	0[7]	0[7]																																
8[0]	0[1]	0[2]	0[4]	0[8]	0[8]																																
9[0]	1[1]	1[2]	1[4]	0[9]	0[9]																																
10[0]	2[1]	2[2]	0[5]	0[10]	0[10]																																
11[0]	3[1]	3[2]	1[5]	0[11]	0[11]																																
12[0]	4[1]	0[3]	0[6]	0[12]	0[12]																																
13[0]	5[1]	1[3]	1[6]	0[13]	0[13]																																
14[0]	6[1]	2[3]	0[7]	0[14]	0[14]																																
15[0]	7[1]	3[3]	1[7]	0[15]	0[15]																																
0[1]	0[2]	0[4]	0[8]	0[16]	0[16]																																
1[1]	1[2]	1[4]	1[8]	0[17]	0[17]																																
2[1]	2[2]	2[4]	0[9]	0[18]	0[18]																																
3[1]	3[2]	3[4]	1[9]	0[19]	0[19]																																
4[1]	4[2]	0[5]	0[10]	0[20]	0[20]																																
5[1]	5[2]	1[5]	1[10]	0[21]	0[21]																																
6[1]	6[2]	2[5]	0[11]	0[22]	0[22]																																
7[1]	7[2]	3[5]	1[11]	0[23]	0[23]																																
8[1]	0[3]	0[6]	0[12]	0[24]	0[24]																																
9[1]	1[3]	1[6]	1[12]	0[25]	0[25]																																
10[1]	2[3]	2[6]	0[13]	0[26]	0[26]																																
11[1]	3[3]	3[6]	1[13]	0[27]	0[27]																																
12[1]	4[3]	0[7]	0[14]	0[28]	0[28]																																
13[1]	5[3]	1[7]	1[14]	0[29]	0[29]																																
14[1]	6[3]	2[7]	0[15]	0[30]	0[30]																																
15[1]	7[3]	3[7]	1[15]	0[31]	0[31]																																

Chapter B3

Floating-point behaviors

B3.1 Overview

SME modifies some of the Armv9-A floating-point behaviors when a PE is in *Streaming SVE mode*, and introduces an `FPCR` control which extends BFloat16 dot product calculations to support a wider range of numeric behaviors.

See also:

- [FPCR](#).
- [B1.1.1 Process state](#)

B3.1.1 Extended BFloat16

`DDLN` When the optional FEAT_EBF16 feature is implemented, the following control bit, `FPCR.EBF`, is present at bit [13]. For more information, see [FPCR.EBF](#).

`RBSHYK` When the `ID_AA64ZFR0_EL1.BF16` and `ID_AA64ISAR1_EL1.BF16` fields have the value `0b0010` the PE implements FEAT_EBF16 and supports the `FPCR.EBF` control.

See also:

- [ID_AA64ISAR1_EL1](#).
- [ID_AA64ZFR0_EL1](#).

B3.1.2 BFloat16 behaviors

If FEAT_EBF16 is implemented, the `FPCR.EBF` control can be used to enable the Extended BFloat16 behaviors.

R_{RKGSJ}

If FEAT_EBF16 is implemented, then:

- The FEAT_EBF16 feature is enabled when `FPCR.EBF` is 1.
- The FEAT_EBF16 feature is not enabled when `FPCR.EBF` is 0.

B3.1.2.1 Common BFloat16 behaviors

These are the behaviors currently defined in *Arm® Architecture Reference Manual for A-profile architecture* [1] which are not changed by the optional `FPCR.EBF` control.

R_{GJWLL}

The `BFDOT`, `BFMMLA`, `BFMOPA`, and `BFMOPS` instructions support the BFloat16 and the IEEE 754-2008 Single-precision floating-point data types defined respectively in sections *BFloat16 floating-point format* and *Single-precision floating-point format* of *Arm® Architecture Reference Manual for A-profile architecture* [1].

R_{RDCPW}

The `BFDOT`, `BFMMLA`, `BFMOPA`, and `BFMOPS` instructions which detect exceptional floating-point conditions produce the expected single-precision default result but do not modify the cumulative floating-point exception flag bits, `FPSR.{IDC,IXC,UFC,OFC,DZC,IOC}`.

R_{PFFFF}

The `BFDOT`, `BFMMLA`, `BFMOPA`, and `BFMOPS` instructions generate default NaN values, as if the `FPCR.DN` control had the value 1.

See also:

- `FPSR`, Floating-point Status Register in *Arm® Architecture Reference Manual for A-profile architecture* [1].

B3.1.2.2 Standard BFloat16 behaviors

These are the behaviors currently defined in *Arm® Architecture Reference Manual for A-profile architecture* [1] which may be changed by the `FPCR.EBF` control provided by FEAT_EBF16.

R_{JFWDV}

If FEAT_EBF16 is either not implemented or not enabled, then the `BFDOT`, `BFMMLA`, `BFMOPA`, and `BFMOPS` instructions ignore the `FPCR.RMode` control and use the rounding mode defined for BFloat16 in section *Round to Odd mode* of *Arm® Architecture Reference Manual for A-profile architecture* [1].

R_{WBLQD}

If FEAT_EBF16 is either not implemented or not enabled, then the `BFDOT`, `BFMMLA`, `BFMOPA`, and `BFMOPS` instructions flush denormalized inputs and outputs to zero, as if the `FPCR.FZ` control had the value 1.

R_{JPNSN}

If FEAT_EBF16 is either not implemented or not enabled, then the `BFDOT`, `BFMMLA`, `BFMOPA`, and `BFMOPS` instructions perform unfused multiplies and additions with intermediate rounding of all products and sums.

B3.1.2.3 Extended BFloat16 behaviors

These are the behaviors that may be enabled by the `FPCR.EBF` control provided by FEAT_EBF16.

R_{RQKQZ}

If FEAT_EBF16 is implemented and enabled, then the `BFDOT`, `BFMMLA`, `BFMOPA`, and `BFMOPS` instructions support all four IEEE 754 rounding modes selected by the `FPCR.RMode` control.

R_{JZVPD}

If FEAT_EBF16 is implemented and enabled, then the `BFDOT`, `BFMMLA`, `BFMOPA`, and `BFMOPS` instructions may flush denormalized inputs and outputs to zero, governed by the `FPCR.FZ` control.

R_{LJGTX}

If FEAT_EBF16 is implemented and enabled, then the `BFDOT`, `BFMMLA`, `BFMOPA`, and `BFMOPS` instructions perform a fused two-way sum-of-products for each pair of adjacent BFloat16 elements in the source vectors, without intermediate rounding of the products, but rounding the single-precision sum before addition to the single-precision accumulator element.

R_{CQFQT}

If FEAT_EBF16 is implemented and enabled, then the `BFDOT`, `BFMMLA`, `BFMOPA`, and `BFMOPS` instructions generate the default NaN as intermediate sum-of-products when any multiplier input is a NaN, or any product is infinity \times 0.0, or there are infinite products with differing signs.

R_{WQDBR}

If FEAT_EBF16 is implemented and enabled, then the `BFDOT`, `BFMMLA`, `BFMOPA`, and `BFMOPS` instructions generate an intermediate sum-of-products of the same infinity when there are infinite products all with the same sign.

R _{YPGLB}	When a PE implements the FEAT_AFP feature and FEAT_EBF16 is implemented and enabled, the BFDOT, BFMLLA, BFMOPA, and BFMOPS instructions honor the FPCR.AH and FPCR.FIZ controls.
I _{WKNML}	When a PE implements the FEAT_AFP feature and FEAT_EBF16 is implemented and enabled, the following alternate floating-point behaviors affect the BFDOT, BFMLLA, BFMOPA, and BFMOPS instructions: <ul style="list-style-type: none"> When FPCR.AH is 1, the sign bit of a generated default NaN result is set to 1 instead of 0. When FPCR.AH is 1 and FPCR.FZ is 1, a denormal result, detected after rounding with an unbounded exponent has been applied, is flushed to zero. When FPCR.AH is 1, the FPCR.FZ control does not cause denormalized inputs to be flushed to zero. When FPCR.FIZ is 1, all denormalized inputs are flushed to zero.

B3.1.3 Floating-point behaviors in Streaming SVE mode

R _{NHZCN}	When the PE is in <i>Streaming SVE mode</i> , the legal floating-point instructions that operate on half-precision, single-precision, and double-precision input data types and write to SIMD&FP registers or SVE Z vectors, and the SVE BFMLALB and BFMLALT instructions, honor the Non-streaming scalar and SVE floating-point behaviors, as governed by the FPCR.{RMode, AHP, DN, FZ, FZ16} controls.
R _{GTYSK}	When the PE is in <i>Streaming SVE mode</i> , the legal floating-point instructions that operate on half-precision, single-precision, and double-precision input data types and write to SIMD&FP registers or SVE Z vectors, and the SVE BFMLALB and BFMLALT instructions, which detect exceptional floating-point conditions produce the expected IEEE 754 default result and set the appropriate cumulative floating-point exception flag bits in FPSR.{IDC, IXC, UFC, OFC, DZC, IOC} to 1.
R _{FBFNT}	When the PE is in <i>Streaming SVE mode</i> and FEAT_SME_FA64 is not implemented or not enabled at the current Exception level, the <i>Effective value</i> of the FPCR is as if all of the IDE, IXE, UFE, OFE, DZE, and IOE floating-point exception trap enable controls, and the NEP element preserve control, are 0 for all purposes other than a direct read or write of the register.
R _{SGZLB}	When a PE which implements the FEAT_AFP feature is in <i>Streaming SVE mode</i> , the legal floating-point instructions that operate on half-precision, single-precision, and double-precision input data types and write to SIMD&FP registers or SVE Z vectors, and the SVE BFMLALB and BFMLALT instructions, honor the FPCR.AH and FPCR.FIZ controls.

See also:

- FPSR, Floating-point Status Register in *Arm® Architecture Reference Manual for A-profile architecture* [1].

B3.1.4 ZA-targeting floating-point behaviors

R _{KVYHW}	SME floating-point instructions that write to the ZA array, except for BFMOPA and BFMOPS, support the IEEE 754-2008 floating-point data types as defined in the following sections of <i>Arm® Architecture Reference Manual for A-profile architecture</i> [1]: <ul style="list-style-type: none"> Half-precision floating-point formats (but not the Arm alternative half-precision format). Single-precision floating-point format. Double-precision floating-point format.
R _{TGSKG}	SME floating-point instructions that write to the ZA array, except for BFMOPA and BFMOPS, which detect exceptional floating-point conditions produce the expected IEEE 754 default result but do not modify any of the cumulative floating-point exception flag bits, FPSR.{IDC, IXC, UFC, OFC, DZC, IOC}.
R _{RKHHZ}	SME floating-point instructions that write to the ZA array, except for BFMOPA and BFMOPS, generate default NaN values, as if the FPCR.DN control had the value 1.
R _{TCLRM}	SME floating-point instructions that write to the ZA array, except for BFMOPA and BFMOPS, support all four IEEE 754 rounding modes selected by the FPCR.RMode control.
R _{VVVR}	

	SME floating-point instructions that write to the ZA array, except for <code>BFMOA</code> and <code>BFMOPS</code> , may flush denormalized single- and double-precision inputs and outputs to zero, governed by the <code>FPCR.FZ</code> control.
<code>R_TXKVK</code>	SME floating-point instructions which accumulate dot products of pairs of adjacent half-precision elements in the source vectors into single-precision elements in the ZA array may flush denormalized half-precision inputs to zero, governed by the <code>FPCR.FZ16</code> control.
<code>R_JRRMJ</code>	SME floating-point instructions which multiply single elements from each source vector and accumulate their product into the ZA array perform a fused multiply-add to each accumulator tile element, without intermediate rounding.
<code>R_NNCFV</code>	SME floating-point instructions which accumulate dot products of pairs of adjacent half-precision elements in the source vectors into single-precision elements in the ZA array perform a fused sum-of-products, without intermediate rounding of the products, but rounding the single-precision sum before addition to the accumulator.
<code>R_QPKJC</code>	SME floating-point instructions which accumulate dot products of pairs of adjacent half-precision elements in the source vectors into single-precision elements in the ZA array generate the default NaN as intermediate sum-of-products when any multiplier input is a NaN, or any product is infinity \times 0.0, or there are infinite products with differing signs.
<code>R_ZBLND</code>	SME floating-point instructions which accumulate dot products of pairs of adjacent half-precision elements in the source vectors into single-precision elements in the ZA array generate an intermediate sum-of-products of the same infinity when there are infinite products all with the same sign.
<code>R_RPSLK</code>	When a PE implements the FEAT_AFP feature the SME floating-point instructions that write to the ZA array, except for <code>BFMOA</code> and <code>BFMOPS</code> , honor the <code>FPCR.AH</code> and <code>FPCR.FIZ</code> controls.
<code>I_YPCHJ</code>	When a PE implements the FEAT_AFP feature, the following alternate floating-point behaviors affect the SME <code>FMOPA</code> and <code>FMOPS</code> instructions: <ul style="list-style-type: none"> • When <code>FPCR.AH</code> is 1, the sign bit of a generated default NaN result is set to 1 instead of 0. • When <code>FPCR.AH</code> is 1 and <code>FPCR.FZ</code> is 1, a denormal result, detected after rounding with an unbounded exponent has been applied, is flushed to zero. • When <code>FPCR.AH</code> is 1, the <code>FPCR.FZ</code> control does not cause denormalized inputs to be flushed to zero. • When <code>FPCR.FIZ</code> is 1, all denormalized single-precision and double-precision inputs are flushed to zero.

Part C

SME system level programmers' model

Chapter C1

Introduction

The SME System Management architecture provides mechanisms for system software to:

- Discover the presence of SME.
- Discover the capabilities of SME.
- Control SME usage.
- Monitor SME usage.

The architecture consists of extensions to processor mode, the Exception model, and System registers for trap control and identification.

Chapter C2

System management

C2.1 Overview

SME extends the AArch64 System registers and processor state, by introducing the following:

- An SME presence identification field added to `ID_AA64PFR1_EL1`.
- An SME-specific ID register, `ID_AA64SMFR0_EL1`, for SME feature discovery.
- An execution mode, *Streaming SVE mode*.
- Mode flags in `PSTATE` to enable *Streaming SVE mode* and SME architectural state, accessible using the `SVCR` register.
- Configuration settings for SME in `CPACR_EL1`, `CPTR_EL2`, `CPTR_EL3`, `HCR_EL2`, `HCRX_EL2`, `HFGTR_EL2`, and `HFGWTR_EL2`.
- An SME exception type, with new `ESR_ELx.EC` and `ESR_ELx.ISS` encodings.
- A field in `ESR_ELx.ISS` to signal that an imprecise `FAR_ELx` value has been reported on a synchronous Data Abort exception.
- SME controls to set the Effective Streaming SVE vector length in `SMCR_EL1`, `SMCR_EL2`, and `SMCR_EL3`.
- ID and control registers that influence streaming execution priority in multiprocessor systems: `SMIDR_EL1`, `SMPRI_EL1`, and `SMPRMAP_EL2`.
- A software thread ID register to manage per-thread SME context, `TPIDR2_EL0`, with enables in `SCR_EL3`, `SCTLR_EL2`, and `SCTLR_EL1`.

See also:

- [ID_AA64PFR1_EL1](#)
- [CPACR_EL1](#)
- [CPTR_EL2](#)
- [CPTR_EL3](#)

- [ESR_EL1, ESR_EL2, and ESR_EL3](#)
- [HCR_EL2](#)
- [HCRX_EL2](#)
- [HFGTR_EL2](#)
- [HFGWTR_EL2](#)
- [SCR_EL3](#)
- [SCTLR_EL2](#)
- [SCTLR_EL1](#)
- [ID_AA64SMFR0_EL1](#)
- [SMCR_EL1](#)
- [SMCR_EL2](#)
- [SMCR_EL3](#)
- [SMIDR_EL1](#)
- [SMPRI_EL1](#)
- [SMPRMAP_EL2](#)
- [SVC](#)
- [TPIDR2_EL0](#)

C2.1.1 Identification

`I_XTNNP` `ID_AA64PFR1_EL1.SME` indicates whether SME is implemented in a PE.

`I_RLFX` If SME is implemented, the SME features that are implemented in a PE are determined from `ID_AA64SMFR0_EL1`.

See also:

- [ID_AA64PFR1_EL1](#)
- [ID_AA64SMFR0_EL1](#)

C2.1.2 Traps and exceptions

`D_DMBHW` The SME-related instructions that may be configured to trap by `CPACR_EL1`, `CPTR_EL2`, and `CPTR_EL3` controls, unless otherwise stated, include all of the following:

- SME data-processing instructions.
- SME mode change instructions `SMSTART` and `SMSTOP`.
- AArch64 `MRS` and `MSR` instructions which directly access any of the `SVC`, `SMCR_EL1`, `SMCR_EL2`, or `SMCR_EL3` registers.

`I_MQBFG` Execution of SME-related instructions can be trapped by supervisor software. The mechanisms provided are:

- `CPACR_EL1`, which enables execution of SME-related instructions at EL0 or EL1 to be trapped to EL1 or EL2.
- `CPTR_EL2`, which enables execution of SME-related instructions at EL0, EL1 or EL2 to be trapped to EL2.
- `CPTR_EL3`, which enables execution of SME-related instructions at any Exception level to be trapped to EL3.

`I_XKMKY` SME adds an exception syndrome value `0b011101 (0x1D)`, which is used to identify instructions that are trapped by the SME controls in `CPACR_EL1`, `CPTR_EL2`, and `CPTR_EL3`, or by the `PSTATE.SM` and `PSTATE.ZA` modes.

See also:

- [CPACR_EL1](#)
- [CPTR_EL2](#)
- [CPTR_EL3](#)
- [ESR_EL1, ESR_EL2, and ESR_EL3](#)
- [C2.2.1 Exception priorities](#)

C2.1.3 Vector lengths

D _{QQRNR}	The Effective Streaming SVE vector length, SVL, is the accessible length in bits of the ZA array vectors and Streaming SVE vector registers at the current Exception level. SVL is determined by the <code>LEN</code> field of the appropriate SME <code>SMCR_ELx</code> registers, as defined in rule R _{GWVHP} .
I _{BHFWG}	SVL is used explicitly by the unpredicated SME <code>LDR</code> , <code>STR</code> , and <code>ZERO</code> instructions which can access the ZA array irrespective of whether the PE is in <i>Streaming SVE mode</i> .
R _{VCQBB}	The Effective SVE vector length, VL, is equal to SVL when the PE is in <i>Streaming SVE mode</i> .
I _{LRBQY}	The Effective SVE vector length, VL, is determined by the <code>LEN</code> field of the <code>ZCR_ELx</code> registers when the PE is not in <i>Streaming SVE mode</i> and FEAT_SVE is implemented.
I _{WQZKK}	Unlike the Non-streaming SVE vector length, SVL can only be a power of two.
R _{JRCSH}	An implementation is permitted to support any subset of the architecturally defined SVL values.
I _{FQKMN}	For example, this means that the set of supported SVLs might be discontinuous and might not start at the smallest permitted SVL.
R _{RZNVH}	An implementation is permitted but not required to support more than one SVL.
R _{WDKGR}	An implementation is permitted to support a set of SVLs that do not overlap with the set of VLs that are supported when the PE is not in <i>Streaming SVE mode</i> .
I _{GZRPL}	There is no requirement for the maximum implemented Streaming SVE vector length to be greater than or equal to the maximum implemented SVE vector length.
R _{GWVHP}	The Effective Streaming SVE vector length at a given Exception level is determined from the requested length, encoded as a multiple of 16 bytes in <code>SMCR_EL1.LEN</code> , <code>SMCR_EL2.LEN</code> , or <code>SMCR_EL3.LEN</code> , according to the Exception level, following these steps: <ol style="list-style-type: none"> 1. If the requested length is less than the minimum implemented Streaming SVE vector length, the Effective length is the minimum implemented Streaming SVE vector length. 2. If this is the highest implemented Exception level and the requested length is greater than the maximum implemented Streaming SVE vector length, then the Effective length is the maximum implemented Streaming SVE vector length. 3. If this is not the highest implemented Exception level and the requested length is greater than the Effective length at the next more privileged implemented Exception level in the current Security state, then the Effective length at the more privileged Exception level is used. 4. If the requested length is not supported by the implementation, then the Effective length is the highest supported Streaming SVE vector length that is less than the requested length. 5. Otherwise, the Effective length is the requested length.
I _{VRRYR}	The set of supported values of SVL at Exception level ELx (where ELx is EL1, EL2 or EL3) can be discovered by privileged software in a similar way to determining the set of supported values of VL. For example, when SME is enabled by the appropriate control fields in <code>CPACR_EL1</code> , <code>CPTR_EL2</code> and <code>CPTR_EL3</code> : <ol style="list-style-type: none"> 1. Request an out of range vector length of 8192 bytes by writing <code>0x1fff</code> to <code>SMCR_ELx[8:0]</code>. 2. Use the SME <code>RDSVL</code> instruction to read SVL. 3. If <code>SMCR_ELx</code> requests a supported Streaming SVE vector length, the requested length in bytes will be returned by <code>RDSVL</code>. 4. If <code>SMCR_ELx</code> requests an unsupported Streaming SVE vector length, a supported length in bytes will be returned by <code>RDSVL</code>. 5. If the returned length <code>len</code> is less than or equal to the requested vector length, and greater than 16 bytes (128 bits), then request the next lower vector length by writing $(len/16)-2$ to <code>SMCR_ELx[8:0]</code> and go to step 2.
R _{YRPDH}	When SVL changes from a smaller to a larger value without leaving <i>Streaming SVE mode</i> , a new area of storage becomes architecturally visible in the Streaming SVE registers and, if enabled, the SME ZA storage. The values in

the area common to the previous and current length are preserved, and the values in the newly accessible area are a CONSTRAINED UNPREDICTABLE choice between the following:

- Zero.
- The value the bits had before executing at the more constrained size.

I _{HGYPV}	The SVL might change without leaving <i>Streaming SVE mode</i> because of an explicit action such as a write to SMCR_ELx, or an implicit action such as taking an exception to an Exception level with a less constrained SVL.
I _{PDLWX}	The SVL can be raised and then restored to a previous value without affecting the original contents of the Streaming SVE registers and the SME ZA storage.
I _{FMWZZ}	Supervisory software must guarantee that values generated by one body of software are not observable by another body of software in a different trust or security scope. When SVL is increased, steps must be taken to ensure the newly accessible area does not contain values unrelated to another body of software. This might be achieved by ensuring that the PE exits <i>Streaming SVE mode</i> and disables SME ZA storage when performing a context switch, or by explicitly resetting all register values.
I _{BRMMV}	System software provides a maximum SVL to lower-privileged software, which might further constrain the SVL. However, system software must initialize and context switch values consistent with the maximum SVL provided and should not make assumptions about any smaller size being in use by lower-privileged software. For example, if a hypervisor exposes an SVL of 512 to a VM, that VM might choose to constrain SVL to 256. The hypervisor should still save and restore 512-bit vectors to prevent leakage of values between VMs, because the VM might later raise its SVL to 512 and must not be able to observe values created by other software in the newly visible upper portion of the registers.

See also:

- [SMCR_EL1](#)
- [SMCR_EL2](#)
- [SMCR_EL3](#)

C2.1.4 Streaming execution priority

D _{DXMSW}	Streaming execution refers to the execution of instructions by a PE when that PE is in <i>Streaming SVE mode</i> .
I _{CMRVS}	Arm expects a variety of implementation styles for SME, including styles where more than one PE shares SME and Streaming SVE compute resources.
I _{PQNJS}	Shared SME and Streaming SVE compute resources are called a <i>Streaming Mode Compute Unit</i> (SMCU).
I _{MZXWD}	For implementations that share an SMCU, this architecture provides per-PE mechanisms that software can use to dynamically prioritize performance characteristics experienced by each PE.

See also:

- [C2.2.4 Streaming execution priority for shared implementations](#)

C2.2 Processor behavior

C2.2.1 Exception priorities

R_{NXSFT}

Exceptions due to configuration settings and modes resulting from the attempted execution of an SME data-processing instruction are evaluated in the following order from highest to lowest priority:

1. If FEAT_SME is not implemented, then the instruction is UNDEFINED.
2. If CPACR_EL1.SMEN configures the instruction to trap, it is reported using ESR_ELx.EC value 0x1D with ISS code 0x0000000.
3. If CPACR_EL1.FPEN configures the instruction to trap, it is reported using ESR_ELx.EC value 0x07.
4. If CPTR_EL2.SMEN configures the instruction to trap, it is reported using ESR_ELx.EC value 0x1D with ISS code 0x0000000.
5. If CPTR_EL2.FPEN configures the instruction to trap, it is reported using ESR_ELx.EC value 0x07.
6. If CPTR_EL2.TSM configures the instruction to trap, it is reported using ESR_ELx.EC value 0x1D with ISS code 0x0000000.
7. If CPTR_EL2.TFP configures the instruction to trap, it is reported using ESR_ELx.EC value 0x07.
8. If CPTR_EL3.ESM configures the instruction to trap, it is reported using ESR_ELx.EC value 0x1D with ISS code 0x0000000.
9. If CPTR_EL3.TFP configures the instruction to trap, it is reported using ESR_ELx.EC value 0x07.
10. If the PE is not in *Streaming SVE mode* and the SME instruction would access any of the SVE registers Z0-Z31 or P0-P15, then an SME exception is taken, using ESR_ELx.EC value 0x1D with ISS code 0x0000002.
11. If the SME ZA storage is disabled and the SME instruction would access ZA, then an SME exception is taken, using ESR_ELx.EC value 0x1D with ISS code 0x0000003.
12. Otherwise, the instruction executes.

R_{XQKHH}

Exceptions due to configuration settings resulting from attempted execution of MRS or MSR instructions which directly access one of the SVCR, SMCR_EL1, SMCR_EL2, or SMCR_EL3 registers, are evaluated in the following order from highest to lowest priority:

1. If FEAT_SME is not implemented, then the instruction is UNDEFINED.
2. If CPACR_EL1.SMEN configures the instruction to trap, it is reported using ESR_ELx.EC value 0x1D, with ISS code 0x0000000.
3. If CPTR_EL2.SMEN configures the instruction to trap, it is reported using ESR_ELx.EC value 0x1D, with ISS code 0x0000000.
4. If CPTR_EL2.TSM configures the instruction to trap, it is reported using ESR_ELx.EC value 0x1D, with ISS code 0x0000000.
5. If CPTR_EL3.ESM configures the instruction to trap, it is reported using ESR_ELx.EC value 0x1D, with ISS code 0x0000000.
6. Otherwise, the instruction executes.

R_{PLYVH}

Exceptions due to configuration settings and modes resulting from the attempted execution of an SVE or SVE2 instruction when FEAT_SVE is not implemented or when the PE is in *Streaming SVE mode*, are evaluated in the following order from highest to lowest priority:

1. If FEAT_SME is not implemented and FEAT_SVE is not implemented, then the instruction is UNDEFINED.
2. If FEAT_SVE is not implemented and the instruction is illegal when the PE is in *Streaming SVE mode*, then the instruction is UNDEFINED.
3. If FEAT_SME is not implemented and the instruction is defined as part of the SME architecture in [D1.3 SVE2 instructions](#), then the instruction is UNDEFINED.
4. If CPACR_EL1.SMEN configures the instruction to trap, it is reported using ESR_ELx.EC value 0x1D with ISS code 0x0000000.
5. If CPACR_EL1.FPEN configures the instruction to trap, it is reported using ESR_ELx.EC value 0x07.
6. If CPTR_EL2.SMEN configures the instruction to trap, it is reported using ESR_ELx.EC value 0x1D with ISS code 0x0000000.
7. If CPTR_EL2.FPEN configures the instruction to trap, it is reported using ESR_ELx.EC value 0x07.

8. If CPTR_EL2.TSM configures the instruction to trap, it is reported using ESR_ELx.EC value 0x1D with ISS code 0x0000000.
9. If CPTR_EL2.TFP configures the instruction to trap, it is reported using ESR_ELx.EC value 0x07.
10. If CPTR_EL3.ESM configures the instruction to trap, it is reported using ESR_ELx.EC value 0x1D with ISS code 0x0000000.
11. If CPTR_EL3.TFP configures the instruction to trap, it is reported using ESR_ELx.EC value 0x07.
12. If the PE is in *Streaming SVE mode* and the SVE instruction is *illegal* in that mode, then an SME exception is taken, using ESR_ELx.EC value 0x1D with ISS code 0x0000001.
13. If the PE is not in *Streaming SVE mode* and FEAT_SVE is not implemented, then an SME exception is taken, using ESR_ELx.EC value 0x1D with ISS code 0x0000002.
14. Otherwise, the instruction executes.

R_ZTKXF

Exceptions due to configuration settings resulting from the attempted execution of an SVE or SVE2 instruction when FEAT_SVE is implemented and when the PE is not in *Streaming SVE mode*, are evaluated in the following order from highest to lowest priority:

1. If FEAT_SME is not implemented and the instruction is defined as part of the SME architecture in [D1.3 SVE2 instructions](#), then the instruction is UNDEFINED.
2. If CPACR_EL1.ZEN configures the instruction to trap, it is reported using ESR_ELx.EC value 0x19.
3. If CPACR_EL1.FPEN configures the instruction to trap, it is reported using ESR_ELx.EC value 0x07.
4. If CPTR_EL2.ZEN configures the instruction to trap, it is reported using ESR_ELx.EC value 0x19.
5. If CPTR_EL2.FPEN configures the instruction to trap, it is reported using ESR_ELx.EC value 0x07.
6. If CPTR_EL2.TZ configures the instruction to trap, it is reported using ESR_ELx.EC value 0x19.
7. If CPTR_EL2.TFP configures the instruction to trap, it is reported using ESR_ELx.EC value 0x07.
8. If CPTR_EL3.EZ configures the instruction to trap, it is reported using ESR_ELx.EC value 0x19.
9. If CPTR_EL3.TFP configures the instruction to trap, it is reported using ESR_ELx.EC value 0x07.
10. Otherwise, the instruction executes.

R_DTC LZ

Exceptions due to configuration settings and modes resulting from the attempted execution of an AArch64 Advanced SIMD and floating-point instruction when the PE is in *Streaming SVE mode* are evaluated in the following order from highest to lowest priority:

1. If CPACR_EL1.FPEN configures the instruction to trap, it is reported using ESR_ELx.EC value 0x07.
2. If CPTR_EL2.FPEN configures the instruction to trap, it is reported using ESR_ELx.EC value 0x07.
3. If CPTR_EL2.TFP configures the instruction to trap, it is reported using ESR_ELx.EC value 0x07.
4. If CPTR_EL3.TFP configures the instruction to trap, it is reported using ESR_ELx.EC value 0x07.
5. If the instruction is *illegal* when the PE is in **Streaming SVE mode*, then an SME exception is taken, using ESR_ELx.EC value 0x1D with ISS code 0x0000001.
6. Otherwise, the instruction executes.

I_NWNQZ

When the PE is in *Streaming SVE mode* or FEAT_SVE is not implemented, the CPACR_EL1.SMEN, CPTR_EL2.SMEN, CPTR_EL2.TSM, and CPTR_EL3.ESM controls configure SVE instructions to trap, and the CPACR_EL1.ZEN, CPTR_EL2.ZEN, CPTR_EL2.TZ, and CPTR_EL3.EZ controls do not cause any SVE instructions to be trapped.

I_PKGPR

When the PE is not in *Streaming SVE mode* and FEAT_SVE is implemented, the CPACR_EL1.ZEN, CPTR_EL2.ZEN, CPTR_EL2.TZ, and CPTR_EL3.EZ controls configure SVE instructions to trap, and the CPACR_EL1.SMEN, CPTR_EL2.SMEN, CPTR_EL2.TSM, and CPTR_EL3.ESM controls do not cause any SVE instructions to be trapped.

R_ZZBRC

An Undefined Instruction exception due to the pairing of an SVE MOVPRFX with an instruction which cannot be predictably prefixed has a higher exception priority than a PSTATE mode-dependent SME exception with ESR_ELx.EC value 0x1D and an ISS code that is not 0x0000000.

See also:

- [CPACR_EL1](#)
- [CPTR_EL2](#)
- [CPTR_EL3](#)
- [ESR_EL1, ESR_EL2, and ESR_EL3](#)
- [C2.2 Processor behavior](#)

- [Chapter E1 Instructions affected by SME](#)

C2.2.2 Synchronous Data Abort

<code>R_JXPNL</code>	<p>If SME is implemented, then on taking a Data Abort exception which sets <code>ESR_ELx.ISV</code> to 0 caused by an SVE contiguous vector load/store instruction when the PE is in Streaming SVE mode, or by an SME load/store instruction, the PE:</p> <ul style="list-style-type: none"> • Sets <code>ESR_ELx.FnP</code> to 1 if the value written to the corresponding <code>FAR_ELx</code> might not be the same as the faulting virtual address that generated the Data Abort. • Otherwise, sets <code>ESR_ELx.FnP</code> to 0.
<code>R_XMWQT</code>	<p>If the PE sets <code>ESR_ELx.ISV</code> to 0 and <code>ESR_ELx.FnP</code> to 1 on taking a Data Abort exception, then the PE sets the corresponding <code>FAR_ELx</code> to any address within the <i>naturally-aligned fault granule</i> which contains the faulting virtual address that generated the Data Abort.</p>
<code>D_BRGHW</code>	<p>The <i>naturally-aligned fault granule</i> is one of the following:</p> <ul style="list-style-type: none"> • A 16-byte tag granule when <code>ESR_ELx.DFSC</code> is 0b010001, indicating a Synchronous Tag Check fault. • An IMPLEMENTATION DEFINED granule when <code>ESR_ELx.DFSC</code> is 0b11010x, indicating an IMPLEMENTATION DEFINED fault. • Otherwise, the smallest implemented translation granule. <p>See also:</p> <ul style="list-style-type: none"> • ESR_EL1, ESR_EL2, and ESR_EL3 • FAR_EL1 • FAR_EL2 • FAR_EL3

C2.2.3 Validity of SME and SVE state

<code>I_WFHKZ</code>	<p>Fields in <code>CPACR_EL1</code>, <code>CPTR_EL2</code>, and <code>CPTR_EL3</code> configure whether SME-related instructions can be executed, or are trapped.</p>
<code>I_VBJBR</code>	<p>The Effective values of <code>PSTATE.SM</code> and <code>PSTATE.ZA</code> configure whether SME architectural state is valid and accessible.</p>
<code>R_XCCXW</code>	<p>The controls for trapping SME-related instructions and controls for validity of SME architectural state are independent.</p>
<code>I_JGRTR</code>	<p>Because the trap and architectural state validity are controlled independently, the following scenarios are all permissible:</p> <ul style="list-style-type: none"> • Instructions trap, state invalid. <ul style="list-style-type: none"> – For example, an OS traps the first usage of SME-related instructions by a process. • Instructions trap, state valid. <ul style="list-style-type: none"> – For example, a process was running with valid SME architectural state and an OS configures traps to detect when the next usage of SME architectural state occurs. – Enabling the trap does not affect or corrupt the SME architectural state. • Instructions permitted, state invalid. <ul style="list-style-type: none"> – For example, a process is permitted to execute SME-related instructions but is currently not running in <i>Streaming SVE mode</i>. SME data-processing instructions which access SVE vector or predicate registers are <i>illegal</i> and are trapped, but SVE instructions operate on the Non-streaming SVE register state. The process can execute an <code>SMSTART</code> instruction to enter <i>Streaming SVE mode</i>. – For example, a process is running in <i>Streaming SVE mode</i>, but has not enabled access to the SME ZA storage. SME instructions that access <code>ZA</code> are <i>illegal</i> and are trapped, but the process can execute an <code>SMSTART</code> instruction to enable access to the <code>ZA</code> storage. • Instructions permitted, state valid.

- For example, a process is running in *Streaming SVE mode*, and has enabled access to ZA storage.

R _{SWQGH}	An exception return from AArch64 to AArch32 Execution state does not change the values of <code>PSTATE.SM</code> and <code>PSTATE.ZA</code> .
R _{MZLVB}	An exception taken from AArch32 to AArch64 Execution state does not change the values of <code>PSTATE.SM</code> and <code>PSTATE.ZA</code> .
R _{GXKNK}	When a PE is executing in the AArch32 Execution state, the Effective value of <code>PSTATE.SM</code> is 0.
I _{MWQNV}	When <code>PSTATE.SM</code> is 1, a change in Execution state from AArch64 to AArch32, or from AArch32 to AArch64, causes all implemented bits of the SVE registers (including SIMD&FP registers) and the <code>FPSR</code> to be reset to a fixed value, which software must mitigate.
I _{WYKRM}	The Effective value of <code>PSTATE.ZA</code> does not change in AArch32 Execution state, so a transition between AArch64 and AArch32 Execution state when <code>PSTATE.ZA</code> is 1 has no effect on the contents of SME ZA storage.
I _{VGWQW}	An implementation might use the activity of the <code>PSTATE.SM</code> and <code>PSTATE.ZA</code> bits to influence the choice of power-saving states for both functional units and retention of architected state.

C2.2.4 Streaming execution priority for shared implementations

I _{YYRZQ}	Execution of certain instructions by a PE in <i>Streaming SVE mode</i> might experience a performance dependency on other PEs in the system that are also executing instructions in <i>Streaming SVE mode</i> . For example, this might occur when a <i>Streaming Mode Compute Unit</i> (SMCU) is shared between PEs.
I _{WPQVV}	The architecture provides a mechanism to control the streaming execution priority of a PE, in <code>SMPRI_EL1</code> . The streaming execution priority of a PE is relative to the streaming execution priority of other PEs, when a performance dependency exists between PEs executing in <i>Streaming SVE mode</i> .
D _{DGRIS}	All PEs that share a given SMCU form a <i>Priority domain</i> .
D _{YQFWM}	Different <i>Priority domains</i> represent unrelated SMCUs.
R _{WPVQK}	All PEs in a <i>Priority domain</i> have the same value of <code>SMIDR_EL1.Affinity</code> .
R _{CVLSF}	PEs in differing <i>Priority domains</i> have different values of <code>SMIDR_EL1.Affinity</code> .
R _{GGDRC}	The streaming execution priority in <code>SMPRI_EL1</code> affects execution of a PE relative to all other PEs in the same <i>Priority domain</i> .
I _{WVGW}	System software can use the streaming execution priority mechanism to manage scenarios where multiple concurrent software threads contend on shared SMCUs.
R _{RQXFC}	The streaming execution priority mechanism affects the execution of instructions by a shared SMCU when the PE is in <i>Streaming SVE mode</i> and does not directly control the execution of other types of instruction.
I _{HQXBH}	The streaming execution priority mechanism is optional.
I _{KTYTD}	An implementation that does not share SMCUs or has no performance dependency between PEs might not need to limit or prioritize execution of one PE relative to another.
I _{YBQNW}	The architecture considers <i>Priority domains</i> to be non-overlapping sets, meaning that in a shared-SMCU system a PE is associated with at most one SMCU.

C2.2.4.1 Streaming execution context management

I _{PRNMJ}	<p>Arm expects that the SVE- and SME-related instructions used to save, restore, and clear routines for <i>Streaming SVE mode</i> SVE register state and ZA array state are limited to using the following SME and SVE instructions:</p> <ul style="list-style-type: none"> • Unpredicated SME ZA array vector <code>LDR</code> and <code>STR</code> instructions. • Unpredicated SVE vector register <code>LDR</code> and <code>STR</code> instructions. • Unpredicated SVE predicate register <code>LDR</code> and <code>STR</code> instructions.
--------------------	--

- SME ZA tile `ZERO` instruction.
- SVE vector `DUP` (immediate) instruction with zero immediate.
- SVE predicate `PFALSE` instruction.

For implementations with a shared SMCU, PEs are expected to execute these instructions in a way that experiences a reduced effect of contention from other PEs, compared to other SME and SVE instructions executed in *Streaming SVE mode*.

C2.2.4.2 Streaming execution priority control

<code>I_RMDCP</code>	The streaming execution priority is controlled by a 4-bit priority value. When the streaming execution priority mechanism is not supported, the priority value is ignored.
<code>I_FJQRG</code>	A higher priority value corresponds to a higher streaming execution priority. Priority value 15 is the highest priority.
<code>I_QHKWP</code>	The behavior of any given priority value relative to that of another PE is IMPLEMENTATION DEFINED.

C2.2.4.3 Streaming execution priority virtualization

<code>I_SQBCZ</code>	The Effective streaming execution priority is either the value configured in <code>SMPRI_EL1</code> , or the value of <code>SMPRI_EL1</code> mapped into a new value by indexing the fields in <code>SMPRMAP_EL2</code> . This choice is affected by the current Exception level, and the <code>HCRX_EL2.SMPME</code> configuration.
<code>I_LSZZG</code>	A hypervisor can use <code>SMPRMAP_EL2</code> to map the virtual streaming execution priority values written into <code>SMPRI_EL1</code> by a guest OS into different physical priority values.

See also:

- [SMIDR_EL1](#)
- [SMPRI_EL1](#)
- [SMPRMAP_EL2](#)
- [HCRX_EL2](#)

C2.2.5 Security considerations

<code>I_DXRGG</code>	All load and store instructions introduced by SME adhere to the memory access permissions model in VMSA.
<code>I_MGLWR</code>	<p>The entire architectural state added by SME can be access-controlled, meaning that higher levels of privilege can trap access to the state from the same or lower levels of privilege.</p> <p>For example, execution of SME instructions including entry to or exit from <i>Streaming SVE mode</i> in EL0 might be trapped to EL2.</p>
<code>I_CYPJJ</code>	System software has controls available to save and restore state between unrelated pieces of software, and must ensure that steps are taken to preserve isolation and privacy.
<code>I_TDPHC</code>	Operations performed in <i>Streaming SVE mode</i> respect the requirements of <code>PSTATE.DIT</code> . <code>DIT</code> requires data-independent timing when enabled.

C2.3 Changes to existing System registers

C2.3.1 ID_AA64PFR1_EL1

- D_{KHPZL}** If SME is implemented, the field `ID_AA64PFR1_EL1.SME` is defined at bits [27:24]. For more information, see [ID_AA64PFR1_EL1.SME](#).
- I_{DJQVZ}** A non-zero value in `ID_AA64PFR1_EL1.SME` does not imply that `ID_AA64PFR0_EL1.SVE` must also contain a non-zero value.

C2.3.2 ID_AA64ZFR0_EL1

- R_{SYRGK}** The `ID_AA64ZFR0_EL1` *Purpose* is modified to state that it provides information about the implemented features of the AArch64 Scalable Vector Extension instruction set when any of `ID_AA64PFR0_EL1.SVE` and `ID_AA64PFR1_EL1.SME` are non-zero.
- R_{JLSQK}** If SME is implemented, then `ID_AA64ZFR0_EL1.SVEver` has the value `0b0001`, indicating that *legal* SVE and SVE2 instructions can be executed when the PE is in *Streaming SVE mode*.
- R_{RFZRM}** If SME is implemented and `ID_AA64PFR0_EL1.SVE` is non-zero, then FEAT_SVE2 is implemented and SVE and SVE2 instructions can be executed when the PE is not in *Streaming SVE mode*.
- R_{XFFLH}** If SME is implemented and `ID_AA64PFR0_EL1.SVE` is zero, then FEAT_SVE is not implemented and the `ID_AA64ZFR0_EL1` fields named `F64MM`, `F32MM`, `SM4`, `SHA3`, `BitPerm`, and `AES` hold the value zero.

See also:

- `ID_AA64PFR0_EL1`, defined in *Arm® Architecture Registers Armv9, for Armv9-A architecture profile* [2].
- [ID_AA64PFR1_EL1](#)
- [ID_AA64ZFR0_EL1](#)

C2.3.3 CPACR_EL1

- R_{QBTKS}** If SME is implemented, the field `CPACR_EL1.SMEN` is defined at bits [25:24]. For more information, see [CPACR_EL1.SMEN](#).
- I_{ZNSLS}** The set of SME-related instructions trapped by this control is defined by rule [D_{DMBHW}](#) in [C2.1.2 Traps and exceptions](#).

See also:

- [ESR_EL1](#), [ESR_EL2](#), and [ESR_EL3](#)
- [C2.1.2 Traps and exceptions](#)
- [C2.2.1 Exception priorities](#)
- [C2.2.3 Validity of SME and SVE state](#)

C2.3.4 CPTR_EL2

- R_{ZXZZC}** If SME is implemented, FEAT_VHE is implemented, and `HCR_EL2.E2H` is 1, the field `CPTR_EL2.SMEN` is defined at bits [25:24]. For more information, see [CPTR_EL2.SMEN](#).
- R_{QLKFH}** When SME is implemented, FEAT_VHE is implemented, and `HCR_EL2.E2H` is 0, the field `CPTR_EL2.TSM` is defined at bit [12]. For more information, see [CPTR_EL2.TSM](#).
- I_{DYLZC}** The set of SME-related instructions trapped by this control is defined by rule [D_{DMBHW}](#) in [C2.1.2 Traps and exceptions](#).

See also:

- [ESR_EL1, ESR_EL2, and ESR_EL3](#)
- [C2.1.2 Traps and exceptions](#)
- [C2.2.1 Exception priorities](#)
- [C2.2.3 Validity of SME and SVE state](#)

C2.3.5 CPTR_EL3

R_{NPVSR} If SME is implemented, the field `CPTR_EL3.ESM` is defined at bit [12]. For more information, see [CPTR_EL3.ESM](#).

I_{LGJZY} The set of SME-related instructions trapped by this control is defined by rule [D_{DMBHW}](#) in [C2.1.2 Traps and exceptions](#).

See also:

- [ESR_EL1, ESR_EL2, and ESR_EL3](#)
- [C2.1.2 Traps and exceptions](#)
- [C2.2.1 Exception priorities](#)
- [C2.2.3 Validity of SME and SVE state](#)

C2.3.6 HCR_EL2

R_{BWHVR} If SME is implemented, the `HCR_EL2.TID3` field causes accesses to `ID_AA64SMFR0_EL1` to be trapped. EL1 reads of `ID_AA64SMFR0_EL1` are trapped to EL2, reported using EC syndrome value `0x18`.

R_{ZRBBT} If SME is implemented, the `HCR_EL2.TID1` field causes accesses to `SMIDR_EL1` to be trapped. EL1 reads of `SMIDR_EL1` are trapped to EL2, reported using EC syndrome value `0x18`.

C2.3.7 HCRX_EL2

R_{YDHJV} If SME is implemented, the field `HCRX_EL2.SMPME` is defined at bit [5]. For more information, see [HCRX_EL2.SMPME](#).

See also:

- [SMPRI_EL1](#)
- [SMPRMAP_EL2](#)
- [C2.2.4 Streaming execution priority for shared implementations](#)

C2.3.8 SCR_EL3

R_{TCPTK} If SME is implemented, the field `SCR_EL3.EnTP2` is defined at bit [41]. For more information, see [SCR_EL3.EnTP2](#).

See also:

- [TPIDR2_EL0](#)

C2.3.9 SCTL_EL1

R_{NMVMQ} If SME is implemented, the field `SCTL_EL1.EnTP2` is defined at bit [60]. For more information, see [SCTL_EL1.EnTP2](#).

R_{MXHMD} When EL2 is implemented and enabled in the current Security state and `HCR_EL2.{E2H, TGE}` is `{1, 1}`, the `SCTL_EL1.EnTP2` control has no effect on execution at EL0 and the `SCTL_EL2.EnTP2` control is used for this purpose.

See also:

- [TPIDR2_EL0](#)

C2.3.10 SCTL2_EL2

R_KGMHQ If SME is implemented and `HCR_EL2.{E2H, TGE}` is `{1, 1}`, the field `SCTL2_EL2.EnTP2` is defined at bit [60]. For more information, see [SCTL2_EL2.EnTP2](#).

See also:

- [TPIDR2_EL0](#)

C2.3.11 HFGTR_EL2

R_SXDSB If SME is implemented, the fields `HFGTR_EL2.nTPIDR2_EL0` and `HFGTR_EL2.nSMPRI_EL1` are defined at bits [55] and [54]. For more information, see:

- [HFGTR_EL2.nTPIDR2_EL0](#)
- [HFGTR_EL2.nSMPRI_EL1](#)

See also:

- [TPIDR2_EL0](#)
- [SMPRI_EL1](#)

C2.3.12 HFGWTR_EL2

R_XKLBN If SME is implemented, the fields `HFGWTR_EL2.nTPIDR2_EL0` and `HFGWTR_EL2.nSMPRI_EL1` are defined at bits [55] and [54]. For more information, see:

- [HFGWTR_EL2.nTPIDR2_EL0](#)
- [HFGWTR_EL2.nSMPRI_EL1](#)

See also:

- [TPIDR2_EL0](#)
- [SMPRI_EL1](#)

C2.3.13 ESR_EL1, ESR_EL2, and ESR_EL3

D_DZSWB If SME is implemented, an Exception Class value `0x1D` is added to `ESR_EL1`, `ESR_EL2`, and `ESR_EL3`, for exceptions taken from AArch64.

EC	Meaning	ISS[2:0]
0b011101	Access to SME functionality trapped as a result of <code>CPACR_EL1.SMEN</code> , <code>CPTR_EL2.SMEN</code> , <code>CPTR_EL2.TSM</code> , or <code>CPTR_EL3.ESM</code> , that is not reported using EC <code>0b000000</code> .	0b000
0b011101	Illegal Advanced SIMD, SVE, or SVE2 instruction trapped because <code>PSTATE.SM</code> is 1	0b001
0b011101	Illegal SME instruction trapped because <code>PSTATE.SM</code> is 0	0b010
0b011101	Illegal SME instruction trapped because <code>PSTATE.ZA</code> is 0	0b011
0b011101	Reserved	0b1xx

R_{DYSFV} If SME is implemented, then the following field is defined in the ESR_{EL1} , ESR_{EL2} , and ESR_{EL3} ISS encoding for an exception from a Data Abort, when the EC value is $0b100100$ ($0x24$) or $0b100101$ ($0x25$):

Field	Name	Meaning
[15]	FnP	<p>When ISV == 0: FAR not Precise. 0b0 The FAR holds the faulting virtual address that generated the Data Abort. 0b1 The FAR holds any address within the <i>naturally-aligned fault granule</i> (see D_{BRGHW}) which contains the faulting virtual address that generated a Data Abort exception caused by an SVE contiguous vector load/store instruction when the PE is in Streaming SVE mode, or by an SME load/store instruction. On a Warm reset, this field resets to an architecturally UNKNOWN value.</p>

C2.3.14 ZCR_EL1, ZCR_EL2, and ZCR_EL3

I_{CRKQJ} If FEAT_SVE is implemented, then the ZCR_{ELx} registers have their described effect on the Effective SVE vector length only when the PE is not in *Streaming SVE mode*.

C2.4 SME-specific System registers

C2.4.1 ID_AA64SMFR0_EL1

The AArch64 SME Feature ID Register describes the set of implemented SME data-processing instructions.

D_{GRPHH} If SME is implemented, the register `ID_AA64SMFR0_EL1` is added. For more information, see [ID_AA64SMFR0_EL1](#).

C2.4.2 SMCR_EL1

The Streaming SVE Mode Control Register for EL1 configures the Effective Streaming SVE vector length when the PE is in *Streaming SVE mode* and executing at EL1 or EL0.

R_{HRTZQ} If SME is implemented, the register `SMCR_EL1` is added. For more information, see [SMCR_EL1](#).

R_{NWXVG} When EL2 is implemented and enabled in the current Security state and `HCR_EL2.{E2H, TGE}` is `{1, 1}`, the `SMCR_EL1` register has no effect on execution at EL0 and EL1 and the `SMCR_EL2` register is used for this purpose.

C2.4.3 SMCR_EL2

The Streaming Mode Control Register for EL2 configures the Effective SVE Streaming SVE vector length when the PE is in *Streaming SVE mode* and executing at EL2, and at EL1 or EL0 in the same Security state as EL2.

R_{JPZPH} If SME is implemented, the register `SMCR_EL2` is added. For more information, see [SMCR_EL2](#).

C2.4.4 SMCR_EL3

The Streaming Mode Control Register for EL3 configures the Effective Streaming SVE vector length when the PE is in *Streaming SVE mode* and executing at EL3, EL2, EL1, or EL0.

R_{DBGWC} If SME is implemented, the register `SMCR_EL3` is added. For more information, see [SMCR_EL3](#).

C2.4.5 SVCR

The Streaming Vector Control Register provides direct access to the `PSTATE.SM` and `PSTATE.ZA` mode bits from any Exception level.

D_{JXVQJ} If SME is implemented, the register `SVCR` is added. For more information, see [SVCR](#).

See also:

- [B1.1.1.3 Changing PSTATE.SM and PSTATE.ZA](#)

C2.4.6 SMPRI_EL1

The Streaming Mode Priority register configures the streaming execution priority for instructions executed in *Streaming SVE mode* on a shared SMCU at any Exception level.

R_{JKMFH} If SME is implemented, the register `SMPRI_EL1` is added. For more information, see [SMPRI_EL1](#).

R_{DWGZP} In an implementation that shares execution resources between PEs, higher streaming execution priority values are allocated more processing resource than other PEs configured with lower streaming execution priority values in the same *Priority domain*.

R_{DFQLX} The precise meaning and behavior of each streaming execution priority value is IMPLEMENTATION DEFINED.

I_{TCJVV}	Arm expects that, under contention for SMCU resources, PEs assigned higher streaming execution priority values achieve higher performance relative to PEs with lower streaming execution priority values.
R_{GLKXP}	PEs that have equal streaming execution priority values in the same <i>Priority domain</i> are allocated equal resources by the SMCU.
I_{BLMYK}	If system software does not support differentiation of streaming execution priority of threads, it is safe to use a value of 0 for all threads.
R_{SRMZT}	The streaming execution priority of all PEs in the <i>Priority domain</i> have no effect when there is no contention for an SMCU. An uncontended client PE request achieves full performance.
R_{SBCRG}	All SMCUs in the system have a consistent interpretation of the streaming execution priority values.

C2.4.7 SMPRMAP_EL2

The Streaming Mode Priority Mapping register maps the current virtual streaming execution priority value to a physical streaming execution priority value for instructions executed in *Streaming SVE mode* on a shared SMCU at EL1 or EL0 in the same Security states as EL2.

D_{CHVZD}	If SME is implemented, the register <code>SMPRMAP_EL2</code> is added. For more information, see SMPRMAP_EL2 . <code>SMPRMAP_EL2</code> contains 16 equally spaced fields of 4 bits. Fields are numbered 0 to 15 upwards in sequence starting from field 0 at bits [3:0].
-------------	--

C2.4.8 SMIDR_EL1

The Streaming Mode Identification Register provides additional information about the *Streaming SVE mode* implementation.

D_{NBDMK}	If SME is implemented, the register <code>SMIDR_EL1</code> is added. For more information, see SMIDR_EL1 .
-------------	--

C2.4.9 TPIDR2_EL0

The Software Thread ID Register #2 provides additional thread identifying information that can be read and written from all Exception levels.

D_{FJMMT}	If SME is implemented, the register <code>TPIDR2_EL0</code> is added. For more information, see TPIDR2_EL0 .
I_{QPMJN}	This register is reserved for use by the ABI to manage per-thread SME context.

Chapter C3

Interaction with other Armv9-A architectural features

C3.1 Overview

This section describes the interaction of SME with other aspects and features of the Armv9-A architecture.

It covers:

- Self-hosted debug.
- External debug.
- Memory Tagging Extension (MTE).
- Reliability, Availability, and Serviceability (RAS).
- Transactional Memory Extension (TME).
- Memory Partitioning and Monitoring (MPAM).

See also:

- *Arm[®] Architecture Reference Manual for A-profile architecture* [1].
- *Arm[®] Architecture Reference Manual Supplement Armv9, for A-profile architecture* [4].
- *Arm[®] Reliability, Availability, and Serviceability (RAS) Specification, for A-profile architecture* [5].
- *Arm[®] Architecture Reference Manual Supplement, Memory System Resource Partitioning and Monitoring (MPAM), for A-profile architecture* [6].

C3.2 Other architectural features

C3.2.1 Watchpoints

R_{PDGZL}	For a memory access or set of contiguous memory accesses generated by an SVE contiguous vector load/store instruction when the PE is in <i>Streaming SVE mode</i> , or by an SME load/store instruction, if a watchpoint matches a range where the lowest accessed address is rounded down to the nearest multiple of 16 bytes and the highest accessed address is rounded up to the nearest multiple of 16 bytes minus 1, but the watchpoint does not match the range of the original access or set of contiguous accesses, then it is CONstrained UNPREDICTABLE whether or not a Watchpoint debug event is triggered.
R_{XKRPV}	If a watchpoint matches only the rounded access address ranges of Inactive elements in a predicated vector load/store instruction, then it does not trigger a Watchpoint debug event.
I_{YVSFL}	If a Watchpoint debug event is triggered by a match on a rounded access address range that would not have been triggered by the original access address range, then this may report a false-positive match. Debug software must attempt to detect and step over false-positive matches. The architecture does not permit missed, or false-negative matches.

C3.2.1.1 Reporting watchpoints

R_{KDRCX}	If SME is implemented, then the following fields are added to ESR_{EL1} and ESR_{EL2} in the ISS encoding for an exception from a Watchpoint exception, when the EC value is $0x24$ or $0x25$:
-------------	---

Field	Name	Meaning
[24]	ISV	$RES0$
[23:18]	WPT	Watchpoint number, 0 to 15 inclusive. All other values are reserved.
[17]	WPTV	Watchpoint number Valid. 0b0 The WPT field is invalid, and holds an UNKNOWN value. 0b1 The WPT field is valid, and holds the number of a watchpoint that triggered a Watchpoint debug event.
[16]	WPF	Watchpoint might be false-positive. 0b0 The watchpoint matched the original access or set of contiguous accesses. 0b1 The watchpoint matched an access or set of contiguous accesses where the lowest accessed address was rounded down to the nearest multiple of 16 bytes and the highest accessed address was rounded up to the nearest multiple of 16 bytes minus 1, but the watchpoint might not have matched the original access or set of contiguous accesses.
[15]	FnP	FAR not Precise. This field only has meaning if the FAR is valid; that is, when the FnV field is 0. If the FnV field is 1, the FnP field is 0. 0b0 If the FnV field is 0, the FAR holds the virtual address of an access or set of contiguous accesses that triggered a Watchpoint debug event. 0b1 The FAR holds any address within the smallest implemented translation granule that contains the virtual address of an access or set of contiguous accesses that triggered a Watchpoint debug event.

Field	Name	Meaning
[10]	FnV	FAR not Valid. 0b0 The <code>FAR</code> is valid, and its value is as described by the <code>FnP</code> field. 0b1 The <code>FAR</code> is invalid, and holds an UNKNOWN value.

`R_MFRPZ` If SME is implemented, then the following field is added to the `EDDEVID1` register:

Field	Name	Meaning
[7:4]	HSR	Indicates support for the External Debug Halt Status Register (<code>EDHSR</code>). Defined values are: 0b0000 <code>EDHSR</code> not implemented, and the PE follows behaviors consistent with all of the <code>EDHSR</code> fields having a zero value. 0b0001 <code>EDHSR</code> implemented. All other values are reserved. If <code>FEAT_SME</code> is implemented, the permitted values are <code>0b0000</code> and <code>0b0001</code> . If <code>FEAT_SME</code> is not implemented, the only permitted value is <code>0b0000</code> .

`R_QBKWY` If SME is implemented, then the read-only External Debug Halt Status Register (`EDHSR`) may be implemented at offset `0x038`. The field `EDDEVID1.HSR1` indicates whether the `EDHSR` is implemented.

`R_LXGXN` If the `EDHSR` is implemented, it is in the Core power domain.

`R_SDSFM` If the `EDHSR` is implemented, then it is only valid when the PE is in Debug state and `EDSCR.STATUS` indicates a Watchpoint debug event (`0b101011`), otherwise it has an UNKNOWN value.

The `EDHSR` fields are defined as follows:

Field	Name	Meaning
[23:18]	WPT	Watchpoint number, 0 to 15 inclusive. All other values are reserved.
[17]	WPTV	Watchpoint number Valid. 0b0 The <code>WPT</code> field is invalid, and holds an UNKNOWN value. 0b1 The <code>WPT</code> field is valid, and holds the number of a watchpoint that triggered a Watchpoint debug event. On a Cold reset, this field resets to an architecturally UNKNOWN value.
[16]	WPF	Watchpoint might be false-positive. 0b0 The watchpoint matched the original access or set of contiguous accesses. 0b1 The watchpoint matched an access or set of contiguous accesses where the lowest accessed address was rounded down to the nearest multiple of 16 bytes and the highest accessed address was rounded up to the nearest multiple of 16 bytes minus 1, but the watchpoint might not have matched the original access or set of contiguous accesses. On a Cold reset, this field resets to an architecturally UNKNOWN value.

Field	Name	Meaning
[15]	FnP	<p>FAR not Precise.</p> <p>This field only has meaning if the <code>EDWAR</code> is valid; that is, when the <code>FnV</code> field is 0. If the <code>FnV</code> field is 1, the <code>FnP</code> field is 0.</p> <p>0b0 If the <code>FnV</code> field is 0, the <code>EDWAR</code> holds the virtual address of an access or set of contiguous accesses that triggered a Watchpoint debug event.</p> <p>0b1 The <code>EDWAR</code> holds any address within the smallest implemented translation granule that contains the virtual address of an access or set of contiguous accesses that triggered a Watchpoint debug event.</p> <p>On a Cold reset, this field resets to an architecturally UNKNOWN value.</p>
[10]	FnV	<p>FAR not Valid.</p> <p>0b0 The <code>EDWAR</code> is valid, and its value is as described by the <code>FnP</code> field.</p> <p>0b1 The <code>EDWAR</code> is invalid, and holds an UNKNOWN value.</p> <p>On a Cold reset, this field resets to an architecturally UNKNOWN value.</p>

<code>R_XWDJT</code>	If the <code>EDHSR</code> is not implemented, then the PE must follow behaviors consistent with all of the <code>EDHSR</code> fields having a zero value.
<code>R_CXZCY</code>	<p>If a Watchpoint debug event is triggered by an SVE contiguous load/store instruction when the PE is in <i>Streaming SVE mode</i>, or by an SME load/store instruction, then a virtual address recorded in <code>FAR_ELx</code> or <code>EDWAR</code> must be derived from an address that is both:</p> <ul style="list-style-type: none"> • In the inclusive range between: <ul style="list-style-type: none"> – The lowest address accessed by the vector instruction that triggered the watchpoint. – The highest watchpointed address accessed by the vector instruction that triggered the watchpoint. • Within a naturally-aligned block of memory.
<code>R_SQDKJ</code>	<p>If a watchpoint matches an access or set of contiguous accesses where the lowest accessed address was rounded down to the nearest multiple of 16 bytes and the highest accessed address was rounded up to the nearest multiple of 16 bytes minus 1, but the watchpoint might not have matched the original address of the access or set of contiguous accesses, the PE:</p> <ul style="list-style-type: none"> • Sets <code>ESR_ELx.WPF</code> to 1, on taking a Watchpoint exception generated by the watchpoint match. • Sets <code>EDHSR.WPF</code> to 1, on entering Debug state on a Watchpoint debug event generated by the watchpoint match. • Otherwise, <code>ESR_ELx.WPF</code> or <code>EDHSR.WPF</code> (as applicable) is set to 0.
<code>R_KSSHC</code>	<p>If a watchpoint matches an access that is due to an SVE contiguous load/store instruction when the PE is in <i>Streaming SVE mode</i>, or is due to an SME load/store instruction, then the PE:</p> <ul style="list-style-type: none"> • Sets <code>ESR_ELx.FnV</code> to an IMPLEMENTATION DEFINED value, 0 or 1, on taking a Watchpoint exception generated by the watchpoint match. • Sets <code>EDHSR.FnV</code> to an IMPLEMENTATION DEFINED value, 0 or 1, on entering Debug state on a Watchpoint debug event generated by the watchpoint match. • Otherwise, <code>ESR_ELx.FnV</code> or <code>EDHSR.FnV</code> (as applicable) is set to 0.
<code>R_RLWSF</code>	<p>When the PE sets <code>ESR_ELx.FnV</code> to 0 on taking a Watchpoint exception generated by the watchpoint match:</p> <ul style="list-style-type: none"> • If the lowest watchpointed address higher than or the same as the address recorded in <code>FAR_ELx</code> might not have been accessed by the instruction, other than as permitted by R_PDGZL, then the PE sets <code>ESR_ELx.FnP</code> to 1. • Otherwise, the PE sets <code>ESR_ELx.FnP</code> to 0.
<code>R_CJWYX</code>	<p>When the PE sets <code>EDHSR.FnV</code> to 0 on entering Debug state on a Watchpoint debug event generated by a watchpoint match:</p> <ul style="list-style-type: none"> • If the lowest watchpointed address higher than or the same as the address recorded in <code>EDWAR</code> might not have been accessed by the instruction, other than as permitted by R_PDGZL, then the PE sets <code>EDHSR.FnP</code> to 1.

- Otherwise, the PE sets `EDHSR.FnP` to 0.

`R_DTWTH`

When a Watchpoint debug event is triggered by a watchpoint match:

- If the PE sets `ESR_ELx.FnV` to 1 or `ESR_ELx.FnP` to 1 on taking a Watchpoint exception generated by the watchpoint match, then the PE sets `ESR_ELx.WPTV` to 1.
- If the PE sets `ESR_ELx.FnV` to 0 and `ESR_ELx.FnP` to 0 on taking a Watchpoint exception generated by the watchpoint match, then the PE sets `ESR_ELx.WPTV` to an IMPLEMENTATION DEFINED value, 0 or 1.
- If the PE sets `EDHSR.FnV` to 1 or `EDHSR.FnP` to 1 on entering Debug state on a Watchpoint debug event generated by the watchpoint match, then the PE sets `EDHSR.WPTV` to 1.
- If the PE sets `EDHSR.FnV` to 0 and `EDHSR.FnP` to 0 on entering Debug state on a Watchpoint debug event generated by the watchpoint match, then the PE sets `EDHSR.WPTV` to an IMPLEMENTATION DEFINED value, 0 or 1.

`R_PVYNL`

On a watchpoint match generated by watchpoint `<n>`:

- If the PE sets `ESR_ELx.WPTV` to 1 on taking a Watchpoint exception generated by the watchpoint match, then `ESR_ELx.WPT` is set to `<n>`.
- If the PE sets `EDHSR.WPTV` to 1 on entering Debug state on a Watchpoint debug event generated by the watchpoint match, then `EDHSR.WPT` is set to `<n>`.
- Otherwise, `ESR_ELx.WPT` or `EDHSR.WPT` (as applicable) is UNKNOWN.

`R_KHSFH`

When an instruction generates multiple watchpoint matches and the PE sets `ESR_ELx.WPTV` or `EDHSR.WPTV` to 1, then it is UNPREDICTABLE which matched watchpoint is reported in `ESR_ELx.WPT` or `EDHSR.WPT` (as applicable).

`D_LXZPC`

The *naturally-aligned block of memory* is all of the following:

- A power-of-two size.
- No larger than the `DC ZVA` block size if `ESR_ELx.FnP` or `EDHSR.FnP` (as appropriate) is 0.
- No larger than the smallest implemented translation granule if `ESR_ELx.FnP` or `EDHSR.FnP` (as appropriate) is 1.
- Contains a [watchpointed address](#) accessed by the memory access or set of contiguous memory accesses that triggered the watchpoint.

The size of the block is IMPLEMENTATION DEFINED.

There is no architectural means of discovering the size.

`D_LTGKY`

A *watchpointed address* is an address that a watchpoint is watching.

C3.2.2 Self-hosted debug

`I_CDBZX`

SME has no additional effect on self-hosted debug.

C3.2.3 External debug

`R_XQQRS`

The following SME-related instructions are unchanged in Debug state:

- `MOVA` (array to vector).
- `MOVA` (vector to array).
- `MRS SVCR`.
- `MSR SVCR`.
- `RDSVL`.

All other SME-related instructions are CONSTRAINED UNPREDICTABLE in Debug state, with the same set of CONSTRAINED UNPREDICTABLE options as other instructions in Debug state, as defined in *Arm® Architecture Reference Manual for A-profile architecture* [1].

C3.2.4 Memory Tagging Extension (MTE)

The following rules apply when the optional FEAT_MTE feature is implemented.

R_{BGGMD}	When the Memory Tagging Extension is implemented, it is IMPLEMENTATION DEFINED whether memory accesses due to SME, SVE, and SIMD&FP load and store instructions executed when the PE is in <i>Streaming SVE mode</i> will perform a Tag Check.
R_{GLYMK}	When the Memory Tagging Extension is implemented, it is IMPLEMENTATION DEFINED whether memory accesses due to SME <code>LDR</code> and <code>STR</code> instructions that access the SME ZA array vectors will perform a Tag Check.
I_{RBPMT}	An implementation of FEAT_MTE is only expected to perform Tag Checking when the PE is in <i>Streaming SVE mode</i> if it can do so with a similar relative performance impact to Tag Checking memory accesses due to SVE and SIMD&FP load and store instructions executed when the PE is not in <i>Streaming SVE mode</i> .

C3.2.5 Reliability, Availability, and Serviceability (RAS)

R_{RVYHY}	Rules I_{NTXKV} and R_{NQDWB} in Arm® <i>Reliability, Availability, and Serviceability (RAS) Specification, for A-profile architecture</i> [5] are extended by adding the SME ZA storage to any list of program-visible architectural state or registers that includes the SIMD&FP or SVE registers.
-------------	--

C3.2.6 Memory Partitioning and Monitoring (MPAM)

The following System registers are modified or added when the optional FEAT_MPAM feature is implemented.

C3.2.6.1 MPAMSM_EL1

If FEAT_MPAM is implemented, the MPAM Streaming Mode register is added by SME to generate MPAM labels for memory requests issued at any Exception level by SME load/store instructions and, when the PE is in *Streaming SVE mode*, SVE and SIMD&FP load/store instructions and SVE prefetch instructions.

R_{NXYP}	If SME and FEAT_MPAM are implemented, the register <code>MPAMSM_EL1</code> is added. For more information, see MPAMSM_EL1 .
------------	---

C3.2.6.2 MPAM2_EL2

R_{LJVWP}	If SME, FEAT_MPAM, and EL2 are implemented, the field <code>MPAM2_EL2.EnMPAMSM</code> is defined at bit [50]. For more information, see MPAM2_EL2.EnMPAMSM .
-------------	--

C3.2.7 Transactional Memory Extension (TME)

The following rules apply when the optional FEAT_TME feature is implemented.

R_{KHVVR}	Executing a <code>TSTART</code> instruction when <code>PSTATE.SM</code> is 1 fails the transaction with the <code>ERR</code> cause.
R_{LYBMR}	Executing an SME <code>LDR</code> , <code>STR</code> , or <code>ZERO</code> instruction that accesses the SME ZA array while in Transactional state will cause the transaction to fail with the <code>ERR</code> cause.
I_{TNZSW}	Any MSR instruction that writes to the <code>PSTATE.SM</code> or <code>PSTATE.ZA</code> bits in Transactional state, including the <code>SMSTART</code> and <code>SMSTOP</code> aliases, are UNDEFINED according to the rules in Arm® <i>Architecture Reference Manual Supplement Armv9, for A-profile architecture</i> [4] and will cause the transaction to fail with the <code>ERR</code> cause, without trapping. For more information about the rules, see the “MSR (register)” and “MSR (immediate)” sections in <i>The Transactional Memory Extension</i> chapter of Arm® <i>Architecture Reference Manual Supplement Armv9, for A-profile architecture</i> [4].

C3.2.8 Memory consistency model

R_BQSCG

SME and Streaming SVE memory accesses are subjected to the same ordering rules as existing SVE memory accesses, defined in *Arm® Architecture Reference Manual for A-profile architecture* [\[1\]](#).

Part D

SME instruction set

Chapter D1

SME instructions

This chapter defines the instructions added to the A64 instruction set when SME is implemented.

This content is from the **2021-12** version of *Arm® A64 Instruction Set Architecture Armv9, for Armv9-A architecture profile* [\[3\]](#), which contains the definitive details of the instruction set.

D1.1 SME data-processing instructions

The following SME data-processing instructions are available when SME is implemented. The new SME instructions are identified by the presence of the FEAT_SME symbol, or a call to one of the `HaveSME` pseudocode functions.

D1.1.1 ADDHA

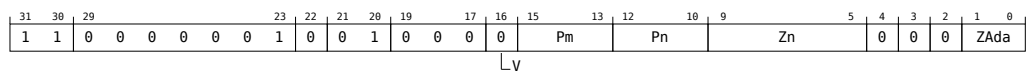
Add horizontally vector elements to ZA tile

Add each element of the source vector to the corresponding active element of each horizontal slice of a ZA tile. The tile elements are predicated by a pair of governing predicates. An element of a horizontal slice is considered active if its corresponding element in the second governing predicate is TRUE and the element corresponding to its horizontal slice number in the first governing predicate is TRUE. Inactive elements in the destination tile remain unmodified.

ID_AA64SMFR0_EL1.I16I64 indicates whether the 64-bit integer variant is implemented.

It has encodings from 2 classes: [32-bit](#) and [64-bit](#)

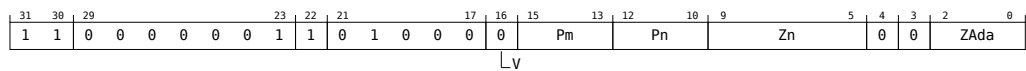
32-bit (FEAT_SME)



ADDHA `<ZAda>.S, <Pn>/M, <Pm>/M, <Zn>.S`

```
1 if !HaveSME() then UNDEFINED;
2 integer esize = 32;
3 integer a = UInt(Pn);
4 integer b = UInt(Pm);
5 integer n = UInt(Zn);
6 integer da = UInt(ZAda);
```

64-bit (FEAT_SME_I16I64)



ADDHA `<ZAda>.D, <Pn>/M, <Pm>/M, <Zn>.D`

```
1 if !HaveSMEI16I64() then UNDEFINED;
2 integer esize = 64;
3 integer a = UInt(Pn);
4 integer b = UInt(Pm);
5 integer n = UInt(Zn);
6 integer da = UInt(ZAda);
```

Assembler Symbols

- `<ZAda>` For the 32-bit variant: is the name of the ZA tile ZA0-ZA3, encoded in the "ZAda" field.
For the 64-bit variant: is the name of the ZA tile ZA0-ZA7, encoded in the "ZAda" field.
- `<Pn>` Is the name of the first governing scalable predicate register P0-P7, encoded in the "Pn" field.
- `<Pm>` Is the name of the second governing scalable predicate register P0-P7, encoded in the "Pm"

field.

<Zn> Is the name of the source scalable vector register, encoded in the "Zn" field.

Operation

```

1 CheckStreamingSVEAndZAAEnabled();
2 integer dim = VL DIV esize;
3 bits(PL) mask1 = P[a];
4 bits(PL) mask2 = P[b];
5 bits(VL) operand_src = Z[n];
6 bits(dim*dim*esize) operand_acc = ZAtile[da, esize];
7 bits(dim*dim*esize) result;
8
9 for col = 0 to dim-1
10     bits(esize) element = Elem[operand_src, col, esize];
11     for row = 0 to dim-1
12         bits(esize) res = Elem[operand_acc, row*dim+col, esize];
13         if ElemP[mask1, row, esize] == '1' && ElemP[mask2, col, esize] == '1' then
14             res = res + element;
15         Elem[result, row*dim+col, esize] = res;
16
17 ZAtile[da, esize] = result;

```

Operational information

If FEAT_SVE2 is implemented or FEAT_SME is implemented, then when PSTATE.DIT is 1:

- The execution time of this instruction is independent of:
 - The values of the data supplied in any of its operand registers when its governing predicate registers contain the same value for each execution.
 - The values of the NZCV flags.
- The response of this instruction to asynchronous exceptions does not vary based on:
 - The values of the data supplied in any of its operand registers when its governing predicate registers contain the same value for each execution.
 - The values of the NZCV flags.

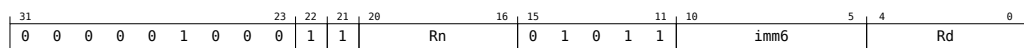
D1.1.2 ADDSPL

Add multiple of Streaming SVE predicate register size to scalar register

Add the Streaming SVE predicate register size in bytes multiplied by an immediate in the range -32 to 31 to the 64-bit source general-purpose register or current stack pointer and place the result in the 64-bit destination general-purpose register or current stack pointer.

This instruction does not require the PE to be in Streaming SVE mode.

SME
(FEAT_SME)



ADDSP <Xd|SP>, <Xn|SP>, #<imm>

```

1 if !HaveSME() then UNDEFINED;
2 integer n = UInt(Rn);
3 integer d = UInt(Rd);
4 integer imm = SInt(imm6);

```

Assembler Symbols

- <Xd|SP> Is the 64-bit name of the destination general-purpose register or stack pointer, encoded in the "Rd" field.
- <Xn|SP> Is the 64-bit name of the source general-purpose register or stack pointer, encoded in the "Rn" field.
- <imm> Is the signed immediate operand, in the range -32 to 31, encoded in the "imm6" field.

Operation

```

1 ChecksMEEnabled();
2 integer len = imm * (SVL DIV 64);
3 bits(64) operand1 = if n == 31 then SP[] else X[n];
4 bits(64) result = operand1 + len;
5
6 if d == 31 then
7     SP[] = result;
8 else
9     X[d] = result;

```

Operational information

If FEAT_SVE2 is implemented or FEAT_SME is implemented, then when PSTATE.DIT is 1:

- The execution time of this instruction is independent of:
 - The values of the data supplied in any of its registers.
 - The values of the NZCV flags.
- The response of this instruction to asynchronous exceptions does not vary based on:
 - The values of the data supplied in any of its registers.
 - The values of the NZCV flags.

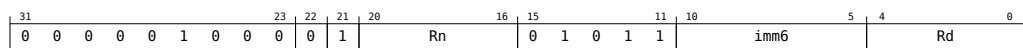
D1.1.3 ADDSVL

Add multiple of Streaming SVE vector register size to scalar register

Add the Streaming SVE vector register size in bytes multiplied by an immediate in the range -32 to 31 to the 64-bit source general-purpose register or current stack pointer, and place the result in the 64-bit destination general-purpose register or current stack pointer.

This instruction does not require the PE to be in Streaming SVE mode.

SME
(FEAT_SME)



ADDSVL <Xd|SP>, <Xn|SP>, #<imm>

```

1 if !HaveSME() then UNDEFINED;
2 integer n = UInt(Rn);
3 integer d = UInt(Rd);
4 integer imm = SInt(imm6);

```

Assembler Symbols

- <Xd|SP> Is the 64-bit name of the destination general-purpose register or stack pointer, encoded in the "Rd" field.
- <Xn|SP> Is the 64-bit name of the source general-purpose register or stack pointer, encoded in the "Rn" field.
- <imm> Is the signed immediate operand, in the range -32 to 31, encoded in the "imm6" field.

Operation

```

1 ChecksMEEnabled();
2 integer len = imm * (SVL DIV 8);
3 bits(64) operand1 = if n == 31 then SP[] else X[n];
4 bits(64) result = operand1 + len;
5
6 if d == 31 then
7     SP[] = result;
8 else
9     X[d] = result;

```

Operational information

If FEAT_SVE2 is implemented or FEAT_SME is implemented, then when PSTATE.DIT is 1:

- The execution time of this instruction is independent of:
 - The values of the data supplied in any of its registers.
 - The values of the NZCV flags.
- The response of this instruction to asynchronous exceptions does not vary based on:
 - The values of the data supplied in any of its registers.
 - The values of the NZCV flags.

D1.1.4 ADDVA

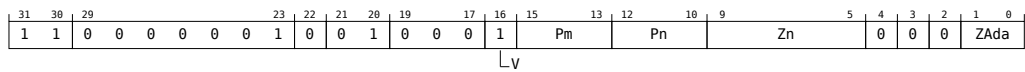
Add vertically vector elements to ZA tile

Add each element of the source vector to the corresponding active element of each vertical slice of a ZA tile. The tile elements are predicated by a pair of governing predicates. An element of a vertical slice is considered active if its corresponding element in the first governing predicate is TRUE and the element corresponding to its vertical slice number in the second governing predicate is TRUE. Inactive elements in the destination tile remain unmodified.

ID_AA64SMFR0_EL1.I16I64 indicates whether the 64-bit integer variant is implemented.

It has encodings from 2 classes: 32-bit and 64-bit

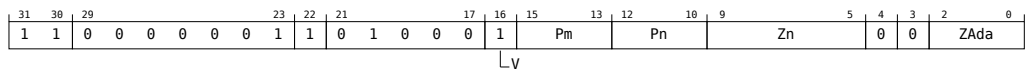
32-bit (FEAT_SME)



ADDVA <ZAda>.S, <Pn>/M, <Pm>/M, <Zn>.S

```
1 if !HaveSME() then UNDEFINED;
2 integer esize = 32;
3 integer a = UInt(Pn);
4 integer b = UInt(Pm);
5 integer n = UInt(Zn);
6 integer da = UInt(ZAda);
```

64-bit (FEAT_SME_I16I64)



ADDVA <ZAda>.D, <Pn>/M, <Pm>/M, <Zn>.D

```
1 if !HaveSMEI16I64() then UNDEFINED;
2 integer esize = 64;
3 integer a = UInt(Pn);
4 integer b = UInt(Pm);
5 integer n = UInt(Zn);
6 integer da = UInt(ZAda);
```

Assembler Symbols

- <ZAda> For the 32-bit variant: is the name of the ZA tile ZA0-ZA3, encoded in the "ZAda" field.
For the 64-bit variant: is the name of the ZA tile ZA0-ZA7, encoded in the "ZAda" field.
- <Pn> Is the name of the first governing scalable predicate register P0-P7, encoded in the "Pn" field.
- <Pm> Is the name of the second governing scalable predicate register P0-P7, encoded in the "Pm" field.
- <Zn> Is the name of the source scalable vector register, encoded in the "Zn" field.

Operation

```
1 CheckStreamingSVEAndZAAEnabled();
2 integer dim = VL DIV esize;
3 bits(PL) mask1 = P[a];
4 bits(PL) mask2 = P[b];
```

```
5  bits(VL) operand_src = Z[n];
6  bits(dim*dim*esize) operand_acc = ZAtile[da, esize];
7  bits(dim*dim*esize) result;
8
9  for row = 0 to dim-1
10     bits(esize) element = Elem[operand_src, row, esize];
11     for col = 0 to dim-1
12         bits(esize) res = Elem[operand_acc, row*dim+col, esize];
13         if ElemP[mask1, row, esize] == '1' && ElemP[mask2, col, esize] == '1' then
14             res = res + element;
15         Elem[result, row*dim+col, esize] = res;
16
17  ZAtile[da, esize] = result;
```

Operational information

If FEAT_SVE2 is implemented or FEAT_SME is implemented, then when PSTATE.DIT is 1:

- The execution time of this instruction is independent of:
 - The values of the data supplied in any of its operand registers when its governing predicate registers contain the same value for each execution.
 - The values of the NZCV flags.
- The response of this instruction to asynchronous exceptions does not vary based on:
 - The values of the data supplied in any of its operand registers when its governing predicate registers contain the same value for each execution.
 - The values of the NZCV flags.

D1.1.5 BFMOPA

BFloat16 sum of outer products and accumulate

The BFloat16 floating-point sum of outer products and accumulate instruction works with a 32-bit element ZA tile.

This instruction multiplies the $SVL_S \times 2$ sub-matrix of BFloat16 values held in the first source vector by the $2 \times SVL_S$ sub-matrix of BFloat16 values in the second source vector.

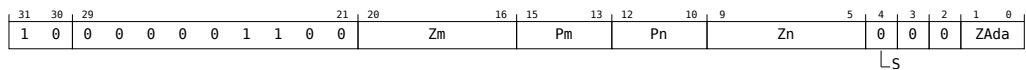
Each source vector is independently predicated by a corresponding governing predicate. When a 16-bit source element is Inactive it is treated as having the value +0.0, but if both pairs of source vector elements that correspond to a 32-bit destination element contain Inactive elements, then the destination element remains unmodified.

The resulting $SVL_S \times SVL_S$ single-precision floating-point sum of outer products is then destructively added to the single-precision floating-point destination tile. This is equivalent to performing a 2-way dot product and accumulate to each of the destination tile elements.

Each 32-bit container of first source vector holds 2 consecutive column elements of each row of a $SVL_S \times 2$ sub-matrix. Similarly, each 32-bit container of second source vector holds 2 consecutive row elements of each column of a $2 \times SVL_S$ sub-matrix.

This instruction follows SME BFloat16 numerical behaviors.

SME
(FEAT_SME)



BFMOPA <ZAda>.S, <Pn>/M, <Pm>/M, <Zn>.H, <Zm>.H

```
1 if !HaveSME() then UNDEFINED;
2 integer a = UInt(Pn);
3 integer b = UInt(Pm);
4 integer n = UInt(Zn);
5 integer m = UInt(Zm);
6 integer da = UInt(ZAda);
7 boolean sub_op = FALSE;
```

Assembler Symbols

- <ZAda> Is the name of the ZA tile ZA0-ZA3, encoded in the "ZAda" field.
- <Pn> Is the name of the first governing scalable predicate register P0-P7, encoded in the "Pn" field.
- <Pm> Is the name of the second governing scalable predicate register P0-P7, encoded in the "Pm" field.
- <Zn> Is the name of the first source scalable vector register, encoded in the "Zn" field.
- <Zm> Is the name of the second source scalable vector register, encoded in the "Zm" field.

Operation

```
1 CheckStreamingSVEAndZaEnabled();
2 integer dim = VL DIV 32;
3 bits(PL) mask1 = P[a];
4 bits(PL) mask2 = P[b];
5 bits(VL) operand1 = Z[n];
6 bits(VL) operand2 = Z[m];
7 bits(dim*dim*32) operand3 = ZAtila[da, 32];
8 bits(dim*dim*32) result;
9
10 for row = 0 to dim-1
11     for col = 0 to dim-1
12         // determine row/col predicates
13         boolean prow_0 = (ElemP[mask1, 2*row + 0, 16] == '1');
```

```

14     boolean prow_1 = (ElemP[mask1, 2*row + 1, 16] == '1');
15     boolean pcol_0 = (ElemP[mask2, 2*col + 0, 16] == '1');
16     boolean pcol_1 = (ElemP[mask2, 2*col + 1, 16] == '1');
17
18     bits(32) sum = Elem[operand3, row*dim+col, 32];
19     if (prow_0 && pcol_0) || (prow_1 && pcol_1) then
20         bits(16) erow_0 = (if prow_0 then Elem[operand1, 2*row + 0, 16] else FPZero('0'));
21         bits(16) erow_1 = (if prow_1 then Elem[operand1, 2*row + 1, 16] else FPZero('0'));
22         bits(16) ecol_0 = (if pcol_0 then Elem[operand2, 2*col + 0, 16] else FPZero('0'));
23         bits(16) ecol_1 = (if pcol_1 then Elem[operand2, 2*col + 1, 16] else FPZero('0'));
24         if sub_op then
25             if prow_0 then erow_0 = BFNeg(erow_0);
26             if prow_1 then erow_1 = BFNeg(erow_1);
27             sum = BFDotAdd(sum, erow_0, erow_1, ecol_0, ecol_1, FPCR[]);
28
29     Elem[result, row*dim+col, 32] = sum;
30
31     ZAtile[da, 32] = result;

```

D1.1.6 BFMOPS

BFloat16 sum of outer products and subtract

The BFloat16 floating-point sum of outer products and subtract instruction works with a 32-bit element ZA tile.

This instruction multiplies the $SVL_S \times 2$ sub-matrix of BFloat16 values held in the first source vector by the $2 \times SVL_S$ sub-matrix of BFloat16 values in the second source vector.

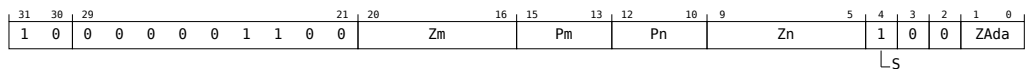
Each source vector is independently predicated by a corresponding governing predicate. When a 16-bit source element is Inactive it is treated as having the value +0.0, but if both pairs of source vector elements that correspond to a 32-bit destination element contain Inactive elements, then the destination element remains unmodified.

The resulting $SVL_S \times SVL_S$ single-precision floating-point sum of outer products is then destructively subtracted from the single-precision floating-point destination tile. This is equivalent to performing a 2-way dot product and subtract from each of the destination tile elements.

Each 32-bit container of first source vector holds 2 consecutive column elements of each row of a $SVL_S \times 2$ sub-matrix. Similarly, each 32-bit container of second source vector holds 2 consecutive row elements of each column of a $2 \times SVL_S$ sub-matrix.

This instruction follows SME BFloat16 numerical behaviors.

SME
(FEAT_SME)



BFMOPS <ZAda>.S, <Pn>/M, <Pm>/M, <Zn>.H, <Zm>.H

```
1 if !HaveSME() then UNDEFINED;
2 integer a = UInt(Pn);
3 integer b = UInt(Pm);
4 integer n = UInt(Zn);
5 integer m = UInt(Zm);
6 integer da = UInt(ZAda);
7 boolean sub_op = TRUE;
```

Assembler Symbols

- <ZAda> Is the name of the ZA tile ZA0-ZA3, encoded in the "ZAda" field.
- <Pn> Is the name of the first governing scalable predicate register P0-P7, encoded in the "Pn" field.
- <Pm> Is the name of the second governing scalable predicate register P0-P7, encoded in the "Pm" field.
- <Zn> Is the name of the first source scalable vector register, encoded in the "Zn" field.
- <Zm> Is the name of the second source scalable vector register, encoded in the "Zm" field.

