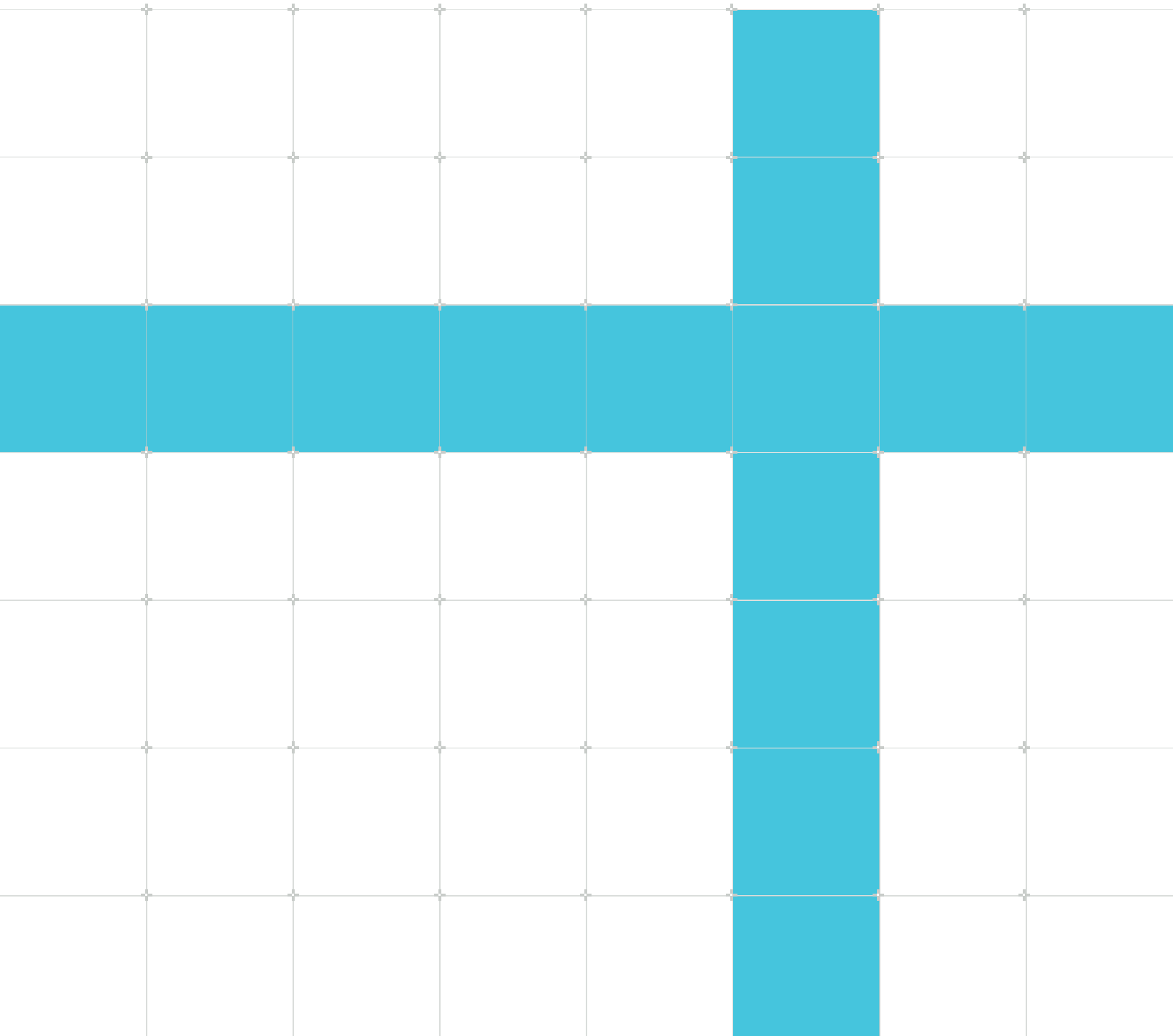
Arm® Architecture Reference Manual for A-profile architecture

Known issues in Issue L.a

Issue 05
102105_L.a_05_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-05	6 May 2025	Non-Confidential	Known Issues in Arm® Architecture Reference Manual, Issue L.a, as of 6 May 2025
L.a-04	1 April 2025	Non-Confidential	Known Issues in Arm® Architecture Reference Manual, Issue L.a, as of 1 April 2025
L.a-03	6 March 2025	Non-Confidential	Known Issues in Arm® Architecture Reference Manual, Issue L.a, as of 6 March 2025
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

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.

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....11
1.1 Conventions....11
1.2 Useful resources....12
1.3 Other information....13

2. Known issues....14
2.1 D17119....14
2.2 D17268....14
2.3 C17296....17
2.4 C17536....18
2.5 D18379....20
2.6 D18847....20
2.7 R19582....21
2.8 C19822....22
2.9 D20234....23
2.10 C20313....24
2.11 D20634....25
2.12 C20706....25
2.13 D20966....27
2.14 R21168....28
2.15 D21186....29
2.16 D21842....31
2.17 D22105....32
2.18 C22230....36
2.19 C22244....38
2.20 D22346....41
2.21 C22401....42
2.22 C22430....43
2.23 D22483....45
2.24 C22566....45
2.25 D22595....45
2.26 D22647....46

2.27 D22719 46
2.28 D22834 49
2.29 R22882 50
2.30 C22897 50
2.31 R22907 53
2.32 C22913 54
2.33 D22979 55
2.34 D22999 55
2.35 D23002 56
2.36 C23004 58
2.37 D23021 58
2.38 R23026 59
2.39 E23034 60
2.40 C23062 62
2.41 R23103 63
2.42 D23109 65
2.43 D23122 66
2.44 D23129 68
2.45 C23130 68
2.46 R23144 69
2.47 R23151 70
2.48 D23206 70
2.49 D23233 71
2.50 D23239 71
2.51 D23253 72
2.52 D23282 72
2.53 D23305 73
2.54 D23315 74
2.55 D23325 76
2.56 R23333 76
2.57 D23338 77
2.58 D23339 77
2.59 R23354 78
2.60 R23355 79
2.61 D23362 79
2.62 D23379 80

2.63 C23401....81
2.64 D23403....81
2.65 D23410....82
2.66 D23414....82
2.67 C23419....83
2.68 C23431....84
2.69 E23438....85
2.70 D23442....86
2.71 D23455....88
2.72 D23462....88
2.73 D23463....88
2.74 D23470....89
2.75 C23479....89
2.76 D23480....96
2.77 D23504....97
2.78 D23518....98
2.79 D23521....99
2.80 C23522....99
2.81 D23529....100
2.82 D23530....101
2.83 D23541....102
2.84 D23543....103
2.85 R23545....104
2.86 D23548....105
2.87 D23549....106
2.88 D23573....107
2.89 D23577....108
2.90 D23585....108
2.91 C23586....109
2.92 C23587....113
2.93 D23598....114
2.94 D23599....115
2.95 D23637....115
2.96 C23641....116
2.97 D23647....117
2.98 D23659....117

2.99 D23661 119
2.100 R23665 120
2.101 D23670 120
2.102 C23678 121
2.103 D23680 122
2.104 D23683 123
2.105 D23684 123
2.106 D23688 124
2.107 C23700 124
2.108 C23701 126
2.109 D23712 127
2.110 D23713 127
2.111 D23734 130
2.112 D23741 130
2.113 D23744 131
2.114 C23753 132
2.115 C23766 133
2.116 D23768 134
2.117 D23772 134
2.118 C23775 142
2.119 D23778 144
2.120 D23794 144
2.121 D23796 145
2.122 D23797 145
2.123 D23807 147
2.124 C23808 147
2.125 D23812 148
2.126 R23867 148
2.127 D23823 149
2.128 C23842 149
2.129 C23844 151
2.130 C23845 152
2.131 D23854 153
2.132 C23859 153
2.133 C23866 154
2.134 D23869 154

2.135 R23909 155
2.136 R23917 155
2.137 D23919 156
2.138 D23933 156
2.139 D23934 160
2.140 C23935 160
2.141 D23936 161
2.142 D23947 162
2.143 D23962 162
2.144 C23967 162
2.145 D23988 165
2.146 D23995 165

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.

Arm documents are available on developer.arm.com/documentation.

Confidential documents are only available to licensees, when logged in. Each document link in the tables below provides direct access to the online version of the document.

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 D17268

In section D24.2.182 “TCR_EL1, Translation Control Register (EL1)”, in the description for field “IPS, bits [34:32]”, the text that reads:

> The value `0b110` represents the following output address sizes:
>
> - For the 64KB translation granule size, if FEAT_LPA is implemented, the value `0b110` represents 52 bits.
> - For the 4KB and 16KB granule sizes, only if both FEAT_LPA and the effective value of TCR_EL1.DS is `0b1`, the value `0b110` represents 52 bits.
> - Otherwise, the value `0b110` behaves as the `0b101` value and represents 48 bits.
>
> If the value of ID_AA64MMFR0_EL1.PARange is `0b0111`, and the value of this field is not `0b111` or a value treated as `0b111`, then bits[55:52] of every translation table base address are `0b0000` for the stage of translation controlled by TCR_EL1.

is changed to read:

> The values `0b110` and `0b111` represent different output address sizes depending on implementation choices and translation configuration.
>
> The following table captures the output address size represented by the value `0b110`:

Descriptor Format	ID_AA64MMFR0_EL1.PARange	Translation Granule	TCR_EL1.DS	Represented OA size
Any	0b0101	Any	Any	48 bits, 256TB
VMSAv8-64	0b011x	4KB or 16KB	0	48 bits, 256TB
VMSAv8-64	0b011x	4KB or 16KB	1	52 bits, 4PB
VMSAv8-64	0b011x	64KB	N/A	52 bits, 4PB
VMSAv9-128	0b011x	Any	N/A	52 bits, 4PB

If 52-bit PA is supported, and the translation table descriptors cannot express an OA larger than 48 bits, then bits[51:48] of every translation table base address are treated as 0b0000 for the stage of translation controlled by TCR_EL1.

The following table captures the output address size represented by the value 0b111:

Descriptor Format	ID_AA64MMFR0_EL1.PARange	Translation Granule	TCR_EL1.DS	Represented OA size
Any	0b0110	Any	Any	OA size represented by 0b110
VMSAv8-64	0b0111	Any	Any	OA size represented by 0b110
VMSAv9-128	0b0111	Any	N/A	56 bits, 64PB

If 56-bit PA is supported, and the translation table descriptors cannot express an OA larger than 52 bits, then bits[55:52] of every translation table base address are treated as 0b0000 for the stage of translation controlled by TCR_EL1.

If the output address size represented by this field is larger than the supported PA size expressed in ID_AA64MMFR0_EL1.PARange, then the output address size is treated as being the same as the supported PA size. Arm strongly recommends that software avoids configuring this field to a value representing an output address size larger than the supported PA size.

The equivalent changes are made in the following sections:

- D24.2.183 “TCR_EL2, Translation Control Register (EL2)”
- D24.2.184 “TCR_EL3, Translation Control Register (EL3)”
- D24.2.210 “VTCR_EL2, Virtualization Translation Control Register”

In section D24.2.195 “TTBR0_EL1, Translation Table Base Register 0 (EL1)”, under the heading “When FEAT_D128 is not implemented or TCR2_EL1.D128 == 0:”, in field “BADDR[47:1], bits [47:1]”, the text that reads:

The BADDR field represents a 52-bit address if one of the following applies:

- FEAT_LPA is implemented, the 64KB granule size is in use, and the value of TCR_EL1.IPS is 0b110.
- FEAT_LPA2 is implemented, the 4KB or 16KB granule size is in use, and the Effective value of TCR_EL1.DS is 1.
- FEAT_D128 is implemented, 56-bit PAs are supported, the 64KB granule size is in use, and the value of TCR2_EL1.D128 is 0.

is changed to read:

> The BADDR field represents a 52-bit address if the value of TCR_EL1.IPS represents an output address size of 52 bits.
>
> If 52-bit PA is supported, and BADDR field does not represent a 52-bit address, then bits[51:48] of the base address are treated as `0b0000`.

