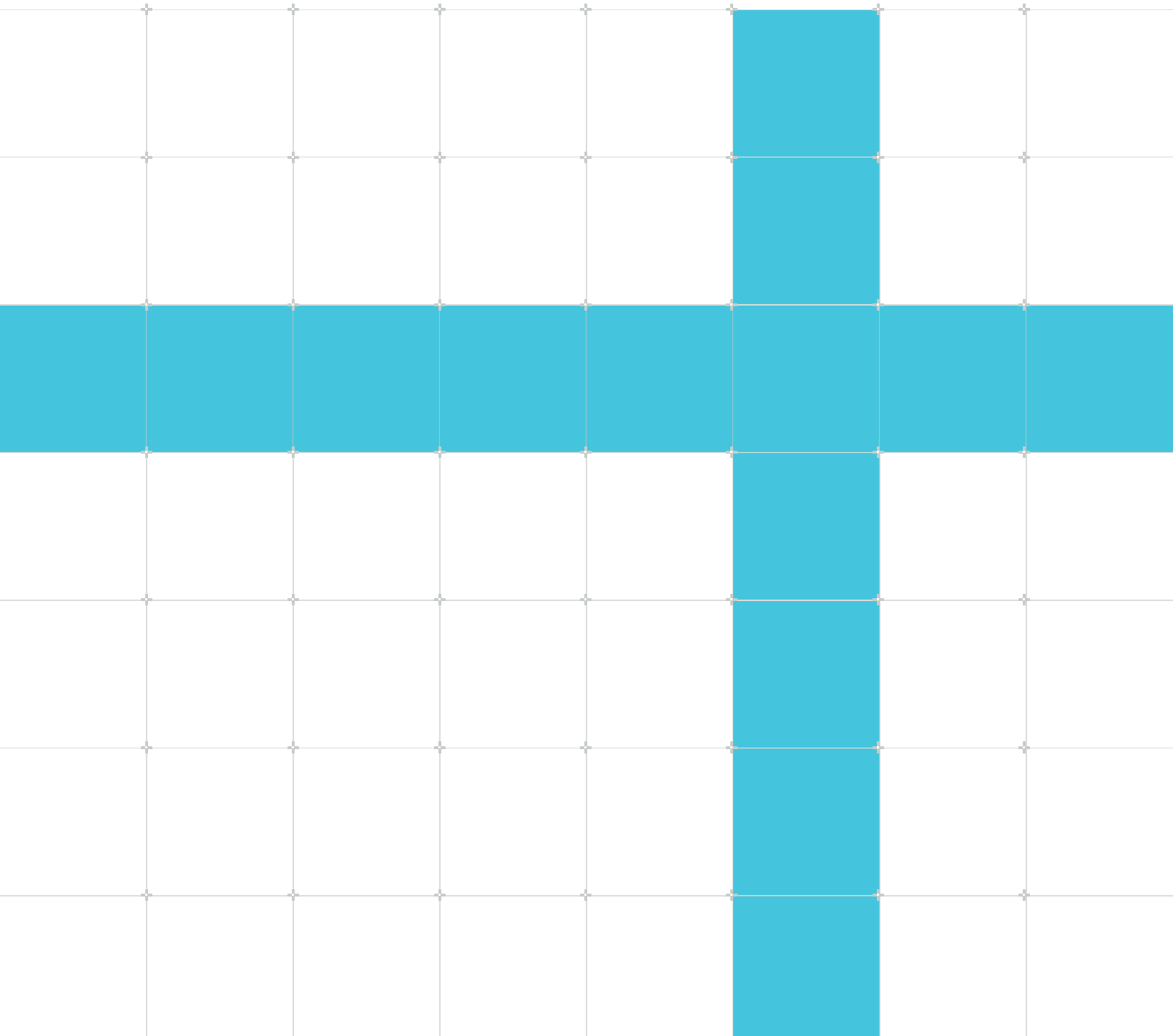
Arm® Architecture Reference Manual for A-profile architecture

Known issues in Issue L.a

Issue 02
102105_L.a_02_en



Arm® Architecture Reference Manual for A-profile architecture

Known issues in Issue L.a

Release information

Document history

Issue	Date	Confidentiality	Change
L.a-02	7 February 2025	Non-Confidential	Known Issues in Arm® Architecture Reference Manual, Issue L.a, as of 7 February 2025
L.a-01	7 January 2025	Non-Confidential	Known Issues in Arm® Architecture Reference Manual, Issue L.a, as of 7 January 2025
L.a-00	2 December 2024	Non-Confidential	Known Issues in Arm® Architecture Reference Manual, Issue L.a, as of 2 December 2024
K.a-08	30 November 2024	Non-Confidential	Known Issues in Arm® Architecture Reference Manual, Issue K.a, as of 30 November 2024
J.a-08	4 March 2024	Non-Confidential	Known Issues in Arm® Architecture Reference Manual, Issue J.a, as of 4 March 2024
I.a-06	21 April 2023	Non-Confidential	Known Issues in Arm® Architecture Reference Manual, Issue I.a, as of 31 March 2023
H.a-06	22 July 2022	Non-Confidential	Known Issues in Arm® Architecture Reference Manual, Issue H.a, as of 22 July 2022
G.b-05	31 January 2022	Non-Confidential	Known Issues in Arm® Architecture Reference Manual, Issue G.b, as of 7 January 2022
F.c-04	18 December 2020	Non-Confidential	Known Issues in Arm® Architecture Reference Manual, Issue F.c, as of 18 December 2020

Proprietary Notice

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 the subject matter of this document infringes any third party patents.

The content of this document is informational only. Any solutions presented herein are subject to changing conditions, information, scope, and data. This document was produced using reasonable efforts based on information available as of the date of issue of this document. The scope of information in this document may exceed that which Arm is required to provide, and such additional information is merely intended to further assist the recipient and does not represent Arm's view of the scope of its obligations. You acknowledge and agree that you possess the necessary expertise in system security and functional safety and that you shall be solely responsible for compliance with all legal, regulatory, safety and security related requirements concerning your products, notwithstanding any information or support that may be provided by Arm herein. In addition, you are responsible for any applications which are used in conjunction with any Arm technology described in this document, and to minimize risks, adequate design and operating safeguards should be provided for by you.

This document may include technical inaccuracies or typographical errors. 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, any patents, copyrights, trade secrets, trademarks, or other rights.

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.

Reference by Arm to any third party's products or services within this document is not an express or implied approval or endorsement of the use thereof.

This document consists solely of commercial items. You shall be responsible for ensuring that any permitted 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 this document shall prevail.

The validity, construction and performance of this notice shall be governed by English Law.

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. Please follow Arm's trademark usage

Arm Limited. Company 02557590 registered in England.

110 Fulbourn Road, Cambridge, England CB1 9NJ.

PRE-1121-V1.0

Confidentiality Status

This document is Non-Confidential. The right to use, copy and disclose this document may be subject to license restrictions in accordance with the terms of the agreement entered into by Arm and the party that Arm delivered this document to.

Unrestricted Access is an Arm internal classification.

Product Status

The information in this document is Final, that is for a developed product.

Feedback

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

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

Inclusive language commitment

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

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

Contents

1. Introduction

1.1 Conventions

The following subsections describe conventions used in Arm documents.

Glossary

The Arm Glossary is a list of terms used in Arm documentation, together with definitions for those terms. The Arm Glossary does not contain terms that are industry standard unless the Arm meaning differs from the generally accepted meaning.

See the Arm Glossary for more information: developer.arm.com/glossary.

Typographic conventions

Arm documentation uses typographical conventions to convey specific meaning.

Convention	Use
italic	Citations.
bold	Interface elements, such as menu names. Terms in descriptive lists, where appropriate.
`monospace`	Text that you can enter at the keyboard, such as commands, file and program names, and source code.
`monospace underline`	A permitted abbreviation for a command or option. You can enter the underlined text instead of the full command or option name.
`<and>`	Encloses replaceable terms for assembler syntax where they appear in code or code fragments. For example: `MRC p15, 0, <Rd>, <CRn>, <CRm>, <Opcode_2>`
SMALL CAPITALS	Terms that have specific technical meanings as defined in the *Arm® Glossary*. For example, **IMPLEMENTATION DEFINED**, **IMPLEMENTATION SPECIFIC**, **UNKNOWN**, and **UNPREDICTABLE**.

We recommend the following. If you do not follow these recommendations your system might not work.

Your system requires the following. If you do not follow these requirements your system will not work.

You are at risk of causing permanent damage to your system or your equipment, or harming yourself.

This information is important and needs your attention.

A useful tip that might make it easier, better or faster to perform a task.

A reminder of something important that relates to the information you are reading.

1.2 Useful resources

This document contains information that is specific to this product. See the following resources for other useful information.

Access to Arm documents depends on their confidentiality:

- Non-Confidential documents are available at developer.arm.com/documentation. Each document link in the following tables goes to the online version of the document.
- Confidential documents are available to licensees only through the product package.

Arm product resources	Document ID	Confidentiality
Arm® Architecture Reference Manual for A-profile architecture, Issue L.a	DDI 0487L.a	Non-Confidential

1.3 Other information

See the Arm website for other relevant information.

- Arm® Developer.
- Arm® Documentation.
- Technical Support.
- Arm® Glossary.

2. Known issues

This document records known issues in the Arm Architecture Reference Manual for A-profile architecture (DDI 0487), Issue L.a.

Key

- C = Clarification.
- D = Defect.
- R = Relaxation.
- E = Enhancement.

2.1 D17119

In sections **F3.1.10.2 "Advanced SIMD two registers and shift amount"** and **F4.1.22.2 "Advanced SIMD two registers and shift amount"**, the following constraints are added to VMOVL:

- 'L' must be '0'.
- 'imm3H' cannot be '000'.

2.2 C17296

In section **D14.2 "The PMU event number space and common events"**, the text that reads:

> `0x0005`, L1D_TLB_REFILL, Level 1 data TLB refill
>
> The counter does not count the access if any of the following are true:
>
> - The access misses in the TLB and generates a translation table walk but does not cause a refill of the TLB.
> - The access is due to a TLB maintenance instruction.
> - The access generates a Translation fault because the applicable TCR_ELx.EPDy bit is 1.
> - FEAT_E0PD is implemented and the access is an unprivileged access that generates a Translation fault because the applicable TCR_ELx.E0PDy bit is 1.
> - FEAT_SVE is implemented and the access is a non-fault access that fails because the applicable TCR_ELx.NFDy bit is 1.
>
> It is **IMPLEMENTATION DEFINED** whether the counter counts the access if any of the following are true:
>
> - The refill is not allocated in the TLB.
> - The access generates a Translation fault for any other reason.

is changed to read:

> `0x0005`, L1D_TLB_REFILL, Level 1 data TLB refill
>
> The counter does not count the access if any of the following are true:
>
> - The access is due to a TLB maintenance instruction.
> - The access generates a Translation fault because the applicable TCR_ELx.EPDy bit is 1.
> - FEAT_E0PD is implemented and the access is an unprivileged access that generates a Translation fault because the applicable TCR_ELx.E0PDy bit is 1.
> - FEAT_SVE is implemented and the access is a non-fault access that fails because the applicable TCR_ELx.NFDy bit is 1.
>
> It is **IMPLEMENTATION DEFINED** whether the counter counts the access if any of the following are true:
>
> - The access generates a Translation fault for any other reason.
> - The access misses in the TLB and generates a translation table walk, but the result is not allocated into the TLB for any reason other than a Translation fault.

The equivalent changes are made in the following events: L1I_TLB_REFILL, L2D_TLB_REFILL, and L2I_TLB_REFILL.

2.3 C17536

In section **D14.3.2 "Common microarchitectural events"**, the description of DP_SPEC that reads:

> `0x0073`, DP_SPEC, Operation speculatively executed, integer data processing
>
> The counter counts each operation counted by INST_SPEC that is an integer data processing operation.
>
> Operations due to the following instructions are counted as integer data-processing operation:
>
> - In AArch64 state instructions from the following sections:
> - "Data processing - immediate"
> - "Data processing - register"
> - "System register instructions"
> - "System instructions" other than Memory-writing instructions
> - "Hint instructions"
> - When FEAT_SVE is implemented and the SVE_SPEC event is implemented, non-SIMD SVE instructions.
> - In AArch32 state instructions from the following sections:
> - "Data-processing instructions".

- "PSTATE and banked register access instructions".
- "Banked register access instructions".
- "Miscellaneous instructions other than ISB and prefetches".
- "System register access instructions other than LDC and STC instructions".

This includes MOV and MVN instructions.

It is **IMPLEMENTATION DEFINED** whether the preload instructions PRDM, PLD, PLDW, and PLI count as integer data-processing operations or load operations. Arm recommends that if the instructions are not implemented as a **NOP** then it is counted as a load operation.

When FEAT_PMUv3p9 is not implemented, it is **IMPLEMENTATION DEFINED** whether ISB is counted as an integer data-processing operation of a Software change of the PC.

When FEAT_PMUv3p9 is implemented, ISB is counted as an integer data-processing operation.

It is **IMPLEMENTATION DEFINED** whether the following instructions are counted as integer data-processing operations, SIMD operations, or floating-point operations, but Arm recommends that the instructions are counted as integer data-processing operations:

- In AArch64 state:
  - Instructions from Floating-point move (register) that transfers data between a general-purpose register and a SIMD&FP register without conversion: FMOV (general).
  - Instructions from "SIMD Move" that transfers data between a genera-purpose register and an element or elements in a SIMD&FP register: DUP (general), SMOV, UMOV, and INS (general). This includes the aliases MOV (from general) and MOV (to general).
  - When FEAT_SVE is implemented ant the SVE_SPEC event is not implemented, non-SIM SVE instructions.
- In AArch32 state:
  - VDUP (general-purpose register).
  - All VMOV instructions that transfer data between a general-purpose register and a SIMD&FP register.
  - VMRS and VMSR.

When FEAT_PMUv3p8 is not implemented, this is an **IMPLEMENTATION DEFINED** event.

is changed to read:

`0x0073`, DP_SPEC, Operation speculatively executed, integer data processing

The counter counts each operation counted by INST_SPEC and not counted as any of:

- A load or store operation, counted by LDST_SPEC.
- A Software change of the PC operation, counted by PC_WRITE_SPEC.
- A scalar floating-point data processing operation, counted by VFP_SPEC.

- An Advanced SIMD, SVE, or SME data processing operation, counted by SE_SPEC.
- A cryptographic operation, counted by CRYPTO_SPEC.

This includes operations due to the following instructions, which are counted as integer data-processing operations

- In AArch64 state instructions from the following sections:
  - “Data processing - immediate”
  - “Data processing - register”
  - “System register instructions”
  - “Instructions with register argument”
  - “System instructions” other than Memory-writing instructions
  - “Hint instructions”
  - When FEAT_SVE is implemented and the SVE_SPEC event is implemented, non-SIMD SVE instructions.
- In AArch32 state instructions from the following sections:
  - “Data-processing instructions”.
  - “PSTATE and banked register access instructions”.
  - “Banked register access instructions”.
  - “Miscellaneous instructions other than ISB and prefetches”.
  - “System register access instructions other than LDC and STC instructions”.

When FEAT_PMUv3p8 is not implemented, this is an **IMPLEMENTATION DEFINED** event.

2.4 D18847

In section **J1.1.3 “aarch64/functions”**, in the pseudocode function AArch64.MemSingleRead(), the code segment that reads:

```
(bits(size*8), AddressDescriptor, PhysMemRetStatus) AArch64.MemSingleRead(bits(64)
 address,
                                                                          integer
 size,

 AccessDescriptor accdesc_in,
                                                                          boolean
 aligned)
    assert size IN {1, 2, 4, 8, 16};
    bits(size*8) value = bits(size*8) UNKNOWN;
    PhysMemRetStatus memstatus = PhysMemRetStatus UNKNOWN;
    AccessDescriptor accdesc = accdesc_in;
    if IsFeatureImplemented(FEAT_LSE2) then
       assert AllInAlignedQuantity(address, size, 16);
    else
       assert IsAligned(address, size);
    if IsFeatureImplemented(FEAT_MTE2) && accdesc.tagchecked then
       accdesc.tagchecked = AArch64.AccessIsTagChecked(address, accdesc);
    AddressDescriptor memaddrdesc;
```

```
    memaddrdesc = AArch64.TranslateAddress(address, accdesc, aligned, size);
    if IsFault(memaddrdesc) then
       return (value, memaddrdesc, memstatus);
```

is changed to read:

```
(bits(size*8), AddressDescriptor, PhysMemRetStatus) AArch64.MemSingleRead(bits(64)
 address,
                                                                            integer
 size,

 AccessDescriptor accdesc_in,
                                                                            boolean
 aligned)
    assert size IN {1, 2, 4, 8, 16};
    bits(size*8) value = bits(size*8) UNKNOWN;
    PhysMemRetStatus memstatus = PhysMemRetStatus UNKNOWN;
    AccessDescriptor accdesc = accdesc_in;
    if IsFeatureImplemented(FEAT_LSE2) then
       assert AllInAlignedQuantity(address, size, 16);
    else
       assert IsAligned(address, size);
    if IsFeatureImplemented(FEAT_MTE2) && accdesc.tagchecked then
       accdesc.tagchecked = AArch64.AccessIsTagChecked(address, accdesc);
    AddressDescriptor memaddrdesc;
    memaddrdesc = AArch64.TranslateAddress(address, accdesc, aligned, size);
    if !IsFault(memstatus) && accdesc.acctype == AccessType_IFETCH then
        memaddrdesc.fault = AArch64.CheckDebug(memaddrdesc.fault.vaddress, accdesc,
 size);
    if IsFault(memaddrdesc) then
       return (value, memaddrdesc, memstatus);
```

2.5 R19582

In section **D21.20.4 "Translation table accesses by AT instructions"**, the text that reads:

> Accesses to translation tables by AT instructions are given the MPAM information specified for translation table accesses by a data load instruction that is issued from the Exception level that the AT instruction was executed from. The stage and Exception level specified in the AT instructions do not affect the MPAM information to use.

is changed to read:

> Accesses to translation tables by AT instructions are given the MPAM information specified for translation table accesses by a data load instruction that is issued from an **IMPLEMENTATION DEFINED** choice of:
>
> - The Exception level that the AT instruction was executed from.
> - The Exception level that configures the translation regime targeted by the AT instruction.