Operation

```
1 CheckStreamingSVEAndZaEnabled();
2 integer dim = VL DIV 32;
3 bits(PL) mask1 = P[a];
4 bits(PL) mask2 = P[b];
5 bits(VL) operand1 = Z[n];
6 bits(VL) operand2 = Z[m];
7 bits(dim*dim*32) operand3 = ZAtile[da, 32];
8 bits(dim*dim*32) result;
9
10 for row = 0 to dim-1
11     for col = 0 to dim-1
12         // determine row/col predicates
13         boolean prow_0 = (ElemP[mask1, 2*row + 0, 16] == '1');
```

Chapter D1. SME instructions
D1.1. SME data-processing instructions

```
14      boolean prow_1 = (ElemP[mask1, 2*row + 1, 16] == '1');
15      boolean pcol_0 = (ElemP[mask2, 2*col + 0, 16] == '1');
16      boolean pcol_1 = (ElemP[mask2, 2*col + 1, 16] == '1');
17
18      bits(32) sum = Elem[operand3, row*dim+col, 32];
19      if (prow_0 && pcol_0) || (prow_1 && pcol_1) then
20          bits(16) erow_0 = (if prow_0 then Elem[operand1, 2*row + 0, 16] else FPZero('0'));
21          bits(16) erow_1 = (if prow_1 then Elem[operand1, 2*row + 1, 16] else FPZero('0'));
22          bits(16) ecol_0 = (if pcol_0 then Elem[operand2, 2*col + 0, 16] else FPZero('0'));
23          bits(16) ecol_1 = (if pcol_1 then Elem[operand2, 2*col + 1, 16] else FPZero('0'));
24          if sub_op then
25              if prow_0 then erow_0 = BFNeg(erow_0);
26              if prow_1 then erow_1 = BFNeg(erow_1);
27              sum = BFDotAdd(sum, erow_0, erow_1, ecol_0, ecol_1, FPCR[]);
28
29      Elem[result, row*dim+col, 32] = sum;
30
31  ZAtile[da, 32] = result;
```

D1.1.7 FMOPA (non-widening)

Floating-point outer product and accumulate

The single-precision variant works with a 32-bit element ZA tile.

The double-precision variant works with a 64-bit element ZA tile.

These instructions generate an outer product of the first source vector and the second source vector. In case of the single-precision variant, the first source is $SVL_S \times 1$ vector and the second source is $1 \times SVL_S$ vector. In case of the double-precision variant, the first source is $SVL_D \times 1$ vector and the second source is $1 \times SVL_D$ vector.

Each source vector is independently predicated by a corresponding governing predicate. When either source vector element is Inactive the corresponding destination tile element remains unmodified.

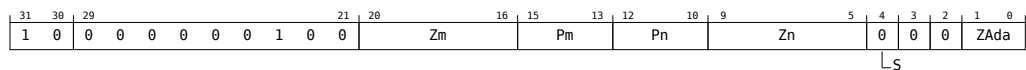
The resulting outer product, $SVL_S \times SVL_S$ in case of single-precision variant or $SVL_D \times SVL_D$ in case of double-precision variant, is then destructively added to the destination tile. This is equivalent to performing a single multiply-accumulate to each of the destination tile elements.

This instruction follows SME floating-point numerical behaviors.

ID_AA64SMFR0_EL1.F64F64 indicates whether the double-precision variant is implemented.

It has encodings from 2 classes: [Single-precision](#) and [Double-precision](#)

Single-precision (FEAT_SME)



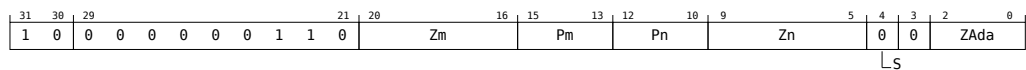
FMOPA <ZAda>.S, <Pn>/M, <Pm>/M, <Zn>.S, <Zm>.S

```

1 if !HaveSME() then UNDEFINED;
2 integer esize = 32;
3 integer a = UInt(Pn);
4 integer b = UInt(Pm);
5 integer n = UInt(Zn);
6 integer m = UInt(Zm);
7 integer da = UInt(ZAda);
8 boolean sub_op = FALSE;

```

Double-precision (FEAT_SME_F64F64)



FMOPA <ZAda>.D, <Pn>/M, <Pm>/M, <Zn>.D, <Zm>.D

```

1 if !HaveSMEF64F64() then UNDEFINED;
2 integer esize = 64;
3 integer a = UInt(Pn);
4 integer b = UInt(Pm);
5 integer n = UInt(Zn);
6 integer m = UInt(Zm);
7 integer da = UInt(ZAda);
8 boolean sub_op = FALSE;

```

Assembler Symbols

<ZAda> For the single-precision variant: is the name of the ZA tile ZA0-ZA3, encoded in the "ZAda" field.

For the double-precision variant: is the name of the ZA tile ZA0-ZA7, encoded in the "ZAda" field.

<Pn> Is the name of the first governing scalable predicate register P0-P7, encoded in the "Pn" field.

<Pm> Is the name of the second governing scalable predicate register P0-P7, encoded in the "Pm" field.

<Zn> Is the name of the first source scalable vector register, encoded in the "Zn" field.

<Zm> Is the name of the second source scalable vector register, encoded in the "Zm" field.

Operation

```

1  CheckStreamingSVEAndZAAEnabled();
2  integer dim = VL DIV esize;
3  bits(PL) mask1 = P[a];
4  bits(PL) mask2 = P[b];
5  bits(VL) operand1 = Z[n];
6  bits(VL) operand2 = Z[m];
7  bits(dim*dim*esize) operand3 = ZAtile[da, esize];
8  bits(dim*dim*esize) result;
9
10 for row = 0 to dim-1
11     for col = 0 to dim-1
12         bits(esize) element1 = Elem[operand1, row, esize];
13         bits(esize) element2 = Elem[operand2, col, esize];
14         bits(esize) element3 = Elem[operand3, row*dim+col, esize];
15
16         if ElemP[mask1, row, esize] == '1' && ElemP[mask2, col, esize] == '1' then
17             if sub_op then element1 = FPNeg(element1);
18             Elem[result, row*dim+col, esize] = FPMulAdd_ZA(element3, element1, element2, FPCR[]);
19         else
20             Elem[result, row*dim+col, esize] = element3;
21
22 ZAtile[da, esize] = result;

```


D1.1.8 FMOPA (widening)

Half-precision floating-point sum of outer products and accumulate

The half-precision floating-point sum of outer products and accumulate instruction works with a 32-bit element ZA tile.

This instruction widens the $SVL_S \times 2$ sub-matrix of half-precision floating-point values held in the first source vector to single-precision floating-point values and multiplies it by the widened $2 \times SVL_S$ sub-matrix of half-precision floating-point values in the second source vector to single-precision floating-point values.

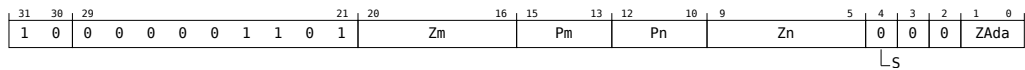
Each source vector is independently predicated by a corresponding governing predicate. When a 16-bit source element is Inactive it is treated as having the value +0.0, but if both pairs of source vector elements that correspond to a 32-bit destination element contain Inactive elements, then the destination element remains unmodified.

The resulting $SVL_S \times SVL_S$ single-precision floating-point sum of outer products is then destructively added to the single-precision floating-point destination tile. This is equivalent to performing a 2-way dot product and accumulate to each of the destination tile elements.

Each 32-bit container of first source vector holds 2 consecutive column elements of each row of a $SVL_S \times 2$ sub-matrix. Similarly, each 32-bit container of second source vector holds 2 consecutive row elements of each column of a $2 \times SVL_S$ sub-matrix.

This instruction follows SME floating-point numerical behaviors.

SME (FEAT_SME)



FMOPA <ZAda>.S, <Pn>/M, <Pm>/M, <Zn>.H, <Zm>.H

```

1 if !HaveSME() then UNDEFINED;
2 integer a = UInt(Pn);
3 integer b = UInt(Pm);
4 integer n = UInt(Zn);
5 integer m = UInt(Zm);
6 integer da = UInt(ZAda);
7 boolean sub_op = FALSE;

```

Assembler Symbols

- <ZAda> Is the name of the ZA tile ZA0-ZA3, encoded in the "ZAda" field.
- <Pn> Is the name of the first governing scalable predicate register P0-P7, encoded in the "Pn" field.
- <Pm> Is the name of the second governing scalable predicate register P0-P7, encoded in the "Pm" field.
- <Zn> Is the name of the first source scalable vector register, encoded in the "Zn" field.
- <Zm> Is the name of the second source scalable vector register, encoded in the "Zm" field.

Operation

```

1 CheckStreamingSVEAndZAAEnabled();
2 integer dim = VL DIV 32;
3 bits(PL) mask1 = P[a];
4 bits(PL) mask2 = P[b];
5 bits(VL) operand1 = Z[n];
6 bits(VL) operand2 = Z[m];
7 bits(dim*dim*32) operand3 = ZAtile[da, 32];
8 bits(dim*dim*32) result;
9
10 for row = 0 to dim-1

```

Chapter D1. SME instructions
D1.1. SME data-processing instructions

```
11  for col = 0 to dim-1
12      // determine row/col predicates
13      boolean prow_0 = (ElemP[mask1, 2*row + 0, 16] == '1');
14      boolean prow_1 = (ElemP[mask1, 2*row + 1, 16] == '1');
15      boolean pcol_0 = (ElemP[mask2, 2*col + 0, 16] == '1');
16      boolean pcol_1 = (ElemP[mask2, 2*col + 1, 16] == '1');
17
18      bits(32) sum = Elem[operand3, row*dim+col, 32];
19      if (prow_0 && pcol_0) || (prow_1 && pcol_1) then
20          bits(16) erow_0 = (if prow_0 then Elem[operand1, 2*row + 0, 16] else FPZero('0'));
21          bits(16) erow_1 = (if prow_1 then Elem[operand1, 2*row + 1, 16] else FPZero('0'));
22          bits(16) ecol_0 = (if pcol_0 then Elem[operand2, 2*col + 0, 16] else FPZero('0'));
23          bits(16) ecol_1 = (if pcol_1 then Elem[operand2, 2*col + 1, 16] else FPZero('0'));
24          if sub_op then
25              if prow_0 then erow_0 = FPNeg(erow_0);
26              if prow_1 then erow_1 = FPNeg(erow_1);
27              sum = FPDotAdd_ZA(sum, erow_0, erow_1, ecol_0, ecol_1, FPCR[]);
28
29      Elem[result, row*dim+col, 32] = sum;
30
31  ZAtile[da, 32] = result;
```

D1.1.9 FMOPS (non-widening)

Floating-point outer product and subtract

The single-precision variant works with a 32-bit element ZA tile.

The double-precision variant works with a 64-bit element ZA tile.

These instructions generate an outer product of the first source vector and the second source vector. In case of the single-precision variant, the first source is $SVL_S \times 1$ vector and the second source is $1 \times SVL_S$ vector. In case of the double-precision variant, the first source is $SVL_D \times 1$ vector and the second source is $1 \times SVL_D$ vector.

Each source vector is independently predicated by a corresponding governing predicate. When either source vector element is Inactive the corresponding destination tile element remains unmodified.

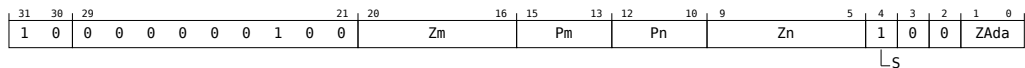
The resulting outer product, $SVL_S \times SVL_S$ in case of single-precision variant or $SVL_D \times SVL_D$ in case of double-precision variant, is then destructively subtracted from the destination tile. This is equivalent to performing a single multiply-subtract from each of the destination tile elements.

This instruction follows SME floating-point numerical behaviors.

ID_AA64SMFR0_EL1.F64F64 indicates whether the double-precision variant is implemented.

It has encodings from 2 classes: [Single-precision](#) and [Double-precision](#)

Single-precision (FEAT_SME)



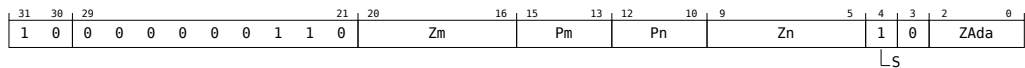
FMOPS <ZAda>.S, <Pn>/M, <Pm>/M, <Zn>.S, <Zm>.S

```

1 if !HaveSME() then UNDEFINED;
2 integer esize = 32;
3 integer a = UInt(Pn);
4 integer b = UInt(Pm);
5 integer n = UInt(Zn);
6 integer m = UInt(Zm);
7 integer da = UInt(ZAda);
8 boolean sub_op = TRUE;

```

Double-precision (FEAT_SME_F64F64)



FMOPS <ZAda>.D, <Pn>/M, <Pm>/M, <Zn>.D, <Zm>.D

```

1 if !HaveSMEF64F64() then UNDEFINED;
2 integer esize = 64;
3 integer a = UInt(Pn);
4 integer b = UInt(Pm);
5 integer n = UInt(Zn);
6 integer m = UInt(Zm);
7 integer da = UInt(ZAda);
8 boolean sub_op = TRUE;

```

Assembler Symbols

<ZAda> For the single-precision variant: is the name of the ZA tile ZA0-ZA3, encoded in the "ZAda" field.

For the double-precision variant: is the name of the ZA tile ZA0-ZA7, encoded in the "ZAda" field.

- <Pn> Is the name of the first governing scalable predicate register P0-P7, encoded in the "Pn" field.
- <Pm> Is the name of the second governing scalable predicate register P0-P7, encoded in the "Pm" field.
- <Zn> Is the name of the first source scalable vector register, encoded in the "Zn" field.
- <Zm> Is the name of the second source scalable vector register, encoded in the "Zm" field.

Operation

```

1  CheckStreamingSVEAndZAAEnabled();
2  integer dim = VL DIV esize;
3  bits(PL) mask1 = P[a];
4  bits(PL) mask2 = P[b];
5  bits(VL) operand1 = Z[n];
6  bits(VL) operand2 = Z[m];
7  bits(dim*dim*esize) operand3 = ZAtile[da, esize];
8  bits(dim*dim*esize) result;
9
10 for row = 0 to dim-1
11   for col = 0 to dim-1
12     bits(esize) element1 = Elem[operand1, row, esize];
13     bits(esize) element2 = Elem[operand2, col, esize];
14     bits(esize) element3 = Elem[operand3, row*dim+col, esize];
15
16     if ElemP[mask1, row, esize] == '1' && ElemP[mask2, col, esize] == '1' then
17       if sub_op then element1 = FPNeg(element1);
18       Elem[result, row*dim+col, esize] = FPMulAdd_ZA(element3, element1, element2, FPCR[]);
19     else
20       Elem[result, row*dim+col, esize] = element3;
21
22 ZAtile[da, esize] = result;
```

D1.1.10 FMOPS (widening)

Half-precision floating-point sum of outer products and subtract

The half-precision floating-point sum of outer products and subtract instruction works with a 32-bit element ZA tile.

This instruction widens the $SVL_S \times 2$ sub-matrix of half-precision floating-point values held in the first source vector to single-precision floating-point values and multiplies it by the widened $2 \times SVL_S$ sub-matrix of half-precision floating-point values in the second source vector to single-precision floating-point values.

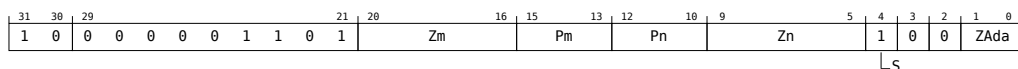
Each source vector is independently predicated by a corresponding governing predicate. When a 16-bit source element is Inactive it is treated as having the value +0.0, but if both pairs of source vector elements that correspond to a 32-bit destination element contain Inactive elements, then the destination element remains unmodified.

The resulting $\text{SVL}_S \times \text{SVL}_S$ single-precision floating-point sum of outer products is then destructively subtracted from the single-precision floating-point destination tile. This is equivalent to performing a 2-way dot product and subtract from each of the destination tile elements.

Each 32-bit container of first source vector holds 2 consecutive column elements of each row of a $\text{SVL}_S \times 2$ sub-matrix. Similarly, each 32-bit container of second source vector holds 2 consecutive row elements of each column of a $2 \times \text{SVL}_S$ sub-matrix.

This instruction follows SME floating-point numerical behaviors.

SME
(FEAT SME)



FMOPS <ZAda>.S, <Pn>/M, <Pm>/M, <Zn>.H, <Zm>.H

```

1  if !HaveSME() then UNDEFINED;
2  integer a = UInt(Pn);
3  integer b = UInt(Pm);
4  integer n = UInt(Zn);
5  integer m = UInt(Zm);
6  integer da = UInt(ZAda);
7  boolean sub_op = TRUE;

```

Assembler Symbols

- | | |
|--------|---|
| <ZAda> | Is the name of the ZA tile ZA0-ZA3, encoded in the "ZAda" field. |
| <Pn> | Is the name of the first governing scalable predicate register P0-P7, encoded in the "Pn" field. |
| <Pm> | Is the name of the second governing scalable predicate register P0-P7, encoded in the "Pm" field. |
| <Zn> | Is the name of the first source scalable vector register, encoded in the "Zn" field. |
| <Zm> | Is the name of the second source scalable vector register, encoded in the "Zm" field. |

Operation

```

1  CheckStreamingSVEAndZAAEnabled();
2  integer dim = VL DIV 32;
3  bits(PL) mask1 = P[a];
4  bits(PL) mask2 = P[b];
5  bits(VL) operand1 = Z[n];
6  bits(VL) operand2 = Z[m];
7  bits(dim*dim*32) operand3 = ZAtile[da, 32];
8  bits(dim*dim*32) result;
9
10 for row = 0 to dim-1

```

Chapter D1. SME instructions
D1.1. SME data-processing instructions

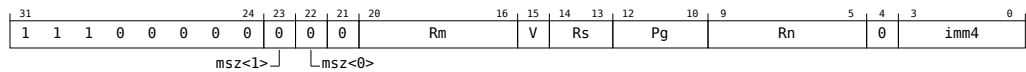
```
11  for col = 0 to dim-1
12      // determine row/col predicates
13      boolean prow_0 = (ElemP[mask1, 2*row + 0, 16] == '1');
14      boolean prow_1 = (ElemP[mask1, 2*row + 1, 16] == '1');
15      boolean pcol_0 = (ElemP[mask2, 2*col + 0, 16] == '1');
16      boolean pcol_1 = (ElemP[mask2, 2*col + 1, 16] == '1');
17
18      bits(32) sum = Elem[operand3, row*dim+col, 32];
19      if (prow_0 && pcol_0) || (prow_1 && pcol_1) then
20          bits(16) erow_0 = (if prow_0 then Elem[operand1, 2*row + 0, 16] else FPZero('0'));
21          bits(16) erow_1 = (if prow_1 then Elem[operand1, 2*row + 1, 16] else FPZero('0'));
22          bits(16) ecol_0 = (if pcol_0 then Elem[operand2, 2*col + 0, 16] else FPZero('0'));
23          bits(16) ecol_1 = (if pcol_1 then Elem[operand2, 2*col + 1, 16] else FPZero('0'));
24          if sub_op then
25              if prow_0 then erow_0 = FPNeg(erow_0);
26              if prow_1 then erow_1 = FPNeg(erow_1);
27              sum = FPDotAdd_ZA(sum, erow_0, erow_1, ecol_0, ecol_1, FPCR[]);
28
29      Elem[result, row*dim+col, 32] = sum;
30
31  ZAtile[da, 32] = result;
```

D1.1.11 LD1B

Contiguous load of bytes to 8-bit element ZA tile slice

The slice number within the tile is selected by the sum of the slice index register and an immediate, modulo the number of 8-bit elements in a Streaming SVE vector. The immediate is in the range 0 to 15. The memory address is generated by scalar base and optional scalar offset which is added to the base address. Inactive elements will not cause a read from Device memory or signal a fault, and are set to zero in the destination vector.

SME
(FEAT_SME)



```
LD1B { ZA0<HV>.B[<Ws>, <imm>] }, <Pg>/Z, [<Xn|SP>{, <Xm>}]
```

```
1 if !HaveSME() then UNDEFINED;
2 integer n = UInt(Rn);
3 integer m = UInt(Rm);
4 integer g = UInt('0':Pg);
5 integer s = UInt('011':Rs);
6 integer t = 0;
7 integer imm = UInt(imm4);
8 integer esize = 8;
9 boolean vertical = V == '1';
```

Assembler Symbols

<HV> Is the horizontal or vertical slice indicator, encoded in "V":

V	<HV>
0	H
1	V

<Ws> Is the 32-bit name of the slice index register W12-W15, encoded in the "Rs" field.

<imm> Is the slice index offset, in the range 0 to 15, encoded in the "imm4" field.

<Pg> Is the name of the governing scalable predicate register P0-P7, encoded in the "Pg" field.

<Xn|SP> Is the 64-bit name of the general-purpose base register or stack pointer, encoded in the "Rn" field.

<Xm> Is the optional 64-bit name of the general-purpose offset register, defaulting to XZR, encoded in the "Rm" field.

Operation

```
1 CheckStreamingSVEAndZAEEnabled();
2 integer dim = VL DIV esize;
3 bits(64) base;
4 bits(64) addr;
5 bits(64) offset;
6 bits(PL) mask = P[g];
7 bits(VL) result;
8 bits(32) index = X[s];
9 integer slice = (UInt(index) + imm) MOD dim;
10 constant integer mbytes = esize DIV 8;
11
12 if HaveMTEExt() then SetTagCheckedInstruction(TRUE);
13
14 if n == 31 then
15     if AnyActiveElement(mask, esize) ||
16         ConstrainUnpredictableBool(Unpredictable_CHECKSPNONEACTIVE) then
17         CheckSPAlignment();
18         base = SP[];
19     else
20         base = X[n];
```

Chapter D1. SME instructions

D1.1. SME data-processing instructions

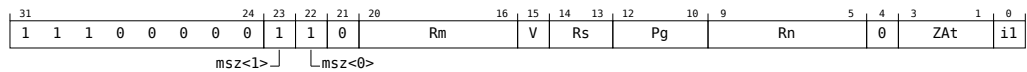
```
21 offset = X[m];
22
23 for e = 0 to dim - 1
24     addr = base + UInt(offset) * mbytes;
25     if ElemP[mask, e, esize] == '1' then
26         Elem[result, e, esize] = Mem[addr, mbytes, AccType_SME];
27     else
28         Elem[result, e, esize] = Zeros();
29     offset = offset + 1;
30
31 ZAslice[t, esize, vertical, slice] = result;
```


D1.1.12 LD1D

Contiguous load of doublewords to 64-bit element ZA tile slice

The slice number within the tile is selected by the sum of the slice index register and an immediate, modulo the number of 64-bit elements in a Streaming SVE vector. The immediate is in the range 0 to 1. The memory address is generated by scalar base and optional scalar offset which is multiplied by 8 and added to the base address. Inactive elements will not cause a read from Device memory or signal a fault, and are set to zero in the destination vector.

SME
(FEAT_SME)



```
LD1D { <ZAt><HV>.D[<Ws>, <imm>] }, <Pg>/Z, [<Xn|SP>{, <Xm>, LSL #3}]
```

```
1 if !HaveSME() then UNDEFINED;
2 integer n = UInt(Rn);
3 integer m = UInt(Rm);
4 integer g = UInt('0':Pg);
5 integer s = UInt('011':Rs);
6 integer t = UInt(ZAt);
7 integer imm = UInt(i1);
8 integer esize = 64;
9 boolean vertical = V == '1';
```

Assembler Symbols

<ZAt> Is the name of the ZA tile ZA0-ZA7 to be accessed, encoded in the "ZAt" field.

<HV> Is the horizontal or vertical slice indicator, encoded in "V":

V	<HV>
0	H
1	V

<Ws> Is the 32-bit name of the slice index register W12-W15, encoded in the "Rs" field.

<imm> Is the slice index offset, in the range 0 to 1, encoded in the "i1" field.

<Pg> Is the name of the governing scalable predicate register P0-P7, encoded in the "Pg" field.

<Xn|SP> Is the 64-bit name of the general-purpose base register or stack pointer, encoded in the "Rn" field.

<Xm> Is the optional 64-bit name of the general-purpose offset register, defaulting to XZR, encoded in the "Rm" field.

Operation

```
1 CheckStreamingSVEAndZEnabled();
2 integer dim = VL DIV esize;
3 bits(64) base;
4 bits(64) addr;
5 bits(64) offset;
6 bits(PL) mask = P[g];
7 bits(VL) result;
8 bits(32) index = X[s];
9 integer slice = (UInt(index) + imm) MOD dim;
10 constant integer mbytes = esize DIV 8;
11
12 if HaveMTEExt() then SetTagCheckedInstruction(TRUE);
13
14 if n == 31 then
15     if AnyActiveElement(mask, esize) ||
16         ConstrainUnpredictableBool(Unpredictable_CHECKSPNONEACTIVE) then
17         CheckSPAlignment();
18     base = SP[];
```

Chapter D1. SME instructions

D1.1. SME data-processing instructions

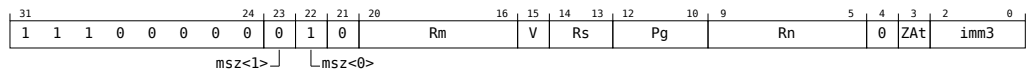
```
19  else
20      base = X[n];
21  offset = X[m];
22
23  for e = 0 to dim - 1
24      addr = base + UInt(offset) * mbytes;
25      if ElemP[mask, e, esize] == '1' then
26          Elem[result, e, esize] = Mem[addr, mbytes, AccType_SME];
27      else
28          Elem[result, e, esize] = Zeros();
29      offset = offset + 1;
30
31  ZAslice[t, esize, vertical, slice] = result;
```

D1.1.13 LD1H

Contiguous load of halfwords to 16-bit element ZA tile slice

The slice number within the tile is selected by the sum of the slice index register and an immediate, modulo the number of 16-bit elements in a Streaming SVE vector. The immediate is in the range 0 to 7. The memory address is generated by scalar base and optional scalar offset which is multiplied by 2 and added to the base address. Inactive elements will not cause a read from Device memory or signal a fault, and are set to zero in the destination vector.

SME
(FEAT_SME)



```
LD1H    { <ZAt><HV>.H[<Ws>, <imm>] }, <Pg>/Z, [<Xn|SP>{, <Xm>, LSL #1}]
```

```
1  if !HaveSME() then UNDEFINED;
2  integer n = UInt(Rn);
3  integer m = UInt(Rm);
4  integer g = UInt('0':Pg);
5  integer s = UInt('011':Rs);
6  integer t = UInt(ZAt);
7  integer imm = UInt(imm3);
8  integer esize = 16;
9  boolean vertical = V == '1';
```

Assembler Symbols

<ZAt> Is the name of the ZA tile ZA0-ZA1 to be accessed, encoded in the "ZAt" field.

<HV> Is the horizontal or vertical slice indicator, encoded in "V":

V	<HV>
0	H
1	V

<Ws> Is the 32-bit name of the slice index register W12-W15, encoded in the "Rs" field.

<imm> Is the slice index offset, in the range 0 to 7, encoded in the "imm3" field.

<Pg> Is the name of the governing scalable predicate register P0-P7, encoded in the "Pg" field.

<Xn|SP> Is the 64-bit name of the general-purpose base register or stack pointer, encoded in the "Rn" field.

<Xm> Is the optional 64-bit name of the general-purpose offset register, defaulting to XZR, encoded in the "Rm" field.

Operation

```
1  CheckStreamingSVEAndZEnabled();
2  integer dim = VL DIV esize;
3  bits(64) base;
4  bits(64) addr;
5  bits(64) offset;
6  bits(PL) mask = P[g];
7  bits(VL) result;
8  bits(32) index = X[s];
9  integer slice = (UInt(index) + imm) MOD dim;
10 constant integer mbytes = esize DIV 8;
11
12 if HaveMTEExt() then SetTagCheckedInstruction(TRUE);
13
14 if n == 31 then
15     if AnyActiveElement(mask, esize) ||
16         ConstrainUnpredictableBool(Unpredictable_CHECKSPNONEACTIVE) then
17         CheckSPAlignment();
18     base = SP[];
```

Chapter D1. SME instructions

D1.1. SME data-processing instructions

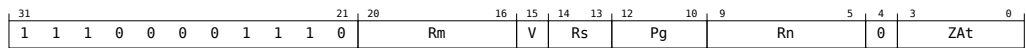
```
19  else
20      base = X[n];
21  offset = X[m];
22
23  for e = 0 to dim - 1
24      addr = base + UInt(offset) * mbytes;
25      if ElemP[mask, e, esize] == '1' then
26          Elem[result, e, esize] = Mem[addr, mbytes, AccType_SME];
27      else
28          Elem[result, e, esize] = Zeros();
29      offset = offset + 1;
30
31  ZAslice[t, esize, vertical, slice] = result;
```

D1.1.14 LD1Q

Contiguous load of quadwords to 128-bit element ZA tile slice

The slice number in the tile is selected by the slice index register, modulo the number of 128-bit elements in a Streaming SVE vector. The memory address is generated by scalar base and optional scalar offset which is multiplied by 16 and added to the base address. Inactive elements will not cause a read from Device memory or signal a fault, and are set to zero in the destination vector.

SME
(FEAT_SME)



```
LD1Q { <ZAt><HV>.Q[<Ws>, <imm>] }, <Pg>/Z, [<Xn|SP>{, <Xm>, LSL #4}]
```

```
1 if !HaveSME() then UNDEFINED;
2 integer n = UInt(Rn);
3 integer m = UInt(Rm);
4 integer g = UInt('0':Pg);
5 integer s = UInt('011':Rs);
6 integer t = UInt(ZAt);
7 integer imm = 0;
8 integer esize = 128;
9 boolean vertical = V == '1';
```

Assembler Symbols

- <ZAt> Is the name of the ZA tile ZA0-ZA15 to be accessed, encoded in the "ZAt" field.
- <HV> Is the horizontal or vertical slice indicator, encoded in "V":

V	<HV>
0	H
1	V
- <Ws> Is the 32-bit name of the slice index register W12-W15, encoded in the "Rs" field.
- <imm> Is the slice index offset 0.
- <Pg> Is the name of the governing scalable predicate register P0-P7, encoded in the "Pg" field.
- <Xn|SP> Is the 64-bit name of the general-purpose base register or stack pointer, encoded in the "Rn" field.
- <Xm> Is the optional 64-bit name of the general-purpose offset register, defaulting to XZR, encoded in the "Rm" field.

Operation

```
1 CheckStreamingSVEAndZEnabled();
2 integer dim = VL DIV esize;
3 bits(64) base;
4 bits(64) addr;
5 bits(64) offset;
6 bits(PL) mask = P[g];
7 bits(VL) result;
8 bits(32) index = X[s];
9 integer slice = (UInt(index) + imm) MOD dim;
10 constant integer mbytes = esize DIV 8;
11
12 if HaveMTEExt() then SetTagCheckedInstruction(TRUE);
13
14 if n == 31 then
15     if AnyActiveElement(mask, esize) ||
16         ConstrainUnpredictableBool(Unpredictable_CHECKSPNONEACTIVE) then
17         CheckSPAlignment();
18     base = SP[];
```

Chapter D1. SME instructions

D1.1. SME data-processing instructions

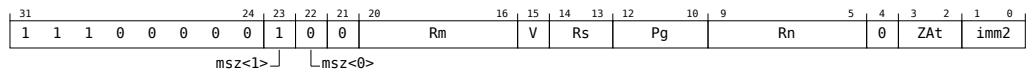
```
19  else
20      base = X[n];
21  offset = X[m];
22
23  for e = 0 to dim - 1
24      addr = base + UInt(offset) * mbytes;
25      if ElemP[mask, e, esize] == '1' then
26          Elem[result, e, esize] = Mem[addr, mbytes, AccType_SME];
27      else
28          Elem[result, e, esize] = Zeros();
29      offset = offset + 1;
30
31  ZAslice[t, esize, vertical, slice] = result;
```

D1.1.15 LD1W

Contiguous load of words to 32-bit element ZA tile slice

The slice number within the tile is selected by the sum of the slice index register and an immediate, modulo the number of 32-bit elements in a Streaming SVE vector. The immediate is in the range 0 to 3. The memory address is generated by scalar base and optional scalar offset which is multiplied by 4 and added to the base address. Inactive elements will not cause a read from Device memory or signal a fault, and are set to zero in the destination vector.

SME
(FEAT_SME)



```
LD1W { <ZAt><HV>.S[<Ws>, <imm>] }, <Pg>/Z, [<Xn|SP>{, <Xm>, LSL #2}]
```

```
1 if !HaveSME() then UNDEFINED;
2 integer n = UInt(Rn);
3 integer m = UInt(Rm);
4 integer g = UInt('0':Pg);
5 integer s = UInt('011':Rs);
6 integer t = UInt(ZAt);
7 integer imm = UInt(imm2);
8 integer esize = 32;
9 boolean vertical = V == '1';
```

Assembler Symbols

<ZAt> Is the name of the ZA tile ZA0-ZA3 to be accessed, encoded in the "ZAt" field.

<HV> Is the horizontal or vertical slice indicator, encoded in "V":

V	<HV>
0	H
1	V

<Ws> Is the 32-bit name of the slice index register W12-W15, encoded in the "Rs" field.

<imm> Is the slice index offset, in the range 0 to 3, encoded in the "imm2" field.

<Pg> Is the name of the governing scalable predicate register P0-P7, encoded in the "Pg" field.

<Xn|SP> Is the 64-bit name of the general-purpose base register or stack pointer, encoded in the "Rn" field.

<Xm> Is the optional 64-bit name of the general-purpose offset register, defaulting to XZR, encoded in the "Rm" field.

Operation

```
1 CheckStreamingSVEAndZEnabled();
2 integer dim = VL DIV esize;
3 bits(64) base;
4 bits(64) addr;
5 bits(64) offset;
6 bits(PL) mask = P[g];
7 bits(VL) result;
8 bits(32) index = X[s];
9 integer slice = (UInt(index) + imm) MOD dim;
10 constant integer mbytes = esize DIV 8;
11
12 if HaveMTEExt() then SetTagCheckedInstruction(TRUE);
13
14 if n == 31 then
15     if AnyActiveElement(mask, esize) ||
16         ConstrainUnpredictableBool(Unpredictable_CHECKSPNONEACTIVE) then
17         CheckSPAlignment();
18     base = SP[];
```

Chapter D1. SME instructions

D1.1. SME data-processing instructions

```
19  else
20      base = X[n];
21  offset = X[m];
22
23  for e = 0 to dim - 1
24      addr = base + UInt(offset) * mbytes;
25      if ElemP[mask, e, esize] == '1' then
26          Elem[result, e, esize] = Mem[addr, mbytes, AccType_SME];
27      else
28          Elem[result, e, esize] = Zeros();
29      offset = offset + 1;
30
31  ZAslice[t, esize, vertical, slice] = result;
```


D1.1.16 LDR

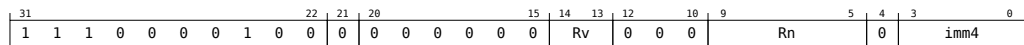
Load vector to ZA array

The ZA array vector is selected by the sum of the vector select register and an immediate, modulo the number of bytes in a Streaming SVE vector. The immediate is in the range 0 to 15. The memory address is generated by scalar base, plus the same optional immediate offset multiplied by the current vector length in bytes. This instruction is unpredicated.

The load is performed as contiguous byte accesses, with no endian conversion and no guarantee of single-copy atomicity larger than a byte. However, if alignment is checked, then the base register must be aligned to 16 bytes.

This instruction does not require the PE to be in Streaming SVE mode, and it is expected that this instruction will not experience a significant slowdown due to contention with other PEs that are executing in Streaming SVE mode.

SME
(FEAT_SME)



LDR ZA[<Wv>, <imm>], [<Xn|SP>{, #<imm>, MUL VL}]

```
1 if !HaveSME() then UNDEFINED;
2 integer n = UInt(Rn);
3 integer v = UInt('011':Rv);
4 integer imm = UInt(imm4);
```

Assembler Symbols

- <Wv> Is the 32-bit name of the vector select register W12-W15, encoded in the "Rv" field.
- <imm> Is the vector select offset and optional memory offset, in the range 0 to 15, defaulting to 0, encoded in the "imm4" field.
- <Xn|SP> Is the 64-bit name of the general-purpose base register or stack pointer, encoded in the "Rn" field.

Operation

```
1 CheckSMEAndZEnabled();
2 integer dim = SVL DIV 8;
3 bits(64) base;
4 integer offset = imm * dim;
5 bits(SVL) result;
6 bits(32) idx = X[v];
7 integer vec = (UInt(idx) + imm) MOD dim;
8
9 if HaveTME() && TSTATE.depth > 0 then
10     FailTransaction(TMFailure_ERR, FALSE);
11
12 if n == 31 then
13     if HaveMTEExt() then SetTagCheckedInstruction(FALSE);
14     CheckSPAlignment();
15     base = SP[];
16 else
17     if HaveMTEExt() then SetTagCheckedInstruction(TRUE);
18     base = X[n];
19
20 boolean aligned = AArch64.CheckAlignment(base + offset, 16, AccType_SME, FALSE);
21 for e = 0 to dim-1
22     Elem[result, e, 8] = AArch64.MemSingle[base + offset, 1, AccType_SME, aligned];
23     offset = offset + 1;
24
25 ZAVector[vec] = result;
```

D1.1.17 MOV (tile to vector)

Move ZA tile slice to vector register

The instruction operates on individual horizontal or vertical slices within a named ZA tile of the specified element size. The slice number within the tile is selected by the sum of the slice index register and an immediate, modulo the number of such elements in a Streaming SVE vector. The immediate is in the range 0 to the number of elements in a 128-bit vector segment minus 1.

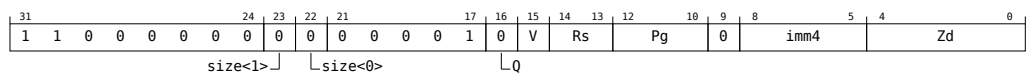
Inactive elements in the destination vector remain unmodified.

This is an alias of [MOVA \(tile to vector\)](#). This means:

- The encodings in this description are named to match the encodings of [MOVA \(tile to vector\)](#).
- The description of [MOVA \(tile to vector\)](#) gives the operational pseudocode for this instruction.

It has encodings from 5 classes: [8-bit](#) , [16-bit](#) , [32-bit](#) , [64-bit](#) and [128-bit](#)

8-bit



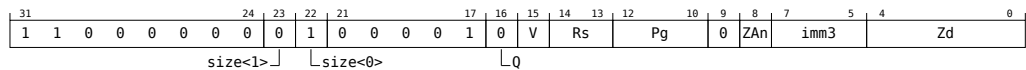
MOV [<Zd>](#).B, [<Pg>](#)/M, ZA0[<HV>](#).B[[<Ws>](#), [<imm>](#)]

is equivalent to

[MOVA<Zd>](#).B, [<Pg>](#)/M, ZA0[<HV>](#).B[[<Ws>](#), [<imm>](#)]

and is always the preferred disassembly.

16-bit



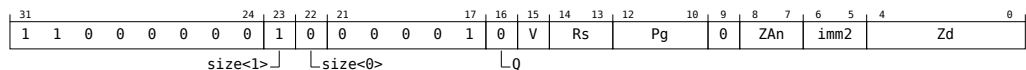
MOV [<Zd>](#).H, [<Pg>](#)/M, [<ZAn>](#)[<HV>](#).H[[<Ws>](#), [<imm>](#)]

is equivalent to

[MOVA<Zd>](#).H, [<Pg>](#)/M, [<ZAn>](#)[<HV>](#).H[[<Ws>](#), [<imm>](#)]

and is always the preferred disassembly.

32-bit



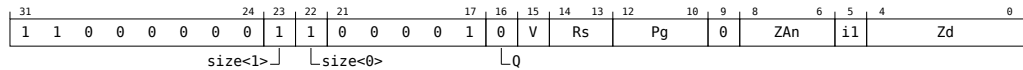
MOV [<Zd>](#).S, [<Pg>](#)/M, [<ZAn>](#)[<HV>](#).S[[<Ws>](#), [<imm>](#)]

is equivalent to

[MOVA<Zd>](#).S, [<Pg>](#)/M, [<ZAn>](#)[<HV>](#).S[[<Ws>](#), [<imm>](#)]

and is always the preferred disassembly.

64-bit



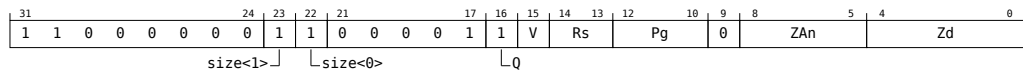
MOV <Zd>.D, <Pg>/M, <ZAn><HV>.D[<Ws>, <imm>]

is equivalent to

MOVA<Zd>.D, <Pg>/M, <ZAn><HV>.D[<Ws>, <imm>]

and is always the preferred disassembly.

128-bit



MOV <Zd>.Q, <Pg>/M, <ZAn><HV>.Q[<Ws>, <imm>]

is equivalent to

MOVA<Zd>.Q, <Pg>/M, <ZAn><HV>.Q[<Ws>, <imm>]

and is always the preferred disassembly.

Assembler Symbols

- <Zd> Is the name of the destination scalable vector register, encoded in the "Zd" field.
- <Pg> Is the name of the governing scalable predicate register P0-P7, encoded in the "Pg" field.
- <ZAn> For the 16-bit variant: is the name of the ZA tile ZA0-ZA1 to be accessed, encoded in the "ZAn" field.

For the 32-bit variant: is the name of the ZA tile ZA0-ZA3 to be accessed, encoded in the "ZAn" field.

For the 64-bit variant: is the name of the ZA tile ZA0-ZA7 to be accessed, encoded in the "ZAn" field.

For the 128-bit variant: is the name of the ZA tile ZA0-ZA15 to be accessed, encoded in the "ZAn" field.
- <HV> Is the horizontal or vertical slice indicator, encoded in "V":

V	<HV>
0	H
1	V
- <Ws> Is the 32-bit name of the slice index register W12-W15, encoded in the "Rs" field.
- <imm> For the 8-bit variant: is the slice index offset, in the range 0 to 15, encoded in the "imm4" field.

For the 16-bit variant: is the slice index offset, in the range 0 to 7, encoded in the "imm3" field.

For the 32-bit variant: is the slice index offset, in the range 0 to 3, encoded in the "imm2" field.

For the 64-bit variant: is the slice index offset, in the range 0 to 1, encoded in the "i1" field.

For the 128-bit variant: is the slice index offset 0.

Operation

The description of [MOVA \(tile to vector\)](#) gives the operational pseudocode for this instruction.

Operational information

If FEAT_SVE2 is implemented or FEAT_SME is implemented, then when PSTATE.DIT is 1:

- The execution time of this instruction is independent of:
 - The values of the data supplied in any of its operand registers when its governing predicate register contains the same value for each execution.
 - The values of the NZCV flags.
- The response of this instruction to asynchronous exceptions does not vary based on:
 - The values of the data supplied in any of its operand registers when its governing predicate register contains the same value for each execution.
 - The values of the NZCV flags.

D1.1.18 MOV (vector to tile)

Move vector register to ZA tile slice

The instruction operates on individual horizontal or vertical slices within a named ZA tile of the specified element size. The slice number within the tile is selected by the sum of the slice index register and an immediate, modulo the number of such elements in a Streaming SVE vector. The immediate is in the range 0 to the number of elements in a 128-bit vector segment minus 1.

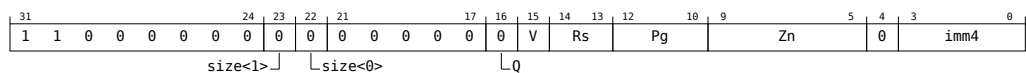
Inactive elements in the destination slice remain unmodified.

This is an alias of [MOVA \(vector to tile\)](#). This means:

- The encodings in this description are named to match the encodings of [MOVA \(vector to tile\)](#).
- The description of [MOVA \(vector to tile\)](#) gives the operational pseudocode for this instruction.

It has encodings from 5 classes: [8-bit](#) , [16-bit](#) , [32-bit](#) , [64-bit](#) and [128-bit](#)

8-bit



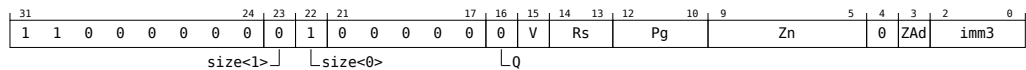
MOV ZA0<HV>.B[<Ws>, <imm>], <Pg>/M, <Zn>.B

is equivalent to

MOVAZA0<HV>.B[<Ws>, <imm>], <Pg>/M, <Zn>.B

and is always the preferred disassembly.

16-bit



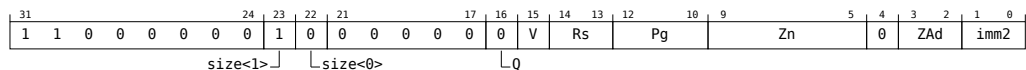
MOV <ZAd><HV>.H[<Ws>, <imm>], <Pg>/M, <Zn>.H

is equivalent to

MOVA<ZAd><HV>.H[<Ws>, <imm>], <Pg>/M, <Zn>.H

and is always the preferred disassembly.

32-bit



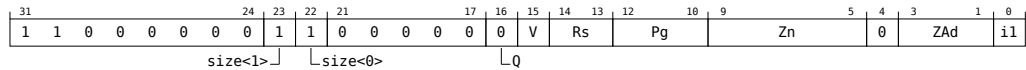
MOV <ZAd><HV>.S[<Ws>, <imm>], <Pg>/M, <Zn>.S

is equivalent to

MOVA<ZAd><HV>.S[<Ws>, <imm>], <Pg>/M, <Zn>.S

and is always the preferred disassembly.

64-bit



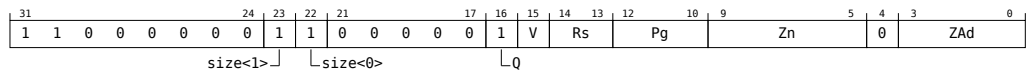
MOV <ZAd><HV>.D[<Ws>, <imm>], <Pg>/M, <Zn>.D

is equivalent to

MOVA<ZAd><HV>.D[<Ws>, <imm>], <Pg>/M, <Zn>.D

and is always the preferred disassembly.

128-bit



MOV <ZAd><HV>.Q[<Ws>, <imm>], <Pg>/M, <Zn>.Q

is equivalent to

MOVA<ZAd><HV>.Q[<Ws>, <imm>], <Pg>/M, <Zn>.Q

and is always the preferred disassembly.

Assembler Symbols

<ZAd> For the 16-bit variant: is the name of the ZA tile ZA0-ZA1 to be accessed, encoded in the "ZAd" field.

For the 32-bit variant: is the name of the ZA tile ZA0-ZA3 to be accessed, encoded in the "ZAd" field.

For the 64-bit variant: is the name of the ZA tile ZA0-ZA7 to be accessed, encoded in the "ZAd" field.

For the 128-bit variant: is the name of the ZA tile ZA0-ZA15 to be accessed, encoded in the "ZAd" field.

<HV> Is the horizontal or vertical slice indicator, encoded in "V":

V	<HV>
0	H
1	V

<Ws> Is the 32-bit name of the slice index register W12-W15, encoded in the "Rs" field.

<imm> For the 8-bit variant: is the slice index offset, in the range 0 to 15, encoded in the "imm4" field.

For the 16-bit variant: is the slice index offset, in the range 0 to 7, encoded in the "imm3" field.

For the 32-bit variant: is the slice index offset, in the range 0 to 3, encoded in the "imm2" field.

For the 64-bit variant: is the slice index offset, in the range 0 to 1, encoded in the "i1" field.

For the 128-bit variant: is the slice index offset 0.

<Pg> Is the name of the governing scalable predicate register P0-P7, encoded in the "Pg" field.

<Zn> Is the name of the source scalable vector register, encoded in the "Zn" field.

Operation

The description of [MOVA \(vector to tile\)](#) gives the operational pseudocode for this instruction.

Operational information

If FEAT_SVE2 is implemented or FEAT_SME is implemented, then when PSTATE.DIT is 1:

- The execution time of this instruction is independent of:
 - The values of the data supplied in any of its operand registers when its governing predicate register contains the same value for each execution.
 - The values of the NZCV flags.
- The response of this instruction to asynchronous exceptions does not vary based on:
 - The values of the data supplied in any of its operand registers when its governing predicate register contains the same value for each execution.
 - The values of the NZCV flags.

D1.1.19 MOVA (tile to vector)

Move ZA tile slice to vector register

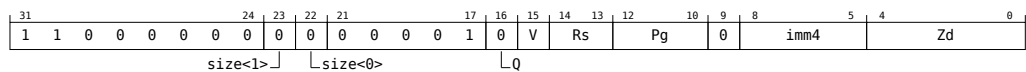
The instruction operates on individual horizontal or vertical slices within a named ZA tile of the specified element size. The slice number within the tile is selected by the sum of the slice index register and an immediate, modulo the number of such elements in a Streaming SVE vector. The immediate is in the range 0 to the number of elements in a 128-bit vector segment minus 1.

Inactive elements in the destination vector remain unmodified.

This instruction is used by the alias **MOV (tile to vector)**.

It has encodings from 5 classes: **8-bit**, **16-bit**, **32-bit**, **64-bit** and **128-bit**

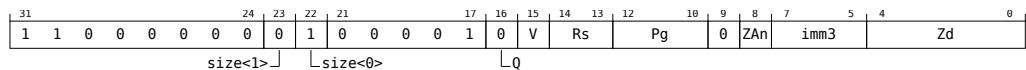
8-bit (FEAT_SME)



MOVA <Zd>.B, <Pg>/M, ZA0<HV>.B[<Ws>, <imm>]

```
1 if !HaveSME() then UNDEFINED;
2 integer g = UInt(Pg);
3 integer s = UInt('011':Rs);
4 integer n = 0;
5 integer imm = UInt(imm4);
6 integer esize = 8;
7 integer d = UInt(Zd);
8 boolean vertical = V == '1';
```

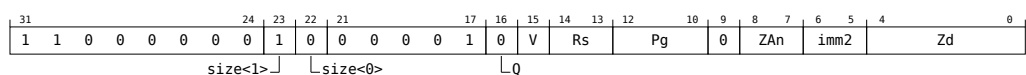
16-bit (FEAT_SME)



MOVA <Zd>.H, <Pg>/M, <ZAn><HV>.H[<Ws>, <imm>]

```
1 if !HaveSME() then UNDEFINED;
2 integer g = UInt(Pg);
3 integer s = UInt('011':Rs);
4 integer n = UInt(ZAn);
5 integer imm = UInt(imm3);
6 integer esize = 16;
7 integer d = UInt(Zd);
8 boolean vertical = V == '1';
```

32-bit (FEAT_SME)



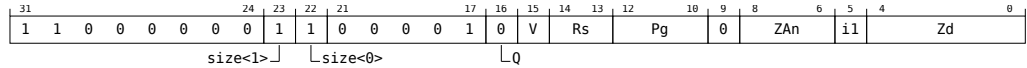
MOVA <Zd>.S, <Pg>/M, <ZAn><HV>.S[<Ws>, <imm>]


```

1 if !HaveSME() then UNDEFINED;
2 integer g = UInt(Pg);
3 integer s = UInt('011':Rs);
4 integer n = UInt(ZAn);
5 integer imm = UInt(imm2);
6 integer esize = 32;
7 integer d = UInt(Zd);
8 boolean vertical = V == '1';

```

64-bit (FEAT_SME)



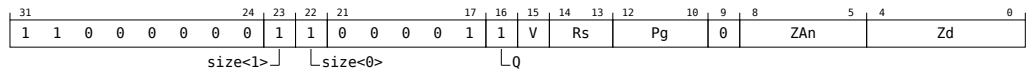
MOVA <Zd>.D, <Pg>/M, <ZAn><HV>.D[<Ws>, <imm>]

```

1 if !HaveSME() then UNDEFINED;
2 integer g = UInt(Pg);
3 integer s = UInt('011':Rs);
4 integer n = UInt(ZAn);
5 integer imm = UInt(il);
6 integer esize = 64;
7 integer d = UInt(Zd);
8 boolean vertical = V == '1';

```

128-bit (FEAT_SME)



MOVA <Zd>.Q, <Pg>/M, <ZAn><HV>.Q[<Ws>, <imm>]

```

1 if !HaveSME() then UNDEFINED;
2 integer g = UInt(Pg);
3 integer s = UInt('011':Rs);
4 integer n = UInt(ZAn);
5 integer imm = 0;
6 integer esize = 128;
7 integer d = UInt(Zd);
8 boolean vertical = V == '1';

```

Assembler Symbols

- <Zd> Is the name of the destination scalable vector register, encoded in the "Zd" field.
- <Pg> Is the name of the governing scalable predicate register P0-P7, encoded in the "Pg" field.
- <ZAn> For the 16-bit variant: is the name of the ZA tile ZA0-ZA1 to be accessed, encoded in the "ZAn" field.
For the 32-bit variant: is the name of the ZA tile ZA0-ZA3 to be accessed, encoded in the "ZAn" field.
For the 64-bit variant: is the name of the ZA tile ZA0-ZA7 to be accessed, encoded in the "ZAn" field.
For the 128-bit variant: is the name of the ZA tile ZA0-ZA15 to be accessed, encoded in the "ZAn" field.
- <HV> Is the horizontal or vertical slice indicator, encoded in "V":

V	<HV>
0	H
1	V

- <Ws> Is the 32-bit name of the slice index register W12-W15, encoded in the "Rs" field.
- <imm> For the 8-bit variant: is the slice index offset, in the range 0 to 15, encoded in the "imm4" field.
- For the 16-bit variant: is the slice index offset, in the range 0 to 7, encoded in the "imm3" field.
- For the 32-bit variant: is the slice index offset, in the range 0 to 3, encoded in the "imm2" field.
- For the 64-bit variant: is the slice index offset, in the range 0 to 1, encoded in the "i1" field.
- For the 128-bit variant: is the slice index offset 0.

Operation

```

1 CheckStreamingSVEAndZAAEnabled();
2 integer dim = VL DIV esize;
3 bits(PL) mask = P[g];
4 bits(32) index = X[s];
5 integer slice = (UInt(index) + imm) MOD dim;
6 bits(VL) operand = ZAslice[n, esize, vertical, slice];
7 bits(VL) result = Z[d];
8
9 for e = 0 to dim-1
10     bits(esize) element = Elem[operand, e, esize];
11     if ElemP[mask, e, esize] == '1' then
12         Elem[result, e, esize] = element;
13
14 Z[d] = result;
```

Operational information

If FEAT_SVE2 is implemented or FEAT_SME is implemented, then when PSTATE.DIT is 1:

- The execution time of this instruction is independent of:
 - The values of the data supplied in any of its operand registers when its governing predicate register contains the same value for each execution.
 - The values of the NZCV flags.
- The response of this instruction to asynchronous exceptions does not vary based on:
 - The values of the data supplied in any of its operand registers when its governing predicate register contains the same value for each execution.
 - The values of the NZCV flags.

D1.1.20 MOVA (vector to tile)

Move vector register to ZA tile slice

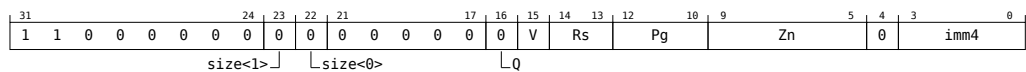
The instruction operates on individual horizontal or vertical slices within a named ZA tile of the specified element size. The slice number within the tile is selected by the sum of the slice index register and an immediate, modulo the number of such elements in a Streaming SVE vector. The immediate is in the range 0 to the number of elements in a 128-bit vector segment minus 1.

Inactive elements in the destination slice remain unmodified.

This instruction is used by the alias [MOV \(vector to tile\)](#).

It has encodings from 5 classes: [8-bit](#) , [16-bit](#) , [32-bit](#) , [64-bit](#) and [128-bit](#)

8-bit (FEAT_SME)



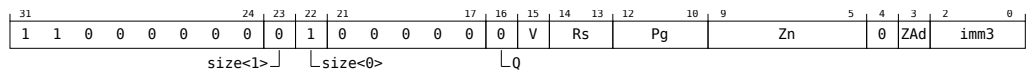
MOVA ZA0<HV>.B[<Ws>, <imm>], <Pg>/M, <Zn>.B

```

1 if !HaveSME() then UNDEFINED;
2 integer g = UInt(Pg);
3 integer s = UInt('011':Rs);
4 integer n = UInt(Zn);
5 integer d = 0;
6 integer imm = UInt(imm4);
7 integer esize = 8;
8 boolean vertical = V == '1';

```

16-bit (FEAT_SME)



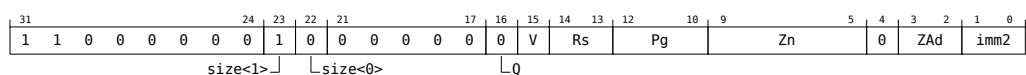
MOVA <ZAd><HV>.H[<Ws>, <imm>], <Pg>/M, <Zn>.H

```

1 if !HaveSME() then UNDEFINED;
2 integer g = UInt(Pg);
3 integer s = UInt('011':Rs);
4 integer n = UInt(Zn);
5 integer d = UInt(ZAd);
6 integer imm = UInt(imm3);
7 integer esize = 16;
8 boolean vertical = V == '1';

```

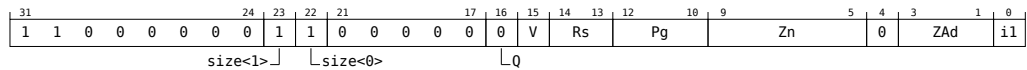
32-bit (FEAT_SME)



MOVA <ZAd><HV>.S[<Ws>, <imm>], <Pg>/M, <Zn>.S

```
1 if !HaveSME() then UNDEFINED;
2 integer g = UInt(Pg);
3 integer s = UInt('011':Rs);
4 integer n = UInt(Zn);
5 integer d = UInt(ZAd);
6 integer imm = UInt(imm2);
7 integer esize = 32;
8 boolean vertical = V == '1';
```

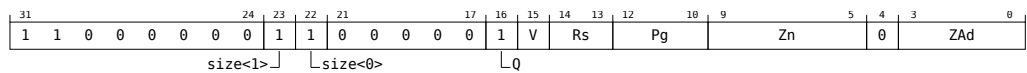
64-bit (FEAT_SME)



MOVA <ZAd><HV>.D[<Ws>, <imm>], <Pg>/M, <Zn>.D

```
1 if !HaveSME() then UNDEFINED;
2 integer g = UInt(Pg);
3 integer s = UInt('011':Rs);
4 integer n = UInt(Zn);
5 integer d = UInt(ZAd);
6 integer imm = UInt(i1);
7 integer esize = 64;
8 boolean vertical = V == '1';
```

128-bit (FEAT_SME)



MOVA <ZAd><HV>.Q[<Ws>, <imm>], <Pg>/M, <Zn>.Q

```
1 if !HaveSME() then UNDEFINED;
2 integer g = UInt(Pg);
3 integer s = UInt('011':Rs);
4 integer n = UInt(Zn);
5 integer d = UInt(ZAd);
6 integer imm = 0;
7 integer esize = 128;
8 boolean vertical = V == '1';
```

Assembler Symbols

<ZAd> For the 16-bit variant: is the name of the ZA tile ZA0-ZA1 to be accessed, encoded in the "ZAd" field.

For the 32-bit variant: is the name of the ZA tile ZA0-ZA3 to be accessed, encoded in the "ZAd" field.

For the 64-bit variant: is the name of the ZA tile ZA0-ZA7 to be accessed, encoded in the "ZAd" field.

For the 128-bit variant: is the name of the ZA tile ZA0-ZA15 to be accessed, encoded in the "ZAd" field.

<HV> Is the horizontal or vertical slice indicator, encoded in "V":

V	<HV>
0	H
1	V

<Ws> Is the 32-bit name of the slice index register W12-W15, encoded in the "Rs" field.

- <imm> For the 8-bit variant: is the slice index offset, in the range 0 to 15, encoded in the "imm4" field.
For the 16-bit variant: is the slice index offset, in the range 0 to 7, encoded in the "imm3" field.
For the 32-bit variant: is the slice index offset, in the range 0 to 3, encoded in the "imm2" field.
For the 64-bit variant: is the slice index offset, in the range 0 to 1, encoded in the "i1" field.
For the 128-bit variant: is the slice index offset 0.
- <Pg> Is the name of the governing scalable predicate register P0-P7, encoded in the "Pg" field.
- <Zn> Is the name of the source scalable vector register, encoded in the "Zn" field.

Operation

```

1 CheckStreamingSVEAndZAAEnabled();
2 integer dim = VL DIV esize;
3 bits(PL) mask = P[g];
4 bits(VL) operand = Z[n];
5 bits(32) index = X[s];
6 integer slice = (UInt(index) + imm) MOD dim;
7 bits(VL) result = ZAslice[d, esize, vertical, slice];
8
9 for e = 0 to dim-1
10     bits(esize) element = Elem[operand, e, esize];
11     if ElemP[mask, e, esize] == '1' then
12         Elem[result, e, esize] = element;
13
14 ZAslice[d, esize, vertical, slice] = result;
```

Operational information

If FEAT_SVE2 is implemented or FEAT_SME is implemented, then when PSTATE.DIT is 1:

- The execution time of this instruction is independent of:
 - The values of the data supplied in any of its operand registers when its governing predicate register contains the same value for each execution.
 - The values of the NZCV flags.
- The response of this instruction to asynchronous exceptions does not vary based on:
 - The values of the data supplied in any of its operand registers when its governing predicate register contains the same value for each execution.
 - The values of the NZCV flags.

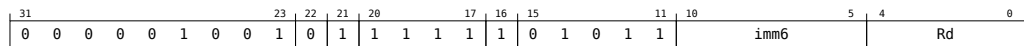
D1.1.21 RDSVL

Read multiple of Streaming SVE vector register size to scalar register

Multiply the Streaming SVE vector register size in bytes by an immediate in the range -32 to 31 and place the result in the 64-bit destination general-purpose register.

This instruction does not require the PE to be in Streaming SVE mode.

SME
(FEAT_SME)



RDSVL <Xd>, #<imm>

```
1 if !HaveSME() then UNDEFINED;
2 integer d = UInt(Rd);
3 integer imm = SInt(imm6);
```

Assembler Symbols

<Xd> Is the 64-bit name of the destination general-purpose register, encoded in the "Rd" field.

<imm> Is the signed immediate operand, in the range -32 to 31, encoded in the "imm6" field.

Operation

```
1 CheckSMEEnabled();
2 integer len = imm * (SVL DIV 8);
3 X[d] = len<63:0>;
```

Operational information

If FEAT_SVE2 is implemented or FEAT_SME is implemented, then when PSTATE.DIT is 1:

- The execution time of this instruction is independent of:
 - The values of the data supplied in any of its registers.
 - The values of the NZCV flags.
- The response of this instruction to asynchronous exceptions does not vary based on:
 - The values of the data supplied in any of its registers.
 - The values of the NZCV flags.

D1.1.22 SMOPA

Signed integer sum of outer products and accumulate

The 8-bit integer variant works with a 32-bit element ZA tile.

The 16-bit integer variant works with a 64-bit element ZA tile.

The signed integer sum of outer products and accumulate instructions multiply the sub-matrix in the first source vector by the sub-matrix in the second source vector. In case of the 8-bit integer variant, the first source holds $SVL_S \times 4$ sub-matrix of signed 8-bit integer values, and the second source holds $4 \times SVL_S$ sub-matrix of signed 8-bit integer values. In case of the 16-bit integer variant, the first source holds $SVL_D \times 4$ sub-matrix of signed 16-bit integer values, and the second source holds $4 \times SVL_D$ sub-matrix of signed 16-bit integer values.

Each source vector is independently predicated by a corresponding governing predicate. When an 8-bit source element in case of 8-bit integer variant or a 16-bit source element in case of 16-bit integer variant is Inactive, it is treated as having the value 0.

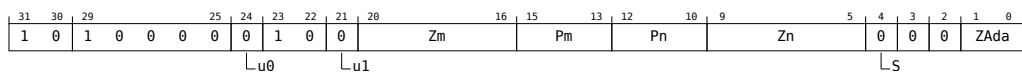
The resulting $SVL_S \times SVL_S$ widened 32-bit integer or $SVL_D \times SVL_D$ widened 64-bit integer sum of outer products is then destructively added to the 32-bit integer or 64-bit integer destination tile, respectively for 8-bit integer and 16-bit integer instruction variants. This is equivalent to performing a 4-way dot product and accumulate to each of the destination tile elements.

In case of the 8-bit integer variant, each 32-bit container of first source vector holds 4 consecutive column elements of each row of a $SVL_S \times 4$ sub-matrix, and each 32-bit container of second source vector holds 4 consecutive row elements of each column of a $4 \times SVL_S$ sub-matrix. In case of the 16-bit integer variant, each 64-bit container of first source vector holds 4 consecutive column elements of each row of a $SVL_D \times 4$ sub-matrix, and each 64-bit container of second source vector holds 4 consecutive row elements of each column of a $4 \times SVL_D$ sub-matrix.

ID_AA64SMFR0_EL1.I16I64 indicates whether the 16-bit integer variant is implemented.

It has encodings from 2 classes: [32-bit](#) and [64-bit](#)

32-bit (FEAT_SME)



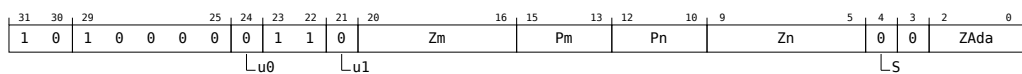
SMOPA <ZAda>.S, <Pn>/M, <Pm>/M, <Zn>.B, <Zm>.B

```

1 if !HaveSME() then UNDEFINED;
2 integer esize = 32;
3 integer a = UInt(Pn);
4 integer b = UInt(Pm);
5 integer n = UInt(Zn);
6 integer m = UInt(Zm);
7 integer da = UInt(ZAda);
8 boolean sub_op = FALSE;
9 boolean op1_unsigned = FALSE;
10 boolean op2_unsigned = FALSE;

```

64-bit (FEAT_SME_I16I64)



SMOPA <ZAda>.D, <Pn>/M, <Pm>/M, <Zn>.H, <Zm>.H

```

1 if !HaveSMEI16I64() then UNDEFINED;
2 integer esize = 64;
3 integer a = UInt(Pn);
4 integer b = UInt(Pm);
5 integer n = UInt(Zn);
6 integer m = UInt(Zm);
7 integer da = UInt(ZAda);
8 boolean sub_op = FALSE;
9 boolean op1_unsigned = FALSE;
10 boolean op2_unsigned = FALSE;

```

Assembler Symbols

- <ZAda> For the 32-bit variant: is the name of the ZA tile ZA0-ZA3, encoded in the "ZAda" field.
For the 64-bit variant: is the name of the ZA tile ZA0-ZA7, encoded in the "ZAda" field.
- <Pn> Is the name of the first governing scalable predicate register P0-P7, encoded in the "Pn" field.
- <Pm> Is the name of the second governing scalable predicate register P0-P7, encoded in the "Pm" field.
- <Zn> Is the name of the first source scalable vector register, encoded in the "Zn" field.
- <Zm> Is the name of the second source scalable vector register, encoded in the "Zm" field.

Operation

```

1 CheckStreamingSVEAndZAAEnabled();
2 integer dim = VL DIV esize;
3 bits(PL) mask1 = P[a];
4 bits(PL) mask2 = P[b];
5 bits(VL) operand1 = Z[n];
6 bits(VL) operand2 = Z[m];
7 bits(dim*dim*esize) operand3 = ZAtile[da, esize];
8 bits(dim*dim*esize) result;
9 integer prod;
10
11 for row = 0 to dim-1
12     for col = 0 to dim-1
13         bits(esize) sum = Elem[operand3, row*dim+col, esize];
14         for k = 0 to 3
15             if ElemP[mask1, 4*row + k, esize DIV 4] == '1' &&
16                 ElemP[mask2, 4*col + k, esize DIV 4] == '1' then
17                 prod = (Int(Elem[operand1, 4*row + k, esize DIV 4], op1_unsigned) *
18                     Int(Elem[operand2, 4*col + k, esize DIV 4], op2_unsigned));
19                 if sub_op then prod = -prod;
20                 sum = sum + prod;
21
22         Elem[result, row*dim+col, esize] = sum;
23
24 ZAtile[da, esize] = result;

```

Operational information

If FEAT_SVE2 is implemented or FEAT_SME is implemented, then when PSTATE.DIT is 1:

- The execution time of this instruction is independent of:
 - The values of the data supplied in any of its operand registers when its governing predicate registers contain the same value for each execution.
 - The values of the NZCV flags.
- The response of this instruction to asynchronous exceptions does not vary based on:
 - The values of the data supplied in any of its operand registers when its governing predicate registers contain the same value for each execution.
 - The values of the NZCV flags.

D1.1.23 SMOPS

Signed integer sum of outer products and subtract

The 8-bit integer variant works with a 32-bit element ZA tile.

The 16-bit integer variant works with a 64-bit element ZA tile.

The signed integer sum of outer products and subtract instructions multiply the sub-matrix in the first source vector by the sub-matrix in the second source vector. In case of the 8-bit integer variant, the first source holds $SVL_S \times 4$ sub-matrix of signed 8-bit integer values, and the second source holds $4 \times SVL_S$ sub-matrix of signed 8-bit integer values. In case of the 16-bit integer variant, the first source holds $SVL_D \times 4$ sub-matrix of signed 16-bit integer values, and the second source holds $4 \times SVL_D$ sub-matrix of signed 16-bit integer values.

Each source vector is independently predicated by a corresponding governing predicate. When an 8-bit source element in case of 8-bit integer variant or a 16-bit source element in case of 16-bit integer variant is Inactive, it is treated as having the value 0.

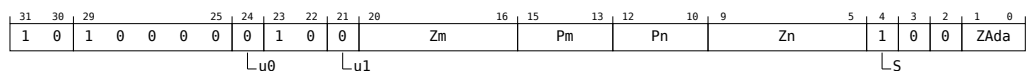
The resulting $SVL_S \times SVL_S$ widened 32-bit integer or $SVL_D \times SVL_D$ widened 64-bit integer sum of outer products is then destructively subtracted from the 32-bit integer or 64-bit integer destination tile, respectively for 8-bit integer and 16-bit integer instruction variants. This is equivalent to performing a 4-way dot product and subtract from each of the destination tile elements.

In case of the 8-bit integer variant, each 32-bit container of first source vector holds 4 consecutive column elements of each row of a $SVL_S \times 4$ sub-matrix, and each 32-bit container of second source vector holds 4 consecutive row elements of each column of a $4 \times SVL_S$ sub-matrix. In case of the 16-bit integer variant, each 64-bit container of first source vector holds 4 consecutive column elements of each row of a $SVL_D \times 4$ sub-matrix, and each 64-bit container of second source vector holds 4 consecutive row elements of each column of a $4 \times SVL_D$ sub-matrix.

ID_AA64SMFR0_EL1.I16I64 indicates whether the 16-bit integer variant is implemented.

It has encodings from 2 classes: [32-bit](#) and [64-bit](#)

32-bit (FEAT_SME)



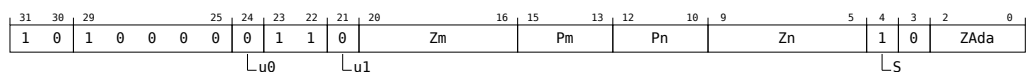
SMOPS [<ZAda>.S](#), [<Pn>/M](#), [<Pm>/M](#), [<Zn>.B](#), [<Zm>.B](#)

```

1 if !HaveSME() then UNDEFINED;
2 integer esize = 32;
3 integer a = UInt(Pn);
4 integer b = UInt(Pm);
5 integer n = UInt(Zn);
6 integer m = UInt(Zm);
7 integer da = UInt(ZAda);
8 boolean sub_op = TRUE;
9 boolean op1_unsigned = FALSE;
10 boolean op2_unsigned = FALSE;

```

64-bit (FEAT_SME_I16I64)



SMOPS [<ZAda>.D](#), [<Pn>/M](#), [<Pm>/M](#), [<Zn>.H](#), [<Zm>.H](#)

```

1 if !HaveSMEI16I64() then UNDEFINED;
2 integer esize = 64;
3 integer a = UInt(Pn);
4 integer b = UInt(Pm);
5 integer n = UInt(Zn);
6 integer m = UInt(Zm);
7 integer da = UInt(ZAda);
8 boolean sub_op = TRUE;
9 boolean op1_unsigned = FALSE;
10 boolean op2_unsigned = FALSE;

```

Assembler Symbols

- <ZAda> For the 32-bit variant: is the name of the ZA tile ZA0-ZA3, encoded in the "ZAda" field.
For the 64-bit variant: is the name of the ZA tile ZA0-ZA7, encoded in the "ZAda" field.
- <Pn> Is the name of the first governing scalable predicate register P0-P7, encoded in the "Pn" field.
- <Pm> Is the name of the second governing scalable predicate register P0-P7, encoded in the "Pm" field.
- <Zn> Is the name of the first source scalable vector register, encoded in the "Zn" field.
- <Zm> Is the name of the second source scalable vector register, encoded in the "Zm" field.

Operation

```

1 CheckStreamingSVEAndZAAEnabled();
2 integer dim = VL DIV esize;
3 bits(PL) mask1 = P[a];
4 bits(PL) mask2 = P[b];
5 bits(VL) operand1 = Z[n];
6 bits(VL) operand2 = Z[m];
7 bits(dim*dim*esize) operand3 = ZAtile[da, esize];
8 bits(dim*dim*esize) result;
9 integer prod;
10
11 for row = 0 to dim-1
12     for col = 0 to dim-1
13         bits(esize) sum = Elem[operand3, row*dim+col, esize];
14         for k = 0 to 3
15             if ElemP[mask1, 4*row + k, esize DIV 4] == '1' &&
16                 ElemP[mask2, 4*col + k, esize DIV 4] == '1' then
17                 prod = (Int(Elem[operand1, 4*row + k, esize DIV 4], op1_unsigned) *
18                     Int(Elem[operand2, 4*col + k, esize DIV 4], op2_unsigned));
19                 if sub_op then prod = -prod;
20                 sum = sum + prod;
21
22         Elem[result, row*dim+col, esize] = sum;
23
24 ZAtile[da, esize] = result;

```

Operational information

If FEAT_SVE2 is implemented or FEAT_SME is implemented, then when PSTATE.DIT is 1:

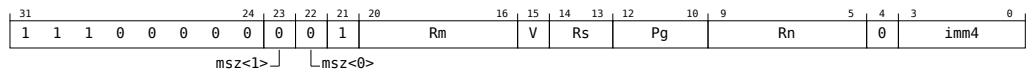
- The execution time of this instruction is independent of:
 - The values of the data supplied in any of its operand registers when its governing predicate registers contain the same value for each execution.
 - The values of the NZCV flags.
- The response of this instruction to asynchronous exceptions does not vary based on:
 - The values of the data supplied in any of its operand registers when its governing predicate registers contain the same value for each execution.
 - The values of the NZCV flags.

D1.1.24 ST1B

Contiguous store of bytes from 8-bit element ZA tile slice

The slice number within the tile is selected by the sum of the slice index register and an immediate, modulo the number of 8-bit elements in a Streaming SVE vector. The immediate is in the range 0 to 15. The memory address is generated by scalar base and optional scalar offset which is added to the base address. Inactive elements are not written to memory.

SME
(FEAT_SME)



ST1B { ZA0<HV>.B[<Ws>, <imm>] }, <Pg>, [<Xn|SP>{, <Xm>}]

```

1 if !HaveSME() then UNDEFINED;
2 integer n = UInt(Rn);
3 integer m = UInt(Rm);
4 integer g = UInt('0':Pg);
5 integer s = UInt('011':Rs);
6 integer t = 0;
7 integer imm = UInt(imm4);
8 integer esize = 8;
9 boolean vertical = V == '1';

```

Assembler Symbols

<HV> Is the horizontal or vertical slice indicator, encoded in "V":

V	<HV>
0	H
1	V

<Ws> Is the 32-bit name of the slice index register W12-W15, encoded in the "Rs" field.

<imm> Is the slice index offset, in the range 0 to 15, encoded in the "imm4" field.

<Pg> Is the name of the governing scalable predicate register P0-P7, encoded in the "Pg" field.

<Xn|SP> Is the 64-bit name of the general-purpose base register or stack pointer, encoded in the "Rn" field.

<Xm> Is the optional 64-bit name of the general-purpose offset register, defaulting to XZR, encoded in the "Rm" field.

Operation

```

1 CheckStreamingSVEAndZAEEnabled();
2 integer dim = VL DIV esize;
3 bits(64) base;
4 bits(64) addr;
5 bits(PL) mask = P[g];
6 bits(64) offset = X[m];
7 bits(32) index = X[s];
8 integer slice = (UInt(index) + imm) MOD dim;
9 bits(VL) src;
10 constant integer mbytes = esize DIV 8;
11
12 if HaveMTEExt() then SetTagCheckedInstruction(TRUE);
13
14 if n == 31 then
15     if AnyActiveElement(mask, esize) ||
16         ConstrainUnpredictableBool(Unpredictable_CHECKSPNONEACTIVE) then
17         CheckSPAlignment();
18         base = SP[];
19     else
20         base = X[n];

```

Chapter D1. SME instructions

D1.1. SME data-processing instructions

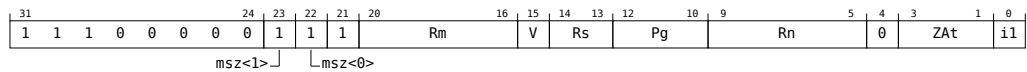
```
21
22 src = ZAslice[t, esize, vertical, slice];
23 for e = 0 to dim-1
24     addr = base + UInt(offset) * mbytes;
25     if ElemP[mask, e, esize] == '1' then
26         Mem[addr, mbytes, AccType_SME] = Elem[src, e, esize];
27     offset = offset + 1;
```

D1.1.25 ST1D

Contiguous store of doublewords from 64-bit element ZA tile slice

The slice number within the tile is selected by the sum of the slice index register and an immediate, modulo the number of 64-bit elements in a Streaming SVE vector. The immediate is in the range 0 to 1. The memory address is generated by scalar base and optional scalar offset which is multiplied by 8 and added to the base address. Inactive elements are not written to memory.

SME (FEAT_SME)



```
ST1D { <ZAt><HV>.D[<Ws>, <imm>] }, <Pg>, [<Xn|SP>{, <Xm>, LSL #3}]
```

```
1 if !HaveSME() then UNDEFINED;
2 integer n = UInt(Rn);
3 integer m = UInt(Rm);
4 integer g = UInt('0':Pg);
5 integer s = UInt('011':Rs);
6 integer t = UInt(ZAt);
7 integer imm = UInt(il);
8 integer esize = 64;
9 boolean vertical = V == '1';
```

Assembler Symbols

<ZAt> Is the name of the ZA tile ZA0-ZA7 to be accessed, encoded in the "ZAt" field.

<HV> Is the horizontal or vertical slice indicator, encoded in "V":

V	<HV>
0	H
1	V

<Ws> Is the 32-bit name of the slice index register W12-W15, encoded in the "Rs" field.

<imm> Is the slice index offset, in the range 0 to 1, encoded in the "il" field.

<Pg> Is the name of the governing scalable predicate register P0-P7, encoded in the "Pg" field.

<Xn|SP> Is the 64-bit name of the general-purpose base register or stack pointer, encoded in the "Rn" field.

<Xm> Is the optional 64-bit name of the general-purpose offset register, defaulting to XZR, encoded in the "Rm" field.

Operation

```
1 CheckStreamingSVEAndZEnabled();
2 integer dim = VL DIV esize;
3 bits(64) base;
4 bits(64) addr;
5 bits(PL) mask = P[g];
6 bits(64) offset = X[m];
7 bits(32) index = X[s];
8 integer slice = (UInt(index) + imm) MOD dim;
9 bits(VL) src;
10 constant integer mbytes = esize DIV 8;
11
12 if HaveMTEExt() then SetTagCheckedInstruction(TRUE);
13
14 if n == 31 then
15     if AnyActiveElement(mask, esize) ||
16         ConstrainUnpredictableBool(Unpredictable_CHECKSPNONEACTIVE) then
17         CheckSPAlignment();
18     base = SP[];
```

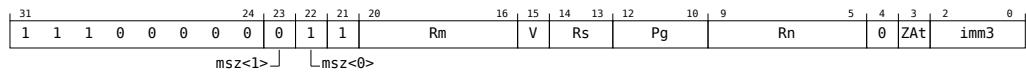
```
19  else
20      base = X[n];
21
22  src = ZAslice[t, esize, vertical, slice];
23  for e = 0 to dim-1
24      addr = base + UInt(offset) * mbytes;
25      if ElemP[mask, e, esize] == '1' then
26          Mem[addr, mbytes, AccType_SME] = Elem[src, e, esize];
27      offset = offset + 1;
```

D1.1.26 ST1H

Contiguous store of halfwords from 16-bit element ZA tile slice

The slice number within the tile is selected by the sum of the slice index register and an immediate, modulo the number of 16-bit elements in a Streaming SVE vector. The immediate is in the range 0 to 7. The memory address is generated by scalar base and optional scalar offset which is multiplied by 2 and added to the base address. Inactive elements are not written to memory.

SME (FEAT_SME)



```
ST1H { <ZAt><HV>.H[<Ws>, <imm>] }, <Pg>, [<Xn|SP>{, <Xm>, LSL #1}]
```

```
1 if !HaveSME() then UNDEFINED;
2 integer n = UInt(Rn);
3 integer m = UInt(Rm);
4 integer g = UInt('0':Pg);
5 integer s = UInt('011':Rs);
6 integer t = UInt(ZAt);
7 integer imm = UInt(imm3);
8 integer esize = 16;
9 boolean vertical = V == '1';
```

Assembler Symbols

<ZAt> Is the name of the ZA tile ZA0-ZA1 to be accessed, encoded in the "ZAt" field.

<HV> Is the horizontal or vertical slice indicator, encoded in "V":

V	<HV>
0	H
1	V

<Ws> Is the 32-bit name of the slice index register W12-W15, encoded in the "Rs" field.

<imm> Is the slice index offset, in the range 0 to 7, encoded in the "imm3" field.

<Pg> Is the name of the governing scalable predicate register P0-P7, encoded in the "Pg" field.

<Xn|SP> Is the 64-bit name of the general-purpose base register or stack pointer, encoded in the "Rn" field.

<Xm> Is the optional 64-bit name of the general-purpose offset register, defaulting to XZR, encoded in the "Rm" field.

Operation

```
1 CheckStreamingSVEAndZEnabled();
2 integer dim = VL DIV esize;
3 bits(64) base;
4 bits(64) addr;
5 bits(PL) mask = P[g];
6 bits(64) offset = X[m];
7 bits(32) index = X[s];
8 integer slice = (UInt(index) + imm) MOD dim;
9 bits(VL) src;
10 constant integer mbytes = esize DIV 8;
11
12 if HaveMTEExt() then SetTagCheckedInstruction(TRUE);
13
14 if n == 31 then
15     if AnyActiveElement(mask, esize) ||
16         ConstrainUnpredictableBool(Unpredictable_CHECKSPNONEACTIVE) then
17         CheckSPAlignment();
18     base = SP[];
```

Chapter D1. SME instructions

D1.1. SME data-processing instructions

```
19  else
20      base = X[n];
21
22  src = ZAslice[t, esize, vertical, slice];
23  for e = 0 to dim-1
24      addr = base + UInt(offset) * mbytes;
25      if ElemP[mask, e, esize] == '1' then
26          Mem[addr, mbytes, AccType_SME] = Elem[src, e, esize];
27      offset = offset + 1;
```


D1.1.27 ST1Q

Contiguous store of quadwords from 128-bit element ZA tile slice

The slice number in the tile is selected by the slice index register, modulo the number of 128-bit elements in a Streaming SVE vector. The memory address is generated by scalar base and optional scalar offset which is multiplied by 16 and added to the base address. Inactive elements are not written to memory.

SME
(FEAT_SME)

31											21	20	16			15	14	13	12	10		9	5		4	3	0
1	1	1	0	0	0	0	1	1	1	1	Rm			V	Rs	Pg			Rn			0	ZAt				

```
ST1Q { <ZAt><HV>.Q[<Ws>, <imm>] }, <Pg>, [<Xn|SP>{, <Xm>, LSL #4}]
```

```
1 if !HaveSME() then UNDEFINED;
2 integer n = UInt(Rn);
3 integer m = UInt(Rm);
4 integer g = UInt('0':Pg);
5 integer s = UInt('011':Rs);
6 integer t = UInt(ZAt);
7 integer imm = 0;
8 integer esize = 128;
9 boolean vertical = V == '1';
```

Assembler Symbols

- <ZAt> Is the name of the ZA tile ZA0-ZA15 to be accessed, encoded in the "ZAt" field.
- <HV> Is the horizontal or vertical slice indicator, encoded in "V":

V	<HV>
0	H
1	V
- <Ws> Is the 32-bit name of the slice index register W12-W15, encoded in the "Rs" field.
- <imm> Is the slice index offset 0.
- <Pg> Is the name of the governing scalable predicate register P0-P7, encoded in the "Pg" field.
- <Xn|SP> Is the 64-bit name of the general-purpose base register or stack pointer, encoded in the "Rn" field.
- <Xm> Is the optional 64-bit name of the general-purpose offset register, defaulting to XZR, encoded in the "Rm" field.

Operation

```
1 CheckStreamingSVEAndZEnabled();
2 integer dim = VL DIV esize;
3 bits(64) base;
4 bits(64) addr;
5 bits(PL) mask = P[g];
6 bits(64) offset = X[m];
7 bits(32) index = X[s];
8 integer slice = (UInt(index) + imm) MOD dim;
9 bits(VL) src;
10 constant integer mbytes = esize DIV 8;
11
12 if HaveMTEExt() then SetTagCheckedInstruction(TRUE);
13
14 if n == 31 then
15     if AnyActiveElement(mask, esize) ||
16         ConstrainUnpredictableBool(Unpredictable_CHECKSPNONEACTIVE) then
17         CheckSPAlignment();
18         base = SP[];
19     else
20         base = X[n];
```

Chapter D1. SME instructions

D1.1. SME data-processing instructions

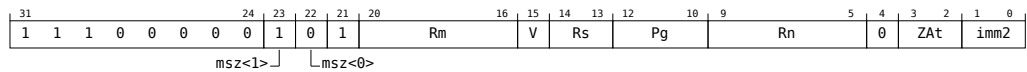
```
21
22 src = ZAslice[t, esize, vertical, slice];
23 for e = 0 to dim-1
24     addr = base + UInt(offset) * mbytes;
25     if ElemP[mask, e, esize] == '1' then
26         Mem[addr, mbytes, AccType_SME] = Elem[src, e, esize];
27     offset = offset + 1;
```

D1.1.28 ST1W

Contiguous store of words from 32-bit element ZA tile slice

The slice number within the tile is selected by the sum of the slice index register and an immediate, modulo the number of 32-bit elements in a Streaming SVE vector. The immediate is in the range 0 to 3. The memory address is generated by scalar base and optional scalar offset which is multiplied by 4 and added to the base address. Inactive elements are not written to memory.

SME
(FEAT_SME)



```
ST1W    { <ZAt><HV>.S[<Ws>, <imm>] }, <Pg>, [<Xn|SP>{, <Xm>, LSL #2}]
```

```
1  if !HaveSME() then UNDEFINED;
2  integer n = UInt(Rn);
3  integer m = UInt(Rm);
4  integer g = UInt('0':Pg);
5  integer s = UInt('011':Rs);
6  integer t = UInt(ZAt);
7  integer imm = UInt(imm2);
8  integer esize = 32;
9  boolean vertical = V == '1';
```

Assembler Symbols

- <ZAt> Is the name of the ZA tile ZA0-ZA3 to be accessed, encoded in the "ZAt" field.
- <HV> Is the horizontal or vertical slice indicator, encoded in "V":

V	<HV>
0	H
1	V
- <Ws> Is the 32-bit name of the slice index register W12-W15, encoded in the "Rs" field.
- <imm> Is the slice index offset, in the range 0 to 3, encoded in the "imm2" field.
- <Pg> Is the name of the governing scalable predicate register P0-P7, encoded in the "Pg" field.
- <Xn|SP> Is the 64-bit name of the general-purpose base register or stack pointer, encoded in the "Rn" field.
- <Xm> Is the optional 64-bit name of the general-purpose offset register, defaulting to XZR, encoded in the "Rm" field.