Additionally, the note in the same field that reads:

> For the 64KB granule, if FEAT_LPA is not implemented, and the value of TCR_EL1.IPS is `0b110`, one the following **IMPLEMENTATION DEFINED** behaviors occur:

is changed to read:

> For the 64KB granule, if 52-bit PA is not supported, and the value of TCR_EL1.IPS is `0b110` or `0b111`, one of the following **IMPLEMENTATION DEFINED** behaviors occur:

In section D8.1.7 “Output address size configuration”, under the rule I_{HGDWP}, the text that reads:

> If a translation stage is enabled and the translation stage OA size is larger than the implemented PA size, then an Address size fault is generated at the lookup level in the translation stage that generated the OA.

is changed to read:

> If a translation stage is enabled and the translation stage OA is larger than the implemented PA size, then an Address size fault is generated at the lookup level in the translation stage that generated the OA.

Under the rule R_{BZHGM}, the text that reads:

> If all of the following apply, then a stage 2 Translation fault is generated:
> - Two address translation stages are used.
> - Stage 2 address translation is enabled.
> - The stage 1 OA size does not generate a stage 1 Address size fault.
> - The stage 1 OA size is larger than the specified stage 2 translation IA size.

is changed to read:

> If all of the following apply, then a stage 2 Translation fault is generated:
> - Two address translation stages are used.
> - Stage 2 address translation is enabled.
> - The stage 1 OA does not generate a stage 1 Address size fault.
> - The stage 1 OA is larger than the specified stage 2 translation IA size.

2.3 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.

2.4 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.

2.5 D18379

In section D1.3.4.2 “Illegal exception returns from AArch64 state”, under the rule R_{VWJHB}, the text that reads:

On an illegal exception return from an Exception level, ELx, all of the following occur:

- ...
- All of the following are **UNKNOWN**:
  - ...
  - If FEAT_NMI is implemented, then PSTATE.ALLINT.
  - ...
- ...

is changed to read:

On an illegal exception return from an Exception level, ELx, all of the following occur:

- ...
- All of the following are **UNKNOWN**:
  - ...
- If FEAT_NMI is implemented, then PSTATE.ALLINT is set to the associated value in SPSR_ELx.
- ...

2.6 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);
```

2.7 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.

2.8 C19822

In sections D24.2.40 “ESR_EL1, Exception Syndrome Register (EL1)”, D24.2.41 “ESR_EL2, Exception Syndrome Register (EL2)” and D24.2.42 “ESR_EL3, Exception Syndrome Register (EL3)”, under the heading “ISS encoding for an exception from the Memory Copy and Memory Set instructions”, the text that reads:

> WrongOption, bit [17]
>
> Algorithm option.

0b0	**WrongOption is false.**
0b1	WrongOption is true.

> The reset behavior of this field is:
>
> - On a Warm reset, this field resets to an architecturally **UNKNOWN** value.
>
> OptionA, bit [16]
>
> Algorithm type indicated by the PSTATE.C bit.

0b0	**OptionB indicated by PSTATE.C is 0.**
0b1	OptionA indicated by PSTATE.C is 1.

> The reset behavior of this field is:
>
> - On a Warm reset, this field resets to an architecturally **UNKNOWN** value.

is changed to read:

> FormatOption, bit [17:16]
>
> Reports the Option used to encode the initial Xs, Xd and Xn register values provided to the instruction that generated the exception.

FormatOption	Meaning
0b00	Option B
0b01	Option A
0b10	Option A
0b11	Option B

> For more information, see D1.3.5.7 “Memory Copy and Memory Set exceptions”.
>
> The reset behavior of this field is:
>
> - On a Warm reset, this field resets to an architecturally **UNKNOWN** value.
>
> Note:

This field was previously presented as two separate bits, WrongOption (bit [17]) and OptionA (bit [16]), which were already expected to be used together and not as individual bits.

In section D1.3.5.7 Memory Copy and Memory Set exceptions, under rule I_{CNTMJ}, the code that reads:

```
if OptionA && WrongOption || OptionB && !WrongOption then
    // format is from option B
    ...
```

is changed to read:

```
if FormatOption == 0b00 || FormatOption == 0b11 then
    // format is from option B
    ...
```

In the same section, under rule I_{MWFQH}, the code that reads:

```
if !OptionA && WrongOption || (OptionA && !WrongOption) then
    // format is from option A
    ...
```

is changed to read:

```
if FormatOption == 0b01 || FormatOption == 0b10 then
    // format is from option A
    ...
```

2.9 D20234

In section D14.2 “The PMU event number space and common events”, table D14-2 “Allocation of the PMU event number space”, the text that reads:

Event numbers	Allocation
...	...
0x8000 - 0x80FF	Common architectural and microarchitectural events.
0x8100 - 0x81FF	From Armv8.6, common architectural and microarchitectural events.
0x8200 - 0xC0BF	Reserved.
...	...

is changed to read:

Event numbers	Allocation
...	...
0x8000 - 0x8FFF	Common architectural and microarchitectural events.
0x9000 - 0xC0BF	Reserved.

Event numbers	Allocation
...	...

2.10 C20313

In section D8.10.3 “PAC Instructions”, under the rule R_{TMSKZ}, the text that reads:

> An instruction that inserts a PAC into the PAC field of a 64-bit general-purpose register, operates on the PAC field in one of the following ways:
>
> - If FEAT_PAuth2 is not implemented, then the pointer authentication mechanism replaces the PAC field with the PAC.
> - If FEAT_PAuth2 is implemented, then the pointer authentication mechanism exclusive-ORs the bits in the PAC field with the PAC.
> - If FEAT_EPAC is implemented, then when performing a PAC operation on a non-canonical address, the pointer authentication mechanism sets the PAC field to 0.

is changed to read:

> An instruction that inserts a PAC into the PAC field of a 64-bit general-purpose register, operates on the PAC field in one of the following ways:
>
> - If FEAT_PAuth2 is not implemented, then the pointer authentication mechanism replaces the PAC field with the PAC.
> - If FEAT_PAuth2 is implemented, then the pointer authentication mechanism exclusive-ORs the bits in the PAC field with the PAC.

Additionally, the following rule is added:

> R_{X0001}
>
> When FEAT_PAuth2 is not implemented and when performing a PAC operation on a non-canonical address then the PAC is modified to make it incorrect, and one of the following applies:
>
> - If FEAT_EPAC is implemented, then the pointer authentication mechanism sets the PAC field to 0.
> - If FEAT_EPAC is not implemented, then the pointer authentication mechanism inverts one of the bits of the PAC as follows:
>   - When address tagging is not used, bit [62] is inverted.
>   - Otherwise, bit [54] is inverted.

2.11 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.

2.12 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.

2.13 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.

2.14 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.

2.15 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;
...
```

2.16 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”.

2.17 D22105

In section D24.2.56 “HCR_EL2, Hypervisor Configuration Register”, in the field description of ‘IMO, bit [4]’, the text that reads:

IMO	Meaning
0b0	When executing at Exception levels below EL2, and EL2 is enabled in the current Security state: • When the value of HCR_EL2.TGE is 0, Physical IRQ interrupts are not taken to EL2. • When the value of HCR_EL2.TGE is 1, Physical IRQ interrupts are taken to EL2 unless they are routed to EL3. Virtual IRQ interrupts are disabled.
0b1	When executing at any Exception level, and EL2 is enabled in the current Security state: • Physical IRQ interrupts are taken to EL2, unless they are routed to EL3. • When the value of HCR_EL2.TGE is 0, then Virtual IRQ interrupts are enabled.

> If EL2 is enabled in the current Security state and the value of HCR_EL2.TGE is 1:
>
> • Regardless of the value of the FMO bit, physical FIQ Interrupts target EL2 unless they are routed to EL3.
> • When the Effective value of HCR_EL2.E2H is not 1, the Effective value of this field is 1.
> • When the Effective value of HCR_EL2.E2H is 1, the Effective value of this field is 0.
>
> For more information, see ‘Asynchronous exception routing’.

is changed to read:

IMO	Meaning
0b0	When executing at Exception levels below EL3 and the Effective value of HCR_EL2.TGE is 0: • Physical IRQ interrupts are not taken to EL2. • Virtual IRQ interrupts are disabled.
0b1	When executing at Exception levels below EL3, EL2 is enabled in the current Security state, and the Effective value of HCR_EL2.TGE is 0: • Physical IRQ exceptions are taken to EL2, unless they are routed to EL3. • Virtual IRQ interrupts are enabled.

> When executing at EL3, the Effective value of HCR_EL2.IMO has no impact on the behavior of the PE.
>
> When executing at Exception levels below EL3, and EL2 is not enabled in the current Security state, the Effective value of HCR_EL2.IMO is 0.
>
> When the Effective value of HCR_EL2.TGE is 1, regardless of the value of the IMO bit, all of the following are true:
>
> • Physical IRQ Interrupts target EL2 unless they are routed to EL3.
> • Virtual IRQ interrupts are disabled.

> When executing at EL2 and the Effective value of HCR_EL2.{E2H, TGE} is {1, 0}, it is **IMPLEMENTATION DEFINED** whether the Effective value of HCR_EL2.IMO is 1 or the value programmed.
>
> For more information, see 'Asynchronous exception types'.

In the same section, in the field description of 'FMO, bit [3]', the text that reads:

FMO	Meaning
0b0	When executing at Exception levels below EL2, and EL2 is enabled in the current Security state: • When the value of HCR_EL2.TGE is 0, Physical FIQ interrupts are not taken to EL2. • When the value of HCR_EL2.TGE is 1, Physical FIQ interrupts are taken to EL2 unless they are routed to EL3. • Virtual FIQ interrupts are disabled.
0b1	When executing at any Exception level, and EL2 is enabled in the current Security state: • Physical FIQ interrupts are taken to EL2, unless they are routed to EL3. • When HCR_EL2.TGE is 0, then Virtual FIQ interrupts are enabled.

> If EL2 is enabled in the current Security state and the value of HCR_EL2.TGE is 1:
>
> - Regardless of the value of the FMO bit, physical FIQ Interrupts target EL2 unless they are routed to EL3.
> - When the Effective value of HCR_EL2.E2H is not 1, the Effective value of this field is 1.
> - When the Effective value of HCR_EL2.E2H is 1, the Effective value of this field is 0.
>
> For more information, see 'Asynchronous exception routing'.

is changed to read:

FMO	Meaning
0b0	When executing at Exception levels below EL3 and the Effective value of HCR_EL2.TGE is 0: • Physical FIQ interrupts are not taken to EL2. • Virtual FIQ interrupts are disabled.
0b1	When executing at Exception levels below EL3, EL2 is enabled in the current Security state, and the Effective value of HCR_EL2.TGE is 0: • Physical FIQ exceptions are taken to EL2, unless they are routed to EL3. • Virtual FIQ interrupts are enabled.

> When executing at EL3, the Effective value of HCR_EL2.FMO has no impact on the behavior of the PE.

> When executing at Exception levels below EL3, and EL2 is not enabled in the current Security state, the Effective value of HCR_EL2.FMO is 0.

> When the Effective value of HCR_EL2.TGE is 1, regardless of the value of the FMO bit, all of the following are true:
>
> - Physical FIQ Interrupts target EL2 unless they are routed to EL3.

- Virtual FIQ interrupts are disabled.

When executing at EL2 and the Effective value of HCR_EL2.{E2H, TGE} is {1, 0}, it is **IMPLEMENTATION DEFINED** whether the Effective value of HCR_EL2.FMO is 1 or the value programmed.

For more information, see 'Asynchronous exception types'.