2.6 D20634

In section **D24.4.20 "TRCCNTCTLR<n>, Trace Counter Control Register <n>, n = 0 - 3"**, in the field description of 'CNTEVENT_SEL', the following text is added:

> If an unimplemented Resource Selector is selected using this field, the value returned on a read of this field is **UNKNOWN**.

In the same section, the equivalent text added to 'RLDEVENT_SEL' field, as well as in the following:

- Section **D24.4.26 "TRCEVENTCTL0R, Trace Event Control 0 Register"**, fields 'EVENT0_SEL', 'EVENT1_SEL', 'EVENT2_SEL', and 'EVENT3_SEL'.
- Section **D24.4.53 "TRCSEQEVR<n>, Trace Sequencer State Transition Control Register <n>, n = 0 - 2"**, fields 'F_SEL' and 'B_SEL'.
- Section **D24.4.54 "TRCSEQRSTEVR, Trace Sequencer Reset Control Register"**, 'RST_SEL' field.
- Section **D24.4.63 "TRCTSCTLR, Trace Timestamp Control Register"**, 'EVENT_SEL' field.
- Section **D24.4.64 "TRCVICTLR, Trace ViewInst Main Control Register"**, 'EVENT_SEL' field.

2.7 C20706

In section **A1.5.7.3 "NaN propagation"**, the following text is added:

> For NaN propagation rules for BFDOT and BFMMLA, see section BFloat16 behaviors for instructions that compute sum-of-products.
>
> For NaN propagation rules for FP8 instructions, see section Summary of FP8 instruction behaviors.

In the same section, the text that reads:

> - If all of the following apply, the output is a NaN derived from the <Zn>, <Vn>, <Hn>, <Sn>, or <Dn> registers:
>   - FPCR.AH == 1.
>   - The operation has two floating-point inputs.
>   - The operation has two NaN operands.

is changed to read:

> - If all of the following apply, the output is a NaN derived from the <Zn>, <Vn>, <Hn>, <Sn>, or <Dn> register, except for SVE FSUBR (vectors) and SVE FDIVR. For SVE FSUBR (vectors) and SVE FDIVR instructions, the output is a NaN derived from the <Zm> register.
>   - FPCR.AH == 1.
>   - The operation has two floating-point inputs.
>   - The operation has two NaN operands.

In the same section, the text that reads:

- If all of the following apply, the output is a NaN derived from the NaN held in the <Zn>, <Vn>, <Hn>, <Sn>, or <Dn> registers:
  - FPCR.AH == 1
  - The instruction is one of: BFMLALB, BFMLALT (by element), BFMLALB, BFMLALT (vector), FCMLA, FMADD, FMLA (by element), FMLA (vector), FMLAL, FMLAL2 (by element), FMLAL, FMLAL2 (vector), FMLS (by element), FMLS (vector), FMLSL, FMLSL2 (by element), FMLSL, FMLSL2 (vector), FMSUB, FNMADD, and FNMSUB.
  - One of the following applies:
    - The operation has three NaN operands.
    - The operation has two NaN operands and the <Zn>, <Vn>, <Hn>, <Sn> or <Dn> register holds a NaN.
- If all of the following apply, the output is a NaN derived from the NaN held in the <Zm>, <Vm>, <Hm>, <Sm>, or <Dm> registers:
  - FPCR.AH == 1
  - The instruction is one of: BFMLALB, BFMLALT (by element), BFMLALB, BFMLALT (vector), FCMLA, FMADD, FMLA (by element), FMLA (vector), FMLAL, FMLAL2 (by element), FMLAL, FMLAL2 (vector), FMLS (by element), FMLS (vector), FMLSL, FMLSL2 (by element), FMLSL, FMLSL2 (vector), FMSUB, FNMADD, and FNMSUB.
  - The operation has two NaN operands and the <Zn>, <Vn>, <Hn>, <Sn> or <Dn> register does not hold a NaN.

is changed to read:

- If all of the following apply, the output is a NaN derived from the NaN held in the <Zn>, <Vn>, <Hn>, <Sn>, or <Dn> registers:
  - FPCR.AH == 1
  - The instruction is one of those shown in Table 1.
  - One of the following applies:
    - The operation has three NaN operands.
    - The operation has two NaN operands and the <Zn>, <Vn>, <Hn>, <Sn> or <Dn> register holds a NaN.
- If all of the following apply, the output is a NaN derived from the NaN held in the <Zm>, <Vm>, <Hm>, <Sm>, or <Dm> registers:
  - FPCR.AH == 1
  - The instruction is one of those shown in Table 1.
  - The operation has two NaN operands and the <Zn>, <Vn>, <Hn>, <Sn> or <Dn> register does not hold a NaN.

Table 1

Advanced SIMD&FP	BFMLALB, BFMLALT, FCMLA, FMADD, FMLA, FMLAL, FMLAL2, FMLS, FMLSL, FMLSL2, FMSUB, FNMADD, and FNMSUB.
SVE	BFMLA, BFMLALB, BFMLALT, BFMLS, BFMLSLB, BFMLSLT, FCMLA, FMAD, FMLA, FMLALB, FMLALT, FMLS, FMLSLB, FMLSLT, FMSB, FNMAD, FNMLA, FNMLS, FNMSB.

The following text is added as the final bullet point to that list:

> - When the operation has four or more floating-point inputs:
>   - If at least one argument to FPDot() is a signaling NaN, a function outputs a quiet NaN derived from the first signaling NaN argument.
>   - If at least one argument to FPDot() is a NaN, but none of the arguments is a signaling NaN, a function outputs a quiet NaN derived from the first NaN argument.

2.8 D20966

In section **D16.6.6 "Sample operation records for misspeculated and non-architectural operations"**, the text that reads:

> The record must not contain information that cannot be accessed by privileged software of the owning Exception level.

is changed to read:

> The record must not contain or infer values that cannot be accessed or used by the sampled operation, as defined by Restrictions on the effects of speculation.
>
> For example:
> - An address that is not permitted to be used by the sampled operation, such as might be recorded in an Address packet.
> - A condition code or predicate value that is not permitted to be used by the sampled operation, such as might be inferred from the Mispredicted, Partial or empty predicate, or Empty predicate bits in the Events packet.
> - A branch target address predicted in a way that is prohibited, such as might be recorded in an Address packet.

In section **D13.12.1.1 "Definition of terms"**, the text that reads:

> Operation speculatively executed
>
> An Operation that is Speculatively executed.
>
> There is no architecturally guaranteed relationship between a Speculatively executed micro-op and an architecturally executed instruction.

> The results of such an operation can also be discarded, if it transpires that the operation was not required, such as following a mispredicted branch. Therefore, the architecture defines these events as operations speculatively executed, where appropriate.

is changed to read:

> Operation speculatively executed
>
> An Operation that is Speculatively executed.
>
> There is no architecturally guaranteed relationship between a Speculatively executed micro-op and an architecturally executed instruction.
>
> The results of such an operation can also be discarded, if it transpires that the operation was not required, such as following a mispredicted branch. Therefore, the architecture defines these events as operations speculatively executed, where appropriate.
>
> Many PMU events count operations that are speculatively executed. This means that the PMU can observe and record the behavior of operations executed under the effects of speculation. However, the PMU must not count an event that might be used to infer values that cannot be accessed or used by the operation, as defined by Restrictions on the effects of speculation.
>
> For example:
>
> - An address that is not permitted to be used by the operation might be inferred by whether it is present in a cache or TLB.
> - A condition code that is not permitted to be used by the operation can be inferred by whether a branch was taken or not taken.
> - A predicate value that is not permitted to be used by the operation can be inferrred by whether a predicate value is empty or not.

2.9 R21168

In section **C3.2.6 “Single-copy atomic 64-byte load/store”**, the text that reads:

> When the instructions access a memory type that is not one of the following, a data abort for unsupported Exclusive or atomic access is generated:
>
> - Normal Inner Non-cacheable, Outer Non-cacheable.
> - Device-GRE.
> - Device-nGRE.
> - Device-nGnRE.
> - Device-nGnRnE.
>
> It is **IMPLEMENTATION DEFINED** which of the following approaches is used to provide this check:

- The check is performed at each enabled stage of translation, and the fault is reported for the first stage of translation that provides an inappropriate memory type. In this case, the value of the HCR_EL2.DC bit does not cause accesses generated by the instructions to generate a stage 1 Data abort,
- The check is performed against the resulting memory type after all enabled stages of translation. In this case the fault is reported at the final enabled stage of translation.

is changed to read:

When the instructions access a memory type that is not one of the following, a data abort for unsupported Exclusive or atomic access is generated:

- Normal Inner Non-cacheable, Outer Non-cacheable.
- Device-GRE.
- Device-nGRE.
- Device-nGnRE.
- Device-nGnRnE.

It is **IMPLEMENTATION DEFINED** which of the following approaches is used to provide this check:

- The check is performed at each enabled stage of translation, and the fault is reported for the first stage of translation that provides an inappropriate memory type. In this case, the value of the HCR_EL2.DC bit does not cause accesses generated by the instructions to generate a stage 1 Data abort,
- The check is performed against the resulting memory type after all enabled stages of translation. In this case the fault is reported at the final enabled stage of translation.
- The check is performed against the resulting memory type after all enabled stages of translation. In this case the fault is reported for the first stage of translation that provided an inappropriate memory type. The value of the HCR_EL2.DC bit does not cause accesses generated by the instructions to generate a stage 1 Data abort.

2.10 D21186

In section **J1.1 "Pseudocode for AArch64 operation"**, in the pseudocode function AArch64.SettingDirtyStatePermitted(), the following code segment that reads:

```
boolean AArch64.SettingDirtyStatePermitted(FaultRecord fault)
    if fault.statuscode == Fault_None then
        return TRUE;
    elsif fault.statuscode == Fault_Alignment then
        return ConstrainUnpredictableBool(Unpredictable_DBUPDATE);
    else
        return FALSE;
```

is changed to read:

```
boolean AArch64.SettingDirtyStatePermitted(FaultRecord fault, FaultRecord
 fault_perm)
```

```
    if fault_perm.statuscode != Fault_None then
        return FALSE;
    elsif fault.statuscode == Fault_None then
        return TRUE;
    elsif fault.statuscode == Fault_Alignment then
        return ConstrainUnpredictableBool(Unpredictable_DBUPDATE);
    else
        return FALSE;
```

In the same section, in the pseudocode function AArch64.SettingDirtyStatePermitted() within AArch64.SetDirtyFlag(), the following code segment that reads:

```
boolean AArch64.SetDirtyFlag(bit hd, bit db_present, AccessDescriptor accdesc,
 FaultRecord fault)
   if hd == '0' || !AArch64.SettingDirtyStatePermitted(fault) then
        return FALSE;
   elsif accdesc.acctype IN {AccessType_AT, AccessType_IC, AccessType_DC} then
        return FALSE;
   elsif !accdesc.write then
        return FALSE;
   else
       return db_present == '1';
```

is changed to read:

```
boolean AArch64.SetDirtyFlag(bits(1) hd, bits(1) dbm, AccessDescriptor accdesc,
 FaultRecord fault,
                             FaultRecord fault_perm)
    if hd == '0' then
        return FALSE;
    elsif !AArch64.SettingDirtyStatePermitted(fault, fault_perm) then
        return FALSE;
    elsif accdesc.acctype IN {AccessType_AT, AccessType_IC, AccessType_DC} then
        return FALSE;
    elsif !accdesc.write then
        return FALSE;
    else
        return dbm == '1';
```

In the same section, in the pseudocode function AArch64.SetDirtyFlag() within AArch64.S2Translate(), the following code segment that reads:

```
if fault.statuscode == Fault_None then
    (fault, s2fs1mro) = AArch64.S2CheckPermissions(fault, walkstate, walkparams,
 ipa, accdesc);
    ...
...
if AArch64.SetDirtyFlag(walkparams.hd, (walkparams.s2pie OR descriptor<51>),accdesc,
 fault) then
    // Set descriptor S2AP[1]/Dirty bit permitting stage 2 writes
    new_desc<7> = '1';
...
```

is changed to read:

```
FaultRecord fault_perm;
(fault_perm, s2fs1mro) = AArch64.S2CheckPermissions(fault, walkstate, walkparams,
 ipa, accdesc)
    ...
...
```

```
if AArch64.SetDirtyFlag(walkparams.hd, (walkparams.s2pie OR descriptor<51>),
                        accdesc, fault, fault_perm) then
    // Set descriptor S2AP[1]/Dirty bit permitting stage 2 writes
    new_desc<7> = '1';
if fault.statuscode == Fault_None && fault_perm.statuscode != Fault_None then
    fault = fault_perm;
...
```

2.11 D21842

In section **C6.2.328 "SETGP, SETGM, SETGE"**, the text that reads:

```
if memset.stage != MOPSStage_Prologue then
    CheckMemSetParams(memset, options);

    if (memset.setsize != 0 && (memset.stagesetsize != 0 ||
 MemStageSetZeroSizeCheck()) &&
          !IsAligned(memset.toaddress, TAG_GRANULE)) then
        AArch64.Abort(memset.toaddress, AlignmentFault(accdesc));
    if ((memset.stagesetsize != 0 || MemStageSetZeroSizeCheck()) &&
          !IsAligned(memset.setsize<63:0>, TAG_GRANULE)) then
        AArch64.Abort(memset.toaddress, AlignmentFault(accdesc));
```

is changed to read:

```
if memset.stage != MOPSStage_Prologue then
    CheckMemSetParams(memset, options);

    bits(64) fault_address;
    if memset.implements_option_a then
        fault_address = memset.to_address+memset.setsize;
    else
        fault_address = memset.to_address;

    if (memset.setsize != 0 && (memset.stagesetsize != 0 ||
 MemStageSetZeroSizeCheck()) &&
          !IsAligned(memset.toaddress, TAG_GRANULE)) then
        AArch64.Abort(fault_address, AlignmentFault(accdesc));
    if ((memset.stagesetsize != 0 || MemStageSetZeroSizeCheck()) &&
          !IsAligned(memset.setsize<63:0>, TAG_GRANULE)) then
        AArch64.Abort(fault_address, AlignmentFault(accdesc));
```

The equivalent changes are made to the following sections:

- **C6.2.329 "SETGPN, SETGMN, SETGEN"**.
- **C6.2.330 "SETGPT, SETGMT, SETGET"**.
- **C6.2.331 "SETGPTN, SETGMTN, SETGETN"**.

2.12 D22232

In section **D14.3.1 "Common architectural events"**, in the description of the CHAIN event, the text that reads:

> `0x001E`, CHAIN, Chain a pair of event counters
>
> ...
>
> For an odd-numbered counter <n+1>, the counter increments when an event increments the preceding even-numbered counter <n> on the same PE causing unsigned overflow of bits [31:0] of the event counter <n>, and any of the following are true:
>
> - FEAT_PMUv3p5 is not implemented.
> - EL2 is not implemented and PMCR.LP is 1.
> - EL2 is implemented, <n> is less than HDCR.HPMN, and PMCR.LP is 0.
> - EL2 is implemented, <n> is greater than or equal to HDCR.HPMN, and HDCR.HLP is 0.

is changed to read:

> `0x001E`, CHAIN, Chain a pair of event counters
>
> ...
>
> For an odd-numbered counter <n+1>, the counter increments when an event increments the preceding even-numbered counter <n> on the same PE causing unsigned overflow of bits [31:0] of the event counter <n>, and any of the following are true:
>
> - FEAT_PMUv3p5 is not implemented.
> - EL2 is not implemented and PMCR.LP is 0.
> - EL2 is implemented, <n> is less than HDCR.HPMN, and PMCR.LP is 0.
> - EL2 is implemented, <n> is greater than or equal to HDCR.HPMN, and HDCR.HLP is 0.