Operation

```
1  CheckStreamingSVEAndZEnabled();
2  integer dim = VL DIV esize;
3  bits(64) base;
4  bits(64) addr;
5  bits(PL) mask = P[g];
6  bits(64) offset = X[m];
7  bits(32) index = X[s];
8  integer slice = (UInt(index) + imm) MOD dim;
9  bits(VL) src;
10 constant integer mbytes = esize DIV 8;
11
12 if HaveMTEExt() then SetTagCheckedInstruction(TRUE);
13
14 if n == 31 then
15     if AnyActiveElement(mask, esize) ||
16         ConstrainUnpredictableBool(Unpredictable_CHECKSPNONEACTIVE) then
17         CheckSPAlignment();
18     base = SP[];
```

```
19  else
20      base = X[n];
21
22  src = ZAslice[t, esize, vertical, slice];
23  for e = 0 to dim-1
24      addr = base + UInt(offset) * mbytes;
25      if ElemP[mask, e, esize] == '1' then
26          Mem[addr, mbytes, AccType_SME] = Elem[src, e, esize];
27      offset = offset + 1;
```

D1.1.29 STR

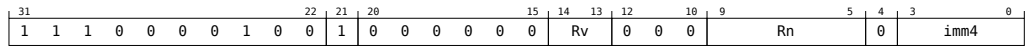
Store vector from ZA array

The ZA array vector is selected by the sum of the vector select register and an immediate, modulo the number of bytes in a Streaming SVE vector. The immediate is in the range 0 to 15. The memory address is generated by scalar base, plus the same optional immediate offset multiplied by the current vector length in bytes. This instruction is unpredicated.

The store is performed as contiguous byte accesses, with no endian conversion and no guarantee of single-copy atomicity larger than a byte. However, if alignment is checked, then the base register must be aligned to 16 bytes.

This instruction does not require the PE to be in Streaming SVE mode, and it is expected that this instruction will not experience a significant slowdown due to contention with other PEs that are executing in Streaming SVE mode.

SME
(FEAT_SME)



STR ZA[<Wv>, <imm>], [<Xn|SP>{, #<imm>, MUL VL}]

```
1 if !HaveSME() then UNDEFINED;
2 integer n = UInt(Rn);
3 integer v = UInt('011':Rv);
4 integer imm = UInt(imm4);
```

Assembler Symbols

- <Wv> Is the 32-bit name of the vector select register W12-W15, encoded in the "Rv" field.
- <imm> Is the vector select offset and optional memory offset, in the range 0 to 15, defaulting to 0, encoded in the "imm4" field.
- <Xn|SP> Is the 64-bit name of the general-purpose base register or stack pointer, encoded in the "Rn" field.

Operation

```
1 CheckSMEAndZAEEnabled();
2 integer dim = SVL DIV 8;
3 bits(32) idx = X[v];
4 integer vec = (UInt(idx) + imm) MOD dim;
5 bits(SVL) src;
6 bits(64) base;
7 integer offset = imm * dim;
8
9 if HaveTME() && TSTATE.depth > 0 then
10     FailTransaction(TMFailure_ERR, FALSE);
11
12 if n == 31 then
13     if HaveMTEExt() then SetTagCheckedInstruction(FALSE);
14     CheckSPAlignment();
15     base = SP[];
16 else
17     if HaveMTEExt() then SetTagCheckedInstruction(TRUE);
18     base = X[n];
19
20 src = ZAVector[vec];
21 boolean aligned = AArch64.CheckAlignment(base + offset, 16, AccType_SME, TRUE);
22 for e = 0 to dim-1
23     AArch64.MemSingle[base + offset, 1, AccType_SME, aligned] = Elem[src, e, 8];
24     offset = offset + 1;
```

D1.1.30 SUMOPA

Signed by unsigned integer sum of outer products and accumulate

The 8-bit integer variant works with a 32-bit element ZA tile.

The 16-bit integer variant works with a 64-bit element ZA tile.

The signed by unsigned integer sum of outer products and accumulate instructions multiply the sub-matrix in the first source vector by the sub-matrix in the second source vector. In case of the 8-bit integer variant, the first source holds $SVL_S \times 4$ sub-matrix of signed 8-bit integer values, and the second source holds $4 \times SVL_S$ sub-matrix of unsigned 8-bit integer values. In case of the 16-bit integer variant, the first source holds $SVL_D \times 4$ sub-matrix of signed 16-bit integer values, and the second source holds $4 \times SVL_D$ sub-matrix of unsigned 16-bit integer values.

Each source vector is independently predicated by a corresponding governing predicate. When an 8-bit source element in case of 8-bit integer variant or a 16-bit source element in case of 16-bit integer variant is Inactive, it is treated as having the value 0.

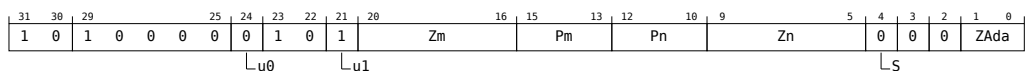
The resulting $SVL_S \times SVL_S$ widened 32-bit integer or $SVL_D \times SVL_D$ widened 64-bit integer sum of outer products is then destructively added to the 32-bit integer or 64-bit integer destination tile, respectively for 8-bit integer and 16-bit integer instruction variants. This is equivalent to performing a 4-way dot product and accumulate to each of the destination tile elements.

In case of the 8-bit integer variant, each 32-bit container of first source vector holds 4 consecutive column elements of each row of a $SVL_S \times 4$ sub-matrix, and each 32-bit container of second source vector holds 4 consecutive row elements of each column of a $4 \times SVL_S$ sub-matrix. In case of the 16-bit integer variant, each 64-bit container of first source vector holds 4 consecutive column elements of each row of a $SVL_D \times 4$ sub-matrix, and each 64-bit container of second source vector holds 4 consecutive row elements of each column of a $4 \times SVL_D$ sub-matrix.

ID_AA64SMFR0_EL1.I16I64 indicates whether the 16-bit integer variant is implemented.

It has encodings from 2 classes: [32-bit](#) and [64-bit](#)

32-bit (FEAT_SME)



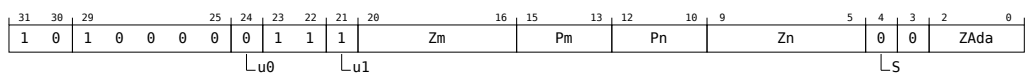
SUMOPA <ZAda>.S, <Pn>/M, <Pm>/M, <Zn>.B, <Zm>.B

```

1 if !HaveSME() then UNDEFINED;
2 integer esize = 32;
3 integer a = UInt(Pn);
4 integer b = UInt(Pm);
5 integer n = UInt(Zn);
6 integer m = UInt(Zm);
7 integer da = UInt(ZAda);
8 boolean sub_op = FALSE;
9 boolean op1_unsigned = FALSE;
10 boolean op2_unsigned = TRUE;

```

64-bit (FEAT_SME_I16I64)



SUMOPA <ZAda>.D, <Pn>/M, <Pm>/M, <Zn>.H, <Zm>.H

```

1 if !HaveSMEI16I64() then UNDEFINED;
2 integer esize = 64;
3 integer a = UInt(Pn);
4 integer b = UInt(Pm);
5 integer n = UInt(Zn);
6 integer m = UInt(Zm);
7 integer da = UInt(ZAda);
8 boolean sub_op = FALSE;
9 boolean op1_unsigned = FALSE;
10 boolean op2_unsigned = TRUE;

```

Assembler Symbols

- <ZAda> For the 32-bit variant: is the name of the ZA tile ZA0-ZA3, encoded in the "ZAda" field.
For the 64-bit variant: is the name of the ZA tile ZA0-ZA7, encoded in the "ZAda" field.
- <Pn> Is the name of the first governing scalable predicate register P0-P7, encoded in the "Pn" field.
- <Pm> Is the name of the second governing scalable predicate register P0-P7, encoded in the "Pm" field.
- <Zn> Is the name of the first source scalable vector register, encoded in the "Zn" field.
- <Zm> Is the name of the second source scalable vector register, encoded in the "Zm" field.

Operation

```

1 CheckStreamingSVEAndZAAEnabled();
2 integer dim = VL DIV esize;
3 bits(PL) mask1 = P[a];
4 bits(PL) mask2 = P[b];
5 bits(VL) operand1 = Z[n];
6 bits(VL) operand2 = Z[m];
7 bits(dim*dim*esize) operand3 = Ztile[da, esize];
8 bits(dim*dim*esize) result;
9 integer prod;
10
11 for row = 0 to dim-1
12     for col = 0 to dim-1
13         bits(esize) sum = Elem[operand3, row*dim+col, esize];
14         for k = 0 to 3
15             if ElemP[mask1, 4*row + k, esize DIV 4] == '1' &&
16                 ElemP[mask2, 4*col + k, esize DIV 4] == '1' then
17                 prod = (Int(Elem[operand1, 4*row + k, esize DIV 4], op1_unsigned) *
18                     Int(Elem[operand2, 4*col + k, esize DIV 4], op2_unsigned));
19                 if sub_op then prod = -prod;
20                 sum = sum + prod;
21
22         Elem[result, row*dim+col, esize] = sum;
23
24 Ztile[da, esize] = result;

```

Operational information

If FEAT_SVE2 is implemented or FEAT_SME is implemented, then when PSTATE.DIT is 1:

- The execution time of this instruction is independent of:
 - The values of the data supplied in any of its operand registers when its governing predicate registers contain the same value for each execution.
 - The values of the NZCV flags.
- The response of this instruction to asynchronous exceptions does not vary based on:
 - The values of the data supplied in any of its operand registers when its governing predicate registers contain the same value for each execution.
 - The values of the NZCV flags.

D1.1.31 SUMOPS

Signed by unsigned integer sum of outer products and subtract

The 8-bit integer variant works with a 32-bit element ZA tile.

The 16-bit integer variant works with a 64-bit element ZA tile.

The signed by unsigned integer sum of outer products and subtract instructions multiply the sub-matrix in the first source vector by the sub-matrix in the second source vector. In case of the 8-bit integer variant, the first source holds $SVL_S \times 4$ sub-matrix of signed 8-bit integer values, and the second source holds $4 \times SVL_S$ sub-matrix of unsigned 8-bit integer values. In case of the 16-bit integer variant, the first source holds $SVL_D \times 4$ sub-matrix of signed 16-bit integer values, and the second source holds $4 \times SVL_D$ sub-matrix of unsigned 16-bit integer values.

Each source vector is independently predicated by a corresponding governing predicate. When an 8-bit source element in case of 8-bit integer variant or a 16-bit source element in case of 16-bit integer variant is Inactive, it is treated as having the value 0.

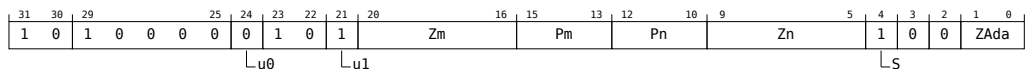
The resulting $SVL_S \times SVL_S$ widened 32-bit integer or $SVL_D \times SVL_D$ widened 64-bit integer sum of outer products is then destructively subtracted from the 32-bit integer or 64-bit integer destination tile, respectively for 8-bit integer and 16-bit integer instruction variants. This is equivalent to performing a 4-way dot product and subtract from each of the destination tile elements.

In case of the 8-bit integer variant, each 32-bit container of first source vector holds 4 consecutive column elements of each row of a $SVL_S \times 4$ sub-matrix, and each 32-bit container of second source vector holds 4 consecutive row elements of each column of a $4 \times SVL_S$ sub-matrix. In case of the 16-bit integer variant, each 64-bit container of first source vector holds 4 consecutive column elements of each row of a $SVL_D \times 4$ sub-matrix, and each 64-bit container of second source vector holds 4 consecutive row elements of each column of a $4 \times SVL_D$ sub-matrix.

ID_AA64SMFR0_EL1.I16I64 indicates whether the 16-bit integer variant is implemented.

It has encodings from 2 classes: [32-bit](#) and [64-bit](#)

32-bit (FEAT_SME)



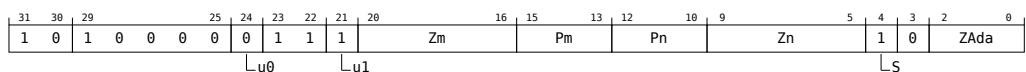
SUMOPS <ZAda>.S, <Pn>/M, <Pm>/M, <Zn>.B, <Zm>.B

```

1 if !HaveSME() then UNDEFINED;
2 integer esize = 32;
3 integer a = UInt(Pn);
4 integer b = UInt(Pm);
5 integer n = UInt(Zn);
6 integer m = UInt(Zm);
7 integer da = UInt(ZAda);
8 boolean sub_op = TRUE;
9 boolean op1_unsigned = FALSE;
10 boolean op2_unsigned = TRUE;

```

64-bit (FEAT_SME_I16I64)



SUMOPS <ZAda>.D, <Pn>/M, <Pm>/M, <Zn>.H, <Zm>.H


```

1 if !HaveSMEI16I64() then UNDEFINED;
2 integer esize = 64;
3 integer a = UInt(Pn);
4 integer b = UInt(Pm);
5 integer n = UInt(Zn);
6 integer m = UInt(Zm);
7 integer da = UInt(ZAda);
8 boolean sub_op = TRUE;
9 boolean op1_unsigned = FALSE;
10 boolean op2_unsigned = TRUE;

```

Assembler Symbols

- <ZAda> For the 32-bit variant: is the name of the ZA tile ZA0-ZA3, encoded in the "ZAda" field.
For the 64-bit variant: is the name of the ZA tile ZA0-ZA7, encoded in the "ZAda" field.
- <Pn> Is the name of the first governing scalable predicate register P0-P7, encoded in the "Pn" field.
- <Pm> Is the name of the second governing scalable predicate register P0-P7, encoded in the "Pm" field.
- <Zn> Is the name of the first source scalable vector register, encoded in the "Zn" field.
- <Zm> Is the name of the second source scalable vector register, encoded in the "Zm" field.

Operation

```

1 CheckStreamingSVEAndZAAEnabled();
2 integer dim = VL DIV esize;
3 bits(PL) mask1 = P[a];
4 bits(PL) mask2 = P[b];
5 bits(VL) operand1 = Z[n];
6 bits(VL) operand2 = Z[m];
7 bits(dim*dim*esize) operand3 = Ztile[da, esize];
8 bits(dim*dim*esize) result;
9 integer prod;
10
11 for row = 0 to dim-1
12     for col = 0 to dim-1
13         bits(esize) sum = Elem[operand3, row*dim+col, esize];
14         for k = 0 to 3
15             if ElemP[mask1, 4*row + k, esize DIV 4] == '1' &&
16                 ElemP[mask2, 4*col + k, esize DIV 4] == '1' then
17                 prod = (Int(Elem[operand1, 4*row + k, esize DIV 4], op1_unsigned) *
18                     Int(Elem[operand2, 4*col + k, esize DIV 4], op2_unsigned));
19                 if sub_op then prod = -prod;
20                 sum = sum + prod;
21
22         Elem[result, row*dim+col, esize] = sum;
23
24 Ztile[da, esize] = result;

```

Operational information

If FEAT_SVE2 is implemented or FEAT_SME is implemented, then when PSTATE.DIT is 1:

- The execution time of this instruction is independent of:
 - The values of the data supplied in any of its operand registers when its governing predicate registers contain the same value for each execution.
 - The values of the NZCV flags.
- The response of this instruction to asynchronous exceptions does not vary based on:
 - The values of the data supplied in any of its operand registers when its governing predicate registers contain the same value for each execution.
 - The values of the NZCV flags.

D1.1.32 UMOPA

Unsigned integer sum of outer products and accumulate

The 8-bit integer variant works with a 32-bit element ZA tile.

The 16-bit integer variant works with a 64-bit element ZA tile.

The unsigned integer sum of outer products and accumulate instructions multiply the sub-matrix in the first source vector by the sub-matrix in the second source vector. In case of the 8-bit integer variant, the first source holds $SVL_S \times 4$ sub-matrix of unsigned 8-bit integer values, and the second source holds $4 \times SVL_S$ sub-matrix of unsigned 8-bit integer values. In case of the 16-bit integer variant, the first source holds $SVL_D \times 4$ sub-matrix of unsigned 16-bit integer values, and the second source holds $4 \times SVL_D$ sub-matrix of unsigned 16-bit integer values.

Each source vector is independently predicated by a corresponding governing predicate. When an 8-bit source element in case of 8-bit integer variant or a 16-bit source element in case of 16-bit integer variant is Inactive, it is treated as having the value 0.

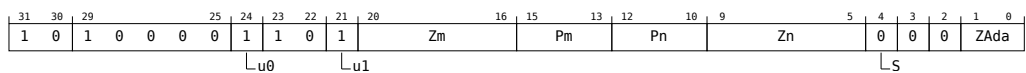
The resulting $SVL_S \times SVL_S$ widened 32-bit integer or $SVL_D \times SVL_D$ widened 64-bit integer sum of outer products is then destructively added to the 32-bit integer or 64-bit integer destination tile, respectively for 8-bit integer and 16-bit integer instruction variants. This is equivalent to performing a 4-way dot product and accumulate to each of the destination tile elements.

In case of the 8-bit integer variant, each 32-bit container of first source vector holds 4 consecutive column elements of each row of a $SVL_S \times 4$ sub-matrix, and each 32-bit container of second source vector holds 4 consecutive row elements of each column of a $4 \times SVL_S$ sub-matrix. In case of the 16-bit integer variant, each 64-bit container of first source vector holds 4 consecutive column elements of each row of a $SVL_D \times 4$ sub-matrix, and each 64-bit container of second source vector holds 4 consecutive row elements of each column of a $4 \times SVL_D$ sub-matrix.

ID_AA64SMFR0_EL1.I16I64 indicates whether the 16-bit integer variant is implemented.

It has encodings from 2 classes: [32-bit](#) and [64-bit](#)

32-bit (FEAT_SME)



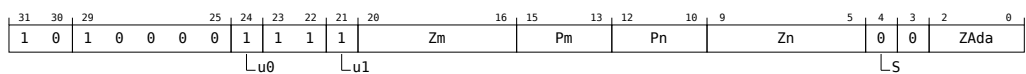
UMOPA <ZAda>.S, <Pn>/M, <Pm>/M, <Zn>.B, <Zm>.B

```

1 if !HaveSME() then UNDEFINED;
2 integer esize = 32;
3 integer a = UInt(Pn);
4 integer b = UInt(Pm);
5 integer n = UInt(Zn);
6 integer m = UInt(Zm);
7 integer da = UInt(ZAda);
8 boolean sub_op = FALSE;
9 boolean op1_unsigned = TRUE;
10 boolean op2_unsigned = TRUE;

```

64-bit (FEAT_SME_I16I64)



UMOPA <ZAda>.D, <Pn>/M, <Pm>/M, <Zn>.H, <Zm>.H

```

1 if !HaveSMEI16I64() then UNDEFINED;
2 integer esize = 64;
3 integer a = UInt(Pn);
4 integer b = UInt(Pm);
5 integer n = UInt(Zn);
6 integer m = UInt(Zm);
7 integer da = UInt(ZAda);
8 boolean sub_op = FALSE;
9 boolean op1_unsigned = TRUE;
10 boolean op2_unsigned = TRUE;

```

Assembler Symbols

- <ZAda> For the 32-bit variant: is the name of the ZA tile ZA0-ZA3, encoded in the "ZAda" field.
For the 64-bit variant: is the name of the ZA tile ZA0-ZA7, encoded in the "ZAda" field.
- <Pn> Is the name of the first governing scalable predicate register P0-P7, encoded in the "Pn" field.
- <Pm> Is the name of the second governing scalable predicate register P0-P7, encoded in the "Pm" field.
- <Zn> Is the name of the first source scalable vector register, encoded in the "Zn" field.
- <Zm> Is the name of the second source scalable vector register, encoded in the "Zm" field.

Operation

```

1 CheckStreamingSVEAndZAAEnabled();
2 integer dim = VL DIV esize;
3 bits(PL) mask1 = P[a];
4 bits(PL) mask2 = P[b];
5 bits(VL) operand1 = Z[n];
6 bits(VL) operand2 = Z[m];
7 bits(dim*dim*esize) operand3 = Ztile[da, esize];
8 bits(dim*dim*esize) result;
9 integer prod;
10
11 for row = 0 to dim-1
12     for col = 0 to dim-1
13         bits(esize) sum = Elem[operand3, row*dim+col, esize];
14         for k = 0 to 3
15             if ElemP[mask1, 4*row + k, esize DIV 4] == '1' &&
16                 ElemP[mask2, 4*col + k, esize DIV 4] == '1' then
17                 prod = (Int(Elem[operand1, 4*row + k, esize DIV 4], op1_unsigned) *
18                     Int(Elem[operand2, 4*col + k, esize DIV 4], op2_unsigned));
19                 if sub_op then prod = -prod;
20                 sum = sum + prod;
21
22         Elem[result, row*dim+col, esize] = sum;
23
24 Ztile[da, esize] = result;

```

Operational information

If FEAT_SVE2 is implemented or FEAT_SME is implemented, then when PSTATE.DIT is 1:

- The execution time of this instruction is independent of:
 - The values of the data supplied in any of its operand registers when its governing predicate registers contain the same value for each execution.
 - The values of the NZCV flags.
- The response of this instruction to asynchronous exceptions does not vary based on:
 - The values of the data supplied in any of its operand registers when its governing predicate registers contain the same value for each execution.
 - The values of the NZCV flags.

D1.1.33 UMOPS

Unsigned integer sum of outer products and subtract

The 8-bit integer variant works with a 32-bit element ZA tile.

The 16-bit integer variant works with a 64-bit element ZA tile.

The unsigned integer sum of outer products and subtract instructions multiply the sub-matrix in the first source vector by the sub-matrix in the second source vector. In case of the 8-bit integer variant, the first source holds $SVL_S \times 4$ sub-matrix of unsigned 8-bit integer values, and the second source holds $4 \times SVL_S$ sub-matrix of unsigned 8-bit integer values. In case of the 16-bit integer variant, the first source holds $SVL_D \times 4$ sub-matrix of unsigned 16-bit integer values, and the second source holds $4 \times SVL_D$ sub-matrix of unsigned 16-bit integer values.

Each source vector is independently predicated by a corresponding governing predicate. When an 8-bit source element in case of 8-bit integer variant or a 16-bit source element in case of 16-bit integer variant is Inactive, it is treated as having the value 0.

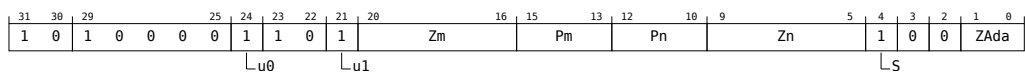
The resulting $SVL_S \times SVL_S$ widened 32-bit integer or $SVL_D \times SVL_D$ widened 64-bit integer sum of outer products is then destructively subtracted from the 32-bit integer or 64-bit integer destination tile, respectively for 8-bit integer and 16-bit integer instruction variants. This is equivalent to performing a 4-way dot product and subtract from each of the destination tile elements.

In case of the 8-bit integer variant, each 32-bit container of first source vector holds 4 consecutive column elements of each row of a $SVL_S \times 4$ sub-matrix, and each 32-bit container of second source vector holds 4 consecutive row elements of each column of a $4 \times SVL_S$ sub-matrix. In case of the 16-bit integer variant, each 64-bit container of first source vector holds 4 consecutive column elements of each row of a $SVL_D \times 4$ sub-matrix, and each 64-bit container of second source vector holds 4 consecutive row elements of each column of a $4 \times SVL_D$ sub-matrix.

ID_AA64SMFR0_EL1.I16I64 indicates whether the 16-bit integer variant is implemented.

It has encodings from 2 classes: [32-bit](#) and [64-bit](#)

32-bit (FEAT_SME)



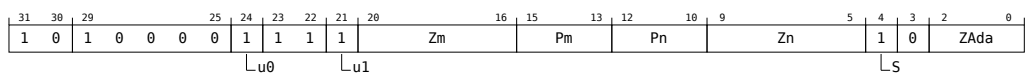
UMOPS [<ZAda>.S](#), [<Pn>/M](#), [<Pm>/M](#), [<Zn>.B](#), [<Zm>.B](#)

```

1 if !HaveSME() then UNDEFINED;
2 integer esize = 32;
3 integer a = UInt(Pn);
4 integer b = UInt(Pm);
5 integer n = UInt(Zn);
6 integer m = UInt(Zm);
7 integer da = UInt(ZAda);
8 boolean sub_op = TRUE;
9 boolean op1_unsigned = TRUE;
10 boolean op2_unsigned = TRUE;

```

64-bit (FEAT_SME_I16I64)



UMOPS [<ZAda>.D](#), [<Pn>/M](#), [<Pm>/M](#), [<Zn>.H](#), [<Zm>.H](#)

```

1 if !HaveSMEI16I64() then UNDEFINED;
2 integer esize = 64;
3 integer a = UInt(Pn);
4 integer b = UInt(Pm);
5 integer n = UInt(Zn);
6 integer m = UInt(Zm);
7 integer da = UInt(ZAda);
8 boolean sub_op = TRUE;
9 boolean op1_unsigned = TRUE;
10 boolean op2_unsigned = TRUE;

```

Assembler Symbols

- <ZAda> For the 32-bit variant: is the name of the ZA tile ZA0-ZA3, encoded in the "ZAda" field.
For the 64-bit variant: is the name of the ZA tile ZA0-ZA7, encoded in the "ZAda" field.
- <Pn> Is the name of the first governing scalable predicate register P0-P7, encoded in the "Pn" field.
- <Pm> Is the name of the second governing scalable predicate register P0-P7, encoded in the "Pm" field.
- <Zn> Is the name of the first source scalable vector register, encoded in the "Zn" field.
- <Zm> Is the name of the second source scalable vector register, encoded in the "Zm" field.

Operation

```

1 CheckStreamingSVEAndZAAEnabled();
2 integer dim = VL DIV esize;
3 bits(PL) mask1 = P[a];
4 bits(PL) mask2 = P[b];
5 bits(VL) operand1 = Z[n];
6 bits(VL) operand2 = Z[m];
7 bits(dim*dim*esize) operand3 = ZAtile[da, esize];
8 bits(dim*dim*esize) result;
9 integer prod;
10
11 for row = 0 to dim-1
12     for col = 0 to dim-1
13         bits(esize) sum = Elem[operand3, row*dim+col, esize];
14         for k = 0 to 3
15             if ElemP[mask1, 4*row + k, esize DIV 4] == '1' &&
16                 ElemP[mask2, 4*col + k, esize DIV 4] == '1' then
17                 prod = (Int(Elem[operand1, 4*row + k, esize DIV 4], op1_unsigned) *
18                     Int(Elem[operand2, 4*col + k, esize DIV 4], op2_unsigned));
19                 if sub_op then prod = -prod;
20                 sum = sum + prod;
21
22         Elem[result, row*dim+col, esize] = sum;
23
24 ZAtile[da, esize] = result;

```

Operational information

If FEAT_SVE2 is implemented or FEAT_SME is implemented, then when PSTATE.DIT is 1:

- The execution time of this instruction is independent of:
 - The values of the data supplied in any of its operand registers when its governing predicate registers contain the same value for each execution.
 - The values of the NZCV flags.
- The response of this instruction to asynchronous exceptions does not vary based on:
 - The values of the data supplied in any of its operand registers when its governing predicate registers contain the same value for each execution.
 - The values of the NZCV flags.

D1.1.34 USMOPA

Unsigned by signed integer sum of outer products and accumulate

The 8-bit integer variant works with a 32-bit element ZA tile.

The 16-bit integer variant works with a 64-bit element ZA tile.

The unsigned by signed integer sum of outer products and accumulate instructions multiply the sub-matrix in the first source vector by the sub-matrix in the second source vector. In case of the 8-bit integer variant, the first source holds $SVL_S \times 4$ sub-matrix of unsigned 8-bit integer values, and the second source holds $4 \times SVL_S$ sub-matrix of signed 8-bit integer values. In case of the 16-bit integer variant, the first source holds $SVL_D \times 4$ sub-matrix of unsigned 16-bit integer values, and the second source holds $4 \times SVL_D$ sub-matrix of signed 16-bit integer values.

Each source vector is independently predicated by a corresponding governing predicate. When an 8-bit source element in case of 8-bit integer variant or a 16-bit source element in case of 16-bit integer variant is Inactive, it is treated as having the value 0.

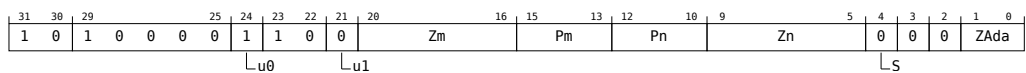
The resulting $SVL_S \times SVL_S$ widened 32-bit integer or $SVL_D \times SVL_D$ widened 64-bit integer sum of outer products is then destructively added to the 32-bit integer or 64-bit integer destination tile, respectively for 8-bit integer and 16-bit integer instruction variants. This is equivalent to performing a 4-way dot product and accumulate to each of the destination tile elements.

In case of the 8-bit integer variant, each 32-bit container of first source vector holds 4 consecutive column elements of each row of a $SVL_S \times 4$ sub-matrix, and each 32-bit container of second source vector holds 4 consecutive row elements of each column of a $4 \times SVL_S$ sub-matrix. In case of the 16-bit integer variant, each 64-bit container of first source vector holds 4 consecutive column elements of each row of a $SVL_D \times 4$ sub-matrix, and each 64-bit container of second source vector holds 4 consecutive row elements of each column of a $4 \times SVL_D$ sub-matrix.

ID_AA64SMFR0_EL1.I16I64 indicates whether the 16-bit integer variant is implemented.

It has encodings from 2 classes: [32-bit](#) and [64-bit](#)

32-bit (FEAT_SME)



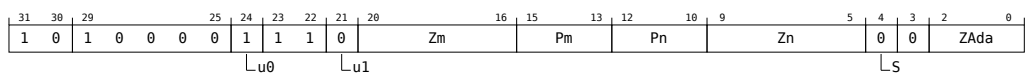
USMOPA <ZAda>.S, <Pn>/M, <Pm>/M, <Zn>.B, <Zm>.B

```

1 if !HaveSME() then UNDEFINED;
2 integer esize = 32;
3 integer a = UInt(Pn);
4 integer b = UInt(Pm);
5 integer n = UInt(Zn);
6 integer m = UInt(Zm);
7 integer da = UInt(ZAda);
8 boolean sub_op = FALSE;
9 boolean op1_unsigned = TRUE;
10 boolean op2_unsigned = FALSE;

```

64-bit (FEAT_SME_I16I64)



USMOPA <ZAda>.D, <Pn>/M, <Pm>/M, <Zn>.H, <Zm>.H

```

1 if !HaveSMEI16I64() then UNDEFINED;
2 integer esize = 64;
3 integer a = UInt(Pn);
4 integer b = UInt(Pm);
5 integer n = UInt(Zn);
6 integer m = UInt(Zm);
7 integer da = UInt(ZAda);
8 boolean sub_op = FALSE;
9 boolean op1_unsigned = TRUE;
10 boolean op2_unsigned = FALSE;

```

Assembler Symbols

- <ZAda> For the 32-bit variant: is the name of the ZA tile ZA0-ZA3, encoded in the "ZAda" field.
For the 64-bit variant: is the name of the ZA tile ZA0-ZA7, encoded in the "ZAda" field.
- <Pn> Is the name of the first governing scalable predicate register P0-P7, encoded in the "Pn" field.
- <Pm> Is the name of the second governing scalable predicate register P0-P7, encoded in the "Pm" field.
- <Zn> Is the name of the first source scalable vector register, encoded in the "Zn" field.
- <Zm> Is the name of the second source scalable vector register, encoded in the "Zm" field.

Operation

```

1 CheckStreamingSVEAndZAAEnabled();
2 integer dim = VL DIV esize;
3 bits(PL) mask1 = P[a];
4 bits(PL) mask2 = P[b];
5 bits(VL) operand1 = Z[n];
6 bits(VL) operand2 = Z[m];
7 bits(dim*dim*esize) operand3 = Ztile[da, esize];
8 bits(dim*dim*esize) result;
9 integer prod;
10
11 for row = 0 to dim-1
12     for col = 0 to dim-1
13         bits(esize) sum = Elem[operand3, row*dim+col, esize];
14         for k = 0 to 3
15             if ElemP[mask1, 4*row + k, esize DIV 4] == '1' &&
16                 ElemP[mask2, 4*col + k, esize DIV 4] == '1' then
17                 prod = (Int(Elem[operand1, 4*row + k, esize DIV 4], op1_unsigned) *
18                     Int(Elem[operand2, 4*col + k, esize DIV 4], op2_unsigned));
19                 if sub_op then prod = -prod;
20                 sum = sum + prod;
21
22         Elem[result, row*dim+col, esize] = sum;
23
24 Ztile[da, esize] = result;

```

Operational information

If FEAT_SVE2 is implemented or FEAT_SME is implemented, then when PSTATE.DIT is 1:

- The execution time of this instruction is independent of:
 - The values of the data supplied in any of its operand registers when its governing predicate registers contain the same value for each execution.
 - The values of the NZCV flags.
- The response of this instruction to asynchronous exceptions does not vary based on:
 - The values of the data supplied in any of its operand registers when its governing predicate registers contain the same value for each execution.
 - The values of the NZCV flags.

D1.1.35 USMOPS

Unsigned by signed integer sum of outer products and subtract

The 8-bit integer variant works with a 32-bit element ZA tile.

The 16-bit integer variant works with a 64-bit element ZA tile.

The unsigned by signed integer sum of outer products and subtract instructions multiply the sub-matrix in the first source vector by the sub-matrix in the second source vector. In case of the 8-bit integer variant, the first source holds $SVL_S \times 4$ sub-matrix of unsigned 8-bit integer values, and the second source holds $4 \times SVL_S$ sub-matrix of signed 8-bit integer values. In case of the 16-bit integer variant, the first source holds $SVL_D \times 4$ sub-matrix of unsigned 16-bit integer values, and the second source holds $4 \times SVL_D$ sub-matrix of signed 16-bit integer values.

Each source vector is independently predicated by a corresponding governing predicate. When an 8-bit source element in case of 8-bit integer variant or a 16-bit source element in case of 16-bit integer variant is Inactive, it is treated as having the value 0.

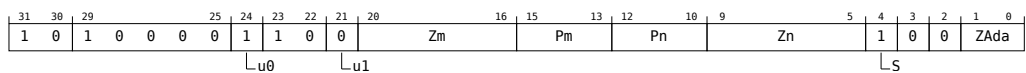
The resulting $SVL_S \times SVL_S$ widened 32-bit integer or $SVL_D \times SVL_D$ widened 64-bit integer sum of outer products is then destructively subtracted from the 32-bit integer or 64-bit integer destination tile, respectively for 8-bit integer and 16-bit integer instruction variants. This is equivalent to performing a 4-way dot product and subtract from each of the destination tile elements.

In case of the 8-bit integer variant, each 32-bit container of first source vector holds 4 consecutive column elements of each row of a $SVL_S \times 4$ sub-matrix, and each 32-bit container of second source vector holds 4 consecutive row elements of each column of a $4 \times SVL_S$ sub-matrix. In case of the 16-bit integer variant, each 64-bit container of first source vector holds 4 consecutive column elements of each row of a $SVL_D \times 4$ sub-matrix, and each 64-bit container of second source vector holds 4 consecutive row elements of each column of a $4 \times SVL_D$ sub-matrix.

ID_AA64SMFR0_EL1.I16I64 indicates whether the 16-bit integer variant is implemented.

It has encodings from 2 classes: [32-bit](#) and [64-bit](#)

32-bit (FEAT_SME)



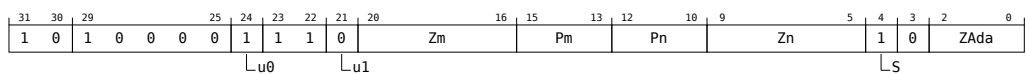
USMOPS [<ZAda>.S](#), [<Pn>/M](#), [<Pm>/M](#), [<Zn>.B](#), [<Zm>.B](#)

```

1 if !HaveSME() then UNDEFINED;
2 integer esize = 32;
3 integer a = UInt(Pn);
4 integer b = UInt(Pm);
5 integer n = UInt(Zn);
6 integer m = UInt(Zm);
7 integer da = UInt(ZAda);
8 boolean sub_op = TRUE;
9 boolean op1_unsigned = TRUE;
10 boolean op2_unsigned = FALSE;

```

64-bit (FEAT_SME_I16I64)



USMOPS [<ZAda>.D](#), [<Pn>/M](#), [<Pm>/M](#), [<Zn>.H](#), [<Zm>.H](#)


```

1 if !HaveSMEI16I64() then UNDEFINED;
2 integer esize = 64;
3 integer a = UInt(Pn);
4 integer b = UInt(Pm);
5 integer n = UInt(Zn);
6 integer m = UInt(Zm);
7 integer da = UInt(ZAda);
8 boolean sub_op = TRUE;
9 boolean op1_unsigned = TRUE;
10 boolean op2_unsigned = FALSE;

```

Assembler Symbols

- <ZAda> For the 32-bit variant: is the name of the ZA tile ZA0-ZA3, encoded in the "ZAda" field.
For the 64-bit variant: is the name of the ZA tile ZA0-ZA7, encoded in the "ZAda" field.
- <Pn> Is the name of the first governing scalable predicate register P0-P7, encoded in the "Pn" field.
- <Pm> Is the name of the second governing scalable predicate register P0-P7, encoded in the "Pm" field.
- <Zn> Is the name of the first source scalable vector register, encoded in the "Zn" field.
- <Zm> Is the name of the second source scalable vector register, encoded in the "Zm" field.

Operation

```

1 CheckStreamingSVEAndZAAEnabled();
2 integer dim = VL DIV esize;
3 bits(PL) mask1 = P[a];
4 bits(PL) mask2 = P[b];
5 bits(VL) operand1 = Z[n];
6 bits(VL) operand2 = Z[m];
7 bits(dim*dim*esize) operand3 = Ztile[da, esize];
8 bits(dim*dim*esize) result;
9 integer prod;
10
11 for row = 0 to dim-1
12     for col = 0 to dim-1
13         bits(esize) sum = Elem[operand3, row*dim+col, esize];
14         for k = 0 to 3
15             if ElemP[mask1, 4*row + k, esize DIV 4] == '1' &&
16                 ElemP[mask2, 4*col + k, esize DIV 4] == '1' then
17                 prod = (Int(Elem[operand1, 4*row + k, esize DIV 4], op1_unsigned) *
18                     Int(Elem[operand2, 4*col + k, esize DIV 4], op2_unsigned));
19                 if sub_op then prod = -prod;
20                 sum = sum + prod;
21
22         Elem[result, row*dim+col, esize] = sum;
23
24 Ztile[da, esize] = result;

```

Operational information

If FEAT_SVE2 is implemented or FEAT_SME is implemented, then when PSTATE.DIT is 1:

- The execution time of this instruction is independent of:
 - The values of the data supplied in any of its operand registers when its governing predicate registers contain the same value for each execution.
 - The values of the NZCV flags.
- The response of this instruction to asynchronous exceptions does not vary based on:
 - The values of the data supplied in any of its operand registers when its governing predicate registers contain the same value for each execution.
 - The values of the NZCV flags.

D1.1.36 ZERO

Zero a list of 64-bit element ZA tiles

Zeroes all bytes within each of the up to eight listed 64-bit element tiles named ZA0.D to ZA7.D, leaving the other 64-bit element tiles unmodified.

This instruction does not require the PE to be in Streaming SVE mode, and it is expected that this instruction will not experience a significant slowdown due to contention with other PEs that are executing in Streaming SVE mode.

For programmer convenience an assembler must also accept the names of 32-bit, 16-bit and 8-bit element tiles which are converted into the corresponding set of 64-bit element tiles.

In accordance with the architecturally defined mapping between different element size tiles:

* Zeroing the 8-bit element tile name ZA0.B, or the entire array name ZA, is equivalent to zeroing all eight 64-bit element tiles named ZA0.D to ZA7.D.

* Zeroing the 16-bit element tile name ZA0.H is equivalent to zeroing 64-bit element tiles named ZA0.D, ZA2.D, ZA4.D and ZA6.D.

* Zeroing the 16-bit element tile name ZA1.H is equivalent to zeroing 64-bit element tiles named ZA1.D, ZA3.D, ZA5.D and ZA7.D.

* Zeroing the 32-bit element tile name ZA0.S is equivalent to zeroing 64-bit element tiles named ZA0.D and ZA4.D.

* Zeroing the 32-bit element tile name ZA1.S is equivalent to zeroing 64-bit element tiles named ZA1.D and ZA5.D.

* Zeroing the 32-bit element tile name ZA2.S is equivalent to zeroing 64-bit element tiles named ZA2.D and ZA6.D.

* Zeroing the 32-bit element tile name ZA3.S is equivalent to zeroing 64-bit element tiles named ZA3.D and ZA7.D.

The preferred disassembly of this instruction uses the shortest list of tile names that represent the encoded immediate mask.

For example:

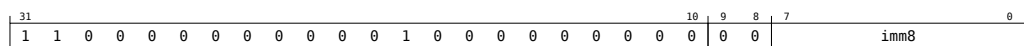
* An immediate which encodes 64-bit element tiles ZA0.D, ZA1.D, ZA4.D and ZA5.D is disassembled as {ZA0.S, ZA1.S}.

* An immediate which encodes 64-bit element tiles ZA0.D, ZA2.D, ZA4.D and ZA6.D is disassembled as {ZA0.H}.

* An all-ones immediate is disassembled as {ZA}.

* An all-zeros immediate is disassembled as an empty list $\{ \}$.

SME
(FEAT_SME)



ZERO { <mask> }

```
1  if !HaveSME() then UNDEFINED;
2  bits(8) mask = imm8;
3  integer esize = 64;
```

Assembler Symbols

<mask> Is a list of up to eight 64-bit element tile names separated by commas, encoded in the "imm8" field.

Operation

```
1 CheckSMEAndZEnabled();
2 integer dim = SVL DIV esize;
3 bits(dim*dim*esize) result = Zeros();
4
5 if HaveTME() && TSTATE.depth > 0 then
6     FailTransaction(TMFailure_ERR, FALSE);
7
8 for i = 0 to 7
9     if mask<i> == '1' then ZAtile[i, esize] = result;
```

D1.2 Base A64 instructions

The following Base A64 instructions are added or modified by the SME architecture.

D1.2.1 MSR (immediate)

Move immediate value to Special Register moves an immediate value to selected bits of the PSTATE. For more information, see *Process state, PSTATE*.

The bits that can be written by this instruction are:

- PSTATE.D, PSTATE.A, PSTATE.I, PSTATE.F, and PSTATE.SP.
- If *FEAT_SSBS* is implemented, PSTATE.SSBS.
- If *FEAT_PAN* is implemented, PSTATE.PAN.
- If *FEAT_UAO* is implemented, PSTATE.UAO.
- If *FEAT_DIT* is implemented, PSTATE.DIT.
- If *FEAT_MTE* is implemented, PSTATE.TCO.
- If *FEAT_NMI* is implemented, PSTATE.ALLINT.
- If *FEAT_SME* is implemented, PSTATE.SM and PSTATE.ZA.

This instruction is used by the aliases [SMSTART](#), and [SMSTOP](#).

31	22	21	20	19	18	16	15	12	11	8	7	5	4	0
1	1	0	1	0	1	0	1	0	0	0	0	0	0	1
					op1						CRm	op2		1 1 1 1 1

MSR <pstatefield>, #<imm>

```

1  if op1 == '000' && op2 == '000' then SEE "CFINV";
2  if op1 == '000' && op2 == '001' then SEE "XAFLAG";
3  if op1 == '000' && op2 == '010' then SEE "AXFLAG";
4
5  AArch64.CheckSystemAccess('00', op1, '0100', CRm, op2, '11111', '0');
6  bits(2) min_EL;
7  boolean need_secure = FALSE;
8
9  case op1 of
10     when '00x'
11         min_EL = EL1;
12     when '010'
13         min_EL = EL1;
14     when '011'
15         min_EL = EL0;
16     when '100'
17         min_EL = EL2;
18     when '101'
19         if !HaveVirtHostExt() then
20             UNDEFINED;
21         min_EL = EL2;
22     when '110'
23         min_EL = EL3;
24     when '111'
25         min_EL = EL1;
26         need_secure = TRUE;
27
28  if UInt(PSTATE.EL) < UInt(min_EL) || (need_secure && !IsSecure()) then
29     UNDEFINED;
30
31  bits(4) operand = CRm;
32  PSTATEField field;
33  case op1:op2 of
34     when '000 011'
35         if !HaveUAOExt() then UNDEFINED;
36         field = PSTATEField_UAO;
```

```

37     when '000 100'
38         if !HavePANExt() then UNDEFINED;
39         field = PSTATEfield_PAN;
40     when '000 101' field = PSTATEfield_SP;
41     when '001 000'
42         if !HaveFeatNMI() then UNDEFINED;
43         if CRm<3:1> != '000' then UNDEFINED;
44         field = PSTATEfield_ALLINT;
45     when '011 010'
46         if !HaveDITExt() then UNDEFINED;
47         field = PSTATEfield_DIT;
48     when '011 011'
49         case CRm of
50             when '001x'
51                 if !HaveSME() then UNDEFINED;
52                 field = PSTATEfield_SVCRSM;
53             when '010x'
54                 if !HaveSME() then UNDEFINED;
55                 field = PSTATEfield_SVCRZA;
56             when '011x'
57                 if !HaveSME() then UNDEFINED;
58                 field = PSTATEfield_SVCRSMZA;
59             otherwise
60                 UNDEFINED;
61     when '011 100'
62         if !HaveMTEExt() then UNDEFINED;
63         field = PSTATEfield_TCO;
64     when '011 110' field = PSTATEfield_DAIFSet;
65     when '011 111' field = PSTATEfield_DAIFClr;
66     when '011 001'
67         if !HaveSSBSExt() then UNDEFINED;
68         field = PSTATEfield_SSBS;
69     otherwise UNDEFINED;
70
71 // Check that an AArch64 MSR/MRS access to the DAIF flags is permitted
72 if PSTATE.EL == EL0 && field IN {PSTATEfield_DAIFSet, PSTATEfield_DAIFClr} then
73     if !ELUsingAArch32(EL1) && ((EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') || SCTLRL_EL1.UMA == '0') then
74         if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
75             AArch64.SystemAccessTrap(EL2, 0x18);
76         else
77             AArch64.SystemAccessTrap(EL1, 0x18);

```

Assembler Symbols

<pstatefield> Is a PSTATE field name. For the MSR instruction, this is encoded in "op1:op2:CRm":

op1	op2	CRm	<pstatefield>	Architectural Feature
000	00x	xxxx	SEE PSTATE	—
000	010	xxxx	SEE PSTATE	—
000	011	xxxx	UAO	FEAT_UAO
000	100	xxxx	PAN	FEAT_PAN
000	101	xxxx	SPSel	—
000	11x	xxxx	RESERVED	—
001	000	000x	ALLINT	FEAT_NMI
001	000	001x	RESERVED	—
001	000	01xx	RESERVED	—
001	000	1xxx	RESERVED	—
001	001	xxxx	RESERVED	—
001	01x	xxxx	RESERVED	—
001	1xx	xxxx	RESERVED	—
010	xxx	xxxx	RESERVED	—
011	000	xxxx	RESERVED	—
011	001	xxxx	SSBS	FEAT_SSBS
011	010	xxxx	DIT	FEAT_DIT
011	011	000x	RESERVED	—
011	011	001x	SVCRSM	FEAT_SME
011	011	010x	SVCRZA	FEAT_SME
011	011	011x	SVCRSMZA	FEAT_SME
011	011	1xxx	RESERVED	—
011	100	xxxx	TCO	FEAT_MTE
011	101	xxxx	RESERVED	—
011	110	xxxx	DAIFSet	—
011	111	xxxx	DAIFClr	—
1xx	xxx	xxxx	RESERVED	—

<imm> Is a 4-bit unsigned immediate, in the range 0 to 15, encoded in the "CRm" field. Restricted to the range 0 to 1, encoded in "CRm<0>", when <pstatefield> is ALLINT, SVCRSM, SVCRSMZA, or SVCRZA.

Alias Conditions

Alias	Is preferred when
SMSTART	op1 == '011' && CRm == '0xx1' && op2 == '011'
SMSTOP	op1 == '011' && CRm == '0xx0' && op2 == '011'

Operation

```

1 case field of
2   when PSTATEfield_SSBS
3     PSTATE.SSBS = operand<0>;
4   when PSTATEfield_SP
5     PSTATE.SP = operand<0>;
6   when PSTATEfield_DAIFSet
7     PSTATE.D = PSTATE.D OR operand<3>;
8     PSTATE.A = PSTATE.A OR operand<2>;
9     PSTATE.I = PSTATE.I OR operand<1>;
10    PSTATE.F = PSTATE.F OR operand<0>;
11  when PSTATEfield_DAIFClr
12    PSTATE.D = PSTATE.D AND NOT(operand<3>);
13    PSTATE.A = PSTATE.A AND NOT(operand<2>);
14    PSTATE.I = PSTATE.I AND NOT(operand<1>);
15    PSTATE.F = PSTATE.F AND NOT(operand<0>);
16  when PSTATEfield_PAN
17    PSTATE.PAN = operand<0>;
18  when PSTATEfield_UAO
19    PSTATE.UAO = operand<0>;

```

```
20  when PSTATEfield_DIT
21      PSTATE.DIT = operand<0>;
22  when PSTATEfield_TCO
23      PSTATE.TCO = operand<0>;
24  when PSTATEfield_ALLINT
25      if (PSTATE.EL == EL1 && IsHCRXEL2Enabled() &&
26          HCRX_EL2.TALLINT == '1' && operand<0> == '1') then
27          AArch64.SystemAccessTrap(EL2, 0x18);
28      PSTATE.ALLINT = operand<0>;
29  when PSTATEfield_SVCRSM
30      CheckSMEAccess();
31      SetPSTATE_SM(operand<0>);
32  when PSTATEfield_SVCRZA
33      CheckSMEAccess();
34      SetPSTATE_ZA(operand<0>);
35  when PSTATEfield_SVCRSMZA
36      CheckSMEAccess();
37      SetPSTATE_SM(operand<0>);
38      SetPSTATE_ZA(operand<0>);
```

D1.2.2 SMSTART

Enables access to Streaming SVE mode and SME architectural state.

SMSTART enters Streaming SVE mode, and enables the SME ZA array storage.

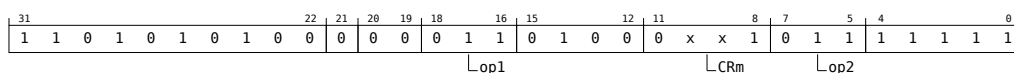
SMSTART SM enters Streaming SVE mode, but does not enable the SME ZA array storage.

SMSTART ZA enables the SME ZA array storage, but does not cause an entry to Streaming SVE mode.

This is an alias of [MSR \(immediate\)](#). This means:

- The encodings in this description are named to match the encodings of [MSR \(immediate\)](#).
- The description of [MSR \(immediate\)](#) gives the operational pseudocode for this instruction.

System (FEAT_SME)



SMSTART {<option>}

is equivalent to

[MSR<pstatefield>](#), #1

and is always the preferred disassembly.

Assembler Symbols

<option> Is an optional mode, encoded in "CRm<2:1>":

CRm<2:1>	<option>
00	RESERVED
01	SM
10	ZA
11	[no specifier]

<pstatefield> Is a PSTATE field name. For the MSR instruction, this is encoded in "op1:op2:CRm":

op1	op2	CRm	<pstatefield>	Architectural Feature
000	00x	xxxx	SEE PSTATE	–
000	010	xxxx	SEE PSTATE	–
000	011	xxxx	UAO	FEAT_UAO
000	100	xxxx	PAN	FEAT_PAN
000	101	xxxx	SPSel	–
000	11x	xxxx	RESERVED	–
001	000	000x	ALLINT	FEAT_NMI
001	000	001x	RESERVED	–
001	000	01xx	RESERVED	–
001	000	1xxx	RESERVED	–
001	001	xxxx	RESERVED	–
001	01x	xxxx	RESERVED	–
001	1xx	xxxx	RESERVED	–
010	xxx	xxxx	RESERVED	–
011	000	xxxx	RESERVED	–
011	001	xxxx	SSBS	FEAT_SSBS
011	010	xxxx	DIT	FEAT_DIT
011	011	000x	RESERVED	–
011	011	001x	SVCRSM	FEAT_SME
011	011	010x	SVCRZA	FEAT_SME
011	011	011x	SVCRSMZA	FEAT_SME
011	011	1xxx	RESERVED	–
011	100	xxxx	TCO	FEAT_MTE
011	101	xxxx	RESERVED	–
011	110	xxxx	DAIFSet	–
011	111	xxxx	DAIFClr	–
1xx	xxx	xxxx	RESERVED	–

Operation

The description of [MSR \(immediate\)](#) gives the operational pseudocode for this instruction.

D1.2.3 SMSTOP

Disables access to Streaming SVE mode and SME architectural state.

SMSTOP exits Streaming SVE mode, and disables the SME ZA array storage.

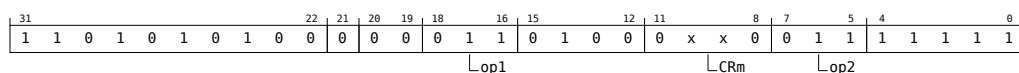
SMSTOP SM exits Streaming SVE mode, but does not disable the SME ZA array storage.

SMSTOP ZA disables the SME ZA array storage, but does not cause an exit from Streaming SVE mode.

This is an alias of [MSR \(immediate\)](#). This means:

- The encodings in this description are named to match the encodings of [MSR \(immediate\)](#).
- The description of [MSR \(immediate\)](#) gives the operational pseudocode for this instruction.

System (FEAT_SME)



SMSTOP {<option>}

is equivalent to

[MSR<pstatefield>](#), #0

and is always the preferred disassembly.

Assembler Symbols

<option> Is an optional mode, encoded in "CRm<2:1>":

CRm<2:1>	<option>
00	RESERVED
01	SM
10	ZA
11	[no specifier]

<pstatefield> Is a PSTATE field name. For the MSR instruction, this is encoded in "op1:op2:CRm":

op1	op2	CRm	<pstatefield>	Architectural Feature
000	00x	xxxx	SEE PSTATE	–
000	010	xxxx	SEE PSTATE	–
000	011	xxxx	UAO	FEAT_UAO
000	100	xxxx	PAN	FEAT_PAN
000	101	xxxx	SPSel	–
000	11x	xxxx	RESERVED	–
001	000	000x	ALLINT	FEAT_NMI
001	000	001x	RESERVED	–
001	000	01xx	RESERVED	–
001	000	1xxx	RESERVED	–
001	001	xxxx	RESERVED	–
001	01x	xxxx	RESERVED	–
001	1xx	xxxx	RESERVED	–
010	xxx	xxxx	RESERVED	–
011	000	xxxx	RESERVED	–
011	001	xxxx	SSBS	FEAT_SSBS
011	010	xxxx	DIT	FEAT_DIT
011	011	000x	RESERVED	–
011	011	001x	SVCRSM	FEAT_SME
011	011	010x	SVCRZA	FEAT_SME
011	011	011x	SVCRSMZA	FEAT_SME
011	011	1xxx	RESERVED	–
011	100	xxxx	TCO	FEAT_MTE
011	101	xxxx	RESERVED	–
011	110	xxxx	DAIFSet	–
011	111	xxxx	DAIFClr	–
1xx	xxx	xxxx	RESERVED	–

Operation

The description of [MSR \(immediate\)](#) gives the operational pseudocode for this instruction.

D1.3 SVE2 instructions

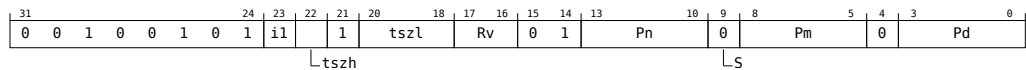
The following SVE2 instructions are added by the SME architecture, but are also usable when the PE is not in *Streaming SVE mode*, if FEAT_SVE2 is implemented.

D1.3.1 PSEL

Predicate select between predicate register or all-false

If the indexed element of the second source predicate is true, place the contents of the first source predicate register into the destination predicate register, otherwise set the destination predicate to all-false. The indexed element is determined by the sum of a general-purpose index register and an immediate, modulo the number of elements. Does not set the condition flags.

SVE2
(FEAT_SME)



PSEL <Pd>, <Pn>, <Pm>.<T>[<Wv>, <imm>]

```

1  if !HaveSME() then UNDEFINED;
2  bits(5) imm5 = il:tszh:tszl;
3  integer esize;
4  integer imm;
5  case tszh:tszl of
6    when '0000' UNDEFINED;
7    when '1000' esize = 64;  imm = UInt(imm5<4>);
8    when 'x100' esize = 32;  imm = UInt(imm5<4:3>);
9    when 'xx10' esize = 16;  imm = UInt(imm5<4:2>);
10   when 'xxx1' esize = 8;   imm = UInt(imm5<4:1>);
11  integer n = UInt(Pn);
12  integer m = UInt(Pm);
13  integer d = UInt(Pd);
14  integer v = UInt('011':Rv);

```

Assembler Symbols

- <Pd> Is the name of the destination scalable predicate register, encoded in the "Pd" field.
- <Pn> Is the name of the first source scalable predicate register, encoded in the "Pn" field.
- <Pm> Is the name of the second source scalable predicate register, encoded in the "Pm" field.
- <T> Is the size specifier, encoded in "tszh:tszl":

tszh	tszl	<T>
0	000	RESERVED
x	xx1	B
x	x10	H
x	100	S
1	000	D

- <Wv> Is the 32-bit name of the vector select register W12-W15, encoded in the "Rv" field.
- <imm> Is the element index, in the range 0 to one less than the number of vector elements in a 128-bit vector register, encoded in "i1:tszh:tszl".

Operation

```

1  CheckSVEEnabled();
2  integer elements = VL DIV esize;

```

```

3  bits(PL) operand1 = P[n];
4  bits(PL) operand2 = P[m];
5  bits(32) idx = X[v];
6  integer element = (UInt(idx) + imm) MOD elements;
7  bits(PL) result;
8
9  if ElemP[operand2, element, esize] == '1' then
10     result = operand1;
11 else
12     result = Zeros();
13
14 P[d] = result;

```

Operational information

If FEAT_SVE2 is implemented or FEAT_SME is implemented, then when PSTATE.DIT is 1:

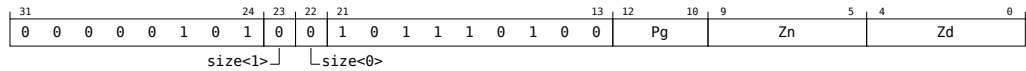
- The execution time of this instruction is independent of:
 - The values of the data supplied in any of its registers.
 - The values of the NZCV flags.
- The response of this instruction to asynchronous exceptions does not vary based on:
 - The values of the data supplied in any of its registers.
 - The values of the NZCV flags.

D1.3.2 REVD

Reverse 64-bit doublewords in elements (predicated)

Reverse the order of 64-bit doublewords within each active element of the source vector, and place the results in the corresponding elements of the destination vector. Inactive elements in the destination vector register remain unmodified.

SVE2 (FEAT_SME)



REVD <Zd>.Q, <Pg>/M, <Zn>.Q

```

1  if !HaveSME() then UNDEFINED;
2  integer esize = 128;
3  integer g = UInt(Pg);
4  integer n = UInt(Zn);
5  integer d = UInt(Zd);
6  integer swsize = 64;

```

Assembler Symbols

- <Zd> Is the name of the destination scalable vector register, encoded in the "Zd" field.
- <Pg> Is the name of the governing scalable predicate register P0-P7, encoded in the "Pg" field.
- <Zn> Is the name of the source scalable vector register, encoded in the "Zn" field.

Operation

```

1  CheckSVEEnabled();
2  integer elements = VL DIV esize;
3  bits(PL) mask = P[g];
4  bits(VL) operand = if AnyActiveElement(mask, esize) then Z[n] else Zeros();
5  bits(VL) result = Z[d];
6
7  for e = 0 to elements-1
8      if ElemP[mask, e, esize] == '1' then
9          bits(esize) element = Elem[operand, e, esize];
10         Elem[result, e, esize] = Reverse(element, swsize);
11
12  Z[d] = result;

```

Operational information

If FEAT_SVE2 is implemented or FEAT_SME is implemented, then when PSTATE.DIT is 1:

- The execution time of this instruction is independent of:
 - The values of the data supplied in any of its operand registers when its governing predicate register contains the same value for each execution.
 - The values of the NZCV flags.
- The response of this instruction to asynchronous exceptions does not vary based on:
 - The values of the data supplied in any of its operand registers when its governing predicate register contains the same value for each execution.
 - The values of the NZCV flags.

This instruction might be immediately preceded in program order by a MOVPRFX instruction. The MOVPRFX instruction must conform to all of the following requirements, otherwise the behavior of the MOVPRFX and this instruction is UNPREDICTABLE:

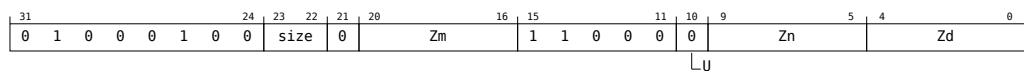
- The `MOVPRFX` instruction must be unpredicated.
- The `MOVPRFX` instruction must specify the same destination register as this instruction.
- The destination register must not refer to architectural register state referenced by any other source operand register of this instruction.

D1.3.3 SCLAMP

Signed clamp to minimum/maximum vector

Clamp each signed element in the destination vector to between the signed minimum value in the corresponding element of the first source vector and the signed maximum value in the corresponding element of the second source vector and destructively write the results in the corresponding elements of the destination vector. This instruction is unpredicated.

SVE2
(FEAT_SME)



SCLAMP <Zd>.<T>, <Zn>.<T>, <Zm>.<T>

```
1  if !HaveSME() then UNDEFINED;
2  integer esize = 8 << UInt(size);
3  integer n = UInt(Zn);
4  integer m = UInt(Zm);
5  integer d = UInt(Zd);
```

Assembler Symbols

<Zd> Is the name of the destination scalable vector register, encoded in the "Zd" field.

<T> Is the size specifier, encoded in "size":

size	$\langle T \rangle$
00	B
01	H
10	S
11	D

<Zn> Is the name of the first source scalable vector register, encoded in the "Zn" field.

<Zm> Is the name of the second source scalable vector register, encoded in the "Zm" field.

Operation

```

1  CheckSVEEnabled();
2  integer elements = VL DIV esize;
3  bits(VL) operand1 = Z[n];
4  bits(VL) operand2 = Z[m];
5  bits(VL) operand3 = Z[d];
6  bits(VL) result;
7
8  for e = 0 to elements-1
9      integer element1 = SInt(Elem[operand1, e, esize]);
10     integer element2 = SInt(Elem[operand2, e, esize]);
11     integer element3 = SInt(Elem[operand3, e, esize]);
12     integer res = Min(Max(element1, element3), element2);
13     Elem[result, e, esize] = res<esize-1:0>;
14
15  Z[d] = result;

```

Operational information

If FEAT_SVE2 is implemented or FEAT_SME is implemented, then when PSTATE.DIT is 1:

- The execution time of this instruction is independent of:
 - The values of the data supplied in any of its registers.
 - The values of the NZCV flags.

- The response of this instruction to asynchronous exceptions does not vary based on:
 - The values of the data supplied in any of its registers.
 - The values of the NZCV flags.

This instruction might be immediately preceded in program order by a `MOVPRFX` instruction. The `MOVPRFX` instruction must conform to all of the following requirements, otherwise the behavior of the `MOVPRFX` and this instruction is UNPREDICTABLE:

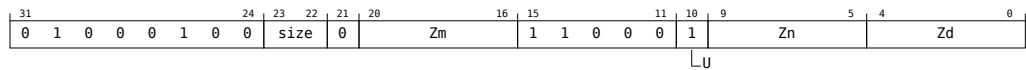
- The `MOVPRFX` instruction must be unpredicated.
- The `MOVPRFX` instruction must specify the same destination register as this instruction.
- The destination register must not refer to architectural register state referenced by any other source operand register of this instruction.

D1.3.4 UCLAMP

Unsigned clamp to minimum/maximum vector

Clamp each unsigned element in the destination vector to between the unsigned minimum value in the corresponding element of the first source vector and the unsigned maximum value in the corresponding element of the second source vector and destructively write the results in the corresponding elements of the destination vector. This instruction is unpredicated.

SVE2
(FEAT_SME)



UCLAMP <Zd>.<T>, <Zn>.<T>, <Zm>.<T>

```
1 if !HaveSME() then UNDEFINED;
2 integer esize = 8 << UInt(size);
3 integer n = UInt(Zn);
4 integer m = UInt(Zm);
5 integer d = UInt(Zd);
```

Assembler Symbols

<Zd> Is the name of the destination scalable vector register, encoded in the "Zd" field.

<T> Is the size specifier, encoded in "size":

size	<T>
00	B
01	H
10	S
11	D

<Zn> Is the name of the first source scalable vector register, encoded in the "Zn" field.

<Zm> Is the name of the second source scalable vector register, encoded in the "Zm" field.

Operation

```
1 CheckSVEEnabled();
2 integer elements = VL DIV esize;
3 bits(VL) operand1 = Z[n];
4 bits(VL) operand2 = Z[m];
5 bits(VL) operand3 = Z[d];
6 bits(VL) result;
7
8 for e = 0 to elements-1
9     integer element1 = UInt(Elem[operand1, e, esize]);
10    integer element2 = UInt(Elem[operand2, e, esize]);
11    integer element3 = UInt(Elem[operand3, e, esize]);
12    integer res = Min(Max(element1, element3), element2);
13    Elem[result, e, esize] = res<esize-1:0>;
14
15 Z[d] = result;
```

Operational information

If FEAT_SVE2 is implemented or FEAT_SME is implemented, then when PSTATE.DIT is 1:

- The execution time of this instruction is independent of:
 - The values of the data supplied in any of its registers.
 - The values of the NZCV flags.

- The response of this instruction to asynchronous exceptions does not vary based on:
 - The values of the data supplied in any of its registers.
 - The values of the NZCV flags.

This instruction might be immediately preceded in program order by a `MOVPRFX` instruction. The `MOVPRFX` instruction must conform to all of the following requirements, otherwise the behavior of the `MOVPRFX` and this instruction is UNPREDICTABLE:

- The `MOVPRFX` instruction must be unpredicated.
- The `MOVPRFX` instruction must specify the same destination register as this instruction.
- The destination register must not refer to architectural register state referenced by any other source operand register of this instruction.

Part E

Appendices

Chapter E1

Instructions affected by SME

The behavior of some non-SME instructions is affected when SME is implemented and the PE is in *Streaming SVE mode*.

This section lists affected instructions by the type of effect, with a description of the changes. It is a reference summary of information that can be viewed in more detail in *Arm® A64 Instruction Set Architecture Armv9, for Armv9-A architecture profile* [\[3\]](#).

E1.1 Illegal instructions in Streaming SVE mode

E1.1.1 Illegal Advanced SIMD instructions

The instruction encoding tables in this section are provided as an aid to understanding, and are consistent with the A64 ISA in Armv8.7-A and Armv9.2-A, but will require correction if subsequent versions of the A64 ISA add new instructions which overlap with these encodings.

AArch64 Advanced SIMD instructions with encodings that match the following patterns are *illegal* when the PE is in *Streaming SVE mode* and FEAT_SME_FA64 is not implemented or not enabled at the current Exception level:

A64 Encoding Pattern	Encoding Block
0x00 110x xxxx xxxx xxxx xxxx xxxx	Advanced SIMD structure load/store
0xx0 111x xxxx xxxx xxxx xxxx xxxx	Advanced SIMD vector operations
01x1 111x xxxx xxxx xxxx xxxx xxxx	Advanced SIMD single-element operations
1100 1110 xxxx xxxx xxxx xxxx xxxx	Advanced SIMD cryptography extensions

With the exception of certain vector to GPR integer move instructions, and some single-element floating-point instructions that match the following patterns and which execute normally when the PE is in *Streaming SVE mode*:

A64 Encoding Pattern	Instructions or Instruction Class
0x00 1110 0000 0001 0010 11xx xxxx xxxx	SMOV W Xd, Vn.B[0]
0x00 1110 0000 0010 0010 11xx xxxx xxxx	SMOV W Xd, Vn.H[0]
0100 1110 0000 0100 0010 11xx xxxx xxxx	SMOV Xd, Vn.S[0]
0000 1110 0000 0001 0011 11xx xxxx xxxx	UMOV Wd, Vn.B[0]
0000 1110 0000 0010 0011 11xx xxxx xxxx	UMOV Wd, Vn.H[0]
0000 1110 0000 0100 0011 11xx xxxx xxxx	UMOV Wd, Vn.S[0]
0100 1110 0000 1000 0011 11xx xxxx xxxx	UMOV Xd, Vn.D[0]
0101 1110 xx1x xxxx 11x1 11xx xxxx xxxx	FMULX/FRECPS/FRSQRTS (scalar)
0101 1110 x10x xxxx 00x1 11xx xxxx xxxx	FMULX/FRECPS/FRSQRTS (scalar, FP16)
01x1 1110 1x10 0001 11x1 10xx xxxx xxxx	FRECPE/FRSQRTS/FRECPX (scalar)
01x1 1110 1111 1001 11x1 10xx xxxx xxxx	FRECPE/FRSQRTS/FRECPX (scalar, FP16)

For the avoidance of doubt, A64 scalar floating-point instructions which match following encoding patterns remain *legal* when the PE is in *Streaming SVE mode*:

A64 Encoding Pattern	Instructions or Instruction Class
x001 111x xxxx xxxx xxxx xxxx xxxx	Scalar floating-point operations
xx10 110x xxxx xxxx xxxx xxxx xxxx	Load/store pair of FP registers
xx01 1100 xxxx xxxx xxxx xxxx xxxx	Load FP register (PC-relative literal)

A64 Encoding Pattern	Instructions or Instruction Class
xx11 1100 xx0x xxxx xxxx xxxx xxxx	Load/store FP register (unscaled imm)
xx11 1100 xx1x xxxx xxxx xxxx xxxx xx10	Load/store FP register (register offset)
xx11 1101 xxxx xxxx xxxx xxxx xxxx xxxx	Load/store FP register (scaled imm)

With the exception of the following floating-point operation which is *illegal* when the PE is in *Streaming SVE mode*:

A64 Encoding Pattern	Instructions or Instruction Class
0001 1110 0111 1110 0000 00xx xxxx xxxx	FJCVTZS

E1.1.1.1 Vector instructions

This section lists by name those A64 Advanced SIMD instruction pages in which all encoding variants are *illegal* when the PE is in *Streaming SVE mode* and FEAT_SME_FA64 is not implemented or not enabled at the current Exception level.

The Advanced SIMD instructions described in the following pages, and their aliases, are affected in this way:

- ABS: Absolute value (vector).
- ADD (vector): Add (vector).
- ADDHN, ADDHN2: Add returning High Narrow.
- ADDP (scalar): Add Pair of elements (scalar).
- ADDP (vector): Add Pairwise (vector).
- ADDV: Add across Vector.
- AESD: AES single round decryption.
- AESE: AES single round encryption.
- AESIMC: AES inverse mix columns.
- AESMC: AES mix columns.
- AND (vector): Bitwise AND (vector).
- BCAX: Bit Clear and XOR.
- BFCVTN, BFCVTN2: Floating-point convert from single-precision to BFloat16 format (vector).
- BFDOT (by element): BFloat16 floating-point dot product (vector, by element).
- BFDOT (vector): BFloat16 floating-point dot product (vector).
- BFMLALB, BFMLALT (by element): BFloat16 floating-point widening multiply-add long (by element).
- BFMLALB, BFMLALT (vector): BFloat16 floating-point widening multiply-add long (vector).
- BFMMMLA: BFloat16 floating-point matrix multiply-accumulate into 2x2 matrix.
- BIC (vector, immediate): Bitwise bit Clear (vector, immediate).
- BIC (vector, register): Bitwise bit Clear (vector, register).
- BIF: Bitwise Insert if False.
- BIT: Bitwise Insert if True.
- BSL: Bitwise Select.
- CLS (vector): Count Leading Sign bits (vector).
- CLZ (vector): Count Leading Zero bits (vector).
- CMEQ (register): Compare bitwise Equal (vector).
- CMEQ (zero): Compare bitwise Equal to zero (vector).
- CMGE (register): Compare signed Greater than or Equal (vector).
- CMGE (zero): Compare signed Greater than or Equal to zero (vector).
- CMGT (register): Compare signed Greater than (vector).
- CMGT (zero): Compare signed Greater than zero (vector).

- CMHI (register): Compare unsigned Higher (vector).
- CMHS (register): Compare unsigned Higher or Same (vector).
- CMLE (zero): Compare signed Less than or Equal to zero (vector).
- CMLT (zero): Compare signed Less than zero (vector).
- CMTST: Compare bitwise Test bits nonzero (vector).
- CNT: Population Count per byte.
- DUP (element): Duplicate vector element to vector or scalar.
- DUP (general): Duplicate general-purpose register to vector.
- EOR (vector): Bitwise Exclusive OR (vector).
- EOR3: Three-way Exclusive OR.
- EXT: Extract vector from pair of vectors.
- FABD: Floating-point Absolute Difference (vector).
- FABS (vector): Floating-point Absolute value (vector).
- FACGE: Floating-point Absolute Compare Greater than or Equal (vector).
- FACGT: Floating-point Absolute Compare Greater than (vector).
- FADD (vector): Floating-point Add (vector).
- FADDP (scalar): Floating-point Add Pair of elements (scalar).
- FADDP (vector): Floating-point Add Pairwise (vector).
- FCADD: Floating-point Complex Add.
- FCMEQ (register): Floating-point Compare Equal (vector).
- FCMEQ (zero): Floating-point Compare Equal to zero (vector).
- FCMGE (register): Floating-point Compare Greater than or Equal (vector).
- FCMGE (zero): Floating-point Compare Greater than or Equal to zero (vector).
- FCMGT (register): Floating-point Compare Greater than (vector).
- FCMGT (zero): Floating-point Compare Greater than zero (vector).
- FCMLA: Floating-point Complex Multiply Accumulate.
- FCMLA (by element): Floating-point Complex Multiply Accumulate (by element).
- FCMLE (zero): Floating-point Compare Less than or Equal to zero (vector).
- FCMLT (zero): Floating-point Compare Less than zero (vector).
- FCVTAS (vector): Floating-point Convert to Signed integer, rounding to nearest with ties to Away (vector).
- FCVTAU (vector): Floating-point Convert to Unsigned integer, rounding to nearest with ties to Away (vector).
- FCVTL, FCVTL2: Floating-point Convert to higher precision Long (vector).
- FCVTMS (vector): Floating-point Convert to Signed integer, rounding toward Minus infinity (vector).
- FCVTMU (vector): Floating-point Convert to Unsigned integer, rounding toward Minus infinity (vector).
- FCVTN, FCVTN2: Floating-point Convert to lower precision Narrow (vector).
- FCVTNS (vector): Floating-point Convert to Signed integer, rounding to nearest with ties to even (vector).
- FCVTNU (vector): Floating-point Convert to Unsigned integer, rounding to nearest with ties to even (vector).
- FCVTPS (vector): Floating-point Convert to Signed integer, rounding toward Plus infinity (vector).
- FCVTPU (vector): Floating-point Convert to Unsigned integer, rounding toward Plus infinity (vector).
- FCVTXN, FCVTXN2: Floating-point Convert to lower precision Narrow, rounding to odd (vector).
- FCVTZS (vector, fixed-point): Floating-point Convert to Signed fixed-point, rounding toward Zero (vector).
- FCVTZS (vector, integer): Floating-point Convert to Signed integer, rounding toward Zero (vector).
- FCVTZU (vector, fixed-point): Floating-point Convert to Unsigned fixed-point, rounding toward Zero (vector).
- FCVTZU (vector, integer): Floating-point Convert to Unsigned integer, rounding toward Zero (vector).
- FDIV (vector): Floating-point Divide (vector).
- FJCVTZS: Floating-point Javascript Convert to Signed fixed-point, rounding toward Zero.
- FMAX (vector): Floating-point Maximum (vector).
- FMAXNM (vector): Floating-point Maximum Number (vector).
- FMAXNMP (scalar): Floating-point Maximum Number of Pair of elements (scalar).
- FMAXNMP (vector): Floating-point Maximum Number Pairwise (vector).
- FMAXNMV: Floating-point Maximum Number across Vector.
- FMAXP (scalar): Floating-point Maximum of Pair of elements (scalar).
- FMAXP (vector): Floating-point Maximum Pairwise (vector).
- FMAXV: Floating-point Maximum across Vector.

- FMIN (vector): Floating-point minimum (vector).
- FMINNM (vector): Floating-point Minimum Number (vector).
- FMINNMP (scalar): Floating-point Minimum Number of Pair of elements (scalar).
- FMINNMP (vector): Floating-point Minimum Number Pairwise (vector).
- FMINNMV: Floating-point Minimum Number across Vector.
- FMINP (scalar): Floating-point Minimum of Pair of elements (scalar).
- FMINP (vector): Floating-point Minimum Pairwise (vector).
- FMINV: Floating-point Minimum across Vector.
- FMLA (by element): Floating-point fused Multiply-Add to accumulator (by element).
- FMLA (vector): Floating-point fused Multiply-Add to accumulator (vector).
- FMLAL, FMLAL2 (by element): Floating-point fused Multiply-Add Long to accumulator (by element).
- FMLAL, FMLAL2 (vector): Floating-point fused Multiply-Add Long to accumulator (vector).
- FMLS (by element): Floating-point fused Multiply-Subtract from accumulator (by element).
- FMLS (vector): Floating-point fused Multiply-Subtract from accumulator (vector).
- FMLSL, FMLSL2 (by element): Floating-point fused Multiply-Subtract Long from accumulator (by element).
- FMLSL, FMLSL2 (vector): Floating-point fused Multiply-Subtract Long from accumulator (vector).
- FMOV (vector, immediate): Floating-point move immediate (vector).
- FMUL (by element): Floating-point Multiply (by element).
- FMUL (vector): Floating-point Multiply (vector).
- FMULX (by element): Floating-point Multiply extended (by element).
- FNEG (vector): Floating-point Negate (vector).
- FRINT32X (vector): Floating-point Round to 32-bit Integer, using current rounding mode (vector).
- FRINT32Z (vector): Floating-point Round to 32-bit Integer toward Zero (vector).
- FRINT64X (vector): Floating-point Round to 64-bit Integer, using current rounding mode (vector).
- FRINT64Z (vector): Floating-point Round to 64-bit Integer toward Zero (vector).
- FRINTA (vector): Floating-point Round to Integral, to nearest with ties to Away (vector).
- FRINTI (vector): Floating-point Round to Integral, using current rounding mode (vector).
- FRINTM (vector): Floating-point Round to Integral, toward Minus infinity (vector).
- FRINTN (vector): Floating-point Round to Integral, to nearest with ties to even (vector).
- FRINTP (vector): Floating-point Round to Integral, toward Plus infinity (vector).
- FRINTX (vector): Floating-point Round to Integral exact, using current rounding mode (vector).
- FRINTZ (vector): Floating-point Round to Integral, toward Zero (vector).
- FSQRT (vector): Floating-point Square Root (vector).
- FSUB (vector): Floating-point Subtract (vector).
- INS (element): Insert vector element from another vector element.
- INS (general): Insert vector element from general-purpose register.
- LD1 (multiple structures): Load multiple single-element structures to one, two, three, or four registers.
- LD1 (single structure): Load one single-element structure to one lane of one register.
- LD1R: Load one single-element structure and Replicate to all lanes (of one register).
- LD2 (multiple structures): Load multiple 2-element structures to two registers.
- LD2 (single structure): Load single 2-element structure to one lane of two registers.
- LD2R: Load single 2-element structure and Replicate to all lanes of two registers.
- LD3 (multiple structures): Load multiple 3-element structures to three registers.
- LD3 (single structure): Load single 3-element structure to one lane of three registers).
- LD3R: Load single 3-element structure and Replicate to all lanes of three registers.
- LD4 (multiple structures): Load multiple 4-element structures to four registers.
- LD4 (single structure): Load single 4-element structure to one lane of four registers.
- LD4R: Load single 4-element structure and Replicate to all lanes of four registers.
- MLA (by element): Multiply-Add to accumulator (vector, by element).
- MLA (vector): Multiply-Add to accumulator (vector).
- MLS (by element): Multiply-Subtract from accumulator (vector, by element).
- MLS (vector): Multiply-Subtract from accumulator (vector).
- MOVI: Move Immediate (vector).
- MUL (by element): Multiply (vector, by element).
- MUL (vector): Multiply (vector).

- MVNI: Move inverted Immediate (vector).
- NEG (vector): Negate (vector).
- NOT: Bitwise NOT (vector).
- ORN (vector): Bitwise inclusive OR NOT (vector).
- ORR (vector, immediate): Bitwise inclusive OR (vector, immediate).
- ORR (vector, register): Bitwise inclusive OR (vector, register).
- PMUL: Polynomial Multiply.
- PMULL, PMULL2: Polynomial Multiply Long.
- RADDHN, RADDHN2: Rounding Add returning High Narrow.
- RAX1: Rotate and Exclusive OR.
- RBIT (vector): Reverse Bit order (vector).
- REV16 (vector): Reverse elements in 16-bit halfwords (vector).
- REV32 (vector): Reverse elements in 32-bit words (vector).
- REV64: Reverse elements in 64-bit doublewords (vector).
- RSHRN, RSHRN2: Rounding Shift Right Narrow (immediate).
- RSUBHN, RSUBHN2: Rounding Subtract returning High Narrow.
- SABA: Signed Absolute difference and Accumulate.
- SABAL, SABAL2: Signed Absolute difference and Accumulate Long.
- SABD: Signed Absolute Difference.
- SABDL, SABDL2: Signed Absolute Difference Long.
- SADALP: Signed Add and Accumulate Long Pairwise.
- SADDL, SADDL2: Signed Add Long (vector).
- SADDLP: Signed Add Long Pairwise.
- SADDLV: Signed Add Long across Vector.
- SADDW, SADDW2: Signed Add Wide.
- SCVTF (vector, fixed-point): Signed fixed-point Convert to Floating-point (vector).
- SCVTF (vector, integer): Signed integer Convert to Floating-point (vector).
- SDOT (by element): Dot Product signed arithmetic (vector, by element).
- SDOT (vector): Dot Product signed arithmetic (vector).
- SHA1C: SHA1 hash update (choose).
- SHA1H: SHA1 fixed rotate.
- SHA1M: SHA1 hash update (majority).
- SHA1P: SHA1 hash update (parity).
- SHA1SU0: SHA1 schedule update 0.
- SHA1SU1: SHA1 schedule update 1.
- SHA256H: SHA256 hash update (part 1).
- SHA256H2: SHA256 hash update (part 2).
- SHA256SU0: SHA256 schedule update 0.
- SHA256SU1: SHA256 schedule update 1.
- SHA512H: SHA512 Hash update part 1.
- SHA512H2: SHA512 Hash update part 2.
- SHA512SU0: SHA512 Schedule Update 0.
- SHA512SU1: SHA512 Schedule Update 1.
- SHADD: Signed Halving Add.
- SHL: Shift Left (immediate).
- SHLL, SHLL2: Shift Left Long (by element size).
- SHRN, SHRN2: Shift Right Narrow (immediate).
- SHSUB: Signed Halving Subtract.
- SLI: Shift Left and Insert (immediate).
- SM3PARTW1: SM3PARTW1.
- SM3PARTW2: SM3PARTW2.
- SM3SS1: SM3SS1.
- SM3TT1A: SM3TT1A.
- SM3TT1B: SM3TT1B.
- SM3TT2A: SM3TT2A.

- SM3TT2B: SM3TT2B.
- SM4E: SM4 Encode.
- SM4EKEY: SM4 Key.
- SMAX: Signed Maximum (vector).
- SMAXP: Signed Maximum Pairwise.
- SMAXV: Signed Maximum across Vector.
- SMIN: Signed Minimum (vector).
- SMINP: Signed Minimum Pairwise.
- SMINV: Signed Minimum across Vector.
- SMLAL, SMLAL2 (by element): Signed Multiply-Add Long (vector, by element).
- SMLAL, SMLAL2 (vector): Signed Multiply-Add Long (vector).
- SMLSL, SMLSL2 (by element): Signed Multiply-Subtract Long (vector, by element).
- SMLSL, SMLSL2 (vector): Signed Multiply-Subtract Long (vector).
- SMMLA (vector): Signed 8-bit integer matrix multiply-accumulate (vector).
- SMULL, SMULL2 (by element): Signed Multiply Long (vector, by element).
- SMULL, SMULL2 (vector): Signed Multiply Long (vector).
- SQABS: Signed saturating Absolute value.
- SQADD: Signed saturating Add.
- SQDMLAL, SQDMLAL2 (by element): Signed saturating Doubling Multiply-Add Long (by element).
- SQDMLAL, SQDMLAL2 (vector): Signed saturating Doubling Multiply-Add Long.
- SQDMLSL, SQDMLSL2 (by element): Signed saturating Doubling Multiply-Subtract Long (by element).
- SQDMLSL, SQDMLSL2 (vector): Signed saturating Doubling Multiply-Subtract Long.
- SQDMULH (by element): Signed saturating Doubling Multiply returning High half (by element).
- SQDMULH (vector): Signed saturating Doubling Multiply returning High half.
- SQDMULL, SQDMULL2 (by element): Signed saturating Doubling Multiply Long (by element).
- SQDMULL, SQDMULL2 (vector): Signed saturating Doubling Multiply Long.
- SQNEG: Signed saturating Negate.
- SQRDMLAH (by element): Signed Saturating Rounding Doubling Multiply Accumulate returning High Half (by element).
- SQRDMLAH (vector): Signed Saturating Rounding Doubling Multiply Accumulate returning High Half (vector).
- SQRDMLSH (by element): Signed Saturating Rounding Doubling Multiply Subtract returning High Half (by element).
- SQRDMLSH (vector): Signed Saturating Rounding Doubling Multiply Subtract returning High Half (vector).
- SQRDMULH (by element): Signed saturating Rounding Doubling Multiply returning High half (by element).
- SQRDMULH (vector): Signed saturating Rounding Doubling Multiply returning High half.
- QRSHL: Signed saturating Rounding Shift Left (register).
- QRSHRN, QRSHRN2: Signed saturating Rounded Shift Right Narrow (immediate).
- QRSHRUN, QRSHRUN2: Signed saturating Rounded Shift Right Unsigned Narrow (immediate).
- QQSHL (immediate): Signed saturating Shift Left (immediate).
- QQSHL (register): Signed saturating Shift Left (register).
- QQSHLU: Signed saturating Shift Left Unsigned (immediate).
- QQSHRN, QQSHRN2: Signed saturating Shift Right Narrow (immediate).
- QQSHRUN, QQSHRUN2: Signed saturating Shift Right Unsigned Narrow (immediate).
- QQSUB: Signed saturating Subtract.
- SQXTN, SQXTN2: Signed saturating extract Narrow.
- SQXTUN, SQXTUN2: Signed saturating extract Unsigned Narrow.
- SRHADD: Signed Rounding Halving Add.
- SRI: Shift Right and Insert (immediate).
- SRSHL: Signed Rounding Shift Left (register).
- SRSHR: Signed Rounding Shift Right (immediate).
- SRSRA: Signed Rounding Shift Right and Accumulate (immediate).
- SSHL: Signed Shift Left (register).
- SSHLL, SSHLL2: Signed Shift Left Long (immediate).

- SSHR: Signed Shift Right (immediate).
- SSRA: Signed Shift Right and Accumulate (immediate).
- SSUBL, SSUBL2: Signed Subtract Long.
- SSUBW, SSUBW2: Signed Subtract Wide.
- ST1 (multiple structures): Store multiple single-element structures from one, two, three, or four registers.
- ST1 (single structure): Store a single-element structure from one lane of one register.
- ST2 (multiple structures): Store multiple 2-element structures from two registers.
- ST2 (single structure): Store single 2-element structure from one lane of two registers.
- ST3 (multiple structures): Store multiple 3-element structures from three registers.
- ST3 (single structure): Store single 3-element structure from one lane of three registers.
- ST4 (multiple structures): Store multiple 4-element structures from four registers.
- ST4 (single structure): Store single 4-element structure from one lane of four registers.
- SUB (vector): Subtract (vector).
- SUBHN, SUBHN2: Subtract returning High Narrow.
- SUDOT (by element): Dot product with signed and unsigned integers (vector, by element).
- SUQADD: Signed saturating Accumulate of Unsigned value.
- TBL: Table vector Lookup.
- TBX: Table vector lookup extension.
- TRN1: Transpose vectors (primary).
- TRN2: Transpose vectors (secondary).
- UABA: Unsigned Absolute difference and Accumulate.
- UABAL, UABAL2: Unsigned Absolute difference and Accumulate Long.
- UABD: Unsigned Absolute Difference (vector).
- UABDL, UABDL2: Unsigned Absolute Difference Long.
- UADALP: Unsigned Add and Accumulate Long Pairwise.
- UADDL, UADDL2: Unsigned Add Long (vector).
- UADDLP: Unsigned Add Long Pairwise.
- UADDLV: Unsigned sum Long across Vector.
- UADDW, UADDW2: Unsigned Add Wide.
- UCVTF (vector, fixed-point): Unsigned fixed-point Convert to Floating-point (vector).
- UCVTF (vector, integer): Unsigned integer Convert to Floating-point (vector).
- UDOT (by element): Dot Product unsigned arithmetic (vector, by element).
- UDOT (vector): Dot Product unsigned arithmetic (vector).
- UHADD: Unsigned Halving Add.
- UHSUB: Unsigned Halving Subtract.
- UMAX: Unsigned Maximum (vector).
- UMAXP: Unsigned Maximum Pairwise.
- UMAXV: Unsigned Maximum across Vector.
- UMIN: Unsigned Minimum (vector).
- UMINP: Unsigned Minimum Pairwise.
- UMINV: Unsigned Minimum across Vector.
- UMLAL, UMLAL2 (by element): Unsigned Multiply-Add Long (vector, by element).
- UMLAL, UMLAL2 (vector): Unsigned Multiply-Add Long (vector).
- UMLSL, UMLSL2 (by element): Unsigned Multiply-Subtract Long (vector, by element).
- UMLSL, UMLSL2 (vector): Unsigned Multiply-Subtract Long (vector).
- UMMLA (vector): Unsigned 8-bit integer matrix multiply-accumulate (vector).
- UMULL, UMULL2 (by element): Unsigned Multiply Long (vector, by element).
- UMULL, UMULL2 (vector): Unsigned Multiply long (vector).
- UQADD: Unsigned saturating Add.
- UQRSHL: Unsigned saturating Rounding Shift Left (register).
- UQRSHRN, UQRSHRN2: Unsigned saturating Rounded Shift Right Narrow (immediate).
- UQSHL (immediate): Unsigned saturating Shift Left (immediate).
- UQSHL (register): Unsigned saturating Shift Left (register).
- UQSHRN, UQSHRN2: Unsigned saturating Shift Right Narrow (immediate).
- UQSUB: Unsigned saturating Subtract.