In the same section, in the field description of 'AMO, bit [5]', the text that reads:

AMO	Meaning
0b0	Physical SError exceptions are unaffected by this mechanism. That is, physical SError exceptions are not taken to EL2 unless routed to EL2 by another control. Virtual SError exceptions are not enabled by this mechanism.
0b1	When executing at any Exception level, and EL2 is enabled in the current Security state, all of the following apply: • Physical SError exceptions are taken to EL2, unless they are routed to EL3. • If FEAT_E3DSE is implemented then delegated SError exceptions enabled by SCR_EL3.DSE are taken to EL2. • If HCR_EL2.TGE is 0 then virtual SError exceptions are enabled.

If EL2 is enabled in the current Security state and the value of HCR_EL2.TGE is 1:

- Regardless of the value of HCR_EL2.AMO, physical SError exceptions target EL2 unless they are routed to EL3.
- If FEAT_E3DSE is implemented then regardless of the value of HCR_EL2.AMO, delegated SError exceptions target EL2.
- When the Effective value of HCR_EL2.E2H is not 1, the Effective value of this field is 1.
- When the Effective value of HCR_EL2.E2H is 1, the Effective value of this field is 0.

For more information, see 'Asynchronous exception routing'.

is changed to read:

AMO	Meaning
0b0	When executing at Exception levels below EL3 and the Effective value of HCR_EL2.TGE is 0: • The routing of physical SError exceptions is unaffected by this mechanism. • If FEAT_E3DSE is implemented then delegated SError exceptions enabled by SCR_EL3.DSE are not taken to EL2. • Virtual SError exceptions are not enabled by this mechanism.
0b1	When executing at Exception levels below EL3, EL2 is enabled in the current Security state, and the Effective value of HCR_EL2.TGE is 0: • Physical SError exceptions are taken to EL2, unless they are routed to EL3. • If FEAT_E3DSE is implemented then delegated SError exceptions enabled by SCR_EL3.DSE are taken to EL2. • Virtual SError exceptions are enabled.

When executing at EL3, the value of HCR_EL2.AMO has no impact on the behavior of the PE.

When executing at Exception levels below EL3, and EL2 is not enabled in the current Security state, the Effective value of HCR_EL2.AMO is 0.

When the Effective value of HCR_EL2.TGE is 1, regardless of the value of the AMO bit, all of the following are true:

- Regardless of the value of HCR_EL2.AMO, physical SError exceptions target EL2 unless they are routed to EL3.
- If FEAT_E3DSE is implemented then regardless of the value of HCR_EL2.AMO, delegated SError exceptions target EL2.
- Virtual SError exceptions are disabled.

When executing at EL2 and the Effective value of HCR_EL2.{E2H, TGE} is {1, 0}, it is **IMPLEMENTATION DEFINED** whether the Effective value of HCR_EL2.AMO is 1 or the value programmed.

For more information, see 'Asynchronous exception types'.

In section D24.2.165 "SCTLR2_EL2, System Control Register (EL2)", in the field description of 'NMEA, bit [2]', the text that reads:

NMEA	Meaning
0b0	SError exceptions are not taken at EL2 if PSTATE.A == 1, unless routed to a higher Exception level.
0b1	SError exceptions are taken at EL2 regardless of the value of PSTATE.A, unless routed to a higher Exception level.

is changed to read:

NMEA	Meaning
0b0	The masking of SError exceptions at EL2 is unaffected by this mechanism.
0b1	SError exceptions are taken at EL2 regardless of whether they are masked by any mechanism, unless they are routed to EL3.

For more information, see 'Asynchronous exception types'.

In section D1.3.6.1 "Virtual interrupts", the rule R_{BCXJB} that reads:

If HCR_EL2.TGE is 0, setting an HCR_EL2.{FMO, IMO} routing control bit to 1 enables the corresponding virtual interrupt. If HCR_EL2.TGE is 1, all virtual interrupts are disabled and the Effective values of HCR_EL2.{FMO, IMO} are 0.

is changed to read:

If HCR_EL2.TGE is 0, setting an HCR_EL2.{FMO, IMO} routing control bit to 1 enables the corresponding virtual interrupt.

If HCR_EL2.TGE is 1, virtual interrupts are disabled.

2.18 C22230

In section D7.5.9.2 “The data cache maintenance instruction (DC)”, the following statement is added:

> For all the following DC instructions, it is **IMPLEMENTATION DEFINED** whether the instruction is treated as a **NOP**:
>
> - A DC to the PoU and the LoUIS and LoUU are 0.
> - A DC to the PoC and the LoC is 0.
> - A DC to the PoP and the memory system does not support the Point of Persistence, and the LoC is 0.
> - A DC to the PoDP and the memory system does not support the Point of Deep Persistence, and does not support the Point of Persistence, and the LoC is 0.
>
> If a DC instruction that operates by VA is treated as a **NOP**, it does not perform translation and all the following apply:
>
> - No MMU faults can be generated.
> - If HW update of AF is enabled, AF is not required to be set.
>
> If a DC instruction is treated as a **NOP**, then it is **IMPLEMENTATION DEFINED** whether the execution of the instruction can be trapped.

In section D7.5.9.1 “The instruction cache maintenance instruction (IC)”, the following statement is added:

> If CTR_EL0.DIC is 1, then it is **IMPLEMENTATION DEFINED** whether each of the IC instructions is treated as a **NOP**.
>
> If an IC instruction that operates by VA is treated as a **NOP**, it does not perform translation and all the following apply:
>
> - No MMU faults can be generated.
> - If HW update of AF is enabled, AF is not required to be set.
>
> If an IC instruction is treated as a **NOP**, then it is **IMPLEMENTATION DEFINED** whether the execution of the instruction can be trapped.

In section D24.2.56 “HCR_EL2, Hypervisor Configuration Register”, in the description of the field ‘TPCP’, the table that reads:

TPCP	Meaning
`0b0`	This control does not cause any instructions to be trapped.
`0b1`	Execution of the specified instructions is trapped to EL2, when EL2 is enabled in the current Security state.

> If the Point of Coherency is before any level of data cache, it is **IMPLEMENTATION DEFINED** whether the execution of any data or unified cache clean, invalidate, or clean and invalidate instruction that operates by VA to the point of coherency can be trapped when the value of this control is 1.

is changed to read:

TPCP	Meaning
0b0	This control does not cause any instructions to be trapped.
0b1	For each of the specified instructions, if the execution of the instruction can be trapped, it is trapped to EL2, when EL2 is enabled in the current Security state.

In section D8.15.3 “MMU faults generated by cache maintenance operations”, the rule R_{BDCCY} that reads:

> If the Point of Coherency is before any cache level, then it is **IMPLEMENTATION DEFINED** whether the execution of any cache clean instruction, or clean and invalidate instruction, that operates by VA to the Point of Coherency, can generate any MMU fault.

is changed to read:

> Cache maintenance instructions treated as a **NOP** cannot generate MMU faults.

Additionally, the rule R_{MZTNR} is removed:

> If the Point of Unification is before any data cache level, then it is **IMPLEMENTATION DEFINED** whether the execution of any data or unified cache clean instruction, or clean and invalidate instruction, that operates by VA to the Point of Unification, can generate any MMU fault.

The rule R_{DNZYL} is also removed:

> It is **IMPLEMENTATION DEFINED** whether the execution of any instruction cache invalidate by VA to the Point of Unification instruction can generate any MMU fault.

The rule R_{TRTWX} in the same section that reads:

> R_{TRTWX}
>
> It is **IMPLEMENTATION DEFINED** whether execution of any cache maintenance instruction by VA can generate an Access flag fault.

is changed to read:

> R_{TRTWX}
>
> It is **IMPLEMENTATION DEFINED** whether execution of any cache maintenance instruction by VA that is not treated as a **NOP** can generate an Access flag fault.

In section A2.2.9 “The Armv8.8 architecture extension”, the statement that reads:

> FEAT_CMOW, Control for cache maintenance permission

> FEAT_CMOW introduces support for cache maintenance instructions that controls whether:
>
> - Cache maintenance instructions executed at EL0 require stage 1 read and write permission to prevent the instructions from generating a Permission fault.
> - Cache maintenance instructions executed at EL1 or EL0 require stage 2 read and write permission to prevent the instructions from generating a Permission fault.

is changed to read:

> FEAT_CMOW, Control for cache maintenance permission
>
> FEAT_CMOW introduces support for controlling the required permissions for some cache maintenance instructions that are subject to translation such that:
>
> - Stage 1 can be configured to generate a Permission fault if write permission is not present for cache maintenance instructions executed at EL0.
> - Stage 2 can be configured to generate a Permission fault if write permission is not present for cache maintenance instructions executed at EL1 or EL0.

In section D2.9.6.2 “Watchpoint behavior on accesses by the DC IVAC instruction and the DC ZVA, DC GVA, and DC GZVA instructions”, the text that reads:

> If the Point of Coherency is before any level of cache, it is **IMPLEMENTATION DEFINED** whether a DC IVAC instruction can generate a Watchpoint exception. Otherwise, DC IVAC operations can generate Watchpoint exceptions.

is changed to read:

> If DC IVAC is not treated as a **NOP**, it can generate Watchpoint exceptions.

2.19 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:

2.20 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".

2.21 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 changed to 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	-

Bit position	Field	Condition
[55]	IGNORED	-

are changed to read:

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

2.22 C22430

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

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

is changed to read:

> R_{RBFJJV}
>
> 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 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 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.

2.23 D22483

In section D18.2.1 “Address packet”, under the heading ‘Address packet payload (data access physical address)’, the text that reads:

> NSE, byte 7 bit [7]
>
> ...
>
> NS, byte 7 bit [4]
>
> ...

is changed to read:

> NS, byte 7 bit [7]
>
> ...
>
> NSE, byte 7 bit [4]
>
> ...

2.24 C22566

In section A1.5.9.4 “Underflow exceptions”, the text that reads:

> If the result of a floating-point operation is a denormalized number that is flushed to zero, then the Underflow floating-point exception is not generated.

is changed to read:

> If the result of a floating-point operation is a denormalized number that is flushed to zero, then the Underflow floating-point exception is generated, however this floating-point exception is not trapped regardless of the value of FPCR.UFE.

2.25 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.

2.26 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.

2.27 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:

- `0b01`xx 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`.
  - ...
- `0b10`xx 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:

- `0b01`xx The entry comes from a 4KB translation granule. The level of walk for the leaf level 0bxx is encoded as:
  - `0b00` : Level 0.
  - ...
- `0b10`xx 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.

2.28 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.

2.29 R22882

In section B2.12.5 “Load-Exclusive and Store-Exclusive instruction usage restrictions”, the text that reads:

> LoadExcl/StoreExcl loops are guaranteed to make forward progress only if, for any LoadExcl/StoreExcl loop within a single thread of execution, the software meets all of the following conditions:
>
> - ...

is changed to read:

> If FEAT_LSE is not implemented, LoadExcl/StoreExcl loops are guaranteed to make forward progress only if, for any LoadExcl/StoreExcl loop within a single thread of execution, the software meets all of the following conditions.
>
> If FEAT_LSE is implemented, software can maximize the likelihood that LoadExcl/StoreExcl loops make forward progress, for any LoadExcl/StoreExcl loop within a single thread of execution, by meeting all of the following conditions:
>
> - ...

2.30 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]'.

2.31 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.

2.32 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.

2.33 D22979

In section D8.5.2.2 “Implications of enabling the dirty state management mechanism”, under the rule R_{RKMHW}, the text that reads:

> For the following instructions that require write permission, if the address specified in the instruction is translated by a writable-clean descriptor, then the descriptor is considered to grant write access and the hardware does not update the Dirty state of that descriptor:
>
> - Data cache invalidation instruction, DC IVAC.
> - Data cache and instruction cache maintenance instructions that are affected by FEAT_CMOW.
> - Address translation instructions, AT S12E0W, AT S12E1W, AT S1E0W, AT S1E1W, AT S1E2W, AT S1E3W.

is changed to read:

> For the following instructions that require write permission, if the address specified in the instruction is translated by a writable-clean descriptor and hardware management of Dirty state is enabled for the corresponding stage of translation, then the descriptor is considered to grant write access and the hardware does not update the Dirty state of that descriptor:
>
> - Data cache invalidation instruction, DC IVAC.
> - Data cache and instruction cache maintenance instructions that are affected by FEAT_CMOW.
> - Address translation instructions, AT S12E0W, AT S12E1W, AT S1E0W, AT S1E1W, AT S1E2W, AT S1E3W.