2.13 C22244

In section **A2.2.7 "The Armv8.6 architecture extension"**, under the heading 'FEAT_ECV, Enhanced Counter Virtualization', the text that reads:

> FEAT_ECV enhances the Generic Timer architecture.
>
> When executing in AArch64 state or AArch32 state, FEAT_ECV provides:
>
> - Self-synchronizing views of the virtual and physical timers in AArch64 and AArch32 state.
> - The ability to scale the generation of the event stream.
>
> When EL2 is using AArch64 state, FEAT_ECV provides:

- An optional offset between the EL1 or EL0 view of physical time, and the EL2 or EL3 view of physical time.
- Traps configurable in CNTHCTL_EL2 that trap EL0 and EL1 access to the virtual counter or timer registers, and accesses to the physical timer registers when they are accessed using an EL02 descriptor.

The optional offset to views of physical time, and the configurable traps in CNTHCTL_EL2, both apply to EL1 and EL0 whether EL1 and EL0 are in AArch64 state or AArch32 state.

This feature is supported in both AArch64 and AArch32 states.

FEAT_ECV is OPTIONAL from Armv8.5.

FEAT_ECV is mandatory from Armv8.6.

The following field identifies the presence of FEAT_ECV:

- ID_AA64MMFR0_EL1.ECV.

is changed to read:

FEAT_ECV enhances the Generic Timer architecture. FEAT_ECV provides:

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

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

This feature is supported in both AArch64 and AArch32 states.

FEAT_ECV is OPTIONAL from Armv8.5.

FEAT_ECV is mandatory from Armv8.6.

The following field identifies the presence of FEAT_ECV:

- ID_AA64MMFR0_EL1.ECV.

In the same section, new feature FEAT_ECV_POFF is introduced, under the heading 'FEAT_ECV_POFF, Enhanced Counter Virtualization Physical Offset':

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

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

> This feature is supported in both AArch64 and AArch32 states.
>
> FEAT_ECV_POFF is OPTIONAL from Armv8.5.
>
> If FEAT_ECV_POFF is implemented, then FEAT_ECV is implemented.
>
> If FEAT_ECV_POFF is implemented, then FEAT_AA64EL2 is implemented.
>
> If FEAT_RME is implemented, then FEAT_ECV_POFF is implemented.
>
> The following field identifies the presence of FEAT_ECV_POFF:
> - ID_AA64MMFR0_EL1.ECV.

In section **D12.1.1 "The full set of Generic Timer components"**, the text that reads:

> - A physical counter, which gives access to the count value of the system counter. When FEAT_ECV is implemented, the CNTPOFF_EL2 register allows offsetting of physical timers and counters.

is changed to read:

> - A physical counter, which gives access to the count value of the system counter. When FEAT_ECV_POFF is implemented, the CNTPOFF_EL2 register allows offsetting of physical timers and counters.

In section **D12.2.1 "The physical counter"**, the text that reads:

> When FEAT_ECV is implemented, the CNTPOFF_EL2 register holds the optional physical offset that can be applied at EL0 and EL1 whether EL0 and EL1 are using AArch64 state or AArch32 state.

is changed to read:

> When FEAT_ECV_POFF is implemented, the CNTPOFF_EL2 register holds the optional physical offset that can be applied at EL0 and EL1 whether EL0 and EL1 are using AArch64 state or AArch32 state.

In section **D23.2.74 "ID_AA64MMFR0_EL1, AArch64 Memory Model Feature Register 0"**, in the field description of 'ECV, bits [63:60]' the text that reads:

> FEAT_ECV implements the functionality identified by the values `0b0001` and `0b0010`.

is changed to read:

> FEAT_ECV implements the functionality identified by the value `0b0001`.
>
> FEAT_ECV_POFF implements the functionality identified by the value `0b0010`.

In the same field description, the text that reads:

`0b0010`	As `0b0001`, and also includes support for CNTHCTL_EL2.ECV and CNTPOFF_EL2.

is changed to read:

`0b0010`	As `0b0001`, and the CNTPOFF_EL2 register and the CNTHCTL_EL2.ECV and SCR_EL3.ECVEn fields are implemented.

In section **D23.10.22 "CNTPOFF_EL2, Counter-timer Physical Offset Register"**, the text that reads:

> This register is present only when FEAT_ECV is implemented. Otherwise, direct accesses to CNTPOFF_EL2 are **UNDEFINED**.

is changed to read:

> This register is present only when FEAT_ECV_POFF is implemented. Otherwise, direct accesses to CNTPOFF_EL2 are **UNDEFINED**.

In section **D23.10.2 "CNTHCTL_EL2, Counter-timer Hypervisor Control Register"**, in the field description of 'ECV, bit [12]', the text that reads:

> When SCR_EL3.{NS, EEL2} is {0, 0}, the Effective value of this field is 0.

is changed to read:

> When SCR_EL3.{NS, EEL2} is {0, 0} or if FEAT_ECV_POFF is not implemented, the Effective value of this field is 0.

In section **D23.10.2 "CNTHCTL_EL2, Counter-timer Hypervisor Control Register"**, in the field description of 'ECV, bit [12]', and in section **D23.2.155 "SCR_EL3, Secure Configuration Register"**, in the field description of 'ECVEn, bit [28]', the text that reads:

> When FEAT_ECV is implemented:

is changed to read:

> When FEAT_ECV_POFF is implemented:

2.14 D22346

In section **D14.3.2 "Common microarchitectural events"**, the event description of FP_FMA_SPEC that reads:

> `0x8028`, FP_FMA_SPEC, Floating-point operation speculatively executed, FMA
>
> The counter counts each Speculatively executed floating-point fused multiply-add or multiply-subtract operation counted by FP_SPEC due to any of the following A64 instructions:
>
> - Scalar: FMADD, FMSUB, FNMADD, or FNMSUB.
> - Advanced SIMD: BFMLALB, BFMLALT, FCMLA, FMLA, or FMLS.

- SVE: BFMLALB (vectors), BFMLALT (vectors), FCMLA (vectors), FMAD, FMLA (vectors), FMLS (vectors), FMSB, FNMAD, FNMLA, FNMLS, FNMSB, or FTMAD.
- SVE2: BFMLALB (vectors), BFMLALT (vectors), FMLALB (vectors), FMLALT (vectors), FMLSLB (vectors), or FMLSLT (vectors).

It is **IMPLEMENTATION DEFINED** which floating-point fused multiply-add or multiply-subtract operations are counted in AArch32 state.

is changed to read:

`0x8028`, FP_FMA_SPEC, Floating-point operation speculatively executed, FMA

The counter counts each Speculatively executed floating-point fused multiply-add or multiply-subtract operation counted by FP_SPEC due to any of the following A64 instructions:

- Scalar: FMADD, FMSUB, FNMADD, or FNMSUB.
- Advanced SIMD: BFMLALB, BFMLALT, FCMLA, FMLA, FMLAL, FMLAL2, FMLS, FMLSL, or FMLSL2.
- SVE: BFMLALB, BFMLALT, FCMLA, FMAD, FMLA, FMLS, FMSB, FNMAD, FNMLA, FNMLS, FNMSB, or FTMAD.
- SVE2: FMLALB, FMLALT, FMLSLB, or FMLSLT.

It is **IMPLEMENTATION DEFINED** which floating-point fused multiply-add or multiply-subtract operations are counted in AArch32 state.

Similar changes are made to the following event descriptions:

- "`0x8029`, ASE_FP_FMA_SPEC, Floating-point operation speculatively executed, Advanced SIMD FMA".
- "`0x802B`, ASE_SVE_FP_FMA_SPEC, Floating-point operation speculatively executed, Advanced SIMD or SVE FMA".

2.15 C22401

In section **D8.3.1 "VMSAv8-64 descriptor formats"**, in R_{JJNHR}, the following rows in the table:

Bit position	Field	Condition
[63]	IGNORED	Non secure or Secure state, or Realm EL1&0 translation regime.
Alternate MECID (AMEC)	Realm EL2 or Realm EL2&0 translation regimes, and the descriptor NS field is 0. See Memory Encryption Contexts extension.	
RES0	Realm EL2 or Realm EL2&0 translation regimes, and the descriptor NS field is 1.	

are changedto read:

Bit position	Field	Condition
[63]	IGNORED	Secure state
Reserved for software use	Non-secure state, or the Realm EL1&0 translation regime.	
RES0	Realm EL2 or Realm EL2&0 translation regimes, and the descriptor NS field is 1.	

In the same rule, the rows that read:

Bit position	Field	Condition
[58:56]	Reserved for software use	-
[55]	IGNORED	-

are changed to read:

Bit position	Field	Condition
[58:55]	Reserved for software use	-

2.16 C22430

In section **D.1.3.2.1 "Synchronous exception entry"**, the following rule that reads:

> R_{RBFJJV}
>
> For GPC exceptions, a PA that characterizes the exception is captured in MFAR_EL3.

is changed to read:

> For GPC exceptions, a PA and physical address space that characterizes the exception are captured in MFAR_EL3.

In section **D.1.3.2.1 "Synchronous exception entry"**, the following rule that reads:

> R_{RTGQRC}
>
> For Instruction Abort or Data Abort exceptions caused by an External abort, when FEAT_PFAR is implemented:
>
> - If all of the following apply, then ESR_ELx.PFV is set to 0:
> - The exception is taken to EL1.
> - EL2 is implemented and enabled in the current Security state.
> - The Effective value of HCR_EL2.VM is 1
> - Otherwise, ESR_ELx.PFV is set to an **IMPLEMENTATION DEFINED** value of 0 or 1.
>
> For all other Instruction Aborts and Data Aborts, ESR_ELx.PFV is set to 0.
>
> On taking a synchronous External Abort, if ESR_ELx.PFV is set to 1 by the PE then:

- An address within the same naturally-aligned fault granule as the faulting physical address is written to PFAR_ELx.PA or MFAR_EL3.PA as applicable. The fault granule size is defined by DLVGRB.
- The faulting physical address space is written to PFAR_ELx.{NSE,NS} or MFAR_EL3.{NSE,NS} as applicable.

Note: PFAR_ELx never records the Intermediate Physical Address (IPA). PFAR_ELx might reveal a faulting physical addresses to a guest operating system if stage 2 translation is not being used and some other method is used to hide physical addresses from the guest (such as shadow page tables).

is changed to read:

R_{RTGQRC}

For Instruction Abort or Data Abort exceptions caused by an External abort, when FEAT_PFAR is implemented:

- If all of the following apply, then ESR_ELx.PFV is set to 0:
  - The exception is taken to EL1.
  - EL2 is implemented and enabled in the current Security state.
  - The Effective value of HCR_EL2.VM is 1.
- Otherwise, ESR_ELx.PFV is set to an **IMPLEMENTATION DEFINED** value of 0 or 1.

For all other Instruction Abort and Data Abort exceptions, ESR_ELx.PFV is set to 0.

On taking an Instruction Abort or Data Abort exception:

- If ESR_ELx.PFV is set to 1 by the PE, then a PA and physical address space that characterizes the exception are captured in PFAR_ELx or MFAR_EL3 as applicable.
- If ESR_ELx.PFV is set to 0 by the PE, then the applicable PFAR_ELx or MFAR_EL3 register is set to an **UNKNOWN** value.

In section **D.1.3.2.1 “Synchronous exception entry”**,the following rule that reads:

R_{RZTNQN}

On taking a synchronous External Abort, the following registers are **UNKNOWN** based on the ESR_ELx.PFV value:

- If ESR_EL1.PFV is set to 0, PFAR_EL1 is **UNKNOWN**.
- If ESR_EL2.PFV is set to 0, PFAR_EL2 is **UNKNOWN**.
- If ESR_EL3.PFV is set to 0, MFAR_EL3 is **UNKNOWN**.

is changed to read:

R_{RX0001}

> For all exceptions other than Instruction Abort, Data Abort, and GPC exceptions, the applicable PFAR_ELx or MFAR_EL3 register is set to an **UNKNOWN** value.

2.17 D22595

In section **D14.3.2 "Common microarchitectural events"**, the text for the PMU event INT_MUL64_SPEC that reads:

> `0x804c`, INT_MUL64_SPEC, Integer operation speculatively executed, 64x64 multiply
>
> The counter counts each Speculatively executed 64x64 integer multiply operation counted by INT_SPEC due to any of the following A64 instructions:
>
> - Scalar: MADD, MSUB, MUL, SMULH, or UMULH.
> - SVE: MAD, MLA, MLS, MSB, MUL, SMULH, or UMULH.
> - SVE2: SVE2: CMLA (vectors), MLA, MLS, MUL, SMLALB, SMLALT, SMLSLB, SMLSLT, SMULH, SMULLB, SMULLT, SQDMLALB, SQDMLALBT, SQDMLALT, SQDMLSLB, SQDMLSLBT, SQDMLSLT, SQDMULH, SQDMULLB, SQDMULLT, SQRDCMLAH (vectors), SQRDMLAH, SQRDMLSH, SQRDMULH, UMLALB, UMLALT, UMLSLB, UMLSLT, UMULH, UMULLB, or UMULLT.

is changed to read:

> `0x804c`, INT_MUL64_SPEC, Integer operation speculatively executed, 64x64 multiply
>
> The counter counts each Speculatively executed 64x64 integer multiply operation counted by INT_SPEC due to any of the following A64 instructions:
>
> - Scalar: MADD, MSUB, MUL, SMULH, or UMULH.
> - SVE: MAD, MLA, MLS, MSB, MUL, SMULH, or UMULH.
> - SVE2: CMLA (vectors), MLA, MLS, MUL, SMULH, SQDMULH, SQRDCMLAH (vectors), SQRDMLAH, SQRDMLSH, SQRDMULH, or UMULH.

The equivalent changes are made in event SVE_INT_MUL64_SPEC.

2.18 D22647

In section **B2.13.2.1.2 "Load-Exclusive/ Store-Exclusive and Atomic instructions"**, the text that reads:

> If FEAT_THE is implemented, Read-Check-Write instructions, RCW, and Read-Check-Write Software instructions, RCWS, generate an Alignment fault if the address being accessed is not aligned to the data transfer size regardless of the value of SCTLR_ELx.A.

is changed to read:

If FEAT_THE is implemented, Read-Check-Write instructions, RCW, and Read-Check-Write Software instructions, RCWS, generate an Alignment fault if the address being accessed is not aligned to the overall access size regardless of the value of SCTLR_ELx.A.

2.19 E22689

In **D24.2.74 "ID_AA64DFR0_EL1, AArch64 Debug Feature Register 0"**, in the field description of 'BRPs, bits [15:12]', the text that reads:

Number of breakpoints, minus 1. The value of this field is an **IMPLEMENTATION DEFINED** choice of:

`0b0001...0b1111`	The number of breakpoints, minus 1.

If FEAT_Debugv8p9 is implemented and 16 or more breakpoints are implemented, then this field reads as `0b1111` and ID_AA64DFR1_EL1.BRPs indicates the number of breakpoints.

is changed to read:

Index of the highest numbered context-aware breakpoint. Identifies which of the implemented breakpoints are context-aware breakpoints.

The value of this field is an **IMPLEMENTATION DEFINED** choice of:

`0b0001...0b1110`	Index of the highest numbered context-aware breakpoint.
`0b1111`	If ID_AA64DFR0_EL1.CTX_CMPs and ID_AA64DFR1_EL1.CTX_CMPs indicate that 16 or fewer breakpoints are context-aware, then the index of the highest context-aware breakpoint is 15. Otherwise, the context-aware breakpoints are the lowest numbered breakpoints.

If ID_AA64DFR1_EL1.BRPs is zero, then this is also the number of implemented breakpoints, minus 1, meaning the context-aware breakpoints are the highest numbered breakpoints.

The value of this field is always less than the number of implemented breakpoints.

In the same section, in the field description of 'CTX_CMPs, bits [31:28]', the text that reads:

Number of context-aware breakpoints, minus 1.

The value of this field is an **IMPLEMENTATION DEFINED** choice of:

`0b0000...0b1111`	The number of context-aware breakpoints, minus 1.

The value of this field is never greater than ID_AA64DFR0_EL1.BRPs. If FEAT_Debugv8p9 is implemented and 16 or more context-aware breakpoints are implemented, then this field reads as `0b1111` and ID_AA64DFR1_EL1.CTX_CMPs indicates the number of context-aware breakpoints.

is changed to read:

> Number of context-aware breakpoints, minus 1.
>
> The value of this field is an **IMPLEMENTATION DEFINED** choice of:

`0b0000...0b1110`	The number of context-aware breakpoints, minus 1.
`0b1111`	If ID_AA64DFR1_EL1.CTX_CMPs is zero, then 16 context-aware breakpoints are implemented. Otherwise, 16 or more context-aware breakpoints are implemented and ID_AA64DFR1_EL1.CTX_CMPs indicates how many.