- UQXTN, UQXTN2: Unsigned saturating extract Narrow.
- URECPE: Unsigned Reciprocal Estimate.
- URHADD: Unsigned Rounding Halving Add.
- URSHL: Unsigned Rounding Shift Left (register).
- URSR: Unsigned Rounding Shift Right (immediate).
- URSQRTE: Unsigned Reciprocal Square Root Estimate.
- URSRA: Unsigned Rounding Shift Right and Accumulate (immediate).
- USDOT (by element): Dot Product with unsigned and signed integers (vector, by element).
- USDOT (vector): Dot Product with unsigned and signed integers (vector).
- USHL: Unsigned Shift Left (register).
- USHLL, USHLL2: Unsigned Shift Left Long (immediate).
- USHR: Unsigned Shift Right (immediate).
- USMMLA (vector): Unsigned and signed 8-bit integer matrix multiply-accumulate (vector).
- USQADD: Unsigned saturating Accumulate of Signed value.
- USRA: Unsigned Shift Right and Accumulate (immediate).
- USUBL, USUBL2: Unsigned Subtract Long.
- USUBW, USUBW2: Unsigned Subtract Wide.
- UZP1: Unzip vectors (primary).
- UZP2: Unzip vectors (secondary).
- XAR: Exclusive OR and Rotate.
- XTN, XTN2: Extract Narrow.
- ZIP1: Zip vectors (primary).
- ZIP2: Zip vectors (secondary).

If execution of an illegal Advanced SIMD instruction is attempted when the PE is in *Streaming SVE mode*, and the instructions are not configured to trap, this will cause an SME exception to be taken, as defined by rule [R_{DTCLZ}](#) in [C2.2.1 Exception priorities](#).

E1.1.1.2 Single-element instructions

This section lists by name those A64 Advanced SIMD instruction pages in which only the SIMD “Vector” encoding variants can be *illegal* when the PE is in *Streaming SVE mode*, but in which the single-element “Scalar” encoding variants are always *legal* in *Streaming SVE mode*.

The Vector encodings of Advanced SIMD instructions described in the following pages are affected in this way:

- FMULX: Floating-point Multiply extended.
- FRECPE: Floating-point Reciprocal Estimate.
- FRECPX: Floating-point Reciprocal Step.
- FRECPX: Floating-point Reciprocal Exponent.¹
- FRSQRTE: Floating-point Reciprocal Square Root Estimate.
- FRSQRTS: Floating-point Reciprocal Square Root Step.

E1.1.1.3 Element move to general register

The following Advanced SIMD instructions and their aliases can only be *illegal* when the PE is in *Streaming SVE mode* if their immediate vector element index is greater than zero. They are always *legal* in *Streaming SVE mode* when their element index is zero:

- SMOV: Signed Move vector element to general-purpose register.
- UMOV: Unsigned Move vector element to general-purpose register.

The *64-bit to top half of 128-bit* and *Top half of 128-bit to 64-bit* variants from the following instruction page are part of the scalar floating-point instruction set and therefore execute normally when the PE is in *Streaming SVE mode*:

¹FRECPX is an exception in that it only has a single-element form.

- FMOV (general): Floating-point Move to or from general-purpose register without conversion.

E1.1.2 Illegal SVE instructions

Allocated SVE and SVE2 instructions with encodings that match the following patterns are *illegal* when the PE is in *Streaming SVE mode* and FEAT_SME_FA64 is not implemented or not enabled at the current Exception level:

A64 Encoding Pattern	SVE Instructions or Instruction Class
0000 0100 xx1x xxxx 1010 xxxx xxxx xxxx	ADR
0000 0100 xx1x xxxx 1011 x0xx xxxx xxxx	FTSSEL, FEXPA
0000 0101 xx10 0001 100x xxxx xxxx xxxx	COMPACT
0010 0101 xx01 100x 1111 000x xxx0 xxxx	RDFFR, RDFFRS
0010 0101 xx10 1xxx 1001 xxxx xxxx xxxx	WRFFR, SETFFR
0100 0101 xx0x xxxx 1011 xxxx xxxx xxxx	BDEP, BEXT, BGRP
0100 0101 000x xxxx 0110 1xxx xxxx xxxx	PMULLB, PMULLT (128b result)
0110 0100 xx1x xxxx 1110 01xx xxxx xxxx	FMMLA, BFMMLA
0110 0101 xx0x xxxx 0000 11xx xxxx xxxx	FTSMUL
0110 0101 xx01 0xxx 100x xxxx xxxx xxxx	FTMAD
0110 0101 xx01 1xxx 001x xxxx xxxx xxxx	FADDA
0100 0101 xx0x xxxx 1001 10xx xxxx xxxx	SMMLA, UMMLA, USMMLA
0100 0101 xx1x xxxx 1xxx xxxx xxxx xxxx	SVE2 string/histo/crypto instructions
1000 010x x00x xxxx 10xx xxxx xxxx xxxx	SVE2 32-bit gather NT load (vector+scalar)
1000 010x x00x xxxx 111x xxxx xxxx xxxx	SVE 32-bit gather prefetch (vector+imm)
1000 0100 0x1x xxxx 0xxx xxxx xxxx xxxx	SVE 32-bit gather prefetch (scalar+vector)
1000 010x x01x xxxx 1xxx xxxx xxxx xxxx	SVE 32-bit gather load (vector+imm)
1000 0100 0x0x xxxx 0xxx xxxx xxxx xxxx	SVE 32-bit gather load byte (scalar+vector)
1000 0100 1xxx xxxx 0xxx xxxx xxxx xxxx	SVE 32-bit gather load half (scalar+vector)
1000 0101 0xxx xxxx 0xxx xxxx xxxx xxxx	SVE 32-bit gather load word (scalar+vector)
1010 010x xxxx xxxx 011x xxxx xxxx xxxx	SVE contiguous FF load (scalar+scalar)
1010 010x xx1x xxxx 101x xxxx xxxx xxxx	SVE contiguous NF load (scalar+imm)
1010 010x x10x xxxx 000x xxxx xxxx xxxx	SVE load & replicate 32 bytes (scalar+scalar)
1010 010x x100 xxxx 001x xxxx xxxx xxxx	SVE load & replicate 32 bytes (scalar+imm)
1100 010x xxxx xxxx xxxx xxxx xxxx xxxx	SVE 64-bit gather load/prefetch
1110 010x x00x xxxx 001x xxxx xxxx xxxx	SVE2 64-bit scatter NT store (vector+scalar)
1110 010x x10x xxxx 001x xxxx xxxx xxxx	SVE2 32-bit scatter NT store (vector+scalar)
1110 010x xxxx xxxx 1x0x xxxx xxxx xxxx	SVE scatter store (scalar+32-bit vector)
1110 010x xxxx xxxx 101x xxxx xxxx xxxx	SVE scatter store (misc)

The following SVE and SVE2 instructions and their aliases are affected:

- ADR: Compute vector address.
- AESD: AES single round decryption.
- AESE: AES single round encryption.
- AESIMC: AES inverse mix columns.
- AESMC: AES mix columns.
- BDEP: Scatter lower bits into positions selected by bitmask
- BEXT: Gather lower bits from positions selected by bitmask.
- BFMLLA: BFloat16 floating-point matrix multiply-accumulate.
- BGRP: Group bits to right or left as selected by bitmask.
- COMPACT: Shuffle active elements of vector to the right and fill with zero.
- FADDA: Floating-point add strictly-ordered reduction, accumulating in scalar.
- FEXPA: Floating-point exponential accelerator.
- FMMLA: Floating-point matrix multiply-accumulate.
- FTMAD: Floating-point trigonometric multiply-add coefficient.
- FTSMUL: Floating-point trigonometric starting value.
- FTSSSEL: Floating-point trigonometric select coefficient.
- HISTCNT: Count matching elements in vector.
- HISTSEG: Count matching elements in vector segments.
- LD1B (scalar plus vector): Gather load unsigned bytes to vector (vector index).
- LD1B (vector plus immediate): Gather load unsigned bytes to vector (immediate index).
- LD1D (scalar plus vector): Gather load doublewords to vector (vector index).
- LD1D (vector plus immediate): Gather load doublewords to vector (immediate index).
- LD1H (scalar plus vector): Gather load unsigned halfwords to vector (vector index).
- LD1H (vector plus immediate): Gather load unsigned halfwords to vector (immediate index).
- LD1ROB (scalar plus immediate): Contiguous load and replicate thirty-two bytes (immediate index).
- LD1ROB (scalar plus scalar): Contiguous load and replicate thirty-two bytes (scalar index).
- LD1ROD (scalar plus immediate): Contiguous load and replicate four doublewords (immediate index).
- LD1ROD (scalar plus scalar): Contiguous load and replicate four doublewords (scalar index).
- LD1ROH (scalar plus immediate): Contiguous load and replicate sixteen halfwords (immediate index).
- LD1ROH (scalar plus scalar): Contiguous load and replicate sixteen halfwords (scalar index).
- LD1ROW (scalar plus immediate): Contiguous load and replicate eight words (immediate index).
- LD1ROW (scalar plus scalar): Contiguous load and replicate eight words (scalar index).
- LD1SB (scalar plus vector): Gather load signed bytes to vector (vector index).
- LD1SB (vector plus immediate): Gather load signed bytes to vector (immediate index).
- LD1SH (scalar plus vector): Gather load signed halfwords to vector (vector index).
- LD1SH (vector plus immediate): Gather load signed halfwords to vector (immediate index).
- LD1SW (scalar plus vector): Gather load signed words to vector (vector index).
- LD1SW (vector plus immediate): Gather load signed words to vector (immediate index).
- LD1W (scalar plus vector): Gather load unsigned words to vector (vector index).
- LD1W (vector plus immediate): Gather load unsigned words to vector (immediate index).
- LDFF1B (scalar plus scalar): Contiguous load first-fault unsigned bytes to vector (scalar index).
- LDFF1B (scalar plus vector): Gather load first-fault unsigned bytes to vector (vector index).
- LDFF1B (vector plus immediate): Gather load first-fault unsigned bytes to vector (immediate index).
- LDFF1D (scalar plus scalar): Contiguous load first-fault doublewords to vector (scalar index).
- LDFF1D (scalar plus vector): Gather load first-fault doublewords to vector (vector index).
- LDFF1D (vector plus immediate): Gather load first-fault doublewords to vector (immediate index).
- LDFF1H (scalar plus scalar): Contiguous load first-fault unsigned halfwords to vector (scalar index).
- LDFF1H (scalar plus vector): Gather load first-fault unsigned halfwords to vector (vector index).
- LDFF1H (vector plus immediate): Gather load first-fault unsigned halfwords to vector (immediate index).
- LDFF1SB (scalar plus scalar): Contiguous load first-fault signed bytes to vector (scalar index).
- LDFF1SB (scalar plus vector): Gather load first-fault signed bytes to vector (vector index).
- LDFF1SB (vector plus immediate): Gather load first-fault signed bytes to vector (immediate index).
- LDFF1SH (scalar plus scalar): Contiguous load first-fault signed halfwords to vector (scalar index).

- LDFF1SH (scalar plus vector): Gather load first-fault signed halfwords to vector (vector index).
- LDFF1SH (vector plus immediate): Gather load first-fault signed halfwords to vector (immediate index).
- LDFF1SW (scalar plus scalar): Contiguous load first-fault signed words to vector (scalar index).
- LDFF1SW (scalar plus vector): Gather load first-fault signed words to vector (vector index).
- LDFF1SW (vector plus immediate): Gather load first-fault signed words to vector (immediate index).
- LDFF1W (scalar plus scalar): Contiguous load first-fault unsigned words to vector (scalar index).
- LDFF1W (scalar plus vector): Gather load first-fault unsigned words to vector (vector index).
- LDFF1W (vector plus immediate): Gather load first-fault unsigned words to vector (immediate index).
- LDNF1B: Contiguous load non-fault unsigned bytes to vector (immediate index).
- LDNF1D: Contiguous load non-fault doublewords to vector (immediate index).
- LDNF1H: Contiguous load non-fault unsigned halfwords to vector (immediate index).
- LDNF1SB: Contiguous load non-fault signed bytes to vector (immediate index).
- LDNF1SH: Contiguous load non-fault signed halfwords to vector (immediate index).
- LDNF1SW: Contiguous load non-fault signed words to vector (immediate index).
- LDNF1W: Contiguous load non-fault unsigned words to vector (immediate index).
- LDNT1B (vector plus scalar): Gather load non-temporal unsigned bytes.
- LDNT1D (vector plus scalar): Gather load non-temporal unsigned doublewords.
- LDNT1H (vector plus scalar): Gather load non-temporal unsigned halfwords.
- LDNT1SB: Gather load non-temporal signed bytes.
- LDNT1SH: Gather load non-temporal signed halfwords.
- LDNT1SW: Gather load non-temporal signed words.
- LDNT1W (vector plus scalar): Gather load non-temporal unsigned words.
- MATCH: Detect any matching elements, setting the condition flags.
- NMATCH: Detect no matching elements, setting the condition flags.
- PMULLB: Polynomial multiply long (bottom) [128b result only].
- PMULLT: Polynomial multiply long (top) [128b result only].
- PRFB (scalar plus vector): Gather prefetch bytes (scalar plus vector).
- PRFB (vector plus immediate): Gather prefetch bytes (vector plus immediate).
- PRFD (scalar plus vector): Gather prefetch doublewords (scalar plus vector).
- PRFD (vector plus immediate): Gather prefetch doublewords (vector plus immediate).
- PRFH (scalar plus vector): Gather prefetch halfwords (scalar plus vector).
- PRFH (vector plus immediate): Gather prefetch halfwords (vector plus immediate).
- PRFW (scalar plus vector): Gather prefetch words (scalar plus vector).
- PRFW (vector plus immediate): Gather prefetch words (vector plus immediate).
- RAX1: Bitwise rotate left by 1 and exclusive OR.
- RDFFR (unpredicated): Read the first-fault register.
- RDFFR, RDFFRS (predicated): Return predicate of successfully loaded elements.
- SETFFR: Initialise the first-fault register to all true.
- SM4E: SM4 encryption and decryption.
- SM4EKEY: SM4 key updates.
- SMMLA: Signed integer matrix multiply-accumulate.
- ST1B (scalar plus vector): Scatter store bytes from a vector (vector index).
- ST1B (vector plus immediate): Scatter store bytes from a vector (immediate index).
- ST1D (scalar plus vector): Scatter store doublewords from a vector (vector index).
- ST1D (vector plus immediate): Scatter store doublewords from a vector (immediate index).
- ST1H (scalar plus vector): Scatter store halfwords from a vector (vector index).
- ST1H (vector plus immediate): Scatter store halfwords from a vector (immediate index).
- ST1W (scalar plus vector): Scatter store words from a vector (vector index).
- ST1W (vector plus immediate): Scatter store words from a vector (immediate index).
- STNT1B (vector plus scalar): Scatter store non-temporal bytes.
- STNT1D (vector plus scalar): Scatter store non-temporal doublewords.
- STNT1H (vector plus scalar): Scatter store non-temporal halfwords.
- STNT1W (vector plus scalar): Scatter store non-temporal words.
- UMMLA: Unsigned integer matrix multiply-accumulate.
- USMMLA: Unsigned by signed integer matrix multiply-accumulate.

- WRFFR: Write the first-fault register.

If execution of an illegal SVE or SVE2 instruction is attempted when the PE is in *Streaming SVE mode*, and SVE instructions are not configured to trap, this will cause an SME exception to be taken, as defined by rule [R_{PLYVH}](#) in [C2.2.1 Exception priorities](#).

E1.2 Unimplemented SVE instructions

If execution of any SVE or SVE2 instruction is attempted when the PE is not in *Streaming SVE mode* and FEAT_SVE or FEAT_SVE2 is not implemented by the PE, and the instructions are not configured to trap, this will cause an SME exception to be taken, as defined by rule [RPLYVH](#) in [C2.2.1 Exception priorities](#).

E1.3 Reduced performance in Streaming SVE mode

Instructions which are dependent on results generated from vector or SIMD&FP register sources written to a general-purpose destination register, a predicate destination register, or the NZCV condition flags, might be significantly delayed if the PE is in *Streaming SVE mode* and FEAT_SME_FA64 is not implemented or not enabled at the current Exception level.

The following subsections list the instructions that are affected by this change.

E1.3.1 Scalar floating-point instructions

The following scalar floating-point instructions are affected.

- FCCMP: Floating-point Conditional quiet Compare (scalar).
- FCCMPE: Floating-point Conditional signaling Compare (scalar).
- FCMP: Floating-point quiet Compare (scalar).
- FCMPE: Floating-point signaling Compare (scalar).
- FCVTAS (scalar): Floating-point Convert to Signed integer, rounding to nearest with ties to Away (scalar).
- FCVTAU (scalar): Floating-point Convert to Unsigned integer, rounding to nearest with ties to Away (scalar).
- FCVTMS (scalar): Floating-point Convert to Signed integer, rounding toward Minus infinity (scalar).
- FCVTMU (scalar): Floating-point Convert to Unsigned integer, rounding toward Minus infinity (scalar).
- FCVTNS (scalar): Floating-point Convert to Signed integer, rounding to nearest with ties to even (scalar).
- FCVTNU (scalar): Floating-point Convert to Unsigned integer, rounding to nearest with ties to even (scalar).
- FCVTPS (scalar): Floating-point Convert to Signed integer, rounding toward Plus infinity (scalar).
- FCVTPU (scalar): Floating-point Convert to Unsigned integer, rounding toward Plus infinity (scalar).
- FCVTZS (scalar, fixed-point): Floating-point Convert to Signed fixed-point, rounding toward Zero (scalar).
- FCVTZS (scalar, integer): Floating-point Convert to Signed integer, rounding toward Zero (scalar).
- FCVTZU (scalar, fixed-point): Floating-point Convert to Unsigned fixed-point, rounding toward Zero (scalar).
- FCVTZU (scalar, integer): Floating-point Convert to Unsigned integer, rounding toward Zero (scalar).

This only applies to the variants of the following scalar floating-point instructions that write to a general-purpose register:

- FMOV (general): Floating-point Move to or from general-purpose register without conversion.

E1.3.2 SVE instructions

The following SVE instructions are affected.

- ANDS (predicates): Bitwise AND predicates.
- BICS (predicates): Bitwise clear predicates.
- BRKAS: Break after first true condition.
- BRKBS: Break before first true condition.
- BRKNS: Propagate break to next partition.
- BRKPAS: Break after first true condition, propagating from previous partition.
- BRKPBS: Break before first true condition, propagating from previous partition.
- CLASTA (scalar): Conditionally extract element after last to general-purpose register.
- CLASTB (scalar): Conditionally extract last element to general-purpose register.
- CMP<cc> (immediate): Compare vector to immediate.
- CMP<cc> (vectors): Compare vectors.
- CMP<cc> (wide elements): Compare vector to 64-bit wide elements.
- CNTP: Set scalar to count of true predicate elements.
- DECP (scalar): Decrement scalar by count of true predicate elements.
- EORS (predicates): Bitwise exclusive OR predicates.

- FAC<cc>: Floating-point absolute compare vectors.
- FCM<cc> (vectors): Floating-point compare vectors.
- FCM<cc> (zero): Floating-point compare vector with zero.
- INCP (scalar): Increment scalar by count of true predicate elements.
- LASTA (scalar): Extract element after last to general-purpose register.
- LASTB (scalar): Extract last element to general-purpose register.
- NANDS: Bitwise NAND predicates.
- NORs: Bitwise NOR predicates.
- ORNS (predicates): Bitwise inclusive OR inverted predicate.
- ORRS (predicates): Bitwise inclusive OR predicate.
- PFIRST: Set the first active predicate element to true.
- PNEXT: Find next active predicate.
- PTEST: Set condition flags for predicate.
- PTRUES: Initialise predicate from named constraint.
- SQDECP (scalar): Signed saturating decrement scalar by count of true predicate elements.
- SQINCP (scalar): Signed saturating increment scalar by count of true predicate elements.
- UQDECP (scalar): Unsigned saturating decrement scalar by count of true predicate elements.
- UQINCP (scalar): Unsigned saturating increment scalar by count of true predicate elements.

Chapter E2

SME Shared pseudocode

E2.1 AArch64.CheckFPAdvSIMDEnabled

```
1 // AArch64.CheckFPAdvSIMDEnabled()
2 // =====
3
4 AArch64.CheckFPAdvSIMDEnabled()
5     AArch64.CheckFPEEnabled();
6     // Check for illegal use of Advanced
7     // SIMD in Streaming SVE Mode
8     if HaveSME() && PSTATE.SM == '1' && !IsFullA64Enabled() then
9         SMEAccessTrap(SMEExceptionType_Streaming, PSTATE.EL);
```

E2.2 BFDotAdd

```
1 // BFDotAdd()
2 // =====
3 // BFloat16 2-way dot-product and add to single-precision
4 // result = addend + op1_a*op2_a + op1_b*op2_b
5
6 bits(32) BFDotAdd(bits(32) addend, bits(16) op1_a, bits(16) op1_b,
7                 bits(16) op2_a, bits(16) op2_b, FPCRTType fpcr_in)
8     FPCRTType fpcr = fpcr_in;
9
10    bits(32) prod;
11
12    bits(32) result;
13    if !HaveEBF16() || fpcr.EBF == '0' then // Standard BFloat16 behaviors
14        prod = BFAAdd(BFMul(op1_a, op2_a), BFMul(op1_b, op2_b));
15        result = BFAAdd(addend, prod);
16    else // Extended BFloat16 behaviors
17        boolean isbfloat16 = TRUE;
18        boolean fpexc = FALSE; // Do not generate floating-point exceptions
19        fpcr.DN = '1'; // Generate default NaN values
```

```

20     prod = FPDot(op1_a, op1_b, op2_a, op2_b, fpcr, isbfloat16, fpexc);
21     result = FPAdd(addend, prod, fpcr, fpexc);
22
23     return result;

```

E2.3 CheckFPAdvSIMDEnabled64

```

1 // CheckFPAdvSIMDEnabled64()
2 // =====
3 // AArch64 instruction wrapper
4
5 CheckFPAdvSIMDEnabled64()
6     AArch64.CheckFPAdvSIMDEnabled();

```

E2.4 CheckNonStreamingSVEEnabled

```

1 // CheckNonStreamingSVEEnabled()
2 // =====
3 // Checks for traps on SVE instructions that are not legal in streaming mode.
4
5 CheckNonStreamingSVEEnabled()
6     CheckSVEEnabled();
7
8     if HaveSME() && PSTATE.SM == '1' && !IsFullA64Enabled() then
9         SMEAccessTrap(SMEExceptionType_Streaming, PSTATE.EL);

```

E2.5 CheckSMEAccess

```

1 // CheckSMEAccess()
2 // =====
3 // Check that access to SME System registers is enabled.
4
5 CheckSMEAccess()
6     boolean disabled;
7     // Check if access disabled in CPACR_EL1
8     if PSTATE.EL IN {EL0, EL1} && !IsInHost() then
9         // Check SME at EL0/EL1
10        case CPACR_EL1.SMEN of
11            when 'x0' disabled = TRUE;
12            when '01' disabled = PSTATE.EL == EL0;
13            when '11' disabled = FALSE;
14        if disabled then SMEAccessTrap(SMEExceptionType_AccessTrap, EL1);
15
16    if PSTATE.EL IN {EL0, EL1, EL2} && EL2Enabled() then
17        if HaveVirtHostExt() && HCR_EL2.E2H == '1' then
18            // Check SME at EL2
19            case CPTR_EL2.SMEN of
20                when 'x0' disabled = TRUE;
21                when '01' disabled = PSTATE.EL == EL0 && HCR_EL2.TGE == '1';
22                when '11' disabled = FALSE;
23            if disabled then SMEAccessTrap(SMEExceptionType_AccessTrap, EL2);
24        else
25            if CPTR_EL2.TSM == '1' then SMEAccessTrap(SMEExceptionType_AccessTrap, EL2);
26
27    // Check if access disabled in CPTR_EL3
28    if HaveEL(EL3) then
29        if CPTR_EL3.ESM == '0' then SMEAccessTrap(SMEExceptionType_AccessTrap, EL3);

```

E2.6 CheckSMEAndZAAEnabled

```

1 // CheckSMEAndZAAEnabled()
2 // =====
3
4 CheckSMEAndZAAEnabled()
5     CheckSMEEnabled();
6
7     if PSTATE.ZA == '0' then
8         SMEAccessTrap(SMEExceptionType_InactiveZA, PSTATE.EL);

```

E2.7 CheckSMEEnabled

```
1 // CheckSMEEnabled()
2 // =====
3
4 CheckSMEEnabled()
5     boolean disabled;
6     // Check if access disabled in CPACR_EL1
7     if PSTATE.EL IN {EL0, EL1} && !IsInHost() then
8         // Check SME at EL0/EL1
9         case CPACR_EL1.SMEN of
10             when 'x0' disabled = TRUE;
11             when '01' disabled = PSTATE.EL == EL0;
12             when '11' disabled = FALSE;
13         if disabled then SMEAccessTrap(SMEExceptionType_AccessTrap, EL1);
14
15         // Check SIMD&FP at EL0/EL1
16         case CPACR_EL1.FPEN of
17             when 'x0' disabled = TRUE;
18             when '01' disabled = PSTATE.EL == EL0;
19             when '11' disabled = FALSE;
20         if disabled then AArch64.AdvSIMDFPAccessTrap(EL1);
21
22     if PSTATE.EL IN {EL0, EL1, EL2} && EL2Enabled() then
23         if HaveVirtHostExt() && HCR_EL2.E2H == '1' then
24             // Check SME at EL2
25             case CPTR_EL2.SMEN of
26                 when 'x0' disabled = TRUE;
27                 when '01' disabled = PSTATE.EL == EL0 && HCR_EL2.TGE == '1';
28                 when '11' disabled = FALSE;
29             if disabled then SMEAccessTrap(SMEExceptionType_AccessTrap, EL2);
30
31             // Check SIMD&FP at EL2
32             case CPTR_EL2.FPEN of
33                 when 'x0' disabled = TRUE;
34                 when '01' disabled = PSTATE.EL == EL0 && HCR_EL2.TGE == '1';
35                 when '11' disabled = FALSE;
36             if disabled then AArch64.AdvSIMDFPAccessTrap(EL2);
37         else
38             if CPTR_EL2.TSM == '1' then SMEAccessTrap(SMEExceptionType_AccessTrap, EL2);
39             if CPTR_EL2.TFP == '1' then AArch64.AdvSIMDFPAccessTrap(EL2);
40
41         // Check if access disabled in CPTR_EL3
42         if HaveEL(EL3) then
43             if CPTR_EL3.ESM == '0' then SMEAccessTrap(SMEExceptionType_AccessTrap, EL3);
44             if CPTR_EL3.TFP == '1' then AArch64.AdvSIMDFPAccessTrap(EL3);
```

E2.8 CheckStreamingSVEAndZAAEnabled

```
1 // CheckStreamingSVEAndZAAEnabled()
2 // =====
3
4 CheckStreamingSVEAndZAAEnabled()
5     CheckStreamingSVEEnabled();
6
7     if PSTATE.ZA == '0' then
8         SMEAccessTrap(SMEExceptionType_InactiveZA, PSTATE.EL);
```

E2.9 CheckStreamingSVEEnabled

```
1 // CheckStreamingSVEEnabled()
2 // =====
3
4 CheckStreamingSVEEnabled()
5     CheckSMEEnabled();
6
7     if PSTATE.SM == '0' then
8         SMEAccessTrap(SMEExceptionType_NotStreaming, PSTATE.EL);
```

E2.10 FPDot

```

1 // FPDot()
2 // =====
3 // Calculates single-precision result of 2-way 16-bit floating-point dot-product
4 // with a single rounding.
5 // The 'fpcr' argument supplies the FPCR control bits and 'isbfloat16'
6 // determines whether input operands are BFloat16 or half-precision type.
7 // and 'fpexc' controls the generation of floating-point exceptions.
8
9 bits(N) FPDot(bits(N DIV 2) op1_a, bits(N DIV 2) op1_b, bits(N DIV 2) op2_a,
10             bits(N DIV 2) op2_b, FPCRTYPE fpcr, boolean isbfloat16)
11     boolean fpexc = TRUE; // Generate floating-point exceptions
12     return FPDot(op1_a, op1_b, op2_a, op2_b, fpcr, isbfloat16, fpexc);
13
14 bits(N) FPDot(bits(N DIV 2) op1_a, bits(N DIV 2) op1_b, bits(N DIV 2) op2_a,
15             bits(N DIV 2) op2_b, FPCRTYPE fpcr_in, boolean isbfloat16, boolean fpexc)
16     FPCRTYPE fpcr = fpcr_in;
17
18     assert N == 32;
19     bits(N) result;
20     fpcr.AHP = '0'; // Ignore alternative half-precision option
21     rounding = FPRoundingMode(fpcr);
22
23     (type1_a, sign1_a, value1_a) = FPUnpackBase(op1_a, fpcr, fpexc, isbfloat16);
24     (type1_b, sign1_b, value1_b) = FPUnpackBase(op1_b, fpcr, fpexc, isbfloat16);
25     (type2_a, sign2_a, value2_a) = FPUnpackBase(op2_a, fpcr, fpexc, isbfloat16);
26     (type2_b, sign2_b, value2_b) = FPUnpackBase(op2_b, fpcr, fpexc, isbfloat16);
27
28     inf1_a = (type1_a == FPCRTYPE_Infinity); zero1_a = (type1_a == FPCRTYPE_Zero);
29     inf1_b = (type1_b == FPCRTYPE_Infinity); zero1_b = (type1_b == FPCRTYPE_Zero);
30     inf2_a = (type2_a == FPCRTYPE_Infinity); zero2_a = (type2_a == FPCRTYPE_Zero);
31     inf2_b = (type2_b == FPCRTYPE_Infinity); zero2_b = (type2_b == FPCRTYPE_Zero);
32
33     (done, result) = FPPProcessNaNs4(type1_a, type1_b, type2_a, type2_b,
34                                     op1_a, op1_b, op2_a, op2_b, fpcr, fpexc);
35
36     if ((inf1_a && zero2_a) || (zero1_a && inf2_a) &&
37         (inf1_b && zero2_b) || (zero1_b && inf2_b)) then
38         result = FPDefaultNaN(fpcr);
39         if fpexc then FPPProcessException(FPExc_InvalidOp, fpcr);
40
41     if !done then
42         // Determine sign and type products will have if it does not cause an Invalid
43         // Operation.
44         signPa = sign1_a EOR sign2_a;
45         signPb = sign1_b EOR sign2_b;
46         infPa = inf1_a || inf2_a;
47         infPb = inf1_b || inf2_b;
48         zeroPa = zero1_a || zero2_a;
49         zeroPb = zero1_b || zero2_b;
50
51         // Non SNaN-generated Invalid Operation cases are multiplies of zero
52         // by infinity and additions of opposite-signed infinities.
53         invalidop = ((inf1_a && zero2_a) || (zero1_a && inf2_a) ||
54                     (inf1_b && zero2_b) || (zero1_b && inf2_b) || (infPa && infPb && signPa != signPb));
55
56         if invalidop then
57             result = FPDefaultNaN(fpcr);
58             if fpexc then FPPProcessException(FPExc_InvalidOp, fpcr);
59
60         // Other cases involving infinities produce an infinity of the same sign.
61         elseif (infPa && signPa == '0') || (infPb && signPb == '0') then
62             result = FPInfinity('0');
63         elseif (infPa && signPa == '1') || (infPb && signPb == '1') then
64             result = FPInfinity('1');
65
66         // Cases where the result is exactly zero and its sign is not determined by the
67         // rounding mode are additions of same-signed zeros.
68         elseif zeroPa && zeroPb && signPa == signPb then
69             result = FPZero(signPa);
70
71         // Otherwise calculate fused sum of products and round it.
72         else
73             result_value = (value1_a * value2_a) + (value1_b * value2_b);
74             if result_value == 0.0 then // Sign of exact zero result depends on rounding mode
75                 result_sign = if rounding == FPRounding_NEGINF then '1' else '0';
76                 result = FPZero(result_sign);
77             else
78                 result = FPRound(result_value, fpcr, rounding, fpexc);
79
80     if !invalidop && fpexc then
81         FPPProcessDenorms4(type1_a, type1_b, type2_a, type2_b, N, fpcr);
82

```



```
83      return result;
```

E2.11 FPDotAdd_ZA

```
1  // FPDotAdd_ZA()
2  // =====
3  // Half-precision 2-way dot-product and add to single-precision
4  // for SME ZA-targeting instructions.
5
6  bits(N) FPDotAdd_ZA(bits(N) addend, bits(N DIV 2) op1_a, bits(N DIV 2) op1_b,
7                      bits(N DIV 2) op2_a, bits(N DIV 2) op2_b, FPCRType fpcr_in)
8      FPCRType fpcr = fpcr_in;
9      assert N == 32;
10
11     bits(N) prod;
12     boolean isbfloat16 = FALSE;
13     boolean fpxc = FALSE;    // Do not generate floating-point exceptions
14     fpcr.DN = '1';          // Generate default NaN values
15     prod = FPDot(op1_a, op1_b, op2_a, op2_b, fpcr, isbfloat16, fpxc);
16     result = FPAdd(addend, prod, fpcr, fpxc);
17
18     return result;
```

E2.12 FPMulAdd_ZA

```
1  // FPMulAdd_ZA()
2  // =====
3  // Calculates addend + op1*op2 with a single rounding for SME ZA-targeting
4  // instructions.
5
6  bits(N) FPMulAdd_ZA(bits(N) addend, bits(N) op1, bits(N) op2, FPCRType fpcr_in)
7      FPCRType fpcr = fpcr_in;
8      boolean fpxc = FALSE;    // Do not generate floating-point exceptions
9      fpcr.DN = '1';          // Generate default NaN values
10     return FPMulAdd(addend, op1, op2, fpcr, fpxc);
```

E2.13 FPProcessDenorms4

```
1  // FPProcessDenorms4()
2  // =====
3  // Handles denormal input in case of single-precision or double-precision
4  // when using alternative floating-point mode.
5
6  FPProcessDenorms4(FPType type1, FPType type2, FPType type3, FPType type4, integer N, FPCRType fpcr)
7      boolean altfp = HaveAltFP() && !UsingAArch32() && fpcr.AH == '1';
8      if altfp && N != 16 && (type1 == FPType_Denormal || type2 == FPType_Denormal ||
9                             type3 == FPType_Denormal || type4 == FPType_Denormal) then
10         FPProcessException(FPExc_InputDenorm, fpcr);
```

E2.14 FPProcessNaNs4

```
1  // FPProcessNaNs4()
2  // =====
3  // The boolean part of the return value says whether a NaN has been found and
4  // processed. The bits(N) part is only relevant if it has and supplies the
5  // result of the operation.
6  //
7  // The 'fpcr' argument supplies FPCR control bits.
8  // Status information is updated directly in the FPSR where appropriate.
9  // The 'fpxc' controls the generation of floating-point exceptions.
10
11 (boolean, bits(N)) FPProcessNaNs4(FPType type1, FPType type2, FPType type3, FPType type4,
12                                   bits(N DIV 2) op1, bits(N DIV 2) op2, bits(N DIV 2) op3,
13                                   bits(N DIV 2) op4, FPCRType fpcr, boolean fpxc)
14
15     assert N == 32;
16
17     bits(N) result;
```

```
18  boolean done;
19  // The FPCR.AH control does not affect these checks
20  if type1 == FType_SNaN then
21      done = TRUE; result = FPConvertNaN(FPProcessNaN(type1, op1, fpcr, fpexc));
22  elseif type2 == FType_SNaN then
23      done = TRUE; result = FPConvertNaN(FPProcessNaN(type2, op2, fpcr, fpexc));
24  elseif type3 == FType_SNaN then
25      done = TRUE; result = FPConvertNaN(FPProcessNaN(type3, op3, fpcr, fpexc));
26  elseif type4 == FType_SNaN then
27      done = TRUE; result = FPConvertNaN(FPProcessNaN(type4, op4, fpcr, fpexc));
28  elseif type1 == FType_QNaN then
29      done = TRUE; result = FPConvertNaN(FPProcessNaN(type1, op1, fpcr, fpexc));
30  elseif type2 == FType_QNaN then
31      done = TRUE; result = FPConvertNaN(FPProcessNaN(type2, op2, fpcr, fpexc));
32  elseif type3 == FType_QNaN then
33      done = TRUE; result = FPConvertNaN(FPProcessNaN(type3, op3, fpcr, fpexc));
34  elseif type4 == FType_QNaN then
35      done = TRUE; result = FPConvertNaN(FPProcessNaN(type4, op4, fpcr, fpexc));
36  else
37      done = FALSE; result = Zeros(); // 'Don't care' result
38
39  return (done, result);
```

E2.15 HaveEBF16

```
1  // HaveEBF16()
2  // =====
3  // Returns TRUE if the EBF16 extension is implemented, FALSE otherwise.
4
5  boolean HaveEBF16()
6      return boolean IMPLEMENTATION_DEFINED "Have EBF16 extension";
```

E2.16 HaveSME

```
1  // HaveSME()
2  // =====
3  // Returns TRUE if the SME extension is implemented, FALSE otherwise.
4
5  boolean HaveSME()
6      return boolean IMPLEMENTATION_DEFINED "Have SME extension";
```

E2.17 HaveSMEF64F64

```
1  // HaveSMEF64F64()
2  // =====
3  // Returns TRUE if the SMEF64F64 extension is implemented, FALSE otherwise.
4
5  boolean HaveSMEF64F64()
6      return HaveSME() && boolean IMPLEMENTATION_DEFINED "Have SMEF64F64 extension";
```

E2.18 HaveSMEI16I64

```
1  // HaveSMEI16I64()
2  // =====
3  // Returns TRUE if the SMEI16I64 extension is implemented, FALSE otherwise.
4
5  boolean HaveSMEI16I64()
6      return HaveSME() && boolean IMPLEMENTATION_DEFINED "Have SMEI16I64 extension";
```

E2.19 ImplementedSMEVectorLength

```
1  // ImplementedSMEVectorLength()
2  // =====
3  // Reduce SVE/SME vector length to a supported value (power of two)
4
```

```
5 integer ImplementedSMEVectorLength(integer nbits_in)
6     integer maxbits = MaxImplementedSVL();
7     assert 128 <= maxbits && maxbits <= 2048 && IsPow2(maxbits);
8     integer nbits = Min(nbits_in, maxbits);
9     assert 128 <= nbits && nbits <= 2048 && Align(nbits, 128) == nbits;
10
11     // Search for a supported power-of-two VL less than or equal to nbits
12     while nbits > 128 do
13         if IsPow2(nbits) && SupportedPowerTwoSVL(nbits) then return nbits;
14         nbits = nbits - 128;
15
16     // Return the smallest supported power-of-two VL
17     nbits = 128;
18     while nbits < maxbits do
19         if SupportedPowerTwoSVL(nbits) then return nbits;
20         nbits = nbits * 2;
21
22     // The only option is maxbits
23     return maxbits;
```

E2.20 InStreamingMode

```
1 // InStreamingMode()
2 // =====
3
4 boolean InStreamingMode()
5     return HaveSME() && PSTATE.SM == '1';
```

E2.21 IsFullA64Enabled

```
1 // IsFullA64Enabled()
2 // =====
3 // Returns TRUE if full A64 is enabled in Streaming mode and FALSE otherwise.
4
5 boolean IsFullA64Enabled()
6     if !HaveSMEFullA64() then return FALSE;
7
8     // Check if full SVE disabled in SMCR_EL1
9     if PSTATE.EL IN {EL0, EL1} && !IsInHost() then
10         // Check full SVE at EL0/EL1
11         if SMCR_EL1.FA64 == '0' then return FALSE;
12
13     // Check if full SVE disabled in SMCR_EL2
14     if PSTATE.EL IN {EL0, EL1, EL2} && EL2Enabled() then
15         if SMCR_EL2.FA64 == '0' then return FALSE;
16
17     // Check if full SVE disabled in SMCR_EL3
18     if HaveEL(EL3) then
19         if SMCR_EL3.FA64 == '0' then return FALSE;
20
21     return TRUE;
```

E2.22 IsMerging

```
1 // IsMerging()
2 // =====
3 // Returns TRUE if the output elements other than the lowest are taken from
4 // the destination register.
5
6 boolean IsMerging(FPCRType fpcr)
7     bit nep = if HaveSME() && PSTATE.SM == '1' && !IsFullA64Enabled() then '0' else fpcr.NEP;
8     return HaveAltFP() && !UsingAArch32() && nep == '1';
```

E2.23 IsNormalSVEEnabled

```
1 // IsNormalSVEEnabled()
2 // =====
3 // Returns TRUE if access to normal SVE is enabled at the target
```

```

4 // exception level and FALSE otherwise.
5
6 boolean IsNormalSVEEnabled(bits(2) el)
7     boolean disabled;
8     if ELUsingAArch32(el) then
9         return FALSE;
10
11     // Check if access disabled in CPACR_EL1
12     if el IN {EL0, EL1} && !IsInHost() then
13         // Check SVE at EL0/EL1
14         case CPACR_EL1.ZEN of
15             when 'x0' disabled = TRUE;
16             when '01' disabled = el == EL0;
17             when '11' disabled = FALSE;
18         if disabled then return FALSE;
19
20     // Check if access disabled in CPTR_EL2
21     if el IN {EL0, EL1, EL2} && EL2Enabled() then
22         if HaveVirtHostExt() && HCR_EL2.E2H == '1' then
23             case CPTR_EL2.ZEN of
24                 when 'x0' disabled = TRUE;
25                 when '01' disabled = el == EL0 && HCR_EL2.TGE == '1';
26                 when '11' disabled = FALSE;
27             if disabled then return FALSE;
28         else
29             if CPTR_EL2.TZ == '1' then return FALSE;
30
31     // Check if access disabled in CPTR_EL3
32     if HaveEL(EL3) then
33         if CPTR_EL3.EZ == '0' then return FALSE;
34
35     return TRUE;

```

E2.24 IsStreamingSVEEnabled

```

1 // IsStreamingSVEEnabled()
2 // =====
3 // Returns TRUE if access to streaming SVE is enabled at the
4 // target exception level and FALSE otherwise.
5
6 boolean IsStreamingSVEEnabled(bits(2) el)
7     boolean disabled;
8     if ELUsingAArch32(el) then
9         return FALSE;
10
11     // Check if access disabled in CPACR_EL1
12     if el IN {EL0, EL1} && !IsInHost() then
13         // Check SME at EL0/EL1
14         case CPACR_EL1.SMEN of
15             when 'x0' disabled = TRUE;
16             when '01' disabled = el == EL0;
17             when '11' disabled = FALSE;
18         if disabled then return FALSE;
19
20     // Check if access disabled in CPTR_EL2
21     if el IN {EL0, EL1, EL2} && EL2Enabled() then
22         if HaveVirtHostExt() && HCR_EL2.E2H == '1' then
23             case CPTR_EL2.SMEN of
24                 when 'x0' disabled = TRUE;
25                 when '01' disabled = el == EL0 && HCR_EL2.TGE == '1';
26                 when '11' disabled = FALSE;
27             if disabled then return FALSE;
28         else
29             if CPTR_EL2.TSM == '1' then return FALSE;
30
31     // Check if access disabled in CPTR_EL3
32     if HaveEL(EL3) then
33         if CPTR_EL3.ESM == '0' then return FALSE;
34
35     return TRUE;

```

E2.25 IsSVEEnabled

```

1 // IsSVEEnabled()
2 // =====

```

```

3 // Returns TRUE if access to SVE instructions and System registers is
4 // enabled at the target exception level and FALSE otherwise.
5
6 boolean IsSVEEnabled(bits(2) el)
7     if HaveSME() && PSTATE.SM == '1' then
8         return IsStreamingSVEEnabled(el);
9     elseif HaveSVE() then
10        return IsNormalSVEEnabled(el);
11    else
12        return FALSE;

```

E2.26 MaybeZeroSVEUppers

```

1 // MaybeZeroSVEUppers()
2 // =====
3
4 MaybeZeroSVEUppers(bits(2) target_el)
5     boolean lower_enabled;
6
7     if UInt(target_el) <= UInt(PSTATE.EL) || !IsSVEEnabled(target_el) then
8         return;
9
10    if target_el == EL3 then
11        if EL2Enabled() then
12            lower_enabled = IsFPEnabled(EL2);
13        else
14            lower_enabled = IsFPEnabled(EL1);
15    elseif target_el == EL2 then
16        assert !ELUsingAArch32(EL2);
17        if HCR_EL2.TGE == '0' then
18            lower_enabled = IsFPEnabled(EL1);
19        else
20            lower_enabled = IsFPEnabled(EL0);
21    else
22        assert target_el == EL1 && !ELUsingAArch32(EL1);
23        lower_enabled = IsFPEnabled(EL0);
24
25    if lower_enabled then
26        integer vl = if IsSVEEnabled(PSTATE.EL) then VL else 128;
27        integer pl = vl DIV 8;
28        for n = 0 to 31
29            if ConstrainUnpredictableBool(Unpredictable_SVEZEROUPPER) then
30                _Z[n] = ZeroExtend(_Z[n]<vl-1:0>);
31        for n = 0 to 15
32            if ConstrainUnpredictableBool(Unpredictable_SVEZEROUPPER) then
33                _P[n] = ZeroExtend(_P[n]<pl-1:0>);
34            if ConstrainUnpredictableBool(Unpredictable_SVEZEROUPPER) then
35                _FFR = ZeroExtend(_FFR<pl-1:0>);
36        if HaveSME() && PSTATE.ZA == '1' then
37            integer accessiblerows = VL DIV 8;
38            integer allrows = MAX_VL DIV 8;
39
40            for n = 0 to accessiblerows - 1
41                if ConstrainUnpredictableBool(Unpredictable_SMEZEROUPPER) then
42                    _ZA[n] = ZeroExtend(_ZA[n]<VL-1:0>);
43            for n = accessiblerows to allrows - 1
44                if ConstrainUnpredictableBool(Unpredictable_SMEZEROUPPER) then
45                    _ZA[n] = Zeros();

```

E2.27 NVL

```

1 // NVL - non-assignment form
2 // =====
3 // Normal VL
4
5 integer NVL
6     integer vl;
7
8     if PSTATE.EL == EL1 || (PSTATE.EL == EL0 && !IsInHost()) then
9         vl = UInt(ZCR_EL1.LEN);
10
11    if PSTATE.EL == EL2 || (PSTATE.EL == EL0 && IsInHost()) then
12        vl = UInt(ZCR_EL2.LEN);
13    elseif PSTATE.EL IN {EL0, EL1} && EL2Enabled() then
14        vl = Min(vl, UInt(ZCR_EL2.LEN));

```

```

15
16     if PSTATE.EL == EL3 then
17         vl = UInt(ZCR_EL3.LEN);
18     elseif HaveEL(EL3) then
19         vl = Min(vl, UInt(ZCR_EL3.LEN));
20
21     vl = (vl + 1) * 128;
22     vl = ImplementedSVEVectorLength(vl);
23
24     return vl;

```

E2.28 ResetSMESState

```

1  // ResetSMESState()
2  // =====
3
4  ResetSMESState()
5      integer vectors = MAX_VL DIV 8;
6      for n = 0 to vectors - 1
7          _ZA[n] = Zeros();

```

E2.29 ResetSVEState

```

1  // ResetSVEState()
2  // =====
3
4  ResetSVEState()
5      for n = 0 to 31
6          _Z[n] = Zeros();
7      for n = 0 to 15
8          _P[n] = Zeros();
9      _FFR = Zeros();
10     FPSR = ZeroExtend(0x0800009f<31:0>);

```

E2.30 SetPSTATE_SM

```

1  // SetPSTATE_SM()
2  // =====
3
4  SetPSTATE_SM(bit value)
5      if PSTATE.SM != value then
6          ResetSVEState();
7          PSTATE.SM = value;

```

E2.31 SetPSTATE_SVCR

```

1  // SetPSTATE_SVCR
2  // =====
3
4  SetPSTATE_SVCR(bits(32) svcr)
5      SetPSTATE_SM(svcr<0>);
6      SetPSTATE_ZA(svcr<1>);

```

E2.32 SetPSTATE_ZA

```

1  // SetPSTATE_ZA()
2  // =====
3
4  SetPSTATE_ZA(bit value)
5      if PSTATE.ZA != value then
6          ResetSMESState();
7          PSTATE.ZA = value;

```

E2.33 SMEAccessTrap

```
1 // SMEAccessTrap()
2 // =====
3 // Trapped access to SME registers due to CPACR_EL1, CPTR_EL2, or CPTR_EL3.
4
5 SMEAccessTrap(SMEExceptionType etype, bits(2) target_el_in)
6     bits(2) target_el = target_el_in;
7     assert UInt(target_el) >= UInt(PSTATE.EL);
8     if target_el == EL0 then
9         target_el = EL1;
10    boolean route_to_el2 = PSTATE.EL == EL0 && target_el == EL1 && EL2Enabled() && HCR_EL2.TGE == '1';
11
12    exception = ExceptionSyndrome(Exception_SMEAccessTrap);
13    bits(64) preferred_exception_return = ThisInstrAddr();
14    vect_offset = 0x0;
15
16    case etype of
17        when SMEExceptionType_AccessTrap
18            exception.syndrome<1:0> = '00';
19        when SMEExceptionType_Streaming
20            exception.syndrome<1:0> = '01';
21        when SMEExceptionType_NotStreaming
22            exception.syndrome<1:0> = '10';
23        when SMEExceptionType_InactiveZA
24            exception.syndrome<1:0> = '11';
25
26    if route_to_el2 then
27        AArch64.TakeException(EL2, exception, preferred_exception_return, vect_offset);
28    else
29        AArch64.TakeException(target_el, exception, preferred_exception_return, vect_offset);
```

E2.34 SVL

```
1 // SVL - non-assignment form
2 // =====
3 // Streaming SVL
4
5 integer SVL
6     integer vl;
7
8     if PSTATE.EL == EL1 || (PSTATE.EL == EL0 && !IsInHost()) then
9         vl = UInt(SMCR_EL1.LEN);
10
11    if PSTATE.EL == EL2 || (PSTATE.EL == EL0 && IsInHost()) then
12        vl = UInt(SMCR_EL2.LEN);
13    elseif PSTATE.EL IN {EL0, EL1} && EL2Enabled() then
14        vl = Min(vl, UInt(SMCR_EL2.LEN));
15
16    if PSTATE.EL == EL3 then
17        vl = UInt(SMCR_EL3.LEN);
18    elseif HaveEL(EL3) then
19        vl = Min(vl, UInt(SMCR_EL3.LEN));
20
21    vl = (vl + 1) * 128;
22    vl = ImplementedSMEVectorLength(vl);
23
24    return vl;
```

E2.35 VL

```
1 // VL - non-assignment form
2 // =====
3
4 integer VL
5     return if HaveSME() && PSTATE.SM == '1' then SVL else NVL;
```

E2.36 ZAhslice

```

1 // ZAhslice[] - non-assignment form
2 // =====
3
4 bits(width) ZAhslice[integer tile, integer esize, integer slice]
5     assert esize IN {8, 16, 32, 64, 128};
6     integer tiles = esize DIV 8;
7     assert tile >= 0 && tile < tiles;
8     integer slices = SVL DIV esize;
9     assert slice >= 0 && slice < slices;
10
11     return ZAvector[tile + slice * tiles];
12
13 // ZAhslice[] - assignment form
14 // =====
15
16 ZAhslice[integer tile, integer esize, integer slice] = bits(width) value
17     assert esize IN {8, 16, 32, 64, 128};
18     integer tiles = esize DIV 8;
19     assert tile >= 0 && tile < tiles;
20     integer slices = SVL DIV esize;
21     assert slice >= 0 && slice < slices;
22
23     ZAvector[tile + slice * tiles] = value;

```

E2.37 ZAslice

```

1 // ZAslice[] - non-assignment form
2 // =====
3
4 bits(width) ZAslice[integer tile, integer esize, boolean vertical, integer slice]
5     bits(width) result;
6
7     if vertical then
8         result = ZAvslice[tile, esize, slice];
9     else
10        result = ZAhslice[tile, esize, slice];
11
12    return result;
13
14 // ZAslice[] - assignment form
15 // =====
16
17 ZAslice[integer tile, integer esize, boolean vertical, integer slice] = bits(width) value
18     if vertical then
19         ZAvslice[tile, esize, slice] = value;
20     else
21         ZAhslice[tile, esize, slice] = value;

```

E2.38 ZAtile

```

1 // ZAtile[] - non-assignment form
2 // =====
3
4 bits(width) ZAtile[integer tile, integer esize]
5     integer slices = SVL DIV esize;
6     assert width == SVL * slices;
7     bits(width) result;
8
9     for slice = 0 to slices-1
10        Elem[result, slice, SVL] = ZAhslice[tile, esize, slice];
11
12    return result;
13
14 // ZAtile[] - assignment form
15 // =====
16
17 ZAtile[integer tile, integer esize] = bits(width) value
18     integer slices = SVL DIV esize;
19     assert width == SVL * slices;
20
21     for slice = 0 to slices-1
22        ZAhslice[tile, esize, slice] = Elem[value, slice, SVL];

```


E2.39 ZAvector

```
1 // ZAvector[] - non-assignment form
2 // =====
3
4 bits(width) ZAvector[integer index]
5     assert width == SVL;
6     assert index >= 0 && index < (width DIV 8);
7
8     return _ZA[index]<width-1:0>;
9
10 // ZAvector[] - assignment form
11 // =====
12
13 ZAvector[integer index] = bits(width) value
14     assert width == SVL;
15     assert index >= 0 && index < (width DIV 8);
16
17     if ConstrainUnpredictableBool(Unpredictable_SMEZERoupper) then
18         _ZA[index] = ZeroExtend(value);
19     else
20         _ZA[index]<width-1:0> = value;
```

E2.40 ZAvslice

```
1 // ZAvslice[] - non-assignment form
2 // =====
3
4 bits(width) ZAvslice[integer tile, integer esize, integer slice]
5     integer slices = SVL DIV esize;
6     bits(width) result;
7
8     for s = 0 to slices-1
9         bits(width) hslice = ZAhslice[tile, esize, s];
10         Elem[result, s, esize] = Elem[hslice, slice, esize];
11
12     return result;
13
14 // ZAvslice[] - assignment form
15 // =====
16
17 ZAvslice[integer tile, integer esize, integer slice] = bits(width) value
18     integer slices = SVL DIV esize;
19
20     for s = 0 to slices-1
21         bits(width) hslice = ZAhslice[tile, esize, s];
22         Elem[hslice, slice, esize] = Elem[value, s, esize];
23         ZAhslice[tile, esize, s] = hslice;
```

Chapter E3

System registers affected by SME

This section provides the full information for System registers added or modified by SME.

This content is from the **2021-12** version of *Arm® Architecture Registers Armv9, for Armv9-A architecture profile* [\[2\]](#), which contains the definitive version of the register information.

E3.1 SME-Specific System registers

System registers that are added to support SME architecture.

E3.1.1 ID_AA64SMFR0_EL1, SME Feature ID register 0

The ID_AA64SMFR0_EL1 characteristics are:

Purpose

Provides information about the implemented features of the AArch64 Scalable Matrix Extension.

For general information about the interpretation of the ID registers, see Principles of the ID scheme for fields in ID registers .

Attributes

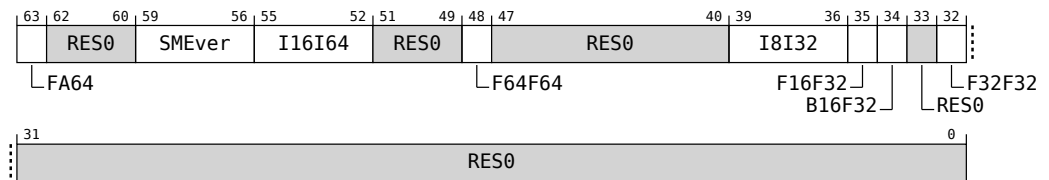
ID_AA64SMFR0_EL1 is a 64-bit register.

Configuration

Prior to the introduction of the features described by this register, this register was unnamed and reserved, RES0 from EL1, EL2, and EL3.

Field descriptions

The ID_AA64SMFR0_EL1 bit assignments are:



FA64, bit [63]

Indicates support for execution of the full A64 instruction set when the PE is in Streaming SVE mode. Defined values are:

FA64	Meaning
0b0	Only those A64 instructions defined as being legal can be executed in Streaming SVE mode.
0b1	All implemented A64 instructions can be executed in Streaming SVE mode, when enabled at the current Exception level by SMCR_EL1.FA64 , SMCR_EL2.FA64 , and SMCR_EL3.FA64 .

FEAT_SME_FA64 implements the functionality identified by the value 0b1.

Bits [62:60]

Reserved, RES0.

SMEver, bits [59:56]

Indicates support for SME instructions when [ID_AA64PFR1_EL1.SME](#) is not zero. Defined values are:

SMEver	Meaning
0b0000	The non-optional SME instructions are implemented.

All other values are reserved.

FEAT_SME implements the functionality identified by the value 0b0000, when [ID_AA64PFR1_EL1.SME](#) is not zero.

From Armv9.2, the only permitted value is 0b0000.

I16I64, bits [55:52]

Indicates SME support for instructions that accumulate into 64-bit integer elements in the ZA array. Defined values are:

I16I64	Meaning
0b0000	Instructions that accumulate into 64-bit integer elements in the ZA array are not implemented.
0b1111	The variants of the ADDHA, ADDVA, SMOPA, SMOPS, SUMOPA, SUMOPS, UMOPA, UMOPS, USMOPA, and USMOPS instructions that accumulate into 64-bit integer tiles are implemented.

All other values are reserved.

FEAT_SME_I16I64 implements the functionality identified by the value 0b1111.

The only permitted values are 0b0000 and 0b1111.

Bits [51:49]

Reserved, RES0.

F64F64, bit [48]

Indicates SME support for instructions that accumulate into FP64 double-precision floating-point elements in the ZA array. Defined values are:

F64F64	Meaning
0b0	Instructions that accumulate into double-precision floating-point elements in the ZA array are not implemented.
0b1	The variants of the FMOPA and FMOPS instructions that accumulate into double-precision tiles are implemented.

FEAT_SME_F64F64 implements the functionality identified by the value 0b1111.

Bits [47:40]

Reserved, RES0.

I8I32, bits [39:36]

Indicates SME support for instructions that accumulate 8-bit integer outer products into 32-bit integer tiles. Defined values are:

I8I32	Meaning
0b0000	Instructions that accumulate 8-bit outer products into 32-bit tiles are not implemented.
0b1111	The SMOPA, SMOPS, SUMOPA, SUMOPS, UMOA, UMOPS, USMOPA, and USMOPS instructions that accumulate 8-bit outer products into 32-bit tiles are implemented.

All other values are reserved.

If FEAT_SME is implemented, the only permitted value is 0b1111.

F16F32, bit [35]

Indicates SME support for instructions that accumulate FP16 half-precision floating-point outer products into FP32 single-precision floating-point tiles. Defined values are:

F16F32	Meaning
0b0	Instructions that accumulate half-precision outer products into single-precision tiles are not implemented.
0b1	The FMOPA and FMOPS instructions that accumulate half-precision outer products into single-precision tiles are implemented.

If FEAT_SME is implemented, the only permitted value is 0b1.

B16F32, bit [34]

Indicates SME support for instructions that accumulate BFloat16 outer products into FP32 single-precision floating-point tiles. Defined values are:

B16F32	Meaning
0b0	Instructions that accumulate BFloat16 outer products into single-precision tiles are not implemented.
0b1	The BFMOPA and BFMOPS instructions that accumulate BFloat16 outer products into single-precision tiles are implemented.

If FEAT_SME is implemented, the only permitted value is 0b1.

Bit [33]

Reserved, RES0.

F32F32, bit [32]

Indicates SME support for instructions that accumulate FP32 single-precision floating-point outer products into single-precision floating-point tiles. Defined values are:

F32F32	Meaning
0b0	Instructions that accumulate single-precision outer products into single-precision tiles are not implemented.
0b1	The FMOPA and FMOPS instructions that accumulate single-precision outer products into single-precision tiles are implemented.

If FEAT_SME is implemented, the only permitted value is 0b1.

Bits [31:0]

Reserved, RES0.

Accessing the ID_AA64SMFR0_EL1

This register is read-only and can be accessed from EL1 and higher.

This register is only accessible from the AArch64 state.

Accesses to this register use the following encodings in the instruction encoding space:

MRS <Xt>, ID_AA64SMFR0_EL1

op0	op1	CRn	CRm	op2
0b11	0b000	0b0000	0b0100	0b101

```

1  if PSTATE.EL == EL0 then
2      if IsFeatureImplemented(FEAT_IDST) then
3          if EL2Enabled() && HCR_EL2.TGE == '1' then
4              AArch64.SystemAccessTrap(EL2, 0x18);
5          else
6              AArch64.SystemAccessTrap(EL1, 0x18);
7          else
8              UNDEFINED;
9  elseif PSTATE.EL == EL1 then
10     if EL2Enabled() && HCR_EL2.TID3 == '1' then
11         AArch64.SystemAccessTrap(EL2, 0x18);
12     else
13         X[t, 64] = ID_AA64SMFR0_EL1;
14 elseif PSTATE.EL == EL2 then
15     X[t, 64] = ID_AA64SMFR0_EL1;
16 elseif PSTATE.EL == EL3 then
17     X[t, 64] = ID_AA64SMFR0_EL1;

```

E3.1.2 MPAMSM_EL1, MPAM Streaming Mode Register

The MPAMSM_EL1 characteristics are:

Purpose

Holds information to generate MPAM labels for memory requests that are:

- Issued due to the execution of SME load and store instructions.
- Issued when the PE is in Streaming SVE mode due to the execution of SVE and SIMD&FP load and store instructions and SVE prefetch instructions.

If an implementation uses a shared SMCU, then the MPAM labels in this register have precedence over the labels in MPAM0_EL1, MPAM1_EL1, [MPAM2_EL2](#), and MPAM3_EL3.

If an implementation includes an SMCU that is not shared with other PEs, then it is IMPLEMENTATION DEFINED whether the MPAM labels in this register have precedence over the labels in MPAM0_EL1, MPAM1_EL1, [MPAM2_EL2](#), and MPAM3_EL3.

The MPAM labels in this register are only used if MPAM1_EL1.MPAMEN is 1.

For memory requests issued from EL0, the MPAM PARTID in this register is virtual and mapped into a physical PARTID when all of the following are true:

- EL2 is implemented and enabled in the current Security state, and HCR_EL2.{E2H, TGE} is not {1, 1}.
- The MPAM virtualization option is implemented and MPAMHCR_EL2.EL0_VPMEN is 1.

For memory requests issued from EL1, the MPAM PARTID in this register is virtual and mapped into a physical PARTID when all of the following are true:

- EL2 is implemented and enabled in the current Security state.
- The MPAM virtualization option is implemented and MPAMHCR_EL2.EL1_VPMEN is 1.

Attributes

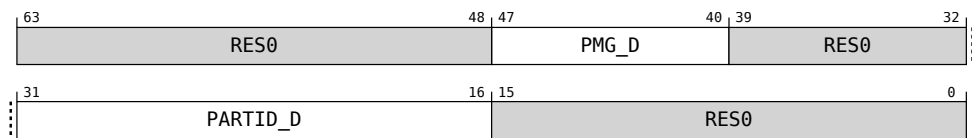
MPAMSM_EL1 is a 64-bit register.

Configuration

This register is present only when FEAT_MPAM is implemented and FEAT_SME is implemented. Otherwise, direct accesses to MPAMSM_EL1 are UNDEFINED.

Field descriptions

The MPAMSM_EL1 bit assignments are:



Bits [63:48]

Reserved, RES0.

PMG_D, bits [47:40]

Performance monitoring group property for PARTID_D.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [39:32]

Reserved, RES0.

PARTID_D, bits [31:16]

Partition ID for requests issued due to the execution at any Exception level of SME load and store instructions and, when the PE is in Streaming SVE mode, SVE and SIMD&FP load and store instructions and SVE prefetch instructions.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [15:0]

Reserved, RES0.

Accessing the MPAMSM_EL1

None of the fields in this register are permitted to be cached in a TLB.

Accesses to this register use the following encodings in the instruction encoding space:

MRS <Xt>, MPAMSM_EL1

op0	op1	CRn	CRm	op2
0b11	0b000	0b1010	0b0101	0b011

```

1 if PSTATE.EL == EL0 then
2     UNDEFINED;
3 elseif PSTATE.EL == EL1 then
4     if HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
5         if Halted() && EDSCR.SDD == '1' then
6             UNDEFINED;
7         else
8             AArch64.SystemAccessTrap(EL3, 0x18);
9     elseif EL2Enabled() && MPAM2_EL2.EnMPAMSM == '0' then
10        AArch64.SystemAccessTrap(EL2, 0x18);
11    else
12        X[t, 64] = MPAMSM_EL1;
13 elseif PSTATE.EL == EL2 then
14     if HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
15         if Halted() && EDSCR.SDD == '1' then
16             UNDEFINED;
17         else
18             AArch64.SystemAccessTrap(EL3, 0x18);
19     else
20        X[t, 64] = MPAMSM_EL1;
21 elseif PSTATE.EL == EL3 then
22     X[t, 64] = MPAMSM_EL1;

```

MSR MPAMSM_EL1, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b000	0b1010	0b0101	0b011

```

1 if PSTATE.EL == EL0 then

```

```

2      UNDEFINED;
3  elseif PSTATE.EL == EL1 then
4      if HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
5          if Halted() && EDSCR.SDD == '1' then
6              UNDEFINED;
7          else
8              AArch64.SystemAccessTrap(EL3, 0x18);
9      elseif EL2Enabled() && MPAM2_EL2.EnMPAMSM == '0' then
10         AArch64.SystemAccessTrap(EL2, 0x18);
11     else
12         MPAMSM_EL1 = X[t, 64];
13 elseif PSTATE.EL == EL2 then
14     if HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
15         if Halted() && EDSCR.SDD == '1' then
16             UNDEFINED;
17         else
18             AArch64.SystemAccessTrap(EL3, 0x18);
19     else
20         MPAMSM_EL1 = X[t, 64];
21 elseif PSTATE.EL == EL3 then
22     MPAMSM_EL1 = X[t, 64];

```

E3.1.3 SMCR_EL1, SME Control Register (EL1)

The SMCR_EL1 characteristics are:

Purpose

This register controls aspects of Streaming SVE that are visible at Exception levels EL1 and EL0.

Attributes

SMCR_EL1 is a 64-bit register.

Configuration

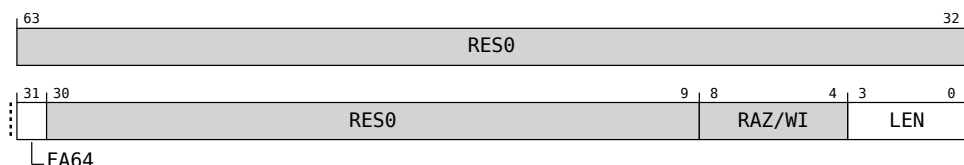
This register has no effect if the PE is not in Streaming SVE mode.

When HCR_EL2.{E2H, TGE} == {1, 1} and EL2 is enabled in the current Security state, this register has no effect on execution at EL0 and EL1.

This register is present only when FEAT_SME is implemented. Otherwise, direct accesses to SMCR_EL1 are UNDEFINED.

Field descriptions

The SMCR_EL1 bit assignments are:



Bits [63:32]

Reserved, RES0.

FA64, bit [31]

When FEAT_SME_FA64 is implemented:

Controls whether execution of an A64 instruction is considered legal when the PE is in Streaming SVE mode.

FA64	Meaning
0b0	This control does not cause any instruction to be treated as legal in Streaming SVE mode.
0b1	This control causes all implemented A64 instructions to be treated as legal in Streaming SVE mode at EL1 and EL0, if they are treated as legal at more privileged Exception levels in the current Security state.

Arm recommends that portable SME software should not rely on this optional feature, and that operating systems should provide a means to test for compliance with this recommendation.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

RES0

Bits [30:9]

Reserved, RES0.

Bits [8:4]

Reserved, RAZ/WI.

LEN, bits [3:0]

Effective Streaming SVE Vector Length (SVL).

Constrains the effective Streaming SVE vector register length for EL1 and EL0 to (LEN+1)*128 bits. SVL only takes effect when the PE is in Streaming SVE mode.

An implementation is permitted to include any set of Streaming SVE vector lengths that are powers of two, from 128 bits to 2048 bits inclusive.

For all purposes other than returning the result of a direct read of SMCR_EL1, this field selects the effective vector length as follows:

- If the requested length is smaller than the minimum implemented Streaming SVE vector length, then the minimum implemented Streaming SVE vector length is used.
- If the requested length is larger than the effective vector length at the next more privileged Exception level in the current Security state, if any, then the effective vector length at the more privileged Exception level is used.
- If the requested length is not implemented, then the requested length rounded down to the nearest implemented Streaming SVE vector length is used.
- Otherwise, the requested length is used.

An indirect read of SMCR_EL1.LEN appears to occur in program order relative to a direct write of the same register, without the need for explicit synchronization.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing the SMCR_EL1

When HCR_EL2.E2H is 1, without explicit synchronization, access from EL3 using the mnemonic SMCR_EL1 or SMCR_EL12 are not guaranteed to be ordered with respect to accesses using the other mnemonic.

Accesses to this register use the following encodings in the instruction encoding space:

MRS <Xt>, SMCR_EL1

op0	op1	CRn	CRm	op2
0b11	0b000	0b0001	0b0010	0b110

```

1 if PSTATE.EL == EL0 then
2     UNDEFINED;
3 elseif PSTATE.EL == EL1 then
4     if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap priority
5         ↳when SDD == '1'" && CPTR_EL3.ESM == '0' then
6             UNDEFINED;
7     elseif CPACR_EL1.SMEN == 'x0' then

```

```

7      AArch64.SystemAccessTrap(EL1, 0x1D);
8      elseif EL2Enabled() && HCR_EL2.E2H == '0' && CPTR_EL2.TSM == '1' then
9          AArch64.SystemAccessTrap(EL2, 0x1D);
10     elseif EL2Enabled() && HCR_EL2.E2H == '1' && CPTR_EL2.SMEN == 'x0' then
11         AArch64.SystemAccessTrap(EL2, 0x1D);
12     elseif HaveEL(EL3) && CPTR_EL3.ESM == '0' then
13         if Halted() && EDSCR.SDD == '1' then
14             UNDEFINED;
15         else
16             AArch64.SystemAccessTrap(EL3, 0x1D);
17     elseif EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '111' then
18         X[t, 64] = NVMem[0x1F0];
19     else
20         X[t, 64] = SMCR_EL1;
21 elseif PSTATE.EL == EL2 then
22     if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap priority
    ↳when SDD == '1'" && CPTR_EL3.ESM == '0' then
23         UNDEFINED;
24     elseif HCR_EL2.E2H == '0' && CPTR_EL2.TSM == '1' then
25         AArch64.SystemAccessTrap(EL2, 0x1D);
26     elseif HCR_EL2.E2H == '1' && CPTR_EL2.SMEN == 'x0' then
27         AArch64.SystemAccessTrap(EL2, 0x1D);
28     elseif HaveEL(EL3) && CPTR_EL3.ESM == '0' then
29         if Halted() && EDSCR.SDD == '1' then
30             UNDEFINED;
31         else
32             AArch64.SystemAccessTrap(EL3, 0x1D);
33     elseif HCR_EL2.E2H == '1' then
34         X[t, 64] = SMCR_EL2;
35     else
36         X[t, 64] = SMCR_EL1;
37 elseif PSTATE.EL == EL3 then
38     if CPTR_EL3.ESM == '0' then
39         AArch64.SystemAccessTrap(EL3, 0x1D);
40     else
41         X[t, 64] = SMCR_EL1;

```

MSR SMCR_EL1, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b000	0b0001	0b0010	0b110

```

1  if PSTATE.EL == EL0 then
2      UNDEFINED;
3  elseif PSTATE.EL == EL1 then
4      if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap priority
    ↳when SDD == '1'" && CPTR_EL3.ESM == '0' then
5          UNDEFINED;
6      elseif CPACR_EL1.SMEN == 'x0' then
7          AArch64.SystemAccessTrap(EL1, 0x1D);
8      elseif EL2Enabled() && HCR_EL2.E2H == '0' && CPTR_EL2.TSM == '1' then
9          AArch64.SystemAccessTrap(EL2, 0x1D);
10     elseif EL2Enabled() && HCR_EL2.E2H == '1' && CPTR_EL2.SMEN == 'x0' then
11         AArch64.SystemAccessTrap(EL2, 0x1D);
12     elseif HaveEL(EL3) && CPTR_EL3.ESM == '0' then
13         if Halted() && EDSCR.SDD == '1' then
14             UNDEFINED;
15         else
16             AArch64.SystemAccessTrap(EL3, 0x1D);
17     elseif EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '111' then
18         NVMem[0x1F0] = X[t, 64];
19     else
20         SMCR_EL1 = X[t, 64];
21 elseif PSTATE.EL == EL2 then
22     if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap priority
    ↳when SDD == '1'" && CPTR_EL3.ESM == '0' then
23         UNDEFINED;
24     elseif HCR_EL2.E2H == '0' && CPTR_EL2.TSM == '1' then
25         AArch64.SystemAccessTrap(EL2, 0x1D);
26     elseif HCR_EL2.E2H == '1' && CPTR_EL2.SMEN == 'x0' then
27         AArch64.SystemAccessTrap(EL2, 0x1D);
28     elseif HaveEL(EL3) && CPTR_EL3.ESM == '0' then

```

```

29         if Halted() && EDSCR.SDD == '1' then
30             UNDEFINED;
31         else
32             AArch64.SystemAccessTrap(EL3, 0x1D);
33         elsif HCR_EL2.E2H == '1' then
34             SMCR_EL2 = X[t, 64];
35         else
36             SMCR_EL1 = X[t, 64];
37     elsif PSTATE.EL == EL3 then
38         if CPTR_EL3.ESM == '0' then
39             AArch64.SystemAccessTrap(EL3, 0x1D);
40         else
41             SMCR_EL1 = X[t, 64];

```

MRS <Xt>, SMCR_EL12

op0	op1	CRn	CRm	op2
0b11	0b101	0b0001	0b0010	0b110

```

1  if PSTATE.EL == EL0 then
2      UNDEFINED;
3  elsif PSTATE.EL == EL1 then
4      if EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '101' then
5          X[t, 64] = NVMem[0x1F0];
6      elsif EL2Enabled() && HCR_EL2.NV == '1' then
7          AArch64.SystemAccessTrap(EL2, 0x18);
8      else
9          UNDEFINED;
10 elsif PSTATE.EL == EL2 then
11     if HCR_EL2.E2H == '1' then
12         if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
13             ↳priority when SDD == '1'" && CPTR_EL3.ESM == '0' then
14             UNDEFINED;
15         elsif CPTR_EL2.SMEN == 'x0' then
16             AArch64.SystemAccessTrap(EL2, 0x1D);
17         elsif HaveEL(EL3) && CPTR_EL3.ESM == '0' then
18             if Halted() && EDSCR.SDD == '1' then
19                 UNDEFINED;
20             else
21                 AArch64.SystemAccessTrap(EL3, 0x1D);
22             else
23                 X[t, 64] = SMCR_EL1;
24             else
25                 UNDEFINED;
26 elsif PSTATE.EL == EL3 then
27     if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '1' then
28         if CPTR_EL3.ESM == '0' then
29             AArch64.SystemAccessTrap(EL3, 0x1D);
30         else
31             X[t, 64] = SMCR_EL1;
32     else
33         UNDEFINED;

```

MSR SMCR_EL12, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b101	0b0001	0b0010	0b110

```

1  if PSTATE.EL == EL0 then
2      UNDEFINED;
3  elsif PSTATE.EL == EL1 then
4      if EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '101' then

```

```

5      NVMem[0x1F0] = X[t, 64];
6      elsif EL2Enabled() && HCR_EL2.NV == '1' then
7          AArch64.SystemAccessTrap(EL2, 0x18);
8      else
9          UNDEFINED;
10     elsif PSTATE.EL == EL2 then
11         if HCR_EL2.E2H == '1' then
12             if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
13                 ↳priority when SDD == '1'" && CPTR_EL3.ESM == '0' then
14                     UNDEFINED;
15                 elsif CPTR_EL2.SMEN == 'x0' then
16                     AArch64.SystemAccessTrap(EL2, 0x1D);
17                 elsif HaveEL(EL3) && CPTR_EL3.ESM == '0' then
18                     if Halted() && EDSCR.SDD == '1' then
19                         UNDEFINED;
20                     else
21                         AArch64.SystemAccessTrap(EL3, 0x1D);
22                     else
23                         SMCR_EL1 = X[t, 64];
24                     else
25                         UNDEFINED;
26     elsif PSTATE.EL == EL3 then
27         if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '1' then
28             if CPTR_EL3.ESM == '0' then
29                 AArch64.SystemAccessTrap(EL3, 0x1D);
30             else
31                 SMCR_EL1 = X[t, 64];
32         else
33             UNDEFINED;

```

E3.1.4 SMCR_EL2, SME Control Register (EL2)

The SMCR_EL2 characteristics are:

Purpose

This register controls aspects of Streaming SVE that are visible at Exception levels EL2, EL1, and EL0.

Attributes

SMCR_EL2 is a 64-bit register.

Configuration

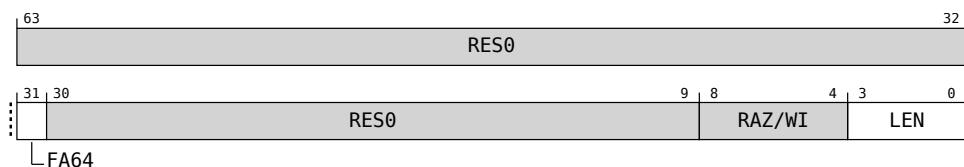
This register has no effect if the PE is not in Streaming SVE mode, or if EL2 is not enabled in the current Security state.

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

This register is present only when FEAT_SME is implemented. Otherwise, direct accesses to SMCR_EL2 are UNDEFINED.

Field descriptions

The SMCR_EL2 bit assignments are:



Bits [63:32]

Reserved, RES0.

FA64, bit [31]

When FEAT_SME_FA64 is implemented:

Controls whether execution of an A64 instruction is considered legal when the PE is in Streaming SVE mode.

FA64	Meaning
0b0	This control does not cause any instruction to be treated as legal in Streaming SVE mode.
0b1	This control causes all implemented A64 instructions to be treated as legal in Streaming SVE mode at EL2, if they are treated as legal at EL3.

Arm recommends that portable SME software should not rely on this optional feature, and that operating systems should provide a means to test for compliance with this recommendation.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

RES0

Bits [30:9]

Reserved, RES0.

Bits [8:4]

Reserved, RAZ/WI.

LEN, bits [3:0]

Effective Streaming SVE Vector Length (SVL).

Constrains the effective Streaming SVE vector register length for EL2, EL1, and EL0 to (LEN+1)*128 bits when EL2 is enabled in the current Security state. SVL only takes effect when the PE is in Streaming SVE mode.

An implementation is permitted to include any set of Streaming SVE vector lengths that are powers of two, from 128 bits to 2048 bits inclusive.

For all purposes other than returning the result of a direct read of SMCR_EL2, this field selects the effective vector length as follows:

- If the requested length is smaller than the minimum implemented Streaming SVE vector length, then the minimum implemented Streaming SVE vector length is used.
- If the requested length is larger than the effective vector length at the next more privileged Exception level in the current Security state, if any, then the effective vector length at the more privileged Exception level is used.
- If the requested length is not implemented, then the requested length rounded down to the nearest implemented Streaming SVE vector length is used.
- Otherwise, the requested length is used.

An indirect read of SMCR_EL2.LEN appears to occur in program order relative to a direct write of the same register, without the need for explicit synchronization.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing the SMCR_EL2

When HCR_EL2.E2H is 1, without explicit synchronization, access from EL2 using the mnemonic SMCR_EL2 or SMCR_EL1 are not guaranteed to be ordered with respect to accesses using the other mnemonic.

Accesses to this register use the following encodings in the instruction encoding space:

MRS <Xt>, SMCR_EL2

op0	op1	CRn	CRm	op2
0b11	0b100	0b0001	0b0010	0b110

```

1 if PSTATE.EL == EL0 then
2     UNDEFINED;
3 elseif PSTATE.EL == EL1 then
4     if EL2Enabled() && HCR_EL2.NV == '1' then
5         AArch64.SystemAccessTrap(EL2, 0x18);
6     else
7         UNDEFINED;
8 elseif PSTATE.EL == EL2 then
9     if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap priority
    ↳when SDD == '1' && CPTR_EL3.ESM == '0' then

```

Chapter E3. System registers affected by SME

E3.1. SME-Specific System registers

```

10      UNDEFINED;
11  elseif HCR_EL2.E2H == '0' && CPTR_EL2.TSM == '1' then
12      AArch64.SystemAccessTrap(EL2, 0x1D);
13  elseif HCR_EL2.E2H == '1' && CPTR_EL2.SMEN == 'x0' then
14      AArch64.SystemAccessTrap(EL2, 0x1D);
15  elseif HaveEL(EL3) && CPTR_EL3.ESM == '0' then
16      if Halted() && EDSCR.SDD == '1' then
17          UNDEFINED;
18      else
19          AArch64.SystemAccessTrap(EL3, 0x1D);
20  else
21      X[t, 64] = SMCR_EL2;
22  elseif PSTATE.EL == EL3 then
23      if CPTR_EL3.ESM == '0' then
24          AArch64.SystemAccessTrap(EL3, 0x1D);
25      else
26          X[t, 64] = SMCR_EL2;

```

MSR SMCR_EL2, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b100	0b0001	0b0010	0b110

```

1  if PSTATE.EL == EL0 then
2      UNDEFINED;
3  elseif PSTATE.EL == EL1 then
4      if EL2Enabled() && HCR_EL2.NV == '1' then
5          AArch64.SystemAccessTrap(EL2, 0x18);
6      else
7          UNDEFINED;
8  elseif PSTATE.EL == EL2 then
9      if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap priority
10         ↳when SDD == '1' && CPTR_EL3.ESM == '0' then
11             UNDEFINED;
12         elseif HCR_EL2.E2H == '0' && CPTR_EL2.TSM == '1' then
13             AArch64.SystemAccessTrap(EL2, 0x1D);
14         elseif HCR_EL2.E2H == '1' && CPTR_EL2.SMEN == 'x0' then
15             AArch64.SystemAccessTrap(EL2, 0x1D);
16         elseif HaveEL(EL3) && CPTR_EL3.ESM == '0' then
17             if Halted() && EDSCR.SDD == '1' then
18                 UNDEFINED;
19             else
20                 AArch64.SystemAccessTrap(EL3, 0x1D);
21         else
22             SMCR_EL2 = X[t, 64];
23  elseif PSTATE.EL == EL3 then
24      if CPTR_EL3.ESM == '0' then
25          AArch64.SystemAccessTrap(EL3, 0x1D);
26      else
27          SMCR_EL2 = X[t, 64];

```

MRS <Xt>, SMCR_EL1

op0	op1	CRn	CRm	op2
0b11	0b000	0b0001	0b0010	0b110

```

1  if PSTATE.EL == EL0 then
2      UNDEFINED;
3  elseif PSTATE.EL == EL1 then
4      if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap priority
5         ↳when SDD == '1' && CPTR_EL3.ESM == '0' then
6             UNDEFINED;

```

```

6      elsif CPACR_EL1.SMEN == 'x0' then
7          AArch64.SystemAccessTrap(EL1, 0x1D);
8      elsif EL2Enabled() && HCR_EL2.E2H == '0' && CPTR_EL2.TSM == '1' then
9          AArch64.SystemAccessTrap(EL2, 0x1D);
10     elsif EL2Enabled() && HCR_EL2.E2H == '1' && CPTR_EL2.SMEN == 'x0' then
11         AArch64.SystemAccessTrap(EL2, 0x1D);
12     elsif HaveEL(EL3) && CPTR_EL3.ESM == '0' then
13         if Halted() && EDSCR.SDD == '1' then
14             UNDEFINED;
15         else
16             AArch64.SystemAccessTrap(EL3, 0x1D);
17     elsif EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '111' then
18         X[t, 64] = NVMem[0x1F0];
19     else
20         X[t, 64] = SMCR_EL1;
21 elsif PSTATE.EL == EL2 then
22     if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap priority
    ↳when SDD == '1'" && CPTR_EL3.ESM == '0' then
23         UNDEFINED;
24     elsif HCR_EL2.E2H == '0' && CPTR_EL2.TSM == '1' then
25         AArch64.SystemAccessTrap(EL2, 0x1D);
26     elsif HCR_EL2.E2H == '1' && CPTR_EL2.SMEN == 'x0' then
27         AArch64.SystemAccessTrap(EL2, 0x1D);
28     elsif HaveEL(EL3) && CPTR_EL3.ESM == '0' then
29         if Halted() && EDSCR.SDD == '1' then
30             UNDEFINED;
31         else
32             AArch64.SystemAccessTrap(EL3, 0x1D);
33     elsif HCR_EL2.E2H == '1' then
34         X[t, 64] = SMCR_EL2;
35     else
36         X[t, 64] = SMCR_EL1;
37 elsif PSTATE.EL == EL3 then
38     if CPTR_EL3.ESM == '0' then
39         AArch64.SystemAccessTrap(EL3, 0x1D);
40     else
41         X[t, 64] = SMCR_EL1;

```

MSR SMCR_EL1, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b000	0b0001	0b0010	0b110

```

1  if PSTATE.EL == EL0 then
2      UNDEFINED;
3  elsif PSTATE.EL == EL1 then
4      if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap priority
    ↳when SDD == '1'" && CPTR_EL3.ESM == '0' then
5          UNDEFINED;
6      elsif CPACR_EL1.SMEN == 'x0' then
7          AArch64.SystemAccessTrap(EL1, 0x1D);
8      elsif EL2Enabled() && HCR_EL2.E2H == '0' && CPTR_EL2.TSM == '1' then
9          AArch64.SystemAccessTrap(EL2, 0x1D);
10     elsif EL2Enabled() && HCR_EL2.E2H == '1' && CPTR_EL2.SMEN == 'x0' then
11         AArch64.SystemAccessTrap(EL2, 0x1D);
12     elsif HaveEL(EL3) && CPTR_EL3.ESM == '0' then
13         if Halted() && EDSCR.SDD == '1' then
14             UNDEFINED;
15         else
16             AArch64.SystemAccessTrap(EL3, 0x1D);
17     elsif EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '111' then
18         NVMem[0x1F0] = X[t, 64];
19     else
20         SMCR_EL1 = X[t, 64];
21 elsif PSTATE.EL == EL2 then
22     if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap priority
    ↳when SDD == '1'" && CPTR_EL3.ESM == '0' then
23         UNDEFINED;
24     elsif HCR_EL2.E2H == '0' && CPTR_EL2.TSM == '1' then
25         AArch64.SystemAccessTrap(EL2, 0x1D);
26     elsif HCR_EL2.E2H == '1' && CPTR_EL2.SMEN == 'x0' then
27         AArch64.SystemAccessTrap(EL2, 0x1D);

```

```

28     elsif HaveEL(EL3) && CPTR_EL3.ESM == '0' then
29         if Halted() && EDSCR.SDD == '1' then
30             UNDEFINED;
31         else
32             AArch64.SystemAccessTrap(EL3, 0x1D);
33     elsif HCR_EL2.E2H == '1' then
34         SMCR_EL2 = X[t, 64];
35     else
36         SMCR_EL1 = X[t, 64];
37 elsif PSTATE.EL == EL3 then
38     if CPTR_EL3.ESM == '0' then
39         AArch64.SystemAccessTrap(EL3, 0x1D);
40     else
41         SMCR_EL1 = X[t, 64];

```

E3.1.5 SMCR_EL3, SME Control Register (EL3)

The SMCR_EL3 characteristics are:

Purpose

This register controls aspects of Streaming SVE that are visible at all Exception levels.