2.34 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”.

2.35 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

Event	Instructions added
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
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)

Event	Instructions
ASE_SVE_INT_DOT_SPEC	SUDOT (by element)

2.36 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.

2.37 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’.

2.38 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.

2.39 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**.

2.40 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.

2.41 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.

2.42 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]'.

2.43 D23122

In section D14.3.2 "Common microarchitectural events", in the event description of '`0x0015`, L1D_CACHE_WB, Level 1 data cache write-back' the text that reads:

> The counter counts each write-back of data from the Level 1 data or unified cache to outside of the Level 1. For example:
>
> - A write-back of a dirty cache line to a Level 2 cache or memory.
> - A write-back of a recently fetched cache line that has not been allocated to the Level 1 data cache.

is changed to read:

> The counter counts each write-back of data from the Level 1 data or unified cache to outside of the Level 1. For example:
>
> - A write-back of a cache line to a Level 2 cache or memory.
> - A write-back of a recently fetched cache line that has not been allocated to the Level 1 data cache.

In the same event description, the text that reads:

> A write-back is attributable to the agent that generated the request that caused the write-back. This might not be the same agent that caused the data being written back to be allocated into the cache.
>
> An Unattributable write-back event occurs when a requestor outside of the PE makes a coherency request that results in write-back. If the cache is shared, then an Unattributable write-back event is not counted. If the cache is not shared, then the event is counted.

is changed to read:

> A write-back is attributable to one of the following, and it is **IMPLEMENTATION DEFINED** which:
>
> - The PE or other agent that generated the request that caused the write-back. This might not be the same agent that caused the data to be allocated into the cache.

- If the cache is shared, the PE or other agent that caused the data to be allocated into the cache.
- If the cache is shared, the PE or other agent that last accessed the data when it was in the cache. If the data has not been accessed since being allocated, then the PE or other agent that caused the data to be allocated into the cache.

If the cache is shared, then the write-back event is counted only if it is attributable to the PE counting the event.

If the cache is not shared, then the event is counted by the PE that the cache is attached to, even if the write-back is not attributable to that PE. For example, a requestor outside of the PE made a coherency request that resulted in write-back.

In the same section, in the event description of '`0x0046`, L1D_CACHE_WB_VICTIM, Level 1 data cache write-back, victim', the text that reads:

> The counter counts each write-back counted by L1D_CACHE_WB that occurs because the line is allocated for an access made by the PE.

is changed to read:

> The counter counts each write-back counted by L1D_CACHE_WB that occurs because of a capacity eviction due to a line being allocated into the cache.
>
> It is **IMPLEMENTATION DEFINED** whether this includes capacity evictions due to an agent other the PE counting the event. For example, a stashing request from outside of the PE, or an allocation due to another PE that shares the cache.
>
> In all cases, the event is counted only if the eviction is also an L1D_CACHE_WB event. This means that, if the cache is shared, the eviction is counted only if it is attributable to the PE counting the event, as defined by L1D_CACHE_WB.

In the same section, in the event description of '`0x0047`, L1D_CACHE_WB_CLEAN, Level 1 data cache write-back, cleaning and coherency', the text that reads:

> The counter counts each write-back counted by L1D_CACHE_WB that occurs because of a coherency operation made by another PE or, optionally, the execution of a cache maintenance instruction.
>
> Whether write-backs that are caused by the execution of a cache maintenance instruction are counted is **IMPLEMENTATION DEFINED**.
>
> Note:
>
> The transfer of a dirty cache line from the Level 1 data cache of this PE to the data cache of another PE due to a hardware coherency operation is not counted unless the dirty cache line is also written back to a Level 2 cache or memory. If a coherency request from a requestor outside of the PE results in a write-back, it is an Unattributable event.

is changed to read:

> The counter counts each write-back counted by L1D_CACHE_WB that occurs because of a coherency operation made by another PE or, optionally, the execution of a cache maintenance instruction.
>
> It is **IMPLEMENTATION DEFINED** whether the transfer of a dirty cache line from the Level 1 data cache of this PE to the data cache of another PE due to a hardware coherency operation is counted when the dirty cache line is not also written back to a Level 2 cache or memory.
>
> In all cases, the event is counted only if the write-back is also an L1D_CACHE_WB event. This means that, if the cache is shared, the write-back is counted only if it is attributable to the PE counting the event, as defined by L1D_CACHE_WB.

Similar updates are made for the following events:

- L2D_CACHE_WB (`0x0018`)
- L2D_CACHE_VICTIM (`0x0056`)
- L2D_CACHE_CLEAN (`0x0057`)
- L3D_CACHE_WB (`0x002C`)
- L3D_CACHE_VICTIM (`0x00A6`)
- L3D_CACHE_CLEAN (`0x00A7`)

2.44 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.

2.45 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.

2.46 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.

2.47 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.
...	...	...

2.48 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.

2.49 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.

2.50 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

2.51 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]’.

2.52 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)”.

2.53 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

2.54 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.

2.55 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**.

2.56 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.

2.57 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().

2.58 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.

2.59 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.

2.60 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.

2.61 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)”.

2.62 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.

2.63 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.

2.64 D23403

In section D23.5.12 “PMEVTYPER<n>_EL0, Performance Monitors Event Type Registers, n = 0 - 30”, the following text is added to the description of field “MT, bit [25]”:

Unless otherwise stated:

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

For the stall events, the stall condition means the applicable condition described by the STALL, STALL_FRONTEND, or STALL_BACKEND event. The second condition is any condition in addition to this. For example, for the STALL_FRONTEND_L1I event, the stall condition is STALL_FRONTEND, and the second condition is when there is a demand instruction miss in the first level of instruction cache. For the STALL, STALL_FRONTEND, and STALL_BACKEND events themselves, the second condition is the null TRUE condition.

See also Cycle event counting.

Section D13.6 “Multithreaded implementations” is renamed to “Cycle event counting in multithreaded implementations” and the equivalent text is added.

More specific information is added to the affected STALL events.

2.65 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**.

2.66 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”.

2.67 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:

IPS	Meaning	Applies when
0b111	56 bits, 64PB	When FEAT_D128 is implemented

is changed to read:

IPS	Meaning	Applies when
0b111	56 bits, 64PB	

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”.

2.68 C23431

In section J1.1 “Pseudocode for AArch64 operation”, the pseudocode for function “AArch64.MemAtomicRCW()” that reads:

```
(bits(4), bits(size)) MemAtomicRCW(bits(64) address, bits(size) cmpoperand,
 bits(size) operand,
 AccessDescriptor accdesc_in)
 ...
```

```
 return (nzcv, retvalue);
```

is changed to read:

```
(bits(4), bits(size)) MemAtomicRCW(bits(64) address, bits(size) cmpoperand,
 bits(size) operand,
 AccessDescriptor accdesc_in)
 ...
 if SPESampleInFlight then
   constant boolean is_load = TRUE;
   SPESampleLoadStore(is_load, accdesc, memaddrdesc);
 return (nzcv, retvalue);
```

Similar changes are made in the following functions:

- AArch64.MemLoad64B().
- AArch64.MemStore64BWithRet().
- AArch64.MemStore64B().

2.69 E23438

In section B2.5 “Restrictions on the effects of speculation”, the following bullet list is added:

> - A load will not speculatively read data from any store that appears in program order after the load itself.

In section B2.6.5 “Speculative Store Bypass Barrier (SSBB)”, the text that reads:

> When a load to a location appears in program order after the SSBB instruction, then the load does not speculatively read an entry earlier in the coherence order for that location than the entry generated by the latest store satisfying all of the following conditions:
>
> - The store is to the same location as the load.
> - The store uses the same virtual address as the load.
> - The store appears in program order before the SSBB instruction.
>
> When a load to a location appears in program order before the SSBB instruction, then the load does not speculatively read data from any store satisfying all of the following conditions:
>
> - The store is to the same location as the load.
> - The store uses the same virtual address as the load.
> - The store appears in program order after the SSBB instruction.

is changed to read:

> When a load to a location appears in program order after the SSBB instruction, then the load does not speculatively read an entry earlier in the coherence order for that location than the entry generated by the latest store satisfying all of the following conditions:

- The store is to the same location as the load.
- The store uses the same virtual address as the load.
- The store appears in program order before the SSBB instruction.

In section B2.6.7 “Physical Speculative Store Bypass Barrier (PSSBB)”, the text that reads:

When a load to a location appears in program order after the PSSBB instruction, then the load does not speculatively read an entry earlier in the coherence order for that location than the entry generated by the latest store satisfying all of the following conditions:

- The store is to the same location as the load.
- The store appears in program order before the PSSBB instruction.

When a load to a location appears in program order before the PSSBB instruction, then the load does not speculatively read data from any store satisfying all of the following conditions:

- The store is to the same location as the load.
- The store appears in program order after the PSSBB instruction.

is changed to read:

When a load to a location appears in program order after the PSSBB instruction, then the load does not speculatively read an entry earlier in the coherence order for that location than the entry generated by the latest store satisfying all of the following conditions:

- The store is to the same location as the load.
- The store appears in program order before the PSSBB instruction.

2.70 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.
0b1	Operation executed on an SMCU or other shared resource.

> 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.
0b1	Do not record samples that have the SMCU or other shared resource operation event == 0.

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

2.71 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;
```

2.72 D23462

In section J1.3 “Shared Pseudocode”, for the function IsDataAccess() the pseudocode that reads:

```
boolean IsDataAccess(AccessType acctype)
    return ! acctype IN {AccessType_IFETCH,
                         AccessType_TTW,
                         AccessType_DC,
                         AccessType_IC,
                         AccessType_AT};
```

is changed to read:

```
boolean IsDataAccess(AccessType acctype)
    return ! acctype IN {AccessType_IFETCH,
                         AccessType_TTW,
                         AccessType_DC,
                         AccessType_IC,
                         AccessType_SPE,
                         AccessType_TRBE,
                         AccessType_AT};
```

2.73 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;
```