> Values greater than ID_AA64DFR0_EL1.BRPs are not permitted.

In **D24.2.75 "ID_AA64DFR1_EL1, AArch64 Debug Feature Register 1"**, in the field description of 'BRPs, bits [15:8]', the following text is added:

> If more than 16 breakpoints are implemented, or the index of the highest numbered context-aware breakpoint is not the number of breakpoints minus 1, then the value `0x00` is not permitted.
>
> Nonzero values less than ID_AA64DFR0_EL1.BRPs are not permitted.

In the same section, in the field description of 'CTX_CMPs, bits [31:24]', the text that reads:

> The value of this field is never greater than ID_AA64DFR1_EL1.BRPs.

is changed to read:

> If 16 or fewer context-aware breakpoints are implemented, then the value of this field is either `0x00` or the same as ID_AA64DFR0_EL1.CTX_CMPs. If more than 16 context-aware breakpoints are implemented, then the value `0x00` is not permitted.
>
> Values greater than ID_AA64DFR1_EL1.BRPs are not permitted.

2.20 D22719

In **C5.5.92 "TLBIP RIPAS2E1, TLBIP RIPAS2E1NXS, TLB Range Invalidate by Intermediate Physical Address, Stage 2, EL1"**, under the heading 'TTL, bits [38:37]', the following text is removed:

> If FEAT_LPA2 is not implemented, when using a 16KB translation granule, this value is reserved and hardware should treat this field as `0b00`.

The equivalent change is applied to all other TLBIP instructions that apply to a range of addresses.

In **C5.5.86 "TLBIP IPAS2E1, TLBIP IPAS2E1NXS, TLB Invalidate Pair by Intermediate Physical Address, Stage 2, EL1"**, under heading 'TTL, bits [47:44]', the text that reads:

> - 0b01xx The entry comes from a 4KB translation granule. The level of walk for the leaf level 0bxx is encoded as:

    - `0b00` : If FEAT_LPA2 is implemented, level 0. Otherwise, treat as if TTL[3:2] is `0b00`.
    - ...
- 0b10xx The entry comes from a 16KB translation granule. The level of walk for the leaf level 0bxx is encoded as:
    - `0b00` : Reserved. Treat as if TTL[3:2] is `0b00`.
    - `0b01` : If FEAT_LPA2 is implemented, level 1. Otherwise, treat as if TTL[3:2] is `0b00`.
    - ...

is changed to read:

- 0b01xx The entry comes from a 4KB translation granule. The level of walk for the leaf level 0bxx is encoded as:
    - `0b00` : Level 0.
    - ...
- 0b10xx The entry comes from a 16KB translation granule. The level of walk for the leaf level 0bxx is encoded as:
    - `0b00` : Reserved. Treat as if TTL[3:2] is `0b00`.
    - `0b01` : Level 1.
    - ...

The equivalent change is applied to all other TLBIP instructions that do not apply to a range of addresses.

In **C5.5.59 "TLBI VAE1, TLBI VAE1NXS, TLB Invalidate by VA, EL1"**, under the heading 'Purpose', the text that reads:

- The entry is one of the following:
    - A 64-bit stage 1 translation table entry.
    - If FEAT_D128 is implemented, a 128-bit stage 1 translation table entry, if TTL[3:2] is `0b00`.
- The entry would be used to translate the specified VA, and one of the following applies:
    - The entry is from a level of lookup above the final level and matches the specified ASID.
    - The entry is a global entry from the final level of lookup.
    - The entry is a non-global entry from the final level of lookup that matches the specified ASID.

is changed to read:

- The entry would be used to translate the specified VA, and one of the following applies:
    - The entry is a Table entry and all the following apply:
        - The entry matches the specified ASID.
        - If FEAT_TTL is implemented, then one of the following applies:

        - TTL[3:2] is `0b00`.
        - The entry is in VMSAv8-32 LPAE or VMSAv8-64 format and leads to a Page or Block entry translating the specified VA that matches the granule and level specified in TTL.
    - The entry is a Page or Block entry and all of the following apply:
        - For a non-global entry, the entry matches the specified ASID.
        - If FEAT_TTL is implemented, then one of the following applies:
            - TTL[3:2] is `0b00`.
            - The entry is in VMSAv8-32 LPAE or VMSAv8-64 format and matches the granule and level specified in TTL.

The equivalent changes are applied to other TLBI instructions that have a TTL hint.

In **C5.5.128 "TLBIP VAE1, TLBIP VAE1NXS, TLB Invalidate Pair by VA, EL1"**, under the heading 'Purpose', the text that reads:

- The entry is one of the following:
    - A 128-bit stage 1 translation table entry.
    - A 64-bit stage 1 translation table entry, if TTL[3:2] is `0b00`.
- The entry would be used to translate the specified VA, and one of the following applies:
    - The entry is from a level of lookup above the final level and matches the specified ASID.
    - The entry is a global entry from the final level of lookup.
    - The entry is a non-global entry from the final level of lookup that matches the specified ASID.

is changed to read:

- The entry would be used to translate the specified VA, and one of the following applies:
    - The entry is a Table entry and all the following apply:
        - The entry matches the specified ASID.
        - If FEAT_TTL is implemented, then one of the following applies:
            - TTL[3:2] is `0b00`.
            - The entry is in VMSAv9-128 format and leads to a Page or Block entry translating the specified VA that matches the granule and level specified in TTL.
    - The entry is a Page or Block entry and all of the following apply:
        - For a non-global entry, the entry matches the specified ASID.
        - If FEAT_TTL is implemented, then one of the following applies:
            - TTL[3:2] is `0b00`.
            - The entry is in VMSAv9-128 format and matches the granule and level specified in TTL.

The equivalent changes are applied to other TLBIP instructions that have a TTL hint.

In **D8.17.5.3 "Translation table level hint"**, the following new information statement is added:

> For non-final levels of walk, a translation using VMSAv9-128 descriptors resolves a different number of IA bits from a translation using VMSAv8-64 descriptors. Because of this mismatch, the TTL hint applies differently between TLBI and TLBIP instructions:
>
> - For TLBI instructions, the TTL hint applies for 64-bit descriptors, including VMSAv8-64 and VMSAv8-32LPAE descriptors.
> - For TLBIP instructions, the TTL hint applies for 128-bit descriptors.

2.21 D22834

In section **A2.2.4 "The Armv8.3 architecture extension"**, the text that reads:

> When FEAT_PAuth is implemented, one of the following must be true:
>
> - Exactly one of the PAC algorithms is implemented.
> - If the PACGA instruction and other Pointer authentication instructions use different PAC algorithms, exactly two PAC algorithms are implemented.

is changed to read:

> When FEAT_PAuth is implemented, all of the following must be true:
>
> - Exactly one of the PAC algorithms is implemented.
> - The PACGA instruction and other Pointer authentication instructions use the same algorithm.

In section **D8.10.3 "PAC instructions"**, the following text is added:

> Arm strongly recommends that software at EL0 does not mix pointer authentication mechanisms. For example, using PACGA to create a PAC and using other pointer authentication instructions to authenticate, or using an instruction which uses a specific key to sign a pointer and using an instruction which uses a different key to authenticate that pointer.

2.22 C22897

In section **B2.9.4 "Restrictions on the effects of speculation from Armv8.5"**, the text that reads:

> FEAT_CSV3 introduces these restrictions:
>
> - Data loaded under speculation with a Permission or Domain fault cannot be used to form an address, generate condition codes, or generate SVE predicate values to be used by other instructions in the speculative sequence.

- Any read under speculation from a register that is not architecturally accessible from the current Exception level cannot be used to form an address, to generate condition codes, or to generate SVE predicate values to be used by other instructions in the speculative sequence.

is changed to read:

FEAT_CSV3 introduces these restrictions:

- An attempt to access data under speculation, where the data is inaccessible given the current state of the PE cannot be used to alter microarchitectural state in a manner that allows the value of the inaccessible data to be recovered by code architecturally executed through means such as measuring the timing of code. This restriction applies to any of the following cases which make the data inaccessible:
  - reading from a Location with a Permission or Domain fault.
  - directly reading from a system register that is not architecturally readable.
  - reading from a SIMD&FP registers, SVE register, or SME register that is not accessible due to traps configured by a more privileged Exception level.
  - deriving the value produced from a value indirectly read from a system register that is not required to be indirectly read per the architectural behavior of the instruction.

Note:

The current state of the PE is defined as the EL, values of all special-purpose registers, and values of all system registers under which the hardware attempts to access the data, which may be on the speculative execution path. If on the speculative execution path, the hardware must respect any changes to the PE state that occur on the speculative path, respecting the same rules that would be required if executing on the non-speculative path.

Note:

The following is a list of examples in which the value of the inaccessible data could be used by the microarchitecture that could lead to recovery of the value of the inaccessible data:

- to form an address, generate condition codes, or generate SVE predicate values to be used by an instruction I2 which is newer than I1 in the speculative sequence.
- to form the register value used for the comparison of a conditional branch instruction I2 which is newer than I1 in the speculative sequence. Note that this is to cover conditional branches which do not use the condition codes to determine whether or not to branch.
- by prediction mechanisms such as Cache Prefetch predictions.
- as an input to an instruction I2 which is newer than I1 in the speculative sequence where the execution timing of the I2 is dependent on the data value.

Note:

It is permissible for a value of zero to be produced by the load (or register read) and consumed by an instruction newer than the load in the speculative sequence even if the value of zero results in different execution timing compared to non-zero values.

In section **E2.3.4.4 "Further restrictions on the effects of speculation from Armv8.5"**, the text that reads:

> FEAT_CSV3 introduces these restrictions:
>
> - Data loaded under speculation with a Permission or Domain fault cannot be used to form an address or generate condition codes to be used by other instructions in the speculative sequence.
> - Any read under speculation from a register that is not architecturally accessible from the current Exception level cannot be used to form an address or to generate condition codes to be used by other instructions in the speculative sequence.

is changed to read:

> FEAT_CSV3 introduces these restrictions:
>
> - An attempt to access data under speculation, where the data is inaccessible given the current state of the PE cannot be used to alter microarchitectural state in a manner that allows the value of the inaccessible data to be recovered by code architecturally executed through means such as measuring the timing of code. This restriction applies to any of the following cases which make the data inaccessible:
> - reading from a Location with a Permission or Domain fault.
> - directly reading from a system register that is not architecturally readable.
> - reading from a SIMD&FP registers, SVE register, or SME register that is not accessible due to traps configured by a more privileged Exception level.
> - deriving the value produced from a value indirectly read from a system register that is not required to be indirectly read per the architectural behavior of the instruction.
>
> Note:
>
> The current state of the PE is defined as the EL, values of all special-purpose registers, and values of all system registers under which the hardware attempts to access the data, which may be on the speculative execution path. If on the speculative execution path, the hardware must respect any changes to the PE state that occur on the speculative path, respecting the same rules that would be required if executing on the non-speculative path.
>
> Note:
>
> The following is a list of examples in which the value of the inaccessible data could be used by the microarchitecture that could lead to recovery of the value of the inaccessible data:
>
> - to form an address, generate condition codes, or generate SVE predicate values to be used by an instruction I2 which is newer than I1 in the speculative sequence.
> - to form the register value used for the comparison of a conditional branch instruction I2 which is newer than I1 in the speculative sequence. Note that this is to cover conditional branches which do not use the condition codes to determine whether or not to branch.
> - by prediction mechanisms such as Cache Prefetch predictions.

- as an input to an instruction I2 which is newer than I1 in the speculative sequence where the execution timing of the I2 is dependent on the data value.

Note:

It is permissible for a value of zero to be produced by the load (or register read) and consumed by an instruction newer than the load in the speculative sequence even if the value of zero results in different execution timing compared to non-zero values.

In section **A2.2.6 "The Armv8.5 architecture extension"**, in the field description of FEAT_CSV3, the text that reads:

FEAT_CSV3 adds a mechanism to identify if hardware cannot disclose information about whether data loaded under speculation with a permission or domain fault can be used to form an address, generate condition codes, or generate SVE predicate values, to be used by instructions newer than the load in the speculative sequence.

is changed to read:

FEAT_CSV3 adds a mechanism to identify whether data loaded or read from a register under speculation where the data load or register read would not be permitted architecturally, can be used by instructions newer than the load or register read in a speculatively exploitable manner.

In section **D23.2.79 "ID_AA64PFR0_EL1, AArch64 Processor Feature Register 0"**, field 'CSV3, bits [63:60]', the text that reads:

The value of this field is an **IMPLEMENTATION DEFINED** choice of:

CSV3	Meaning
0b0000	This PE does not disclose whether data loaded under speculation with a permission or domain fault can be used to form an address or generate condition codes or SVE predicate values to be used by other instructions in the speculative sequence.
0b0001	Data loaded under speculation with a permission or domain fault cannot be used to form an address, generate condition codes, or generate SVE predicate values to be used by other instructions in the speculative sequence. The execution timing of any other instructions in the speculative sequence is not a function of the data loaded under speculation.

is updated to read:

CSV3	Meaning
0b0000	This PE does not disclose whether data loaded or read from a register under speculation where the data load or register read would not be permitted architecturally, can be used by instructions newer than the load or register read in a speculatively exploitable manner.
0b0001	Data loaded or read from a register under speculation where the data load or register read would not be permitted architecturally, cannot be used by instructions newer than the load or register read in a speculatively exploitable manner.

An equivalent change is made in section **G8.2.101 "ID_PFR2, Processor Feature Register 2"**, field 'CSV3, bits [3:0]'.

2.23 R22907

In section **D24.2.77 "AArch64 Floating-point Feature Register 0"**, the text that reads:

> Bits [27:2]
>
> Reserved, **RES0**.

is changed to read:

> Bits [27:8]
>
> Reserved, **RES0**.
>
> Bits [7:2]
>
> Reserved for data formats 2 to 7, **RES0**.

In section **C5.2.9 "Floating-point Mode Register"**, in the field description of 'F8S1, bits [2:0]', the text that reads:

> All other values are reserved.
>
> Reserved values identify an unsupported format, treated as a signaling NaN input by FP8 instructions.

is changed to read:

> All other values are reserved.
>
> Reserved values identify an unsupported format, and FP8 instructions treat the corresponding input as a **CONSTRAINED UNPREDICTABLE** choice of one of the following:
>
> - A signaling NaN.
> - Any of the supported FP8 formats.
>
> It is software's responsibility to check that a format value is supported in ID_AA64FPFR0_EL1[7:0], before writing it to this field.

In the same section, the equivalent change is made in the field description of 'F8S2, bits [5:3].

In the same section, in the field description of 'F8D, bits [8:6]', the text that reads:

> All other values are reserved.
>
> Reserved values identify an unsupported format, causing convert instructions that generate an FP8 result to set the result to `0xFF` and signal an Invalid Operation floating-point exception.

is changed to read:

All other values are reserved.

Reserved values identify an unsupported format, causing convert instructions that generate an FP8 result to perform a **CONSTRAINED UNPREDICTABLE** choice of one of the following behaviors:

- Setting the result to `0xFF` and signaling an Invalid Operation floating-point exception.
- Generating the expected result of any of the supported FP8 formats.

It is software's responsibility to check that a format value is supported in ID_AA64FPFR0_EL1[7:0], before writing it to this field.

2.24 C22913

In section **A2.3.6 "The Armv9.5 architecture extension"**, under the heading 'FEAT_SPE_SME, Statistical Profiling extensions for SME', the text that reads:

FEAT_SPE_SME is OPTIONAL from Armv9.4.

is changed to read:

FEAT_SPE_SME is OPTIONAL from Armv9.2.

2.25 D22999

In section **D6.5.4 "Faults"**, the rule that reads:

R_{FSPBK}

If a write by the Trace Buffer Unit generates an Alignment fault or MMU fault, including GPC faults, and TRBSR_EL1.S is 0, then all of the following occur:

- A trace buffer management event is generated. This sets TRBSR_EL1.IRQ to 1.
- TRBSR_EL1.S is set to 1, collection is stopped.
- TRBSR_EL1.EC is set to the appropriate one of the following values:
  - `0x24`, stage 1 Data Abort on write to trace buffer.
  - `0x25`, stage 2 Data Abort on write to trace buffer.
- TRBSR_EL1.FSC is set to indicate the type of the fault.
- TRBPTR_EL1 is set to the address that generated the fault.
- The other fields in TRBSR_EL1 are unchanged.