Attributes

SMCR_EL3 is a 64-bit register.

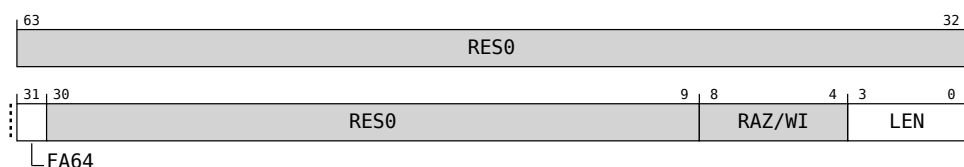
Configuration

This register has no effect if the PE is not in Streaming SVE mode.

This register is present only when FEAT_SME is implemented and EL3 is implemented. Otherwise, direct accesses to SMCR_EL3 are UNDEFINED.

Field descriptions

The SMCR_EL3 bit assignments are:



Bits [63:32]

Reserved, RES0.

FA64, bit [31]

When FEAT_SME_FA64 is implemented:

Controls whether execution of an A64 instruction is considered legal when the PE is in Streaming SVE mode.

FA64	Meaning
0b0	This control does not cause any instruction to be treated as legal in Streaming SVE mode.
0b1	This control causes all implemented A64 instructions to be treated as legal in Streaming SVE mode at EL3.

Arm recommends that portable SME software should not rely on this optional feature, and that operating systems should provide a means to test for compliance with this recommendation.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

RES0

Bits [30:9]

Reserved, RES0.

Bits [8:4]

Reserved, RAZ/WI.

LEN, bits [3:0]

Effective Streaming SVE Vector Length (SVL).

Constrains the effective Streaming SVE vector register length for all Exception levels to (LEN+1)*128 bits. SVL only takes effect when the PE is in Streaming SVE mode.

An implementation is permitted to include any set of Streaming SVE vector lengths that are powers of two, from 128 bits to 2048 bits inclusive.

For all purposes other than returning the result of a direct read of SMCR_EL3, this field selects the effective vector length as follows:

- If the requested length is smaller than the minimum implemented Streaming SVE vector length, then the minimum implemented Streaming SVE vector length is used.
- If the requested length is not implemented, then the requested length rounded down to the nearest implemented Streaming SVE vector length is used.
- Otherwise, the requested length is used.

An indirect read of SMCR_EL3.LEN appears to occur in program order relative to a direct write of the same register, without the need for explicit synchronization.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing the SMCR_EL3

Accesses to this register use the following encodings in the instruction encoding space:

MRS <Xt>, SMCR_EL3

op0	op1	CRn	CRm	op2
0b11	0b110	0b0001	0b0010	0b110

```

1  if PSTATE.EL == EL0 then
2      UNDEFINED;
3  elsif PSTATE.EL == EL1 then
4      UNDEFINED;
5  elsif PSTATE.EL == EL2 then
6      UNDEFINED;
7  elsif PSTATE.EL == EL3 then
8      if CPTR_EL3.ESM == '0' then
9          AArch64.SystemAccessTrap(EL3, 0x1D);
10     else
11         X[t, 64] = SMCR_EL3;

```

MSR SMCR_EL3, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b110	0b0001	0b0010	0b110

```

1  if PSTATE.EL == EL0 then
2      UNDEFINED;
3  elsif PSTATE.EL == EL1 then
4      UNDEFINED;
5  elsif PSTATE.EL == EL2 then
6      UNDEFINED;
7  elsif PSTATE.EL == EL3 then
8      if CPTR_EL3.ESM == '0' then
9          AArch64.SystemAccessTrap(EL3, 0x1D);
10     else
11         SMCR_EL3 = X[t, 64];

```

E3.1.6 SMIDR_EL1, Streaming Mode Identification Register

The SMIDR_EL1 characteristics are:

Purpose

Provides additional identification mechanisms for scheduling purposes, for a PE that supports Streaming SVE mode.

Attributes

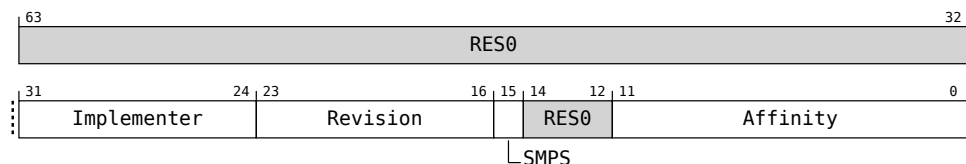
SMIDR_EL1 is a 64-bit register.

Configuration

This register is present only when FEAT_SME is implemented. Otherwise, direct accesses to SMIDR_EL1 are UNDEFINED.

Field descriptions

The SMIDR_EL1 bit assignments are:



Bits [63:32]

Reserved, RES0.

Implementer, bits [31:24]

The Implementer code. This field must hold an implementer code that has been assigned by Arm. Assigned codes include the following:

Implementer	Meaning
0x00	Reserved for software use.
0x41	Arm Limited.
0x42	Broadcom Corporation.
0x43	Cavium Inc.
0x44	Digital Equipment Corporation.
0x46	Fujitsu Ltd.
0x49	Infineon Technologies AG.
0x4D	Motorola or Freescale Semiconductor Inc.
0x4E	NVIDIA Corporation.
0x50	Applied Micro Circuits Corporation.
0x51	Qualcomm Inc.
0x56	Marvell International Ltd.

Implementer	Meaning
0x69	Intel Corporation.
0xC0	Ampere Computing.

Arm can assign codes that are not published in this manual. All values not assigned by Arm are reserved and must not be used.

It is not required that this value is the same as the value of MIDR_EL1.Implementer.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Revision, bits [23:16]

Revision number for the Streaming Mode Compute Unit (SMCU).

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

SMPS, bit [15]

Indicates support for Streaming SVE mode execution priority.

SMPS	Meaning
0b0	Priority control not supported.
0b1	Priority control supported.

Bits [14:12]

Reserved, RES0.

Affinity, bits [11:0]

The SMCU affinity of the accessing PE.

- A value of zero indicates that the PE's implementation of Streaming SVE mode is not shared with other PEs.
- Otherwise, the value identifies which SMCU is associated with this PE. The Affinity value associated with each SMCU is unique within the system as a whole.

Accessing the SMIDR_EL1

Accesses to this register use the following encodings in the instruction encoding space:

MRS <Xt>, SMIDR_EL1

op0	op1	CRn	CRm	op2
0b11	0b001	0b0000	0b0000	0b110

```
1 if PSTATE.EL == EL0 then
```

```
2      if IsFeatureImplemented(FEAT_IDST) then
3          if EL2Enabled() && HCR_EL2.TGE == '1' then
4              AArch64.SystemAccessTrap(EL2, 0x18);
5          else
6              AArch64.SystemAccessTrap(EL1, 0x18);
7          else
8              UNDEFINED;
9  elseif PSTATE.EL == EL1 then
10     if EL2Enabled() && HCR_EL2.TID1 == '1' then
11         AArch64.SystemAccessTrap(EL2, 0x18);
12     else
13         X[t, 64] = SMIDR_EL1;
14  elseif PSTATE.EL == EL2 then
15     X[t, 64] = SMIDR_EL1;
16  elseif PSTATE.EL == EL3 then
17     X[t, 64] = SMIDR_EL1;
```

E3.1.7 SMPRI_EL1, Streaming Mode Priority Register

The SMPRI_EL1 characteristics are:

Purpose

Configures the streaming execution priority for instructions executed on a shared Streaming Mode Compute Unit (SMCU) when the PE is in Streaming SVE mode at any Exception Level.

Attributes

SMPRI_EL1 is a 64-bit register.

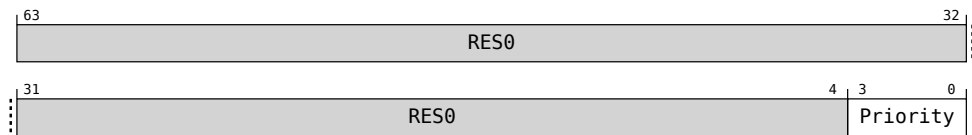
Configuration

When [SMIDR_EL1.SMPS](#) is '0', this register is RES0.

This register is present only when FEAT_SME is implemented. Otherwise, direct accesses to SMPRI_EL1 are UNDEFINED.

Field descriptions

The SMPRI_EL1 bit assignments are:



Bits [63:4]

Reserved, RES0.

Priority, bits [3:0]

Streaming execution priority value.

Either this value is used directly, or it is mapped into an effective priority value using [SMPRMAP_EL2](#).

This value is used directly when any of the following are true:

- The current Exception level is EL3 or EL2.
- The current Exception level is EL1 or EL0, if EL2 is implemented and enabled in the current Security state and [HCRX_EL2.SMPME](#) is '0'.
- The current Exception level is EL1 or EL0, if EL2 is either not implemented or not enabled in the current Security state.

The precise meaning and behavior of each streaming execution priority value is IMPLEMENTATION DEFINED.

In an implementation that shares execution resources between PEs, higher priority values are allocated more processing resource than other PEs configured with lower priority values in the same Priority domain.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing the SMPRI_EL1

Accesses to this register use the following encodings in the instruction encoding space:

MRS <Xt>, SMPRI_EL1

op0	op1	CRn	CRm	op2
0b11	0b000	0b0001	0b0010	0b100

```

1 if PSTATE.EL == EL0 then
2     UNDEFINED;
3 elseif PSTATE.EL == EL1 then
4     if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap priority
5         ↪when SDD == '1'" && CPTR_EL3.ESM == '0' then
6         UNDEFINED;
7     elseif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGTR_EL2.nSMPRI_EL1 == '0' then
8         AArch64.SystemAccessTrap(EL2, 0x18);
9     elseif HaveEL(EL3) && CPTR_EL3.ESM == '0' then
10        if Halted() && EDSCR.SDD == '1' then
11            UNDEFINED;
12        else
13            AArch64.SystemAccessTrap(EL3, 0x18);
14        else
15            X[t, 64] = SMPRI_EL1;
16 elseif PSTATE.EL == EL2 then
17     if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap priority
18         ↪when SDD == '1'" && CPTR_EL3.ESM == '0' then
19         UNDEFINED;
20     elseif HaveEL(EL3) && CPTR_EL3.ESM == '0' then
21         if Halted() && EDSCR.SDD == '1' then
22             UNDEFINED;
23         else
24             AArch64.SystemAccessTrap(EL3, 0x18);
25         else
26             X[t, 64] = SMPRI_EL1;
27 elseif PSTATE.EL == EL3 then
28     if CPTR_EL3.ESM == '0' then
29         AArch64.SystemAccessTrap(EL3, 0x18);
30     else
31         X[t, 64] = SMPRI_EL1;

```

MSR SMPRI_EL1, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b000	0b0001	0b0010	0b100

```

1 if PSTATE.EL == EL0 then
2     UNDEFINED;
3 elseif PSTATE.EL == EL1 then
4     if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap priority
5         ↪when SDD == '1'" && CPTR_EL3.ESM == '0' then
6         UNDEFINED;
7     elseif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGWTR_EL2.nSMPRI_EL1 == '0' then
8         AArch64.SystemAccessTrap(EL2, 0x18);
9     elseif HaveEL(EL3) && CPTR_EL3.ESM == '0' then
10        if Halted() && EDSCR.SDD == '1' then
11            UNDEFINED;
12        else
13            AArch64.SystemAccessTrap(EL3, 0x18);
14        else
15            SMPRI_EL1 = X[t, 64];
16 elseif PSTATE.EL == EL2 then
17     if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap priority
18         ↪when SDD == '1'" && CPTR_EL3.ESM == '0' then
19         UNDEFINED;
20     elseif HaveEL(EL3) && CPTR_EL3.ESM == '0' then
21         if Halted() && EDSCR.SDD == '1' then
22             UNDEFINED;
23         else
24             AArch64.SystemAccessTrap(EL3, 0x18);
25         else
26             SMPRI_EL1 = X[t, 64];
27 elseif PSTATE.EL == EL3 then

```

Chapter E3. System registers affected by SME

E3.1. SME-Specific System registers

```
26     if CPTR_EL3.ESM == '0' then  
27         AArch64.SystemAccessTrap(EL3, 0x18);  
28     else  
29         SMPRI_EL1 = X[t, 64];
```

E3.1.8 SMPRIMAP_EL2, Streaming Mode Priority Mapping Register

The SMPRIMAP_EL2 characteristics are:

Purpose

Maps the value in [SMPRI_EL1](#) to a streaming execution priority value for instructions executed at EL1 and EL0 in the same Security states as EL2.

Attributes

SMPRIMAP_EL2 is a 64-bit register.

Configuration

When [SMIDR_EL1](#).SMPS is '0', this register is RES0.

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

This register is present only when FEAT_SME is implemented. Otherwise, direct accesses to SMPRIMAP_EL2 are UNDEFINED.

Field descriptions

The SMPRIMAP_EL2 bit assignments are:

63	60	59	56	55	52	51	48	47	44	43	40	39	36	35	32
P15	P14	P13	P12	P11	P10	P9	P8								
31	28	27	24	23	20	19	16	15	12	11	8	7	4	3	0
P7	P6	P5	P4	P3	P2	P1	P0								

When all of the following are true, the value in [SMPRI_EL1](#) is mapped to a streaming execution priority using this register:

- The current Exception level is EL1 or EL0.
- EL2 is implemented and enabled in the current Security state.
- [HCRX_EL2](#).SMPME is '1'.

Otherwise, [SMPRI_EL1](#) holds the streaming execution priority value.

P15, bits [63:60]

Priority Mapping Entry 15. This entry is used when priority mapping is supported and enabled, and the [SMPRI_EL1](#).Priority value is '15'.

This value is the highest streaming execution priority.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

P14, bits [59:56]

Priority Mapping Entry 14. This entry is used when priority mapping is supported and enabled, and the [SMPRI_EL1](#).Priority value is '14'.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

P13, bits [55:52]

Priority Mapping Entry 13. This entry is used when priority mapping is supported and enabled, and the [SMPRI_EL1](#). Priority value is '13'.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

P12, bits [51:48]

Priority Mapping Entry 12. This entry is used when priority mapping is supported and enabled, and the [SMPRI_EL1](#). Priority value is '12'.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

P11, bits [47:44]

Priority Mapping Entry 11. This entry is used when priority mapping is supported and enabled, and the [SMPRI_EL1](#). Priority value is '11'.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

P10, bits [43:40]

Priority Mapping Entry 10. This entry is used when priority mapping is supported and enabled, and the [SMPRI_EL1](#). Priority value is '10'.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

P9, bits [39:36]

Priority Mapping Entry 9. This entry is used when priority mapping is supported and enabled, and the [SMPRI_EL1](#). Priority value is '9'.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

P8, bits [35:32]

Priority Mapping Entry 8. This entry is used when priority mapping is supported and enabled, and the [SMPRI_EL1](#). Priority value is '8'.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

P7, bits [31:28]

Priority Mapping Entry 7. This entry is used when priority mapping is supported and enabled, and the [SMPRI_EL1](#). Priority value is '7'.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

P6, bits [27:24]

Priority Mapping Entry 6. This entry is used when priority mapping is supported and enabled, and the [SMPRI_EL1](#). Priority value is '6'.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

P5, bits [23:20]

Priority Mapping Entry 5. This entry is used when priority mapping is supported and enabled, and the [SMPRI_EL1](#). Priority value is '5'.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

P4, bits [19:16]

Priority Mapping Entry 4. This entry is used when priority mapping is supported and enabled, and the [SMPRI_EL1](#). Priority value is '4'.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

P3, bits [15:12]

Priority Mapping Entry 3. This entry is used when priority mapping is supported and enabled, and the [SMPRI_EL1](#). Priority value is '3'.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

P2, bits [11:8]

Priority Mapping Entry 2. This entry is used when priority mapping is supported and enabled, and the [SMPRI_EL1](#). Priority value is '2'.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

P1, bits [7:4]

Priority Mapping Entry 1. This entry is used when priority mapping is supported and enabled, and the [SMPRI_EL1](#). Priority value is '1'.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

P0, bits [3:0]

Priority Mapping Entry 0. This entry is used when priority mapping is supported and enabled, and the [SMPRI_EL1](#).Priority value is '0'.

This value is the lowest streaming execution priority.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing the SMPRMAP_EL2

Accesses to this register use the following encodings in the instruction encoding space:

MRS <Xt>, SMPRMAP_EL2

op0	op1	CRn	CRm	op2
0b11	0b100	0b0001	0b0010	0b101

```

1  if PSTATE.EL == EL0 then
2      UNDEFINED;
3  elsif PSTATE.EL == EL1 then
4      if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
5          X[t, 64] = NVMem[0x1E8];
6      elsif EL2Enabled() && HCR_EL2.NV == '1' then
7          AArch64.SystemAccessTrap(EL2, 0x18);
8      else
9          UNDEFINED;
10  elsif PSTATE.EL == EL2 then
11      if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap priority
12          ↳when SDD == '1' && CPTR_EL3.ESM == '0' then
13              UNDEFINED;
14      elsif HaveEL(EL3) && CPTR_EL3.ESM == '0' then
15          if Halted() && EDSCR.SDD == '1' then
16              UNDEFINED;
17          else
18              AArch64.SystemAccessTrap(EL3, 0x18);
19      else
20          X[t, 64] = SMPRMAP_EL2;
21  elsif PSTATE.EL == EL3 then
22      if CPTR_EL3.ESM == '0' then
23          AArch64.SystemAccessTrap(EL3, 0x18);
24      else
25          X[t, 64] = SMPRMAP_EL2;

```

MSR SMPRMAP_EL2, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b100	0b0001	0b0010	0b101

```

1  if PSTATE.EL == EL0 then
2      UNDEFINED;
3  elsif PSTATE.EL == EL1 then
4      if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
5          NVMem[0x1E8] = X[t, 64];
6      elsif EL2Enabled() && HCR_EL2.NV == '1' then
7          AArch64.SystemAccessTrap(EL2, 0x18);
8      else
9          UNDEFINED;
10  elsif PSTATE.EL == EL2 then

```

```

11     if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap priority
    ↳when SDD == '1' && CPTR_EL3.ESM == '0' then
12         UNDEFINED;
13     elsif HaveEL(EL3) && CPTR_EL3.ESM == '0' then
14         if Halted() && EDSCR.SDD == '1' then
15             UNDEFINED;
16         else
17             AArch64.SystemAccessTrap(EL3, 0x18);
18     else
19         SMPRIMAP_EL2 = X[t, 64];
20 elsif PSTATE.EL == EL3 then
21     if CPTR_EL3.ESM == '0' then
22         AArch64.SystemAccessTrap(EL3, 0x18);
23     else
24         SMPRIMAP_EL2 = X[t, 64];

```

E3.1.9 SVCR, Streaming Vector Control Register

The SVCR characteristics are:

Purpose

Controls Streaming SVE mode and SME behavior.

Attributes

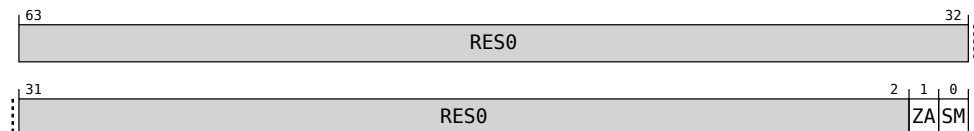
SVCR is a 64-bit register.

Configuration

This register is present only when FEAT_SME is implemented. Otherwise, direct accesses to SVCR are UNDEFINED.

Field descriptions

The SVCR bit assignments are:

**Bits [63:2]**

Reserved, RES0.

ZA, bit [1]

Enables SME ZA storage.

When this storage is disabled, execution of an instruction which can access it is trapped. The exception is reported using an ESR_ELx.{EC, SMTC} value of {0x1D, 0x3}.

The possible values of this bit are:

ZA	Meaning
0b0	ZA storage is invalid and not accessible. This control causes execution at any Exception level of instructions that can access this storage to be trapped.
0b1	ZA storage is valid and accessible. This control does not cause execution of any instructions to be trapped.

When a write to `SVCR.ZA` changes the value of `PSTATE.ZA`, the following applies:

- When changed from 0 to 1, all implemented bits of the storage are set to zero.
- When changed from 1 to 0, there is no observable change to the storage.

Changes to this field do not have an affect on the SVE vector and predicate registers and FPSR.

A direct or indirect read of `ZA` appears to occur in program order relative to a direct write of `SVCRA`, and to `MSR_SVCRA` and `MSR_SVCRSMZA` instructions, without the need for explicit synchronization.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0b0.

SM, bit [0]

Enables Streaming SVE mode.

When the PE is in Streaming SVE mode, the Streaming SVE vector length (SVL) applies to SVE instructions, and execution at any Exception level of an instruction which is illegal in that mode is trapped. The exception is reported using an ESR_ELx.{EC, SMTc} value of {0x1D, 0x1}.

When the PE is not in Streaming SVE mode, the SVE vector length (VL) applies to SVE instructions, and execution at any Exception level of an instruction which is only legal in that mode is trapped. The exception is reported using an ESR_ELx.{EC, SMTc} value of {0x1D, 0x2}.

The possible values of this bit are:

SM	Meaning
0b0	The PE is not in Streaming SVE mode.
0b1	The PE is in Streaming SVE mode.

When a write to SVCR.SM changes the value of PSTATE.SM, the following applies:

- When changed from 0 to 1, an entry to Streaming SVE mode is performed.
- When changed from 1 to 0, an exit from Streaming SVE mode is performed.
- All implemented bits of the SVE registers Z0-Z31, P0-P15, and FFR in the new mode are set to zero.
- FPSR in the new mode is set to 0x0000_0000_0800_009f, in which all cumulative status bits are set to 1.

Changes to this field do not have an affect on ZA storage.

A direct or indirect read of SM appears to occur in program order relative to a direct write of SVCR, and to MSR SVCRSM and MSR SVCRSMZA instructions, without the need for explicit synchronization.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0b0.

Accessing the SVCR

SVCR is read/write and can be accessed from any Exception level.

Accesses to this register use the following encodings in the instruction encoding space:

MRS <Xt>, SVCR

op0	op1	CRn	CRm	op2
0b11	0b011	0b0100	0b0010	0b010

```

1 if PSTATE.EL == EL0 then
2   if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap priority
   ↳when SDD == '1'" && CPTR_EL3.ESM == '0' then
3     UNDEFINED;
4   elseif !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') && CPACR_EL1.SMEN != '11' then
5     if EL2Enabled() && HCR_EL2.TGE == '1' then
6       AArch64.SystemAccessTrap(EL2, 0x1D);
7     else
8       AArch64.SystemAccessTrap(EL1, 0x1D);
9   elseif EL2Enabled() && HCR_EL2.<E2H,TGE> == '11' && CPTR_EL2.SMEN != '11' then

```

```

10      AArch64.SystemAccessTrap(EL2, 0x1D);
11      elseif EL2Enabled() && HCR_EL2.E2H == '1' && CPTR_EL2.SMEN == 'x0' then
12          AArch64.SystemAccessTrap(EL2, 0x1D);
13      elseif EL2Enabled() && HCR_EL2.E2H == '0' && CPTR_EL2.TSM == '1' then
14          AArch64.SystemAccessTrap(EL2, 0x1D);
15      elseif HaveEL(EL3) && CPTR_EL3.ESM == '0' then
16          if Halted() && EDSCR.SDD == '1' then
17              UNDEFINED;
18          else
19              AArch64.SystemAccessTrap(EL3, 0x1D);
20      else
21          X[t, 64] = Zeros(62):PSTATE.<ZA,SM>;
22      elseif PSTATE.EL == EL1 then
23          if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap priority
24              ↳when SDD == '1'" && CPTR_EL3.ESM == '0' then
25              UNDEFINED;
26          elseif CPACR_EL1.SMEN == 'x0' then
27              AArch64.SystemAccessTrap(EL1, 0x1D);
28          elseif EL2Enabled() && HCR_EL2.E2H == '0' && CPTR_EL2.TSM == '1' then
29              AArch64.SystemAccessTrap(EL2, 0x1D);
30          elseif EL2Enabled() && HCR_EL2.E2H == '1' && CPTR_EL2.SMEN == 'x0' then
31              AArch64.SystemAccessTrap(EL2, 0x1D);
32          elseif HaveEL(EL3) && CPTR_EL3.ESM == '0' then
33              if Halted() && EDSCR.SDD == '1' then
34                  UNDEFINED;
35              else
36                  AArch64.SystemAccessTrap(EL3, 0x1D);
37      else
38          X[t, 64] = Zeros(62):PSTATE.<ZA,SM>;
39      elseif PSTATE.EL == EL2 then
40          if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap priority
41              ↳when SDD == '1'" && CPTR_EL3.ESM == '0' then
42              UNDEFINED;
43          elseif HCR_EL2.E2H == '0' && CPTR_EL2.TSM == '1' then
44              AArch64.SystemAccessTrap(EL2, 0x1D);
45          elseif HCR_EL2.E2H == '1' && CPTR_EL2.SMEN == 'x0' then
46              AArch64.SystemAccessTrap(EL2, 0x1D);
47          elseif HaveEL(EL3) && CPTR_EL3.ESM == '0' then
48              if Halted() && EDSCR.SDD == '1' then
49                  UNDEFINED;
50              else
51                  AArch64.SystemAccessTrap(EL3, 0x1D);
52      else
53          X[t, 64] = Zeros(62):PSTATE.<ZA,SM>;
54      elseif PSTATE.EL == EL3 then
55          if CPTR_EL3.ESM == '0' then
56              AArch64.SystemAccessTrap(EL3, 0x1D);
57          else
58              X[t, 64] = Zeros(62):PSTATE.<ZA,SM>;

```

MSR SVCR, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b011	0b0100	0b0010	0b010

```

1  if PSTATE.EL == EL0 then
2      if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap priority
3          ↳when SDD == '1'" && CPTR_EL3.ESM == '0' then
4          UNDEFINED;
5      elseif !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') && CPACR_EL1.SMEN != '11' then
6          if EL2Enabled() && HCR_EL2.TGE == '1' then
7              AArch64.SystemAccessTrap(EL2, 0x1D);
8          else
9              AArch64.SystemAccessTrap(EL1, 0x1D);
10         elseif EL2Enabled() && HCR_EL2.<E2H,TGE> == '11' && CPTR_EL2.SMEN != '11' then
11             AArch64.SystemAccessTrap(EL2, 0x1D);
12         elseif EL2Enabled() && HCR_EL2.E2H == '1' && CPTR_EL2.SMEN == 'x0' then
13             AArch64.SystemAccessTrap(EL2, 0x1D);
14         elseif EL2Enabled() && HCR_EL2.E2H == '0' && CPTR_EL2.TSM == '1' then
15             AArch64.SystemAccessTrap(EL2, 0x1D);
16         elseif HaveEL(EL3) && CPTR_EL3.ESM == '0' then
17             if Halted() && EDSCR.SDD == '1' then

```

```

17         UNDEFINED;
18     else
19         AArch64.SystemAccessTrap(EL3, 0x1D);
20     else
21         SetPSTATE_SVCR(X[t, 32]);
22 elseif PSTATE.EL == EL1 then
23     if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap priority
24         ↪when SDD == '1' && CPTR_EL3.ESM == '0' then
25         UNDEFINED;
26     elseif CPACR_EL1.SMEN == 'x0' then
27         AArch64.SystemAccessTrap(EL1, 0x1D);
28     elseif EL2Enabled() && HCR_EL2.E2H == '0' && CPTR_EL2.TSM == '1' then
29         AArch64.SystemAccessTrap(EL2, 0x1D);
30     elseif EL2Enabled() && HCR_EL2.E2H == '1' && CPTR_EL2.SMEN == 'x0' then
31         AArch64.SystemAccessTrap(EL2, 0x1D);
32     elseif HaveEL(EL3) && CPTR_EL3.ESM == '0' then
33         if Halted() && EDSCR.SDD == '1' then
34             UNDEFINED;
35         else
36             AArch64.SystemAccessTrap(EL3, 0x1D);
37     else
38         SetPSTATE_SVCR(X[t, 32]);
39 elseif PSTATE.EL == EL2 then
40     if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap priority
41         ↪when SDD == '1' && CPTR_EL3.ESM == '0' then
42         UNDEFINED;
43     elseif HCR_EL2.E2H == '0' && CPTR_EL2.TSM == '1' then
44         AArch64.SystemAccessTrap(EL2, 0x1D);
45     elseif HCR_EL2.E2H == '1' && CPTR_EL2.SMEN == 'x0' then
46         AArch64.SystemAccessTrap(EL2, 0x1D);
47     elseif HaveEL(EL3) && CPTR_EL3.ESM == '0' then
48         if Halted() && EDSCR.SDD == '1' then
49             UNDEFINED;
50         else
51             AArch64.SystemAccessTrap(EL3, 0x1D);
52     else
53         SetPSTATE_SVCR(X[t, 32]);
54 elseif PSTATE.EL == EL3 then
55     if CPTR_EL3.ESM == '0' then
56         AArch64.SystemAccessTrap(EL3, 0x1D);
57     else
58         SetPSTATE_SVCR(X[t, 32]);

```

MSR SVCRSM, #<imm>

op0	op1	CRn	CRm	op2
0b00	0b011	0b0100	0b001x	0b011

MSR SVCRZA, #<imm>

op0	op1	CRn	CRm	op2
0b00	0b011	0b0100	0b010x	0b011

MSR SVCRSMZA, #<imm>

op0	op1	CRn	CRm	op2
0b00	0b011	0b0100	0b011x	0b011

E3.1.10 TPIDR2_EL0, EL0 Read/Write Software Thread ID Register 2

The TPIDR2_EL0 characteristics are:

Purpose

Provides a location where SME-aware software executing at EL0 can store thread identifying information, for context management purposes.

The PE makes no use of this register.

Attributes

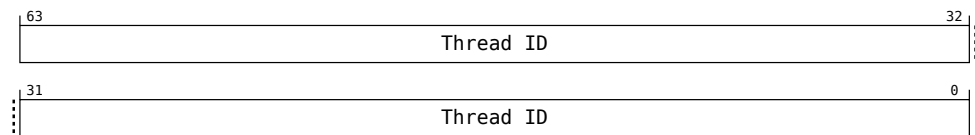
TPIDR2_EL0 is a 64-bit register.

Configuration

This register is present only when FEAT_SME is implemented. Otherwise, direct accesses to TPIDR2_EL0 are UNDEFINED.

Field descriptions

The TPIDR2_EL0 bit assignments are:



Bits [63:0]

Thread identifying information stored by software running at this Exception level.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing the TPIDR2_EL0

Accesses to this register use the following encodings in the instruction encoding space:

MRS <Xt>, TPIDR2_EL0

op0	op1	CRn	CRm	op2
0b11	0b011	0b1101	0b0000	0b101

```
1 if PSTATE.EL == EL0 then
2   if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap priority
   ↳when SDD == '1' && SCR_EL3.EnTP2 == '0' then
3     UNDEFINED;
4   elseif !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') && SCTLR_EL1.EnTP2 == '0' then
5     if EL2Enabled() && HCR_EL2.TGE == '1' then
6       AArch64.SystemAccessTrap(EL2, 0x18);
7     else
8       AArch64.SystemAccessTrap(EL1, 0x18);
9     elseif EL2Enabled() && HCR_EL2.<E2H,TGE> == '11' && SCTLR_EL2.EnTP2 == '0' then
10      AArch64.SystemAccessTrap(EL2, 0x18);
11     elseif EL2Enabled() && HCR_EL2.<E2H,TGE> != '11' && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
   ↳HFGRTR_EL2.nTPIDR2_EL0 == '0' then
12      AArch64.SystemAccessTrap(EL2, 0x18);
13     elseif HaveEL(EL3) && SCR_EL3.EnTP2 == '0' then
```



```

14         if Halted() && EDSCR.SDD == '1' then
15             UNDEFINED;
16         else
17             AArch64.SystemAccessTrap(EL3, 0x18);
18         else
19             X[t, 64] = TPIDR2_EL0;
20     elseif PSTATE.EL == EL1 then
21         if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap priority
22             ↪when SDD == '1'" && SCR_EL3.EnTP2 == '0' then
23                 UNDEFINED;
24             elseif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGTR_EL2.nTPIDR2_EL0 == '0' then
25                 AArch64.SystemAccessTrap(EL2, 0x18);
26             elseif HaveEL(EL3) && SCR_EL3.EnTP2 == '0' then
27                 if Halted() && EDSCR.SDD == '1' then
28                     UNDEFINED;
29                 else
30                     AArch64.SystemAccessTrap(EL3, 0x18);
31             else
32                 X[t, 64] = TPIDR2_EL0;
33     elseif PSTATE.EL == EL2 then
34         if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap priority
35             ↪when SDD == '1'" && SCR_EL3.EnTP2 == '0' then
36                 UNDEFINED;
37             elseif HaveEL(EL3) && SCR_EL3.EnTP2 == '0' then
38                 if Halted() && EDSCR.SDD == '1' then
39                     UNDEFINED;
40                 else
41                     AArch64.SystemAccessTrap(EL3, 0x18);
42             else
43                 X[t, 64] = TPIDR2_EL0;
44     elseif PSTATE.EL == EL3 then
45         X[t, 64] = TPIDR2_EL0;

```

MSR TPIDR2_EL0, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b011	0b1101	0b0000	0b101

```

1  if PSTATE.EL == EL0 then
2      if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap priority
3          ↪when SDD == '1'" && SCR_EL3.EnTP2 == '0' then
4              UNDEFINED;
5          elseif !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') && SCTLR_EL1.EnTP2 == '0' then
6              if EL2Enabled() && HCR_EL2.TGE == '1' then
7                  AArch64.SystemAccessTrap(EL2, 0x18);
8              else
9                  AArch64.SystemAccessTrap(EL1, 0x18);
10             elseif EL2Enabled() && HCR_EL2.<E2H,TGE> == '11' && SCTLR_EL2.EnTP2 == '0' then
11                 AArch64.SystemAccessTrap(EL2, 0x18);
12             elseif EL2Enabled() && HCR_EL2.<E2H,TGE> != '11' && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
13                 ↪HFGWTR_EL2.nTPIDR2_EL0 == '0' then
14                 AArch64.SystemAccessTrap(EL2, 0x18);
15             elseif HaveEL(EL3) && SCR_EL3.EnTP2 == '0' then
16                 if Halted() && EDSCR.SDD == '1' then
17                     UNDEFINED;
18                 else
19                     AArch64.SystemAccessTrap(EL3, 0x18);
20             else
21                 TPIDR2_EL0 = X[t, 64];
22     elseif PSTATE.EL == EL1 then
23         if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap priority
24             ↪when SDD == '1'" && SCR_EL3.EnTP2 == '0' then
25                 UNDEFINED;
26             elseif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGWTR_EL2.nTPIDR2_EL0 == '0' then
27                 AArch64.SystemAccessTrap(EL2, 0x18);
28             elseif HaveEL(EL3) && SCR_EL3.EnTP2 == '0' then
29                 if Halted() && EDSCR.SDD == '1' then
30                     UNDEFINED;
31                 else
32                     AArch64.SystemAccessTrap(EL3, 0x18);
33             else
34                 TPIDR2_EL0 = X[t, 64];

```

```

32  elseif PSTATE.EL == EL2 then
33      if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap priority
      ↪when SDD == '1'" && SCR_EL3.EnTP2 == '0' then
34          UNDEFINED;
35      elseif HaveEL(EL3) && SCR_EL3.EnTP2 == '0' then
36          if Halted() && EDSCR.SDD == '1' then
37              UNDEFINED;
38          else
39              AArch64.SystemAccessTrap(EL3, 0x18);
40      else
41          TPIDR2_EL0 = X[t, 64];
42  elseif PSTATE.EL == EL3 then
43      TPIDR2_EL0 = X[t, 64];

```

E3.1.11 EDHSR, External Debug Halt Status Register

The EDHSR characteristics are:

Purpose

Provides Debug Halt Status information.

Attributes

EDHSR is a 64-bit register.

Configuration

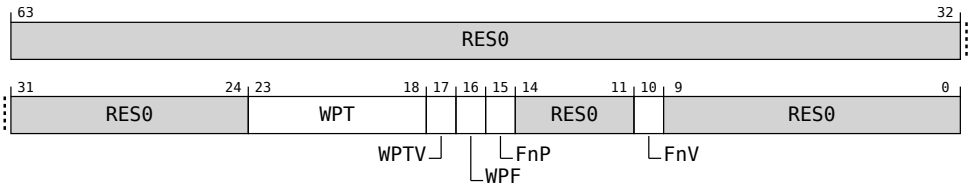
This register is only valid when the PE is in Debug state and EDSCR.STATUS is 0b101011, indicating a Watchpoint debug event. Otherwise, it has an UNKNOWN value.

The field [EDDEVID1.HSR](#) indicates support for this register.

This register is present only when FEAT_SME is implemented. Otherwise, direct accesses to EDHSR are RES0.

Field descriptions

The EDHSR bit assignments are:



Bits [63:24]

Reserved, RES0.

WPT, bits [23:18]

Watchpoint number, 0 to 15 inclusive.

All other values are reserved.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

WPTV, bit [17]

Watchpoint number Valid.

WPTV	Meaning
0b0	The WPT field is invalid, and holds an UNKNOWN value.
0b1	The WPT field is valid, and holds the number of a watchpoint that triggered a Watchpoint exception.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

WPF, bit [16]

Watchpoint might be false-positive.

WPF	Meaning
0b0	The watchpoint matched the original access or set of contiguous accesses.
0b1	The watchpoint matched an access or set of contiguous accesses where the lowest accessed address was rounded down to the nearest multiple of 16 bytes and the highest accessed address was rounded up to the nearest multiple of 16 bytes minus 1, but the watchpoint might not have matched the original access or set of contiguous accesses.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

FnP, bit [15]

FAR not Precise.

This field only has meaning if the EDWAR is valid; that is, when the FnV field is 0. If the FnV field is 1, the FnP field is 0.

FnP	Meaning
0b0	If the FnV field is 0, the EDWAR holds the virtual address of an access or set of contiguous accesses that triggered a Watchpoint exception.
0b1	The EDWAR holds any address within the smallest implemented translation granule that contains the virtual address of an access or set of contiguous accesses that triggered a Watchpoint exception.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Bits [14:11]

Reserved, RES0.

FnV, bit [10]

FAR not Valid.

FnV	Meaning
0b0	The EDWAR is valid, and its value is as described by the FnP field.

FnV	Meaning
0b1	The EDWAR is invalid, and holds an UNKNOWN value.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Bits [9:0]

Reserved, RES0.

Accessing the EDHSR

Accesses to this register use the following encodings in the instruction encoding space:

EDHSR can be accessed through the external debug interface:

Component	Offset	Instance	Range
Debug	0x038	EDHSR	31:0

This interface is accessible as follows:

- When IsCorePowered(), !DoubleLockStatus() and !OSLockStatus() access to this register is **RO**.
- Otherwise access to this register returns an ERROR.

EDHSR can be accessed through the external debug interface:

Component	Offset	Instance	Range
Debug	0x03C	EDHSR	63:32

This interface is accessible as follows:

- When IsCorePowered(), !DoubleLockStatus() and !OSLockStatus() access to this register is **RO**.
- Otherwise access to this register returns an ERROR.

E3.2 Changes to existing System registers

System registers that are updated with additional fields, values, or description changes, to support SME functionality.

E3.2.1 CPACR_EL1, Architectural Feature Access Control Register

The CPACR_EL1 characteristics are:

Purpose

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

Attributes

CPACR_EL1 is a 64-bit register.

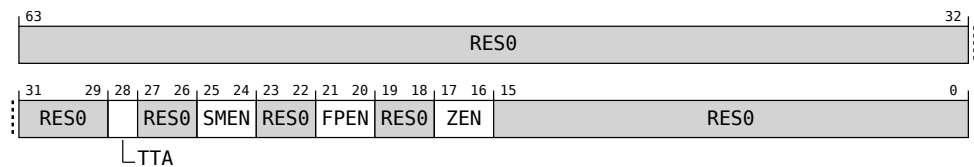
Configuration

When EL2 is implemented and enabled in the current Security state and HCR_EL2.{E2H, TGE} == {1, 1}, the fields in this register have no effect on execution at EL0 and EL1. In this case, the controls provided by [CPTR_EL2](#) are used.

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

Field descriptions

The CPACR_EL1 bit assignments are:



Bits [63:29]

Reserved, RES0.

TTA, bit [28]

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

- In AArch64 state, accesses to trace registers are trapped, reported using ESR_ELx.EC value 0x18.
- In AArch32 state, MRC and MCR accesses to trace registers are trapped, reported using ESR_ELx.EC value 0x05.
- In AArch32 state, MRRC and MCRR accesses to trace registers are trapped, reported using ESR_ELx.EC value 0x0C.

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

- The ETMv4 architecture and ETE do not permit EL0 to access the trace registers. If the PE trace unit implements FEAT_ETMv4 or FEAT_ETE, EL0 accesses to the trace registers are UNDEFINED, and any resulting exception is higher priority than an exception that would be generated because the value of [CPACR_EL1.TTA](#) is 1.

- The Arm architecture does not provide traps on trace register accesses through the optional memory-mapped interface.

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

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

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [27:26]

Reserved, RES0.

SMEN, bits [25:24]

When FEAT_SME is implemented:

Traps execution at EL1 and EL0 of SME instructions, SVE instructions when FEAT_SVE is not implemented or the PE is in Streaming SVE mode, and instructions that directly access the [SVCR](#) or [SMCR_EL1](#) System registers to EL1, or to EL2 when EL2 is implemented and enabled in the current Security state and HCR_EL2.TGE is 1.

When instructions that directly access the [SVCR](#) System register are trapped with reference to this control, the [MSR SVCRSM](#), [MSR SVCRZA](#), and [MSR SVCRSMZA](#) instructions are also trapped.

The exception is reported using ESR_ELx.EC value of 0x1D, with an ISS code of 0x0000000.

This field does not affect whether Streaming SVE or SME register values are valid.

A trap taken as a result of CPACR_EL1.SMEN has precedence over a trap taken as a result of CPACR_EL1.FPEN.

SMEN	Meaning
0b00	This control causes execution of these instructions at EL1 and EL0 to be trapped.
0b01	This control causes execution of these instructions at EL0 to be trapped, but does not cause execution of any instructions at EL1 to be trapped.
0b10	This control causes execution of these instructions at EL1 and EL0 to be trapped.
0b11	This control does not cause execution of any instructions to be trapped.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

RES0

Bits [23:22]

Reserved, RES0.

FPEN, bits [21:20]

Traps execution at EL1 and EL0 of instructions that access the Advanced SIMD and floating-point registers from

both Execution states to EL1, reported using ESR_ELx.EC value 0x07, or to EL2 reported using ESR_ELx.EC value 0x00 when EL2 is implemented and enabled in the current Security state and HCR_EL2.TGE is 1, as follows:

- In AArch64 state, accesses to [FPCR](#), FPSR, any of the SIMD and floating-point registers V0-V31, including their views as D0-D31 registers or S0-31 registers.
- In AArch32 state, FPSCR, and any of the SIMD and floating-point registers Q0-15, including their views as D0-D31 registers or S0-31 registers.

Traps execution at EL1 and EL0 of SME and SVE instructions to EL1, or to EL2 when EL2 is implemented and enabled for the current Security state and HCR_EL2.TGE is 1. The exception is reported using ESR_ELx.EC value 0x07.

A trap taken as a result of CPACR_EL1.SMEN has precedence over a trap taken as a result of CPACR_EL1.FPEN.

A trap taken as a result of CPACR_EL1.ZEN has precedence over a trap taken as a result of CPACR_EL1.FPEN.

FPEN	Meaning
0b00	This control causes execution of these instructions at EL1 and EL0 to be trapped.
0b01	This control causes execution of these instructions at EL0 to be trapped, but does not cause execution of any instructions at EL1 to be trapped.
0b10	This control causes execution of these instructions at EL1 and EL0 to be trapped.
0b11	This control does not cause execution of any instructions to be trapped.

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

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

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [19:18]

Reserved, RES0.

ZEN, bits [17:16]

When FEAT_SVE is implemented:

Traps execution at EL1 and EL0 of SVE instructions when the PE is not in Streaming SVE mode, and instructions that directly access the ZCR_EL1 System register to EL1, or to EL2 when EL2 is implemented and enabled in the current Security state and HCR_EL2.TGE is 1.

The exception is reported using ESR_ELx.EC value 0x19.

A trap taken as a result of CPACR_EL1.ZEN has precedence over a trap taken as a result of CPACR_EL1.FPEN.

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

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

RES0

Bits [15:0]

Reserved, RES0.

Accessing the CPACR_EL1

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

Accesses to this register use the following encodings in the instruction encoding space:

MRS <Xt>, CPACR_EL1

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

```

1  if PSTATE.EL == EL0 then
2      UNDEFINED;
3  elseif PSTATE.EL == EL1 then
4      if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap priority
      ↳when SDD == '1'" && CPTR_EL3.TCPAC == '1' then
5          UNDEFINED;
6      elseif EL2Enabled() && CPTR_EL2.TCPAC == '1' then
7          AArch64.SystemAccessTrap(EL2, 0x18);
8      elseif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGTR_EL2.CPACR_EL1 == '1' then
9          AArch64.SystemAccessTrap(EL2, 0x18);
10     elseif HaveEL(EL3) && CPTR_EL3.TCPAC == '1' then
11         if Halted() && EDSCR.SDD == '1' then
12             UNDEFINED;
13         else
14             AArch64.SystemAccessTrap(EL3, 0x18);
15     elseif EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '111' then
16         X[t, 64] = NVMem[0x100];
17     else
18         X[t, 64] = CPACR_EL1;
19 elseif PSTATE.EL == EL2 then
20     if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap priority
        ↳when SDD == '1'" && CPTR_EL3.TCPAC == '1' then
21         UNDEFINED;
22     elseif HaveEL(EL3) && CPTR_EL3.TCPAC == '1' then
23         if Halted() && EDSCR.SDD == '1' then

```

```

24     UNDEFINED;
25     else
26         AArch64.SystemAccessTrap(EL3, 0x18);
27     elsif HCR_EL2.E2H == '1' then
28         X[t, 64] = CPTR_EL2;
29     else
30         X[t, 64] = CPACR_EL1;
31 elsif PSTATE.EL == EL3 then
32     X[t, 64] = CPACR_EL1;

```

MSR CPACR_EL1, <Xt>

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

```

1  if PSTATE.EL == EL0 then
2      UNDEFINED;
3  elsif PSTATE.EL == EL1 then
4      if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap priority
      ↳when SDD == '1'" && CPTR_EL3.TCPAC == '1' then
5          UNDEFINED;
6      elsif EL2Enabled() && CPTR_EL2.TCPAC == '1' then
7          AArch64.SystemAccessTrap(EL2, 0x18);
8      elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEN == '1') && HFGWTR_EL2.CPACR_EL1 == '1' then
9          AArch64.SystemAccessTrap(EL2, 0x18);
10     elsif HaveEL(EL3) && CPTR_EL3.TCPAC == '1' then
11         if Halted() && EDSCR.SDD == '1' then
12             UNDEFINED;
13         else
14             AArch64.SystemAccessTrap(EL3, 0x18);
15     elsif EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '111' then
16         NVMem[0x100] = X[t, 64];
17     else
18         CPACR_EL1 = X[t, 64];
19 elsif PSTATE.EL == EL2 then
20     if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap priority
      ↳when SDD == '1'" && CPTR_EL3.TCPAC == '1' then
21         UNDEFINED;
22     elsif HaveEL(EL3) && CPTR_EL3.TCPAC == '1' then
23         if Halted() && EDSCR.SDD == '1' then
24             UNDEFINED;
25         else
26             AArch64.SystemAccessTrap(EL3, 0x18);
27     elsif HCR_EL2.E2H == '1' then
28         CPTR_EL2 = X[t, 64];
29     else
30         CPACR_EL1 = X[t, 64];
31 elsif PSTATE.EL == EL3 then
32     CPACR_EL1 = X[t, 64];

```

MRS <Xt>, CPACR_EL12

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

```

1  if PSTATE.EL == EL0 then
2      UNDEFINED;
3  elsif PSTATE.EL == EL1 then
4      if EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '101' then
5          X[t, 64] = NVMem[0x100];
6      elsif EL2Enabled() && HCR_EL2.NV == '1' then
7          AArch64.SystemAccessTrap(EL2, 0x18);

```

```

8      else
9          UNDEFINED;
10     elsif PSTATE.EL == EL2 then
11         if HCR_EL2.E2H == '1' then
12             if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
13                 ↪priority when SDD == '1'" && CPTR_EL3.TCPAC == '1' then
14                 UNDEFINED;
15             elsif HaveEL(EL3) && CPTR_EL3.TCPAC == '1' then
16                 if Halted() && EDSCR.SDD == '1' then
17                     UNDEFINED;
18                 else
19                     AArch64.SystemAccessTrap(EL3, 0x18);
20             else
21                 X[t, 64] = CPACR_EL1;
22             else
23                 UNDEFINED;
24     elsif PSTATE.EL == EL3 then
25         if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '1' then
26             X[t, 64] = CPACR_EL1;
27         else
28             UNDEFINED;

```

MSR CPACR_EL12, <Xt>

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

```

1  if PSTATE.EL == EL0 then
2      UNDEFINED;
3  elsif PSTATE.EL == EL1 then
4      if EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '101' then
5          NVMem[0x100] = X[t, 64];
6      elsif EL2Enabled() && HCR_EL2.NV == '1' then
7          AArch64.SystemAccessTrap(EL2, 0x18);
8      else
9          UNDEFINED;
10 elsif PSTATE.EL == EL2 then
11     if HCR_EL2.E2H == '1' then
12         if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
13             ↪priority when SDD == '1'" && CPTR_EL3.TCPAC == '1' then
14             UNDEFINED;
15         elsif HaveEL(EL3) && CPTR_EL3.TCPAC == '1' then
16             if Halted() && EDSCR.SDD == '1' then
17                 UNDEFINED;
18             else
19                 AArch64.SystemAccessTrap(EL3, 0x18);
20         else
21             CPACR_EL1 = X[t, 64];
22         else
23             UNDEFINED;
24 elsif PSTATE.EL == EL3 then
25     if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '1' then
26         CPACR_EL1 = X[t, 64];
27     else
28         UNDEFINED;

```

E3.2.2 CPTR_EL2, Architectural Feature Trap Register (EL2)

The CPTR_EL2 characteristics are:

Purpose

Controls trapping to EL2 of accesses to CPACR, [CPACR_EL1](#), trace, Activity Monitor, SME, Streaming SVE, SVE, and Advanced SIMD and floating-point functionality.

Attributes

CPTR_EL2 is a 64-bit register.

Configuration

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

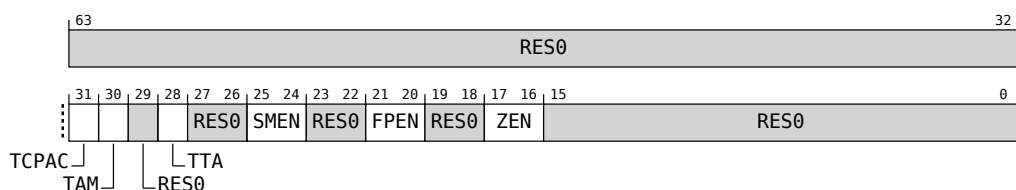
This register has no effect if EL2 is not enabled in the current Security state.

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

Field descriptions

The CPTR_EL2 bit assignments are:

When FEAT_VHE is implemented and HCR_EL2.E2H == 1:



Bits [63:32]

Reserved, RES0.

TCPAC, bit [31]

In AArch64 state, traps accesses to [CPACR_EL1](#) from EL1 to EL2, when EL2 is enabled in the current Security state. The exception is reported using ESR_ELx.EC value 0x18.

In AArch32 state, traps accesses to CPACR from EL1 to EL2, when EL2 is enabled in the current Security state. The exception is reported using ESR_ELx.EC value 0x03.

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

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

[CPACR_EL1](#) and CPACR are not accessible at EL0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

TAM, bit [30]

When FEAT_AMUv1 is implemented:

Trap Activity Monitor access. Traps EL1 and EL0 accesses to all Activity Monitor registers to EL2, as follows:

- In AArch64 state, accesses to the following registers are trapped to EL2, reported using ESR_ELx.EC value 0x18:
 - AMUSERENR_EL0, AMCFGR_EL0, AMCGCR_EL0, AMCNTENCLR0_EL0, AMCNTENCLR1_EL0, AMCNTENSET0_EL0, AMCNTENSET1_EL0, AMCR_EL0, AMEVCNTR0<n>_EL0, AMEVCNTR1<n>_EL0, AMEVTYPER0<n>_EL0, and AMEVTYPER1<n>_EL0.
- In AArch32 state, MRC or MCR accesses to the following registers are trapped to EL2 and reported using ESR_ELx.EC value 0x03:
 - AMUSERENR, AMCFGR, AMCGCR, AMCNTENCLR0, AMCNTENCLR1, AMCNTENSET0, AMCNTENSET1, AMCR, AMEVTYPER0<n>, and AMEVTYPER1<n>.
- In AArch32 state, MRRC or MCRR accesses to AMEVCNTR0<n> and AMEVCNTR1<n>, are trapped to EL2, reported using ESR_ELx.EC value 0x04.

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

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

RES0

Bit [29]

Reserved, RES0.

TTA, bit [28]

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

- In AArch64 state, accesses to trace registers with op0=2, op1=1, and CRn<0b1000 are trapped to EL2, reported using EC syndrome value 0x18.
- In AArch32 state, MRC or MCR accesses to trace registers with cpnum=14, opc1=1, and CRn<0b1000 are trapped to EL2, reported using EC syndrome value 0x05.

TTA	Meaning
0b0	This control does not cause any instructions to be trapped.

TTA	Meaning
0b1	Any attempt at EL0, EL1 or EL2, to execute a System register access to an implemented trace register is trapped to EL2, when EL2 is enabled in the current Security state, unless HCR_EL2.TGE is 0 and it is trapped by CPACR.NSTRCDIS or CPACR_EL1.TTA. When HCR_EL2.TGE is 1, any attempt at EL0 or EL2 to execute a System register access to an implemented trace register is trapped to EL2, when EL2 is enabled in the current Security state.

The ETMv4 architecture and ETE do not permit EL0 to access the trace registers. If the PE trace unit implements FEAT_ETMv4 or ETE, EL0 accesses to the trace registers are UNDEFINED, and any resulting exception is higher priority than an exception that would be generated because the value of CPTR_EL2.TTA is 1.

EL2 does not provide traps on trace register accesses through the optional Memory-mapped interface.

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

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

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [27:26]

Reserved, RES0.

SMEN, bits [25:24]

When FEAT_SME is implemented:

Traps execution at EL2, EL1, and EL0 of SME instructions, SVE instructions when FEAT_SVE is not implemented or the PE is in Streaming SVE mode, and instructions that directly access the SVCR, SMCR_EL1, or SMCR_EL2 System registers to EL2, when EL2 is enabled in the current Security state.

When instructions that directly access the SVCR System register are trapped with reference to this control, the MSR_SVCRSM, MSR_SVCRZA, and MSR_SVCRSMZA instructions are also trapped.

The exception is reported using ESR_EL2.EC value of 0x1D, with an ISS code of 0x0000000.

This field does not affect whether Streaming SVE or SME register values are valid.

A trap taken as a result of CPTR_EL2.SMEN has precedence over a trap taken as a result of CPTR_EL2.FPEN.

SMEN	Meaning
0b00	This control causes execution of these instructions at EL2, EL1, and EL0 to be trapped.
0b01	When HCR_EL2.TGE is 0, this control does not cause execution of any instructions to be trapped. When HCR_EL2.TGE is 1, this control causes execution of these instructions at EL0 to be trapped, but does not cause execution of any instructions at EL2 to be trapped.

SMEN	Meaning
0b10	This control causes execution of these instructions at EL2, EL1, and EL0 to be trapped.
0b11	This control does not cause execution of any instructions to be trapped.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

RES0

Bits [23:22]

Reserved, RES0.

FPEN, bits [21:20]

Traps execution at EL2, EL1, and EL0 of instructions that access the Advanced SIMD and floating-point registers from both Execution states to EL2, when EL2 is enabled in the current Security state. The exception is reported using ESR_ELx.EC value 0x07.

Traps execution at EL2, EL1, and EL0 of SME and SVE instructions to EL2, when EL2 is enabled in the current Security state. The exception is reported using ESR_ELx.EC value 0x07.

A trap taken as a result of CPTR_EL2.SMEN has precedence over a trap taken as a result of CPTR_EL2.FPEN.

A trap taken as a result of CPTR_EL2.ZEN has precedence over a trap taken as a result of CPTR_EL2.FPEN.

FPEN	Meaning
0b00	This control causes execution of these instructions at EL2, EL1, and EL0 to be trapped.
0b01	When HCR_EL2.TGE is 0, this control does not cause execution of any instructions to be trapped. When HCR_EL2.TGE is 1, this control causes execution of these instructions at EL0 to be trapped, but does not cause execution of any instructions at EL2 to be trapped.
0b10	This control causes execution of these instructions at EL2, EL1, and EL0 to be trapped.
0b11	This control does not cause execution of any instructions to be trapped.

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

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

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [19:18]

Reserved, RES0.

ZEN, bits [17:16]

When FEAT_SVE is implemented:

Traps execution at EL2, EL1, and EL0 of SVE instructions when the PE is not in Streaming SVE mode, and instructions that directly access the ZCR_EL1 or ZCR_EL2 System registers to EL2, when EL2 is enabled in the current Security state.

The exception is reported using ESR_ELx.EC value 0x19.

A trap taken as a result of CPTR_EL2.ZEN has precedence over a trap taken as a result of CPTR_EL2.FPEN.

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

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

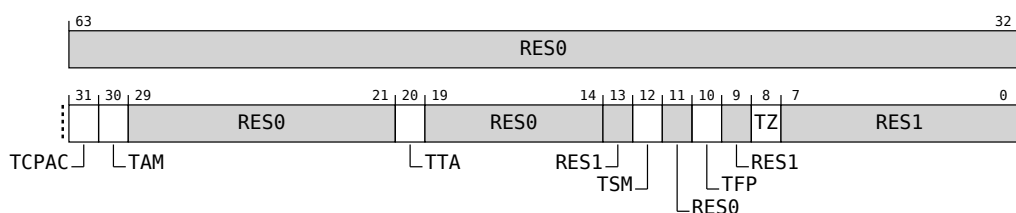
Otherwise:

RES0

Bits [15:0]

Reserved, RES0.

Otherwise:



This format applies in all Armv8.0 implementations.

Bits [63:32]

Reserved, RES0.

TCPAC, bit [31]

In AArch64 state, traps accesses to [CPACR_EL1](#) from EL1 to EL2, when EL2 is enabled in the current Security state. The exception is reported using ESR_ELx.EC value 0x18.

In AArch32 state, traps accesses to CPACR from EL1 to EL2, when EL2 is enabled in the current Security state. The exception is reported using ESR_ELx.EC value 0x03.

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

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

[CPACR_EL1](#) and CPACR are not accessible at EL0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

TAM, bit [30]

When FEAT_AMUv1 is implemented:

Trap Activity Monitor access. Traps EL1 and EL0 accesses to all Activity Monitor registers to EL2, as follows:

- In AArch64 state, accesses to the following registers are trapped to EL2, reported using ESR_ELx.EC value 0x18:
 - AMUSERENR_EL0, AMCFGR_EL0, AMCGCR_EL0, AMCNTENCLR0_EL0, AMCNTENCLR1_EL0, AMCNTENSET0_EL0, AMCNTENSET1_EL0, AMCR_EL0, AMEVCNTR0<n>_EL0, AMEVCNTR1<n>_EL0, AMEVTYPER0<n>_EL0, and AMEVTYPER1<n>_EL0.
- In AArch32 state, MCR or MRC accesses to the following registers are trapped to EL2 and reported using ESR_ELx.EC value 0x03:
 - AMUSERENR, AMCFGR, AMCGCR, AMCNTENCLR0, AMCNTENCLR1, AMCNTENSET0, AMCNTENSET1, AMCR, AMEVTYPER0<n>, and AMEVTYPER1<n>.
- In AArch32 state, MCRR or MRRC accesses to AMEVCNTR0<n> and AMEVCNTR1<n>, are trapped to EL2, reported using ESR_ELx.EC value 0x04.

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

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

RES0

Bits [29:21]

Reserved, RES0.

TTA, bit [20]

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

- In AArch64 state, accesses to trace registers with op0=2, op1=1, and CRn<0b1000 are trapped to EL2, reported using EC syndrome value 0x18.
- In AArch32 state, MRC or MCR accesses to trace registers with cpnum=14, opc1=1, and CRn<0b1000 are trapped to EL2, reported using EC syndrome value 0x05.

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

- The ETMv4 architecture does not permit EL0 to access the trace registers. If the PE trace unit implements FEAT_ETMv4, EL0 accesses to the trace registers are UNDEFINED, and any resulting exception is higher priority than an exception that would be generated because the value of [CPTR_EL2.TTA](#) is 1.
- EL2 does not provide traps on trace register accesses through the optional memory-mapped interface.

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

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

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [19:14]

Reserved, RES0.

Bit [13]

Reserved, RES1.

TSM, bit [12]

When FEAT_SME is implemented:

Traps execution at EL2, EL1, and EL0 of SME instructions, SVE instructions when FEAT_SVE is not implemented or the PE is in Streaming SVE mode, and instructions that directly access the [SVCR](#), [SMCR_EL1](#), or [SMCR_EL2](#) System registers to EL2, when EL2 is enabled in the current Security state.

When instructions that directly access the [SVCR](#) System register are trapped with reference to this control, the [MSR_SVCRSM](#), [MSR_SVCRZA](#), and [MSR_SVCRSMZA](#) instructions are also trapped.

The exception is reported using ESR_EL2.EC value of 0x1D, with an ISS code of 0x0000000.

This field does not affect whether Streaming SVE or SME register values are valid.

A trap taken as a result of CPTR_EL2.TSM has precedence over a trap taken as a result of CPTR_EL2.TFP.

TSM	Meaning
0b0	This control does not cause execution of any instructions to be trapped.
0b1	This control causes execution of these instructions at EL2, EL1, and EL0 to be trapped.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

RES1

Bit [11]

Reserved, RES0.

TFP, bit [10]

Traps execution of instructions which access the Advanced SIMD and floating-point functionality, from both Execution states to EL2, when EL2 is enabled in the current Security state, as follows:

- In AArch64 state, accesses to the following registers are trapped to EL2, reported using ESR_ELx.EC value 0x07:
 - [FPCR](#), [FPSR](#), [FPEXC32_EL2](#), any of the SIMD and floating-point registers V0-V31, including their views as D0-D31 registers or S0-31 registers.
- In AArch32 state, accesses to the following registers are trapped to EL2, reported using ESR_ELx.EC value 0x07:
 - MVFR0, MVFR1, MVFR2, FPSCR, FPEXC, and any of the SIMD and floating-point registers Q0-15, including their views as D0-D31 registers or S0-31 registers. For the purposes of this trap, the architecture defines a VMSR access to FPSID from EL1 or higher as an access to a SIMD and floating-point register. Otherwise, permitted VMSR accesses to FPSID are ignored.

Traps execution at the same Exception levels of SME and SVE instructions to EL2, when EL2 is enabled in the current Security state. The exception is reported using ESR_ELx.EC value 0x07.

A trap taken as a result of CPTR_EL2.TSM has precedence over a trap taken as a result of CPTR_EL2.TFP.

A trap taken as a result of CPTR_EL2.TZ has precedence over a trap taken as a result of CPTR_EL2.TFP.

TFP	Meaning
0b0	This control does not cause execution of any instructions to be trapped.

TFP	Meaning
0b1	This control causes execution of these instructions at EL2, EL1, and EL0 to be trapped.

FPEXC32_EL2 is not accessible from EL0 using AArch64.

FPSID, MVFR0, MVFR1, and FPEXC are not accessible from EL0 using AArch32.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bit [9]

Reserved, RES1.

TZ, bit [8]

When FEAT_SVE is implemented:

Traps execution at EL2, EL1, and EL0 of SVE instructions when the PE is not in Streaming SVE mode, and instructions that directly access the ZCR_EL2 or ZCR_EL1 System registers to EL2, when EL2 is enabled in the current Security state.

The exception is reported using ESR_ELx.EC value 0x19.

A trap taken as a result of CPTR_EL2.TZ has precedence over a trap taken as a result of CPTR_EL2.TFP.

TZ	Meaning
0b0	This control does not cause execution of any instructions to be trapped.
0b1	This control causes execution of these instructions at EL2, EL1, and EL0 to be trapped.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

RES1

Bits [7:0]

Reserved, RES1.

Accessing the CPTR_EL2

Accesses to this register use the following encodings in the instruction encoding space:

MRS <Xt>, CPTR_EL2

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

```

1 if PSTATE.EL == EL0 then
2     UNDEFINED;
3 elseif PSTATE.EL == EL1 then
4     if EL2Enabled() && HCR_EL2.NV == '1' then
5         AArch64.SystemAccessTrap(EL2, 0x18);
6     else
7         UNDEFINED;
8 elseif PSTATE.EL == EL2 then
9     if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap priority
10         ↪when SDD == '1'" && CPTR_EL3.TCPAC == '1' then
11             UNDEFINED;
12         elseif HaveEL(EL3) && CPTR_EL3.TCPAC == '1' then
13             if Halted() && EDSCR.SDD == '1' then
14                 UNDEFINED;
15             else
16                 AArch64.SystemAccessTrap(EL3, 0x18);
17         else
18             X[t, 64] = CPTR_EL2;
19 elseif PSTATE.EL == EL3 then
20     X[t, 64] = CPTR_EL2;

```

MSR CPTR_EL2, <Xt>

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

```

1 if PSTATE.EL == EL0 then
2     UNDEFINED;
3 elseif PSTATE.EL == EL1 then
4     if EL2Enabled() && HCR_EL2.NV == '1' then
5         AArch64.SystemAccessTrap(EL2, 0x18);
6     else
7         UNDEFINED;
8 elseif PSTATE.EL == EL2 then
9     if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap priority
10         ↪when SDD == '1'" && CPTR_EL3.TCPAC == '1' then
11             UNDEFINED;
12         elseif HaveEL(EL3) && CPTR_EL3.TCPAC == '1' then
13             if Halted() && EDSCR.SDD == '1' then
14                 UNDEFINED;
15             else
16                 AArch64.SystemAccessTrap(EL3, 0x18);
17         else
18             CPTR_EL2 = X[t, 64];
19 elseif PSTATE.EL == EL3 then
20     CPTR_EL2 = X[t, 64];

```

MRS <Xt>, CPACR_EL1

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

```

1 if PSTATE.EL == EL0 then
2     UNDEFINED;

```

```

3  elsif PSTATE.EL == EL1 then
4      if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap priority
        ↳when SDD == '1'" && CPTR_EL3.TCPAC == '1' then
5          UNDEFINED;
6      elsif EL2Enabled() && CPTR_EL2.TCPAC == '1' then
7          AArch64.SystemAccessTrap(EL2, 0x18);
8      elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGTR_EL2.CPACR_EL1 == '1' then
9          AArch64.SystemAccessTrap(EL2, 0x18);
10     elsif HaveEL(EL3) && CPTR_EL3.TCPAC == '1' then
11         if Halted() && EDSCR.SDD == '1' then
12             UNDEFINED;
13         else
14             AArch64.SystemAccessTrap(EL3, 0x18);
15     elsif EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '111' then
16         X[t, 64] = NVMem[0x100];
17     else
18         X[t, 64] = CPACR_EL1;
19 elsif PSTATE.EL == EL2 then
20     if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap priority
        ↳when SDD == '1'" && CPTR_EL3.TCPAC == '1' then
21         UNDEFINED;
22     elsif HaveEL(EL3) && CPTR_EL3.TCPAC == '1' then
23         if Halted() && EDSCR.SDD == '1' then
24             UNDEFINED;
25         else
26             AArch64.SystemAccessTrap(EL3, 0x18);
27     elsif HCR_EL2.E2H == '1' then
28         X[t, 64] = CPTR_EL2;
29     else
30         X[t, 64] = CPACR_EL1;
31 elsif PSTATE.EL == EL3 then
32     X[t, 64] = CPACR_EL1;

```

MSR CPACR_EL1, <Xt>

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

```

1  if PSTATE.EL == EL0 then
2      UNDEFINED;
3  elsif PSTATE.EL == EL1 then
4      if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap priority
        ↳when SDD == '1'" && CPTR_EL3.TCPAC == '1' then
5          UNDEFINED;
6      elsif EL2Enabled() && CPTR_EL2.TCPAC == '1' then
7          AArch64.SystemAccessTrap(EL2, 0x18);
8      elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGWTR_EL2.CPACR_EL1 == '1' then
9          AArch64.SystemAccessTrap(EL2, 0x18);
10     elsif HaveEL(EL3) && CPTR_EL3.TCPAC == '1' then
11         if Halted() && EDSCR.SDD == '1' then
12             UNDEFINED;
13         else
14             AArch64.SystemAccessTrap(EL3, 0x18);
15     elsif EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '111' then
16         NVMem[0x100] = X[t, 64];
17     else
18         CPACR_EL1 = X[t, 64];
19 elsif PSTATE.EL == EL2 then
20     if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap priority
        ↳when SDD == '1'" && CPTR_EL3.TCPAC == '1' then
21         UNDEFINED;
22     elsif HaveEL(EL3) && CPTR_EL3.TCPAC == '1' then
23         if Halted() && EDSCR.SDD == '1' then
24             UNDEFINED;
25         else
26             AArch64.SystemAccessTrap(EL3, 0x18);
27     elsif HCR_EL2.E2H == '1' then
28         CPTR_EL2 = X[t, 64];
29     else
30         CPACR_EL1 = X[t, 64];
31 elsif PSTATE.EL == EL3 then
32     CPACR_EL1 = X[t, 64];

```


E3.2.3 CPTR_EL3, Architectural Feature Trap Register (EL3)

The CPTR_EL3 characteristics are:

Purpose

Controls trapping to EL3 of accesses to CPACR, [CPACR_EL1](#), HCPTR, [CPTR_EL2](#), trace, Activity Monitor, SME, Streaming SVE, SVE, and Advanced SIMD and floating-point functionality.

Attributes

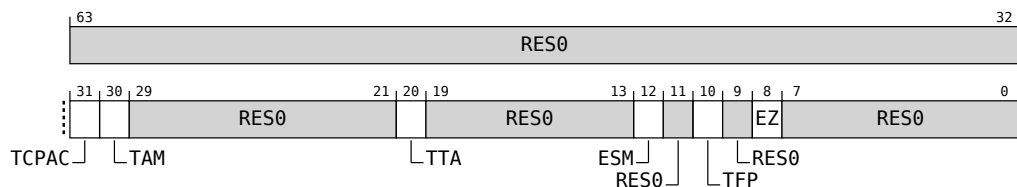
CPTR_EL3 is a 64-bit register.

Configuration

This register is present only when EL3 is implemented. Otherwise, direct accesses to CPTR_EL3 are UNDEFINED.

Field descriptions

The CPTR_EL3 bit assignments are:



Bits [63:32]

Reserved, RES0.

TCPAC, bit [31]

Traps all of the following to EL3, from both Execution states and any Security state.

- EL2 accesses to [CPTR_EL2](#), reported using ESR_ELx.EC value 0x18, or HCPTR, reported using ESR_ELx.EC value 0x03.
- EL2 and EL1 accesses to [CPACR_EL1](#) reported using ESR_ELx.EC value 0x18, or CPACR reported using ESR_ELx.EC value 0x03.

When CPTR_EL3.TCPAC is:

TCPAC	Meaning
0b0	This control does not cause any instructions to be trapped.
0b1	EL2 accesses to the CPTR_EL2 or HCPTR, and EL2 and EL1 accesses to the CPACR_EL1 or CPACR, are trapped to EL3, unless they are trapped by CPTR_EL2 .TCPAC.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

TAM, bit [30]

When FEAT_AMUv1 is implemented:

Trap Activity Monitor access. Traps EL2, EL1, and EL0 accesses to all Activity Monitor registers to EL3.

Accesses to the Activity Monitors registers are trapped as follows:

- In AArch64 state, the following registers are trapped to EL3 and reported with ESR_ELx.EC value 0x18:
 - AMUSERENR_EL0, AMCFGR_EL0, AMCGCR_EL0, AMCNTENCLR0_EL0, AMCNTENCLR1_EL0, AMCNTENSET0_EL0, AMCNTENSET1_EL0, AMCR_EL0, AMEVCNTR0<n>_EL0, AMEVCNTR1<n>_EL0, AMEVTYPER0<n>_EL0, and AMEVTYPER1<n>_EL0.
- In AArch32 state, accesses with MRC or MCR to the following registers reported with ESR_ELx.EC value 0x03:
 - AMUSERENR, AMCFGR, AMCGCR, AMCNTENCLR0, AMCNTENCLR1, AMCNTENSET0, AMCNTENSET1, AMCR, AMEVTYPER0<n>, and AMEVTYPER1<n>.
- In AArch32 state, accesses with MRRC or MCRR to the following registers, reported with ESR_ELx.EC value 0x04:
 - AMEVCNTR0<n>, AMEVCNTR1<n>.

TAM	Meaning
0b0	Accesses from EL2, EL1, and EL0 to Activity Monitor registers are not trapped.
0b1	Accesses from EL2, EL1, and EL0 to Activity Monitor registers are trapped to EL3.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

RES0

Bits [29:21]

Reserved, RES0.

TTA, bit [20]

Traps System register accesses. Accesses to the trace registers, from all Exception levels, any Security state, and both Execution states are trapped to EL3 as follows:

- In AArch64 state, Trace registers with op0=2, op1=1, and CRn<0b1000 are trapped to EL3 and reported using EC syndrome value 0x18.
- In AArch32 state, accesses using MCR or MRC to the Trace registers with cpnum=14, opc1=1, and CRn<0b1000 are reported using EC syndrome value 0x05.

TTA	Meaning
0b0	This control does not cause any instructions to be trapped.
0b1	Any System register access to the trace registers is trapped to EL3, unless it is trapped by CPACR.TRCDIS, CPACR_EL1.TTA, or CPTR_EL2.TTA.

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

The ETMv4 architecture and ETE do not permit EL0 to access the trace registers. If the PE trace unit implements FEAT_ETMv4 or FEAT_ETE, EL0 accesses to the trace registers are UNDEFINED, and any resulting exception is higher priority than this trap exception.

EL3 does not provide traps on trace register accesses through the Memory-mapped interface.

System register accesses to the trace registers can have side-effects. When a System register access is trapped, no side-effects occur before the exception is taken, see ‘Traps on instructions’.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [19:13]

Reserved, RES0.

ESM, bit [12]

When FEAT_SME is implemented:

Traps execution of SME instructions, SVE instructions when FEAT_SVE is not implemented or the PE is in Streaming SVE mode, and instructions that directly access the [SMCR_EL1](#), [SMCR_EL2](#), [SMCR_EL3](#), [SMPRI_EL1](#), [SMPRMAP_EL2](#), or [SVCR](#) System registers, from all Exception levels and any Security state, to EL3.

When instructions that directly access the [SVCR](#) System register are trapped with reference to this control, the [MSR_SVCRSM](#), [MSR_SVCRZA](#), and [MSR_SVCRSMZA](#) instructions are also trapped.

When direct accesses to [SMPRI_EL1](#) and [SMPRMAP_EL2](#) are trapped, the exception is reported using an ESR_EL3.EC value of 0x18. Otherwise, the exception is reported using an ESR_EL3.EC value of 0x1D, with an ISS code of 0x000000.

This field does not affect whether Streaming SVE or SME register values are valid.

A trap taken as a result of CPTR_EL3.ESM has precedence over a trap taken as a result of CPTR_EL3.TFP.

ESM	Meaning
0b0	This control causes execution of these instructions at all Exception levels to be trapped.
0b1	This control does not cause execution of any instructions to be trapped.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

RES0

Bit [11]

Reserved, RES0.

TFP, bit [10]

Traps execution of instructions which access the Advanced SIMD and floating-point functionality, from all

Exception levels, any Security state, and both Execution states, to EL3.

This includes the following registers, all reported using ESR_ELx.EC value 0x07:

- [FPCR](#), FPSR, FPEXC32_EL2, and any of the SIMD and floating-point registers V0-V31, including their views as D0-D31 registers or S0-S31 registers.
- MVFR0, MVFR1, MVFR2, FPSCR, FPEXC, and any of the SIMD and floating-point registers Q0-Q15, including their views as D0-D31 registers or S0-S31 registers.
- VMSR accesses to FPSID.

Permitted VMSR accesses to FPSID are ignored, but for the purposes of this trap the architecture defines a VMSR access to the FPSID from EL1 or higher as an access to a SIMD and floating-point register.

Traps execution at all Exception levels of SME and SVE instructions to EL3 from any Security state. The exception is reported using ESR_ELx.EC value 0x07.

A trap taken as a result of CPTR_EL3.ESM has precedence over a trap taken as a result of CPTR_EL3.TFP.

A trap taken as a result of CPTR_EL3.EZ has precedence over a trap taken as a result of CPTR_EL3.TFP.

Defined values are:

TFP	Meaning
0b0	This control does not cause execution of any instructions to be trapped.
0b1	This control causes execution of these instructions at all Exception levels to be trapped.

FPEXC32_EL2 is not accessible from EL0 using AArch64.

FPSID, MVFR0, MVFR1, and FPEXC are not accessible from EL0 using AArch32.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bit [9]

Reserved, RES0.

EZ, bit [8]

When FEAT_SVE is implemented:

Traps execution of SVE instructions when the PE is not in Streaming SVE mode, and instructions that directly access the ZCR_EL3, ZCR_EL2, or ZCR_EL1 System registers, from all Exception levels and any Security state, to EL3.

The exception is reported using ESR_ELx.EC value 0x19.

A trap taken as a result of CPTR_EL3.EZ has precedence over a trap taken as a result of CPTR_EL3.TFP.

EZ	Meaning
0b0	This control causes execution of these instructions at all Exception levels to be trapped.
0b1	This control does not cause execution of any instructions to be trapped.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

RES0

Bits [7:0]

Reserved, RES0.

Accessing the CPTR_EL3

Accesses to this register use the following encodings in the instruction encoding space:

MRS <Xt>, CPTR_EL3

op0	op1	CRn	CRm	op2
0b11	0b110	0b0001	0b0001	0b010

```

1 if PSTATE.EL == EL0 then
2     UNDEFINED;
3 elsif PSTATE.EL == EL1 then
4     UNDEFINED;
5 elsif PSTATE.EL == EL2 then
6     UNDEFINED;
7 elsif PSTATE.EL == EL3 then
8     X[t, 64] = CPTR_EL3;
```

MSR CPTR_EL3, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b110	0b0001	0b0001	0b010

```

1 if PSTATE.EL == EL0 then
2     UNDEFINED;
3 elsif PSTATE.EL == EL1 then
4     UNDEFINED;
5 elsif PSTATE.EL == EL2 then
6     UNDEFINED;
7 elsif PSTATE.EL == EL3 then
8     CPTR_EL3 = X[t, 64];
```

E3.2.4 FAR_EL1, Fault Address Register (EL1)

The FAR_EL1 characteristics are:

Purpose

Holds the faulting Virtual Address for all synchronous Instruction or Data Abort, PC alignment fault and Watchpoint exceptions that are taken to EL1.

Attributes

FAR_EL1 is a 64-bit register.

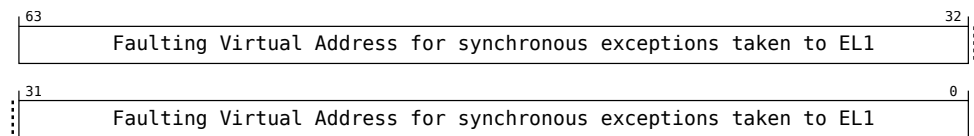
Configuration

AArch64 system register FAR_EL1 bits [31:0] are architecturally mapped to AArch32 system register DFAR31:0.

AArch64 system register FAR_EL1 bits [63:32] are architecturally mapped to AArch32 system register IFAR31:0.

Field descriptions

The FAR_EL1 bit assignments are:



Bits [63:0]

Faulting Virtual Address for synchronous exceptions taken to EL1. Exceptions that set the FAR_EL1 are Instruction Aborts (EC 0x20 or 0x21), Data Aborts (EC 0x24 or 0x25), PC alignment faults (EC 0x22), and Watchpoints (EC 0x34 or 0x35). ESR_EL1.EC holds the EC value for the exception.

For a synchronous External abort, if the VA that generated the abort was from an address range for which TCR_ELx.TBI{<0|1>} == 1 for the translation regime in use when the abort was generated, then the top eight bits of FAR_EL1 are UNKNOWN.

For a synchronous External abort other than a synchronous External abort on a translation table walk, this field is valid only if ESR_EL1.FnV is 0, and FAR_EL1 is UNKNOWN if ESR_EL1.FnV is 1.

If a memory fault that sets FAR_EL1, other than a Tag Check Fault, is generated from a data cache maintenance or other DC instruction, this field holds the address specified in the register argument of the instruction.

On an exception due to a Tag Check Fault caused by a data cache maintenance or other DC instruction, the address held in FAR_EL1 is IMPLEMENTATION DEFINED as one of the following:

- The lowest address that gave rise to the fault.
- The address specified in the register argument of the instruction as generated by MMU faults caused by DC ZVA.

If the exception that updates FAR_EL1 is taken from an Exception level using AArch32, the top 32 bits are all zero, unless both of the following apply, in which case the top 32 bits of FAR_ELx are 0x00000001:

- The faulting address was generated by a load or store instruction that sequentially incremented from address 0xFFFFFFFF. Such a load or store instruction is CONSTRAINED UNPREDICTABLE.
- The implementation treats such incrementing as setting bit[32] of the virtual address to 1.

When FEAT_SME is implemented, and the PE sets ESR_EL1.ISV to 0 and ESR_EL1.FnP to 1 on taking a Data Abort exception or Watchpoint exception, the PE sets FAR_EL1 to any address within the naturally-aligned fault granule that contains the virtual address of the memory access that generated the Data Abort or Watchpoint exception.

The naturally-aligned fault granule is one of:

- When ESR_EL1.DFSC is 0b010001, indicating a Synchronous Tag Check fault, it is a 16-byte tag granule.
- When ESR_EL1.DFSC is 0b11010x, indicating an IMPLEMENTATION DEFINED fault, it is an IMPLEMENTATION DEFINED granule.
- Otherwise, it is the smallest implemented translation granule.

When FEAT_MOPS is implemented, the value in FAR_EL1 on a synchronous exception from any of the Memory Copy and Memory Set instructions represents the first element that has not been copied or set, and is determined as follows:

- For a Data Abort generated by the MMU, the value is within the address range of the current translation granule, aligned to the size of the current translation granule of the address that generated the Data Abort. Bits [n-1:0] of the value are UNKNOWN, where 2^n is the translation granule size in bytes.
- For a Data Abort generated by a Tag Check failure, the value is the lowest address that failed the Tag Check within the block size of the load or store.
- For a Watchpoint exception, the value is an address range of the size defined by the DCZID_EL0.BS field. This address does not need to be the element with a watchpoint, but can be some earlier element.
- Otherwise, the value is the lowest address in the block size of the load or store.

For a Data Abort or Watchpoint exception, if address tagging is enabled for the address accessed by the data access that caused the exception, then this field includes the tag. For more information about address tagging, see ‘Address tagging in AArch64 state’.

For a synchronous Tag Check Fault abort, bits[63:60] are UNKNOWN.

Execution at EL0 makes FAR_EL1 become UNKNOWN.

The address held in this field is an address accessed by the instruction fetch or data access that caused the exception that actually gave rise to the instruction or data abort. It is the lower address that gave rise to the fault. Where different faults from different addresses arise from the same instruction, such as for an instruction that loads or stores an unaligned address that crosses a page boundary, the architecture does not prioritize between those different faults.

For all other exceptions taken to EL1, FAR_EL1 is UNKNOWN.

FAR_EL1 is made UNKNOWN on an exception return from EL1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing the FAR_EL1

When HCR_EL2.E2H is 1, without explicit synchronization, access from EL3 using the mnemonic FAR_EL1 or FAR_EL12 are not guaranteed to be ordered with respect to accesses using the other mnemonic.

Accesses to this register use the following encodings in the instruction encoding space:

MRS <Xt>, FAR_EL1

op0	op1	CRn	CRm	op2
0b11	0b000	0b0110	0b0000	0b000

```

1 if PSTATE.EL == EL0 then
2     UNDEFINED;
3 elseif PSTATE.EL == EL1 then
4     if EL2Enabled() && HCR_EL2.TVM == '1' then
5         AArch64.SystemAccessTrap(EL2, 0x18);
6     elseif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGTR_EL2.FAR_EL1 == '1' then
7         AArch64.SystemAccessTrap(EL2, 0x18);
8     elseif EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '111' then
9         X[t, 64] = NVMem[0x220];
10    else
11        X[t, 64] = FAR_EL1;
12 elseif PSTATE.EL == EL2 then
13     if HCR_EL2.E2H == '1' then
14         X[t, 64] = FAR_EL2;
15     else
16         X[t, 64] = FAR_EL1;
17 elseif PSTATE.EL == EL3 then
18     X[t, 64] = FAR_EL1;

```

MSR FAR_EL1, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b000	0b0110	0b0000	0b000

```

1 if PSTATE.EL == EL0 then
2     UNDEFINED;
3 elseif PSTATE.EL == EL1 then
4     if EL2Enabled() && HCR_EL2.TVM == '1' then
5         AArch64.SystemAccessTrap(EL2, 0x18);
6     elseif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGWTR_EL2.FAR_EL1 == '1' then
7         AArch64.SystemAccessTrap(EL2, 0x18);
8     elseif EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '111' then
9         NVMem[0x220] = X[t, 64];
10    else
11        FAR_EL1 = X[t, 64];
12 elseif PSTATE.EL == EL2 then
13     if HCR_EL2.E2H == '1' then
14         FAR_EL2 = X[t, 64];
15     else
16        FAR_EL1 = X[t, 64];
17 elseif PSTATE.EL == EL3 then
18     FAR_EL1 = X[t, 64];

```

MRS <Xt>, FAR_EL12

op0	op1	CRn	CRm	op2
0b11	0b101	0b0110	0b0000	0b000

```

1 if PSTATE.EL == EL0 then
2     UNDEFINED;
3 elseif PSTATE.EL == EL1 then
4     if EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '101' then
5         X[t, 64] = NVMem[0x220];
6     elseif EL2Enabled() && HCR_EL2.NV == '1' then
7         AArch64.SystemAccessTrap(EL2, 0x18);
8     else
9         UNDEFINED;
10 elseif PSTATE.EL == EL2 then
11     if HCR_EL2.E2H == '1' then
12         X[t, 64] = FAR_EL1;
13     else
14         UNDEFINED;

```


Chapter E3. System registers affected by SME
E3.2. Changes to existing System registers

```

15  elsif PSTATE.EL == EL3 then
16      if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '1' then
17          X[t, 64] = FAR_EL1;
18      else
19          UNDEFINED;

```

MSR FAR_EL12, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b101	0b0110	0b0000	0b000

```

1  if PSTATE.EL == EL0 then
2      UNDEFINED;
3  elsif PSTATE.EL == EL1 then
4      if EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '101' then
5          NVMem[0x220] = X[t, 64];
6      elsif EL2Enabled() && HCR_EL2.NV == '1' then
7          AArch64.SystemAccessTrap(EL2, 0x18);
8      else
9          UNDEFINED;
10 elsif PSTATE.EL == EL2 then
11     if HCR_EL2.E2H == '1' then
12         FAR_EL1 = X[t, 64];
13     else
14         UNDEFINED;
15 elsif PSTATE.EL == EL3 then
16     if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '1' then
17         FAR_EL1 = X[t, 64];
18     else
19         UNDEFINED;

```

MRS <Xt>, FAR_EL2

op0	op1	CRn	CRm	op2
0b11	0b100	0b0110	0b0000	0b000

```

1  if PSTATE.EL == EL0 then
2      UNDEFINED;
3  elsif PSTATE.EL == EL1 then
4      if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
5          X[t, 64] = FAR_EL1;
6      elsif EL2Enabled() && HCR_EL2.NV == '1' then
7          AArch64.SystemAccessTrap(EL2, 0x18);
8      else
9          UNDEFINED;
10 elsif PSTATE.EL == EL2 then
11     X[t, 64] = FAR_EL2;
12 elsif PSTATE.EL == EL3 then
13     X[t, 64] = FAR_EL2;

```

MSR FAR_EL2, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b100	0b0110	0b0000	0b000

```
1  if PSTATE.EL == EL0 then
2      UNDEFINED;
3  elsif PSTATE.EL == EL1 then
4      if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
5          FAR_EL1 = X[t, 64];
6      elsif EL2Enabled() && HCR_EL2.NV == '1' then
7          AArch64.SystemAccessTrap(EL2, 0x18);
8      else
9          UNDEFINED;
10 elsif PSTATE.EL == EL2 then
11     FAR_EL2 = X[t, 64];
12 elsif PSTATE.EL == EL3 then
13     FAR_EL2 = X[t, 64];
```

E3.2.5 FAR_EL2, Fault Address Register (EL2)

The FAR_EL2 characteristics are:

Purpose

Holds the faulting Virtual Address for all synchronous Instruction or Data Abort, PC alignment fault and Watchpoint exceptions that are taken to EL2.