2.74 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;
```

2.75 C23479

In section A2.2.5 “The Armv8.4 architecture extension”, the text that reads:

> FEAT_BBM, Translation table break-before-make levels
>
> FEAT_BBM provides support to identify the requirements of hardware to have break-before-make sequences when changing between block size for a translation.
>
> This feature is supported in AArch64 state only.
>
> FEAT_BBM is OPTIONAL from Armv8.3.
>
> FEAT_BBM is mandatory from Armv8.4.
>
> The following field identifies the presence of FEAT_BBM:
>
> - ID_AA64MMFR2_EL1.BBM.
>
> For more information, see:
>
> - Block descriptor and Page descriptor formats.
> - Block translation entry.
> - Support levels for changing block size..

is changed to read:

> FEAT_BBML1, Translation table break-before-make level 1
>
> FEAT_BBML1 provides support for reducing the requirements for following a break-before-make sequence when changing between block sizes for a translation.

This feature is supported in AArch64 state only.

FEAT_BBML1 is OPTIONAL from Armv8.3.

The following field identifies the presence of FEAT_BBML1:

- ID_AA64MMFR2_EL1.BBM.

For more information, see:

- Block descriptor and Page descriptor formats.
- Block translation entry.
- Support levels for changing block size.

FEAT_BBML2, Translation table break-before-make level 2

FEAT_BBML2 provides support for further reducing the requirements for following break-before-make sequences when changing between block sizes for a translation.

This feature is supported in AArch64 state only.

If FEAT_BBML2 is implemented, FEAT_BBML1 is implemented.

The following field identifies the presence of FEAT_BBML2:

- ID_AA64MMFR2_EL1.BBM.

For more information, see:

- FEAT_BBML1.

In section D8.3.1.2 “VMSAv8-64 Block descriptor and Page descriptor formats”, under rule R_{JJNHR}, the text in the table that reads:

Bit position	Field	Condition
Block[16]	**RES0**	FEAT_BBM is not implemented
Block[16]	nT	FEAT_BBM is implemented

is changed to read:

Bit position	Field	Condition
Block[16]	**RES0**	FEAT_BBML1 is not implemented
Block[16]	nT	FEAT_BBML1 is implemented

A similar table update is applied to rule R_{DMBGN}.

In section D8.7.3 “Block translation entry”, under rule R_{DFJNG},the text that reads:

R_{DFJNG}

> All statements in this section require implementation of FEAT_BBM.

is changed to read:

> R_{DFJNG}
>
> All statements in this section require implementation of FEAT_BBML1.

In the same section, under rule R_{MRRPW}, the text that reads:

> R_{MRRPW}
>
> If the implementation meets either level 1 or level 2 support requirements for changing table or block size, then when using a Table descriptor or Block descriptor with the nT bit set, the PE is permitted to do one of the following:
>
> - Generate a Translation fault and not cache the entry in a TLB.
> - If an entry that does not have the nT bit set is cached within a TLB and translates the same address to the same output address with consistent memory attributes and permissions, then the PE guarantees that accesses translated by the translation table entry with the nT bit set does not break coherency, ordering guarantees or uniprocessor semantics, or fail to clear the Exclusives monitors. For more information, see Support levels for changing table or block size.

is changed to read:

> R_{MRRPW}
>
> When using a Table descriptor or Block descriptor with the nT bit set, the PE is permitted to do one of the following:
>
> - Generate a Translation fault and not cache the entry in a TLB.
> - If an entry that does not have the nT bit set is cached within a TLB and translates the same address to the same output address with consistent memory attributes and permissions, then the PE guarantees that accesses translated by the translation table entry with the nT bit set does not break coherency, ordering guarantees or uniprocessor semantics, or fail to clear the Exclusives monitors. For more information, see Support levels for changing table or block size.

In section D8.15.1.1 “Translation fault”, under rule R_{FDQJL}, the text that reads:

> R_{FDQJL}
>
> If FEAT_BBM is implemented and supported at level 1 or higher, and the Block descriptor has the nT bit set, then the implementation is permitted to generate a Translation fault.

is changed to read:

> R_{FDQJL}
>
> If FEAT_BBML1 is implemented, and the Block descriptor has the nT bit set, then the implementation is permitted to generate a Translation fault.

In section D8.15.4 “MMU fault-checking sequence”, in the informational statement I_{PTYRT}, the text that reads:

> 1. If FEAT_BBM is implemented and supported at level 1 or higher, and the fetched descriptor is a block descriptor with the nT bit set, then the implementation can generate a Translation fault.

is changed to read:

> 1. If FEAT_BBML1 is implemented, and the fetched descriptor is a block descriptor with the nT bit set, then the implementation can generate a Translation fault.

In section D8.17.1 “Using break-before-make when updating translation table entries”, under rule R_{WHZWS}, the text that reads:

> R_{WHZWS}
>
> If the requirements enabled by FEAT_BBM level 1 or above cannot be followed, all of the following changes to the block size used by the translation system:
>
> - Changing from a smaller size to a larger size, such as when a stage 2 Table descriptor is replaced with a Block descriptor.
> - Changing from a larger size to a smaller size, such as when a stage 2 Block descriptor is replaced with a Table descriptor.

is changed to read:

> R_{WHZWS}
>
> If FEAT_BBML1 is not implemented, all of the following changes to the block size used by the translation system:
>
> - Changing from a smaller size to a larger size, such as when a stage 2 Table descriptor is replaced with a Block descriptor.
> - Changing from a larger size to a smaller size, such as when a stage 2 Block descriptor is replaced with a Table descriptor.

In section D8.17.2 “Support levels for changing table or block size”, the following rule is removed:

> R_{KNVDX}
>
> All statements in this section require implementation of FEAT_BBM.

In the same section, under rule R_{KFLJB}, the text that reads:

> R_{KFLJB}
>
> When a translation table entry is modified to change the table or block size, the hardware provides one of the following possible support levels affecting the break-before-make requirement to avoid breaking coherency, ordering guarantees or uniprocessor semantics, or failing to clear the Exclusives monitors:

- If level 0 is supported, then software is required to use break-before-make.
- If level 1 is supported, then software can use the level 0 approach or use the block translation entry bit, nT, in the Table descriptor or Block descriptor.
- If level 2 is supported, then all of the following apply:
  - Software can use the level 0 approach or the level 1 approach.
  - Changing table or block size does not break coherency, ordering guarantees or uniprocessor semantics, or fail to clear the Exclusives monitors. For more information, see Block translation entry.

is changed to read:

R_{KFLJB}

When a translation table entry is modified to change the table or block size, the hardware provides one of the following possible implementations affecting the break-before-make requirement to avoid breaking coherency, ordering guarantees or uniprocessor semantics, or failing to clear the Exclusives monitors:

- If FEAT_BBML1 is not implemented, then software is required to use the break-before-make sequence.
- If FEAT_BBML1 is implemented, then software can use all of the following: - The break-before-make sequence. - The nT bit in the Table descriptor or Block descriptor.
- If FEAT_BBML2 is implemented, then software can change table or block size without breaking coherency, ordering guarantees or uniprocessor semantics, or failing to clear the Exclusives monitors.

In the same section, rule I_{HYQMB} the text that reads:

I_{HYQMB}

If any level is supported and the TLB entries are not invalidated after the writes that modified the translation table entries are completed, then a TLB conflict abort can be generated because in a TLB there might be multiple translation table entries that all translate the same IA. For Table descriptors, this also applies to intermediate TLB caching structures. For more information, see TLB conflict abort.

is changed to read:

I_{HYQMB}

If the TLB entries are not invalidated after the writes that modified the translation table entries are completed, then a TLB conflict abort can be generated because in a TLB there might be multiple translation table entries that all translate the same IA. For Table descriptors, this also applies to intermediate TLB caching structures. For more information, see TLB conflict abort.

In the same section, under rule R_{KHRBC}, the text that reads:

R_{KHRBC}

> If level 1 or level 2 is supported, then changing the Contiguous bit in a set of Block descriptors or Page descriptors can be done without breaking coherency, ordering guarantees or uniprocessor semantics, or failing to clear the Exclusives monitors.

is changed to read:

> R_{KHRBC}
>
> If FEAT_BBML1 is implemented, then changing the Contiguous bit in a set of Block descriptors or Page descriptors can be done without breaking coherency, ordering guarantees or uniprocessor semantics, or failing to clear the Exclusives monitors.

In the same section, under rule R_{FCPSG}, the text that reads:

> R_{FCPSG}
>
> If level 1 or level 2 is supported and the Contiguous bit in a set of Block descriptors or Page descriptors is changed, then a TLB conflict abort can be generated because multiple translation table entries might exist within a TLB that translates the same IA.

is changed to read:

> R_{FCPSG}
>
> If FEAT_BBML1 is implemented and the Contiguous bit in a set of Block descriptors or Page descriptors is changed, then a TLB conflict abort can be generated because multiple translation table entries might exist within a TLB that translates the same IA.

In the same section, rule $_{RFWRMB}$ that reads:

> R_{FWRMB}
>
> If all of the following apply, then a TLB conflict abort is reported to EL2:
>
> - Level 1 or level 2 is supported.
> - Stage 2 translations are enabled in the current translation regime.
> - A TLB conflict abort is generated due to changing the block size or Contiguous bit.

is changed to read:

> R_{FWRMB}
>
> If all of the following apply, then a TLB conflict abort is reported to EL2:
>
> - FEAT_BBML1 is implemented.
> - Stage 2 translations are enabled in the current translation regime.
> - A TLB conflict abort is generated due to changing the block size or Contiguous bit.

In the same section, in the informational statement I_{CFFVK}, the text that reads:

> I_{CFFVK}
>
> If level 1 or level 2 is supported and a TLB conflict abort is generated, then TLB maintenance is required to remove the multiple TLB entries that translate the same address. For Table descriptors, this also applies to intermediate TLB caching structures. For more information, see TLB maintenance due to TLB conflict.

is changed to read:

> I_{CFFVK}
>
> If FEAT_BBML1 is implemented and a TLB conflict abort is generated, then TLB maintenance is required to remove the multiple TLB entries that translate the same address. For Table descriptors, this also applies to intermediate TLB caching structures. For more information, see TLB maintenance due to TLB conflict.

In section D9.4.4 “GPT Contiguous descriptor”, under informational statement I_{JMVJS}, the text that reads:

> I_{JMVJS}
>
> This behavior is intended to be the same as the level 2 behavior that is specified in the FEAT_BBM feature, but with the option of TLB Conflict aborts removed. For more information, see Support levels for changing table or block size.

is changed to read:

> I_{JMVJS}
>
> This behavior is intended to be the same as FEAT_BBML2 behavior, but with the option of TLB Conflict aborts removed. For more information, see Support levels for changing table or block size.

In section D24.2.84 “ID_AA64MMFR2_EL1, AArch64 Memory Model Feature Register 2”, the text that reads:

> BBM, bits [55:52]
>
> Allows identification of the requirements of the hardware to have break-before-make sequences when changing block size for a translation.

BBM	Meaning
`0b0000`	Level 0 support for changing block size is supported.
`0b0001`	Level 1 support for changing block size is supported.
`0b0010`	Level 2 support for changing block size is supported.

> All other values are reserved.
>
> FEAT_BBM implements the functionality identified by the values `0b0000`, `0b0001`, and `0b0010`.

From Armv8.4, the permitted values are `0b0000`, `0b0001`, and `0b0010`.

Access to this field is RO.

is changed to read:

BBM, bits [55:52]

Allows identification of the requirements of the hardware to have break-before-make sequences when changing block or table size for a translation.

BBM	Meaning
`0b0000`	Break-before-make sequence must be used.
`0b0001`	Level 1 support for changing block size is supported.
`0b0010`	Level 2 support for changing block size is supported.

All other values are reserved.

FEAT_BBML1 implements the functionality identified by the value `0b0001`.

FEAT_BBML2 implements the functionality identified by the value `0b0010`.

Access to this field is RO.

2.76 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)”.

2.77 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.

2.78 D23518

In section A.2.2.5 “The Armv8.4 architecture extension”, the text that reads:

> FEAT_MPAM, Memory Partitioning and Monitoring Extension
>
> The MPAM Extension provides a framework for memory-system component controls that partition one or more of the performance resources of the component.
>
> This feature is supported in AArch64 state only.
>
> FEAT_MPAM is OPTIONAL from Armv8.2.
>
> The following field identifies the presence of FEAT_MPAM:
> - ID_AA64PFR0_EL1.MPAM.

is changed to read:

> FEAT_MPAMv1p0, Memory Partitioning and Monitoring Extension version 1.0
>
> FEAT_MPAMv1p0 introduces support for version 1.0 of the MPAM extension.
>
> This feature is supported in AArch64 state only.
>
> FEAT_MPAMv1p0 is OPTIONAL from Armv8.2.
>
> The following field identifies the presence of FEAT_MPAMv1p0:
> - ID_AA64PFR0_EL1.MPAM.
> - ID_AA64PFR1_EL1.MPAM_frac.

The following text is added to the same section:

> FEAT_MPAM, The Memory Partitioning and Monitoring Extension.
>
> The MPAM Extension provides a framework for memory-system component controls that partition one or more of the performance resources of the component.
>
> This feature is supported in AArch64 state only.
>
> FEAT_MPAM is OPTIONAL from Armv8.2.
>
> There are three versions of the MPAM extension: v1p0, v0p1, and v1p1.
>
> The following fields identifies the presence of FEAT_MPAM:
> - ID_AA64PFR0_EL1.MPAM.
> - ID_AA64PFR1_EL1.MPAM_frac.

2.79 D23521

In section D8.5.2 “The dirty state”, the text that reads:

> 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:
>
> ...

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 D6.3.4 “Accesses to the trace buffer”.

2.80 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”.

2.81 D23529

In section B2.8.2.1.3 “Non-atomic Load-Acquire/Store-Release instructions”, the text that reads:

> If FEAT_LSE2 is implemented, then:
>
> - If the memory access is not to Normal Inner Write-Back or Outer Write-Back Cacheable memory, then it is a **CONSTRAINED UNPREDICTABLE** choice of either of the following:
>   - An unaligned access is performed meeting all of the semantics of the instruction.
>   - An Alignment fault is generated.

is changed to read:

> If FEAT_LSE2 is implemented, then:
>
> - If the memory access is not to Normal Inner Write-Back, Outer Write-Back Cacheable, Shareable memory, then it is a **CONSTRAINED UNPREDICTABLE** choice of either of the following:
>   - An unaligned access is performed which is not guaranteed to be single-copy atomic except at the byte access level.
>   - An Alignment fault is generated.

In addition, the text that reads:

> In this case, the architecture does not define the order of the different transactions of the access defined by the single instructions relative to each other.

is changed to read:

> In this case, when an access is performed it meets all the semantics of the instruction, but the architecture does not define the order of the different transactions of the access defined by the single instructions relative to each other.

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