If a write by the Trace Buffer Unit generates an External abort on a translation table walk or translation table update, it is **IMPLEMENTATION DEFINED** whether TRBSR_EL1.EA is set to 1 or unchanged.

is changed to read:

> R_{FSPBK}
>
> If a write by the Trace Buffer Unit generates an Alignment fault, MMU fault, or GPC fault, and TRBSR_EL1.S is 0, then all of the following occur:
>
> - A trace buffer management event is generated. This sets TRBSR_EL1.IRQ to 1.
> - TRBSR_EL1.S is set to 1, collection is stopped.
> - TRBSR_EL1.EC is set to the appropriate one of the following values:
>   - `0x1E`, Granule Protection Check fault on write to trace buffer, other than a Granule Protection Fault (GPF).
>   - `0x24`, stage 1 Data Abort on write to trace buffer.
>   - `0x25`, stage 2 Data Abort on write to trace buffer. A GPF on translation table walk or update is reported as either a Stage 1 or Stage 2 Data Abort, as appropriate. Other GPFs are reported as a Stage 1 Data Abort.
> - TRBSR_EL1.FSC is set to indicate the type of the fault.
> - TRBPTR_EL1 is set to the address that generated the fault.
> - The other fields in TRBSR_EL1 are unchanged.
>
> If a write by the Trace Buffer Unit generates an External abort on a translation table walk or translation table update, it is **IMPLEMENTATION DEFINED** whether TRBSR_EL1.EA is set to 1 or unchanged.

The equivalent change is made in **D17.8.3 "Faults and watchpoints"**.

2.26 D23002

In section **D13.12 "PMU events and event numbers"**, the following instructions are added to the event definitions:

Event	Instructions added
ASE_FP_CVT_SPEC	BF1CVTL, BF1CVTL2, BF2CVTL, BF2CVTL2, F1CVTL, F1CVTL2, F2CVTL, F2CVTL2, FCVTN, FCVTN2, FCVTNS
ASE_FP_DOT_SPEC	FDOT (2-way, by element), FDOT (2-way, vector), FDOT (4-way, by element), FDOT (4-way, vector)
ASE_FP_FMA_SPEC	FMLALB, FMLALLBB, FMLALLBT, FMLALLTT, FMLALT
ASE_FP_PREDUCE_SPEC	FMAXNMP, FMAXP, FMINNMP, FMINP
ASE_INT16_SPEC	LUTI2 (halfword), LUTI4 (halfword)
ASE_INT8_SPEC	LUTI2 (byte), LUTI4 (byte)
ASE_INT_MUL_SPEC	PMULL2, SMLAL2, SMLSL2, SMULL2, SQDMLAL2, SQDMLSL2, SQDMULL2, UMLAL2, UMLSL2, UMULL2
ASE_INT_VREDUCE_SPEC	ADDP, ADDV, SADALP, UADALP, UADDLP, UADDLV, UMAX, UMAXP, UMIN, UMINP

Event	Instructions added
ASE_SVE_FP_CVT_SPEC	BF1CVTL, BF1CVTL2, BF2CVTL, BF2CVTL2, BFCVTN, BFCVTN2, F1CVTL, F1CVTL2, F2CVTL, F2CVTL2, FCVTN, FCVTN2, FCVTNS, FCVTNU, FCVTPS, FCVTPU, FCVTZS, FCVTZU
ASE_SVE_FP_DOT_SPEC	FDOT (2-way, by element), FDOT (2-way, vector), FDOT (4-way, by element), FDOT (4-way, vector)
ASE_SVE_FP_FMA_SPEC	FMLALLBB, FMLALLBT, FMLALLTB, FMLALLTT, FMLALT
ASE_SVE_FP_VREDUCE_SPEC	FAMAX, FAMIN, FMINNMP
ASE_SVE_INT_MUL_SPEC	PMULL2, SMLAL2, SMLSL2, SMULL2, SQDMLAL2, SQDMLSL2, SQDMULL2, UMLAL2, UMLSL2, UMULL2
ASE_SVE_INT_VREDUCE_SPEC	ADDP, ADDV, SADALP, UADALP, UADDLP, UADDLV, UMAX, UMAXP, UMIN, UMINP.
FPASE_LDST_REG_SPEC	LDAP1, LDAPUR, LDNP, LDP, LDR, LDUR, STL1, STLUR, STNP, STP, STR, STUR
FPASE_LD_REG_SPEC	LDAP1, LDAPUR, LDNP, LDP, LDR, LDUR
FPASE_ST_REG_SPEC	STL1, STLUR, STNP, STP, STR, STUR
FP_FMA_SPEC	FMLALB, FMLALLBB, FMLALLBT, FMLALLTB, FMLALLTT, FMLALT
INT_MUL_SPEC	PMULL2, SMLAL2, SMLSL2, SMULL2, SQDMLAL2, SQDMLSL2, SQDMULL2, UMLAL2, UMLSL2, UMULL2

In the same section, the following instructions are removed:

Event	Instructions removed
ASE_INT_VREDUCE_SPEC	UADDVL
ASE_SVE_INT_VREDUCE_SPEC	UADDVL

In the same section, the following Advanced SIMD instructions:

Event	Instructions
ASE_INT_DOT_SPEC	SUDOT
ASE_INT_MMLA_SPEC	SMMLA, UMMLA, USMMLA
ASE_SVE_INT_MMLA_SPEC	SMMLA, UMMLA, USMMLA
ASE_SVE_INT_DOT_SPEC	SUDOT

are changed to read:

Event	Instructions
ASE_INT_DOT_SPEC	SUDOT (by element)
ASE_INT_MMLA_SPEC	SMMLA (vector), UMMLA (vector), USMMLA (vector)
ASE_SVE_INT_MMLA_SPEC	SMMLA (vector), UMMLA (vector), USMMLA (vector)
ASE_SVE_INT_DOT_SPEC	SUDOT (by element)

2.27 C23004

In section **A2.2.1 “The Armv8.0 architecture extension”**, under the heading ‘FEAT_IVIPT, The IVIPT Extension’, the text that reads:

> FEAT_IVIPT is OPTIONAL from Armv8.0.

is changed to read:

> FEAT_IVIPT is mandatory from Armv8.0.

2.28 D23021

In section **I5.3.46 "PMPCSCTL, PC Sample-based Profiling Control Register"**, under the heading 'Accessing PMPCSCTL', the text that reads:

> Accesses to this register use the following encodings in the System register encoding space:
>
> Accessible at offset `0xE50` from PMU PMPCSCTL can be accessed through the PMU block as follows:

Frame	Offset
PMU	`0xE50`

> - When DoubleLockStatus(), or !IsCorePowered(), or OSLockStatus() or ! AllowExternalPMUAccess(), accesses to this register generate an error response.
> - Otherwise, accesses to this register are RW.

is changed to read:

> Accesses to this register use the following encodings:
>
> Accessible at offset `0xE50` from PMU PMPCSCTL can be accessed through the PMU block as follows:

Frame	Offset
PMU	`0xE50`

> - When DoubleLockStatus(), or !IsCorePowered(), or OSLockStatus(), accesses to this register generate an error response.
> - Otherwise, accesses to this register are RW.

In section **I3.1.4 "Access permissions for external views of the Performance Monitors"**, under the heading 'Table I3-1 Access permissions for the Performance Monitors registers when FEAT_PMUv3_EXT64 is implemented', the text that reads:

Offset	Register	Domain	Off	DLK	OSLK	EPMAD	EPMSSAD	Def
`0xE30`	PMPCSCTL	Core	Error	Error	Error	Error	-	RW

is changed to read:

Offset	Register	Domain	Off	DLK	OSLK	EPMAD	EPMSSAD	Def
`0xE30`	PMPCSCTL	Core	Error	Error	Error	-	-	RW

The equivalent changes are made in the same section, under the heading 'Table I3-2 Access permissions for the Performance Monitors registers when FEAT_PMUv3_EXT32 is implemented'.

2.29 R23026

In section **A2.3.5 "The Armv9.4 architecture extension"**, under the heading 'FEAT_D128, 128-bit Translation Tables, 56 bit PA', the text that reads:

> FEAT_D128, 128-bit Translation Tables, 56 bit PA
>
> FEAT_D128 adds support for the VMSAv9-128 translation system, comprising the following:
> - 128-bit translation table descriptors.
> - 56-bit physical addresses.
> - 56-bit virtual addresses.
> - 128-bit System registers.
> - 128-bit atomic instructions.
> - TLBIP VA*, TLBIP RVA*, TLBIP IPA*, TLBIP RIPA* instructions that can take 128-bit inputs.
> - **IMPLEMENTATION DEFINED** System instructions that can take 128-bit inputs.
>
> This feature is supported in AArch64 state only.
>
> FEAT_D128 is OPTIONAL from Armv9.3.
>
> If FEAT_D128 is implemented, then FEAT_SYSREG128 is implemented.
>
> If FEAT_D128 is implemented, then FEAT_SYSINSTR128 is implemented.
>
> If FEAT_D128 is implemented, then FEAT_LSE128 is implemented.
>
> If FEAT_D128 is implemented, then FEAT_S1PIE is implemented.
>
> If FEAT_D128 and EL2 are implemented, then FEAT_S2PIE is implemented.
>
> If FEAT_D128 is implemented, then FEAT_AIE is implemented.
>
> If FEAT_D128 is implemented, then FEAT_TCR2 is implemented.
>
> If FEAT_D128 is implemented, then FEAT_LVA is implemented.
>
> If FEAT_D128 is implemented, then FEAT_LPA2 is implemented.
>
> The following field identifies the presence of FEAT_D128:
> - ID_AA64MMFR3_EL1.D128.

is changed to read:

FEAT_D128, 128-bit Translation Tables

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

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

This feature is supported in AArch64 state only.

FEAT_D128 is OPTIONAL from Armv9.3.

If FEAT_D128 is implemented, then FEAT_SYSREG128 is implemented. If FEAT_D128 is implemented, then FEAT_SYSINSTR128 is implemented. If FEAT_D128 is implemented, then FEAT_LSE128 is implemented. If FEAT_D128 is implemented, then FEAT_S1PIE is implemented. If FEAT_D128 and EL2 are implemented, then FEAT_S2PIE is implemented. If FEAT_D128 is implemented, then FEAT_AIE is implemented. If FEAT_D128 is implemented, then FEAT_TCR2 is implemented.

The following field identifies the presence of FEAT_D128:

- ID_AA64MMFR3_EL1.D128.

2.30 E23034

In the section **A2 "A-profile Architecture"**, the following feature is added:

FEAT_AMU_EXTACR, defined as follows:

FEAT_AMU_EXTACR, is an OPTIONAL feature when FEAT_AMU_EXT is implemented.

When FEAT_AMU_EXTACR is implemented, the PE implements one of the following registers in the external view of the AMU:

Offset	Register	Description	Root access	Secure access	Non-secure and Realm access
0xE40	AMSCR	Activity Monitors Secure Control Register	n/a	R/W	**RAZ/WI**
0xE48	AMROOTCR	Activity Monitors Root Control Register	R/W	**RAZ/WI**	**RAZ/WI**

If FEAT_RME is not implemented then AMSCR is implemented and AMROOTCR is not implemented.

If FEAT_RME is implemented then AMROOTCR is implemented and AMSCR is not implemented.

In the section **I6.5 "Activity Monitors external register descriptions"**, the following optional registers are added:

AMSCR, defined as follows:

> Bits [63:32] Reserved. This field is **RES0**.
>
> IMPL, bit [31] Indicates AMSCR is present. This bit reads-as-one.
>
> Bits [30:2] Reserved. This field is **RES0**.
>
> NSRA, bit [1] Non-secure Register Access.

NSRA	Description
0b0	Non-secure register access is disabled. Non-secure access to all AMU registers is **RAZ/WI**.
0b1	Non-secure register access is enabled.

> The AMU reset value of this field is **IMPLEMENTATION DEFINED**.
>
> Bit[0] Reserved. This field is **RES0**.

Also, AMROOTCR, defined as follows:

> Bits [63:32] Reserved. This field is **RES0**.
>
> IMPL, bit [31] Indicates AMROOTCR is present. This bit reads-as-one.
>
> Bits [30:6] Reserved. This field is **RES0**.
>
> RA, bits [5:4] Register Access.

RA	Description
0b00	Secure, Realm, and Non-secure register access is disabled. Root register access is enabled. Secure, Realm, and Non-secure access to all AMU registers is **RAZ/WI**.
0b01	Secure and Non-secure register access is disabled. Root and Realm register access is enabled. Non-secure and Secure access to all AMU registers is **RAZ/WI**.
0b10	Non-secure and Realm register access is disabled. Root and Secure register access is enabled. Non-secure and Realm access to all AMU registers is **RAZ/WI**.
0b11	Root, Secure, Non-secure, and Realm register access is enabled.

> The AMU reset value of this field is **IMPLEMENTATION DEFINED**.
>
> Bits [3:0] Reserved. This field is **RES0**.

2.31 C23062

In section **A1.1 "About the Arm architecture"**, the text that reads:

> Except where the architecture specifies differently, the programmer-visible behavior of an implementation that is compliant with the Arm architecture must be the same as a simple

> sequential execution of the program on the processing element. This programmer-visible behavior does not include the execution time of the program.

is changed to read:

> Except where the architecture specifies differently, the programmer-visible behavior of an implementation that is compliant with the Arm architecture must be the same as a simple sequential execution of the program on the processing element. The programmer-visible behavior does not include a specified execution time for any instruction but requires each instruction to execute in finite time, as long as another observer is not continually modifying any Location associated with a Memory Effect, other than an Explicit Memory Effect, that is architecturally required for execution of the instruction. In cases where another observer is continually modifying a Location associated with a Memory Effect, other than an Explicit Memory Effect, at least one of the observers must make forward progress in finite time. Unless otherwise specified, the architecturally defined effects of each instruction also complete in finite time.

In section **B2.15.1 "Normal memory"**, the text that reads:

> - A write to a memory location with the Normal attribute completes in finite time.
> - Writes to a memory location with the Normal memory type that is either Non-cacheable or Write-Through cacheable for both the Inner and Outer cacheability must reach the endpoint for that location in the memory system in finite time. Two writes to the same location, where at least one is using the Normal memory type, might be merged before they reach the endpoint unless there is an ordered-before relationship between the two writes. For the purposes of this requirement, the endpoint for a location in Conventional memory is the PoC.

is changed to read:

> - An Explicit Memory Write Effect to a memory location with the Normal attribute completes in finite time.
> - Explicit Memory Write Effects to a memory location with the Normal memory type that is either Non-cacheable or Write-Through cacheable for both the Inner and Outer cacheability must reach the endpoint for that location in the memory system in finite time. Two writes to the same location, where at least one is using the Normal memory type, might be merged before they reach the endpoint unless there is an ordered-before relationship between the two writes. For the purposes of this requirement, the endpoint for a location in Conventional memory is the PoC.

In section **B2.15.2 "Device memory"**, the text that reads:

> - A write to a memory location with any Device memory type completes in finite time.
> - Writes to a memory location with any Device memory attribute must reach the endpoint for that address in the memory system in finite time. Two writes of Device memory type to the same location might be merged before they reach the endpoint, unless both writes have the non-Gathering attribute or there is an ordered-before relationship between the two writes.

is changed to read:

- An Explicit Memory Write Effect to a memory location with any Device memory type completes in finite time.
- Explicit Memory Write Effects to a memory location with any Device memory attribute must reach the endpoint for that address in the memory system in finite time. Two Explicit Memory Write Effects of Device memory type to the same location might be merged before they reach the endpoint, unless both Explicit Memory Write Effects have the non-Gathering attribute or there is an ordered-before relationship between the two Explicit Memory Write Effects.

2.32 R23103

In section **B2.3.1 "Basic definitions"**, under the heading 'Instruction Fetch Barrier effects', the text that reads:

An Instruction Fetch Barrier effect (IFBE) is one of:

- A Context Synchronizing event.
- An exception entry; regardless of the value of SCTLR_ELx.EIS.

Note: If FEAT_ExS is not implemented, or if FEAT_ExS is implemented and the SCTLR ELx.EOS field is set, an Exception Return effect is an IFBE.

is changed to read:

An Instruction Fetch Barrier effect (IFBE) is one of:

- A Context Synchronization event.
- An exception entry to ELx, regardless of the value of SCTLR_ELx.EIS, due to an exception generated for any reason other than any of the following:
  - SVC instruction execution that is not trapped.
  - HVC instruction execution that is not disabled.
  - SMC instruction execution that is not trapped or disabled.
  - BKPT instruction execution.
  - BRK instruction execution.

Note: If FEAT_ExS is not implemented, or if FEAT_ExS is implemented and the SCTLR ELx.EOS field is 1, an Exception Return effect is an IFBE due to it being a Context synchronization event.

Note: If FEAT_ExS is not implemented, or if FEAT_ExS is implemented and the SCTLR ELx.EIS field is 1, the Exception entry effects due to the exceptions excluded above are IFBEs due to being Context Synchronization events.