Attributes

FAR_EL2 is a 64-bit register.

Configuration

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

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

AArch64 system register FAR_EL2 bits [31:0] are architecturally mapped to AArch32 system register HDFAR[31:0].

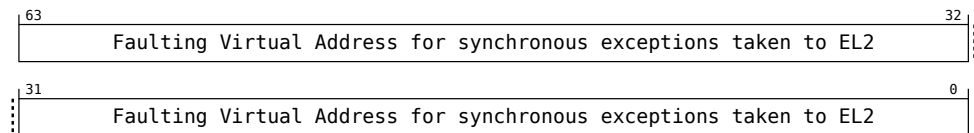
AArch64 system register FAR_EL2 bits [63:32] are architecturally mapped to AArch32 system register HIFAR[31:0].

When EL2 is implemented, AArch64 system register FAR_EL2 bits [31:0] are architecturally mapped to AArch32 system register DFAR31:0.

When EL2 is implemented, AArch64 system register FAR_EL2 bits [63:32] are architecturally mapped to AArch32 system register IFAR31:0.

Field descriptions

The FAR_EL2 bit assignments are:



Bits [63:0]

Faulting Virtual Address for synchronous exceptions taken to EL2. Exceptions that set the FAR_EL2 are Instruction Aborts (EC 0x20 or 0x21), Data Aborts (EC 0x24 or 0x25), PC alignment faults (EC 0x22), and Watchpoints (EC 0x34 or 0x35). ESR_EL2.EC holds the EC value for the exception.

For a synchronous External abort, if the VA that generated the abort was from an address range for which TCR_ELx.TBI{<0|1>} == 1 for the translation regime in use when the abort was generated, then the top eight bits of FAR_EL2 are UNKNOWN.

For a synchronous External abort other than a synchronous External abort on a translation table walk, this field is valid only if ESR_EL2.FnV is 0, and FAR_EL2 is UNKNOWN if ESR_EL2.FnV is 1.

If a memory fault that sets FAR_EL2, other than a Tag Check Fault, is generated from a data cache maintenance or other DC instruction, this field holds the address specified in the register argument of the instruction.

On an exception due to a Tag Check Fault caused by a data cache maintenance or other DC instruction, the address held in FAR_EL2 is IMPLEMENTATION DEFINED as one of the following:

- The lowest address that gave rise to the fault.
- The address specified in the register argument of the instruction as generated by MMU faults caused by DC ZVA.

If the exception that updates FAR_EL2 is taken from an Exception level using AArch32, the top 32 bits are all zero, unless both of the following apply, in which case the top 32 bits of FAR_ELx are 0x00000001:

- The faulting address was generated by a load or store instruction that sequentially incremented from address 0xFFFFFFFF. Such a load or store instruction is CONSTRAINED UNPREDICTABLE.
- The implementation treats such incrementing as setting bit[32] of the virtual address to 1.

When FEAT_SME is implemented, and the PE sets ESR_EL2.ISV to 0 and ESR_EL2.FnP to 1 on taking a Data Abort exception or Watchpoint exception, the PE sets FAR_EL2 to any address within the naturally-aligned fault granule that contains the virtual address of the memory access that generated the Data Abort or Watchpoint exception.

The naturally-aligned fault granule is one of:

- When ESR_EL2.DFSC is 0b010001, indicating a Synchronous Tag Check fault, it is a 16-byte tag granule.
- When ESR_EL2.DFSC is 0b11010x, indicating an IMPLEMENTATION DEFINED fault, it is an IMPLEMENTATION DEFINED granule.
- Otherwise, it is the smallest implemented translation granule.

When FEAT_MOPS is implemented, the value in FAR_EL2 on a synchronous exception from any of the Memory Copy and Memory Set instructions represents the first element that has not been copied or set, and is determined as follows:

- For a Data Abort generated by the MMU, the value is within the address range of the current translation granule, aligned to the size of the current translation granule of the address that generated the Data Abort. Bits [n-1:0] of the value are UNKNOWN, where 2^n is the translation granule size in bytes.
- For a Data Abort generated by a Tag Check failure, the value is the lowest address that failed the Tag Check within the block size of the load or store.
- For a Watchpoint exception, the value is an address range of the size defined by the DCZID_EL0.BS field. This address does not need to be the element with a watchpoint, but can be some earlier element.
- Otherwise, the value is the lowest address in the block size of the load or store.

For a Data Abort or Watchpoint exception, if address tagging is enabled for the address accessed by the data access that caused the exception, then this field includes the tag. For more information about address tagging, see ‘Address tagging in AArch64 state’.

For a synchronous Tag Check Fault abort, bits[63:60] are UNKNOWN.

Execution at EL1 or EL0 makes FAR_EL2 become UNKNOWN.

The address held in this field is an address accessed by the instruction fetch or data access that caused the exception that actually gave rise to the instruction or data abort. It is the lower address that gave rise to the fault. Where different faults from different addresses arise from the same instruction, such as for an instruction that loads or stores an unaligned address that crosses a page boundary, the architecture does not prioritize between those different faults.

For all other exceptions taken to EL2, FAR_EL2 is UNKNOWN.

FAR_EL2 is made UNKNOWN on an exception return from EL2.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing the FAR_EL2

When HCR_EL2.E2H is 1, without explicit synchronization, access from EL2 using the mnemonic FAR_EL2 or FAR_EL1 are not guaranteed to be ordered with respect to accesses using the other mnemonic.

Accesses to this register use the following encodings in the instruction encoding space:

MRS <Xt>, FAR_EL2

op0	op1	CRn	CRm	op2
0b11	0b100	0b0110	0b0000	0b000

```

1 if PSTATE.EL == EL0 then
2     UNDEFINED;
3 elsif PSTATE.EL == EL1 then
4     if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
5         X[t, 64] = FAR_EL1;
6     elsif EL2Enabled() && HCR_EL2.NV == '1' then
7         AArch64.SystemAccessTrap(EL2, 0x18);
8     else
9         UNDEFINED;
10 elsif PSTATE.EL == EL2 then
11     X[t, 64] = FAR_EL2;
12 elsif PSTATE.EL == EL3 then
13     X[t, 64] = FAR_EL2;
```

MSR FAR_EL2, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b100	0b0110	0b0000	0b000

```

1 if PSTATE.EL == EL0 then
2     UNDEFINED;
3 elsif PSTATE.EL == EL1 then
4     if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
5         FAR_EL1 = X[t, 64];
6     elsif EL2Enabled() && HCR_EL2.NV == '1' then
7         AArch64.SystemAccessTrap(EL2, 0x18);
8     else
9         UNDEFINED;
10 elsif PSTATE.EL == EL2 then
11     FAR_EL2 = X[t, 64];
12 elsif PSTATE.EL == EL3 then
13     FAR_EL2 = X[t, 64];
```

MRS <Xt>, FAR_EL1

op0	op1	CRn	CRm	op2
0b11	0b000	0b0110	0b0000	0b000

```

1 if PSTATE.EL == EL0 then
2     UNDEFINED;
3 elsif PSTATE.EL == EL1 then
4     if EL2Enabled() && HCR_EL2.TVM == '1' then
5         AArch64.SystemAccessTrap(EL2, 0x18);
6     elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGTR_EL2.FAR_EL1 == '1' then
7         AArch64.SystemAccessTrap(EL2, 0x18);
8     elsif EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '111' then
9         X[t, 64] = NVMem[0x220];
10    else
```

```

11      X[t, 64] = FAR_EL1;
12  elseif PSTATE.EL == EL2 then
13      if HCR_EL2.E2H == '1' then
14          X[t, 64] = FAR_EL2;
15      else
16          X[t, 64] = FAR_EL1;
17  elseif PSTATE.EL == EL3 then
18      X[t, 64] = FAR_EL1;

```

MSR FAR_EL1, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b000	0b0110	0b0000	0b000

```

1  if PSTATE.EL == EL0 then
2      UNDEFINED;
3  elseif PSTATE.EL == EL1 then
4      if EL2Enabled() && HCR_EL2.TVM == '1' then
5          AArch64.SystemAccessTrap(EL2, 0x18);
6      elseif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGWTR_EL2.FAR_EL1 == '1' then
7          AArch64.SystemAccessTrap(EL2, 0x18);
8      elseif EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '111' then
9          NVMem[0x220] = X[t, 64];
10     else
11         FAR_EL1 = X[t, 64];
12  elseif PSTATE.EL == EL2 then
13      if HCR_EL2.E2H == '1' then
14          FAR_EL2 = X[t, 64];
15      else
16          FAR_EL1 = X[t, 64];
17  elseif PSTATE.EL == EL3 then
18      FAR_EL1 = X[t, 64];

```

E3.2.6 FAR_EL3, Fault Address Register (EL3)

The FAR_EL3 characteristics are:

Purpose

Holds the faulting Virtual Address for all synchronous Instruction or Data Abort and PC alignment fault exceptions that are taken to EL3.

Attributes

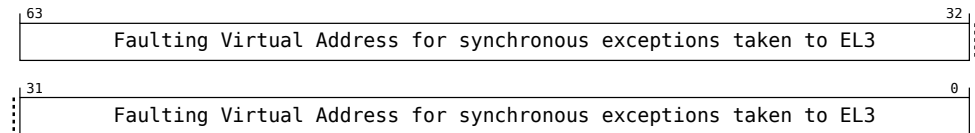
FAR_EL3 is a 64-bit register.

Configuration

This register is present only when EL3 is implemented. Otherwise, direct accesses to FAR_EL3 are UNDEFINED.

Field descriptions

The FAR_EL3 bit assignments are:



Bits [63:0]

Faulting Virtual Address for synchronous exceptions taken to EL3. Exceptions that set the FAR_EL3 are Instruction Aborts (EC 0x20 or 0x21), Data Aborts (EC 0x24 or 0x25), and PC alignment faults (EC 0x22). ESR_EL3.EC holds the EC value for the exception.

For a synchronous External abort, if the VA that generated the abort was from an address range for which TCR_ELx.TBI{<0|1>} == 1 for the translation regime in use when the abort was generated, then the top eight bits of FAR_EL3 are UNKNOWN.

For a synchronous External abort other than a synchronous External abort on a translation table walk, this field is valid only if ESR_EL3.FnV is 0, and FAR_EL3 is UNKNOWN if ESR_EL3.FnV is 1.

If a memory fault that sets FAR_EL3, other than a Tag Check Fault, is generated from a data cache maintenance or other DC instruction, this field holds the address specified in the register argument of the instruction.

On an exception due to a Tag Check Fault caused by a data cache maintenance or other DC instruction, the address held in FAR_EL3 is IMPLEMENTATION DEFINED as one of the following:

- The lowest address that gave rise to the fault.
- The address specified in the register argument of the instruction as generated by MMU faults caused by DC ZVA.

If the exception that updates FAR_EL3 is taken from an Exception level using AArch32, the top 32 bits are all zero, unless both of the following apply, in which case the top 32 bits of FAR_ELx are 0x00000001:

- The faulting address was generated by a load or store instruction that sequentially incremented from address 0xFFFFFFFF. Such a load or store instruction is CONSTRAINED UNPREDICTABLE.
- The implementation treats such incrementing as setting bit[32] of the virtual address to 1.

When FEAT_SME is implemented, and the PE sets ESR_EL3.ISV to 0 and ESR_EL3.FnP to 1 on taking a Data Abort exception, the PE sets FAR_EL3 to any address within the naturally-aligned fault granule that contains the virtual address of the memory access that generated the Data Abort.

The naturally-aligned fault granule is one of:

- When ESR_EL3.DFSC is 0b010001, indicating a Synchronous Tag Check fault, it is a 16-byte tag granule.
- When ESR_EL3.DFSC is 0b11010x, indicating an IMPLEMENTATION DEFINED fault, it is an IMPLEMENTATION DEFINED granule.
- Otherwise, it is the smallest implemented translation granule.

When FEAT_MOPS is implemented, the value in FAR_EL3 on a synchronous exception from any of the Memory Copy and Memory Set instructions represents the first element that has not been copied or set, and is determined as follows:

- For a Data Abort generated by the MMU, the value is within the address range of the current translation granule, aligned to the size of the current translation granule of the address that generated the Data Abort. Bits [n-1:0] of the value are UNKNOWN, where 2^n is the translation granule size in bytes.
- For a Data Abort generated by a Tag Check failure, the value is the lowest address that failed the Tag Check within the block size of the load or store.
- Otherwise, the value is the lowest address in the block size of the load or store.

For a Data Abort exception, if address tagging is enabled for the address accessed by the data access that caused the exception, then this field includes the tag. For more information about address tagging, see ‘Address tagging in AArch64 state’.

For a synchronous Tag Check Fault abort, bits[63:60] are UNKNOWN.

Execution at EL2, EL1, or EL0 makes FAR_EL3 become UNKNOWN.

The address held in this register is an address accessed by the instruction fetch or data access that caused the exception that actually gave rise to the instruction or data abort. It is the lowest address that gave rise to the fault. Where different faults from different addresses arise from the same instruction, such as for an instruction that loads or stores an unaligned address that crosses a page boundary, the architecture does not prioritize between those different faults.

For all other exceptions taken to EL3, FAR_EL3 is UNKNOWN.

FAR_EL3 is made UNKNOWN on an exception return from EL3.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing the FAR_EL3

Accesses to this register use the following encodings in the instruction encoding space:

MRS <Xt>, FAR_EL3

op0	op1	CRn	CRm	op2
0b11	0b110	0b0110	0b0000	0b000

```

1 if PSTATE.EL == EL0 then
2     UNDEFINED;
3 elsif PSTATE.EL == EL1 then
4     UNDEFINED;
5 elsif PSTATE.EL == EL2 then
6     UNDEFINED;
7 elsif PSTATE.EL == EL3 then
8     X[t, 64] = FAR_EL3;
```

MSR FAR_EL3, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b110	0b0110	0b0000	0b000

```

1  if PSTATE.EL == EL0 then
2      UNDEFINED;
3  elsif PSTATE.EL == EL1 then
4      UNDEFINED;
5  elsif PSTATE.EL == EL2 then
6      UNDEFINED;
7  elsif PSTATE.EL == EL3 then
8      FAR_EL3 = X[t, 64];

```

E3.2.7 FPCR, Floating-point Control Register

The FPCR characteristics are:

Purpose

Controls floating-point behavior.

Attributes

FPCR is a 64-bit register.

Configuration

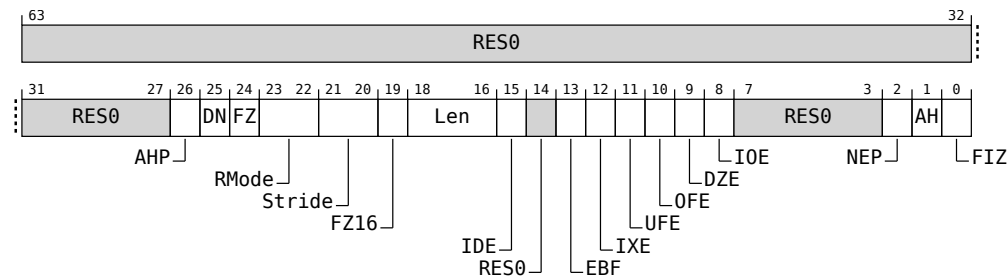
It is IMPLEMENTATION DEFINED whether the Len and Stride fields can be programmed to non-zero values, which will cause some AArch32 floating-point instruction encodings to be UNDEFINED, or whether these fields are RAZ.

AArch64 system register FPCR bits [26:15] are architecturally mapped to AArch32 system register FPSCR[26:15].

AArch64 system register FPCR bits [12:8] are architecturally mapped to AArch32 system register FPSCR[12:8].

Field descriptions

The FPCR bit assignments are:



Bits [63:27]

Reserved, RES0.

AHP, bit [26]

Alternative half-precision control bit.

AHP	Meaning
0b0	IEEE half-precision format selected.
0b1	Alternative half-precision format selected.

This bit is used only for conversions between half-precision floating-point and other floating-point formats.

The data-processing instructions added as part of the FEAT_FP16 extension always use the IEEE half-precision format, and ignore the value of this bit.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

DN, bit [25]

Default NaN use for NaN propagation.

DN	Meaning
0b0	NaN operands propagate through to the output of a floating-point operation.
0b1	Any operation involving one or more NaNs returns the Default NaN. This bit has no effect on the output of FABS, FMAX*, FMIN*, and FNEG instructions, and a default NaN is never returned as a result of these instructions.

The value of this bit controls both scalar and Advanced SIMD floating-point arithmetic.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

FZ, bit [24]

Flushing denormalized numbers to zero control bit.

FZ	Meaning
0b0	If FPCR.AH is 0, the flushing to zero of single-precision and double-precision denormalized inputs to, and outputs of, floating-point instructions not enabled by this control, but other factors might cause the input denormalized numbers to be flushed to zero. If FPCR.AH is 1, the flushing to zero of single-precision and double-precision denormalized outputs of floating-point instructions not enabled by this control, but other factors might cause the input denormalized numbers to be flushed to zero.
0b1	If FPCR.AH is 0, denormalized single-precision and double-precision inputs to, and outputs from, floating-point instructions are flushed to zero. If FPCR.AH is 1, denormalized single-precision and double-precision outputs from floating-point instructions are flushed to zero.

For more information, see ‘Flushing denormalized numbers to zero’ and the pseudocode of the floating-point instructions.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

RMode, bits [23:22]

Rounding Mode control field.

RMode	Meaning
0b00	Round to Nearest (RN) mode.
0b01	Round towards Plus Infinity (RP) mode.
0b10	Round towards Minus Infinity (RM) mode.
0b11	Round towards Zero (RZ) mode.

The specified rounding mode is used by both scalar and Advanced SIMD floating-point instructions.

If FPCR.AH is 1, then the following instructions use Round to Nearest mode regardless of the value of this bit:

- The FRECPE, FRECPs, FRECPX, FRSQRTE, and FRSQRTS instructions.
- The BFCVT, BFCVTN, BFCVTN2, BFCVTNT, BFMLALB, and BFMLALT instructions.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Stride, bits [21:20]

This field has no function in AArch64 state, and non-zero values are ignored during execution in AArch64 state.

This field is included only for context saving and restoration of the AArch32 FPSCR.Stride field.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

FZ16, bit [19]

When FEAT_FP16 is implemented:

Flushing denormalized numbers to zero control bit on half-precision data-processing instructions.

FZ16	Meaning
0b0	For some instructions, this bit disables flushing to zero of inputs and outputs that are half-precision denormalized numbers.
0b1	Flushing denormalized numbers to zero enabled. For some instructions that do not convert a half-precision input to a higher precision output, this bit enables flushing to zero of inputs and outputs that are half-precision denormalized numbers.

The value of this bit applies to both scalar and Advanced SIMD floating-point half-precision calculations.

For more information, see ‘Flushing denormalized numbers to zero’ and the pseudocode of the floating-point instructions.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

RES0

Len, bits [18:16]

This field has no function in AArch64 state, and non-zero values are ignored during execution in AArch64 state.

This field is included only for context saving and restoration of the AArch32 FPSR.Len field.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

IDE, bit [15]

Input Denormal floating-point exception trap enable.

IDE	Meaning
0b0	Untrapped exception handling selected. If the floating-point exception occurs, the FPSR.IDC bit is set to 1.
0b1	Trapped exception handling selected. If the floating-point exception occurs, the PE does not update the FPSR.IDC bit.

The value of this bit controls both scalar and Advanced SIMD floating-point arithmetic.

If the implementation does not support this exception, this bit is RAZ/WI.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bit [14]

Reserved, RES0.

EBF, bit [13]

When FEAT_EBF16 is implemented:

The value of this bit controls the numeric behaviors of BFloat16 dot product calculations performed by the BFDOT, BFMMMLA, BFMOPLA, and BFMOPLS instructions.

When [ID_AA64ISAR1_EL1.BF16](#) and [ID_AA64ZFR0_EL1.BF16](#) are 0b0010, the PE supports the FPCR.EBF field. Otherwise, FPCR.EBF is RES0.

EBF	Meaning
0b0	These instructions use the standard BFloat16 behaviors: <ul style="list-style-type: none"> • Ignoring the FPCR.RMode control and using the rounding mode defined for BFloat16. For more information, see ‘Round to Odd mode’. • Flushing denormalized inputs and outputs to zero, as if the FPCR.FZ and FPCR.FIZ controls had the value ‘1’. • Performing unfused multiplies and additions with intermediate rounding of all products and sums.

EBF	Meaning
0b1	<p>These instructions use the extended BFloat16 behaviors:</p> <ul style="list-style-type: none"> Supporting all four IEEE 754 rounding modes selected by the FPCR.RMode control. Optionally, flushing denormalized inputs and outputs to zero, as governed by the FPCR.FZ and FPCR.FIZ controls. Performing a fused two-way sum-of-products for each pair of adjacent BFloat16 elements, without intermediate rounding of the products, but rounding the single-precision sum before addition to the accumulator. Generating the default NaN as intermediate sum-of-products when any multiplier input is a NaN, or any product is infinity \times 0.0, or there are infinite products with differing signs. Generating an intermediate sum-of-products of the same infinity when there are infinite products all with the same sign.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

RES0

IXE, bit [12]

Inexact floating-point exception trap enable.

IXE	Meaning
0b0	Untrapped exception handling selected. If the floating-point exception occurs, the FPSR.IXC bit is set to 1.
0b1	Trapped exception handling selected. If the floating-point exception occurs, the PE does not update the FPSR.IXC bit.

The value of this bit controls both scalar and Advanced SIMD floating-point arithmetic.

If the implementation does not support this exception, this bit is RAZ/WI.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

UFE, bit [11]

Underflow floating-point exception trap enable.

UFE	Meaning
0b0	Untrapped exception handling selected. If the floating-point exception occurs, the FPSR.UFC bit is set to 1.
0b1	Trapped exception handling selected. If the floating-point exception occurs and Flush-to-zero is not enabled, the PE does not update the FPSR.UFC bit.

The value of this bit controls both scalar and Advanced SIMD floating-point arithmetic.

If the implementation does not support this exception, this bit is RAZ/WI.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

OFE, bit [10]

Overflow floating-point exception trap enable.

OFE	Meaning
0b0	Untrapped exception handling selected. If the floating-point exception occurs, the FPSR.OFC bit is set to 1.
0b1	Trapped exception handling selected. If the floating-point exception occurs, the PE does not update the FPSR.OFC bit.

The value of this bit controls both scalar and Advanced SIMD floating-point arithmetic.

If the implementation does not support this exception, this bit is RAZ/WI.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

DZE, bit [9]

Divide by Zero floating-point exception trap enable.

DZE	Meaning
0b0	Untrapped exception handling selected. If the floating-point exception occurs, the FPSR.DZC bit is set to 1.
0b1	Trapped exception handling selected. If the floating-point exception occurs, the PE does not update the FPSR.DZC bit.

The value of this bit controls both scalar and Advanced SIMD floating-point arithmetic.

If the implementation does not support this exception, this bit is RAZ/WI.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

IOE, bit [8]

Invalid Operation floating-point exception trap enable.

IOE	Meaning
0b0	Untrapped exception handling selected. If the floating-point exception occurs, the FPSR.IOC bit is set to 1.
0b1	Trapped exception handling selected. If the floating-point exception occurs, the PE does not update the FPSR.IOC bit.

The value of this bit controls both scalar and Advanced SIMD floating-point arithmetic.

If the implementation does not support this exception, this bit is RAZ/WI.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [7:3]

Reserved, RES0.

NEP, bit [2]

When FEAT_AFP is implemented:

Controls how the output elements other than the lowest element of the vector are determined for Advanced SIMD scalar instructions.

NEP	Meaning
0b0	Does not affect how the output elements other than the lowest are determined for Advanced SIMD scalar instructions.

NEP	Meaning
0b1	<p>The output elements other than the lowest are taken from the following registers:</p> <ul style="list-style-type: none"> • For 3-input scalar versions of the FMLA (by element) and FMLS (by element) instructions, the <Hd>, <Sd>, or <Dd> register. • For 3-input versions of the FMADD, FMSUB, FNMADD, and FNMSUB instructions, the <Ha>, <Sa>, or <Da> register. • For 2-input scalar versions of the FACGE, FACGT, FCMEQ (register), FCMGE (register), and FCMGT (register) instructions, the <Hm>, <Sm>, or <Dm> register. • For 2-input scalar versions of the FABD, FADD (scalar), FDIV (scalar), FMAX (scalar), FMAXNM (scalar), FMIN (scalar), FMINNM (scalar), FMUL (by element), FMUL (scalar), FMULX (by element), FMULX, FNMUL (scalar), FRECPs, FRSQRTs, and FSUB (scalar) instructions, the <Hn>, <Sn>, or <Dn> register. • For 1-input scalar versions of the following instructions, the <Hd>, <Sd>, or <Dd> register: <ul style="list-style-type: none"> – The (vector) versions of the FCVTAS, FCVTAU, FCVTMS, FCVTMU, FCVTNS, FCVTNU, FCVTPS, and FCVTPU instructions. – The (vector, fixed-point) and (vector, integer) versions of the FCVTZS, FCVTZU, SCVTF, and UCVTF instructions. – The (scalar) versions of the FABS, FNEG, FRINT32X, FRINT32Z, FRINT64X, FRINT64Z, FRINTA, FRINTI, FRINTM, FRINTN, FRINTP, FRINTX, FRINTZ, and FSQRT instructions. – The (scalar, fixed-point) and (scalar, integer) versions of the SCVTF and UCVTF instructions. – The BFCVT, FCVT, FCVTXN, FRECPe, FRECPX, and FRSQRTe instructions.

The value of FPCR.NEP is treated as 0 for all purposes other than a direct read or write of the FPCR when the PE is in Streaming SVE mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

RES0

AH, bit [1]

When FEAT AFP is implemented:

Alternate Handling. Controls alternate handling of floating-point numbers.

The Arm architecture supports two models for handling some of the corner cases of the floating-point behaviors, such as the nature of flushing of denormalized numbers, the detection of tininess and other exceptions and a range of other behaviors. The value of the FPCR.AH bit selects between these models.

For more information on the FPCR.AH bit, see ‘Flushing denormalized numbers to zero’, Floating-point exceptions and exception traps and the pseudocode of the floating-point instructions.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

RES0

FIZ, bit [0]

When FEAT_AFP is implemented:

Flush Inputs to Zero. Controls whether single-precision, double-precision and BFloat16 input operands that are denormalized numbers are flushed to zero.

FIZ	Meaning
0b0	The flushing to zero of single-precision and double-precision denormalized inputs to floating-point instructions not enabled by this control, but other factors might cause the input denormalized numbers to be flushed to zero.
0b1	Denormalized single-precision and double-precision inputs to most floating-point instructions flushed to zero.

For more information, see ‘Flushing denormalized numbers to zero’ and the pseudocode of the floating-point instructions.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

RES0

Accessing the FPCR

Accesses to this register use the following encodings in the instruction encoding space:

MRS <Xt>, FPCR

op0	op1	CRn	CRm	op2
0b11	0b011	0b0100	0b0100	0b000

```

1 if PSTATE.EL == EL0 then
2   if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap priority
   ↳when SDD == '1' && CPTR_EL3.TFP == '1' then
3     UNDEFINED;
4   elseif !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') && CPACR_EL1.FPEN != '11' then
5     if EL2Enabled() && HCR_EL2.TGE == '1' then
6       AArch64.SystemAccessTrap(EL2, 0x00);

```

```

7      else
8          AArch64.SystemAccessTrap(EL1, 0x07);
9      elsif EL2Enabled() && HCR_EL2.<E2H,TGE> == '11' && CPTR_EL2.FPEN != '11' then
10         AArch64.SystemAccessTrap(EL2, 0x07);
11      elsif EL2Enabled() && HCR_EL2.E2H == '1' && CPTR_EL2.FPEN == 'x0' then
12         AArch64.SystemAccessTrap(EL2, 0x07);
13      elsif EL2Enabled() && HCR_EL2.E2H != '1' && CPTR_EL2.TFP == '1' then
14         AArch64.SystemAccessTrap(EL2, 0x07);
15      elsif HaveEL(EL3) && CPTR_EL3.TFP == '1' then
16         if Halted() && EDSCR.SDD == '1' then
17             UNDEFINED;
18         else
19             AArch64.SystemAccessTrap(EL3, 0x07);
20     else
21         X[t, 64] = FPCR;
22     elsif PSTATE.EL == EL1 then
23         if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap priority
24             ↪when SDD == '1' && CPTR_EL3.TFP == '1' then
25                 UNDEFINED;
26         elsif CPACR_EL1.FPEN == 'x0' then
27             AArch64.SystemAccessTrap(EL1, 0x07);
28         elsif EL2Enabled() && HCR_EL2.E2H != '1' && CPTR_EL2.TFP == '1' then
29             AArch64.SystemAccessTrap(EL2, 0x07);
30         elsif EL2Enabled() && HCR_EL2.E2H == '1' && CPTR_EL2.FPEN == 'x0' then
31             AArch64.SystemAccessTrap(EL2, 0x07);
32         elsif HaveEL(EL3) && CPTR_EL3.TFP == '1' then
33             if Halted() && EDSCR.SDD == '1' then
34                 UNDEFINED;
35             else
36                 AArch64.SystemAccessTrap(EL3, 0x07);
37         else
38             X[t, 64] = FPCR;
39     elsif PSTATE.EL == EL2 then
40         if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap priority
41             ↪when SDD == '1' && CPTR_EL3.TFP == '1' then
42                 UNDEFINED;
43         elsif HCR_EL2.E2H == '0' && CPTR_EL2.TFP == '1' then
44             AArch64.SystemAccessTrap(EL2, 0x07);
45         elsif HCR_EL2.E2H == '1' && CPTR_EL2.FPEN == 'x0' then
46             AArch64.SystemAccessTrap(EL2, 0x07);
47         elsif HaveEL(EL3) && CPTR_EL3.TFP == '1' then
48             if Halted() && EDSCR.SDD == '1' then
49                 UNDEFINED;
50             else
51                 AArch64.SystemAccessTrap(EL3, 0x07);
52         else
53             X[t, 64] = FPCR;
54     elsif PSTATE.EL == EL3 then
55         if CPTR_EL3.TFP == '1' then
56             AArch64.SystemAccessTrap(EL3, 0x07);
57         else
58             X[t, 64] = FPCR;

```

MSR FPCR, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b011	0b0100	0b0100	0b000

```

1  if PSTATE.EL == EL0 then
2      if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap priority
3          ↪when SDD == '1' && CPTR_EL3.TFP == '1' then
4              UNDEFINED;
5      elsif !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') && CPACR_EL1.FPEN != '11' then
6          if EL2Enabled() && HCR_EL2.TGE == '1' then
7              AArch64.SystemAccessTrap(EL2, 0x00);
8          else
9              AArch64.SystemAccessTrap(EL1, 0x07);
10         elsif EL2Enabled() && HCR_EL2.<E2H,TGE> == '11' && CPTR_EL2.FPEN != '11' then
11             AArch64.SystemAccessTrap(EL2, 0x07);
12         elsif EL2Enabled() && HCR_EL2.E2H == '1' && CPTR_EL2.FPEN == 'x0' then
13             AArch64.SystemAccessTrap(EL2, 0x07);
14         elsif EL2Enabled() && HCR_EL2.E2H != '1' && CPTR_EL2.TFP == '1' then

```

```

14      AArch64.SystemAccessTrap(EL2, 0x07);
15  elseif HaveEL(EL3) && CPTR_EL3.TFP == '1' then
16      if Halted() && EDSCR.SDD == '1' then
17          UNDEFINED;
18      else
19          AArch64.SystemAccessTrap(EL3, 0x07);
20      else
21          FPCR = X[t, 64];
22  elseif PSTATE.EL == EL1 then
23      if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap priority
24          ↪when SDD == '1' && CPTR_EL3.TFP == '1' then
25          UNDEFINED;
26      elseif CPACR_EL1.FPEN == 'x0' then
27          AArch64.SystemAccessTrap(EL1, 0x07);
28      elseif EL2Enabled() && HCR_EL2.E2H != '1' && CPTR_EL2.TFP == '1' then
29          AArch64.SystemAccessTrap(EL2, 0x07);
30      elseif EL2Enabled() && HCR_EL2.E2H == '1' && CPTR_EL2.FPEN == 'x0' then
31          AArch64.SystemAccessTrap(EL2, 0x07);
32      elseif HaveEL(EL3) && CPTR_EL3.TFP == '1' then
33          if Halted() && EDSCR.SDD == '1' then
34              UNDEFINED;
35          else
36              AArch64.SystemAccessTrap(EL3, 0x07);
37      else
38          FPCR = X[t, 64];
39  elseif PSTATE.EL == EL2 then
40      if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap priority
41          ↪when SDD == '1' && CPTR_EL3.TFP == '1' then
42          UNDEFINED;
43      elseif HCR_EL2.E2H == '0' && CPTR_EL2.TFP == '1' then
44          AArch64.SystemAccessTrap(EL2, 0x07);
45      elseif HCR_EL2.E2H == '1' && CPTR_EL2.FPEN == 'x0' then
46          AArch64.SystemAccessTrap(EL2, 0x07);
47      elseif HaveEL(EL3) && CPTR_EL3.TFP == '1' then
48          if Halted() && EDSCR.SDD == '1' then
49              UNDEFINED;
50          else
51              AArch64.SystemAccessTrap(EL3, 0x07);
52      else
53          FPCR = X[t, 64];
54  elseif PSTATE.EL == EL3 then
55      if CPTR_EL3.TFP == '1' then
56          AArch64.SystemAccessTrap(EL3, 0x07);
57      else
58          FPCR = X[t, 64];

```

E3.2.8 HCRX_EL2, Extended Hypervisor Configuration Register

The HCRX_EL2 characteristics are:

Purpose

Provides configuration controls for virtualization, including defining whether various operations are trapped to EL2.

Attributes

HCRX_EL2 is a 64-bit register.

Configuration

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

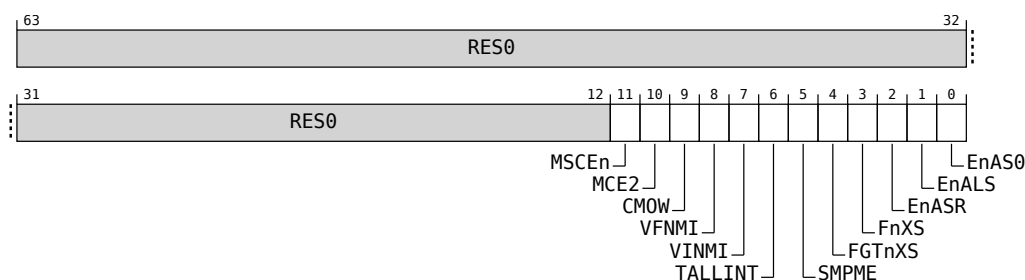
The bits in this register behave as if they are 0 for all purposes other than direct reads of the register if:

- EL2 is not enabled in the current Security state.
- [SCR_EL3.HXEn](#) is 0.

This register is present only when FEAT_HCX is implemented. Otherwise, direct accesses to HCRX_EL2 are UNDEFINED.

Field descriptions

The HCRX_EL2 bit assignments are:



Bits [63:12]

Reserved, RES0.

MSCEn, bit [11]

When FEAT_MOPS is implemented:

Memory Set and Memory Copy instructions Enable. Enables execution of the CPY*, SETG*, SETP*, SETM*, and SETE* instructions at EL1 or EL0.

MSCEn	Meaning
0b0	Execution of the Memory Copy and Memory Set instructions is UNDEFINED at EL1 or EL0.
0b1	This control does not cause any instructions to be UNDEFINED.

This bit behaves as if it is 1 if any of the following are true:

- EL2 is not implemented or enabled.
- The value of HCR_EL2.{E2H, TGE} is {1, 1}.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

RES0

MCE2, bit [10]

When FEAT_MOPS is implemented:

Controls Memory Copy and Memory Set exceptions generated as part of attempting to execute the Memory Copy and Memory Set instructions from EL1.

MCE2	Meaning
0b0	Memory Copy and Memory Set exceptions generated from EL1 are taken to EL1.
0b1	Memory Copy and Memory Set exceptions generated from EL1 are taken to EL2.

When the value of HCR_EL2.{E2H, TGE} is {1, 1}, this control does not affect any exceptions due to the higher priority [SCTLR_EL2.MSCEn](#) control.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

RES0

CMOW, bit [9]

When FEAT_CMOW is implemented:

Controls cache maintenance instruction permission for the following instructions executed at EL1 or EL0.

- IC IVAU, DC CIVAC, DC CIGDVAC and DC CIGVAC.
- ICIMVAU, DCCIMVAC.

CMOW	Meaning
0b0	These instructions executed at EL1 or EL0 with stage 2 read permission, but without stage 2 write permission do not generate a stage 2 permission fault.
0b1	These instructions executed at EL1 or EL0, if enabled as a result of SCTLR_EL1.UCI ==1, with stage 2 read permission, but without stage 2 write permission generate a stage 2 permission fault.

For this control, stage 2 has write permission if S2AP[1] is 1 or DBM is 1 in the stage 2 descriptor. The instructions do not cause an update to the dirty state.

This bit is permitted to be cached in a TLB.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

RES0

VFNMI, bit [8]

When FEAT_NMI is implemented:

Virtual FIQ Interrupt with Superpriority. Enables signaling of virtual FIQ interrupts with Superpriority.

VFNMI	Meaning
0b0	When HCR_EL2.VF is 1, a signaled pending virtual FIQ interrupt does not have Superpriority.
0b1	When HCR_EL2.VF is 1, a signaled pending virtual FIQ interrupt has Superpriority.

When HCR_EL2.VF is 0, this bit has no effect.

The reset behavior of this field is:

- On a Warm reset, when EL3 is not implemented and EL2 is implemented, this field resets to 0b0.

Otherwise:

RES0

VINMI, bit [7]

When FEAT_NMI is implemented:

Virtual IRQ Interrupt with Superpriority. Enables signaling of virtual IRQ interrupts with Superpriority.

VINMI	Meaning
0b0	When HCR_EL2.VI is 1, a signaled pending virtual IRQ interrupt does not have Superpriority.
0b1	When HCR_EL2.VI is 1, a signaled pending virtual IRQ interrupt has Superpriority.

When HCR_EL2.VI is 0, this bit has no effect.

The reset behavior of this field is:

- On a Warm reset, when EL3 is not implemented and EL2 is implemented, this field resets to 0b0.

Otherwise:

RES0

TALLINT, bit [6]

When FEAT_NMI is implemented:

Trap_{MSR} writes of ALLINT at EL1 using AArch64 to EL2, when EL2 is implemented and enabled in the current Security state, reported using EC syndrome value 0x18.

TALLINT	Meaning
0b0	MSR writes of ALLINT are not trapped by this mechanism.
0b1	MSR writes of ALLINT at EL1 using AArch64 are trapped to EL2.

The reset behavior of this field is:

- On a Warm reset, when EL3 is not implemented and EL2 is implemented, this field resets to 0b0.

Otherwise:

RES0

SMPME, bit [5]

When FEAT_SME is implemented:

Streaming Mode Priority Mapping Enable.

Controls mapping of the value of [SMPRI_EL1](#).Priority for streaming execution priority at EL0 or EL1.

SMPME	Meaning
0b0	The effective priority value is taken from SMPRI_EL1 .Priority.
0b1	The effective priority value is: <ul style="list-style-type: none"> When the current Exception level is EL2 or EL3, the value of SMPRI_EL1.Priority. When the current Exception level is EL0 or EL1, the value of the SMPRIMAP_EL2 field corresponding to the value of SMPRI_EL1.Priority.

When [SMIDR_EL1](#).SMPS is '0', this field is RES0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

RES0

FGTnXS, bit [4]

When FEAT_XS is implemented:

Determines if the fine-grained traps in HFGITR_EL2 that apply to each of the TLBI maintenance instructions that are accessible at EL1 also apply to the corresponding TLBI maintenance instructions with the nXS qualifier.

FGTnXS	Meaning
0b0	The fine-grained trap in the HFGITR_EL2 that applies to a TLBI maintenance instruction at EL1 also applies to the corresponding TLBI instruction with the nXS qualifier at EL1.
0b1	The fine-grained trap in the HFGITR_EL2 that applies to a TLBI maintenance instruction at EL1 does not apply to the corresponding TLBI instruction with the nXS qualifier at EL1.

The reset behavior of this field is:

- On a Warm reset, when EL3 is not implemented and EL2 is implemented, this field resets to 0b0.

Otherwise:

RES0

FnXS, bit [3]

When FEAT_XS is implemented:

Determines the behavior of TLBI instructions affected by the XS attribute.

This control bit also determines whether an AArch64 DSB instruction behaves as a DSB instruction with an nXS qualifier when executed at EL0 and EL1.

FnXS	Meaning
0b0	This control does not have any effect on the behavior of the TLBI maintenance instructions.
0b1	A TLBI maintenance instruction without the nXS qualifier executed at EL1 behaves in the same way as the corresponding TLBI maintenance instruction with the nXS qualifier. An AArch64 DSB instruction executed at EL1 or EL0 behaves in the same way as the corresponding DSB instruction with the nXS qualifier executed at EL1 or EL0.

This bit is permitted to be cached in a TLB.

The reset behavior of this field is:

- On a Warm reset, when EL3 is not implemented and EL2 is implemented, this field resets to 0b0.

Otherwise:

RES0

EnASR, bit [2]

When FEAT_LS64_V is implemented:

When HCR_EL2.{E2H, TGE} != {1, 1}, traps execution of an ST64BV instruction at EL0 or EL1 to EL2.

EnASR	Meaning
0b0	Execution of an ST64BV instruction at EL0 is trapped to EL2 if the execution is not trapped by SCTLR_EL1.EnASR . Execution of an ST64BV instruction at EL1 is trapped to EL2.
0b1	This control does not cause any instructions to be trapped.

A trap of an ST64BV instruction is reported using an ESR_ELx.EC value of 0x0A, with an ISS code of 0x0000000.

The reset behavior of this field is:

- On a Warm reset, when EL3 is not implemented and EL2 is implemented, this field resets to 0b0.

Otherwise:

RES0

EnALS, bit [1]

When FEAT_LS64 is implemented:

When HCR_EL2.{E2H, TGE} != {1, 1}, traps execution of an LD64B or ST64B instruction at EL0 or EL1 to EL2.

EnALS	Meaning
0b0	Execution of an LD64B or ST64B instruction at EL0 is trapped to EL2 if the execution is not trapped by SCTLR_EL1.EnALS . Execution of an LD64B or ST64B instruction at EL1 is trapped to EL2.
0b1	This control does not cause any instructions to be trapped.

A trap of an LD64B or ST64B instruction is reported using an ESR_ELx.EC value of 0x0A, with an ISS code of 0x0000002.

The reset behavior of this field is:

- On a Warm reset, when EL3 is not implemented and EL2 is implemented, this field resets to 0b0.

Otherwise:

RES0

EnAS0, bit [0]

When FEAT_LS64_ACCDATA is implemented:

When HCR_EL2.{E2H, TGE} != {1, 1}, traps execution of an ST64BV0 instruction at EL0 or EL1 to EL2.

EnAS0	Meaning
0b0	Execution of an ST64BV0 instruction at EL0 is trapped to EL2 if the execution is not trapped by SCTLR_EL1.EnAS0 . Execution of an ST64BV0 instruction at EL1 is trapped to EL2.
0b1	This control does not cause any instructions to be trapped.

A trap of an ST64BV0 instruction is reported using an ESR_ELx.EC value of 0x0A, with an ISS code of 0x0000001.

The reset behavior of this field is:

- On a Warm reset, when EL3 is not implemented and EL2 is implemented, this field resets to 0b0.

Otherwise:

RES0

Accessing the HCRX_EL2

Accesses to this register use the following encodings in the instruction encoding space:

MRS <Xt>, HCRX_EL2

op0	op1	CRn	CRm	op2
0b11	0b100	0b0001	0b0010	0b010

```

1 if PSTATE.EL == EL0 then
2     UNDEFINED;
3 elseif PSTATE.EL == EL1 then
4     if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
5         X[t, 64] = NVMem[0xA0];
6     elseif EL2Enabled() && HCR_EL2.NV == '1' then
7         AArch64.SystemAccessTrap(EL2, 0x18);
8     else
9         UNDEFINED;
10 elseif PSTATE.EL == EL2 then
11     if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap priority
    ↳when SDD == '1' && SCR_EL3.HXEn == '0' then
12         UNDEFINED;
13     elseif HaveEL(EL3) && SCR_EL3.HXEn == '0' then
14         if Halted() && EDSCR.SDD == '1' then
15             UNDEFINED;
16         else
17             AArch64.SystemAccessTrap(EL3, 0x18);
18     else
19         X[t, 64] = HCRX_EL2;
20 elseif PSTATE.EL == EL3 then
21     X[t, 64] = HCRX_EL2;

```

MSR HCRX_EL2, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b100	0b0001	0b0010	0b010

```

1  if PSTATE.EL == EL0 then
2      UNDEFINED;
3  elsif PSTATE.EL == EL1 then
4      if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
5          NVMem[0xA0] = X[t, 64];
6      elsif EL2Enabled() && HCR_EL2.NV == '1' then
7          AArch64.SystemAccessTrap(EL2, 0x18);
8      else
9          UNDEFINED;
10 elsif PSTATE.EL == EL2 then
11     if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap priority
12         ↪when SDD == '1" && SCR_EL3.HXEn == '0' then
13             UNDEFINED;
14     elsif HaveEL(EL3) && SCR_EL3.HXEn == '0' then
15         if Halted() && EDSCR.SDD == '1' then
16             UNDEFINED;
17         else
18             AArch64.SystemAccessTrap(EL3, 0x18);
19     else
20         HCRX_EL2 = X[t, 64];
21 elsif PSTATE.EL == EL3 then
22     HCRX_EL2 = X[t, 64];

```

E3.2.9 HFGRTR_EL2, Hypervisor Fine-Grained Read Trap Register

The HFGRTR_EL2 characteristics are:

Purpose

Provides controls for traps of `MRS` and `MRC` reads of System registers.

Attributes

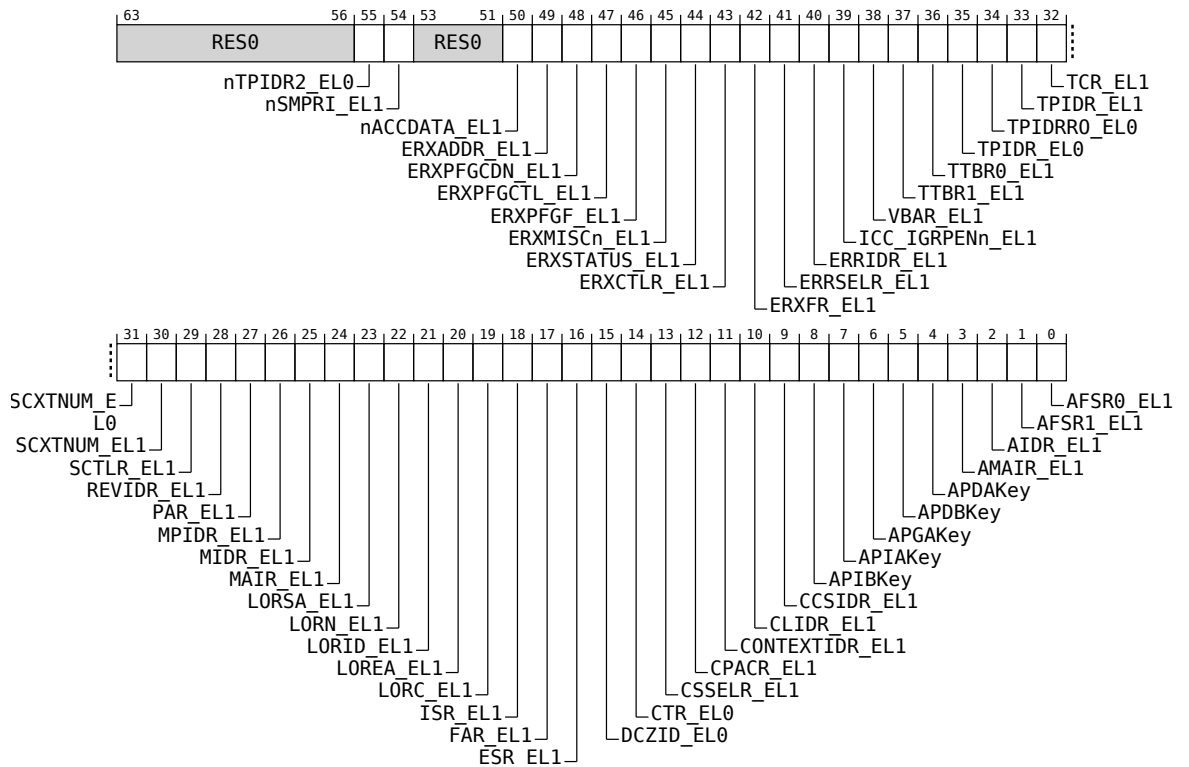
HFGRTR_EL2 is a 64-bit register.

Configuration

This register is present only when `FEAT_FGT` is implemented. Otherwise, direct accesses to HFGRTR_EL2 are UNDEFINED.

Field descriptions

The HFGRTR_EL2 bit assignments are:



Bits [63:56]

Reserved, RES0.

nTPIDR2_EL0, bit [55]

When `FEAT_SME` is implemented:

Trap `MRS` reads of `TPIDR2_EL0` at EL1 and EL0 using AArch64 to EL2.

nTPIDR2_EL0	Meaning
0b0	If EL2 is implemented and enabled in the current Security state, HCR_EL2.{E2H, TGE} != {1, 1}, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then <small>MRS</small> reads of TPIDR2_EL0 at EL1 and EL0 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.
0b1	<small>MRS</small> reads of TPIDR2_EL0 are not trapped by this mechanism.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0b0.

Otherwise:

RES0

nSMPRI_EL1, bit [54]

When FEAT_SME is implemented:

Trap MRS reads of [SMPRI_EL1](#) at EL1 using AArch64 to EL2.

nSMPRI_EL1	Meaning
0b0	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then <small>MRS</small> reads of SMPRI_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.
0b1	<small>MRS</small> reads of SMPRI_EL1 are not trapped by this mechanism.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0b0.

Otherwise:

RES0

Bits [53:51]

Reserved, RES0.

nACCDATA_EL1, bit [50]

When FEAT_LS64_ACCDATA is implemented:

Trap MRS reads of ACCDATA_EL1 at EL1 using AArch64 to EL2.

nACCDATA_EL1	Meaning
0b0	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1 , then <code>MRS</code> reads of ACCDATA_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.
0b1	<code>MRS</code> reads of ACCDATA_EL1 are not trapped by this mechanism.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0b0.

Otherwise:

RES0

ERXADDR_EL1, bit [49]

When FEAT_RAS is implemented:

Trap `MRS` reads of ERXADDR_EL1 at EL1 using AArch64 to EL2.

ERXADDR_EL1	Meaning
0b0	<code>MRS</code> reads of ERXADDR_EL1 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1 , then <code>MRS</code> reads of ERXADDR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0b0.

Otherwise:

RES0

ERXPFGCDN_EL1, bit [48]

When FEAT_RASv1p1 is implemented:

Trap `MRS` reads of ERXPFGCDN_EL1 at EL1 using AArch64 to EL2.

ERXPFGCDN_EL1	Meaning
0b0	<code>MRS</code> reads of ERXPFGCDN_EL1 are not trapped by this mechanism.

ERXPFPGCDN_EL1	Meaning
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then <small>MRS</small> reads of ERXPFPGCDN_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0b0.

Otherwise:

RES0

ERXPFPGCTL_EL1, bit [47]

When FEAT_RASv1p1 is implemented:

Trap MRS reads of ERXPFPGCTL_EL1 at EL1 using AArch64 to EL2.

ERXPFPGCTL_EL1	Meaning
0b0	<small>MRS</small> reads of ERXPFPGCTL_EL1 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then <small>MRS</small> reads of ERXPFPGCTL_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0b0.

Otherwise:

RES0

ERXPFGF_EL1, bit [46]

When FEAT_RAS is implemented:

Trap MRS reads of ERXPFGF_EL1 at EL1 using AArch64 to EL2.

ERXPFGF_EL1	Meaning
0b0	<small>MRS</small> reads of ERXPFGF_EL1 are not trapped by this mechanism.

ERXPFGF_EL1	Meaning
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1 , then <small>MRS</small> reads of ERXPFGF_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0b0.

Otherwise:

RES0

ERXMISCN_EL1, bit [45]

When FEAT_RAS is implemented:

Trap MRS reads of ERXMISCN_EL1 at EL1 using AArch64 to EL2.

ERXMISCN_EL1	Meaning
0b0	<small>MRS</small> reads of ERXMISCN_EL1 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1 , then <small>MRS</small> reads of ERXMISCN_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0b0.

Otherwise:

RES0

ERXSTATUS_EL1, bit [44]

When FEAT_RAS is implemented:

Trap MRS reads of ERXSTATUS_EL1 at EL1 using AArch64 to EL2.

ERXSTATUS_EL1	Meaning
0b0	<small>MRS</small> reads of ERXSTATUS_EL1 are not trapped by this mechanism.

ERXSTATUS_EL1	Meaning
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then <small>MRS</small> reads of ERXSTATUS_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0b0.

Otherwise:

RES0

ERXCTLR_EL1, bit [43]

When FEAT_RAS is implemented:

Trap MRS reads of ERXCTLR_EL1 at EL1 using AArch64 to EL2.

ERXCTLR_EL1	Meaning
0b0	<small>MRS</small> reads of ERXCTLR_EL1 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then <small>MRS</small> reads of ERXCTLR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0b0.

Otherwise:

RES0

ERXFR_EL1, bit [42]

When FEAT_RAS is implemented:

Trap MRS reads of ERXFR_EL1 at EL1 using AArch64 to EL2.

ERXFR_EL1	Meaning
0b0	<small>MRS</small> reads of ERXFR_EL1 are not trapped by this mechanism.

ERXFR_EL1	Meaning
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1 , then <small>MRS</small> reads of ERXFR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0b0.

Otherwise:

RES0

ERRSELR_EL1, bit [41]

When FEAT_RAS is implemented:

Trap MRS reads of ERRSELR_EL1 at EL1 using AArch64 to EL2.

ERRSELR_EL1	Meaning
0b0	<small>MRS</small> reads of ERRSELR_EL1 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1 , then <small>MRS</small> reads of ERRSELR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0b0.

Otherwise:

RES0

ERRIDR_EL1, bit [40]

When FEAT_RAS is implemented:

Trap MRS reads of ERRIDR_EL1 at EL1 using AArch64 to EL2.

ERRIDR_EL1	Meaning
0b0	<small>MRS</small> reads of ERRIDR_EL1 are not trapped by this mechanism.

ERRIDR_EL1	Meaning
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1 , then <small>MRS</small> reads of ERRIDR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0b0.

Otherwise:

RES0

ICC_IGRPENn_EL1, bit [39]

When FEAT_GICv3 is implemented:

Trap MRS reads of ICC_IGRPEN<n>_EL1 at EL1 using AArch64 to EL2.

ICC_IGRPENn_EL1	Meaning
0b0	<small>MRS</small> reads of ICC_IGRPEN<n>_EL1 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1 , then <small>MRS</small> reads of ICC_IGRPEN<n>_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0b0.

Otherwise:

RES0

VBAR_EL1, bit [38]

Trap MRS reads of VBAR_EL1 at EL1 using AArch64 to EL2.

VBAR_EL1	Meaning
0b0	<small>MRS</small> reads of VBAR_EL1 are not trapped by this mechanism.

VBAR_EL1	Meaning
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MRS reads of VBAR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0b0.

TTBR1_EL1, bit [37]

Trap [MRS](#) reads of TTBR1_EL1 at EL1 using AArch64 to EL2.

TTBR1_EL1	Meaning
0b0	MRS reads of TTBR1_EL1 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MRS reads of TTBR1_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0b0.

TTBR0_EL1, bit [36]

Trap [MRS](#) reads of TTBR0_EL1 at EL1 using AArch64 to EL2.

TTBR0_EL1	Meaning
0b0	MRS reads of TTBR0_EL1 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MRS reads of TTBR0_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0b0.

TPIDR_EL0, bit [35]

Trap MRS reads of TPIDR_EL0 at EL1 and EL0 using AArch64 and MRC reads of TPIDRURW at EL0 using AArch32 when EL1 is using AArch64 to EL2.

TPIDR_EL0	Meaning
0b0	<small>MRS</small> reads of TPIDR_EL0 at EL1 and EL0 using AArch64 and <small>MRC</small> reads of TPIDRURW at EL0 using AArch32 are not trapped by this mechanism.
0b1	<p>If EL2 is implemented and enabled in the current Security state, HCR_EL2.{E2H, TGE} != {1, 1}, EL1 is using AArch64, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then, unless the read generates a higher priority exception:</p> <ul style="list-style-type: none"> <small>MRS</small> reads of TPIDR_EL0 at EL1 and EL0 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18. <small>MRC</small> reads of TPIDRURW at EL0 using AArch32 are trapped to EL2 and reported with EC syndrome value 0x03.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0b0.

TPIDRRO_EL0, bit [34]

Trap MRS reads of TPIDRRO_EL0 at EL1 and EL0 using AArch64 and MRC reads of TPIDRURO at EL0 using AArch32 when EL1 is using AArch64 to EL2.

TPIDRRO_EL0	Meaning
0b0	<small>MRS</small> reads of TPIDRRO_EL0 at EL1 and EL0 using AArch64 and <small>MRC</small> reads of TPIDRURO at EL0 using AArch32 are not trapped by this mechanism.
0b1	<p>If EL2 is implemented and enabled in the current Security state, HCR_EL2.{E2H, TGE} != {1, 1}, EL1 is using AArch64, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then, unless the read generates a higher priority exception:</p> <ul style="list-style-type: none"> <small>MRS</small> reads of TPIDRRO_EL0 at EL1 and EL0 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18. <small>MRC</small> reads of TPIDRURO at EL0 using AArch32 are trapped to EL2 and reported with EC syndrome value 0x03.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0b0.

TPIDR_EL1, bit [33]

Trap MRS reads of TPIDR_EL1 at EL1 using AArch64 to EL2.

TPIDR_EL1	Meaning
0b0	<small>MRS</small> reads of TPIDR_EL1 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1 , then <small>MRS</small> reads of TPIDR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0b0.

TCR_EL1, bit [32]

Trap MRS reads of TCR_EL1 at EL1 using AArch64 to EL2.

TCR_EL1	Meaning
0b0	<small>MRS</small> reads of TCR_EL1 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1 , then <small>MRS</small> reads of TCR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0b0.

SCXTNUM_EL0, bit [31]

When FEAT_CSV2_2 is implemented or FEAT_CSV2_1p2 is implemented:

Trap MRS reads of SCXTNUM_EL0 at EL1 and EL0 using AArch64 to EL2.

SCXTNUM_EL0	Meaning
0b0	<small>MRS</small> reads of SCXTNUM_EL0 are not trapped by this mechanism.

SCXTNUM_EL0	Meaning
0b1	If EL2 is implemented and enabled in the current Security state, HCR_EL2.{E2H, TGE} != {1, 1}, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then <small>MRS</small> reads of SCXTNUM_EL0 at EL1 and EL0 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0b0.

Otherwise:

RES0

SCXTNUM_EL1, bit [30]

When FEAT_CSV2_2 is implemented or FEAT_CSV2_1p2 is implemented:

Trap MRS reads of SCXTNUM_EL1 at EL1 using AArch64 to EL2.

SCXTNUM_EL1	Meaning
0b0	<small>MRS</small> reads of SCXTNUM_EL1 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then <small>MRS</small> reads of SCXTNUM_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0b0.

Otherwise:

RES0

SCTLR_EL1, bit [29]

Trap MRS reads of [SCTLR_EL1](#) at EL1 using AArch64 to EL2.

SCTLR_EL1	Meaning
0b0	<small>MRS</small> reads of SCTLR_EL1 are not trapped by this mechanism.

SCTLR_EL1	Meaning
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1 , then <code>MRS</code> reads of SCTLR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0b0.

REVIDR_EL1, bit [28]

Trap `MRS` reads of REVIDR_EL1 at EL1 using AArch64 to EL2.

REVIDR_EL1	Meaning
0b0	<code>MRS</code> reads of REVIDR_EL1 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1 , then <code>MRS</code> reads of REVIDR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0b0.

PAR_EL1, bit [27]

Trap `MRS` reads of PAR_EL1 at EL1 using AArch64 to EL2.

PAR_EL1	Meaning
0b0	<code>MRS</code> reads of PAR_EL1 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1 , then <code>MRS</code> reads of PAR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0b0.

MPIDR_EL1, bit [26]

Trap MRS reads of MPIDR_EL1 at EL1 using AArch64 to EL2.

MPIDR_EL1	Meaning
0b0	<small>MRS</small> reads of MPIDR_EL1 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1 , then <small>MRS</small> reads of MPIDR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0b0.

MIDR_EL1, bit [25]

Trap MRS reads of MIDR_EL1 at EL1 using AArch64 to EL2.

MIDR_EL1	Meaning
0b0	<small>MRS</small> reads of MIDR_EL1 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1 , then <small>MRS</small> reads of MIDR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0b0.

MAIR_EL1, bit [24]

Trap MRS reads of MAIR_EL1 at EL1 using AArch64 to EL2.

MAIR_EL1	Meaning
0b0	<small>MRS</small> reads of MAIR_EL1 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1 , then <small>MRS</small> reads of MAIR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0b0.

LORSA_EL1, bit [23]

When FEAT_LOR is implemented:

Trap MRS reads of LORSA_EL1 at EL1 using AArch64 to EL2.

LORSA_EL1	Meaning
0b0	<small>MRS</small> reads of LORSA_EL1 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1 , then <small>MRS</small> reads of LORSA_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0b0.

Otherwise:

RES0

LORN_EL1, bit [22]

When FEAT_LOR is implemented:

Trap MRS reads of LORN_EL1 at EL1 using AArch64 to EL2.

LORN_EL1	Meaning
0b0	<small>MRS</small> reads of LORN_EL1 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1 , then <small>MRS</small> reads of LORN_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0b0.

Otherwise:

RES0

LORID_EL1, bit [21]

When FEAT_LOR is implemented:

Trap MRS reads of LORID_EL1 at EL1 using AArch64 to EL2.

LORID_EL1	Meaning
0b0	<small>MRS</small> reads of LORID_EL1 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1 , then <small>MRS</small> reads of LORID_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0b0.

Otherwise:

RES0

LOREA_EL1, bit [20]

When FEAT_LOR is implemented:

Trap MRS reads of LOREA_EL1 at EL1 using AArch64 to EL2.

LOREA_EL1	Meaning
0b0	<small>MRS</small> reads of LOREA_EL1 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1 , then <small>MRS</small> reads of LOREA_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0b0.

Otherwise:

RES0

LORC_EL1, bit [19]

When FEAT_LOR is implemented:

Trap MRS reads of LORC_EL1 at EL1 using AArch64 to EL2.

LORC_EL1	Meaning
0b0	<small>MRS</small> reads of LORC_EL1 are not trapped by this mechanism.

LORC_EL1	Meaning
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MRS reads of LORC_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0b0.

Otherwise:

RES0

ISR_EL1, bit [18]

Trap [MRS](#) reads of ISR_EL1 at EL1 using AArch64 to EL2.

ISR_EL1	Meaning
0b0	MRS reads of ISR_EL1 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MRS reads of ISR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0b0.

FAR_EL1, bit [17]

Trap [MRS](#) reads of [FAR_EL1](#) at EL1 using AArch64 to EL2.

FAR_EL1	Meaning
0b0	MRS reads of FAR_EL1 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MRS reads of FAR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0b0.

ESR_EL1, bit [16]

Trap MRS reads of ESR_EL1 at EL1 using AArch64 to EL2.

ESR_EL1	Meaning
0b0	<small>MRS</small> reads of ESR_EL1 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1 , then <small>MRS</small> reads of ESR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0b0.

DCZID_EL0, bit [15]

Trap MRS reads of DCZID_EL0 at EL1 and EL0 using AArch64 to EL2.

DCZID_EL0	Meaning
0b0	<small>MRS</small> reads of DCZID_EL0 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, HCR_EL2.{E2H, TGE} != {1, 1}, and either EL3 is not implemented or SCR_EL3.FGTEn == 1 , then <small>MRS</small> reads of DCZID_EL0 at EL1 and EL0 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0b0.

CTR_EL0, bit [14]

Trap MRS reads of CTR_EL0 at EL1 and EL0 using AArch64 to EL2.

CTR_EL0	Meaning
0b0	<small>MRS</small> reads of CTR_EL0 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, HCR_EL2.{E2H, TGE} != {1, 1}, and either EL3 is not implemented or SCR_EL3.FGTEn == 1 , then <small>MRS</small> reads of CTR_EL0 at EL1 and EL0 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0b0.

CSSELR_EL1, bit [13]

Trap MRS reads of CSSELR_EL1 at EL1 using AArch64 to EL2.

CSSELR_EL1	Meaning
0b0	<small>MRS</small> reads of CSSELR_EL1 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1 , then <small>MRS</small> reads of CSSELR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0b0.

CPACR_EL1, bit [12]

Trap MRS reads of [CPACR_EL1](#) at EL1 using AArch64 to EL2.

CPACR_EL1	Meaning
0b0	<small>MRS</small> reads of CPACR_EL1 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1 , then <small>MRS</small> reads of CPACR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0b0.

CONTEXTIDR_EL1, bit [11]

Trap MRS reads of CONTEXTIDR_EL1 at EL1 using AArch64 to EL2.

CONTEXTIDR_EL1	Meaning
0b0	<small>MRS</small> reads of CONTEXTIDR_EL1 are not trapped by this mechanism.

CONTEXTIDR_EL1	Meaning
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1 , then <small>MRS</small> reads of CONTEXTIDR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0b0.

CLIDR_EL1, bit [10]

Trap MRS reads of CLIDR_EL1 at EL1 using AArch64 to EL2.

CLIDR_EL1	Meaning
0b0	<small>MRS</small> reads of CLIDR_EL1 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1 , then <small>MRS</small> reads of CLIDR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0b0.

CCSIDR_EL1, bit [9]

Trap MRS reads of CCSIDR_EL1 at EL1 using AArch64 to EL2.

CCSIDR_EL1	Meaning
0b0	<small>MRS</small> reads of CCSIDR_EL1 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1 , then <small>MRS</small> reads of CCSIDR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0b0.

APIBKey, bit [8]

When FEAT_PAuth is implemented:

Trap `MRS` reads of multiple System registers. Enables a trap on `MRS` reads at EL1 using AArch64 of any of the following AArch64 System registers to EL2:

- `APIBKeyHi_EL1`.
- `APIBKeyLo_EL1`.

APIBKey	Meaning
0b0	<code>MRS</code> reads of the System registers listed above are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or <code>SCR_EL3.FGTEn == 1</code> , then <code>MRS</code> reads at EL1 using AArch64 of any of the System registers listed above are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0b0.

Otherwise:

RES0

APIAKey, bit [7]

When FEAT_PAuth is implemented:

Trap `MRS` reads of multiple System registers. Enables a trap on `MRS` reads at EL1 using AArch64 of any of the following AArch64 System registers to EL2:

- `APIAKeyHi_EL1`.
- `APIAKeyLo_EL1`.

APIAKey	Meaning
0b0	<code>MRS</code> reads of the System registers listed above are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or <code>SCR_EL3.FGTEn == 1</code> , then <code>MRS</code> reads at EL1 using AArch64 of any of the System registers listed above are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0b0.

Otherwise:

RES0

APGAKey, bit [6]

When FEAT_PAuth is implemented:

Trap MRS reads of multiple System registers. Enables a trap on MRS reads at EL1 using AArch64 of any of the following AArch64 System registers to EL2:

- APGAKeyHi_EL1.
- APGAKeyLo_EL1.

APGAKey	Meaning
0b0	<small>MRS</small> reads of the System registers listed above are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then <small>MRS</small> reads at EL1 using AArch64 of any of the System registers listed above are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0b0.

Otherwise:

RES0

APDBKey, bit [5]

When FEAT_PAuth is implemented:

Trap MRS reads of multiple System registers. Enables a trap on MRS reads at EL1 using AArch64 of any of the following AArch64 System registers to EL2:

- APDBKeyHi_EL1.
- APDBKeyLo_EL1.

APDBKey	Meaning
0b0	<small>MRS</small> reads of the System registers listed above are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then <small>MRS</small> reads at EL1 using AArch64 of any of the System registers listed above are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0b0.

Otherwise:

RES0

APDAKey, bit [4]

When FEAT_PAuth is implemented:

Trap MRS reads of multiple System registers. Enables a trap on MRS reads at EL1 using AArch64 of any of the following AArch64 System registers to EL2:

- APDAKeyHi_EL1.
- APDAKeyLo_EL1.

APDAKey	Meaning
0b0	<small>MRS</small> reads of the System registers listed above are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then <small>MRS</small> reads at EL1 using AArch64 of any of the System registers listed above are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0b0.

Otherwise:

RES0

AMAIR_EL1, bit [3]

Trap MRS reads of AMAIR_EL1 at EL1 using AArch64 to EL2.

AMAIR_EL1	Meaning
0b0	<small>MRS</small> reads of AMAIR_EL1 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then <small>MRS</small> reads of AMAIR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0b0.

AIDR_EL1, bit [2]

Trap MRS reads of AIDR_EL1 at EL1 using AArch64 to EL2.

AIDR_EL1	Meaning
0b0	MRS reads of AIDR_EL1 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1 , then MRS reads of AIDR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0b0.

AFSR1_EL1, bit [1]

Trap MRS reads of AFSR1_EL1 at EL1 using AArch64 to EL2.

AFSR1_EL1	Meaning
0b0	MRS reads of AFSR1_EL1 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1 , then MRS reads of AFSR1_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0b0.

AFSR0_EL1, bit [0]

Trap MRS reads of AFSR0_EL1 at EL1 using AArch64 to EL2.

AFSR0_EL1	Meaning
0b0	MRS reads of AFSR0_EL1 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1 , then MRS reads of AFSR0_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0b0.

Accessing the HFGTR_EL2

Accesses to this register use the following encodings in the instruction encoding space:

MRS <Xt>, HFGTR_EL2

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

```

1 if PSTATE.EL == EL0 then
2     UNDEFINED;
3 elseif PSTATE.EL == EL1 then
4     if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
5         X[t, 64] = NVMem[0x1B8];
6     elseif EL2Enabled() && HCR_EL2.NV == '1' then
7         AArch64.SystemAccessTrap(EL2, 0x18);
8     else
9         UNDEFINED;
10 elseif PSTATE.EL == EL2 then
11     if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap priority
12         ↪when SDD == '1' && SCR_EL3.FGTEn == '0' then
13         UNDEFINED;
14     elseif HaveEL(EL3) && SCR_EL3.FGTEn == '0' then
15         if Halted() && EDSCR.SDD == '1' then
16             UNDEFINED;
17         else
18             AArch64.SystemAccessTrap(EL3, 0x18);
19     else
20         X[t, 64] = HFGTR_EL2;
21 elseif PSTATE.EL == EL3 then
22     X[t, 64] = HFGTR_EL2;

```

MSR HFGTR_EL2, <Xt>

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

```

1 if PSTATE.EL == EL0 then
2     UNDEFINED;
3 elseif PSTATE.EL == EL1 then
4     if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
5         NVMem[0x1B8] = X[t, 64];
6     elseif EL2Enabled() && HCR_EL2.NV == '1' then
7         AArch64.SystemAccessTrap(EL2, 0x18);
8     else
9         UNDEFINED;
10 elseif PSTATE.EL == EL2 then
11     if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap priority
12         ↪when SDD == '1' && SCR_EL3.FGTEn == '0' then
13         UNDEFINED;
14     elseif HaveEL(EL3) && SCR_EL3.FGTEn == '0' then
15         if Halted() && EDSCR.SDD == '1' then
16             UNDEFINED;
17         else
18             AArch64.SystemAccessTrap(EL3, 0x18);
19     else
20         HFGTR_EL2 = X[t, 64];
21 elseif PSTATE.EL == EL3 then
22     HFGTR_EL2 = X[t, 64];

```

E3.2.10 HFGWTR_EL2, Hypervisor Fine-Grained Write Trap Register

The HFGWTR_EL2 characteristics are:

Purpose

Provides controls for traps of `MSR` and `MCR` writes of System registers.

Attributes

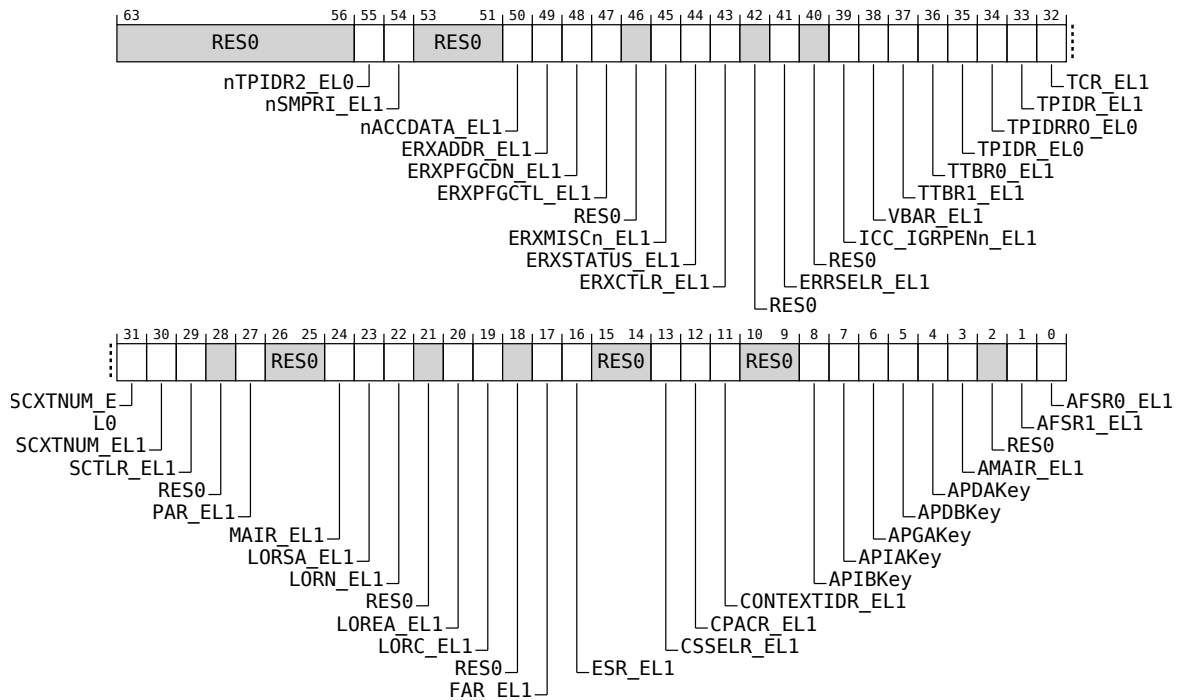
HFGWTR_EL2 is a 64-bit register.

Configuration

This register is present only when FEAT_FGT is implemented. Otherwise, direct accesses to HFGWTR_EL2 are UNDEFINED.

Field descriptions

The HFGWTR_EL2 bit assignments are:



Bits [63:56]

Reserved, RES0.

nTPIDR2_EL0, bit [55]

When FEAT_SME is implemented:

Trap `MSR` writes of `TPIDR2_EL0` at EL1 and EL0 using AArch64 to EL2.

nTPIDR2_EL0	Meaning
0b0	If EL2 is implemented and enabled in the current Security state, HCR_EL2.{E2H, TGE} != {1, 1}, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MSR writes of TPIDR2_EL0 at EL1 and EL0 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.
0b1	MSR writes of TPIDR2_EL0 are not trapped by this mechanism.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0b0.

Otherwise:

RES0

nSMPRI_EL1, bit [54]

When FEAT_SME is implemented:

Trap [MSR](#) writes of [SMPRI_EL1](#) at EL1 using AArch64 to EL2.

nSMPRI_EL1	Meaning
0b0	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MSR writes of SMPRI_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.
0b1	MSR writes of SMPRI_EL1 are not trapped by this mechanism.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0b0.

Otherwise:

RES0

Bits [53:51]

Reserved, RES0.

nACCDATA_EL1, bit [50]

When FEAT_LS64_ACCDATA is implemented:

Trap [MSR](#) writes of [ACCDATA_EL1](#) at EL1 using AArch64 to EL2.

nACCDATA_EL1	Meaning
0b0	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1 , then <code>MSR</code> writes of <code>ACCDATA_EL1</code> at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.
0b1	<code>MSR</code> writes of <code>ACCDATA_EL1</code> are not trapped by this mechanism.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0b0.

Otherwise:

RES0

ERXADDR_EL1, bit [49]

When FEAT_RAS is implemented:

Trap `MSR` writes of `ERXADDR_EL1` at EL1 using AArch64 to EL2.

ERXADDR_EL1	Meaning
0b0	<code>MSR</code> writes of <code>ERXADDR_EL1</code> are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1 , then <code>MSR</code> writes of <code>ERXADDR_EL1</code> at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0b0.

Otherwise:

RES0

ERXPFGCDN_EL1, bit [48]

When FEAT_RASv1p1 is implemented:

Trap `MSR` writes of `ERXPFGCDN_EL1` at EL1 using AArch64 to EL2.

ERXPFGCDN_EL1	Meaning
0b0	<code>MSR</code> writes of <code>ERXPFGCDN_EL1</code> are not trapped by this mechanism.

ERXPFPGCDN_EL1	Meaning
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MSR writes of ERXPFPGCDN_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0b0.

Otherwise:

RES0

ERXPFPGCTL_EL1, bit [47]

When FEAT_RASv1p1 is implemented:

Trap [MSR](#) writes of ERXPFPGCTL_EL1 at EL1 using AArch64 to EL2.

ERXPFPGCTL_EL1	Meaning
0b0	MSR writes of ERXPFPGCTL_EL1 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MSR writes of ERXPFPGCTL_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0b0.

Otherwise:

RES0

Bit [46]

Reserved, RES0.

ERXMISCN_EL1, bit [45]

When FEAT_RAS is implemented:

Trap [MSR](#) writes of ERXMISCN_EL1 at EL1 using AArch64 to EL2.

ERXMISCN_EL1	Meaning
0b0	MSR writes of ERXMISCN_EL1 are not trapped by this mechanism.

ERXMISCn_EL1	Meaning
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MSR writes of ERXMISC<n>_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0b0.

Otherwise:

RES0

ERXSTATUS_EL1, bit [44]

When FEAT_RAS is implemented:

Trap [MSR](#) writes of ERXSTATUS_EL1 at EL1 using AArch64 to EL2.

ERXSTATUS_EL1	Meaning
0b0	MSR writes of ERXSTATUS_EL1 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MSR writes of ERXSTATUS_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0b0.

Otherwise:

RES0

ERXCTLR_EL1, bit [43]

When FEAT_RAS is implemented:

Trap [MSR](#) writes of ERXCTLR_EL1 at EL1 using AArch64 to EL2.

ERXCTLR_EL1	Meaning
0b0	MSR writes of ERXCTLR_EL1 are not trapped by this mechanism.

ERXCTLR_EL1	Meaning
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MSR writes of ERXCTLR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0b0.

Otherwise:

RES0

Bit [42]

Reserved, RES0.

ERRSELR_EL1, bit [41]

When FEAT_RAS is implemented:

Trap [MSR](#) writes of ERRSELR_EL1 at EL1 using AArch64 to EL2.

ERRSELR_EL1	Meaning
0b0	MSR writes of ERRSELR_EL1 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MSR writes of ERRSELR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0b0.

Otherwise:

RES0

Bit [40]

Reserved, RES0.

ICC_IGRPENn_EL1, bit [39]

When FEAT_GICv3 is implemented:

Trap [MSR](#) writes of ICC_IGRPEN<n>_EL1 at EL1 using AArch64 to EL2.

ICC_IGRPEN _n _EL1	Meaning
0b0	MSR writes of ICC_IGRPEN<n>_EL1 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MSR writes of ICC_IGRPEN<n>_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0b0.

Otherwise:

RES0

VBAR_EL1, bit [38]

Trap MSR writes of VBAR_EL1 at EL1 using AArch64 to EL2.

VBAR_EL1	Meaning
0b0	MSR writes of VBAR_EL1 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MSR writes of VBAR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0b0.

TTBR1_EL1, bit [37]

Trap MSR writes of TTBR1_EL1 at EL1 using AArch64 to EL2.

TTBR1_EL1	Meaning
0b0	MSR writes of TTBR1_EL1 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MSR writes of TTBR1_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0b0.

TTBR0_EL1, bit [36]

Trap MSR writes of TTBR0_EL1 at EL1 using AArch64 to EL2.

TTBR0_EL1	Meaning
0b0	<small>MSR</small> writes of TTBR0_EL1 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1 , then <small>MSR</small> writes of TTBR0_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0b0.

TPIDR_EL0, bit [35]

Trap MSR writes of TPIDR_EL0 at EL1 and EL0 using AArch64 and MCR writes of TPIDRURW at EL0 using AArch32 when EL1 is using AArch64 to EL2.

TPIDR_EL0	Meaning
0b0	<small>MSR</small> writes of TPIDR_EL0 at EL1 and EL0 using AArch64 and <small>MCR</small> writes of TPIDRURW at EL0 using AArch32 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, $HCR_EL2.\{E2H, TGE\} \neq \{1, 1\}$, EL1 is using AArch64, and either EL3 is not implemented or SCR_EL3.FGTEn == 1 , then, unless the write generates a higher priority exception: <ul style="list-style-type: none"> • <small>MSR</small> writes of TPIDR_EL0 at EL1 and EL0 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18. • <small>MCR</small> writes of TPIDRURW at EL0 using AArch32 are trapped to EL2 and reported with EC syndrome value 0x03.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0b0.

TPIDRRO_EL0, bit [34]

Trap MSR writes of TPIDRRO_EL0 at EL1 using AArch64 to EL2.

TPIDRRO_EL0	Meaning
0b0	MSR writes of TPIDRRO_EL0 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1 , then MSR writes of TPIDRRO_EL0 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0b0.

TPIDR_EL1, bit [33]

Trap MSR writes of TPIDR_EL1 at EL1 using AArch64 to EL2.

TPIDR_EL1	Meaning
0b0	MSR writes of TPIDR_EL1 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1 , then MSR writes of TPIDR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0b0.

TCR_EL1, bit [32]

Trap MSR writes of TCR_EL1 at EL1 using AArch64 to EL2.

TCR_EL1	Meaning
0b0	MSR writes of TCR_EL1 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1 , then MSR writes of TCR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0b0.

SCXTNUM_EL0, bit [31]

When FEAT_CSV2_2 is implemented or FEAT_CSV2_1p2 is implemented:

Trap MSR writes of SCXTNUM_EL0 at EL1 and EL0 using AArch64 to EL2.

SCXTNUM_EL0	Meaning
0b0	<small>MSR</small> writes of SCXTNUM_EL0 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, HCR_EL2.{E2H, TGE} != {1, 1}, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then <small>MSR</small> writes of SCXTNUM_EL0 at EL1 and EL0 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0b0.

Otherwise:

RES0

SCXTNUM_EL1, bit [30]

When FEAT_CSV2_2 is implemented or FEAT_CSV2_1p2 is implemented:

Trap MSR writes of SCXTNUM_EL1 at EL1 using AArch64 to EL2.

SCXTNUM_EL1	Meaning
0b0	<small>MSR</small> writes of SCXTNUM_EL1 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then <small>MSR</small> writes of SCXTNUM_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0b0.

Otherwise:

RES0

SCTLR_EL1, bit [29]

Trap MSR writes of [SCTLR_EL1](#) at EL1 using AArch64 to EL2.

SCTLR_EL1	Meaning
0b0	MSR writes of SCTLR_EL1 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MSR writes of SCTLR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0b0.

Bit [28]

Reserved, RES0.

PAR_EL1, bit [27]

Trap MSR writes of PAR_EL1 at EL1 using AArch64 to EL2.

PAR_EL1	Meaning
0b0	MSR writes of PAR_EL1 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MSR writes of PAR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0b0.

Bits [26:25]

Reserved, RES0.

MAIR_EL1, bit [24]

Trap MSR writes of MAIR_EL1 at EL1 using AArch64 to EL2.

MAIR_EL1	Meaning
0b0	MSR writes of MAIR_EL1 are not trapped by this mechanism.

MAIR_EL1	Meaning
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1 , then <code>MSR</code> writes of MAIR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0b0.

LORSA_EL1, bit [23]

When FEAT_LOR is implemented:

Trap `MSR` writes of LORSA_EL1 at EL1 using AArch64 to EL2.

LORSA_EL1	Meaning
0b0	<code>MSR</code> writes of LORSA_EL1 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1 , then <code>MSR</code> writes of LORSA_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0b0.

Otherwise:

RES0

LORN_EL1, bit [22]

When FEAT_LOR is implemented:

Trap `MSR` writes of LORN_EL1 at EL1 using AArch64 to EL2.

LORN_EL1	Meaning
0b0	<code>MSR</code> writes of LORN_EL1 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1 , then <code>MSR</code> writes of LORN_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0b0.

Otherwise:

RES0

Bit [21]

Reserved, RES0.

LOREA_EL1, bit [20]

When FEAT_LOR is implemented:

Trap MSR writes of LOREA_EL1 at EL1 using AArch64 to EL2.

LOREA_EL1	Meaning
0b0	<small>MSR</small> writes of LOREA_EL1 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1 , then <small>MSR</small> writes of LOREA_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0b0.

Otherwise:

RES0

LORC_EL1, bit [19]

When FEAT_LOR is implemented:

Trap MSR writes of LORC_EL1 at EL1 using AArch64 to EL2.

LORC_EL1	Meaning
0b0	<small>MSR</small> writes of LORC_EL1 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1 , then <small>MSR</small> writes of LORC_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0b0.

Otherwise:

RES0

Bit [18]

Reserved, RES0.

FAR_EL1, bit [17]

Trap MSR writes of FAR_EL1 at EL1 using AArch64 to EL2.

FAR_EL1	Meaning
0b0	<small>MSR</small> writes of FAR_EL1 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or <small>SCR_EL3.FGTEn</small> == 1, then <small>MSR</small> writes of FAR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0b0.

ESR_EL1, bit [16]

Trap MSR writes of ESR_EL1 at EL1 using AArch64 to EL2.

ESR_EL1	Meaning
0b0	<small>MSR</small> writes of ESR_EL1 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or <small>SCR_EL3.FGTEn</small> == 1, then <small>MSR</small> writes of ESR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0b0.

Bits [15:14]

Reserved, RES0.

CSSELR_EL1, bit [13]

Trap MSR writes of CSSELR_EL1 at EL1 using AArch64 to EL2.

CSSELR_EL1	Meaning
0b0	MSR writes of CSSELR_EL1 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1 , then MSR writes of CSSELR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0b0.

CPACR_EL1, bit [12]

Trap MSR writes of [CPACR_EL1](#) at EL1 using AArch64 to EL2.

CPACR_EL1	Meaning
0b0	MSR writes of CPACR_EL1 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1 , then MSR writes of CPACR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0b0.

CONTEXTIDR_EL1, bit [11]

Trap MSR writes of CONTEXTIDR_EL1 at EL1 using AArch64 to EL2.

CONTEXTIDR_EL1	Meaning
0b0	MSR writes of CONTEXTIDR_EL1 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1 , then MSR writes of CONTEXTIDR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0b0.

Bits [10:9]

Reserved, RES0.

APIBKey, bit [8]

When FEAT_PAuth is implemented:

Trap_{MSR} writes of multiple System registers. Enables a trap on MSR writes at EL1 using AArch64 of any of the following AArch64 System registers to EL2:

- APIBKeyHi_EL1.
- APIBKeyLo_EL1.

APIBKey	Meaning
0b0	MSR writes of the System registers listed above are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1 , then MSR writes at EL1 using AArch64 of any of the System registers listed above are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0b0.

Otherwise:

RES0

APIAKey, bit [7]

When FEAT_PAuth is implemented:

Trap_{MSR} writes of multiple System registers. Enables a trap on MSR writes at EL1 using AArch64 of any of the following AArch64 System registers to EL2:

- APIAKeyHi_EL1.
- APIAKeyLo_EL1.

APIAKey	Meaning
0b0	MSR writes of the System registers listed above are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1 , then MSR writes at EL1 using AArch64 of any of the System registers listed above are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0b0.

Otherwise:

RES0

APGAKey, bit [6]

When FEAT_PAuth is implemented:

Trap_{MSR} writes of multiple System registers. Enables a trap on MSR writes at EL1 using AArch64 of any of the following AArch64 System registers to EL2:

- APGAKeyHi_EL1.
- APGAKeyLo_EL1.

APGAKey	Meaning
0b0	MSR writes of the System registers listed above are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1 , then MSR writes at EL1 using AArch64 of any of the System registers listed above are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0b0.