```
if IsWBShareable(memaddrdesc.memattrs) then
    atomic = TRUE;
elsif (accdesc.exclusive || accdesc.atomicop || accdesc.acqsc || accdesc.acqpc ||
 accdesc.relsc) then
    atomic = TRUE;
...
```

is changed to read:

```
if IsWBShareable(memaddrdesc.memattrs) then
    atomic = TRUE;
elsif accdesc.exclusive || accdesc.atomicop then
    atomic = TRUE;
...
```

The equivalent change is made in the same section, in the pseudocode function AArch64.MemSingleWrite().

2.82 D23530

In section D24.2.56 "HCR_EL2, Hypervisor Configuration Register", under the heading 'MIOCNCE bit [38]' the text that reads:

> MIOCNCE, bit [38]
>
> Mismatched Inner/Outer Cacheable Non-Coherency Enable, for the EL1&0 translation regimes.

MIOCNCE	Meaning
0b0	For the EL1&0 translation regimes, for permitted accesses to a memory location that use a common definition of the Shareability and Cacheability of the location, there must be no loss of coherency if the Inner Cacheability attribute for those accesses differs from the Outer Cacheability attribute.
0b1	For the EL1&0 translation regimes, for permitted accesses to a memory location that use a common definition of the Shareability and Cacheability of the location, there might be a loss of coherency if the Inner Cacheability attribute for those accesses differs from the Outer Cacheability attribute.

> For more information, see 'Mismatched memory attributes'.
>
> This field can be implemented as **RAZ/WI**.
>
> When the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}, the PE ignores the value of this field for all purposes other than a direct read of this field.
>
> The reset behavior of this field is:
>
> - On a Warm reset, this field resets to an architecturally **UNKNOWN** value.

is changed to read:

> Bit [38]
>
> Reserved, **RES0**.
>
> Note:
>
> This bit was previously the MIOCNCE control.

In section B2.11 "Mismatched memory attributes", the text that reads:

> For the following situations, when a physical memory Location is accessed with mismatched attributes, a more restrictive set of behaviors applies. The description of each situation also describes the behaviors that apply:
>
> 1. Any agent that reads that memory Location using the same common definition of the Memory type, Shareability and Cacheability attributes is guaranteed to access it coherently,

to the extent required by that common definition of the memory attributes, only if all the following conditions are met:

- All writes are...
- Either:
  - In the EL1&0 translation regime, HCR_EL2.MIOCNCE has a value of 0.
  - All aliases with write permission have the Inner Cacheability attribute the same as the Outer Cacheability attribute.
- Either:
  - All writes are performed...
  - All aliases to a memory...

is changed to read:

For the following situations, when a physical memory Location is accessed with mismatched attributes, a more restrictive set of behaviors applies. The description of each situation also describes the behaviors that apply:

1. Any agent that reads that memory Location using the same common definition of the Memory type, Shareability and Cacheability attributes is guaranteed to access it coherently, to the extent required by that common definition of the memory attributes, only if all the following conditions are met:
   - All writes are...
   - Either:
     - All writes are performed...
     - All aliases to a memory...

2.83 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().

2.84 D23543

In section J1.1 “Pseudocode for AArch64 operation”, in function AArch64.MemSingleRead(), the code that reads:

```
(bits(size*8), AddressDescriptor, PhysMemRetStatus)
AArch64.MemSingleRead(bits(64) address,
                      integer size,
                      AccessDescriptor accdesc_in,
                      boolean aligned)
  ...
  elsif accdesc.acctype == AccessType_ASIMD && size == 32 && accdesc.ispair
  then
    ...
    for i = 0 to 3 do
      (memstatus, value<i*64+:64>) = PhysMemRead(memaddrdesc, 8, accdesc);
      if IsFault(memstatus) then
        return (value, memaddrdesc, memstatus);
      memaddrdesc.paddress.address = memaddrdesc.paddress.address + 8;
      memaddrdesc.vaddress = memaddrdesc.vaddress + 8;
  elsif aligned && accdesc.ispair then
    ...
    memaddrdesc.paddress.address = memaddrdesc.paddress.address + halfsize;
    memaddrdesc.vaddress = memaddrdesc.vaddress + halfsize;
    (memstatus, highhalf) = PhysMemRead(memaddrdesc, halfsize, accdesc);
    if IsFault(memstatus) then
      return (value, memaddrdesc, memstatus);
    value = highhalf:lowhalf;
  else
    for i = 0 to size-1 do
      (memstatus, Elem[value, i, 8]) = PhysMemRead(memaddrdesc, 1, accdesc);
      if IsFault(memstatus) then
        return (value, memaddrdesc, memstatus);
      memaddrdesc.paddress.address = memaddrdesc.paddress.address + 1;
```

```
      memaddrdesc.vaddress = memaddrdesc.vaddress + 1;
```

is changed to read:

```
(bits(size*8), AddressDescriptor, PhysMemRetStatus)
AArch64.MemSingleRead(bits(64) address,
                      integer size,
                      AccessDescriptor accdesc_in,
                      boolean aligned)
  ...
  elsif accdesc.acctype == AccessType_ASIMD && size == 32 && accdesc.ispair
  then
    ...
    for i = 0 to 3 do
      (memstatus, value<i*64+:64>) = PhysMemRead(memaddrdesc, 8, accdesc);
      if IsFault(memstatus) then
        return (value, memaddrdesc, memstatus);
      memaddrdesc.paddress.address = memaddrdesc.paddress.address + 8;
      memaddrdesc.vaddress = memaddrdesc.vaddress + 8;
  elsif aligned && accdesc.ispair then
    ...
    memaddrdesc.paddress.address = memaddrdesc.paddress.address + halfsize;
    memaddrdesc.vaddress = memaddrdesc.vaddress + halfsize;
    (memstatus, highhalf) = PhysMemRead(memaddrdesc, halfsize, accdesc);
    if IsFault(memstatus) then
      return (value, memaddrdesc, memstatus);
    value = highhalf:lowhalf;
  else
    for i = 0 to size-1 do
      (memstatus, Elem[value, i, 8]) = PhysMemRead(memaddrdesc, 1, accdesc);
      if IsFault(memstatus) then
        return (value, memaddrdesc, memstatus);
      memaddrdesc.paddress.address = memaddrdesc.paddress.address + 1;
      memaddrdesc.vaddress = memaddrdesc.vaddress + 1;
```

Equivalent changes are made in the following functions:

- Arch64.MemSingleWrite().
- MemStore64B().
- MemLoad64B().
- AArch64.WriteTagMem().

2.85 R23545

In section D14.3.2 “Common microarchitectural events”, the following text in the description of ‘`0x8074`, SVE_PRED_SPEC, Operation speculatively executed, SVE predicated’:

> `0x8074`, SVE_PRED_SPEC, Operation speculatively executed, SVE predicated
>
> The counter counts each speculatively executed SIMD data-processing, load, or store operation due to an instruction with a single Governing predicate operand that determines the Active elements.

> When FEAT_SME is implemented, both operations due to SVE instructions and operations due to SME instructions operating on the SVE Z vectors with at least one Governing predicate operand are counted.
>
> Note:
>
> For outer product instructions which are widening, predication is considered with respect to the input element size.

Is changed to read:

> `0x8074`, SVE_PRED_SPEC, Operation speculatively executed, SVE predicated
>
> The counter counts each speculatively executed predicated SIMD data-processing, load, or store operation. This includes all of the following:
>
> - A data processing or load operation that writes to one or more SVE Z vector destination registers under a Governing predicate using either zeroing or merging predication.
> - A predicated store of one or more SVE Z vector registers.
>
> It is **IMPLEMENTATION DEFINED** whether data processing operations due to instructions with a single Governing predicate operand that determines the Active elements that do not write to any SVE Z vector destination registers using either zeroing or merging predication are counted. For example, INCP. Arm recommends these operations are not counted when FEAT_SPE is implemented, for consistency with the Operation Type packet PRED field.
>
> When FEAT_SME is implemented, both operations due to SVE instructions and operations due to SME instructions operating on the SVE Z vectors with at least one Governing predicate operand are counted.
>
> Note:
>
> For outer product instructions which are widening, predication is considered with respect to the input element size.

2.86 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.

2.87 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.

2.88 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';
...
```