In section **D23.2.159 "SCTLR_EL1, System Control Register (EL1)"**, in the field description of 'EIS, bit [22]', the following text is added:

When FEAT_ExS is implemented:

> ...
>
> If SCTLR_EL1.EIS is set to `0b0`:
>
> - ...
> - Some exception entries reported are not considered IFBEs per the memory model. See B2.3.1 “Basic definitions” for the list of exception entries.
>
> The following are not affected by the value of SCTLR_EL1.EIS:
>
> - ...
> - The memory model requirement for exception entry to generate an Instruction Fetch Barrier effect for some exception entries. See B2.3.1 “Basic definitions” for the list of exception entries.

The equivalent changes are made in the following sections:

- **D23.2.160 “SCTLR_EL2, System Control Register (EL2)”**.
- **D23.2.161 “SCTLR_EL3, System Control Register (EL3)”**.

In section **D1.3.2 “Exception entry”**, the rule I_{IJFWCQ} that reads:

> I_{JFWCQ}
>
> For the purposes of the memory model, exception entry is an Instruction Fetch Barrier effect even if this SCTLR_ELx.EIS is 0.

is changed to read:

> I_{JFWCQ}
>
> For the purposes of the memory model, some exception entries are Instruction Fetch Barrier effects, regardless of the value of SCTLR_ELx.EIS. See B2.3.1 “Basic definitions” for the list of exception entries.

2.33 D23109

In section **D24.2.91 “ID_AA64ZFR0_EL1, SVE Feature ID Register 0”**, under the heading ‘AES, bits [7:4]’, the text that reads:

> Software should not attempt to execute the instructions described by nonzero values of this field when the PE is in Streaming SVE mode if it is not known whether FEAT_SME_FA64 is implemented and enabled, irrespective of the value of this field.

is changed to read:

> Software should not attempt to execute the instructions described by nonzero values of this field when the PE is in Streaming SVE mode if it is not known whether FEAT_SME_FA64 is implemented and enabled, irrespective of the value of this field.

Under the heading 'SHA3, bits [35:32]', the text that reads:

> Software should not attempt to execute the instructions described by nonzero values of this field when the PE is in Streaming SVE mode if it is not known whether FEAT_SME_FA64 is implemented and enabled, irrespective of the value of this field.
>
> However, if both FEAT_SME2p1 and FEAT_SVE_SHA3 are implemented, then the SVE RAX1 instruction can be executed when the PE is in Streaming SVE mode regardless of whether FEAT_SME_FA64 or FEAT_SSVE_AES is implemented and enabled.

is changed to read:

> If FEAT_SME2p1 is not implemented, then the instructions described by nonzero values of this field might not be legal when the PE is in Streaming SVE mode.

Under the heading 'F16MM, bits [51:48]', the following text is added:

> The instructions described by nonzero values of this field might not be legal when the PE is in Streaming SVE mode.

Under the heading 'BF16, bits [23:20]', the text that reads:

> Software should not attempt to execute the SVE instruction BFMMLA when the PE is in Streaming SVE mode if it is not known whether FEAT_SME_FA64 is implemented and enabled, irrespective of the value of this field.

is changed to read:

> The SVE instruction BFMMLA might not be legal when the PE is in Streaming SVE mode.

Under the heading 'F64MM, bits [59:56]', the text that reads:

> Software should not attempt to execute the instructions described by nonzero values of this field when the PE is in Streaming SVE mode if it is not known whether FEAT_SME_FA64 is implemented and enabled, irrespective of the value of this field.

is changed to read:

> The instructions described by nonzero values of this field might not be legal when the PE is in Streaming SVE mode.

The equivalent changes are made in the following sections:

- **D24.2.91 "ID_AA64ZFR0_EL1, SVE Feature ID Register 0"** under heading 'F32MM, bits [55:52]'.
- **D24.2.91 "ID_AA64ZFR0_EL1, SVE Feature ID Register 0"**, under heading 'I8MM, bits [47:44]'.
- **D24.2.91 "ID_AA64ZFR0_EL1, SVE Feature ID Register 0"**, under heading 'SM4, bits [43:40]'.

2.34 D23129

In section **D24.2.57 "HCRX_EL2, Extended Hypervisor Configuration Register"**, in the field description of 'MCE2, bit [10]', the following text is removed:

> When the Effective 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.

2.35 C23130

In section **C3.2.15 "Memory Copy and Memory Set instructions"**, the following text is added:

> Note:
>
> The FEAT_MOPS instructions are expected to be the preferred approach for compilation for performance for any of the following cases:
>
> - The size or alignments of the copy or set operation are not known at compile time.
> - The size or alignments of the copy or set operation are known at compile time, but not amenable to the operation being performed using load and store instructions without looping.
>
> Arm recommends that hardware implementations optimize the performance of these cases.

2.36 R23144

In section **D24.2.87 "ID_AA64PFR0_EL1, AArch64 Processor Feature Register 0"**, in the field description of 'RAS, bits [31:28]', the text that reads:

`0b0011`	FEAT_RASv2 implemented. As `0b0010` and adds support for: • ERXGSR_EL1, to support System RAS agents. • Additional fine-grained EL2 traps for additional error record System registers. • The SCR_EL3.TWERR write control for error record System registers. Error records accessed through System registers conform to RAS System Architecture v2.

is changed to read:

`0b0011`	FEAT_RASv2 implemented. As `0b0010` and adds support for: • The error group status register, ERXGSR_EL1. • The SCR_EL3.TWERR write trap control for error record System registers. • Additional syndrome in ESR_ELx for error exceptions. Error records accessed through System registers conform to either RAS System Architecture v1.1 or RAS System Architecture v2.

In section **A2.2.10 "The Armv8.9 architecture extension"**, under the heading 'FEAT_RASv2, RAS Extension v2', the text that reads:

> FEAT_RASv2 adds the following features to the Reliability, Availability, and Serviceability Extension:
>
> - Adds the features defined by FEAT_RASSAv2 to System register error records.
> - Defines the ERXGSR_EL1 register.
> - Adds a Trap exception to EL3 for writes to RAS System registers.
> - Adds additional syndrome to ESR_ELx on an error exception, to give information on whether a location being accessed has been updated.

is changed to read:

> FEAT_RASv2 adds the following features to the Reliability, Availability, and Serviceability Extension:
>
> - Allows the features defined by FEAT_RASSAv2 to System register error records.
> - Defines the error group status register, ERXGSR_EL1.
> - Adds a control to trap writes to RAS error record System registers to EL3.
> - Adds additional syndrome to ESR_ELx for error exceptions.

2.37 R23151

In section **D1.3.5.4.2 "SVE First-fault and Non-fault loads"**, under rule 'R_{NGFTJ}', the text in the table that reads:

Corresponding FFR element	Vector element status	Content of destination vector element
FALSE	Active	Each byte of the element contains an independently **CONSTRAINED UNPREDICTABLE** choice of one of the following: • 1. • The previous value of that byte in the destination vector register. • If and only if all of the following apply, the value read from memory: ◦ The memory access for that byte was not an access to any type of Device memory. ◦ The memory access for that byte does not return information that cannot be accessed at the current or a lower level of privilege.
...	...	...

is changed to read:

Corresponding FFR element	Vector element status	Content of destination vector element
FALSE	Active	Each byte of the element contains an independently **CONSTRAINED UNPREDICTABLE** choice of one of the following: • 1. • The previous value of that byte in the destination vector register. • If and only if all of the following apply, the value read from memory: ◦ The memory access for that byte was not an access to any type of Device memory, or for a First-fault vector load was an access to Device memory inside the 64-byte window of the First active element (see B2.15.2 Device memory). ◦ The memory access for that byte does not return information that cannot be accessed at the current or a lower level of privilege.
...	...	...

2.38 D23206

In section **A2.2.8 "The Armv8.7 architecture extension"**, under the heading "FEAT_PAN3, Support for SCTLR_ELx.EPAN", the text that reads:

> FEAT_PAN3 is OPTIONAL from Armv8.0.

is changed to read:

> FEAT_PAN3 is OPTIONAL from Armv8.1.

2.39 D23233

The following section **K14.2.3 "WFE and WFI and barriers"** is removed:

> The Wait For Event and Wait For Interrupt instructions permit the PE to suspend execution and enter a low-power state. An explicit DSB barrier instruction is required if it is necessary to ensure memory accesses made before the WFI or WFE are visible to other observers, unless some other mechanism has ensured this visibility. Examples of other mechanism that would guarantee the required visibility are the DMB described in Posting a store before polling for acknowledgment, or a dependency on a load.
>
> The following example requires the DSB to ensure that the store is visible:
>
> AArch32

```
P1
    STR R0, [R2]
    DSB
Loop
    WFI
    B Loop
```

AArch64

```
P1
    STR W0, [X2]
    DSB <domain>
Loop
    WFI
    B Loop
```

This requirement is unchanged in Armv8 and later architectures by the presence of Load-Acquire or Store-Release.

2.40 D23239

In **F6.1.32 "VAND (immediate)"** and **F6.1.157 "VORN (immediate)"** the text that reads:

I32

is changed to read:

I16

and the text that reads:

I16

is changed to read:

I32

2.41 D23253

In section **D24.2.174 "SMCR_EL1", SME Control Register EL1"**, under the heading 'Configuration', the text that reads:

When the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}, this register has no effect on execution at EL0 and EL1.

is changed to read:

When the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}, this register has no effect on execution at EL0.

In the description of field 'FA64, bit [31]', the text that reads:

Controls whether execution of an A64 instruction is considered legal when the PE is in Streaming SVE mode.

is changed to read:

> Controls whether execution of an A64 instruction at EL1 is considered legal when the PE is in Streaming SVE mode. When the Effective value of HCR_EL2.{E2H, TGE} is not {1, 1}, controls whether execution of an A64 instruction at EL0 is considered legal when the PE is in Streaming SVE mode.

The equivalent change is made in **D24.2.175 "SMCR_EL2", SME Control Register EL2** in the description of field 'FA64, bit [31]'.

2.42 D23282

In section **D24.2.184 "TCR_EL3, Translation Control Register (EL3)**, in the description of field"PnCH, bit[34]", the text that reads:

> Protected attribute enable. Indicates use of bit[52] of the stage 1 translation table entry for translations using TTBR0_EL3.

PnCH	Meaning
0b0	For translations using TTBR0_EL3, bit[52] of each stage 1 translation table entry does not indicate protected attribute.
0b1	For translations using TTBR0_EL3, bit[52] of each stage 1 translation table entry indicates protected attribute.

> This field is **RES1** when TCR_EL3.D128 is 1.

is changed to read:

> Protected attribute enable. Enables use of bit[52] of stage 1 translation table entries as the Protected bit, for translations using TTBR0_EL3.

PnCH	Meaning
0b0	For translations using TTBR0_EL3, bit[52] of each stage 1 translation table entry is not the Protected bit.
0b1	For translations using TTBR0_EL3, bit[52] of each stage 1 translation table entry is the Protected bit.

> If bit[52] is used as the Protected bit, it is not used as the Contiguous bit. This field is **RES0** when TCR_EL3.D128 is 1.

The equivalent change is made in **D23.2.173 "TCR2_EL2, Extended Translation Control Register (EL2)"** and **D23.2.172 "TCR2_EL1, Extended Translation Control Register (EL1)"**.

2.43 D23305

In section **D8.6.5.2 "Combining stage 1 and stage 2 Cacheability attributes for Normal memory"**, the text that reads:

> R_{JSXRX}

If FEAT_MTE_PERM is implemented, all of the following apply:

- If either translation stage assigns a Device, Non-cacheable, or Write-through memory type, then the stage 1 memory type is treated as not having the Tagged attribute and the resultant memory type is as defined in Stage 2 memory type and Cacheability attributes when FWB is enabled.
- Otherwise, the stage 1 and stage 2 attributes are combined as shown in Table D8-93.

Table D8-93 Combined stage 1 and stage 2 attributes if FEAT_MTE_PERM is implemented

Stage 1 memory type and Cacheability attribute	Stage 2 memory type and Cacheability attribute	Resultant memory type and Cacheability attribute
Normal Write-back	Any	Normal Write-back
Normal Write-Back, Tagged	Normal Write-Back	Normal Write-Back, Tagged
Normal Write-Back, Tagged	Normal Write-back, NoTagAccess	Normal Write-back, Tagged, NoTagAccess

is changed to read:

R_{JSXRX}

If FEAT_MTE_PERM is implemented, all of the following apply:

- If the stage 1 memory type is not Tagged, the stage 2 NoTagAccess attribute is ignored.
- Otherwise, the stage 1 and stage 2 attributes are combined as shown in Table D8-93.

Table D8-93 Combined stage 1 and stage 2 attributes if FEAT_MTE_PERM is implemented

Stage 1 memory type and Cacheability attribute	Stage 2 memory type and Cacheability attribute	Resultant memory type and Cacheability attribute
Normal Write-Back, Tagged	Normal Write-Back	Normal Write-Back, Tagged
Normal Write-Back, Tagged	Normal Write-back, NoTagAccess	Normal Write-back, Tagged, NoTagAccess

2.44 D23315

In the following sections:

- **C8.2.37 “BF1CVT, BF2CVT”**.
- **C8.2.38 “BF1CVTLT, BF2CVTLT”**.
- **C8.2.39 “BFADD (predicated)”**.
- **C8.2.40 “BFADD (unpredicated)”**.
- **C8.2.41 “BFCLAMP”**.
- **C8.2.43 “BFCVTN”**.
- **C8.2.47 “BFMAX”**.
- **C8.2.48 “BFMAXNM”**.

- **C8.2.49 “BFMIN”**.
- **C8.2.50 “BFMINNM”**.
- **C8.2.51 “BFMLA (indexed)”**.
- **C8.2.52 “BFMLA (vectors)”**.
- **C8.2.57 “BFMLS (indexed)”**.
- **C8.2.58 “BFMLS (vectors)”**.
- **C8.2.64 “BFMUL (indexed)”**.
- **C8.2.65 “BFMUL (vectors, predicated)”**.
- **C8.2.66 “BFMUL (vectors, unpredicated)”**.
- **C8.2.67 “BFSUB (predicated)”**.
- **C8.2.68 “BFSUB (unpredicated)”**.
- **C8.2.141 “F1CVT, F2CVT”**.
- **C8.2.142 “F1CVTLT, F2CVTLT”**.
- **C8.2.155 “FAMAX”**.
- **C8.2.156 “FAMIN”**.
- **C8.2.168 “FCVTN”**.
- **C8.2.169 “FCVTNB”**.
- **C8.2.171 “FCVTNT (unpredicated)”**.
- **C8.2.435 “LUTI2”**.
- **C8.2.436 “LUTI4”**.

the operational pseudocode that reads:

```
CheckSVEEnabled();
```

is changed to read:

```
if IsFeatureImplemented(FEAT_SME2) then CheckSVEEnabled(); else
 CheckNonStreamingSVEEnabled();
```

In the following sections:

- **C8.2.179 “FDOT (2-way, indexed, FP8 to FP16)”**.
- **C8.2.181 “FDOT (2-way, vectors, FP8 to FP16)”**.
- **C8.2.182 “FDOT (4-way, indexed)”**.
- **C8.2.183 “FDOT (4-way, vectors)”**.
- **C8.2.211 “FMLALB (indexed, FP8 to FP16)”**.
- **C8.2.213 “FMLALB (vectors, FP8 to FP16)”**.

- **C8.2.214 “FMLALLBB (indexed)”**.
- **C8.2.215 “FMLALLBB (vectors)”**.
- **C8.2.216 “FMLALLBT (indexed)”**.
- **C8.2.217 “FMLALLBT (vectors)”**.
- **C8.2.218 “FMLALLTB (indexed)”**.
- **C8.2.219 “FMLALLTB (vectors)”**.
- **C8.2.220 “FMLALLTT (indexed)”**.
- **C8.2.221 “FMLALLTT (vectors)”**.
- **C8.2.223 “FMLALT (indexed, FP8 to FP16)”**.
- **C8.2.225 “FMLALT (vectors, FP8 to FP16)”**.

the operational pseudocode that reads:

```
if IsFeatureImplemented(FEAT_FP8xxx) then CheckSVEEnabled(); else
 CheckStreamingSVEEnabled();
```

is changed to read:

```
if IsFeatureImplemented(FEAT_SSVE_FP8xxx) && IsFeatureImplemented(FEAT_FP8xxx) then
    CheckSVEEnabled();
elsif IsFeatureImplemented(FEAT_FP8xxx) then
    CheckNonStreamingSVEEnabled();
else
    CheckStreamingSVEEnabled();
```

where xxx is either FMA, DOT2, or DOT4, as appropriate.

2.45 D23325

In section **C5.1.3.2 “Barriers and CLREX”**, the text that reads:

> The value of op2 determines the instruction, as follows.

0b001	DSB instruction, Memory nXS barrier variant.
0b010	CLREX instruction.
0b100	DSB instruction, Memory barrier variant.
0b101	DMB instruction.
0b110	ISB instruction.
0b000, 0b011, 0b111	**UNDEFINED**.

is changed to read:

> The value of op2 determines the instruction, as follows.