Otherwise:

RES0

APDBKey, bit [5]

When FEAT_PAuth is implemented:

Trap_{MSR} writes of multiple System registers. Enables a trap on MSR writes at EL1 using AArch64 of any of the following AArch64 System registers to EL2:

- APDBKeyHi_EL1.
- APDBKeyLo_EL1.

APDBKey	Meaning
0b0	MSR writes of the System registers listed above are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1 , then MSR writes at EL1 using AArch64 of any of the System registers listed above are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0b0.

Otherwise:

RES0

APDAKey, bit [4]

When FEAT_PAuth is implemented:

Trap_{MSR} writes of multiple System registers. Enables a trap on MSR writes at EL1 using AArch64 of any of the following AArch64 System registers to EL2:

- APDAKeyHi_EL1.
- APDAKeyLo_EL1.

APDAKey	Meaning
0b0	MSR writes of the System registers listed above are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MSR writes at EL1 using AArch64 of any of the System registers listed above are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0b0.

Otherwise:

RES0

AMAIR_EL1, bit [3]

Trap_{MSR} writes of AMAIR_EL1 at EL1 using AArch64 to EL2.

AMAIR_EL1	Meaning
0b0	MSR writes of AMAIR_EL1 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MSR writes of AMAIR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0b0.

Bit [2]

Reserved, RES0.

AFSR1_EL1, bit [1]

Trap MSR writes of AFSR1_EL1 at EL1 using AArch64 to EL2.

AFSR1_EL1	Meaning
0b0	<small>MSR</small> writes of AFSR1_EL1 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1 , then <small>MSR</small> writes of AFSR1_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0b0.

AFSR0_EL1, bit [0]

Trap MSR writes of AFSR0_EL1 at EL1 using AArch64 to EL2.

AFSR0_EL1	Meaning
0b0	<small>MSR</small> writes of AFSR0_EL1 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1 , then <small>MSR</small> writes of AFSR0_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0b0.

Accessing the HFGWTR_EL2

Accesses to this register use the following encodings in the instruction encoding space:

MRS <Xt>, HFGWTR_EL2

op0	op1	CRn	CRm	op2
0b11	0b100	0b0001	0b0001	0b101


```

1  if PSTATE.EL == EL0 then
2      UNDEFINED;
3  elsif PSTATE.EL == EL1 then
4      if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
5          X[t, 64] = NVMem[0x1C0];
6      elsif EL2Enabled() && HCR_EL2.NV == '1' then
7          AArch64.SystemAccessTrap(EL2, 0x18);
8      else
9          UNDEFINED;
10  elsif PSTATE.EL == EL2 then
11      if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap priority
12          ↪when SDD == '1'" && SCR_EL3.FGTEn == '0' then
13          UNDEFINED;
14      elsif HaveEL(EL3) && SCR_EL3.FGTEn == '0' then
15          if Halted() && EDSCR.SDD == '1' then
16              UNDEFINED;
17          else
18              AArch64.SystemAccessTrap(EL3, 0x18);
19      else
20          X[t, 64] = HFGWTR_EL2;
21  elsif PSTATE.EL == EL3 then
22      X[t, 64] = HFGWTR_EL2;

```

MSR HFGWTR_EL2, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b100	0b0001	0b0001	0b101

```

1  if PSTATE.EL == EL0 then
2      UNDEFINED;
3  elsif PSTATE.EL == EL1 then
4      if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
5          NVMem[0x1C0] = X[t, 64];
6      elsif EL2Enabled() && HCR_EL2.NV == '1' then
7          AArch64.SystemAccessTrap(EL2, 0x18);
8      else
9          UNDEFINED;
10  elsif PSTATE.EL == EL2 then
11      if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap priority
12          ↪when SDD == '1'" && SCR_EL3.FGTEn == '0' then
13          UNDEFINED;
14      elsif HaveEL(EL3) && SCR_EL3.FGTEn == '0' then
15          if Halted() && EDSCR.SDD == '1' then
16              UNDEFINED;
17          else
18              AArch64.SystemAccessTrap(EL3, 0x18);
19      else
20          HFGWTR_EL2 = X[t, 64];
21  elsif PSTATE.EL == EL3 then
22      HFGWTR_EL2 = X[t, 64];

```

E3.2.11 ID_AA64ISAR1_EL1, AArch64 Instruction Set Attribute Register 1

The ID_AA64ISAR1_EL1 characteristics are:

Purpose

Provides information about the features and instructions implemented in AArch64 state.

For general information about the interpretation of the ID registers, see Principles of the ID scheme for fields in ID registers .

Attributes

ID_AA64ISAR1_EL1 is a 64-bit register.

Field descriptions

The ID_AA64ISAR1_EL1 bit assignments are:

63	60	59	56	55	52	51	48	47	44	43	40	39	36	35	32
LS64		XS			I8MM		DGH		BF16		SPECRES		SB		FRINTTS
31	28	27	24	23	20	19	16	15	12	11	8	7	4	3	0
	GPI		GPA		LRCPC		FCMA		JSCVT		API		APA		DPB

LS64, bits [63:60]

Indicates support for LD64B and ST64B* instructions, and the ACCDATA_EL1 register. Defined values of this field are:

LS64	Meaning
0b0000	The LD64B, ST64B, ST64BV, and ST64BV0 instructions, the ACCDATA_EL1 register, and associated traps are not supported.
0b0001	The LD64B and ST64B instructions are supported.
0b0010	The LD64B, ST64B, and ST64BV instructions, and their associated traps are supported.
0b0011	The LD64B, ST64B, ST64BV, and ST64BV0 instructions, the ACCDATA_EL1 register, and their associated traps are supported.

All other values are reserved.

FEAT_LS64 implements the functionality identified by 0b0001.

FEAT_LS64_V implements the functionality identified by 0b0010.

FEAT_LS64_ACCDATA implements the functionality identified by 0b0011.

From Armv8.7, the permitted values are 0b0000, 0b0001, 0b0010, and 0b0011.

XS, bits [59:56]

Indicates support for the XS attribute, the TLBI and DSB instructions with the nXS qualifier, and the [HCRX_EL2](#).{FGTnXS, FnXS} fields in AArch64 state. Defined values are:

XS	Meaning
0b0000	The XS attribute, the TLBI and DSB instructions with the nXS qualifier, and the HCRX_EL2 .{FGTnXS, FnXS} fields are not supported.
0b0001	The XS attribute, the TLBI and DSB instructions with the nXS qualifier, and the HCRX_EL2 .{FGTnXS, FnXS} fields are supported.

All other values are reserved.

FEAT_XS implements the functionality identified by 0b0001.

From Armv8.7, the only permitted value is 0b0001.

I8MM, bits [55:52]

Indicates support for Advanced SIMD and Floating-point Int8 matrix multiplication instructions in AArch64 state. Defined values are:

I8MM	Meaning
0b0000	Int8 matrix multiplication instructions are not implemented.
0b0001	SMMLA, SUDOT, UMMLA, USMMLA, and USDOT instructions are implemented.

All other values are reserved.

FEAT_I8MM implements the functionality identified by 0b0001.

When Advanced SIMD and SVE are both implemented, this field must return the same value as [ID_AA64ZFR0_EL1](#).I8MM.

From Armv8.6, the only permitted value is 0b0001.

DGH, bits [51:48]

Indicates support for the Data Gathering Hint instruction. Defined values are:

DGH	Meaning
0b0000	Data Gathering Hint is not implemented.
0b0001	Data Gathering Hint is implemented.

All other values are reserved.

FEAT_DGH implements the functionality identified by 0b0001.

From Armv8.0, the permitted values are 0b0000 and 0b0001.

If the [DGH](#) instruction has no effect in preventing the merging of memory accesses, the value of this field is 0b0000.

BF16, bits [47:44]

Indicates support for Advanced SIMD and Floating-point BFloat16 instructions in AArch64 state. Defined values

are:

BF16	Meaning
0b0000	BFloat16 instructions are not implemented.
0b0001	BFCVT, BFCVTN, BFCVTN2, BFDOT, BFMLALB, BFMLALT, and BFMLLA instructions are implemented.
0b0010	As 0b0001, but the FPCR .EBF field is also supported.

All other values are reserved.

FEAT_BF16 implements the functionality identified by 0b0001.

FEAT_EBF16 implements the functionality identified by 0b0010.

When FEAT_SVE or FEAT_SME are implemented, this field must return the same value as [ID_AA64ZFR0_EL1](#).BF16.

If FEAT_SME is implemented, the permitted values are 0b0001 and 0b0010.

Otherwise, from Armv8.6, the only permitted value is 0b0001.

SPECRES, bits [43:40]

Indicates support for prediction invalidation instructions in AArch64 state. Defined values are:

SPECRES	Meaning
0b0000	CFP RCTX, DVP RCTX, and CPP RCTX instructions are not implemented.
0b0001	CFP RCTX, DVP RCTX, and CPP RCTX instructions are implemented.

All other values are reserved.

FEAT_SPECRES implements the functionality identified by 0b0001.

In Armv8.0, the permitted values are 0b0000 and 0b0001.

From Armv8.5, the only permitted value is 0b0001.

SB, bits [39:36]

Indicates support for SB instruction in AArch64 state. Defined values are:

SB	Meaning
0b0000	SB instruction is not implemented.
0b0001	SB instruction is implemented.

All other values are reserved.

FEAT_SB implements the functionality identified by 0b0001.

In Armv8.0, the permitted values are 0b0000 and 0b0001.

From Armv8.5, the only permitted value is 0b0001.

FRINTTS, bits [35:32]

Indicates support for the FRINT32Z, FRINT32X, FRINT64Z, and FRINT64X instructions are implemented. Defined values are:

FRINTTS	Meaning
0b0000	FRINT32Z, FRINT32X, FRINT64Z, and FRINT64X instructions are not implemented.
0b0001	FRINT32Z, FRINT32X, FRINT64Z, and FRINT64X instructions are implemented.

All other values are reserved.

FEAT_FRINTTS implements the functionality identified by 0b0001.

From Armv8.5, the only permitted value is 0b0001.

GPI, bits [31:28]

Indicates support for an IMPLEMENTATION DEFINED algorithm is implemented in the PE for generic code authentication in AArch64 state. Defined values are:

GPI	Meaning
0b0000	Generic Authentication using an IMPLEMENTATION DEFINED algorithm is not implemented.
0b0001	Generic Authentication using an IMPLEMENTATION DEFINED algorithm is implemented. This includes the PACGA instruction.

All other values are reserved.

FEAT_PACIMP implements the functionality identified by 0b0001.

From Armv8.3, the permitted values are 0b0000 and 0b0001.

If the value of ID_AA64ISAR1_EL1.GPA is non-zero, or the value of ID_AA64ISAR2_EL1.GPA3 is non-zero, this field must have the value 0b0000.

GPA, bits [27:24]

Indicates whether the QARMA5 algorithm is implemented in the PE for generic code authentication in AArch64 state. Defined values are:

GPA	Meaning
0b0000	Generic Authentication using the QARMA5 algorithm is not implemented.
0b0001	Generic Authentication using the QARMA5 algorithm is implemented. This includes the PACGA instruction.

All other values are reserved.

FEAT_PACQARMA5 implements the functionality identified by 0b0001.

From Armv8.3, the permitted values are 0b0000 and 0b0001.

If the value of ID_AA64ISAR1_EL1.GPI is non-zero, or the value of ID_AA64ISAR2_EL1.GPA3 is non-zero, this field must have the value 0b0000.

LRCPC, bits [23:20]

Indicates support for weaker release consistency, RCpc, based model. Defined values are:

LRCPC	Meaning
0b0000	The LDAPR*, LDAPUR*, and STLUR* instructions are not implemented.
0b0001	The LDAPR* instructions are implemented. The LDAPUR*, and STLUR* instructions are not implemented.
0b0010	The LDAPR*, LDAPUR*, and STLUR* instructions are implemented.

All other values are reserved.

FEAT_LRCPC implements the functionality identified by the value 0b0001.

FEAT_LRCPC2 implements the functionality identified by the value 0b0010.

In Armv8.2, the permitted values are 0b0000, 0b0001, and 0b0010.

In Armv8.3, the permitted values are 0b0001 and 0b0010.

From Armv8.4, the only permitted value is 0b0010.

FCMA, bits [19:16]

Indicates support for complex number addition and multiplication, where numbers are stored in vectors. Defined values are:

FCMA	Meaning
0b0000	The FCMLA and FCADD instructions are not implemented.
0b0001	The FCMLA and FCADD instructions are implemented.

All other values are reserved.

FEAT_FCMA implements the functionality identified by the value 0b0001.

In Armv8.0, Armv8.1, and Armv8.2, the only permitted value is 0b0000.

From Armv8.3, if Advanced SIMD or Floating-point is implemented, the only permitted value is 0b0001.

From Armv8.3, if Advanced SIMD or Floating-point is not implemented, the only permitted value is 0b0000.

JSCVT, bits [15:12]

Indicates support for JavaScript conversion from double precision floating point values to integers in AArch64 state. Defined values are:

JSCVT	Meaning
0b0000	The FJCVTZS instruction is not implemented.
0b0001	The FJCVTZS instruction is implemented.

All other values are reserved.

FEAT_JSCVT implements the functionality identified by 0b0001.

In Armv8.0, Armv8.1, and Armv8.2, the only permitted value is 0b0000.

From Armv8.3, if Advanced SIMD or Floating-point is implemented, the only permitted value is 0b0001.

From Armv8.3, if Advanced SIMD or Floating-point is not implemented, the only permitted value is 0b0000.

API, bits [11:8]

Indicates whether an IMPLEMENTATION DEFINED algorithm is implemented in the PE for address authentication, in AArch64 state. This applies to all Pointer Authentication instructions other than the PACGA instruction. Defined values are:

API	Meaning
0b0000	Address Authentication using an IMPLEMENTATION DEFINED algorithm is not implemented.
0b0001	Address Authentication using an IMPLEMENTATION DEFINED algorithm is implemented, with the HaveEnhancedPAC() and HaveEnhancedPAC2() functions returning FALSE.
0b0010	Address Authentication using an IMPLEMENTATION DEFINED algorithm is implemented, with the HaveEnhancedPAC() function returning TRUE, and the HaveEnhancedPAC2() function returning FALSE.
0b0011	Address Authentication using an IMPLEMENTATION DEFINED algorithm is implemented, with the HaveEnhancedPAC2() function returning TRUE, and the HaveEnhancedPAC() function returning FALSE.
0b0100	Address Authentication using an IMPLEMENTATION DEFINED algorithm is implemented, with the HaveEnhancedPAC2() function returning TRUE, the HaveFPAC() function returning TRUE, the HaveFPACCombined() function returning FALSE, and the HaveEnhancedPAC() function returning FALSE.
0b0101	Address Authentication using an IMPLEMENTATION DEFINED algorithm is implemented, with the HaveEnhancedPAC2() function returning TRUE, the HaveFPAC() function returning TRUE, the HaveFPACCombined() function returning TRUE, and the HaveEnhancedPAC() function returning FALSE.

All other values are reserved.

FEAT_PAAuth implements the functionality identified by 0b0001.

FEAT_EPAC implements the functionality identified by 0b0010.

FEAT_PAuth2 implements the functionality identified by 0b0011.

FEAT_FPAC implements the functionality identified by 0b0100.

FEAT_FPACCOMBINE implements the functionality identified by 0b0101.

When this field is non-zero, FEAT_PACIMP is implemented.

In Armv8.3, the permitted values are 0b0001, 0b0010, 0b0011, 0b0100, and 0b0101.

From Armv8.6, the permitted values are 0b0011, 0b0100, and 0b0101.

If the value of ID_AA64ISAR1_EL1.APA is non-zero, or the value of ID_AA64ISAR2_EL1.APA3 is non-zero, this field must have the value 0b0000.

APA, bits [7:4]

Indicates whether the QARMA5 algorithm is implemented in the PE for address authentication, in AArch64 state. This applies to all Pointer Authentication instructions other than the `PACGA` instruction. Defined values are:

APA	Meaning
0b0000	Address Authentication using the QARMA5 algorithm is not implemented.
0b0001	Address Authentication using the QARMA5 algorithm is implemented, with the HaveEnhancedPAC() and HaveEnhancedPAC2() functions returning FALSE.
0b0010	Address Authentication using the QARMA5 algorithm is implemented, with the HaveEnhancedPAC() function returning TRUE and the HaveEnhancedPAC2() function returning FALSE.
0b0011	Address Authentication using the QARMA5 algorithm is implemented, with the HaveEnhancedPAC2() function returning TRUE, the HaveFPAC() function returning FALSE, the HaveFPACCombined() function returning FALSE, and the HaveEnhancedPAC() function returning FALSE.
0b0100	Address Authentication using the QARMA5 algorithm is implemented, with the HaveEnhancedPAC2() function returning TRUE, the HaveFPAC() function returning TRUE, the HaveFPACCombined() function returning FALSE, and the HaveEnhancedPAC() function returning FALSE.
0b0101	Address Authentication using the QARMA5 algorithm is implemented, with the HaveEnhancedPAC2() function returning TRUE, the HaveFPAC() function returning TRUE, the HaveFPACCombined() function returning TRUE, and the HaveEnhancedPAC() function returning FALSE.

All other values are reserved.

FEAT_PAuth implements the functionality identified by 0b0001.

FEAT_EPAC implements the functionality identified by 0b0010.

FEAT_PAuth2 implements the functionality identified by 0b0011.

FEAT_FPAC implements the functionality identified by 0b0100.

FEAT_FPACCOMBINE implements the functionality identified by 0b0101.

When this field is non-zero, FEAT_PACQARMA5 is implemented.

In Armv8.3, the permitted values are 0b0001, 0b0010, 0b0011, 0b0100, and 0b0101.

From Armv8.6, the permitted values are 0b0011, 0b0100, and 0b0101.

If the value of ID_AA64ISAR1_EL1.API is non-zero, or the value of ID_AA64ISAR2_EL1.APA3 is non-zero, this field must have the value 0b0000.

DPB, bits [3:0]

Data Persistence writeback. Indicates support for the DC CVAP and DC CVADP instructions in AArch64 state. Defined values are:

DPB	Meaning
0b0000	DC CVAP not supported.
0b0001	DC CVAP supported.
0b0010	DC CVAP and DC CVADP supported.

All other values are reserved.

FEAT_DPB implements the functionality identified by the value 0b0001.

FEAT_DPB2 implements the functionality identified by the value 0b0010.

In Armv8.2, the permitted values are 0b0001 and 0b0010.

From Armv8.5, the only permitted value is 0b0010.

Accessing the ID_AA64ISAR1_EL1

Accesses to this register use the following encodings in the instruction encoding space:

MRS <Xt>, ID_AA64ISAR1_EL1

op0	op1	CRn	CRm	op2
0b11	0b000	0b0000	0b0110	0b001

```

1  if PSTATE.EL == EL0 then
2      if IsFeatureImplemented(FEAT_IDST) then
3          if EL2Enabled() && HCR_EL2.TGE == '1' then
4              AArch64.SystemAccessTrap(EL2, 0x18);
5          else
6              AArch64.SystemAccessTrap(EL1, 0x18);
7      else
8          UNDEFINED;
9  elseif PSTATE.EL == EL1 then
10     if EL2Enabled() && HCR_EL2.TID3 == '1' then
11         AArch64.SystemAccessTrap(EL2, 0x18);
12     else
13         X[t, 64] = ID_AA64ISAR1_EL1;
14 elseif PSTATE.EL == EL2 then
15     X[t, 64] = ID_AA64ISAR1_EL1;
16 elseif PSTATE.EL == EL3 then
17     X[t, 64] = ID_AA64ISAR1_EL1;
```

E3.2.12 ID_AA64PFR1_EL1, AArch64 Processor Feature Register 1

The ID_AA64PFR1_EL1 characteristics are:

Purpose

Reserved for future expansion of information about implemented PE features in AArch64 state.

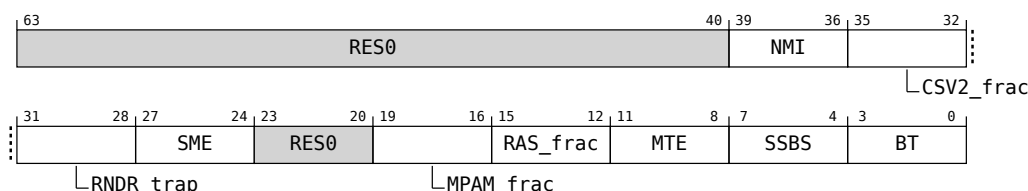
For general information about the interpretation of the ID registers, see Principles of the ID scheme for fields in ID registers .

Attributes

ID_AA64PFR1_EL1 is a 64-bit register.

Field descriptions

The ID_AA64PFR1_EL1 bit assignments are:



Bits [63:40]

Reserved, RES0.

NMI, bits [39:36]

Non-maskable Interrupt. Indicates support for Non-maskable interrupts. Defined values are:

NMI	Meaning
0b0000	SCTLR_ELx.{SPINTMASK, NMI} and PSTATE.ALLINT with its associated instructions are not supported.
0b0001	SCTLR_ELx.{SPINTMASK, NMI} and PSTATE.ALLINT with its associated instructions are supported.

All other values are reserved.

FEAT_NMI implements the functionality identified by the value 0b0001.

From Armv8.8, the only permitted value is 0b0001.

CSV2_frac, bits [35:32]

CSV2 fractional field. Defined values are:

CSV2_frac	Meaning
0b0000	This PE does not disclose whether branch targets trained in one hardware-described context can exploitatively control speculative execution in a different hardware-described context. The SCXTNUM_ELx registers are not supported.

CSV2_frac	Meaning
0b0001	If ID_AA64PFR0_EL1.CSV2 is 0b0001, branch targets trained in one hardware-described context can exploitatively control speculative execution in a different hardware-described context only in a hard-to-determine way. 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. The SCXTNUM_ELx registers are not supported and the contexts do not include the SCXTNUM_ELx register contexts.
0b0010	If ID_AA64PFR0_EL1.CSV2 is 0b0001, branch targets trained in one hardware-described context can exploitatively control speculative execution in a different hardware-described context only in a hard-to-determine way. 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. The SCXTNUM_ELx registers are supported, but the contexts do not include the SCXTNUM_ELx register contexts.

All other values are reserved.

FEAT_CSV2_1p1 implements the functionality identified by the value 0b0001.

FEAT_CSV2_1p2 implements the functionality identified by the value 0b0010.

From Armv8.0, the permitted values are 0b0000, 0b0001, and 0b0010.

This field is valid only if ID_AA64PFR0_EL1.CSV2 is 0b0001.

RNDR_trap, bits [31:28]

Random Number trap to EL3 field. Defined values are:

RNDR_trap	Meaning
0b0000	Trapping of RNDR and RNDRRS to EL3 is not supported.
0b0001	Trapping of RNDR and RNDRRS to EL3 is supported. SCR_EL3.TRNDR is present.

All other values are reserved.

FEAT_RNG_TRAP implements the functionality identified by the value 0b0001.

SME, bits [27:24]

Scalable Matrix Extension field. Defined values are:

SME	Meaning
0b0000	SME architectural state and programmers' model are not implemented.

SME	Meaning
0b0001	SME architectural state and programmers' model are implemented.

All other values are reserved.

FEAT_SME implements the functionality identified by the value 0b0001.

From Armv9.2, the permitted values are 0b0000 and 0b0001.

Bits [23:20]

Reserved, RES0.

MPAM_frac, bits [19:16]

Indicates the minor version number of support for the MPAM Extension.

Defined values are:

MPAM_frac	Meaning
0b0000	The minor version number of the MPAM extension is 0.
0b0001	The minor version number of the MPAM extension is 1.

All other values are reserved.

When combined with the major version number from ID_AA64PFR0_EL1.MPAM, The combined “major.minor” version is:

MPAM Extension version	MPAM	MPAM_frac
Not implemented.	0b0000	0b0000
v0.1 is implemented.	0b0000	0b0001
v1.0 is implemented.	0b0001	0b0000
v1.1 is implemented.	0b0001	0b0001

For more information, see The Memory Partitioning and Monitoring (MPAM) Extension .

RAS_frac, bits [15:12]

RAS Extension fractional field. Defined values are:

RAS_frac	Meaning
0b0000	If ID_AA64PFR0_EL1.RAS == 0b0001, RAS Extension implemented.

RAS_frac	Meaning
0b0001	<p>If ID_AA64PFR0_EL1.RAS == 0b0001, as 0b0000 and adds support for:</p> <ul style="list-style-type: none"> • Additional ERXMISC<m>_EL1 System registers. • Additional System registers ERXPFGCDN_EL1, ERXPFGCTL_EL1, and ERXPFGF_EL1, and the SCR_EL3.FIEN and HCR_EL2.FIEN trap controls, to support the optional RAS Common Fault Injection Model Extension. <p>Error records accessed through System registers conform to RAS System Architecture v1.1, which includes simplifications to ERR<n>STATUS, and support for the optional RAS Timestamp and RAS Common Fault Injection Model Extensions.</p>

All other values are reserved.

FEAT_RASv1p1 implements the functionality identified by the value 0b0001.

This field is valid only if ID_AA64PFR0_EL1.RAS == 0b0001.

MTE, bits [11:8]

Support for the Memory Tagging Extension. Defined values are:

MTE	Meaning
0b0000	Memory Tagging Extension is not implemented.
0b0001	Instruction-only Memory Tagging Extension is implemented.
0b0010	Full Memory Tagging Extension is implemented.
0b0011	Memory Tagging Extension is implemented with support for asymmetric Tag Check Fault handling.

All other values are reserved.

FEAT_MTE implements the functionality identified by the value 0b0001.

FEAT_MTE2 implements the functionality identified by the value 0b0010.

FEAT_MTE3 implements the functionality identified by the value 0b0011.

In Armv8.5, the permitted values are 0b0000, 0b0001 and 0b0010.

From Armv8.7, the value 0b0001 is not permitted.

SSBS, bits [7:4]

Speculative Store Bypassing controls in AArch64 state. Defined values are:

SSBS	Meaning
0b0000	AArch64 provides no mechanism to control the use of Speculative Store Bypassing.

SSBS	Meaning
0b0001	AArch64 provides the PSTATE.SSBS mechanism to mark regions that are Speculative Store Bypass Safe.
0b0010	As 0b0001, and adds the MSR and MRS instructions to directly read and write the PSTATE.SSBS field.

All other values are reserved.

FEAT_SSBS implements the functionality identified by the value 0b0001.

FEAT_SSBS2 implements the functionality identified by the value 0b0010.

BT, bits [3:0]

Branch Target Identification mechanism support in AArch64 state. Defined values are:

BT	Meaning
0b0000	The Branch Target Identification mechanism is not implemented.
0b0001	The Branch Target Identification mechanism is implemented.

All other values are reserved.

FEAT_BTI implements the functionality identified by the value 0b0001.

From Armv8.5, the only permitted value is 0b0001.

Accessing the ID_AA64PFR1_EL1

Accesses to this register use the following encodings in the instruction encoding space:

MRS <Xt>, ID_AA64PFR1_EL1

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

```

1  if PSTATE.EL == EL0 then
2      if IsFeatureImplemented(FEAT_IDST) then
3          if EL2Enabled() && HCR_EL2.TGE == '1' then
4              AArch64.SystemAccessTrap(EL2, 0x18);
5          else
6              AArch64.SystemAccessTrap(EL1, 0x18);
7      else
8          UNDEFINED;
9  elseif PSTATE.EL == EL1 then
10     if EL2Enabled() && HCR_EL2.TID3 == '1' then
11         AArch64.SystemAccessTrap(EL2, 0x18);
12     else
13         X[t, 64] = ID_AA64PFR1_EL1;
14 elseif PSTATE.EL == EL2 then
15     X[t, 64] = ID_AA64PFR1_EL1;
16 elseif PSTATE.EL == EL3 then
17     X[t, 64] = ID_AA64PFR1_EL1;
```


E3.2.13 ID_AA64ZFR0_EL1, SVE Feature ID register 0

The ID_AA64ZFR0_EL1 characteristics are:

Purpose

Provides additional information about the implemented features of the AArch64 Scalable Vector Extension instruction set, when either or both of ID_AA64PFR0_EL1.SVE and ID_AA64PFR1_EL1.SME are not zero.

For general information about the interpretation of the ID registers see Principles of the ID scheme for fields in ID registers .

Attributes

ID_AA64ZFR0_EL1 is a 64-bit register.

Configuration

Prior to the introduction of the features described by this register, this register was unnamed and reserved, RES0 from EL1, EL2, and EL3.

Field descriptions

The ID_AA64ZFR0_EL1 bit assignments are:

63	60	59	56	55	52	51	48	47	44	43	40	39	36	35	32
RES0	F64MM	F32MM	RES0	I8MM	SM4	RES0	SHA3								
31	24	23	20	19	16	15	8	7	4	3	0				
RES0	BF16	BitPerm	RES0	AES	SVEver										

Bits [63:60]

Reserved, RES0.

F64MM, bits [59:56]

Indicates support for SVE FP64 double-precision floating-point matrix multiplication instructions. Defined values are:

F64MM	Meaning
0b0000	Double-precision matrix multiplication and related instructions are not implemented.
0b0001	Double-precision variant of the FMMLA instruction, and the LD1RO* instructions are implemented. The 128-bit element variations of TRN1, TRN2, UZP1, UZP2, ZIP1, and ZIP2 are also implemented.

All other values are reserved.

FEAT_F64MM implements the functionality identified by 0b0001.

From Armv8.2, the permitted values are 0b0000 and 0b0001.

F32MM, bits [55:52]

Indicates support for the SVE FP32 single-precision floating-point matrix multiplication instruction. Defined

values are:

F32MM	Meaning
0b0000	Single-precision matrix multiplication instruction is not implemented.
0b0001	Single-precision variant of the FMMLA instruction is implemented.

All other values are reserved.

FEAT_F32MM implements the functionality identified by 0b0001.

From Arm v8.2, the permitted values are 0b0000 and 0b0001.

Bits [51:48]

Reserved, RES0.

I8MM, bits [47:44]

Indicates support for SVE Int8 matrix multiplication instructions. Defined values are:

I8MM	Meaning
0b0000	Int8 matrix multiplication instructions are not implemented.
0b0001	SMMLA, SUDOT, UMMLA, USMMLA, and USDOT instructions are implemented.

All other values are reserved.

FEAT_I8MM implements the functionality identified by 0b0001.

When Advanced SIMD and SVE are both implemented, this field must return the same value as [ID_AA64ISAR1_EL1.I8MM](#).

From Armv8.6, the only permitted value is 0b0001.

SM4, bits [43:40]

Indicates support for SVE SM4 instructions. Defined values are:

SM4	Meaning
0b0000	SVE SM4 instructions are not implemented.
0b0001	SVE SM4E and SM4EKEY instructions are implemented.

All other values are reserved.

FEAT_SVE_SM4 implements the functionality identified by 0b0001.

Bits [39:36]

Reserved, RES0.

SHA3, bits [35:32]

Indicates support for the SVE SHA3 instructions. Defined values are:

SHA3	Meaning
0b0000	SVE SHA3 instructions are not implemented.
0b0001	SVE RAX1 instruction is implemented.

All other values are reserved.

FEAT_SVE_SHA3 implements the functionality identified by 0b0001.

Bits [31:24]

Reserved, RES0.

BF16, bits [23:20]

Indicates support for SVE BFloat16 instructions. Defined values are:

BF16	Meaning
0b0000	BFloat16 instructions are not implemented.
0b0001	BFCVT, BFCVTNT, BFDOT, BFMLALB, BFMLALT, and BFMMMLA instructions are implemented.
0b0010	As 0b0001, but the FPCR .EBF field is also supported.

All other values are reserved.

FEAT_BF16 implements the functionality identified by 0b0001.

FEAT_EBF16 implements the functionality identified by 0b0010.

This field must return the same value as [ID_AA64ISAR1_EL1](#).BF16.

If FEAT_SME is implemented, the permitted values are 0b0001 and 0b0010.

Otherwise, from Armv8.6, the only permitted value is 0b0001.

BitPerm, bits [19:16]

Indicates support for SVE bit permute instructions. Defined values are:

BitPerm	Meaning
0b0000	SVE bit permute instructions are not implemented.
0b0001	SVE BDEP, BEXT, and BGRP instructions are implemented.

All other values are reserved.

FEAT_SVE_BitPerm implements the functionality identified by 0b0001.

Bits [15:8]

Reserved, RES0.

AES, bits [7:4]

Indicates support for SVE AES instructions. Defined values are:

AES	Meaning
0b0000	SVE AES instructions are not implemented.
0b0001	SVE AESE, AESD, AESMC, and AESIMC instructions are implemented.
0b0010	As 0b0001, plus SVE PMULLB and PMULLT instructions with 64-bit source.

All other values are reserved.

FEAT_SVE_AES implements the functionality identified by the value 0b0001.

FEAT_SVE_PMULL128 implements the functionality identified by the value 0b0010.

The permitted values are 0b0000 and 0b0010.

SVEver, bits [3:0]

Indicates support for SVE. Defined values are:

SVEver	Meaning
0b0000	SVE instructions are implemented.
0b0001	The SVE and non-optional SVE2 instructions are implemented.

All other values are reserved.

FEAT_SVE2 and FEAT_SME implement the functionality identified by the value 0b0001.

From Armv9, if FEAT_SME is implemented, the only permitted value is 0b0001. This value indicates that SVE and SVE2 instructions are implemented when the PE is in Streaming SVE mode.

Irrespective of the value of ID_AA64ZFR0_EL1.SVEver, when the PE is in Streaming SVE mode, software should not attempt to execute any of the SVE and SVE2 instructions that are illegal in Streaming SVE mode.

Accessing the ID_AA64ZFR0_EL1

Accesses to this register use the following encodings in the instruction encoding space:

MRS <Xt>, ID_AA64ZFR0_EL1

op0	op1	CRn	CRm	op2
0b11	0b000	0b0000	0b0100	0b100

```

1  if PSTATE.EL == EL0 then
2      if IsFeatureImplemented(FEAT_IDST) then
3          if EL2Enabled() && HCR_EL2.TGE == '1' then
4              AArch64.SystemAccessTrap(EL2, 0x18);
5          else
6              AArch64.SystemAccessTrap(EL1, 0x18);
7          else
8              UNDEFINED;
9  elseif PSTATE.EL == EL1 then
10     if EL2Enabled() && (IsFeatureImplemented(FEAT_FGT) || !IsZero(ID_AA64ZFR0_EL1) || boolean
        ↳ IMPLEMENTATION_DEFINED "ID_AA64ZFR0_EL1 trapped by HCR_EL2.TID3") && HCR_EL2.TID3 == '1' then
11         AArch64.SystemAccessTrap(EL2, 0x18);
12     else
13         X[t, 64] = ID_AA64ZFR0_EL1;
14 elseif PSTATE.EL == EL2 then
15     X[t, 64] = ID_AA64ZFR0_EL1;
16 elseif PSTATE.EL == EL3 then
17     X[t, 64] = ID_AA64ZFR0_EL1;

```

E3.2.14 MPAM2_EL2, MPAM2 Register (EL2)

The MPAM2_EL2 characteristics are:

Purpose

Holds information to generate MPAM labels for memory requests when executing at EL2.

Attributes

MPAM2_EL2 is a 64-bit register.

Configuration

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

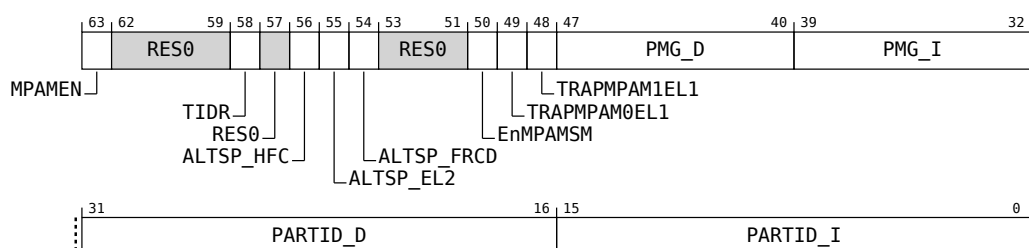
When EL3 is implemented, AArch64 system register MPAM2_EL2 bit [63] is architecturally mapped to AArch64 system register MPAM3_EL3[63].

AArch64 system register MPAM2_EL2 bit [63] is architecturally mapped to AArch64 system register MPAM1_EL1[63].

This register is present only when FEAT_MPAM is implemented. Otherwise, direct accesses to MPAM2_EL2 are UNDEFINED.

Field descriptions

The MPAM2_EL2 bit assignments are:



MPAMEN, bit [63]

MPAM Enable. MPAM is enabled when MPAMEN == 1. When disabled, all PARTIDs and PMGs are output as their default value in the corresponding ID space.

MPAMEN	Meaning
0b0	The default PARTID and default PMG are output in MPAM information from all Exception levels.
0b1	MPAM information is output based on the MPAMn_ELx register for ELn according to the MPAM configuration.

If EL3 is not implemented, this field is read/write.

If EL3 is implemented, this field is read-only and reads the current value of the read/write MPAM3_EL3.MPAMEN bit.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0b0.

Accessing this field has the following behavior:

- **RW** if !HaveEL(EL3)
- Otherwise, access to this field is **RO**

Bits [62:59]

Reserved, RES0.

TIDR, bit [58]

When (FEAT_MPAMv0p1 is implemented or FEAT_MPAMv1p1 is implemented) and MPAMIDR_EL1.HAS_TIDR == 1:

TIDR traps accesses to MPAMIDR_EL1 from EL1 to EL2.

TIDR	Meaning
0b0	This control does not cause any instructions to be trapped.
0b1	Trap accesses to MPAMIDR_EL1 from EL1 to EL2.

MPAMHCR_EL2.TRAP_MPAMIDR_EL1 == 1 also traps MPAMIDR_EL1 accesses from EL1 to EL2. If either TIDR or TRAP_MPAMIDR_EL1 are 1, accesses are trapped.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

RES0

Bit [57]

Reserved, RES0.

ALTSP_HFC, bit [56]

When FEAT_RME is implemented and MPAMIDR_EL1.HAS_ALTSP == 1:

Hierarchical force of alternative PARTID space controls. When MPAM3_EL3.ALTSP_HEN is 0, ALTSP controls in MPAM2_EL2 have no effect. When MPAM3_EL3.ALTSP_HEN is 1, this bit selects whether the PARTIDs in MPAM1_EL1 and MPAM0_EL1 are in the primary (0) or alternative (1) PARTID space for the security state.

ALTSP_HFC	Meaning
0b0	When MPAM3_EL3.ALTSP_HEN is 1, the PARTID space of MPAM1_EL1.PARTID_I, MPAM1_EL1.PARTID_D, MPAM0_EL1.PARTID_I, and MPAM0_EL1.PARTID_D are in the primary PARTID space for the Security state.
0b1	When MPAM3_EL3.ALTSP_HEN is 1, the PARTID space of MPAM1_EL1.PARTID_I, MPAM1_EL1.PARTID_D, MPAM0_EL1.PARTID_I, and MPAM0_EL1.PARTID_D are in the alternative PARTID space for the Security state.

This control has no effect when MPAM3_EL3.ALTSP_HEN is 0.

For more information, see ‘Alternative PARTID spaces and selection’ in Arm® Architecture Reference Manual Supplement, Memory System Resource Partitioning and Monitoring (MPAM), for Armv8-A (ARM DDI 0598).

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

RES0

ALTSP_EL2, bit [55]

When FEAT_RME is implemented and MPAMIDR_EL1.HAS_ALTSP == 1:

Select alternative PARTID space for PARTIDs in MPAM2_EL2 when MPAM3_EL3.ALTSP_HEN is 1.

ALTSP_EL2	Meaning
0b0	When MPAM3_EL3.ALTSP_HEN is 1, selects the primary PARTID space for MPAM2_EL2.PARTID_I and MPAM2_EL2.PARTID_D.
0b1	When MPAM3_EL3.ALTSP_HEN is 1, selects the alternative PARTID space for MPAM2_EL2.PARTID_I and MPAM2_EL2.PARTID_D.

For more information, see ‘Alternative PARTID spaces and selection’ in Arm® Architecture Reference Manual Supplement, Memory System Resource Partitioning and Monitoring (MPAM), for Armv8-A (ARM DDI 0598).

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

RES0

ALTSP_FRCD, bit [54]

When FEAT_RME is implemented and MPAMIDR_EL1.HAS_ALTSP == 1:

Alternative PARTID forced for PARTIDs in this register.

ALTSP_FRCD	Meaning
0b0	The PARTIDs in this register are using the primary PARTID space.
0b1	The PARTIDs in this register are using the alternative PARTID space.

This bit indicates that a higher Exception level has forced the PARTIDs in this register to use the alternative PARTID space defined for the current Security state. In EL2, it is also 1 when MPAM2_EL2.ALTSP_EL2 is 1.

For more information, see ‘Alternative PARTID spaces and selection’ in Arm® Architecture Reference Manual Supplement, Memory System Resource Partitioning and Monitoring (MPAM), for Armv8-A (ARM DDI 0598).

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Access to this field is **RO**.

Otherwise:

RES0

Bits [53:51]

Reserved, RES0.

EnMPAMSM, bit [50]

When FEAT_SME is implemented:

Traps execution at EL1 of instructions that directly access the [MPAMSM_EL1](#) register to EL2. The exception is reported using ESR_ELx.EC value 0x18.

EnMPAMSM	Meaning
0b0	This control causes execution of these instructions at EL1 to be trapped.
0b1	This control does not cause execution of any instructions to be trapped.

This field has no effect on accesses to [MPAMSM_EL1](#) from EL2 or EL3.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

RES0

TRAPMPAM0EL1, bit [49]

Trap accesses from EL1 to the MPAM0_EL1 register trap to EL2.

TRAPMPAM0EL1	Meaning
0b0	Accesses to MPAM0_EL1 from EL1 are not trapped.
0b1	Accesses to MPAM0_EL1 from EL1 are trapped to EL2.

The reset behavior of this field is:

- On a Warm reset:
 - When EL3 is not implemented, this field resets to 0b1.
 - When EL3 is implemented, this field resets to an architecturally UNKNOWN value.

TRAPMPAM1EL1, bit [48]

Trap accesses from EL1 to the MPAM1_EL1 register trap to EL2.

TRAPMPAM1EL1	Meaning
0b0	Accesses to MPAM1_EL1 from EL1 are not trapped.
0b1	Accesses to MPAM1_EL1 from EL1 are trapped to EL2.

The reset behavior of this field is:

- On a Warm reset:
 - When EL3 is not implemented, this field resets to 0b1.
 - When EL3 is implemented, this field resets to an architecturally UNKNOWN value.

PMG_D, bits [47:40]

Performance monitoring group for data accesses.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

PMG_I, bits [39:32]

Performance monitoring group for instruction accesses.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

PARTID_D, bits [31:16]

Partition ID for data accesses, including load and store accesses, made from EL2.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

PARTID_I, bits [15:0]

Partition ID for instruction accesses made from EL2.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing the MPAM2_EL2

None of the fields in this register are permitted to be cached in a TLB.

Accesses to this register use the following encodings in the instruction encoding space:

MRS <Xt>, MPAM2_EL2

op0	op1	CRn	CRm	op2
0b11	0b100	0b1010	0b0101	0b000

```
1 if PSTATE.EL == EL0 then
2   UNDEFINED;
```

```

3  elsif PSTATE.EL == EL1 then
4      if EL2Enabled() && HCR_EL2.NV == '1' then
5          if HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
6              if Halted() && EDSCR.SDD == '1' then
7                  UNDEFINED;
8              else
9                  AArch64.SystemAccessTrap(EL3, 0x18);
10             else
11                 AArch64.SystemAccessTrap(EL2, 0x18);
12             else
13                 UNDEFINED;
14  elsif PSTATE.EL == EL2 then
15      if HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
16          if Halted() && EDSCR.SDD == '1' then
17              UNDEFINED;
18          else
19              AArch64.SystemAccessTrap(EL3, 0x18);
20          else
21              X[t, 64] = MPAM2_EL2;
22  elsif PSTATE.EL == EL3 then
23      X[t, 64] = MPAM2_EL2;

```

MSR MPAM2_EL2, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b100	0b1010	0b0101	0b000

```

1  if PSTATE.EL == EL0 then
2      UNDEFINED;
3  elsif PSTATE.EL == EL1 then
4      if EL2Enabled() && HCR_EL2.NV == '1' then
5          if HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
6              if Halted() && EDSCR.SDD == '1' then
7                  UNDEFINED;
8              else
9                  AArch64.SystemAccessTrap(EL3, 0x18);
10             else
11                 AArch64.SystemAccessTrap(EL2, 0x18);
12             else
13                 UNDEFINED;
14  elsif PSTATE.EL == EL2 then
15      if HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
16          if Halted() && EDSCR.SDD == '1' then
17              UNDEFINED;
18          else
19              AArch64.SystemAccessTrap(EL3, 0x18);
20          else
21              MPAM2_EL2 = X[t, 64];
22  elsif PSTATE.EL == EL3 then
23      MPAM2_EL2 = X[t, 64];

```

MRS <Xt>, MPAM1_EL1

op0	op1	CRn	CRm	op2
0b11	0b000	0b1010	0b0101	0b000

```

1  if PSTATE.EL == EL0 then
2      UNDEFINED;
3  elsif PSTATE.EL == EL1 then
4      if HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
5          if Halted() && EDSCR.SDD == '1' then
6              UNDEFINED;

```

```

7      else
8          AArch64.SystemAccessTrap(EL3, 0x18);
9      elseif EL2Enabled() && MPAM2_EL2.TRAPMPAM1EL1 == '1' then
10         AArch64.SystemAccessTrap(EL2, 0x18);
11      elseif EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '111' then
12         X[t, 64] = NVMem[0x900];
13      else
14         X[t, 64] = MPAM1_EL1;
15  elseif PSTATE.EL == EL2 then
16      if HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
17         if Halted() && EDSCR.SDD == '1' then
18             UNDEFINED;
19         else
20             AArch64.SystemAccessTrap(EL3, 0x18);
21         elseif HCR_EL2.E2H == '1' then
22             X[t, 64] = MPAM2_EL2;
23         else
24             X[t, 64] = MPAM1_EL1;
25  elseif PSTATE.EL == EL3 then
26      X[t, 64] = MPAM1_EL1;

```

MSR MPAM1_EL1, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b000	0b1010	0b0101	0b000

```

1  if PSTATE.EL == EL0 then
2      UNDEFINED;
3  elseif PSTATE.EL == EL1 then
4      if HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
5         if Halted() && EDSCR.SDD == '1' then
6             UNDEFINED;
7         else
8             AArch64.SystemAccessTrap(EL3, 0x18);
9         elseif EL2Enabled() && MPAM2_EL2.TRAPMPAM1EL1 == '1' then
10            AArch64.SystemAccessTrap(EL2, 0x18);
11        elseif EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '111' then
12            NVMem[0x900] = X[t, 64];
13        else
14            MPAM1_EL1 = X[t, 64];
15  elseif PSTATE.EL == EL2 then
16      if HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
17         if Halted() && EDSCR.SDD == '1' then
18             UNDEFINED;
19         else
20             AArch64.SystemAccessTrap(EL3, 0x18);
21         elseif HCR_EL2.E2H == '1' then
22             MPAM2_EL2 = X[t, 64];
23         else
24             MPAM1_EL1 = X[t, 64];
25  elseif PSTATE.EL == EL3 then
26      MPAM1_EL1 = X[t, 64];

```

E3.2.15 SCR_EL3, Secure Configuration Register

The SCR_EL3 characteristics are:

Purpose

Defines the configuration of the current Security state. It specifies:

- The Security state of EL0, EL1, and EL2. The Security state is Secure, Non-secure, or Realm.
- The Execution state at lower Exception levels.
- Whether IRQ, FIQ, SError interrupts, and External abort exceptions are taken to EL3.
- Whether various operations are trapped to EL3.

Attributes

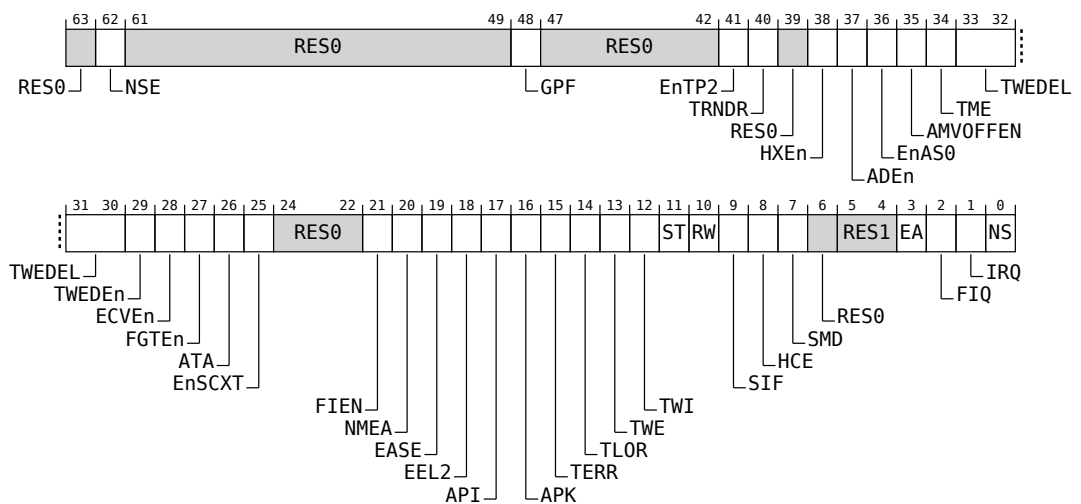
SCR_EL3 is a 64-bit register.

Configuration

This register is present only when EL3 is implemented. Otherwise, direct accesses to SCR_EL3 are UNDEFINED.

Field descriptions

The SCR_EL3 bit assignments are:



Bit [63]

Reserved, RES0.

NSE, bit [62]

When FEAT_RME is implemented:

This field, evaluated with SCR_EL3.NS, selects the Security state of EL2 and lower Exception levels.

For a description of the values derived by evaluating NS and NSE together, see SCR_EL3.NS.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

RES0

Bits [61:49]

Reserved, RES0.

GPF, bit [48]

When FEAT_RME is implemented:

Controls the reporting of Granule protection faults at EL0, EL1 and EL2.

GPF	Meaning
0b0	This control does not cause exceptions to be routed from EL0, EL1 or EL2 to EL3.
0b1	GPFs at EL0, EL1 and EL2 are routed to EL3 and reported as Granule Protection Check exceptions.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

RES0

Bits [47:42]

Reserved, RES0.

EnTP2, bit [41]

When FEAT_SME is implemented:

Traps instructions executed at EL2, EL1, and EL0 that access [TPIDR2_EL0](#) to EL3. The exception is reported using ESR_ELx.EC value 0x18.

EnTP2	Meaning
0b0	This control causes execution of these instructions at EL2, EL1, and EL0 to be trapped.
0b1	This control does not cause execution of any instructions to be trapped.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

RES0

TRNDR, bit [40]

When FEAT_RNG_TRAP is implemented:

Controls trapping of reads of RNDR and RNDRRS. The exception is reported using ESR_ELx.EC value 0x18.

TRNDR	Meaning
0b0	<p>This control does not cause RNDR and RNDRRS to be trapped.</p> <p>When FEAT_RNG is implemented:</p> <ul style="list-style-type: none"> ID_AA64ISAR0_EL1.RNDR returns the value 0b0001. <p>When FEAT_RNG is not implemented:</p> <ul style="list-style-type: none"> ID_AA64ISAR0_EL1.RNDR returns the value 0b0000. MRS reads of RNDR and RNDRRS are UNDEFINED.
0b1	<p>ID_AA64ISAR0_EL1.RNDR returns the value 0b0001.</p> <p>Any attempt to read RNDR or RNDRRS is trapped to EL3.</p>

When FEAT_RNG is not implemented, Arm recommends that SCR_EL3.TRNDR is initialized before entering Exception levels below EL3 and not subsequently changed.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0b0.

Otherwise:

RES0

Bit [39]

Reserved, RES0.

HXEn, bit [38]

When FEAT_HCX is implemented:

Enables access to the [HCRX_EL2](#) register at EL2 from EL3.

HXEn	Meaning
0b0	Accesses at EL2 to HCRX_EL2 are trapped to EL3. Indirect reads of HCRX_EL2 return 0.
0b1	This control does not cause any instructions to be trapped.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

RES0

ADEn, bit [37]

When FEAT_LS64_ACCDATA is implemented:

Enables access to the ACCDATA_EL1 register at EL1 and EL2.

ADEn	Meaning
0b0	Accesses to ACCDATA_EL1 at EL1 and EL2 are trapped to EL3, unless the accesses are trapped to EL2 by the EL2 fine-grained trap.
0b1	This control does not cause accesses to ACCDATA_EL1 to be trapped.

If the [HFGWTR_EL2.nACCDATA_EL1](#) or [HFGTR_EL2.nACCDATA_EL1](#) traps are enabled, they take priority over this trap.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

RES0

EnAS0, bit [36]

When FEAT_LS64_ACCDATA is implemented:

Traps execution of an ST64BV0 instruction at EL0, EL1, or EL2 to EL3.

EnAS0	Meaning
0b0	EL0 execution of an ST64BV0 instruction is trapped to EL3, unless it is trapped to EL1 by SCTLR_EL1.EnAS0 , or to EL2 by either HCRX_EL2.EnAS0 or SCTLR_EL2.EnAS0 . EL1 execution of an ST64BV0 instruction is trapped to EL3, unless it is trapped to EL2 by HCRX_EL2.EnAS0 . EL2 execution of an ST64BV0 instruction is trapped to EL3.
0b1	This control does not cause any instructions to be trapped.

A trap of an ST64BV0 instruction is reported using an ESR_ELx.EC value of 0x0A, with an ISS code of 0x0000001.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

RES0

AMVOFFEN, bit [35]

When FEAT_AMUv1p1 is implemented:

Activity Monitors Virtual Offsets Enable.

AMVOFFEN	Meaning
0b0	Accesses to AMEVCNTVOFF0<n>_EL2 and AMEVCNTVOFF1<n>_EL2 at EL2 are trapped to EL3. Indirect reads of the virtual offset registers are zero.

AMVOFFEN	Meaning
0b1	Accesses to AMEVCNTVOFF0<n>_EL2 and AMEVCNTVOFF1<n>_EL2 are not affected by this field.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

RES0

TME, bit [34]

When FEAT_TME is implemented:

Enables access to the TSTART, TCOMMIT, TTEST and TCANCEL instructions at EL0, EL1 and EL2.

TME	Meaning
0b0	EL0, EL1 and EL2 accesses to TSTART, TCOMMIT, TTEST and TCANCEL instructions are UNDEFINED.
0b1	This control does not cause any instruction to be UNDEFINED.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

RES0

TWEDEL, bits [33:30]

When FEAT_TWED is implemented:

TWE Delay. A 4-bit unsigned number that, when SCR_EL3.TWEDEn is 1, encodes the minimum delay in taking a trap of WFE* caused by SCR_EL3.TWE as $2^{(TWEDEL + 8)}$ cycles.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

RES0

TWEDEn, bit [29]

When FEAT_TWED is implemented:

TWE Delay Enable. Enables a configurable delayed trap of the WFE* instruction caused by SCR_EL3.TWE.

Traps are reported using an ESR_ELx.EC value of 0x01.

TWEDEn	Meaning
0b0	The delay for taking the trap is IMPLEMENTATION DEFINED.
0b1	The delay for taking the trap is at least the number of cycles defined in SCR_EL3.TWEDEL.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

RES0

ECVEn, bit [28]

When FEAT_ECV is implemented:

ECV Enable. Enables access to the CNTPOFF_EL2 register.

ECVEn	Meaning
0b0	EL2 accesses to CNTPOFF_EL2 are trapped to EL3, and the value of CNTPOFF_EL2 is treated as 0 for all purposes other than direct reads or writes to the register from EL3.
0b1	EL2 accesses to CNTPOFF_EL2 are not trapped to EL3 by this mechanism.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

RES0

FGTEn, bit [27]

When FEAT_FGT is implemented:

Fine-Grained Traps Enable. When EL2 is implemented, enables the traps to EL2 controlled by HAFGRTR_EL2, HDFGRTR_EL2, HDFGWTR_EL2, [HFGTR_EL2](#), HFGITR_EL2, and [HFGWTR_EL2](#), and controls access to those registers.

If EL2 is not implemented but EL3 is implemented, FEAT_FGT implements the MDCR_EL3.TDCC traps.

FGTEn	Meaning
0b0	EL2 accesses to HAFGRTR_EL2, HDFGRTR_EL2, HDFGWTR_EL2, HFGTR_EL2 , HFGITR_EL2 and HFGWTR_EL2 registers are trapped to EL3, and the traps to EL2 controlled by those registers are disabled.

FGTE _n	Meaning
0b1	EL2 accesses to HAFGRTR_EL2, HDFGRTR_EL2, HDFGWTR_EL2, HFGTR_EL2 , HFGITR_EL2 and HFGWTR_EL2 registers are not trapped to EL3 by this mechanism.

Traps caused by accesses to the fine-grained trap registers are reported using an ESR_EL_x.EC value of 0x18 and its associated ISS.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

RES0

ATA, bit [26]

When FEAT_MTE2 is implemented:

Allocation Tag Access. Controls access at EL2, EL1 and EL0 to Allocation Tags.

ATA	Meaning
0b0	Access to Allocation Tags is prevented. Accesses at EL1 and EL2 to GCR_EL1, RGSRR_EL1, TFSRR_EL1, TFSRR_EL2 or TFSRE0_EL1 that are not UNDEFINED or trapped to a lower Exception level are trapped to EL3. Accesses at EL2 to TFSRR_EL12 that are not UNDEFINED are trapped to EL3.
0b1	This control does not prevent access to Allocation Tags.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

RES0

EnSCXT, bit [25]

When FEAT_CSV2_2 is implemented or FEAT_CSV2_1p2 is implemented:

Enable access to the SCXTNUM_EL2, SCXTNUM_EL1, and SCXTNUM_EL0 registers.

EnSCXT	Meaning
0b0	Accesses at EL0, EL1 and EL2 to SCXTNUM_EL0, SCXTNUM_EL1, or SCXTNUM_EL2 registers are trapped to EL3 if they are not trapped by a higher priority exception, and the values of these registers are treated as 0.
0b1	This control does not cause any accesses to be trapped, or register values to be treated as 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

RES0

Bits [24:22]

Reserved, RES0.

FIEN, bit [21]

When FEAT_RASv1p1 is implemented:

Fault Injection enable. Trap accesses to the registers ERXPFPGCDN_EL1, ERXPFPGCTL_EL1, and ERXPFGF_EL1 from EL1 and EL2 to EL3, reported using an ESR_ELx.EC value of 0x18.

FIEN	Meaning
0b0	Accesses to the specified registers from EL1 and EL2 generate a Trap exception to EL3.
0b1	This control does not cause any instructions to be trapped.

If EL3 is not implemented, the Effective value of SCR_EL3.FIEN is 0b1.

If ERRIDR_EL1.NUM is zero, meaning no error records are implemented, or no error record accessible using System registers is owned by a node that implements the RAS Common Fault Injection Model Extension, then this bit might be RES0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

RES0

NMEA, bit [20]

When FEAT_DoubleFault is implemented:

Non-maskable External Aborts. When [SCR_EL3.EA == 1](#), controls whether PSTATE.A masks SError interrupts at EL3.

NMEA	Meaning
0b0	If SCR_EL3.EA == 1 , asserted SError interrupts are not taken at EL3 if PSTATE.A == 1.
0b1	If SCR_EL3.EA == 1 , asserted SError interrupts are taken at EL3 regardless of the value of PSTATE.A.

When SCR_EL3.EA == 0:

- Asserted SError interrupts are not taken at EL3 regardless of the value of PSTATE.A and this field.
- This field is ignored and its Effective value is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0b0.

Otherwise:

RES0

EASE, bit [19]

When FEAT_DoubleFault is implemented:

External aborts to SError interrupt vector.

EASE	Meaning
0b0	Synchronous External abort exceptions taken to EL3 are taken to the appropriate synchronous exception vector offset from VBAR_EL3.
0b1	Synchronous External abort exceptions taken to EL3 are taken to the appropriate SError interrupt vector offset from VBAR_EL3.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0b0.

Otherwise:

RES0

EEL2, bit [18]

When FEAT_SEL2 is implemented:

Secure EL2 Enable.

EEL2	Meaning
0b0	All behaviors associated with Secure EL2 are disabled. All registers, including timer registers, defined by FEAT_SEL2 are UNDEFINED, and those timers are disabled.
0b1	All behaviors associated with Secure EL2 are enabled.

When the value of this bit is 1, then:

- When SCR_EL3.NS == 0, the SCR_EL3.RW bit is treated as 1 for all purposes other than reading or writing the register.
- If Secure EL1 is using AArch32, then any of the following operations, executed in Secure EL1, is trapped to Secure EL2, using the EC value of ESR_EL2.EC== 0x3 :
 - A read or write of the SCR.
 - A read or write of the NSACR.
 - A read or write of the MVBAR.
 - A read or write of the SDCR.
 - Execution of an ATS12NSO** instruction.

- If Secure EL1 is using AArch32, then any of the following operations, executed in Secure EL1, is trapped to Secure EL2 using the EC value of ESR_EL2.EC == 0x0 :
 - Execution of an SRS instruction that uses R13_mon.
 - Execution of an MRS (Banked register) or MSR (Banked register) instruction that would access SPSR_mon, R13_mon, or R14_mon.

If the Effective value of SCR_EL3.EEL2 is 0, then these operations executed in Secure EL1 using AArch32 are trapped to EL3.

A Secure only implementation that does not implement EL3 but implements EL2, behaves as if SCR_EL3.EEL2 == 1.

This bit is permitted to be cached in a TLB.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

RES0

API, bit [17]

When FEAT_SEL2 is implemented and FEAT_PAuth is implemented

API, bit [0] of bit [17]

Controls the use of the following instructions related to Pointer Authentication. Traps are reported using an ESR_ELx.EC value of 0x09:

- PACGA, which is always enabled.
- AUTDA, AUTDB, AUTDZA, AUTDZB, AUTIA, AUTIA1716, AUTIASP, AUTIAZ, AUTIB, AUTIB1716, AUTIBSP, AUTIBZ, AUTIZA, AUTIZB, PACDA, PACDB, PACDZA, PACDZB, PACIA, PACIA1716, PACIASP, PACIAZ, PACIB, PACIB1716, PACIBSP, PACIBZ, PACIZA, PACIZB, RETAA, RETAB, BRAA, BRAB, BLRAA, BLRAB, BRAAZ, BRABZ, BLRAAZ, BLRABZ, ERETAA, ERETAB, LDRAA and LDRAB when:
 - In EL0, when HCR_EL2.TGE == 0 or HCR_EL2.E2H == 0, and the associated [SCTLR_EL1.En<N><M>](#) == 1.
 - In EL0, when HCR_EL2.TGE == 1 and HCR_EL2.E2H == 1, and the associated [SCTLR_EL2.En<N><M>](#) == 1.
 - In EL1, when the associated [SCTLR_EL1.En<N><M>](#) == 1.
 - In EL2, when the associated [SCTLR_EL2.En<N><M>](#) == 1.

API	Meaning
0b0	The use of any instruction related to pointer authentication in any Exception level except EL3 when the instructions are enabled are trapped to EL3 unless they are trapped to EL2 as a result of the HCR_EL2.API bit.
0b1	This control does not cause any instructions to be trapped.

An instruction is trapped only if Pointer Authentication is enabled for that instruction, for more information, see System register control of pointer authentication .

If FEAT_PAuth is implemented but EL3 is not implemented, the system behaves as if this bit is 1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

When FEAT_SEL2 is not implemented and FEAT_PAuth is implemented

API, bit [0] of bit [17]

Controls the use of instructions related to Pointer Authentication:

- PACGA.
- AUTDA, AUTDB, AUTDZA, AUTDZB, AUTIA, AUTIA1716, AUTIASP, AUTIAZ, AUTIB, AUTIB1716, AUTIBSP, AUTIBZ, AUTIZA, AUTIZB, PACDA, PACDB, PACDZA, PACDZB, PACIA, PACIA1716, PACIASP, PACIAZ, PACIB, PACIB1716, PACIBSP, PACIBZ, PACIZA, PACIZ, RETAA, RETAB, BRAA, BRAB, BLRAA, BLRAB, BRAAZ, BRABZ, BLRAAZ, BLRABZ, ERETAA, ERETAB, LDRAA and LDRAB when:
 - In Non-secure EL0, when HCR_EL2.TGE == 0 or HCR_EL2.E2H == 0, and the associated [SCTLR_EL1.En<N><M> == 1](#).
 - In Non-secure EL0, when HCR_EL2.TGE == 1 and HCR_EL2.E2H == 1, and the associated [SCTLR_EL2.En<N><M> == 1](#).
 - In Secure EL0, when the associated [SCTLR_EL1.En<N><M> == 1](#).
 - In Secure or Non-secure EL1, when the associated [SCTLR_EL1.En<N><M> == 1](#).
 - In EL2, when the associated [SCTLR_EL2.En<N><M> == 1](#).

API	Meaning
0b0	The use of any instruction related to pointer authentication in any Exception level except EL3 when the instructions are enabled are trapped to EL3 unless they are trapped to EL2 as a result of the HCR_EL2.API bit.
0b1	This control does not cause any instructions to be trapped.

If FEAT_PAuth is implemented but EL3 is not implemented, the system behaves as if this bit is 1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

RES0

APK, bit [16]

When FEAT_PAuth is implemented:

Trap registers holding “key” values for Pointer Authentication. Traps accesses to the following registers, using an ESR_ELx.EC value of 0x18, from EL1 or EL2 to EL3 unless they are trapped to EL2 as a result of the HCR_EL2.APK bit or other traps:

- APIAKeyLo_EL1, APIAKeyHi_EL1, APIBKeyLo_EL1, APIBKeyHi_EL1.
- APDAKeyLo_EL1, APDAKeyHi_EL1, APDBKeyLo_EL1, APDBKeyHi_EL1.
- APGAKeyLo_EL1, and APGAKeyHi_EL1.

APK	Meaning
0b0	Access to the registers holding “key” values for pointer authentication from EL1 or EL2 are trapped to EL3 unless they are trapped to EL2 as a result of the HCR_EL2.APK bit or other traps.
0b1	This control does not cause any instructions to be trapped.

For more information, see System register control of pointer authentication .

If FEAT_PAuth is implemented but EL3 is not implemented, the system behaves as if this bit is 1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

RES0

TERR, bit [15]

When FEAT_RAS is implemented:

Trap Error record accesses. Accesses to the RAS ERR* and RAS ERX* registers from EL1 and EL2 to EL3 are trapped as follows:

- Accesses from EL1 and EL2 using AArch64 to the following registers are trapped and reported using an ESR_ELx.EC value of 0x18:
 - ERRIDR_EL1, ERRSELR_EL1, ERXADDR_EL1, ERXCTLR_EL1, ERXFR_EL1, ERXMISC0_EL1, ERXMISC1_EL1, and ERXSTATUS_EL1.
- If FEAT_RASv1p1 is implemented, accesses from EL1 and EL2 using AArch64 to ERXMISC2_EL1, and ERXMISC3_EL1, are trapped and reported using an ESR_ELx.EC value of 0x18.
- Accesses from EL1 and EL2 using AArch32, to the following registers are trapped and reported using an ESR_ELx.EC value of 0x03:
 - ERRIDR, ERRSELR, ERXADDR, ERXADDR2, ERXCTLR, ERXCTLR2, ERXFR, ERXFR2, ERXMISC0, ERXMISC1, ERXMISC2, ERXMISC3, and ERXSTATUS.
- If FEAT_RASv1p1 is implemented, accesses from EL1 and EL2 using AArch32 to the following registers are trapped and reported using an ESR_ELx.EC value of 0x03:
 - ERXMISC4, ERXMISC5, ERXMISC6, and ERXMISC7.

TERR	Meaning
0b0	This control does not cause any instructions to be trapped.
0b1	Accesses to the specified registers from EL1 and EL2 generate a Trap exception to EL3.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

RES0

TLOR, bit [14]

When FEAT_LOR is implemented:

Trap LOR registers. Traps accesses to the LORSA_EL1, LOREA_EL1, LORN_EL1, LORC_EL1, and LORID_EL1 registers from EL1 and EL2 to EL3, unless the access has been trapped to EL2.

TLOR	Meaning
0b0	This control does not cause any instructions to be trapped.
0b1	EL1 and EL2 accesses to the LOR registers that are not UNDEFINED are trapped to EL3, unless it is trapped HCR_EL2.TLOR.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

RES0

TWE, bit [13]

Traps EL2, EL1, and EL0 execution of WFE instructions to EL3, from any Security state and both Execution states, reported using an ESR_ELx.EC value of 0x01.

When FEAT_WFxT is implemented, this trap also applies to the WFET instruction.

TWE	Meaning
0b0	This control does not cause any instructions to be trapped.
0b1	Any attempt to execute a WFE instruction at any Exception level lower than EL3 is trapped to EL3, if the instruction would otherwise have caused the PE to enter a low-power state and it is not trapped by SCTLR.nTWE, HCR.TWE, SCTLR_EL1.nTWE , SCTLR_EL2.nTWE , or HCR_EL2.TWE.

In AArch32 state, the attempted execution of a conditional WFE instruction is only trapped if the instruction passes its condition code check.

Since a WFE or WFI can complete at any time, even without a Wakeup event, the traps on WFE or WFI are not guaranteed to be taken, even if the WFE or WFI is executed when there is no Wakeup event. The only guarantee is that if the instruction does not complete in finite time in the absence of a Wakeup event, the trap will be taken.

For more information about when WFE instructions can cause the PE to enter a low-power state, see ‘Wait for Event mechanism and Send event’.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

TWI, bit [12]

Traps EL2, EL1, and EL0 execution of WFI instructions to EL3, from any Security state and both Execution states, reported using an ESR_ELx.EC value of 0x01.

When FEAT_WFxT is implemented, this trap also applies to the WFIT instruction.

TWI	Meaning
0b0	This control does not cause any instructions to be trapped.
0b1	Any attempt to execute a WFI instruction at any Exception level lower than EL3 is trapped to EL3, if the instruction would otherwise have caused the PE to enter a low-power state and it is not trapped by SCTLR.nTWI, HCR.TWI, SCTLR_EL1.nTWI , SCTLR_EL2.nTWI , or HCR_EL2.TWI.

In AArch32 state, the attempted execution of a conditional WFI instruction is only trapped if the instruction passes its condition code check.

Since a WFE or WFI can complete at any time, even without a Wakeup event, the traps on WFE or WFI are not guaranteed to be taken, even if the WFE or WFI is executed when there is no Wakeup event. The only guarantee is that if the instruction does not complete in finite time in the absence of a Wakeup event, the trap will be taken.

For more information about when WFI instructions can cause the PE to enter a low-power state, see ‘Wait for Interrupt’.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

ST, bit [11]

Traps Secure EL1 accesses to the Counter-timer Physical Secure timer registers to EL3, from AArch64 state only, reported using an ESR_ELx.EC value of 0x18.

ST	Meaning
0b0	Secure EL1 using AArch64 accesses to the CNTPS_TVAL_EL1, CNTPS_CTL_EL1, and CNTPS_CVAL_EL1 are trapped to EL3 when Secure EL2 is disabled. If Secure EL2 is enabled, the behavior is as if the value of this field was 0b1.
0b1	This control does not cause any instructions to be trapped.

Accesses to the Counter-timer Physical Secure timer registers are always enabled at EL3. These registers are not accessible at EL0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

RW, bit [10]

When EL1 is capable of using AArch32 or EL2 is capable of using AArch32:

Execution state control for lower Exception levels.

RW	Meaning
0b0	Lower levels are all AArch32.
0b1	<p>The next lower level is AArch64.</p> <p>If EL2 is present:</p> <ul style="list-style-type: none"> • EL2 is AArch64. • EL2 controls EL1 and EL0 behaviors. <p>If EL2 is not present:</p> <ul style="list-style-type: none"> • EL1 is AArch64. • EL0 is determined by the Execution state described in the current process state when executing at EL0.

If AArch32 state is supported by the implementation at EL1, SCR_EL3.NS == 1 and AArch32 state is not supported by the implementation at EL2, the Effective value of this bit is 1.

If AArch32 state is supported by the implementation at EL1, FEAT_SEL2 is implemented and SCR_EL3.{EEL2, NS} == {1, 0}, the Effective value of this bit is 1.

This bit is permitted to be cached in a TLB.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

RAO/WI

SIF, bit [9]

Secure instruction fetch. When the PE is in Secure state, this bit disables instruction fetch from Non-secure memory.

SIF	Meaning
0b0	Secure state instruction fetches from Non-secure memory are permitted.
0b1	Secure state instruction fetches from Non-secure memory are not permitted.

When FEAT_PAN3 is implemented, it is IMPLEMENTATION DEFINED whether SCR_EL3.SIF is also used to determine instruction access permission for the purpose of PAN.

This bit is permitted to be cached in a TLB.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

HCE, bit [8]

Hypervisor Call instruction enable. Enables HVC instructions at EL3 and, if EL2 is enabled in the current Security state, at EL2 and EL1, in both Execution states, reported using an ESR_ELx.EC value of 0x00.

HCE	Meaning
0b0	HVC instructions are UNDEFINED.
0b1	HVC instructions are enabled at EL3, EL2, and EL1.

HVC instructions are always UNDEFINED at EL0 and, if Secure EL2 is disabled, at Secure EL1. Any resulting exception is taken from the current Exception level to the current Exception level.

If EL2 is not implemented, this bit is RES0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

SMD, bit [7]

Secure Monitor Call disable. Disables SMC instructions at EL1 and above, from any Security state and both Execution states, reported using an ESR_ELx.EC value of 0x00.

SMD	Meaning
0b0	<small>SMC</small> instructions are enabled at EL3, EL2 and EL1.
0b1	<small>SMC</small> instructions are UNDEFINED.

SMC instructions are always UNDEFINED at EL0. Any resulting exception is taken from the current Exception level to the current Exception level.

If HCR_EL2.TSC or HCR.TSC traps attempted EL1 execution of SMC instructions to EL2, that trap has priority over this disable.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bit [6]

Reserved, RES0.

Bits [5:4]

Reserved, RES1.

EA, bit [3]

External Abort and SError interrupt routing.

EA	Meaning
0b0	When executing at Exception levels below EL3, External aborts and SError interrupts are not taken to EL3. In addition, when executing at EL3: <ul style="list-style-type: none"> • SError interrupts are not taken. • External aborts are taken to EL3.

EA	Meaning
0b1	When executing at any Exception level, External aborts and SError interrupts are taken to EL3.

For more information, see ‘Asynchronous exception routing’.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

FIQ, bit [2]

Physical FIQ Routing.

FIQ	Meaning
0b0	When executing at Exception levels below EL3, physical FIQ interrupts are not taken to EL3. When executing at EL3, physical FIQ interrupts are not taken.
0b1	When executing at any Exception level, physical FIQ interrupts are taken to EL3.

For more information, see ‘Asynchronous exception routing’.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

IRQ, bit [1]

Physical IRQ Routing.

IRQ	Meaning
0b0	When executing at Exception levels below EL3, physical IRQ interrupts are not taken to EL3. When executing at EL3, physical IRQ interrupts are not taken.
0b1	When executing at any Exception level, physical IRQ interrupts are taken to EL3.

For more information, see ‘Asynchronous exception routing’.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

NS, bit [0]

When FEAT_RME is implemented

NS, bit [0] of bit [0]

Non-secure bit. This field is used in combination with SCR_EL3.NSE to select the Security state of EL2 and lower Exception levels.

NSE	NS	Meaning
0b0	0b0	Secure.
0b0	0b1	Non-secure.
0b1	0b0	Reserved.
0b1	0b1	Realm.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise

NS, bit [0] of bit [0]

Non-secure bit.

NS	Meaning
0b0	Indicates that EL0 and EL1 are in Secure state.
0b1	Indicates that Exception levels lower than EL3 are in Non-secure state, so memory accesses from those Exception levels cannot access Secure memory.

When SCR_EL3.{EEL2, NS} == {1, 0}, then EL2 is using AArch64 and in Secure state.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing the SCR_EL3

Accesses to this register use the following encodings in the instruction encoding space:

MRS <Xt>, SCR_EL3

op0	op1	CRn	CRm	op2
0b11	0b110	0b0001	0b0001	0b000

```

1 if PSTATE.EL == EL0 then
2     UNDEFINED;
3 elseif PSTATE.EL == EL1 then

```

```
4      UNDEFINED;  
5  elseif PSTATE.EL == EL2 then  
6      UNDEFINED;  
7  elseif PSTATE.EL == EL3 then  
8      X[t, 64] = SCR_EL3;
```

MSR SCR_EL3, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b110	0b0001	0b0001	0b000

```
1  if PSTATE.EL == EL0 then  
2      UNDEFINED;  
3  elseif PSTATE.EL == EL1 then  
4      UNDEFINED;  
5  elseif PSTATE.EL == EL2 then  
6      UNDEFINED;  
7  elseif PSTATE.EL == EL3 then  
8      SCR_EL3 = X[t, 64];
```

E3.2.16 SCTL_EL1, System Control Register (EL1)

The SCTL_EL1 characteristics are:

Purpose

Provides top level control of the system, including its memory system, at EL1 and EL0.

Attributes

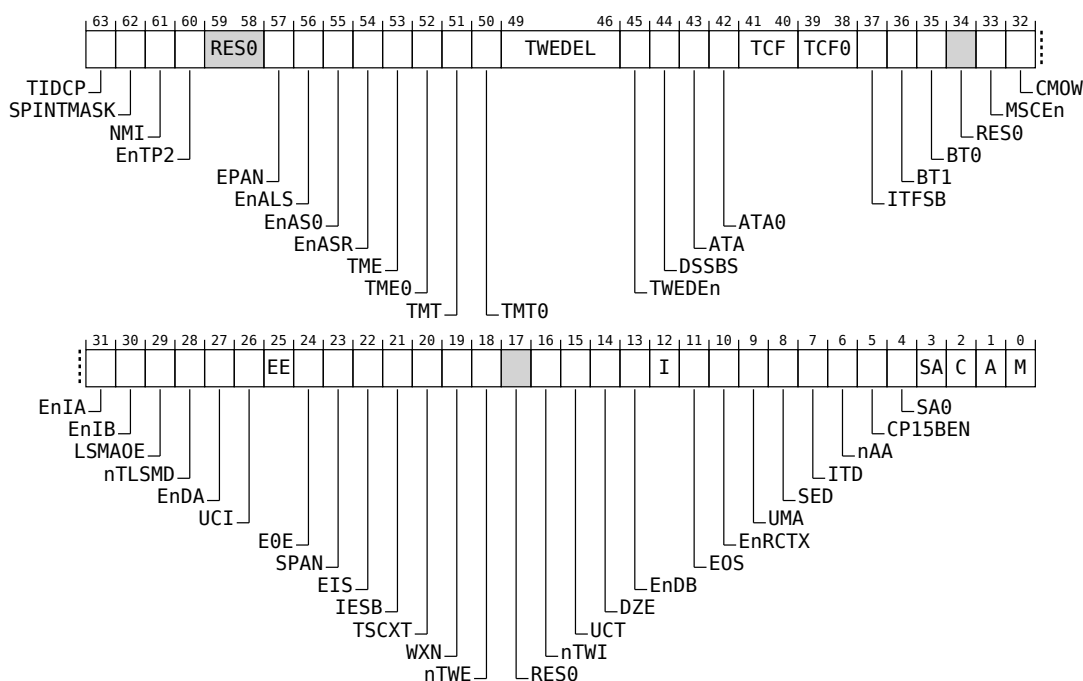
SCTL_EL1 is a 64-bit register.

Configuration

AArch64 system register SCTL_EL1 bits [31:0] are architecturally mapped to AArch32 system register SCTL[31:0].

Field descriptions

The SCTL_EL1 bit assignments are:



TIDCP, bit [63]

When FEAT_TIDCP1 is implemented:

Trap IMPLEMENTATION DEFINED functionality. When HCR_EL2.{E2H, TGE} != {1, 1}, traps EL0 accesses to the encodings reserved for IMPLEMENTATION DEFINED functionality to EL1.

TIDCP	Meaning
0b0	No instructions accessing the System register or System instruction spaces are trapped by this mechanism.

TIDCP	Meaning
0b1	<p>Instructions accessing the following System register or System instruction spaces are trapped to EL1 by this mechanism:</p> <ul style="list-style-type: none"> In AArch64 state, EL0 access to the encodings in the following reserved encoding spaces are trapped and reported using EC syndrome 0x18: <ul style="list-style-type: none"> IMPLEMENTATION DEFINED System instructions, which are accessed using SYS and SYSL, with CRn == {11, 15}. IMPLEMENTATION DEFINED System registers, which are accessed using MRS and MSR with the S3_<op1>_<Cn>_<Cm>_<op2> register name. In AArch32 state, EL0 MCR and MRC access to the following encodings are trapped and reported using EC syndrome 0x03: <ul style="list-style-type: none"> All coproc==p15, CRn==c9, opc1 == {0-7}, CRm == {c0-c2, c5-c8}, opc2 == {0-7}. All coproc==p15, CRn==c10, opc1 == {0-7}, CRm == {c0, c1, c4, c8}, opc2 == {0-7}. All coproc==p15, CRn==c11, opc1 == {0-7}, CRm == {c0-c8, c15}, opc2 == {0-7}.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

RES0

SPINTMASK, bit [62]

When FEAT_NMI is implemented:

SP Interrupt Mask enable. When SCTLR_EL1.NMI is 1, controls whether PSTATE.SP acts as an interrupt mask, and controls the value of PSTATE.ALLINT on taking an exception to EL1.

SPINTMASK	Meaning
0b0	Does not cause PSTATE.SP to mask interrupts. PSTATE.ALLINT is set to 1 on taking an exception to EL1.
0b1	When PSTATE.SP is 1 and execution is at EL1, an IRQ or FIQ interrupt that is targeted to EL1 is masked regardless of any denotation of Superpriority. PSTATE.ALLINT is set to 0 on taking an exception to EL1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

RES0

NMI, bit [61]

When FEAT_NMI is implemented:

Non-maskable Interrupt enable.

NMI	Meaning
0b0	This control does not affect interrupt masking behavior.
0b1	This control enables all of the following: <ul style="list-style-type: none"> • The use of the PSTATE.ALLINT interrupt mask. • IRQ and FIQ interrupts to have Superpriority as an additional attribute. • PSTATE.SP to be used as an interrupt mask.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0b0.

Otherwise:

RES0

EnTP2, bit [60]

When FEAT_SME is implemented:

Traps instructions executed at EL0 that access [TPIDR2_EL0](#) to EL1, or to EL2 when EL2 is implemented and enabled for the current Security state and HCR_EL2.TGE is 1. The exception is reported using ESR_ELx.EC value 0x18.

EnTP2	Meaning
0b0	This control causes execution of these instructions at EL0 to be trapped.
0b1	This control does not cause execution of any instructions to be trapped.

If FEAT_VHE is implemented, EL2 is implemented and enabled in the current Security state, and HCR_EL2.{E2H, TGE} == {1, 1}, this field has no effect on execution at EL0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

RES0

Bits [59:58]

Reserved, RES0.

EPAN, bit [57]

When FEAT_PAN3 is implemented:

Enhanced Privileged Access Never. When PSTATE.PAN is 1, determines whether an EL1 data access to a page with stage 1 EL0 instruction access permission generates a Permission fault as a result of the Privileged Access Never mechanism.

EPAN	Meaning
0b0	No additional Permission faults are generated by this mechanism.
0b1	An EL1 data access to a page with stage 1 EL0 data access permission or stage 1 EL0 instruction access permission generates a Permission fault. Any speculative data accesses that would generate a Permission fault if the accesses were not speculative will not cause an allocation into a cache.

This bit is permitted to be cached in a TLB.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

RES0

EnALS, bit [56]

When FEAT_LS64 is implemented:

When HCR_EL2.{E2H, TGE} != {1, 1}, traps execution of an LD64B or ST64B instruction at EL0 to EL1.

EnALS	Meaning
0b0	Execution of an LD64B or ST64B instruction at EL0 is trapped to EL1.
0b1	This control does not cause any instructions to be trapped.

A trap of an LD64B or ST64B instruction is reported using an ESR_ELx.EC value of 0x0A, with an ISS code of 0x0000002.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

RES0

EnAS0, bit [55]

When FEAT_LS64_ACCDATA is implemented:

When HCR_EL2.{E2H, TGE} != {1, 1}, traps execution of an ST64BV0 instruction at EL0 to EL1.

EnAS0	Meaning
0b0	Execution of an ST64BV0 instruction at EL0 is trapped to EL1.
0b1	This control does not cause any instructions to be trapped.

A trap of an ST64BV0 instruction is reported using an ESR_ELx.EC value of 0x0A, with an ISS code of 0x0000001.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

RES0

EnASR, bit [54]

When FEAT_LS64_V is implemented:

When HCR_EL2.{E2H, TGE} != {1, 1}, traps execution of an ST64BV instruction at EL0 to EL1.

EnASR	Meaning
0b0	Execution of an ST64BV instruction at EL0 is trapped to EL1.
0b1	This control does not cause any instructions to be trapped.

A trap of an ST64BV instruction is reported using an ESR_ELx.EC value of 0x0A, with an ISS code of 0x0000000.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

RES0

TME, bit [53]

When FEAT_TME is implemented:

Enables the Transactional Memory Extension at EL1.

TME	Meaning
0b0	Any attempt to execute a TSTART instruction at EL1 is trapped to EL1, unless HCR_EL2.TME or SCR_EL3.TME causes TSTART instructions to be UNDEFINED at EL1.

TME	Meaning
0b1	This control does not cause any TSTART instruction to be trapped.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

RES0

TME0, bit [52]

When FEAT_TME is implemented:

Enables the Transactional Memory Extension at EL0.

TME0	Meaning
0b0	Any attempt to execute a TSTART instruction at EL0 is trapped to EL1, unless HCR_EL2.TME or SCR_EL3.TME causes TSTART instructions to be UNDEFINED at EL0.
0b1	This control does not cause any TSTART instruction to be trapped.

If FEAT_VHE is implemented, EL2 is implemented and enabled in the current Security state, and HCR_EL2.{E2H, TGE} == {1, 1}, this field has no effect on execution at EL0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

RES0

TMT, bit [51]

When FEAT_TME is implemented:

Forces a trivial implementation of the Transactional Memory Extension at EL1.

TMT	Meaning
0b0	This control does not cause any TSTART instruction to fail.
0b1	When the TSTART instruction is executed at EL1, the transaction fails with a TRIVIAL failure cause.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

RES0

TMT0, bit [50]

When FEAT_TME is implemented:

Forces a trivial implementation of the Transactional Memory Extension at EL0.

TMT0	Meaning
0b0	This control does not cause any TSTART instruction to fail.
0b1	When the TSTART instruction is executed at EL0, the transaction fails with a TRIVIAL failure cause.

If FEAT_VHE is implemented, EL2 is implemented and enabled in the current Security state, and HCR_EL2.{E2H, TGE} == {1, 1}, this field has no effect on execution at EL0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

RES0

TWEDEL, bits [49:46]

When FEAT_TWED is implemented:

TWE Delay. A 4-bit unsigned number that, when SCTLR_EL1.TWEDEn is 1, encodes the minimum delay in taking a trap of WFE* caused by SCTLR_EL1.nTWE as $2^{(TWEDEL + 8)}$ cycles.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

RES0

TWEDEn, bit [45]

When FEAT_TWED is implemented:

TWE Delay Enable. Enables a configurable delayed trap of the WFE* instruction caused by SCTLR_EL1.nTWE.

TWEDEn	Meaning
0b0	The delay for taking the trap is IMPLEMENTATION DEFINED.
0b1	The delay for taking the trap is at least the number of cycles defined in SCTLR_EL1.TWEDEL.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

RES0

DSSBS, bit [44]

When FEAT_SSBS is implemented:

Default PSTATE.SSBS value on Exception Entry.

DSSBS	Meaning
0b0	PSTATE.SSBS is set to 0 on an exception to EL1.
0b1	PSTATE.SSBS is set to 1 on an exception to EL1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an IMPLEMENTATION DEFINED value.

Otherwise:

RES0

ATA, bit [43]

When FEAT_MTE2 is implemented:

Allocation Tag Access in EL1. When [SCR_EL3.ATA](#)=1 and [HCR_EL2.ATA](#)=1, controls EL1 access to Allocation Tags.

ATA	Meaning
0b0	Access to Allocation Tags is prevented.
0b1	This control does not prevent access to Allocation Tags.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

RES0

ATA0, bit [42]

When FEAT_MTE2 is implemented:

Allocation Tag Access in EL0. When [SCR_EL3.ATA](#)=1, [HCR_EL2.ATA](#)=1, and [HCR_EL2.{E2H, TGE}](#) != {1, 1}, controls EL0 access to Allocation Tags.

ATA0	Meaning
0b0	Access to Allocation Tags is prevented.
0b1	This control does not prevent access to Allocation Tags.

Software may change this control bit on a context switch.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

RES0

TCF, bits [41:40]

When FEAT_MTE2 is implemented:

Tag Check Fault in EL1. Controls the effect of Tag Check Faults due to Loads and Stores in EL1.

If FEAT_MTE3 is not implemented, the value 0b11 is reserved.

TCF	Meaning	Applies
0b00	Tag Check Faults have no effect on the PE.	
0b01	Tag Check Faults cause a synchronous exception.	
0b10	Tag Check Faults are asynchronously accumulated.	
0b11	Tag Check Faults cause a synchronous exception on reads, and are asynchronously accumulated on writes.	When FEAT_MTE3 is implemented

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

RES0

TCF0, bits [39:38]

When FEAT_MTE2 is implemented:

Tag Check Fault in EL0. When HCR_EL2.{E2H,TGE} != {1,1}, controls the effect of Tag Check Faults due to Loads and Stores in EL0.

If FEAT_MTE3 is not implemented, the value 0b11 is reserved.

Software may change this control bit on a context switch.

TCF0	Meaning	Applies
0b00	Tag Check Faults have no effect on the PE.	
0b01	Tag Check Faults cause a synchronous exception.	
0b10	Tag Check Faults are asynchronously accumulated.	
0b11	Tag Check Faults cause a synchronous exception on reads, and are asynchronously accumulated on writes.	When FEAT_MTE3 is implemented

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

RES0

ITFSB, bit [37]

When FEAT_MTE2 is implemented:

When synchronous exceptions are not being generated by Tag Check Faults, this field controls whether on exception entry into EL1, all Tag Check Faults due to instructions executed before exception entry, that are reported asynchronously, are synchronized into TFSRE0_EL1 and TFSR_EL1 registers.

ITFSB	Meaning
0b0	Tag Check Faults are not synchronized on entry to EL1.
0b1	Tag Check Faults are synchronized on entry to EL1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

RES0

BT1, bit [36]

When FEAT_BT1 is implemented:

PAC Branch Type compatibility at EL1.

BT1	Meaning
0b0	When the PE is executing at EL1, PACIASP and PACIBSP are compatible with PSTATE.BTYPE == 0b11.
0b1	When the PE is executing at EL1, PACIASP and PACIBSP are not compatible with PSTATE.BTYPE == 0b11.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

RES0

BT0, bit [35]

When FEAT_BT1 is implemented:

PAC Branch Type compatibility at EL0.

BT0	Meaning
0b0	When the PE is executing at EL0, PACIASP and PACIBSP are compatible with PSTATE.BTYPE == 0b11.

BT0	Meaning
0b1	When the PE is executing at EL0, PACIASP and PACIBSP are not compatible with PSTATE.BTYPE == 0b11.

When the value of HCR_EL2.{E2H, TGE} is {1, 1}, the value of SCTLRL_EL1.BT0 has no effect on execution at EL0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

RES0

Bit [34]

Reserved, RES0.

MSCEn, bit [33]

When FEAT_MOPS is implemented and (HCR_EL2.E2H == 0 or HCR_EL2.TGE == 0):

Memory Copy and Memory Set instructions Enable. Enables execution of the Memory Copy and Memory Set instructions at EL0.

MSCEn	Meaning
0b0	Execution of the Memory Copy and Memory Set instructions is UNDEFINED at EL0.
0b1	This control does not cause any instructions to be UNDEFINED.

When FEAT_MOPS is implemented and HCR_EL2.{E2H, TGE} is {1, 1}, the Effective value of this bit is 0b1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

RES0

CMOW, bit [32]

When FEAT_CMOW is implemented:

Controls cache maintenance instruction permission for the following instructions executed at EL0.

- IC IVAU, DC CIVAC, DC CIGDVAC and DC CIGVAC.

CMOW	Meaning
0b0	These instructions executed at EL0 with stage 1 read permission, but without stage 1 write permission, do not generate a stage 1 permission fault.

CMOW	Meaning
0b1	If enabled as a result of SCTLR_EL1.UCI==1 , these instructions executed at EL0 with stage 1 read permission, but without stage 1 write permission, generate a stage 1 permission fault.

When AArch64.HCR_EL2.{E2H, TGE} is {1, 1}, this bit has no effect on execution at EL0.

For this control, stage 1 has write permission if all of the following apply:

- AP[2] is 0 or DBM is 1 in the stage 1 descriptor.
- Where APTable is in use, APTable[1] is 0 for all levels of the translation table.

This bit is permitted to be cached in a TLB.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

RES0

EnIA, bit [31]

When FEAT_PAuth is implemented:

Controls enabling of pointer authentication (using the APIAKey_EL1 key) of instruction addresses in the EL1&0 translation regime.

For more information, see System register control of pointer authentication .

EnIA	Meaning
0b0	Pointer authentication (using the APIAKey_EL1 key) of instruction addresses is not enabled.
0b1	Pointer authentication (using the APIAKey_EL1 key) of instruction addresses is enabled.

This field controls the behavior of the AddPACIA and AuthIA pseudocode functions. Specifically, when the field is 1, AddPACIA returns a copy of a pointer to which a pointer authentication code has been added, and AuthIA returns an authenticated copy of a pointer. When the field is 0, both of these functions are NOP.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

RES0

EnIB, bit [30]

When FEAT_PAuth is implemented:

Controls enabling of pointer authentication (using the APIBKey_EL1 key) of instruction addresses in the EL1&0 translation regime.

For more information, see System register control of pointer authentication .

EnIB	Meaning
0b0	Pointer authentication (using the APIBKey_EL1 key) of instruction addresses is not enabled.
0b1	Pointer authentication (using the APIBKey_EL1 key) of instruction addresses is enabled.

This field controls the behavior of the AddPACIB and AuthIB pseudocode functions. Specifically, when the field is 1, AddPACIB returns a copy of a pointer to which a pointer authentication code has been added, and AuthIB returns an authenticated copy of a pointer. When the field is 0, both of these functions are NOP.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

RES0

LSMAOE, bit [29]

When FEAT_LSMAOC is implemented:

Load Multiple and Store Multiple Atomicity and Ordering Enable.

LSMAOE	Meaning
0b0	For all memory accesses at EL0, A32 and T32 Load Multiple and Store Multiple can have an interrupt taken during the sequence memory accesses, and the memory accesses are not required to be ordered.
0b1	The ordering and interrupt behavior of A32 and T32 Load Multiple and Store Multiple at EL0 is as defined for Armv8.0.

This bit is permitted to be cached in a TLB.

When FEAT_VHE is implemented, and the value of HCR_EL2.{E2H, TGE} is {1,1}, this bit has no effect on execution at EL0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

RES1

nTLSMD, bit [28]

When FEAT_LSMAOC is implemented:

No Trap Load Multiple and Store Multiple to Device-nGRE/Device-nGnRE/Device-nGnRnE memory.

nTLSMD	Meaning
0b0	All memory accesses by A32 and T32 Load Multiple and Store Multiple at EL0 that are marked at stage 1 as Device-nGRE/Device-nGnRE/Device-nGnRnE memory are trapped and generate a stage 1 Alignment fault.
0b1	All memory accesses by A32 and T32 Load Multiple and Store Multiple at EL0 that are marked at stage 1 as Device-nGRE/Device-nGnRE/Device-nGnRnE memory are not trapped.

This bit is permitted to be cached in a TLB.

When FEAT_VHE is implemented, and the value of HCR_EL2.{E2H, TGE} is {1,1}, this bit has no effect on execution at EL0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

RES1

EnDA, bit [27]

When FEAT_PAuth is implemented:

Controls enabling of pointer authentication (using the APDAKey_EL1 key) of instruction addresses in the EL1&0 translation regime.

For more information, see System register control of pointer authentication .

EnDA	Meaning
0b0	Pointer authentication (using the APDAKey_EL1 key) of data addresses is not enabled.
0b1	Pointer authentication (using the APDAKey_EL1 key) of data addresses is enabled.

This field controls the behavior of the AddPACDA and AuthDA pseudocode functions. Specifically, when the field is 1, AddPACDA returns a copy of a pointer to which a pointer authentication code has been added, and AuthDA returns an authenticated copy of a pointer. When the field is 0, both of these functions are NOP.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

RES0

UCI, bit [26]

Traps EL0 execution of cache maintenance instructions, to EL1, or to EL2 when it is implemented and enabled for the current Security state and HCR_EL2.TGE is 1, from AArch64 state only, reported using an ESR_ELx.EC value of 0x18.

This applies to DC CVAU, DC CIVAC, DC CVAC, DC CVAP, and IC IVAU.

If FEAT_DPB2 is implemented, this trap also applies to DC CVADP.

If FEAT_MTE is implemented, this trap also applies to DC CIGVAC, DC CIGDVAC, DC CGVAC, DC CGDVAC, DC CGVAP, and DC CGDVAP.

If FEAT_DPB2 and FEAT_MTE are implemented, this trap also applies to DC CGVADP and DC CGDVADP.

UCI	Meaning
0b0	Execution of the specified instructions at EL0 using AArch64 is trapped.
0b1	This control does not cause any instructions to be trapped.

When FEAT_VHE is implemented, and the value of HCR_EL2.{E2H, TGE} is {1, 1}, this bit has no effect on execution at EL0.

If the Point of Coherency is before any level of data cache, it is IMPLEMENTATION DEFINED whether the execution of any data or unified cache clean, or clean and invalidate instruction that operates by VA to the point of coherency can be trapped when the value of this control is 1.

If the Point of Unification is before any level of data cache, it is IMPLEMENTATION DEFINED whether the execution of any data or unified cache clean by VA to the Point of Unification instruction can be trapped when the value of this control is 1.

If the Point of Unification is before any level of instruction cache, it is IMPLEMENTATION DEFINED whether the execution of any instruction cache invalidate by VA to the Point of Unification instruction can be trapped when the value of this control is 1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

EE, bit [25]

Endianness of data accesses at EL1, and stage 1 translation table walks in the EL1&0 translation regime.

EE	Meaning
0b0	Explicit data accesses at EL1, and stage 1 translation table walks in the EL1&0 translation regime are little-endian.
0b1	Explicit data accesses at EL1, and stage 1 translation table walks in the EL1&0 translation regime are big-endian.

If an implementation does not provide Big-endian support at Exception levels higher than EL0, this bit is RES0.

If an implementation does not provide Little-endian support at Exception levels higher than EL0, this bit is RES1.

The EE bit is permitted to be cached in a TLB.

When FEAT_VHE is implemented, and the value of HCR_EL2.{E2H, TGE} is {1, 1}, this bit has no effect on the PE.

The reset behavior of this field is:

- On a Warm reset, this field resets to an IMPLEMENTATION DEFINED value.

E0E, bit [24]

Endianness of data accesses at EL0.

E0E	Meaning
0b0	Explicit data accesses at EL0 are little-endian.
0b1	Explicit data accesses at EL0 are big-endian.

If an implementation only supports Little-endian accesses at EL0, then this bit is RES0. This option is not permitted when SCTLR_EL1.EE is RES1.

If an implementation only supports Big-endian accesses at EL0, then this bit is RES1. This option is not permitted when SCTLR_EL1.EE is RES0.

This bit has no effect on the endianness of LDTR, LDTRH, LDTRSH, LDTRSW, STTR, and STTRH instructions executed at EL1.

When FEAT_VHE is implemented, and the value of HCR_EL2.{E2H, TGE} is {1, 1}, this bit has no effect on execution at EL0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

SPAN, bit [23]

When FEAT_PAN is implemented:

Set Privileged Access Never, on taking an exception to EL1.

SPAN	Meaning
0b0	PSTATE.PAN is set to 1 on taking an exception to EL1.
0b1	The value of PSTATE.PAN is left unchanged on taking an exception to EL1.

When FEAT_VHE is implemented, and the value of HCR_EL2.{E2H, TGE} is {1, 1}, this bit has no effect on execution at EL0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

RES1

EIS, bit [22]

When FEAT_ExS is implemented:

Exception Entry is Context Synchronizing.

EIS	Meaning
0b0	The taking of an exception to EL1 is not a context synchronizing event.

EIS	Meaning
0b1	The taking of an exception to EL1 is a context synchronizing event.

When FEAT_VHE is implemented, and the value of HCR_EL2.{E2H, TGE} is {1,1}, this bit has no effect on execution at EL0.

If SCTLRL_EL1.EIS is set to 0b0:

- Indirect writes to ESR_EL1, FAR_EL1, SPSR_EL1, ELR_EL1 are synchronized on exception entry to EL1, so that a direct read of the register after exception entry sees the indirectly written value caused by the exception entry.
- Memory transactions, including instruction fetches, from an Exception level always use the translation resources associated with that translation regime.
- Exception Catch debug events are synchronous debug events.
- DCPS* and DRPS instructions are context synchronization events.

The following are not affected by the value of SCTLRL_EL1.EIS:

- Changes to the PSTATE information on entry to EL1.
- Behavior of accessing the banked copies of the stack pointer using the SP register name for loads, stores and data processing instructions.
- Exit from Debug state.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

RES1

IESB, bit [21]

When FEAT_IESB is implemented:

Implicit Error Synchronization event enable. Possible values are:

IESB	Meaning
0b0	Disabled.
0b1	An implicit error synchronization event is added: <ul style="list-style-type: none"> • At each exception taken to EL1. • Before the operational pseudocode of each <code>ERET</code> instruction executed at EL1.

When the PE is in Debug state, the effect of this field is **CONSTRAINED UNPREDICTABLE**, and its Effective value might be 0 or 1 regardless of the value of the field. If the Effective value of the field is 1, then an implicit error synchronization event is added after each `DCPSX` instruction taken to EL1 and before each `DRPS` instruction executed at EL1, in addition to the other cases where it is added.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

RES0

TSCXT, bit [20]

When FEAT_CSV2_2 is implemented or FEAT_CSV2_1p2 is implemented:

Trap EL0 Access to the SCXTNUM_EL0 register, when EL0 is using AArch64.

TSCXT	Meaning
0b0	EL0 access to SCXTNUM_EL0 is not disabled by this mechanism.
0b1	EL0 access to SCXTNUM_EL0 is disabled, causing an exception to EL1, or to EL2 when it is implemented and enabled for the current Security state and HCR_EL2.TGE is 1. The value of SCXTNUM_EL0 is treated as 0.

When FEAT_VHE is implemented, and the value of HCR_EL2.{E2H, TGE} is {1,1}, this bit has no effect on execution at EL0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

RES1

WXN, bit [19]

Write permission implies XN (Execute-never). For the EL1&0 translation regime, this bit can force all memory regions that are writable to be treated as XN.

WXN	Meaning
0b0	This control has no effect on memory access permissions.
0b1	Any region that is writable in the EL1&0 translation regime is forced to XN for accesses from software executing at EL1 or EL0.

This bit applies only when SCTL_EL1.M bit is set.

The WXN bit is permitted to be cached in a TLB.

When FEAT_VHE is implemented, and the value of HCR_EL2.{E2H, TGE} is {1, 1}, this bit has no effect on the PE.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

nTWE, bit [18]

Traps EL0 execution of WFE instructions to EL1, or to EL2 when it is implemented and enabled for the current Security state and HCR_EL2.TGE is 1, from both Execution states, reported using an ESR_ELx.EC value of 0x01.

When FEAT_WFxT is implemented, this trap also applies to the WFET instruction.

nTWE	Meaning
0b0	Any attempt to execute a WFE instruction at EL0 is trapped, if the instruction would otherwise have caused the PE to enter a low-power state.
0b1	This control does not cause any instructions to be trapped.

In AArch32 state, the attempted execution of a conditional WFE instruction is only trapped if the instruction passes its condition code check.

Since a WFE or WFI can complete at any time, even without a Wakeup event, the traps on WFE or WFI are not guaranteed to be taken, even if the WFE or WFI is executed when there is no Wakeup event. The only guarantee is that if the instruction does not complete in finite time in the absence of a Wakeup event, the trap will be taken.

When FEAT_VHE is implemented, and the value of HCR_EL2.{E2H, TGE} is {1, 1}, this bit has no effect on execution at EL0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bit [17]

Reserved, RES0.

nTWI, bit [16]

Traps EL0 execution of WFI instructions to EL1, or to EL2 when it is implemented and enabled for the current Security state and HCR_EL2.TGE is 1, from both Execution states, reported using an ESR_ELx.EC value of 0x01.

When FEAT_WFxT is implemented, this trap also applies to the WFIT instruction.

nTWI	Meaning
0b0	Any attempt to execute a WFI instruction at EL0 is trapped, if the instruction would otherwise have caused the PE to enter a low-power state.
0b1	This control does not cause any instructions to be trapped.

In AArch32 state, the attempted execution of a conditional WFI instruction is only trapped if the instruction passes its condition code check.

Since a WFE or WFI can complete at any time, even without a Wakeup event, the traps on WFE or WFI are not guaranteed to be taken, even if the WFE or WFI is executed when there is no Wakeup event. The only guarantee is that if the instruction does not complete in finite time in the absence of a Wakeup event, the trap will be taken.

When FEAT_VHE is implemented, and the value of HCR_EL2.{E2H, TGE} is {1, 1}, this bit has no effect on execution at EL0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

UCT, bit [15]

Traps EL0 accesses to the CTR_EL0 to EL1, or to EL2 when it is implemented and enabled for the current Security state and HCR_EL2.TGE is 1, from AArch64 state only, reported using an ESR_ELx.EC value of 0x18.

UCT	Meaning
0b0	Accesses to the CTR_EL0 from EL0 using AArch64 are trapped.
0b1	This control does not cause any instructions to be trapped.

When FEAT_VHE is implemented, and the value of HCR_EL2.{E2H, TGE} is {1, 1}, this bit has no effect on execution at EL0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

DZE, bit [14]

Traps EL0 execution of DC ZVA instructions to EL1, or to EL2 when it is implemented and enabled for the current Security state and HCR_EL2.TGE is 1, from AArch64 state only, reported using an ESR_ELx.EC value of 0x18.

If FEAT_MTE is implemented, this trap also applies to DC GVA and DC GZVA.

DZE	Meaning
0b0	Any attempt to execute an instruction that this trap applies to at EL0 using AArch64 is trapped. Reading DCZID_EL0.DZP from EL0 returns 1, indicating that the instructions this trap applies to are not supported.
0b1	This control does not cause any instructions to be trapped.

When FEAT_VHE is implemented, and the value of HCR_EL2.{E2H, TGE} is {1, 1}, this bit has no effect on execution at EL0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

EnDB, bit [13]

When FEAT_PAuth is implemented:

Controls enabling of pointer authentication (using the APDBKey_EL1 key) of instruction addresses in the EL1&0 translation regime.

For more information, see System register control of pointer authentication .

EnDB	Meaning
0b0	Pointer authentication (using the APDBKey_EL1 key) of data addresses is not enabled.

EnDB	Meaning
0b1	Pointer authentication (using the APDBKey_EL1 key) of data addresses is enabled.

This field controls the behavior of the AddPACDB and AuthDB pseudocode functions. Specifically, when the field is 1, AddPACDB returns a copy of a pointer to which a pointer authentication code has been added, and AuthDB returns an authenticated copy of a pointer. When the field is 0, both of these functions are NOP.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

RES0

I, bit [12]

Stage 1 instruction access Cacheability control, for accesses at EL0 and EL1:

I	Meaning
0b0	All instruction access to Stage 1 Normal memory from EL0 and EL1 are Stage 1 Non-cacheable. If the value of SCTLRL_EL1.M is 0, instruction accesses from stage 1 of the EL1&0 translation regime are to Normal, Outer Shareable, Inner Non-cacheable, Outer Non-cacheable memory.
0b1	This control has no effect on the Stage 1 Cacheability of instruction access to Stage 1 Normal memory from EL0 and EL1. If the value of SCTLRL_EL1.M is 0, instruction accesses from stage 1 of the EL1&0 translation regime are to Normal, Outer Shareable, Inner Write-Through, Outer Write-Through memory.

When the value of the HCR_EL2.DC bit is 1, then instruction access to Normal memory from EL0 and EL1 are Cacheable regardless of the value of the SCTLRL_EL1.I bit.

When FEAT_VHE is implemented, and the value of HCR_EL2.{E2H, TGE} is {1, 1}, this bit has no effect on the PE.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0b0.

EOS, bit [11]

When FEAT_ExS is implemented:

Exception Exit is Context Synchronizing.

EOS	Meaning
0b0	An exception return from EL1 is not a context synchronizing event
0b1	An exception return from EL1 is a context synchronizing event

When FEAT_VHE is implemented, and the value of HCR_EL2.{E2H, TGE} is {1,1}, this bit has no effect on execution at EL0.

If SCTL_EL1.EOS is set to 0b0:

- Memory transactions, including instruction fetches, from an Exception level always use the translation resources associated with that translation regime.
- Exception Catch debug events are synchronous debug events.
- DCPS* and DRPS instructions are context synchronization events.

The following are not affected by the value of SCTL_EL1.EOS:

- The indirect write of the PSTATE and PC values from SPSR_EL1 and ELR_EL1 on exception return is synchronized.
- Behavior of accessing the banked copies of the stack pointer using the SP register name for loads, stores and data processing instructions.
- Exit from Debug state.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

RES1

EnRCTX, bit [10]

When FEAT_SPECRES is implemented:

Enable EL0 Access to the following instructions:

- AArch32 CFPRCTX, DVPRCTX and CPPRCTX instructions.
- AArch64 CFP RCTX, DVP RCT and CPP RCTX instructions.

EnRCTX	Meaning
0b0	EL0 access to these instructions is disabled, and these instructions are trapped to EL1, or to EL2 when it is implemented and enabled for the current Security state and HCR_EL2.TGE is 1.
0b1	EL0 access to these instructions is enabled.

When FEAT_VHE is implemented, and the value of HCR_EL2.{E2H, TGE} is {1,1}, this bit has no effect on execution at EL0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

RES0

UMA, bit [9]

User Mask Access. Traps EL0 execution of MSR and MRS instructions that access the PSTATE.{D, A, I, F} masks to EL1, or to EL2 when it is implemented and enabled for the current Security state and HCR_EL2.TGE is 1, from AArch64 state only, reported using an ESR_ELx.EC value of 0x18.

UMA	Meaning
0b0	Any attempt at EL0 using AArch64 to execute an MRS, MSR(REGISTER), or MSR(IMMEDIATE) instruction that accesses the DAIF is trapped.
0b1	This control does not cause any instructions to be trapped.

When FEAT_VHE is implemented, and the value of HCR_EL2.{E2H, TGE} is {1, 1}, this bit has no effect on execution at EL0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

SED, bit [8]

When EL0 is capable of using AArch32:

SETEND instruction disable. Disables SETEND instructions at EL0 using AArch32.

SED	Meaning
0b0	SETEND instruction execution is enabled at EL0 using AArch32.
0b1	SETEND instructions are UNDEFINED at EL0 using AArch32 and any attempt at EL0 to access a SETEND instruction generates an exception to EL1, or to EL2 when it is implemented and enabled for the current Security state and HCR_EL2.TGE is 1, reported using an ESR_ELx.EC value of 0x00.

If the implementation does not support mixed-endian operation at any Exception level, this bit is RES1.

When FEAT_VHE is implemented, and the value of HCR_EL2.{E2H, TGE} is {1, 1}, this bit has no effect on execution at EL0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

RES1

ITD, bit [7]

When EL0 is capable of using AArch32:

IT Disable. Disables some uses of IT instructions at EL0 using AArch32.

ITD	Meaning
0b0	All IT instruction functionality is enabled at EL0 using AArch32.
0b1	<p>Any attempt at EL0 using AArch32 to execute any of the following is UNDEFINED and generates an exception, reported using an ESR_ELx.EC value of 0x00, to EL1 or to EL2 when it is implemented and enabled for the current Security state and HCR_EL2.TGE is 1:</p> <ul style="list-style-type: none"> • All encodings of the IT instruction with $hw1[3:0] \neq 1000$. • All encodings of the subsequent instruction with the following values for $hw1$: <ul style="list-style-type: none"> – 0b11xxxxxxxxxxxx: All 32-bit instructions, and the 16-bit instructions B, UDF, SVC, LDM, and STM. – 0b1011xxxxxxxxxxxx: All instructions in ‘Miscellaneous 16-bit instructions’ in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section F3.2.5. – 0b10100xxxxxxxxxxx: ADD Rd, PC, #imm – 0b01001xxxxxxxxxxx: LDR Rd, [PC, #imm] – 0b0100x1xxx1111xxx: ADD Rdn, PC; CMP Rn, PC; MOV Rd, PC; BX PC; BLX PC. – 0b010001xx1xxx111: ADD PC, Rm; CMP PC, Rm; MOV PC, Rm. This pattern also covers unpredictable cases with BLX Rn. <p>These instructions are always UNDEFINED, regardless of whether they would pass or fail the condition code check that applies to them as a result of being in an IT block. It is IMPLEMENTATION DEFINED whether the IT instruction is treated as:</p> <ul style="list-style-type: none"> • A 16-bit instruction, that can only be followed by another 16-bit instruction. • The first half of a 32-bit instruction. <p>This means that, for the situations that are UNDEFINED, either the second 16-bit instruction or the 32-bit instruction is UNDEFINED.</p> <p>An implementation might vary dynamically as to whether IT is treated as a 16-bit instruction or the first half of a 32-bit instruction.</p>

If an instruction in an active IT block that would be disabled by this field sets this field to 1 then behavior is CONSTRAINED UNPREDICTABLE. For more information, see Changes to an ITD control by an instruction in an IT block .

ITD is optional, but if it is implemented in the SCTLR_EL1 then it must also be implemented in the [SCTLR_EL2](#), HSCTLR, and SCTLR.

When FEAT_VHE is implemented, and the value of HCR_EL2.{E2H, TGE} is {1, 1}, this bit has no effect on execution at EL0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

When an implementation does not implement ITD, access to this field is **RAZ/WI**.

Otherwise:

RES1

nAA, bit [6]

When FEAT_LSE2 is implemented:

Non-aligned access. This bit controls generation of Alignment faults at EL1 and EL0 under certain conditions.

nAA	Meaning
0b0	LDAPR, LDAPRH, LDAPUR, LDAPURH, LDAPURSH, LDAPURSW, LDAR, LDARH, LDLAR, LDLARH, STLLR, STLLRH, STLR, STLRH, STLUR, and STLURH generate an Alignment fault if all bytes being accessed are not within a single 16-byte quantity, aligned to 16 bytes for accesses.
0b1	This control bit does not cause LDAPR, LDAPRH, LDAPUR, LDAPURH, LDAPURSH, LDAPURSW, LDAR, LDARH, LDLAR, LDLARH, STLLR, STLLRH, STLR, STLRH, STLUR, or STLURH to generate an Alignment fault if all bytes being accessed are not within a single 16-byte quantity, aligned to 16 bytes.

When FEAT_VHE is implemented, and the value of HCR_EL2.{E2H, TGE} is {1, 1}, this bit has no effect on execution at EL0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

RES0

CP15BEN, bit [5]

When EL0 is capable of using AArch32:

System instruction memory barrier enable. Enables accesses to the DMB, DSB, and ISB System instructions in the (coproc==0b1111) encoding space from EL0:

CP15BEN	Meaning
0b0	EL0 using AArch32: EL0 execution of the CP15DMB, CP15DSB, and CP15ISB instructions is UNDEFINED and generates an exception to EL1, or to EL2 when it is implemented and enabled for the current Security state and HCR_EL2.TGE is 1. The exception is reported using an ESR_ELx.EC value of 0x00.
0b1	EL0 using AArch32: EL0 execution of the CP15DMB, CP15DSB, and CP15ISB instructions is enabled.

CP15BEN is optional, but if it is implemented in the SCTLR_EL1 then it must also be implemented in the [SCTLR_EL2](#), HSCTLR, and SCTLR.

When FEAT_VHE is implemented, and the value of HCR_EL2.{E2H, TGE} is {1, 1}, this bit has no effect on execution at EL0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

When an implementation does not implement CP15BEN, access to this field is **RAO/WI**.

Otherwise:

RES0

SA0, bit [4]

SP Alignment check enable for EL0. When set to 1, if a load or store instruction executed at EL0 uses the SP as the base address and the SP is not aligned to a 16-byte boundary, then an SP alignment fault exception is generated. For more information, see ‘SP alignment checking’.

When FEAT_VHE is implemented, and the value of HCR_EL2.{E2H, TGE} is {1, 1}, this bit has no effect on execution at EL0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

SA, bit [3]

SP Alignment check enable. When set to 1, if a load or store instruction executed at EL1 uses the SP as the base address and the SP is not aligned to a 16-byte boundary, then an SP alignment fault exception is generated. For more information, see ‘SP alignment checking’.

When FEAT_VHE is implemented, and the value of HCR_EL2.{E2H, TGE} is {1, 1}, this bit has no effect on the PE.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

C, bit [2]

Stage 1 Cacheability control, for data accesses.

C	Meaning
0b0	All data access to Stage 1 Normal memory from EL0 and EL1, and all Normal memory accesses from unified cache to the EL1&0 Stage 1 translation tables, are treated as Stage 1 Non-cacheable.
0b1	This control has no effect on the Stage 1 Cacheability of: <ul style="list-style-type: none"> • Data access to Normal memory from EL0 and EL1. • Normal memory accesses to the EL1&0 Stage 1 translation tables.

When the value of the HCR_EL2.DC bit is 1, the PE ignores SCTLR.C. This means that Non-secure EL0 and Non-secure EL1 data accesses to Normal memory are Cacheable.

When FEAT_VHE is implemented, and the value of HCR_EL2.{E2H, TGE} is {1, 1}, this bit has no effect on the PE.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0b0.

A, bit [1]

Alignment check enable. This is the enable bit for Alignment fault checking at EL1 and EL0.

A	Meaning
0b0	Alignment fault checking disabled when executing at EL1 or EL0. Instructions that load or store one or more registers, other than load/store exclusive and load-acquire/store-release, do not check that the address being accessed is aligned to the size of the data element(s) being accessed.
0b1	Alignment fault checking enabled when executing at EL1 or EL0. All instructions that load or store one or more registers have an alignment check that the address being accessed is aligned to the size of the data element(s) being accessed. If this check fails it causes an Alignment fault, which is taken as a Data Abort exception.

Load/store exclusive and load-acquire/store-release instructions have an alignment check regardless of the value of the A bit.

If FEAT_MOPS is implemented, SETG* instructions have an alignment check regardless of the value of the A bit.

When FEAT_VHE is implemented, and the value of HCR_EL2.{E2H, TGE} is {1, 1}, this bit has no effect on execution at EL0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

M, bit [0]

MMU enable for EL1&0 stage 1 address translation.

M	Meaning
0b0	EL1&0 stage 1 address translation disabled. See the SCTLR_EL1.I field for the behavior of instruction accesses to Normal memory.
0b1	EL1&0 stage 1 address translation enabled.

If the value of HCR_EL2.{DC, TGE} is not {0, 0} then in Non-secure state the PE behaves as if the value of the SCTLR_EL1.M field is 0 for all purposes other than returning the value of a direct read of the field.

When FEAT_VHE is implemented, and the value of HCR_EL2.{E2H, TGE} is {1, 1}, this bit has no effect on the PE.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0b0.

Accessing the SCTLR_EL1

When HCR_EL2.E2H is 1, without explicit synchronization, access from EL3 using the mnemonic SCTLR_EL1 or SCTLR_EL12 are not guaranteed to be ordered with respect to accesses using the other mnemonic.

Accesses to this register use the following encodings in the instruction encoding space:

MRS <Xt>, SCTLR_EL1

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

```

1  if PSTATE.EL == EL0 then
2      UNDEFINED;
3  elsif PSTATE.EL == EL1 then
4      if EL2Enabled() && HCR_EL2.TVM == '1' then
5          AArch64.SystemAccessTrap(EL2, 0x18);
6      elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGTR_EL2.SCTLR_EL1 == '1' then
7          AArch64.SystemAccessTrap(EL2, 0x18);
8      elsif EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '111' then
9          X[t, 64] = NVMem[0x110];
10     else
11         X[t, 64] = SCTLR_EL1;
12     elsif PSTATE.EL == EL2 then
13         if HCR_EL2.E2H == '1' then
14             X[t, 64] = SCTLR_EL2;
15         else
16             X[t, 64] = SCTLR_EL1;
17     elsif PSTATE.EL == EL3 then
18         X[t, 64] = SCTLR_EL1;

```

MSR SCTLR_EL1, <Xt>

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

```

1 if PSTATE.EL == EL0 then
2     UNDEFINED;
3 elseif PSTATE.EL == EL1 then
4     if EL2Enabled() && HCR_EL2.TVM == '1' then
5         AArch64.SystemAccessTrap(EL2, 0x18);
6     elseif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGWTR_EL2.SCTLR_EL1 == '1' then
7         AArch64.SystemAccessTrap(EL2, 0x18);
8     elseif EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '111' then
9         NVMem[0x110] = X[t, 64];
10    else
11        SCTLR_EL1 = X[t, 64];
12 elseif PSTATE.EL == EL2 then
13     if HCR_EL2.E2H == '1' then
14         SCTLR_EL2 = X[t, 64];
15     else
16         SCTLR_EL1 = X[t, 64];
17 elseif PSTATE.EL == EL3 then
18     SCTLR_EL1 = X[t, 64];

```

MRS <Xt>, SCTLR_EL12

op0	op1	CRn	CRm	op2
0b11	0b101	0b0001	0b0000	0b000

```

1 if PSTATE.EL == EL0 then
2     UNDEFINED;
3 elseif PSTATE.EL == EL1 then
4     if EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '101' then
5         X[t, 64] = NVMem[0x110];
6     elseif EL2Enabled() && HCR_EL2.NV == '1' then
7         AArch64.SystemAccessTrap(EL2, 0x18);
8     else
9         UNDEFINED;
10 elseif PSTATE.EL == EL2 then
11     if HCR_EL2.E2H == '1' then
12         X[t, 64] = SCTLR_EL1;
13     else
14         UNDEFINED;
15 elseif PSTATE.EL == EL3 then
16     if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '1' then
17         X[t, 64] = SCTLR_EL1;
18     else
19         UNDEFINED;

```

MSR SCTLR_EL12, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b101	0b0001	0b0000	0b000

```

1 if PSTATE.EL == EL0 then
2     UNDEFINED;
3 elseif PSTATE.EL == EL1 then
4     if EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '101' then
5         NVMem[0x110] = X[t, 64];

```

```
6      elsif EL2Enabled() && HCR_EL2.NV == '1' then
7          AArch64.SystemAccessTrap(EL2, 0x18);
8      else
9          UNDEFINED;
10     elsif PSTATE.EL == EL2 then
11         if HCR_EL2.E2H == '1' then
12             SCTLRR_EL1 = X[t, 64];
13         else
14             UNDEFINED;
15     elsif PSTATE.EL == EL3 then
16         if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '1' then
17             SCTLRR_EL1 = X[t, 64];
18         else
19             UNDEFINED;
```

E3.2.17 SCTLR_EL2, System Control Register (EL2)

The SCTLR_EL2 characteristics are:

Purpose

Provides top level control of the system, including its memory system, at EL2.

When FEAT_VHE is implemented, and the value of HCR_EL2.{E2H, TGE} is {1, 1}, these controls apply also to execution at EL0.

Attributes

SCTLR_EL2 is a 64-bit register.

Configuration

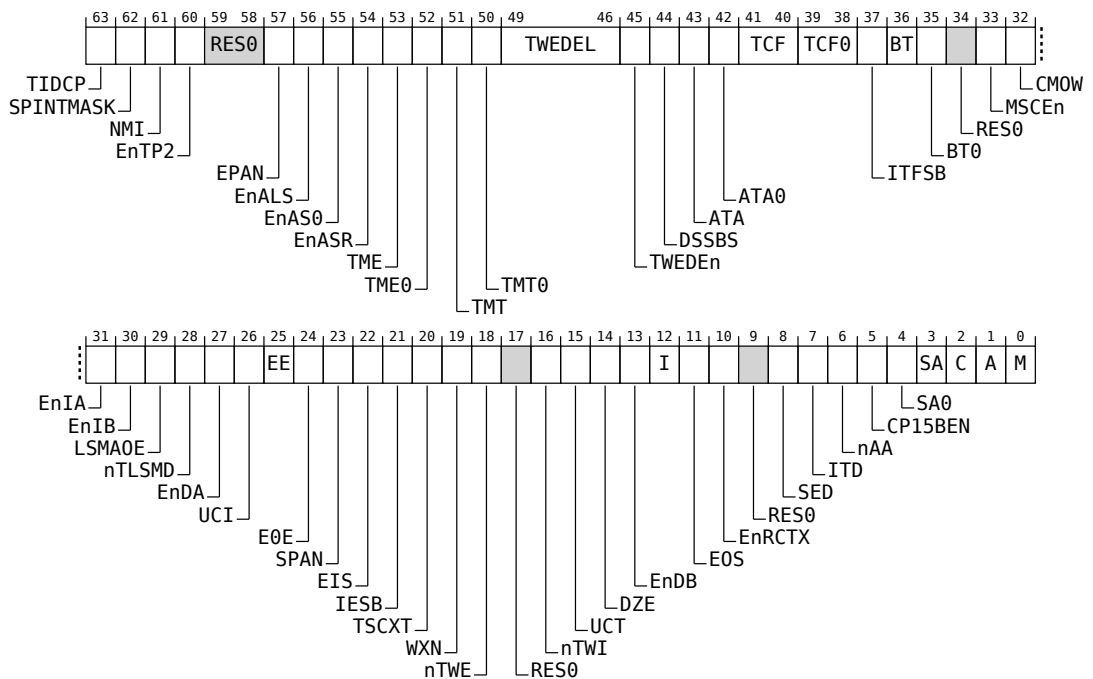
If EL2 is not implemented, this register is RES0 from EL3.

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

AArch64 system register SCTLR_EL2 bits [31:0] are architecturally mapped to AArch32 system register HSCTLR[31:0].

Field descriptions

The SCTLR_EL2 bit assignments are:



TIDCP, bit [63]

When FEAT_TIDCP1 is implemented and HCR_EL2.E2H == 1:

Trap IMPLEMENTATION DEFINED functionality. Traps EL0 accesses to the encodings reserved for IMPLEMENTATION DEFINED functionality to EL2.

TIDCP	Meaning
0b0	No instructions accessing the System register or System instruction spaces are trapped by this mechanism.
0b1	<p>If HCR_EL2.TGE==0, no instructions accessing the System register or System instruction spaces are trapped by this mechanism.</p> <p>If HCR_EL2.TGE==1, instructions accessing the following System register or System instruction spaces are trapped to EL2 by this mechanism:</p> <ul style="list-style-type: none"> • In AArch64 state, EL0 access to the encodings in the following reserved encoding spaces are trapped and reported using EC syndrome 0x18: <ul style="list-style-type: none"> – IMPLEMENTATION DEFINED System instructions, which are accessed using SYS and SYSL, with CRn == {11, 15}. – IMPLEMENTATION DEFINED System registers, which are accessed using MRS and MSR with the S3_<op1>_<Cn>_<Cm>_<op2> register name. • In AArch32 state, EL0 MCR and MRC access to the following encodings are trapped and reported using EC syndrome 0x03: <ul style="list-style-type: none"> – All coproc==p15, CRn==c9, opc1 == {0-7}, CRm == {c0-c2, c5-c8}, opc2 == {0-7}. – All coproc==p15, CRn==c10, opc1 =={0-7}, CRm == {c0, c1, c4, c8}, opc2 == {0-7}. – All coproc==p15, CRn==c11, opc1=={0-7}, CRm == {c0-c8, c15}, opc2 == {0-7}.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

RES0

SPINTMASK, bit [62]

When FEAT_NMI is implemented:

SP Interrupt Mask enable. When SCTLR_EL2.NMI is 1, controls whether PSTATE.SP acts as an interrupt mask, and controls the value of PSTATE.ALLINT on taking an exception to EL2.

SPINTMASK	Meaning
0b0	Does not cause PSTATE.SP to mask interrupts. PSTATE.ALLINT is set to 1 on taking an exception to EL2.
0b1	When PSTATE.SP is 1 and execution is at EL2, an IRQ or FIQ interrupt that is targeted to EL2 is masked regardless of any denotation of Superpriority. PSTATE.ALLINT is set to 0 on taking an exception to EL2.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

RES0

NMI, bit [61]

When FEAT_NMI is implemented:

Non-maskable Interrupt enable.

NMI	Meaning
0b0	This control does not affect interrupt masking behavior.
0b1	This control enables all of the following: <ul style="list-style-type: none"> • The use of the PSTATE.ALLINT interrupt mask. • IRQ and FIQ interrupts to have Superpriority as an additional attribute. • PSTATE.SP to be used as an interrupt mask.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0b0.

Otherwise:

RES0

EnTP2, bit [60]

When FEAT_SME is implemented, HCR_EL2.E2H == 1 and HCR_EL2.TGE == 1

EnTP2, bit [0] of bit [60]

Traps instructions executed at EL0 that access [TPIDR2_EL0](#) to EL2 when EL2 is implemented and enabled for the current Security state. The exception is reported using ESR_ELx.EC value 0x18.

EnTP2	Meaning
0b0	This control causes execution of these instructions at EL0 to be trapped.
0b1	This control does not cause execution of any instructions to be trapped.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

When FEAT_SME is implemented, HCR_EL2.E2H == 1 and HCR_EL2.TGE == 0

Bit [0]

Reserved, RAZ/WI.

Otherwise:

RES0

Bits [59:58]

Reserved, RES0.

EPAN, bit [57]

When FEAT_PAN3 is implemented, HCR_EL2.E2H == 1 and HCR_EL2.TGE == 1

EPAN, bit [0] of bit [57]

Enhanced Privileged Access Never. When PSTATE.PAN is 1, determines whether an EL2 data access to a page with EL0 instruction access permission generates a Permission fault as a result of the Privileged Access Never mechanism.

EPAN	Meaning
0b0	No additional Permission faults are generated by this mechanism.
0b1	An EL2 data access to a page with stage 1 EL0 data access permission or stage 1 EL0 instruction access permission generates a Permission fault. Any speculative data accesses that would generate a Permission fault if the accesses were not speculative will not cause an allocation into a cache.

This bit is permitted to be cached in a TLB.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

When FEAT_PAN3 is implemented, HCR_EL2.E2H == 1 and HCR_EL2.TGE == 0

Bit [0]

Reserved, RAZ/WI.

Otherwise:

RES0

EnALS, bit [56]

When FEAT_LS64 is implemented, HCR_EL2.E2H == 1 and HCR_EL2.TGE == 1

EnALS, bit [0] of bit [56]

Traps execution of an LD64B or ST64B instruction at EL0 to EL2.

EnALS	Meaning
0b0	Execution of an LD64B or ST64B instruction at EL0 is trapped to EL2.
0b1	This control does not cause any instructions to be trapped.

A trap of an LD64B or ST64B instruction is reported using an ESR_ELx.EC value of 0x0A, with an ISS code of 0x0000002.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

When FEAT_LS64 is implemented, HCR_EL2.E2H == 1 and HCR_EL2.TGE == 0

Bit [0]

Reserved, RAZ/WI.

Otherwise:

RES0

EnAS0, bit [55]

When FEAT_LS64_ACCDATA is implemented, HCR_EL2.E2H == 1 and HCR_EL2.TGE == 1

EnAS0, bit [0] of bit [55]

Traps execution of an ST64BV0 instruction at EL0 to EL2.

EnAS0	Meaning
0b0	Execution of an ST64BV0 instruction at EL0 is trapped to EL2.
0b1	This control does not cause any instructions to be trapped.

A trap of an ST64BV0 instruction is reported using an ESR_ELx.EC value of 0x0A, with an ISS code of 0x0000001.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

When FEAT_LS64_ACCDATA is implemented, HCR_EL2.E2H == 1 and HCR_EL2.TGE == 0

Bit [0]

Reserved, RAZ/WI.

Otherwise:

RES0

EnASR, bit [54]

When FEAT_LS64_V is implemented, HCR_EL2.E2H == 1 and HCR_EL2.TGE == 1

EnASR, bit [0] of bit [54]

Traps execution of an ST64BV instruction at EL0 to EL2.

EnASR	Meaning
0b0	Execution of an ST64BV instruction at EL0 is trapped to EL2.

EnASR	Meaning
0b1	This control does not cause any instructions to be trapped.

A trap of an ST64BV instruction is reported using an ESR_ELx.EC value of 0x0A, with an ISS code of 0x0000000.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

When FEAT_LS64_V is implemented, HCR_EL2.E2H == 1 and HCR_EL2.TGE == 0

Bit [0]

Reserved, RAZ/WI.

Otherwise:

RES0

TME, bit [53]

When FEAT_TME is implemented:

Enables the Transactional Memory Extension at EL2.

TME	Meaning
0b0	Any attempt to execute a TSTART instruction at EL2 is trapped, unless HCR_EL2.TME or SCR_EL3.TME causes TSTART instructions to be UNDEFINED at EL2.
0b1	This control does not cause any TSTART instruction to be trapped.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

RES0

TME0, bit [52]

When FEAT_TME is implemented, HCR_EL2.E2H == 1 and HCR_EL2.TGE == 1

TME0, bit [0] of bit [52]

Enables the Transactional Memory Extension at EL0.

TME0	Meaning
0b0	Any attempt to execute a TSTART instruction at EL0 is trapped to EL2, unless HCR_EL2.TME or SCR_EL3.TME causes TSTART instructions to be UNDEFINED at EL0.

TME0	Meaning
0b1	This control does not cause any TSTART instruction to be trapped.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

When FEAT_TME is implemented, HCR_EL2.E2H == 1 and HCR_EL2.TGE == 0

Bit [0]

Reserved, RAZ/WI.

Otherwise:

RES0

TMT, bit [51]

When FEAT_TME is implemented:

Forces a trivial implementation of the Transactional Memory Extension at EL2.

TMT	Meaning
0b0	This control does not cause any TSTART instruction to fail.
0b1	When the TSTART instruction is executed at EL2, the transaction fails with a TRIVIAL failure cause.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

RES0

TMT0, bit [50]

When FEAT_TME is implemented, HCR_EL2.E2H == 1 and HCR_EL2.TGE == 1

TMT0, bit [0] of bit [50]

Forces a trivial implementation of the Transactional Memory Extension at EL0.

TMT0	Meaning
0b0	This control does not cause any TSTART instruction to fail.
0b1	When the TSTART instruction is executed at EL0, the transaction fails with a TRIVIAL failure cause.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

When FEAT_TME is implemented, HCR_EL2.E2H == 1 and HCR_EL2.TGE == 0

Bit [0]

Reserved, RAZ/WI.

Otherwise:

RES0

TWEDEL, bits [49:46]

When FEAT_TWED is implemented, HCR_EL2.E2H == 1 and HCR_EL2.TGE == 1

TWEDEL, bits [3:0] of bits [49:46]

TWE Delay. A 4-bit unsigned number that, when SCTLR_EL2.TWEDEn is 1, encodes the minimum delay in taking a trap of WFE caused by SCTLR_EL2.nTWE as $2^{(TWEDEL + 8)}$ cycles.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

When FEAT_TWED is implemented, HCR_EL2.E2H == 1 and HCR_EL2.TGE == 0

Bits [3:0]

Reserved, RAZ/WI.

Otherwise:

RES0

TWEDEn, bit [45]

When FEAT_TWED is implemented, HCR_EL2.E2H == 1 and HCR_EL2.TGE == 1

TWEDEn, bit [0] of bit [45]

TWE Delay Enable. Enables a configurable delayed trap of the WFE instruction caused by SCTLR_EL2.nTWE.

TWEDEn	Meaning
0b0	The delay for taking a WFE trap is IMPLEMENTATION DEFINED.
0b1	The delay for taking a WFE trap is at least the number of cycles defined in SCTLR_EL2.TWEDEL.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

When FEAT_TWED is implemented, HCR_EL2.E2H == 1 and HCR_EL2.TGE == 0

Bit [0]

Reserved, RAZ/WI.

Otherwise:

RES0

DSSBS, bit [44]

When FEAT_SSBS is implemented:

Default PSTATE.SSBS value on Exception Entry.

DSSBS	Meaning
0b0	PSTATE.SSBS is set to 0 on an exception to EL2.
0b1	PSTATE.SSBS is set to 1 on an exception to EL2.

The reset behavior of this field is:

- On a Warm reset, this field resets to an IMPLEMENTATION DEFINED value.

Otherwise:

RES0

ATA, bit [43]

When FEAT_MTE2 is implemented:

Allocation Tag Access in EL2. When [SCR_EL3.ATA](#) is 1, controls EL2 access to Allocation Tags.

ATA	Meaning
0b0	Access to Allocation Tags is prevented.
0b1	This control does not prevent access to Allocation Tags.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

RES0

ATA0, bit [42]

When FEAT_MTE2 is implemented, HCR_EL2.E2H == 1 and HCR_EL2.TGE == 1

ATA0, bit [0] of bit [42]

Allocation Tag Access in EL0. When [SCR_EL3.ATA](#) is 1, controls EL0 access to Allocation Tags.

ATA0	Meaning
0b0	Access to Allocation Tags is prevented.
0b1	This control does not prevent access to Allocation Tags.

Software may change this control bit on a context switch.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

When FEAT_MTE2 is implemented, HCR_EL2.E2H == 1 and HCR_EL2.TGE == 0

Bit [0]

Reserved, RAZ/WI.

Otherwise:

RES0

TCF, bits [41:40]

When FEAT_MTE2 is implemented:

Tag Check Fault in EL2. Controls the effect of Tag Check Faults due to Loads and Stores in EL2.

TCF	Meaning	Applies
0b00	Tag Check Faults have no effect on the PE.	
0b01	Tag Check Faults cause a synchronous exception.	
0b10	Tag Check Faults are asynchronously accumulated.	
0b11	Tag Check Faults cause a synchronous exception on reads, and are asynchronously accumulated on writes.	When FEAT_MTE3 is implemented

If FEAT_MTE3 is not implemented, the value 0b11 is reserved.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

RES0

TCF0, bits [39:38]

When FEAT_MTE2 is implemented, HCR_EL2.E2H == 1 and HCR_EL2.TGE == 1

TCF0, bits [1:0] of bits [39:38]

Tag Check Fault in EL0. Controls the effect of Tag Check Faults due to Loads and Stores in EL0.

TCF0	Meaning	Applies
0b00	Tag Check Faults have no effect on the PE.	
0b01	Tag Check Faults cause a synchronous exception.	
0b10	Tag Check Faults are asynchronously accumulated.	
0b11	Tag Check Faults cause a synchronous exception on reads, and are asynchronously accumulated on writes.	When FEAT_MTE3 is implemented

If FEAT_MTE3 is not implemented, the value 0b11 is reserved.

Software may change this control bit on a context switch.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

When FEAT_MTE2 is implemented, HCR_EL2.E2H == 1 and HCR_EL2.TGE == 0

Bits [1:0]

Reserved, RAZ/WI.

Otherwise:

RES0

ITFSB, bit [37]

When FEAT_MTE2 is implemented:

When synchronous exceptions are not being generated by Tag Check Faults, this field controls whether on exception entry into EL2, all Tag Check Faults due to instructions executed before exception entry, that are reported asynchronously, are synchronized into TFSRE0_EL1, TFSR_EL1 and TFSR_EL2 registers.

ITFSB	Meaning
0b0	Tag Check Faults are not synchronized on entry to EL2.
0b1	Tag Check Faults are synchronized on entry to EL2.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

RES0

BT, bit [36]

When FEAT_BTI is implemented:

PAC Branch Type compatibility at EL2.

When HCR_EL2.{E2H, TGE} == {1, 1}, this bit is named BT1.

BT	Meaning
0b0	When the PE is executing at EL2, PACIASP and PACIBSP are compatible with PSTATE.BTYPE == 0b11.
0b1	When the PE is executing at EL2, PACIASP and PACIBSP are not compatible with PSTATE.BTYPE == 0b11.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

RES0

BT0, bit [35]

When FEAT_BTI is implemented, HCR_EL2.E2H == 1 and HCR_EL2.TGE == 1

BT0, bit [0] of bit [35]

PAC Branch Type compatibility at EL0.

BT0	Meaning
0b0	When the PE is executing at EL0, PACIASP and PACIBSP are compatible with PSTATE.BTYPE == 0b11.
0b1	When the PE is executing at EL0, PACIASP and PACIBSP are not compatible with PSTATE.BTYPE == 0b11.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

When FEAT_BTI is implemented, HCR_EL2.E2H == 1 and HCR_EL2.TGE == 0

Bit [0]

Reserved, RAZ/WI.

Otherwise:

RES0

Bit [34]

Reserved, RES0.

MSCEn, bit [33]

When FEAT_MOPS is implemented, HCR_EL2.E2H == 1 and HCR_EL2.TGE == 1

MSCEn, bit [0] of bit [33]

Memory Copy and Memory Set instructions Enable. Enables execution of the Memory Copy and Memory Set instructions at EL0.

MSCEn	Meaning
0b0	Execution of the Memory Copy and Memory Set instructions is UNDEFINED at EL0.
0b1	This control does not cause any instructions to be UNDEFINED.

When FEAT_MOPS is implemented and HCR_EL2.{E2H, TGE} is not {1, 1}, the Effective value of this bit is 0b1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

When FEAT_MOPS is implemented, HCR_EL2.E2H == 1 and HCR_EL2.TGE == 0

Bit [0]

Reserved, RAZ/WI.

Otherwise:

RES0

CMOW, bit [32]

When FEAT_CMOW is implemented and HCR_EL2.E2H == 1:

Controls cache maintenance instruction permission for the following instructions executed at EL0.

- IC IVAU, DC CIVAC, DC CIGDVAC and DC CIGVAC.

CMOW	Meaning
0b0	These instructions executed at EL0 with stage 1 read permission, but without stage 1 write permission, do not generate a stage 1 permission fault.
0b1	If enabled as a result of SCTLR_EL2.UCI==1 , these instructions executed at EL0 with stage 1 read permission, but without stage 1 write permission, generate a stage 1 permission fault.

When HCR_EL2.TGE is 0, this bit has no effect on execution at EL0.

For this control, stage 1 has write permission if all of the following apply:

- AP[2] is 0 or DBM is 1 in the stage 1 descriptor.
- Where APTable is in use, APTable[1] is 0 for all levels of the translation table.

This bit is permitted to be cached in a TLB.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

RES0

EnIA, bit [31]

When FEAT_PAuth is implemented:

Controls enabling of pointer authentication (using the APIAKey_EL1 key) of instruction addresses in the EL2 or EL2&0 translation regime.

For more information, see System register control of pointer authentication .

EnIA	Meaning
0b0	Pointer authentication (using the APIAKey_EL1 key) of instruction addresses is not enabled.
0b1	Pointer authentication (using the APIAKey_EL1 key) of instruction addresses is enabled.

This field controls the behavior of the AddPACIA and AuthIA pseudocode functions. Specifically, when the field is 1, AddPACIA returns a copy of a pointer to which a pointer authentication code has been added, and AuthIA returns an authenticated copy of a pointer. When the field is 0, both of these functions are NOP.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

RES0

EnIB, bit [30]

When FEAT_PAAuth is implemented:

Controls enabling of pointer authentication (using the APIBKey_EL1 key) of instruction addresses in the EL2 or EL2&0 translation regime.

For more information, see System register control of pointer authentication .

EnIB	Meaning
0b0	Pointer authentication (using the APIBKey_EL1 key) of instruction addresses is not enabled.
0b1	Pointer authentication (using the APIBKey_EL1 key) of instruction addresses is enabled.

This field controls the behavior of the AddPACIB and AuthIB pseudocode functions. Specifically, when the field is 1, AddPACIB returns a copy of a pointer to which a pointer authentication code has been added, and AuthIB returns an authenticated copy of a pointer. When the field is 0, both of these functions are NOP.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

RES0

LSMAOE, bit [29]

When FEAT_LSMAOC is implemented, HCR_EL2.E2H == 1 and HCR_EL2.TGE == 1

LSMAOE, bit [0] of bit [29]

Load Multiple and Store Multiple Atomicity and Ordering Enable.

LSMAOE	Meaning
0b0	For all memory accesses at EL0, A32 and T32 Load Multiple and Store Multiple can have an interrupt taken during the sequence memory accesses, and the memory accesses are not required to be ordered.
0b1	The ordering and interrupt behavior of A32 and T32 Load Multiple and Store Multiple at EL0 is as defined for Armv8.0.

This bit is permitted to be cached in a TLB.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

When FEAT_LSMAOC is implemented, HCR_EL2.E2H == 1 and HCR_EL2.TGE == 0

Bit [0]

Reserved, RAZ/WI.

Otherwise:

RES1

nTLSMD, bit [28]

When FEAT_LSMAOC is implemented, HCR_EL2.E2H == 1 and HCR_EL2.TGE == 1

nTLSMD, bit [0] of bit [28]

No Trap Load Multiple and Store Multiple to Device-nGRE/Device-nGnRE/Device-nGnRnE memory.

nTLSMD	Meaning
0b0	All memory accesses by A32 and T32 Load Multiple and Store Multiple at EL0 that are marked at stage 1 as Device-nGRE/Device-nGnRE/Device-nGnRnE memory are trapped and generate a stage 1 Alignment fault.
0b1	All memory accesses by A32 and T32 Load Multiple and Store Multiple at EL0 that are marked at stage 1 as Device-nGRE/Device-nGnRE/Device-nGnRnE memory are not trapped.

This bit is permitted to be cached in a TLB.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

When FEAT_LSMAOC is implemented, HCR_EL2.E2H == 1 and HCR_EL2.TGE == 0

Bit [0]

Reserved, RAZ/WI.

Otherwise:

RES1

EnDA, bit [27]

When FEAT_PAuth is implemented:

Controls enabling of pointer authentication (using the APDAKey_EL1 key) of instruction addresses in the EL2 or EL2&0 translation regime.

For more information, see System register control of pointer authentication .

EnDA	Meaning
0b0	Pointer authentication (using the APDAKey_EL1 key) of data addresses is not enabled.
0b1	Pointer authentication (using the APDAKey_EL1 key) of data addresses is enabled.

This field controls the behavior of the AddPACDA and AuthDA pseudocode functions. Specifically, when the field is 1, AddPACDA returns a copy of a pointer to which a pointer authentication code has been added, and AuthDA returns an authenticated copy of a pointer. When the field is 0, both of these functions are NOP.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

RES0

UCI, bit [26]

When $HCR_EL2.E2H == 1$ and $HCR_EL2.TGE == 1$

UCI, bit [0] of bit [26]

Traps execution of cache maintenance instructions at EL0 to EL2, from AArch64 state only. This applies to DC CVAU, DC CIVAC, DC CVAC, DC CVAP, and IC IVAU.

If FEAT_DPB2 is implemented, this trap also applies to DC CVADP.

If FEAT_MTE is implemented, this trap also applies to DC CIGVAC, DC CIGDVAC, DC CGVAC, DC CGDVAC, DC CGVAP, and DC CGDVAP.

If FEAT_DPB2 and FEAT_MTE are implemented, this trap also applies to DC CGVADP and DC CGDVADP.

UCI	Meaning
0b0	Any attempt to execute an instruction that this trap applies to at EL0 using AArch64 is trapped to EL2.
0b1	This control does not cause any instructions to be trapped.

If the Point of Coherency is before any level of data cache, it is IMPLEMENTATION DEFINED whether the execution of any data or unified cache clean, or clean and invalidate instruction that operates by VA to the point of coherency can be trapped when the value of this control is 1.

If the Point of Unification is before any level of data cache, it is IMPLEMENTATION DEFINED whether the execution of any data or unified cache clean by VA to the Point of Unification instruction can be trapped when the value of this control is 1.

If the Point of Unification is before any level of instruction cache, it is IMPLEMENTATION DEFINED whether the execution of any instruction cache invalidate by VA to the Point of Unification instruction can be trapped when the value of this control is 1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

When $HCR_EL2.E2H == 1$ and $HCR_EL2.TGE == 0$

Bit [0]

Reserved, RAZ/WI.

Otherwise:

RES0

EE, bit [25]

Endianness of data accesses at EL2, stage 1 translation table walks in the EL2 or EL2&0 translation regime, and stage 2 translation table walks in the EL1&0 translation regime.

EE	Meaning
0b0	Explicit data accesses at EL2, stage 1 translation table walks in the EL2 or EL2&0 translation regime, and stage 2 translation table walks in the EL1&0 translation regime are little-endian.
0b1	Explicit data accesses at EL2, stage 1 translation table walks in the EL2 or EL2&0 translation regime, and stage 2 translation table walks in the EL1&0 translation regime are big-endian.

If an implementation does not provide Big-endian support at Exception levels higher than EL0, this bit is RES0.

If an implementation does not provide Little-endian support at Exception levels higher than EL0, this bit is RES1.

The EE bit is permitted to be cached in a TLB.

The reset behavior of this field is:

- On a Warm reset, this field resets to an IMPLEMENTATION DEFINED value.

E0E, bit [24]

When $HCR_EL2.E2H == 1$ and $HCR_EL2.TGE == 1$

E0E, bit [0] of bit [24]

Endianness of data accesses at EL0.

E0E	Meaning
0b0	Explicit data accesses at EL0 are little-endian.
0b1	Explicit data accesses at EL0 are big-endian.

If an implementation only supports Little-endian accesses at EL0, then this bit is RES0. This option is not permitted when $SCTLR_EL1.EE$ is RES1.

If an implementation only supports Big-endian accesses at EL0, then this bit is RES1. This option is not permitted when $SCTLR_EL1.EE$ is RES0.

This bit has no effect on the endianness of $LDTR$, $LDTRH$, $LDTRSH$, $LDTRSW$, $STTR$, and $STTRH$ instructions executed at EL1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

When $HCR_EL2.E2H == 1$ and $HCR_EL2.TGE == 0$

Bit [0]

Reserved, RAZ/WI.

Otherwise:

RES0

SPAN, bit [23]

When $HCR_EL2.E2H == 1$ and $HCR_EL2.TGE == 1$

SPAN, bit [0] of bit [23]

Set Privileged Access Never, on taking an exception to EL2.

SPAN	Meaning
0b0	PSTATE.PAN is set to 1 on taking an exception to EL2.
0b1	The value of PSTATE.PAN is left unchanged on taking an exception to EL2.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

When $HCR_EL2.E2H == 1$ and $HCR_EL2.TGE == 0$

Bit [0]

Reserved, RAZ/WI.

Otherwise:

RES1

EIS, bit [22]

When FEAT_ExS is implemented:

Exception entry is a context synchronization event.

EIS	Meaning
0b0	The taking of an exception to EL2 is not a context synchronization event.
0b1	The taking of an exception to EL2 is a context synchronization event.

If SCTLRL_EL2.EIS is set to 0b0:

- Indirect writes to ESR_EL2, FAR_EL2, SPSR_EL2, ELR_EL2, and HPFAR_EL2 are synchronized on exception entry to EL2, so that a direct read of the register after exception entry sees the indirectly written

value caused by the exception entry.

- Memory transactions, including instruction fetches, from an Exception level always use the translation resources associated with that translation regime.
- Exception Catch debug events are synchronous debug events.
- DCPS* and DRPS instructions are context synchronization events.

The following are not affected by the value of SCTL_R_EL2.EIS:

- Changes to the PSTATE information on entry to EL2.
- Behavior of accessing the banked copies of the stack pointer using the SP register name for loads, stores, and data processing instructions.
- Exit from Debug state.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

RES1

IESB, bit [21]

When FEAT_IESB is implemented:

Implicit Error Synchronization event enable.

IESB	Meaning
0b0	Disabled.
0b1	An implicit error synchronization event is added: <ul style="list-style-type: none"> • At each exception taken to EL2. • Before the operational pseudocode of each <code>ERET</code> instruction executed at EL2.

When the PE is in Debug state, the effect of this field is **CONSTRAINED UNPREDICTABLE**, and its Effective value might be 0 or 1 regardless of the value of the field. If the Effective value of the field is 1, then an implicit error synchronization event is added after each `DCPSX` instruction taken to EL2 and before each `DRPS` instruction executed at EL2, in addition to the other cases where it is added.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

RES0

TSCXT, bit [20]

When (FEAT_CSV2_2 is implemented or FEAT_CSV2_1p2 is implemented), HCR_EL2.E2H == 1 and HCR_EL2.TGE == 1

TSCXT, bit [0] of bit [20]

Trap EL0 Access to the SCXTNUM_EL0 register, when EL0 is using AArch64.

TSCXT	Meaning
0b0	EL0 access to SCXTNUM_EL0 is not disabled by this mechanism.
0b1	EL0 access to SCXTNUM_EL0 is disabled, causing an exception to EL2, and the SCXTNUM_EL0 value is treated as 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

When FEAT_CSV2_2 is not implemented, FEAT_CSV2_1p2 is not implemented, HCR_EL2.E2H == 1 and HCR_EL2.TGE == 1

Bit [0]

Reserved, RES1.

When (FEAT_CSV2_2 is implemented or FEAT_CSV2_1p2 is implemented), HCR_EL2.E2H == 1 and HCR_EL2.TGE == 0

Bit [0]

Reserved, RAZ/WI.

Otherwise:

RES0

WXN, bit [19]

Write permission implies XN (Execute-never). For the EL2 or EL2&0 translation regime, this bit can force all memory regions that are writable to be treated as XN.

WXN	Meaning
0b0	This control has no effect on memory access permissions.
0b1	Any region that is writable in the EL2 or EL2&0 translation regime is forced to XN for accesses from software executing at EL2.

This bit applies only when SCTL2_EL2.M bit is set.

The WXN bit is permitted to be cached in a TLB.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

nTWE, bit [18]

When HCR_EL2.E2H == 1 and HCR_EL2.TGE == 1

nTWE, bit [0] of bit [18]

Traps execution of WFE instructions at EL0 to EL2, from both Execution states.

nTWE	Meaning
0b0	Any attempt to execute a WFE instruction at EL0 is trapped to EL2, if the instruction would otherwise have caused the PE to enter a low-power state.
0b1	This control does not cause any instructions to be trapped.

In AArch32 state, the attempted execution of a conditional WFE instruction is only trapped if the instruction passes its condition code check.

Since a WFE or WFI can complete at any time, even without a Wakeup event, the traps on WFE or WFI are not guaranteed to be taken, even if the WFE or WFI is executed when there is no Wakeup event. The only guarantee is that if the instruction does not complete in finite time in the absence of a Wakeup event, the trap will be taken.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

When $HCR_EL2.E2H == 1$ and $HCR_EL2.TGE == 0$

Bit [0]

Reserved, RAZ/WI.

Otherwise:

RES1

Bit [17]

Reserved, RES0.

nTWI, bit [16]

When $HCR_EL2.E2H == 1$ and $HCR_EL2.TGE == 1$

nTWI, bit [0] of bit [16]

Traps execution of WFI instructions at EL0 to EL2, from both Execution states.

nTWI	Meaning
0b0	Any attempt to execute a WFI instruction at EL0 is trapped to EL2, if the instruction would otherwise have caused the PE to enter a low-power state.
0b1	This control does not cause any instructions to be trapped.

In AArch32 state, the attempted execution of a conditional WFI instruction is only trapped if the instruction passes its condition code check.

Since a WFE or WFI can complete at any time, even without a Wakeup event, the traps on WFE or WFI are not guaranteed to be taken, even if the WFE or WFI is executed when there is no Wakeup event. The only guarantee is that if the instruction does not complete in finite time in the absence of a Wakeup event, the trap will be taken.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

When $HCR_EL2.E2H == 1$ and $HCR_EL2.TGE == 0$

Bit [0]

Reserved, RAZ/WI.

Otherwise:

RES1

UCT, bit [15]

When $HCR_EL2.E2H == 1$ and $HCR_EL2.TGE == 1$

UCT, bit [0] of bit [15]

Traps EL0 accesses to the CTR_EL0 to EL2, from AArch64 state only.

UCT	Meaning
0b0	Accesses to the CTR_EL0 from EL0 using AArch64 are trapped to EL2.
0b1	This control does not cause any instructions to be trapped.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

When $HCR_EL2.E2H == 1$ and $HCR_EL2.TGE == 0$

Bit [0]

Reserved, RAZ/WI.

Otherwise:

RES0

DZE, bit [14]

When $HCR_EL2.E2H == 1$ and $HCR_EL2.TGE == 1$

DZE, bit [0] of bit [14]

Traps execution of DC ZVA instructions at EL0 to EL2, from AArch64 state only.

If FEAT_MTE is implemented, this trap also applies to DC GVA and DC GZVA.

DZE	Meaning
0b0	Any attempt to execute an instruction that this trap applies to at EL0 using AArch64 is trapped to EL2. Reading DCZID_EL0.DZP from EL0 returns 1, indicating that the instructions that this trap applies to are not supported.
0b1	This control does not cause any instructions to be trapped.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

When $HCR_EL2.E2H == 1$ and $HCR_EL2.TGE == 0$

Bit [0]

Reserved, RAZ/WI.

Otherwise:

RES0

EnDB, bit [13]

When $FEAT_PAuth$ is implemented:

Controls enabling of pointer authentication (using the APDBKey_EL1 key) of instruction addresses in the EL2 or EL2&0 translation regime.

For more information, see System register control of pointer authentication .

EnDB	Meaning
0b0	Pointer authentication (using the APDBKey_EL1 key) of data addresses is not enabled.
0b1	Pointer authentication (using the APDBKey_EL1 key) of data addresses is enabled.

This field controls the behavior of the AddPACDB and AuthDB pseudocode functions. Specifically, when the field is 1, AddPACDB returns a copy of a pointer to which a pointer authentication code has been added, and AuthDB returns an authenticated copy of a pointer. When the field is 0, both of these functions are NOP.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

RES0

I, bit [12]

Instruction access Cacheability control, for accesses at EL2 and, when EL2 is enabled in the current Security state and $HCR_EL2.\{E2H, TGE\} == \{1, 1\}$, EL0.

I	Meaning
0b0	All instruction accesses to Normal memory from EL2 are Non-cacheable for all levels of instruction and unified cache. When EL2 is enabled in the current Security state and $HCR_EL2.\{E2H, TGE\} == \{1, 1\}$, all instruction accesses to Normal memory from EL0 are Non-cacheable for all levels of instruction and unified cache. If SCTL2_EL2.M is 0, instruction accesses from stage 1 of the EL2 or EL2&0 translation regime are to Normal, Outer Shareable, Inner Non-cacheable, Outer Non-cacheable memory.

I	Meaning
0b1	<p>This control has no effect on the Cacheability of instruction access to Normal memory from EL2 and, when EL2 is enabled in the current Security state and HCR_EL2.{E2H, TGE} == {1, 1}, instruction access to Normal memory from EL0.</p> <p>If the value of SCTL2_EL2.M is 0, instruction accesses from stage 1 of the EL2 or EL2&0 translation regime are to Normal, Outer Shareable, Inner Write-Through, Outer Write-Through memory.</p>

This bit has no effect on the EL3 translation regime.

When EL2 is disabled in the current Security state or HCR_EL2.{E2H,TGE} != {1,1}, this bit has no effect on the EL1&0 translation regime.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0b0.

EOS, bit [11]

When FEAT_ExS is implemented:

Exception exit is a context synchronization event.

EOS	Meaning
0b0	An exception return from EL2 is not a context synchronization event.
0b1	An exception return from EL2 is a context synchronization event.

If SCTL2_EL2.EOS is set to 0b0:

- Memory transactions, including instruction fetches, from an Exception level always use the translation resources associated with that translation regime.
- Exception Catch debug events are synchronous debug events.
- DCPS* and DRPS instructions are context synchronization events.

The following are not affected by the value of SCTL2_EL2.EOS:

- The indirect write of the PSTATE and PC values from SPSR_EL2 and ELR_EL2 on exception return is synchronized.
- Behavior of accessing the banked copies of the stack pointer using the SP register name for loads, stores, and data processing instructions.
- Exit from Debug state.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

RES1

EnRCTX, bit [10]

When FEAT_SPECRES is implemented, HCR_EL2.E2H == 1 and HCR_EL2.TGE == 1

EnRCTX, bit [0] of bit [10]

Enable EL0 Access to the following instructions:

- AArch32 CFPRCTX, DVPRCTX and CPPRCTX instructions.
- AArch64 CFP RCTX, DVP RCT and CPP RCTX instructions.

EnRCTX	Meaning
0b0	EL0 access to these instructions is disabled, and these instructions are trapped to EL1.
0b1	EL0 access to these instructions is enabled.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

When FEAT_SPECRES is implemented, HCR_EL2.E2H == 1 and HCR_EL2.TGE == 0

Bit [0]

Reserved, RAZ/WI.

Otherwise:

RES0

Bit [9]

Reserved, RES0.

SED, bit [8]

When EL0 is capable of using AArch32, HCR_EL2.E2H == 1 and HCR_EL2.TGE == 1

SED, bit [0] of bit [8]

SETEND instruction disable. Disables SETEND instructions at EL0 using AArch32.

SED	Meaning
0b0	SETEND instruction execution is enabled at EL0 using AArch32.
0b1	SETEND instructions are UNDEFINED at EL0 using AArch32.

If the implementation does not support mixed-endian operation at any Exception level, this bit is RES1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

When EL0 can only use AArch64, HCR_EL2.E2H == 1 and HCR_EL2.TGE == 1

Bit [0]

Reserved, RES1.

When EL0 is capable of using AArch32, HCR_EL2.E2H == 1 and HCR_EL2.TGE == 0

Bit [0]

Reserved, RAZ/WI.

Otherwise:

RES0

ITD, bit [7]

When EL0 is capable of using AArch32, HCR_EL2.E2H == 1 and HCR_EL2.TGE == 1

ITD, bit [0] of bit [7]

IT Disable. Disables some uses of IT instructions at EL0 using AArch32.

ITD	Meaning
0b0	All IT instruction functionality is enabled at EL0 using AArch32.

ITD	Meaning
0b1	<p>Any attempt at EL0 using AArch32 to execute any of the following is UNDEFINED:</p> <ul style="list-style-type: none"> • All encodings of the IT instruction with $hw1[3:0] \neq 1000$. • All encodings of the subsequent instruction with the following values for $hw1$: <ul style="list-style-type: none"> – 0b11xxxxxxxxxxxx: All 32-bit instructions, and the 16-bit instructions B, UDF, SVC, LDM, and STM. – 0b1011xxxxxxxxxxxx: All instructions in ‘Miscellaneous 16-bit instructions’ in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section F3.2.5. – 0b10100xxxxxxxxxxx: ADD Rd, PC, #imm – 0b01001xxxxxxxxxxx: LDR Rd, [PC, #imm] – 0b0100x1xxx1111xxx: ADD Rdn, PC; CMP Rn, PC; MOV Rd, PC; BX PC; BLX PC. – 0b010001xx1xxx111: ADD PC, Rm; CMP PC, Rm; MOV PC, Rm. This pattern also covers UNPREDICTABLE cases with BLX Rn. <p>These instructions are always UNDEFINED, regardless of whether they would pass or fail the condition code check that applies to them as a result of being in an IT block. It is IMPLEMENTATION DEFINED whether the IT instruction is treated as:</p> <ul style="list-style-type: none"> • A 16-bit instruction, that can only be followed by another 16-bit instruction. • The first half of a 32-bit instruction. <p>This means that, for the situations that are UNDEFINED, either the second 16-bit instruction or the 32-bit instruction is UNDEFINED.</p> <p>An implementation might vary dynamically as to whether IT is treated as a 16-bit instruction or the first half of a 32-bit instruction.</p>

If an instruction in an active IT block that would be disabled by this field sets this field to 1 then behavior is CONSTRAINED UNPREDICTABLE. For more information see Changes to an ITD control by an instruction in an IT block .

ITD is optional, but if it is implemented in the SCTLR_EL2 then it must also be implemented in the [SCTLR_EL1](#), HSCTLR, and SCTLR.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

When an implementation does not implement ITD, access to this field is **RAZ/WI**.

When EL0 can only use AArch64, $HCR_EL2.E2H == 1$ and $HCR_EL2.TGE == 1$

Bit [0]

Reserved, RES1.

When EL0 is capable of using AArch32, HCR_EL2.E2H == 1 and HCR_EL2.TGE == 0

Bit [0]

Reserved, RAZ/WI.

Otherwise:

RES0

nAA, bit [6]

When FEAT_LSE2 is implemented:

Non-aligned access. This bit controls generation of Alignment faults under certain conditions at EL2, and, when EL2 is enabled in the current Security state and HCR_EL2.{E2H, TGE} == {1, 1}, EL0.

nAA	Meaning
0b0	LDAPR, LDAPRH, LDAPUR, LDAPURH, LDAPURSH, LDAPURSW, LDAR, LDARH, LDLAR, LDLARH, STLLR, STLLRH, STLR, STLRH, STLUR, and STLURH generate an Alignment fault if all bytes being accessed are not within a single 16-byte quantity, aligned to 16 bytes for accesses.
0b1	This control bit does not cause LDAPR, LDAPRH, LDAPUR, LDAPURH, LDAPURSH, LDAPURSW, LDAR, LDARH, LDLAR, LDLARH, STLLR, STLLRH, STLR, STLRH, STLUR, or STLURH to generate an Alignment fault if all bytes being accessed are not within a single 16-byte quantity, aligned to 16 bytes.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

RES0

CP15BEN, bit [5]

When EL0 is capable of using AArch32, HCR_EL2.E2H == 1 and HCR_EL2.TGE == 1

CP15BEN, bit [0] of bit [5]

System instruction memory barrier enable. Enables accesses to the DMB, DSB, and ISB System instructions in the (coproc==0b1111) encoding space from EL0:

CP15BEN	Meaning
0b0	EL0 using AArch32: EL0 execution of the CP15DMB, CP15DSB, and CP15ISB instructions is UNDEFINED.
0b1	EL0 using AArch32: EL0 execution of the CP15DMB, CP15DSB, and CP15ISB instructions is enabled.

CP15BEN is optional, but if it is implemented in the SCTLR_EL2 then it must also be implemented in the [SCTLR_EL1](#), HSCTLR, and SCTLR.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

When an implementation does not implement CP15BEN, access to this field is **RAO/WI**.

When EL0 can only use AArch64, $HCR_EL2.E2H == 1$ and $HCR_EL2.TGE == 1$

Bit [0]

Reserved, RES0.

When EL0 is capable of using AArch32, $HCR_EL2.E2H == 1$ and $HCR_EL2.TGE == 0$

Bit [0]

Reserved, RAZ/WI.

Otherwise:

RES1

SA0, bit [4]

When $HCR_EL2.E2H == 1$ and $HCR_EL2.TGE == 1$:

SP Alignment check enable for EL0. When set to 1, if a load or store instruction executed at EL0 uses the SP as the base address and the SP is not aligned to a 16-byte boundary, then an SP alignment fault exception is generated. For more information, see ‘SP alignment checking’.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

RES1

SA, bit [3]

SP Alignment check enable. When set to 1, if a load or store instruction executed at EL2 uses the SP as the base address and the SP is not aligned to a 16-byte boundary, then an SP alignment fault exception is generated. For more information, see ‘SP alignment checking’.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

C, bit [2]

Data access Cacheability control, for accesses at EL2 and, when EL2 is enabled in the current Security state and $HCR_EL2.\{E2H, TGE\} == \{1, 1\}$, EL0

C	Meaning
0b0	<p>The following are Non-cacheable for all levels of data and unified cache:</p> <ul style="list-style-type: none"> • Data accesses to Normal memory from EL2. • When $\text{HCR_EL2}\{E2H, TGE\} \neq \{1, 1\}$, Normal memory accesses to the EL2 translation tables. • When EL2 is enabled in the current Security state and $\text{HCR_EL2}\{E2H, TGE\} = \{1, 1\}$: <ul style="list-style-type: none"> – Data accesses to Normal memory from EL0. – Normal memory accesses to the EL2&0 translation tables.
0b1	<p>This control has no effect on the Cacheability of:</p> <ul style="list-style-type: none"> • Data access to Normal memory from EL2. • When $\text{HCR_EL2}\{E2H, TGE\} \neq \{1, 1\}$, Normal memory accesses to the EL2 translation tables. • When EL2 is enabled in the current Security state and $\text{HCR_EL2}\{E2H, TGE\} = \{1, 1\}$: <ul style="list-style-type: none"> – Data accesses to Normal memory from EL0. – Normal memory accesses to the EL2&0 translation tables.

This bit has no effect on the EL3 translation regime.

When EL2 is disabled in the current Security state or $\text{HCR_EL2}\{E2H, TGE\} \neq \{1, 1\}$, this bit has no effect on the EL1&0 translation regime.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0b0.

A, bit [1]

Alignment check enable. This is the enable bit for Alignment fault checking at EL2 and, when EL2 is enabled in the current Security state and $\text{HCR_EL2}\{E2H, TGE\} = \{1, 1\}$, EL0.

A	Meaning
0b0	<p>Alignment fault checking disabled when executing at EL2. When EL2 is enabled in the current Security state and $\text{HCR_EL2}\{E2H, TGE\} = \{1, 1\}$, alignment fault checking disabled when executing at EL0.</p> <p>Instructions that load or store one or more registers, other than load/store exclusive and load-acquire/store-release, do not check that the address being accessed is aligned to the size of the data element(s) being accessed.</p>

A	Meaning
0b1	Alignment fault checking enabled when executing at EL2. When EL2 is enabled in the current Security state and $\text{HCR_EL2}\{E2H, TGE\} == \{1, 1\}$, alignment fault checking enabled when executing at EL0. All instructions that load or store one or more registers have an alignment check that the address being accessed is aligned to the size of the data element(s) being accessed. If this check fails it causes an Alignment fault, which is taken as a Data Abort exception.

Load/store exclusive and load-acquire/store-release instructions have an alignment check regardless of the value of the A bit.

If FEAT_MOPS is implemented, SETG* instructions have an alignment check regardless of the value of the A bit.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

M, bit [0]

MMU enable for EL2 or EL2&0 stage 1 address translation.

M	Meaning
0b0	When $\text{HCR_EL2}\{E2H, TGE\} \neq \{1, 1\}$, EL2 stage 1 address translation disabled. When $\text{HCR_EL2}\{E2H, TGE\} == \{1, 1\}$, EL2&0 stage 1 address translation disabled. See the SCTLR_EL2.I field for the behavior of instruction accesses to Normal memory.
0b1	When $\text{HCR_EL2}\{E2H, TGE\} \neq \{1, 1\}$, EL2 stage 1 address translation enabled. When $\text{HCR_EL2}\{E2H, TGE\} == \{1, 1\}$, EL2&0 stage 1 address translation enabled.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0b0.

Accessing the SCTLR_EL2

When HCR_EL2.E2H is 1, without explicit synchronization, access from EL2 using the mnemonic SCTLR_EL2 or SCTLR_EL1 are not guaranteed to be ordered with respect to accesses using the other mnemonic.

Accesses to this register use the following encodings in the instruction encoding space:

MRS <Xt>, SCTLR_EL2

op0	op1	CRn	CRm	op2
0b11	0b100	0b0001	0b0000	0b000

```

1 if PSTATE.EL == EL0 then
2     UNDEFINED;
3 elseif PSTATE.EL == EL1 then
4     if EL2Enabled() && HCR_EL2.NV == '1' then
5         AArch64.SystemAccessTrap(EL2, 0x18);
6     else
7         UNDEFINED;
8 elseif PSTATE.EL == EL2 then
9     X[t, 64] = SCTLR_EL2;
10 elseif PSTATE.EL == EL3 then
11     X[t, 64] = SCTLR_EL2;

```

MSR SCTLR_EL2, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b100	0b0001	0b0000	0b000

```

1 if PSTATE.EL == EL0 then
2     UNDEFINED;
3 elseif PSTATE.EL == EL1 then
4     if EL2Enabled() && HCR_EL2.NV == '1' then
5         AArch64.SystemAccessTrap(EL2, 0x18);
6     else
7         UNDEFINED;
8 elseif PSTATE.EL == EL2 then
9     SCTLR_EL2 = X[t, 64];
10 elseif PSTATE.EL == EL3 then
11     SCTLR_EL2 = X[t, 64];

```

MRS <Xt>, SCTLR_EL1

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

```

1 if PSTATE.EL == EL0 then
2     UNDEFINED;
3 elseif PSTATE.EL == EL1 then
4     if EL2Enabled() && HCR_EL2.TRVM == '1' then
5         AArch64.SystemAccessTrap(EL2, 0x18);
6     elseif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGTR_EL2.SCTLR_EL1 == '1' then
7         AArch64.SystemAccessTrap(EL2, 0x18);
8     elseif EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '111' then
9         X[t, 64] = NVMem[0x110];
10    else
11        X[t, 64] = SCTLR_EL1;
12 elseif PSTATE.EL == EL2 then
13    if HCR_EL2.E2H == '1' then
14        X[t, 64] = SCTLR_EL2;
15    else
16        X[t, 64] = SCTLR_EL1;
17 elseif PSTATE.EL == EL3 then
18    X[t, 64] = SCTLR_EL1;

```

MSR SCTLR_EL1, <Xt>

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

```

1  if PSTATE.EL == EL0 then
2      UNDEFINED;
3  elsif PSTATE.EL == EL1 then
4      if EL2Enabled() && HCR_EL2.TVM == '1' then
5          AArch64.SystemAccessTrap(EL2, 0x18);
6      elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGWTR_EL2.SCTLR_EL1 == '1' then
7          AArch64.SystemAccessTrap(EL2, 0x18);
8      elsif EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '111' then
9          NVMem[0x110] = X[t, 64];
10     else
11         SCTLR_EL1 = X[t, 64];
12 elsif PSTATE.EL == EL2 then
13     if HCR_EL2.E2H == '1' then
14         SCTLR_EL2 = X[t, 64];
15     else
16         SCTLR_EL1 = X[t, 64];
17 elsif PSTATE.EL == EL3 then
18     SCTLR_EL1 = X[t, 64];

```

E3.2.18 EDDEVID1, External Debug Device ID register 1

The EDDEVID1 characteristics are:

Purpose

Provides extra information for external debuggers about features of the debug implementation.

Attributes

EDDEVID1 is a 32-bit register.

Configuration

If FEAT_DoPD is implemented, this register is in the Core power domain.

If FEAT_DoPD is not implemented, this register is in the Debug power domain.

Field descriptions

The EDDEVID1 bit assignments are:



Bits [31:8]

Reserved, RES0.

HSR, bits [7:4]

Indicates support for the External Debug Halt Status Register ([EDHSR](#)). Defined values are:

HSR	Meaning
0b0000	EDHSR is not implemented. The PE follows behaviors consistent with the EDHSR fields having a zero value.
0b0001	EDHSR is implemented.

All other values are reserved.

If FEAT_SME is implemented, the permitted values are 0b0000 and 0b0001.

If FEAT_SME is not implemented, the only permitted value is 0b0000.

PCSROffset, bits [3:0]

Indicates the offset applied to PC samples returned by reads of EDPCSR. Permitted values of this field in Armv8 are:

PCSROffset	Meaning
0b0000	EDPCSR not implemented.
0b0010	EDPCSR implemented, and samples have no offset applied and do not sample the instruction set state in AArch32 state.

When FEAT_PCSRv8p2 is implemented, the only permitted value is 0b0000.

FEAT_PCSRv8p2 implements the PC Sample-based Profiling Extension in the Performance Monitors register space, as indicated by the value of PMDEVID.PCSample.

Accessing the EDDEVID1

Accesses to this register use the following encodings in the instruction encoding space:

EDDEVID1 can be accessed through the external debug interface:

Component	Offset	Instance
Debug	0xFC4	EDDEVID1

This interface is accessible as follows:

- When FEAT_DoPD is not implemented or IsCorePowered() access to this register is **RO**.
- Otherwise access to this register returns an ERROR.

Chapter E4

Glossary terms

Effective Non-streaming SVE vector length

The Non-streaming SVE vector length in bits at the current Exception level, is an implementation-supported multiple of 128 bits up to the Maximum implemented Non-streaming SVE vector length, further constrained by ZCR_ELx at the current and higher Exception levels.

Effective Streaming SVE vector length

The Streaming SVE vector length in bits at the current Exception level is an implementation-supported power of two from 128 up to the Maximum implemented Streaming SVE vector length, further constrained by SMCR_ELx at the current and higher Exception levels.

Effective SVE vector length

The vector length in bits that applies to the execution of SVE instructions at the current Exception level is the Effective Streaming SVE vector length when the PE is in Streaming SVE mode, otherwise it is the Effective Non-streaming SVE vector length.

Maximum implemented Non-streaming SVE vector length

The maximum Non-streaming SVE vector length in bits supported by the implementation.

Maximum implemented Streaming SVE vector length

The maximum Streaming SVE vector length in bits supported by the implementation.

Scalable Matrix Extension

Defines architectural state capable of holding two-dimensional matrix tiles, and a Streaming SVE mode which supports execution of SVE2 instructions with a vector length that matches the tile width, along with instructions that accumulate the outer product of two vectors into a tile, as well as load, store, and move instructions that

transfer a vector to or from a tile row or column. The extension also defines System registers and fields that identify the presence and capabilities of SME, and enable and control its behavior at each Exception level.

SMCU

Streaming Mode Compute Unit.

SME

Scalable Matrix Extension.

Streaming execution

Execution of instructions by a PE when that PE is in Streaming SVE mode.

Streaming Mode Compute Unit

Where more than one PE shares resources for Streaming execution of SVE and SME instructions, those shared resources are called a Streaming Mode Compute Unit (SMCU).

Streaming SVE mode

An execution mode that supports a substantial subset of the SVE2 instruction set and architectural state with a vector length that matches the width of SME tiles, which may be different from the vector length when the PE is not in Streaming SVE.

Streaming SVE register state

The registers *Z0-Z31*, *P0-P15*, and *FFR* that are accessed by SVE and SME instructions when the PE is in Streaming SVE mode.

SVL

Effective Streaming SVE vector length.

VL

Effective SVE vector length.