0b001	DSB instruction, Memory nXS barrier variant.
0b010	CLREX instruction.
0b100	DSB instruction, Memory barrier variant.
0b101	DMB instruction.
0b110	ISB instruction.
0b111	SB instruction.
0b000, 0b011	**UNDEFINED**.

2.46 R23333

In section **D20 "About the RAS Extension"**, under 'Example D20-1 Minimal implementation of RAS Extension', the text that reads:

> The ESB instruction only has an effect on virtual SError exceptions, and only if EL2 is implemented. Otherwise, ESB is implemented as a no-op. See ESB and Virtual SError exceptions.

is changed to read:

> The ESB instruction is implemented as **NOP**. See also ESB and Virtual SError exceptions.

2.47 D23338

In section **J1.1.3 aarch64/functions**, in the pseudocode function MemAtomic(), the code segment that reads:

```
bits(size) MemAtomic(bits(64) address, bits(size) cmpoperand, bits(size) operand,
                     AccessDescriptor accdesc_in)
    ...
    constant integer bytes = size DIV 8;
    ...
    // MMU or MPU lookup
    constant AddressDescriptor memaddrdesc = AArch64.TranslateAddress(address,
 accdesc,
                                                                      aligned,
 size);
    ...
    // Effect on exclusives
    if memaddrdesc.memattrs.shareability != Shareability_NSH then
        ClearExclusiveByAddress(memaddrdesc.paddress, ProcessorID(), size);
```

is changed to read:

```
bits(size) MemAtomic(bits(64) address, bits(size) cmpoperand, bits(size) operand,
                     AccessDescriptor accdesc_in)
    ...
    constant integer bytes = size DIV 8;
    ...
    // MMU or MPU lookup
    constant AddressDescriptor memaddrdesc = AArch64.TranslateAddress(address,
 accdesc,
```

```
                                                                    aligned,
bytes);
    ...
    // Effect on exclusives
    if memaddrdesc.memattrs.shareability != Shareability_NSH then
        ClearExclusiveByAddress(memaddrdesc.paddress, ProcessorID(), bytes);
```

The equivalent change is made in the pseudocode function MemAtomicRCW().

2.48 D23339

In section **D8.6.6 "Stage 2 memory type and Cacheability attributes when FWB is enabled"**, the following text is added:

> R_{00001}
>
> If MemAttr[1:0] bits define Device memory attributes, then stage 2 Device memory attributes are combined with stage 1 memory attributes.

2.49 R23354

In section **B2.6.9 "Data Synchronization Barrier"**, the text that reads:

> If FEAT_MTE2 is implemented, on completion of a DSB instruction operating over the Non-shareable domain, all updates to TFSR_ELx.TFx or TFSRE0_EL1.TFx due to Tag Check fails caused by accesses for which the DSB operates will be complete. For more information on FEAT_MTE2, see The Memory Tagging Extension.

is changed to read:

> If FEAT_MTE_ASYNC is implemented, on completion of a DSB instruction with the LD qualifier, or neither the LD nor ST qualifier, all updates to TFSR_ELx.TFy or TFSRE0_EL1.TFy due to Tag Check Faults caused by accesses generated by instructions occurring in program order before the DSB will be complete. For more information on FEAT_MTE_ASYNC, see Chapter D10 The Memory Tagging Extension.

The equivalent changes are made in **D10.7.1 "Asynchronous Tag Check Faults"**.

The following rule that reads:

> R_{JNNZ}
>
> An implicit write to a Tag Fault Status Register accessible at ELx that is caused by an asynchronous Tag Check Fault is synchronized by any of the following:
>
> - An exception entry to ELx if SCTLR_ELx.ITFSB is 1.
> - A DSB over the Non-shareable domain at ELx in program order, after the instruction causing the Tag Check Fault.

is changed to read:

> R_{JNNZ}
>
> An implicit write to a Tag Fault Status Register accessible at ELx that is caused by an asynchronous Tag Check Fault is synchronized by any of the following:
>
> - An exception entry to ELx if SCTLR_ELx.ITFSB is 1.
> - A DSB instruction with the LD qualifier, or neither the LD nor ST qualifier, at ELx in program order after the instruction causing the Tag Check Fault.

The equivalent text in section **D24.1.2 "General behavior of accesses to the AArch64 System registers"** that reads:

> If FEAT_MTE2 is implemented, a DSB instruction over the Non-shareable domain or an exception entry to ELy with SCTLR_ELy.ITFSB = `0b1` is required between an indirect write to TFSRE0_EL1, or any TFSR_ELx accessible at ELy, and a direct read or direct write of that register.

is changed to read:

> If FEAT_MTE_ASYNC is implemented, a DSB instruction with the LD qualifier, or neither the LD nor ST qualifier, or an exception entry to ELy with SCTLR_ELy.ITFSB = `0b1`, is required between an indirect write to TFSRE0_EL1, or any TFSR_ELx accessible at ELy, and a direct read or direct write of that register.

2.50 R23355

In section **D14.3 "Common event numbers"**, the text that reads:

> For an odd-numbered counter <n+1>, the counter increments when an event increments the preceding even-numbered counter <n> on the same PE causing unsigned overflow of bits [31:0] of the event counter <n>, and any of the following are true:
>
> - FEAT_PMUv3p5 is not implemented.
> - EL2 is not implemented and PMCR.LP is 1.
> - EL2 is implemented, <n> is less than HDCR.HPMN, and PMCR.LP is 0.
> - EL2 is implemented, <n> is greater than or equal to HDCR.HPMN, and HDCR.HLP is 0

is changed to read:

> For an odd-numbered counter <n+1>, the counter increments when an event increments the preceding even-numbered counter <n> on the same PE causing unsigned overflow of bits [31:0] of the event counter <n>, and any of the following are true:
>
> - FEAT_PMUv3p5 is not implemented.
> - EL2 is not implemented and PMCR.LP is 1.
> - EL2 is implemented, <n> is less than the Effective value of HDCR.HPMN, and PMCR.LP is 0.

- EL2 is implemented, <n> is greater than or equal to the Effective value of HDCR.HPMN, and HDCR.HLP is 0.

If EL2 is implemented and <n+1> is equal to the Effective value of HDCR.HPMN, then it is **UNPREDICTABLE** whether the counter counts.

If FEAT_PMUv3_EXTPMN is implemented and <n+1> is equal to the Effective value of PMCCR.EPMN, then it is **UNPREDICTABLE** whether the counter counts.

2.51 D23362

In section **G8.2.129 "SPSR_abt, Saved Program Status Register (Abort mode)"**, in the field description of 'N, bit[31]', the text that reads:

> Negative Condition flag. Set to the value of PSTATE.N on taking an exception to Abort mode, and copied to PSTATE.N on executing an illegal exception return operation in Abort mode.

is changed to read:

> Negative Condition flag. Set to the value of PSTATE.N on taking an exception to Abort mode, and copied to PSTATE.N on executing an exception return operation in Abort mode.

The equivalent changes are made in the following field descriptions:

- Z, bit [30].
- C, bit [29].
- V, bit [28].
- Q, bit [27].
- IT, bits [15:10, 26:25].
- SSBS, bit [23].
- PAN, bit [22].
- DIT, bit [21].
- IL, bit [20].
- GE, bits [19:16].
- E, bit [9].
- A, bit [8].
- I, bit [7].
- F, bit [6].
- T, bit [5].
- M[4:0], bits [4:0].

The equivalent changes are also made in the following sections:

- **G8.2.130 “SPSR_fiq, Saved Program Status Register (FIQ mode)”**.
- **G8.2.132 “SPSR_irq, Saved Program Status Register (IRQ mode)”**.
- **G8.2.135 “SPSR_und, Saved Program Status Register (Undefined mode)”**.

2.52 D23379

In section **A2.2.4 “The Armv8.3 architecture extension”**, under the heading ‘FEAT_FPACC_SPEC, Faulting on combined pointer authentication instructions’, the text that reads:

> FEAT_FPACC_SPEC, Faulting on combined pointer authentication instructions
>
> FEAT_FPACC_SPEC introduces consistent impact of speculation for combined instructions that perform authentication.

is changed to read:

> FEAT_FPACC_SPEC, Faulting on pointer authentication instructions
>
> FEAT_FPACC_SPEC introduces consistent impact of speculation for instructions that perform authentication.

2.53 C23401

In section **D8.18.2 “Instruction caches”**, the following rule is added:

> R_{X0000}
>
> If CTR_EL0.DIC is 1, instruction cache maintenance is not required after overwriting instructions, including for different VA aliases of the affected Locations, regardless of the value of CTR_EL0.L1Ip.

2.54 D23403

In section **D24.5.12 “PMEVTYPER<n>_EL0, Performance Monitors Event Type Registers, n = 0 - 30”**, the text that reads:

0b1	Count events from any PE with the same affinity at level 1 and above as this PE.

is changed to read:

0b1	Count events from any PE with the same affinity at level 1 and above as this PE.

> Unless otherwise stated:

- If the event counts PE cycles when a stall condition is true, then the counter counts Processor cycles when the stall condition is true for all of these PEs.
- If the event counts PE cycles when any other condition is true, then the counter counts Processor cycles when the condition is true for any of these PEs. See Cycle event counting.
- Otherwise, the event counts by the sum of the count across all of these PEs. See also Cycle event counting.

An equivalent change is made in section **D13.6 "Multithreaded implementations"** and the following text is removed:

> If an implementation is multithreaded and the Effective value of PMEVTYPER<n>.MT == 1, events on other PEs with the same level 1 Affinity are also counted.

2.55 D23410

In section **D24.5.29 "PMZR_EL0, Performance Monitors Zero with Mask"**, under the heading 'Configuration', the text that reads:

> External register PMZR_EL0 bits [63:0] are architecturally mapped to AArch64 System register PMZR_EL0[63:0].
>
> This register is present only when FEAT_PMUv3_EXT64 is implemented and FEAT_PMUv3p9 is implemented. Otherwise, direct accesses to PMZR_EL0 are **RES0**.

is changed to read:

> External register PMZR_EL0 bits [63:0] are architecturally mapped to AArch64 System register PMZR_EL0[63:0].
>
> This register is present only when FEAT_PMUv3_EXT is implemented and FEAT_PMUv3p9 is implemented. Otherwise, direct accesses to PMZR_EL0 are **RES0**.

2.56 D23414

In section **D24.10.19, "CNTPCT_EL0, Counter-timer Physical Count Register"**, the code segment that reads:

```
if (((IsFeatureImplemented(FEAT_ECV) && EL2Enabled()) && (SCR_EL3.ECVEn == >   '1'))
 && (CNTHCTL_EL2.ECV == '1')) && (!ELIsInHost(EL0)) then
    X[t, 64] = PhysicalCountInt() - CNTPOFF_EL2;
```

is changed to read:

```
if (((IsFeatureImplemented(FEAT_ECV) && EL2Enabled() && (!HaveEL(EL3) || >
 (SCR_EL3.ECVEn == '1')) && (CNTHCTL_EL2.ECV == '1')) && (!ELIsInHost(EL0)) > then
```

```
    X[t, 64] = PhysicalCountInt() - CNTPOFF_EL2;
```

Equivalent changes are made in code segments in the following sections:

- **D24.10.20, "CNTPCTSS_EL0, Counter-timer Self-Synchronized Physical Counter Register"**.
- **D24.10.18 "CNTP_TVAL_EL0, Counter-timer Physical Timer TimerValue Register"**.
- **D24.10.8 "CNTHPS_TVAL_EL2, Counter-timer Secure Physical Timer TimerValue Register"**.
- **D24.10.5 "CNTHP_TVAL_EL2, Counter-timer Physical Timer TimerValue Register"**.
- **G8.7.19, "CNTPCT, Counter-timer Physical Count register"**.
- **G8.7.20, "CNTPCTSS, Counter-timer Self-Synchronized Physical Count register"**.
- **G8.7.18 "CNTP_TVAL, Counter-timer Physical Timer TimerValue register"**.
- **G8.7.8 "CNTHPS_TVAL, Counter-timer Secure Physical Timer TimerValue Register"**.
- **G8.7.5 "CNTHP_TVAL, Counter-timer Hyp Physical Timer TimerValue register"**.

2.57 C23419

In section **A2.2.3 "The Armv8.2 architecture extension"**, under the heading 'FEAT_LVA, Large VA support', the following text that reads:

> FEAT_LVA supports a larger virtual address (VA) space for each translation table base register of up to 52 bits when using the 64KB translation granule.

is changed to read:

> FEAT_LVA supports a virtual address (VA) space for each translation table base register of up to 52 bits when using any of the following:
>
> - VMSAv9-128 translation format.
> - VMSAv8-64 translation format with the 64KB translation granule.

In section **D8.1.6 "Implemented physical address size"**, the following rule that reads:

> R_{RNRJPM}
>
> A PA size greater than 52 bits is only supported by the VMSAv9-128 translation system.

is changed to read:

> R_{INRJPM}
>
> A PA size greater than 52 bits can only be expressed by the VMSAv9-128 translation system.

In section **D8.1.8 "Supported virtual address ranges"**, the following rule that reads:

> R_{RXQGZR}

> For the VMSAv9-128 translation system, the maximum VA size is 56 bits.

is changed to read:

> R_{RXQGZR}
>
> For the VMSAv9-128 translation system, the maximum VA size is one of the following:
> - If FEAT_LVA3 is implemented, then 56 bits.
> - If FEAT_LVA3 is not implemented, but FEAT_LVA is implemented, then 52 bits.
> - If FEAT_LVA is not implemented, then 48 bits.

In section **D8.1.9 "Input address size configuration"**, the following rule that reads:

> R_{RVRKKV}
>
> For a stage 1 translation using the VMSAv9-128 translation system, the minimum Effective value of TnSZ is one of the following:
> - If the translation regime supports a single IA range, then 8.
> - If the translation regime supports two IA ranges, then 9.

is changed to read:

> R_{RVRKKV}
>
> For a stage 1 translation using the VMSAv9-128 translation system, the minimum Effective value of TnSZ is one of the following:
> - If FEAT_LVA3 is implemented, then one of the following:
>   - If the translation regime supports a single IA range, then 8.
>   - If the translation regime supports two IA ranges, then 9.
> - If FEAT_LVA3 is not implemented, but FEAT_LVA is implemented, then 12.
> - If FEAT_LVA is not implemented, then 16.

In section **D24.2.182 "TCR_EL1, Translation Control Register (EL1)"**, under the description of the field 'IPS, bits [34:32]', the table that reads:

is changed to read:

The equivalent changes are made in the following sections for the 'IPS' field:

- **D24.2.183 "TCR_EL2, Translation Control Register (EL2)"**.

The equivalent changes are made in the following sections for the 'PS' field:

- **D24.2.184 "TCR_EL3, Translation Control Register (EL3)"**.
- **D24.2.210 "VTCR_EL2, Virtualization Translation Control Register"**.

2.58 D23442

In section **D18.2.6 "Events packet"**, under 'Events packet payload', in E[25], the text that reads:

> E[25], byte 3 bit [1], when SZ == `0b10` or SZ == `0b11`
>
> SMCU or external coprocessor operation.
>
> When FEAT_SPE_SME is implemented

E[25]	Description
`0b0`	Operation did not execute on an SMCU or external coprocessor.
`0b1`	Operation executed on an SMCU or external coprocessor.

> When FEAT_SPEv1p4 is implemented
>
> This bit reads-as-zero.
>
> Otherwise
>
> This bit reads as an **IMPLEMENTATION DEFINED** value.

is changed to read:

> E[25], byte 3 bit [1], when SZ == `0b10` or SZ == `0b11`
>
> Streaming Mode Compute Unit (SMCU) or other shared resource operation.
>
> When (FEAT_SPE_SME is implemented or FEAT_SPEv1p5 is implemented) and an SMCU or other shared resource is implemented

E[25]	Description
`0b0`	Operation did not execute on an SMCU or other shared resource. ————————————————————+ Operation executed on an SMCU or other shared resource.
`0b1`	

> A shared resource means a resource that is shared between PEs for the execution of instructions. It is **IMPLEMENTATION DEFINED** which instructions can be executed on a shared resource.
>
> When FEAT_SPEv1p4 is implemented or FEAT_SPE_SME is implemented
>
> This bit reads-as-zero.
>
> Otherwise
>
> This bit reads as an **IMPLEMENTATION DEFINED** value.

In section **D24.7.8 "PMSEVFR_EL1, Sampling Event Filter Register**, in E[25], the text that reads:

> E[25], bit [25] When FEAT_SPE_SME is implemented and event 25 is implemented: Filter on Streaming Mode Compute Unit (SMCU) or external coprocessor operation event.

E[25]	Meaning
0b0	SMCU or external coprocessor operation event is ignored.
0b1	Do not record samples that have the SMCU or external coprocessor operation event == 0.

is changed to read:

> E[25], bit [25]
>
> When (FEAT_SPE_SME is implemented or FEAT_SPEv1p5 is implemented) and event 25 is implemented:
>
> Filter on Streaming Mode Compute Unit (SMCU) or other shared resource operation event.

E[25]	Meaning
0b0	SMCU or other shared resource operation event is ignored. ——————————————————————————+ Do not record samples that have the SMCU or other shared resource operation event == 0.
0b1	

The equivalent change is made in section **D24.7.14 “PMSNEVFR_EL1, Sampling Inverted Event Filter Register”**.

2.59 D23455

In section **J1.1.3 “aarch64/functions”**, in the pseudocode function TLBIMatch(), the following code segment is removed::

```
    if tlbi.attr == TLBI_ExcludeXS && tlb_entry.context.xs == '1' then
        match = FALSE;
```

2.60 D23463

In section **C6.2.374 “STLUR”**, the decode pseudocode for the 64-bit variant of the instruction that reads:

```
constant integer datasize = 32;
```

is changed to read:

```
constant integer datasize = 8 << scale;
```

2.61 D23470

In section **J1.3 "Shared pseudocode"**, the code that reads:

```
bit EffectiveEA()
    if !HaveEL(EL3) || (Halted() && EDSCR.SDD == '0') then
        return '0';
    else
        return if HaveAArch64() then SCR_EL3.EA else SCR.EA;
```

is changed to read:

```
bit EffectiveEA()
    if !HaveEL(EL3) || Halted() then
        return '0';
    else
        return if HaveAArch64() then SCR_EL3.EA else SCR.EA;
```

2.62 D23480

In section **D24.2.40 "ESR_EL1, Exception Syndrome Register (EL1)"**, the text that reads:

> Additional Information for ISS encoding for a PAC Fail exception The following instructions generate a PAC Fail exception when the Pointer Authentication Code (PAC) is incorrect:
>
> - AUTDA, AUTDZA.
> - AUTDB, AUTDZB.
> - AUTIA, AUTIA1716, AUTIASP, AUTIAZ, AUTIZA.
> - AUTIB, AUTIB1716, AUTIBSP, AUTIBZ, AUTIZB.
>
> If FEAT_FPACCOMBINE is implemented, the following instructions generate a PAC Fail exception when the Pointer Authentication Code (PAC) is incorrect:
>
> - RETAA, RETAB.
> - BLRAA, BLRAAZ, BLRAB, BLRABZ.
> - BRAA, BRAB, BRAAZ, BRABZ.
> - ERETAA, ERETAB.
> - LDRAA, LDRAB, whether the authenticated address is written back to the base register or not.

is changed to read:

> Additional Information for ISS encoding for a PAC Fail exception The following instructions generate a PAC Fail exception when the Pointer Authentication Code (PAC) is incorrect:
>
> - AUTDA, AUTDZA.
> - AUTDB, AUTDZB.

- AUTIA, AUTIA1716, AUTIASP, AUTIAZ, AUTIZA, AUTIASPPC, AUTIASPPCR, AUTIA171615.
- AUTIB, AUTIB1716, AUTIBSP, AUTIBZ, AUTIZB, AUTIBSPPC, AUTIBSPPCR, AUTIB171615. If FEAT_FPACCOMBINE is implemented, the following instructions generate a PAC Fail exception when the Pointer Authentication Code (PAC) is incorrect:
- RETAA, RETAB.
- RETAASPPC, RETAASPPCR, RETABSPPC, RETABSPPCR.
- BLRAA, BLRAAZ, BLRAB, BLRABZ.
- BRAA, BRAB, BRAAZ, BRABZ.
- ERETAA, ERETAB.
- LDRAA, LDRAB, whether the authenticated address is written back to the base register or not.

The equivalent changes are made in the following sections:

- **D24.2.41 "ESR_EL2, Exception Syndrome Register (EL2)"**.
- **D24.2.42 "ESR_EL3, Exception Syndrome Register (EL3)"**.

2.63 D23504

In section **A2.3.6 "The Armv9.5 architecture extension"**, under the heading 'FEAT_ETS3, Enhanced Translation Synchronization', the text that reads:

> FEAT_ETS3 introduces support for enhanced memory access ordering requirements for translation table walks. This feature is supported in both AArch64 and AArch32 states. FEAT_ETS3 is OPTIONAL from Armv9.4. FEAT_ETS3 is mandatory from Armv9.5.

Is changed to read:

> FEAT_ETS3 introduces support for enhanced memory access ordering requirements for translation table walks. This feature is supported in both AArch64 and AArch32 states. FEAT_ETS3 is OPTIONAL from Armv8.0. FEAT_ETS3 is mandatory from Armv9.5.

2.64 D23521

In section **D8.5.2 "The dirty state"**, the text that reads:

> ```
> R<sub>DYCFD</sub>
>
> If a write access translated by a writable-clean descriptor is not performed
>  architecturally, then unless specified here hardware does not update the dirty
>  state of that descriptor. In all of the following cases, hardware is permitted to
>  update the dirty state while attempting to translate the explicit write access:
> ```
>
> ...

is changed to read:

> R_{DYCFD}
>
> If a write access translated by a writable-clean descriptor is not performed architecturally, then unless specified here hardware does not update the dirty state of that descriptor. In all of the following cases, hardware is permitted to update the dirty state while attempting to translate the explicit write access:
>
> ...
>
> - The descriptor is used to translate accesses from the Trace Buffer Unit. See "Accesses to the trace buffer" in D6.3.4.

2.65 C23522

In section **D8.1.2.4 "Non-secure EL2&0 translation regime"**, the following rule that reads:

> I_{00001}
>
> The EL2&0 regime might also be used when execution is at EL1, and in some cases at EL0 when HCR_EL2.{E2H, TGE} is not {1, 1}. For example:
>
> - At EL1 when FEAT_NV2 is in use, as described in D8.14.6.2 "Loads and stores generated by transforming register accesses".
> - At EL1 or EL0 when the Statistical Profiling Unit is enabled, as described in D17.7.2 "The owning translation regime".
> - At EL1 or EL0 when the Trace Buffer Extension is enabled, as described in D6.3.5 "The owning translation regime".

The equivalent changes are made in the following sections:

- **D8.1.2.5 "Secure EL2&0 translation regime"**.
- **D8.1.2.7 "Non-secure EL2 translation regime"**.
- **D8.1.2.8 "Secure EL2 translation regime"**.

2.66 D23541

In section **J1.3 "Shared Pseudocode"**, in the pseudocode function Hint_WFE(), the following code segment that reads:

```
Hint_WFE()
    ...
    (trap, target_el) = AArch64.CheckForWFxTrap(WFxType_WFE);
    if trap then
        if IsFeatureImplemented(FEAT_TWED) then
            // Determine if trap delay is enabled and delay amount
            boolean delay_enabled;
```

```
            integer delay;
            (delay_enabled, delay) = WFETrapDelay(target_el);
            if !WaitForEventUntilDelay(delay_enabled, delay) then
                // Event did not arrive before delay expired so trap WFE
                if target_el == EL3 && EL3SDDUndef() then
                    UNDEFINED;
                else
                    AArch64.WFxTrap(WFxType_WFE, target_el);
        else
            WaitForEvent();
    else
        WaitForEvent();
```

is changed to read:

```
Hint_WFE()
    ...
    (trap, target_el) = AArch64.CheckForWFxTrap(WFxType_WFE);
    if trap then
        if IsFeatureImplemented(FEAT_TWED) then
            // Determine if trap delay is enabled and delay amount
            boolean delay_enabled;
            integer delay;
            (delay_enabled, delay) = WFETrapDelay(target_el);
            if WaitForEventUntilDelay(delay_enabled, delay) then
                // Event arrived before the delay expired
                return;

        // Proceed with trapping
        if target_el == EL3 && EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.WFxTrap(WFxType_WFE, target_el);
    else
        WaitForEvent();
```

The equivalent change is made in the same section in the pseudocode function Hint_WFET().

2.67 D23548

In section **D24.2.56 “HCR_EL2, Hypervisor Configuration Register”**, in the field description of ‘TLOR, bit [35]’, the text that reads:

> Trap LOR registers. Traps Non-secure EL1 accesses to LORSA_EL1, LOREA_EL1, LORN_EL1, LORC_EL1, and LORID_EL1 registers to EL2.

is changed to read:

> Trap LOR registers. Traps Non-secure, and Realm EL1 accesses to LORSA_EL1, LOREA_EL1, LORN_EL1, LORC_EL1, and LORID_EL1 registers to EL2.

In the `0b1` field value, the text that reads:

> Non-secure EL1 accesses to the LOR registers are trapped to EL2.

is changed to read:

> Non-secure and Realm EL1 accesses to the LOR registers are trapped to EL2.

The equivalent change is made in section **D24.2.163 "SCR_EL3, Secure Configuration Register"**, in the field description of 'TLOR, bit [14]', where the text that reads:

> 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.

is changed to read:

> Trap LOR registers. Traps Non-Secure and Realm 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.

and in the `0b1` field value, the text that reads:

> EL1 and EL2 accesses to the LOR registers that are not **UNDEFINED** are trapped to EL3, unless it is trapped HCR_EL2.TLOR.

is changed to read:

> Non-secure and Realm EL1 and EL2 accesses to the LOR registers that are not **UNDEFINED** are trapped to EL3, unless it is trapped by HCR_EL2.TLOR.

The accessibility pseudocode is already correct.

2.68 D23549

In section **C7.2.418 "USUBW, USUBW2"**, the text that reads:

> This instruction subtracts each vector element of the second source SIMD&FP register from the corresponding vector element in the lower or upper half of the first source SIMD&FP register, places the result in a vector, and writes the vector to the SIMD&FP destination register. All the values in this instruction are unsigned integer values.
>
> The USUBW instruction extracts vector elements from the lower half of the first source register. The USUBW2 instruction extracts vector elements from the upper half of the first source register.

Is changed to read:

> This instruction subtracts each vector element in the lower or upper half of the second source SIMD&FP register from the corresponding vector element in the first source SIMD&FP register, places the result in a vector, and writes the vector to the SIMD&FP destination register. All the values in this instruction are unsigned integer values.
>
> The USUBW instruction extracts vector elements from the lower half of the second source register. The USUBW2 instruction extracts vector elements from the upper half of the second source register.

2.69 D23573

In section **J1.3.1.58 "Halt"**, the following function is added:

```
// BRBEDebugStateEntry()
// =====================
// Called to write Debug state entry branch record when BRBE is active.

BRBEDebugStateEntry(bits(64) source_address)
    if BranchRecordAllowed(PSTATE.EL) then
        constant bits(6) branch_type = '100001';
        bit ccu;
        bits(14) cc;
        (ccu, cc) = BranchEncCycleCount();
        constant bit lastfailed = (if IsFeatureImplemented(FEAT_TME) then
 BRBFCR_EL1.LASTFAILED else '0');
        constant bit transactional = (if IsFeatureImplemented(FEAT_TME) &&
 TSTATE.depth > 0 then '1' else '0');
        constant bits(2) el = PSTATE.EL;
        constant bit mispredict = '0';

        // Debug state is a prohibited region, therefore target_address=0
        UpdateBranchRecordBuffer(ccu, cc, lastfailed, transactional, branch_type,
 el, mispredict, '10', source_address, Zeros(64));
        BRBFCR_EL1.LASTFAILED = '0';

        PMUEvent(PMU_EVENT_BRB_FILTRATE);
```

The pseudocode function Halt() that reads:

```
// Halt()
// ======

Halt(bits(6) reason, boolean is_async, FaultRecord fault)
...
    if IsFeatureImplemented(FEAT_UINJ) then PSTATE.UINJ = '0';
    if IsFeatureImplemented(FEAT_ETE) then
        TraceDebugStateEntry(preferred_restart_address, source_instr_set);
...
```

is changed to read:

```
// Halt()
// ======

Halt(bits(6) reason, boolean is_async, FaultRecord fault)
...
    if IsFeatureImplemented(FEAT_UINJ) then PSTATE.UINJ = '0';
    if IsFeatureImplemented(FEAT_BRBE) then
        BRBEDebugStateEntry(preferred_restart_address);
    if IsFeatureImplemented(FEAT_ETE) then
        TraceDebugStateEntry(preferred_restart_address, source_instr_set);
...
```

2.70 D23577

In section **C5.3.31 "DC IVAC, Data or unified Cache line Invalidate by VA to PoC"**, under the heading 'Executing DC IVAC', the text that reads:

> If EL0 access is enabled, when executed at EL0, the instruction may generate a Permission fault, subject to the constraints described in MMU faults generated by cache maintenance operations.

is changed to read:

> This instruction requires write access permission to the VA, otherwise it generates a Permission fault, subject to the constraints described in MMU faults generated by cache maintenance operations.

The equivalent changes are made in the following sections:

- **C5.3.27 "DC IGDVAC, Invalidate of Data and Allocation Tags by VA to PoC"**.
- **C5.3.29 "DC IGVAC, Invalidate of Allocation Tags by VA to PoC"**.

In section **D8.15.3 "MMU faults generated by cache maintenance operations"**, the rule that reads:

> R_{JYWZL}
>
> If a DC IVAC does not have write permission to the location it invalidates, then a Permission fault can be generated.

is changed to read:

> R_{JYWZL}
>
> If a DC Invalidate by address instruction, for example, DC IVAC, DC IGVAC or DC IGDVAC does not have write permission to the location it invalidates, then a Permission fault can be generated.

2.71 D23585

In the section **D24.2.167 "SCTLR_EL1, System Control Register (EL1)"**, in the the field description for 'UCI, bit [26]', the text that reads:

> - DC CVAU, DC CIVAC, DC CVAC, and IC IVAU.
> - If FEAT_MTE is implemented, DC CIGVAC, DC CIGDVAC, DC CGVAC, and DC CGDVAC.
> - If FEAT_DPB is implemented, DC CVAP.
> - If FEAT_DPB and FEAT_MTE are implemented, DC CGVAP and DC CGDVAP.
> - If FEAT_DPB2 is implemented, DC CVADP.
> - If FEAT_DPB2 and FEAT_MTE are implemented, DC CGVADP and DC CGDVADP.

is changed to read:

- DC CVAU, DC CIVAC, DC CVAC, and IC IVAU.
- If FEAT_MTE is implemented, DC CIGVAC, DC CIGDVAC, DC CGVAC, and DC CGDVAC.
- If FEAT_DPB is implemented, DC CVAP.
- If FEAT_DPB and FEAT_MTE are implemented, DC CGVAP and DC CGDVAP.
- If FEAT_DPB2 is implemented, DC CVADP.
- If FEAT_DPB2 and FEAT_MTE are implemented, DC CGVADP and DC CGDVADP.
- If FEAT_OCCMO is implemented, DC CIVAOC, DC CIGDVAOC, DC CVAOC and DC CGDVAOC.

An equivalent change is made in section **“D24.2.168”SCTLR_EL2, System Control Register (EL2)“**.

2.72 D23598

In section **B2.8.2 “Alignment of data accesses”**, under the heading ‘Load-Exclusive/ Store-Exclusive and Atomic instructions’, the text that reads:

When instructions that load or store single or multiple registers are executed, if the value of SCTLR_ELx.A applicable to the current Exception level is 1, an Alignment fault is generated if any of the following apply:

- For Load-Exclusive Pair, Store-Exclusive Pair and CASP instructions, the address that is accessed is not aligned to the size of the pair.
- For all other instructions, the address that is accessed is not aligned to the size of the data element being accessed.

is changed to read:

When instructions that load or store single or multiple registers are executed, if the value of SCTLR_ELx.A applicable to the current Exception level is 1, an Alignment fault is generated if any of the following apply:

- For Load-Exclusive Pair, Store-Exclusive Pair and Atomic pair instructions, the address that is accessed is not aligned to the size of the pair.
- For all other instructions, the address that is accessed is not aligned to the size of the data element being accessed.

Furthermore, the text that reads:

If FEAT_LSE2 is not implemented and the value of SCTLR_ELx.A applicable to the current Exception level is 0, Load-Exclusive/Store-Exclusive and Atomic instructions, including those with acquire or release semantics, generate an alignment fault when the address being accessed is not aligned to the size of:

- The pair, for Load-Exclusive Pair, Store-Exclusive Pair, and CASP instructions.

- The data element being accessed for other Load-Exclusive/Store Exclusive, and Atomic instructions.

is changed to read:

If FEAT_LSE2 is not implemented and the value of SCTLR_ELx.A applicable to the current Exception level is 0, Load-Exclusive/Store-Exclusive and Atomic instructions, including those with acquire or release semantics, generate an alignment fault when the address being accessed is not aligned to the size of:

- The pair, for Load-Exclusive Pair, Store-Exclusive Pair, and Atomic pair instructions.
- The data element being accessed for other Load-Exclusive/Store-Exclusive, and Atomic instructions.