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);
...
```

2.89 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.

2.90 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)“.

2.91 C23586

In section B2.3.1 “Basic definitions”, the definition ‘Fault Effects, Exception Entry and Exception Return Effects’ that reads:

> Fault Effects, Exception Entry, and Exception Return Effects
>
> An instruction that encounters a fault generates a Fault Effect. An instruction that encounters a fault leading to a synchronous exception generates a Fault Effect that is also an Exception Entry Effect, for example as a result of reading an invalid TTD or fetching an illegal instruction.
>
> An ERET instruction generates an Exception Return Effect.
>
> A Fault Effect generated by the MMU is called an MMU Fault Effect.
>
> A Fault Effect generated by a failed Tag Check is called a TagCheck Fault Effect.

is changed to read:

> Fault Effects, Exception Entry, and Exception Return Effects
>
> An instruction that encounters a fault generates a Fault Effect. An instruction that encounters a fault leading to a synchronous exception generates a Fault Effect that is also an Exception Entry Effect, for example as a result of reading an invalid TTD or fetching an illegal instruction.
>
> An ERET instruction generates an Exception Return Effect.
>
> A Fault Effect generated by the MMU is called an MMU Fault Effect.
>
> A Fault Effect generated by a failed Tag Check is called a TagCheck Fault Effect.
>
> Instructions that write to memory with Release semantics generate Fault Effects with Release semantics when they encounter a fault.

In section B2.3.7 “Ordering relations”, the definition of ‘Barrier-ordered-before’ that reads:

> Barrier-ordered-before
>
> An Effect E_1 is Barrier-ordered-before an Effect E_2 if one of the following applies:
> - All of the following apply:
>   - E_1 is an Explicit Memory Write Effect and is generated by an atomic instruction with both Acquire and Release semantics.
>   - E_1 appears in program order before E_2.
>   - One of the following applies:
>   - E_2 is an Explicit Memory Effect.
>   - E_2 is an Implicit Tag Memory Read Effect.
>   - E_2 is an MMU Fault Effect.
> - All of the following apply:
>   - E_1 is an Explicit Memory Effect or a Fault Effect and generated by an instruction with Release semantics.
>   - E_1 appears in program order before E_2.
>   - E_2 is an Explicit Memory Effect generated by an instruction with Acquire semantics.
> - All of the following apply:
>   - One of the following applies:
>   - E_1 is an Explicit Memory Effect generated by an instruction with Acquire semantics.
>   - E_1 is an Explicit Memory Effect generated by an instruction with AcquirePC semantics.
>   - E_1 appears in program order before E_2.
>   - One of the following applies:
>   - E_2 is an Explicit Memory Effect.
>   - E_2 is an Implicit Tag Memory Read Effect.
>   - E_2 is an MMU Fault Effect.
> - All of the following apply:
>   - One of the following applies:
>   - E_1 is an Explicit Memory Effect generated by an instruction with Acquire semantics.
>   - E_1 is an Explicit Memory Effect generated by an instruction with AcquirePC semantics.
>   - There is an Intrinsic Order Dependency from E_1 to E_2.
>   - One of the following applies:
>   - E_2 is an Explicit Memory Effect.
>   - E_2 is an Implicit Tag Memory Read Effect.
>   - E_2 is an MMU Fault Effect.
> - All of the following apply:

    - One of the following applies:
        - E_1 is an Explicit Memory Effect.
        - E_1 is an Implicit Tag Memory Read Effect.
    - E_1 appears in program order before E_2.
    - E_2 is an Explicit Memory Effect or a Fault Effect and generated by an instruction with Release semantics.
- All of the following apply:
    - One of the following applies:
        - E_1 is an Explicit Memory Effect.
        - E_1 is an Implicit Tag Memory Read Effect.
    - There is an Intrinsic Order Dependency from E_1 to E_2.
    - E_2 is an Explicit Memory Effect or a Fault Effect and generated by an instruction with Release semantics.

is changed to read:

Barrier-ordered-before

An Effect E_1 is Barrier-ordered-before an Effect E_2 if one of the following applies:

- All of the following apply:
    - E_1 is an Effect with Release semantics.
    - E_1 and E_3 are generated by an atomic instruction.
    - E_3 is an Effect with Acquire semantics.
    - E_1 appears in program order before E_2.
    - One of the following applies:
        - E_2 is an Explicit Memory Effect.
        - E_2 is an Implicit Tag Memory Read Effect.
        - E_2 is an MMU Fault Effect.
- All of the following apply:
    - E_1 is an Effect with Release semantics.
    - E_1 appears in program order before E_2.
    - E_2 is an Effect with Acquire semantics.
- All of the following apply:
    - One of the following applies:
        - E_1 is an Effect with Acquire semantics.
        - E_1 is an Effect with AcquirePC semantics.
    - E_1 appears in program order before E_2.

    - One of the following applies:
        - E_2 is an Explicit Memory Effect.
        - E_2 is an Implicit Tag Memory Read Effect.
        - E_2 is an MMU Fault Effect.
- All of the following apply:
    - One of the following applies:
        - E_1 is an Effect with Acquire semantics.
        - E_1 is an Effect with AcquirePC semantics.
    - There is an Intrinsic Order Dependency from E_1 to E_2.
    - One of the following applies:
        - E_2 is an Explicit Memory Effect.
        - E_2 is an Implicit Tag Memory Read Effect.
        - E_2 is an MMU Fault Effect.
- All of the following apply:
    - One of the following applies:
        - E_1 is an Explicit Memory Effect.
        - E_1 is an Implicit Tag Memory Read Effect.
    - E_1 appears in program order before E_2.
    - E_2 is an Effect with Release semantics.
- All of the following apply:
    - One of the following applies:
        - E_1 is an Explicit Memory Effect.
        - E_1 is an Implicit Tag Memory Read Effect.
    - There is an Intrinsic Order Dependency from E_1 to E_2.
    - E_2 is an Effect with Release semantics.

In the same section, the definition of 'Atomic-ordered-before' that reads:

> Atomic-ordered-before
>
> An Effect E_1 is Atomic-ordered-before an Effect E_2 if one of the following applies:
> - All of the following apply:
>   - E_1 is an Explicit Memory Effect.
>   - E_1 and E_3 form a successful Read-Modify-Write pair.
>   - E_2 is a Local read successor of E_3.
>   - One of the following applies:
>   - E_2 is an Explicit Memory Effect generated by an instruction with Acquire semantics.

        - E_2 is an Explicit Memory Effect generated by an instruction with AcquirePC semantics.

is changed to read:

Atomic-ordered-before

An Effect E_1 is Atomic-ordered-before an Effect E_2 if one of the following applies:

- All of the following apply:
    - E_1 is an Explicit Memory Effect.
    - E_1 and E_3 form a successful Read-Modify-Write pair.
    - E_2 is a Local read successor of E_3.
    - One of the following applies:
        - E_2 is an Effect with Acquire semantics.
        - E_2 is an Effect with AcquirePC semantics.

2.92 C23587

In section B.2.4.1.1 “Peripherals”, the definition of Out-of-band-ordered-before that reads:

A Read Memory or write effect RW1 is Out-of-band-ordered-before a Read Memory or write effect RW2 if either of the following cases apply:

- RW1 appears in program order before a DSB instruction that begins an **IMPLEMENTATION DEFINED** instruction sequence indirectly leading to the generation of RW2.
- RW1 is Ordered-before a Read Memory or write effect RW3 and RW3 is Out-of-band-ordered-before RW2.

If a Memory effect M1 is Out-of-band-ordered-before a Read Memory or write effect M2, then M1 is seen to occur before M2 by all observers.

is changed to read:

An Effect E_1 is Out-of-band-ordered-before an Effect E_2 if and only if all of the following apply:

- E_1 is a Memory Effect.
- Any of the following applies:
    - E_1 is DSB-ordered-before E_3.
    - E_1 is Instruction-fetch-barrier-ordered-before E_3.
    - There is an **IMPLEMENTATION DEFINED** instruction sequence from E_3 indirectly leading to the generation of E_2.
- E_2 is a Memory Effect.

If an Effect E_1 is Out-of-band-ordered-before an Effect E_2, then E_1 is Ordered-before E_2.

2.93 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.

2.94 D23599

In section J1.3 “Shared Pseudocode”, the function PhysicalSErrorTarget(), the code that reads:

```
(boolean, bits(2)) PhysicalSErrorTarget()
    ...

    // External debug might disable the exception in the Security state indicated by
    // SCR_EL3.{NS, NSE}.
    boolean intdis = ExternalDebugInterruptsDisabled(EL1);
    ...
```

is changed to read:

```
(boolean, bits(2)) PhysicalSErrorTarget()
    ...

    // External debug might disable the exception in the current Security state.
    // This is not relevant at EL3.
    boolean intdis = PSTATE.EL != EL3 && ExternalDebugInterruptsDisabled(EL1);
    ...
```

2.95 D23637

In section D24.5.29 “PMZR_EL0, Performance Monitors Zero with Mask”, under the heading ‘Bits [63:33]’ the text that reads:

- This field ignores writes if any of the following are true:
  - ...
  - All of the following are true:
    - FEAT_FGT2 is implemented.
    - HDFGWTR2_EL2.nPMICFILTR_EL0 == 0.
    - EL2 is implemented and enabled in the current Security state.
    - Accessed at EL1 or EL0.
    - The Effective value of HCR_EL2.{E2H,TGE} is not {1,1}.

is changed to read:

- This field ignores writes if any of the following are true:
  - All of the following are true:
    - FEAT_FGT2 is implemented.
    - HDFGWTR2_EL2.nPMICNTR_EL0 == 0.
    - EL2 is implemented and enabled in the current Security state.
    - Accessed at EL1 or EL0.
    - The Effective value of HCR_EL2.{E2H,TGE} is not {1,1}.

2.96 C23641

In section D6.4.2 “Behavior in External mode”, under the rule R_{CMYYF}, the text that reads:

> If the authentication interface prohibits invasive debug of the Security state corresponding to the physical address space selected by TRBMAR_EL1.PAS, or TRBMAR_EL1.PAS is set to a reserved value, then no trace is written to the trace buffer and all of the following occur:
>
> - A trace buffer management event is generated.
>
> ....

is changed to read:

> If the authentication interface prohibits invasive debug of the Security state corresponding to the physical address space selected by TRBMAR_EL1.PAS, or TRBMAR_EL1.PAS is set to a reserved value, then when the Trace Buffer Unit receives trace data from the trace unit, no trace is written to the trace buffer and all of the following occur:
>
> - A trace buffer management event is generated.
>
> ....

In section D6.4.5 “External mode and MPAM”, under the rule R_{VVWKG}, the text that reads:

> If the authentication interface prohibits invasive debug of the Security state corresponding to the MPAM_SP.PARTID space selected by TRBMPAM_EL1.MPAM_SP, or TRBMPAM_EL1.MPAM_SP is set to a reserved value, then no trace is written to the trace buffer and all of the following occur:
>
> - A trace buffer management event is generated.
>
> ....

is changed to read:

> If the authentication interface prohibits invasive debug of the Security state corresponding to the MPAM_SP.PARTID space selected by TRBMPAM_EL1.MPAM_SP, or TRBMPAM_EL1.MPAM_SP is set to a reserved value, then when the Trace Buffer Unit receives trace data from the trace unit, no trace is written to the trace buffer and all of the following occur:
>
> - A trace buffer management event is generated.
>
> ....

2.97 D23647

In section C5.5.21 “TLBI RIPAS2E1, TLBI RIPAS2E1NXS, TLB Range Invalidate by Intermediate Physical Address, Stage 2, EL1”, in the description of the field BaseADDR, bits [36:0], the text that reads:

> When (FEAT_LPA2 is implemented and TCR_EL1.DS == 1) or (FEAT_D128 is implemented and VTCR_EL2.D128 == 1):
>
> The starting address for the range of the maintenance instructions. This field is BaseADDR[52:16] for all translation granules.
>
> - When using a 4KB translation granule, BaseADDR[15:12] is treated as `0b0000`.
> - When using a 16KB translation granule, BaseADDR[15:14] is treated as `0b00`.

is changed to read:

> When (FEAT_LPA2 is implemented and VTCR_EL2.DS == 1) or (FEAT_D128 is implemented and VTCR_EL2.D128 == 1):
>
> The starting address for the range of the maintenance instructions. This field is BaseADDR[52:16] for all translation granules.
>
> - When using a 4KB translation granule, BaseADDR[15:12] is treated as `0b0000`.
> - When using a 16KB translation granule, BaseADDR[15:14] is treated as `0b00`.

The equivalent changes are made in the following sections: - C5.5.22 “TLBI RIPAS2E1IS, TLBI RIPAS2E1ISNXS”. - C5.5.23 “TLBI RIPAS2E1OS, TLBI RIPAS2E1OSNXS”. - C5.5.24 “TLBI RIPAS2LE1, TLBI RIPAS2LE1NXS”. - C5.5.25 “TLBI RIPAS2LE1IS, TLBI RIPAS2LE1ISNXS”. - C5.5.26 “TLBI RIPAS2LE1OS, TLBI RIPAS2LE1OSNXS”.

2.98 D23659

In section B2.5.4 “Restrictions on the effects of speculation from Armv8.5”, the text that reads:

> If either FEAT_CSV2_1p1 or FEAT_CSV2_3 is implemented, code running in one hardware-defined context (context1) cannot either exploitatively control, or predictively leak to, the speculative execution of code in a different hardware-defined context (context2) as a result of the behavior of branch target prediction based on the branch history used in context1.

is changed to read:

> If either FEAT_CSV2_1p1 or FEAT_CSV2_3 is implemented, code running in one hardware-defined context (context1) cannot either exploitatively control, or predictively leak to, the speculative execution of code in a different hardware-defined context (context2) as a result of the behavior of any prediction mechanism based on the branch history used in context1.

The contents of section D7.5.13 “Branch prediction” are moved into a new section named “Branch history” under B2.5 “Restrictions on the effects of speculation”.

This leaves D7.5.13 “Branch prediction” to read as follows:

> If branch prediction is architecturally visible, cache maintenance must also apply to branch prediction.

The exact change will be made available at a later date as C23575.

The new “Branch history” section under B2.5 “Restrictions on the effects of speculation” reads:

> If FEAT_CLRBHB is not implemented, then the architecture does not define any branch history maintenance instructions for AArch64 state.
>
> When FEAT_CLRBHB is implemented, the CLRBHB instruction is available. When the CLRBHB instruction is executed, the branch history is cleared for the current context to the extent that branch history information created before the CLRBHB instruction cannot be used by code before the CLRBHB instruction to exploitatively control the execution of any code in the current context appearing in program order after the instruction.
>
> When FEAT_ECBHB is implemented, the branch history information created in a context before an exception entry to a higher Exception level using AArch64 cannot be used by code before that exception entry to exploitatively control the execution of any code in a different context after the exception entry.
>
> Note:
>
> FEAT_CLRBHB and FEAT_ECBHB apply to all forms of prediction that use the branch history and that are protected by FEAT_CSV2.

In section A2.2.10 “The Armv8.9 architecture extension”, the text under ‘FEAT_CLRBHB, Support for Clear Branch History instruction’ that reads:

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

is changed to read:

> FEAT_CLRBHB provides a CLRBHB instruction which can be used to clear the branch history.

In section D24.2.83 “ID_AA64MMFR1_EL1, AArch64 Memory Model Feature Register 1”, in the field description of ‘ECBHB, bits [63:60]’, the table that reads:

ECBHB	Meaning
0b0000	The implementation does not disclose whether the branch history information created in a context before an exception to a higher Exception level using AArch64 can be used by code before that exception to exploitatively control the execution of any indirect branches in code in a different context after the exception.
0b0001	The branch history information created in a context before an exception to a higher Exception level using AArch64 cannot be used by code before that exception to exploitatively control the execution of any indirect branches in code in a different context after the exception.

is changed to read

ECBHB	Meaning
0b0000	The implementation does not disclose whether restrictions are imposed on branch history speculation around exceptions.
0b0001	The implementation imposes restrictions on branch history speculation around exceptions.

2.99 D23661

In section J1.1.4 aarch64/translation, in the pseudocode function AArch64.S1POR(), the code that reads:

```
S1PORType AArch64.S1POR(Regime regime)
    case regime of
        when Regime_EL3  return POR_EL3;
        when Regime_EL2  return POR_EL2;
        ...
```

is changed to read:

```
S1PORType AArch64.S1POR(Regime regime)
    if HaveEL(EL3) && SCR_EL3.PIEn == '0' && regime != Regime_EL3 then
        return Zeros(64);

    case regime of
        when Regime_EL3  return POR_EL3;
        when Regime_EL2  return POR_EL2;
        ...
```

In the same section, in function AArch64.S1TTWParamsEL2(), the code segment that reads:

```
if IsFeatureImplemented(FEAT_S1PIE) then
        walkparams.pir = PIR_EL2;
```

is changed to read:

```
if IsFeatureImplemented(FEAT_S1PIE) then
    if !HaveEL(EL3) || SCR_EL3.PIEn == '1' then
        walkparams.pir = PIR_EL2;
    else
        walkparams.pir = Zeros(64);
```

Similar changes are made to the following functions:

- AArch64.S1OverlayPermissions().
- AArch64.S2OverlayPermissions().
- AArch64.S1TTWParamsEL20().
- AArch64.S1TTWParamsEL10().
- AArch64.SS2TTWParams().
- AArch64.NSS2TTWParams().

2.100 R23665

In section D19.5.1 “Manual injection of Branch records”, under the rule I_{BCZRK}, the text that reads:

> When BRB INJ is executed outside of a BRBE Prohibited region, it is **CONSTRAINED UNPREDICTABLE** whether a Branch record is injected.

is changed to read:

> When BRB INJ is executed outside of a BRBE Prohibited region, it is **CONSTRAINED UNPREDICTABLE** whether a Branch record is injected and it is **CONSTRAINED UNPREDICTABLE** whether the cycle count value in the next Branch record to be generated is Branch cycle count unknown.

2.101 D23670

In section D8.2.7 “Translation Hardening Extension”, under the rule R_{NYFNK}, the text that reads:

> For a stage 1 descriptor, if RCWS checks fail, then RCWS instructions do not update that descriptor. The RCWS checks are:
>
> RCWS State check:
>
> - This check fails if there is an attempt to change the validity of that descriptor.
>
> RCWS Mask check:
>
> - This check fails if, for a valid descriptor, an attempt is made to update a bit in that descriptor and the Effective value of the corresponding RCWSMASK_EL1 bit is 0.

is changed to read:

> For a stage 1 descriptor, if RCWS checks fail, then RCWS instructions do not update that descriptor. The RCWS checks are:
>
> RCWS State check:
>
> - This check fails if there is an attempt to change the validity of that descriptor and one of the following applies:

- The descriptor is valid.
- The descriptor is invalid and Protection is Disabled.
- The descriptor is invalid and it is a Non-Protected descriptor.

RCWS Mask check:

- This check fails if, for a valid descriptor, an attempt is made to update a bit in that descriptor and the Effective value of the corresponding RCWSMASK_EL1 bit is 0.

2.102 C23678

In section D12.2.4.1 “Operation of the CompareValue views of the timers”, the definition of the symbol Offset that reads:

Offset

For the EL1 physical timer, the offset value is the value held in the CNTPOFF_EL2 register when all of the following are true:

- SCR_EL3.{NS, EEL2} is not {0, 0}.
- SCR_EL3.ECVEn is 1,
- CNTHCTL_EL2.ECV is 1.
- The Effective value of HCR_EL2.{E2H, TGE} is not {1, 1}.

Otherwise the offset value of the EL1 physical timer is zero.

For the EL1 virtual timer, the offset value is held in the CNTVOFF_EL2 register.

For the EL2 physical and virtual timers, the offset value is zero.

is changed to read:

Offset

For the EL1 physical timer, the offset value is the value held in the CNTPOFF_EL2 register when all of the following are true:

- SCR_EL3.{NS, EEL2} is not {0, 0}.
- SCR_EL3.ECVEn is 1,
- CNTHCTL_EL2.ECV is 1.
- The Effective value of HCR_EL2.{E2H, TGE} is not {1, 1}.

Otherwise the offset value of the EL1 physical timer is zero.

For the EL1 virtual timer, the offset value is held in the CNTVOFF_EL2 register.

> For the EL2 Secure and Non-secure physical and virtual timers the offset value is zero.
>
> For the EL3 timer the offset value is zero.

2.103 D23680

In section D24.10.4 “CNTHP_CVAL_EL2, Counter-timer Physical Timer CompareValue Register (EL2)”, the text that reads:

> When CNTHP_CTL_EL2.ENABLE is 1, and TimerConditionMet is TRUE for the EL1 physical timer, the timer condition is met and all of the following are true:

is changed to read:

> When CNTHP_CTL_EL2.ENABLE is 1, and TimerConditionMet is TRUE for the EL2 physical timer, the timer condition is met and all of the following are true:

In section D24.10.7 “CNTHPS_CVAL_EL2, Counter-timer Secure Physical Timer CompareValue Register (EL2)”, the text that reads:

> When CNTHPS_CTL_EL2.ENABLE is 1, and TimerConditionMet is TRUE for the EL1 physical timer, the timer condition is met and all of the following are true:

is changed to read:

> When CNTHPS_CTL_EL2.ENABLE is 1, and TimerConditionMet is TRUE for the Secure EL2 physical timer, the timer condition is met and all of the following are true:

In section D24.10.10 “CNTHV_CVAL_EL2, Counter-timer Virtual Timer CompareValue Register (EL2)”, the text that reads:

> When CNTHV_CTL_EL2.ENABLE is 1, and TimerConditionMet is TRUE for the EL1 physical timer, the timer condition is met and all of the following are true:

is changed to read:

> When CNTHV_CTL_EL2.ENABLE is 1, and TimerConditionMet is TRUE for the EL2 virtual timer, the timer condition is met and all of the following are true:

In section D24.10.13 “CNTHVS_CVAL_EL2, Counter-timer Secure Virtual Timer CompareValue Register (EL2)”, the text that reads:

> When CNTHVS_CTL_EL2.ENABLE is 1, and TimerConditionMet is TRUE for the EL1 physical timer, the timer condition is met and all of the following are true:

is changed to read:

> When CNTHVS_CTL_EL2.ENABLE is 1, and TimerConditionMet is TRUE for the Secure EL2 virtual timer, the timer condition is met and all of the following are true:

In section D24.10.23 “CNTPS_CVAL_EL1, Counter-timer Physical Secure Timer CompareValue Register”, the text that reads:

> When CNTPS_CTL_EL2.ENABLE is 1, and TimerConditionMet is TRUE for the EL1 physical timer, the timer condition is met and all of the following are true:

is changed to read:

> When CNTPS_CTL_EL2.ENABLE is 1, and TimerConditionMet is TRUE for the EL1 secure physical timer, the timer condition is met and all of the following are true:

In section D24.10.26 “CNTV_CVAL_EL0, Counter-timer Virtual Timer CompareValue Register”, the text that reads:

> When CNTV_CTL_EL2.ENABLE is 1, and TimerConditionMet is TRUE for the EL1 physical timer, the timer condition is met and all of the following are true:

is changed to read:

> When CNTV_CTL_EL2.ENABLE is 1, and TimerConditionMet is TRUE for the EL1 virtual timer, the timer condition is met and all of the following are true:

2.104 D23683

In section D24.10.21 “CNTPOFF_EL2, Counter-timer Physical Offset Register”, under the heading ‘Purpose’, the text that reads:

> Holds the 64-bit physical offset. This is the offset for the AArch64 physical timers and counters when Enhanced Counter Virtualization is enabled.

is changed to read:

> Holds the 64-bit physical offset. This is the offset for the AArch64 EL1 physical timer and counter when Enhanced Counter Virtualization is enabled.

2.105 D23684

In section D24.10.30 “CNTVOFF_EL2, Counter-timer Virtual Offset Register”, under the heading ‘Configuration’, the following text:

> The virtual offset applies to:
>
> - Direct reads of CNTVCT_EL0.
> - Indirect reads of the virtual counter by the EL1 virtual timer. See ‘The Generic Timer in AArch64 state’.

- Indirect reads of the physical counter for timestamps generated by profiling logic. See 'The Statistical Profiling Extension' and 'AArch64 Self-hosted Trace'.

is changed to read:

The virtual offset applies to:

- Direct reads of CNTVCT_EL0.
- Indirect reads of the virtual counter by the EL1 virtual timer. See 'The Generic Timer in AArch64 state'.
- Indirect reads of the virtual counter for timestamps generated by profiling logic. See 'The Statistical Profiling Extension' and 'AArch64 Self-hosted Trace'.

2.106 D23688

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

`0x805E`, SVE_INT_VREDUCE_SPEC, Integer operation speculatively executed, SVE reduction

The counter counts each speculatively executed across-vector integer reduction operation counted by ASE_SVE_INT_VREDUCE_SPEC due to any of the following instructions:

...

is changed to read:

`0x805E`, SVE_INT_VREDUCE_SPEC, Integer operation speculatively executed, SVE reduction

The counter counts each speculatively executed across-vector and pairwise integer reduction operation counted by ASE_SVE_INT_VREDUCE_SPEC due to any of the following instructions:

...

2.107 C23700

In section D1.3.2.1 "Synchronous exception entry" the following rule:

R_{FKLWR}

For Instruction Abort or Data Abort exceptions caused by any of the following faults, when an exception is taken to EL2, an intermediate physical address (IPA) that characterizes the exception is captured in HPFAR_EL2:

- A Translation fault on a stage 2 translation.
- An Access Flag fault on a stage 2 translation.
- A stage 2 Address Size fault.

- A fault on stage 2 translation of an address accessed in a stage 1 translation table walk.
- A Granule Protection Fault (GPF) on an access for a stage 2 translation table.

For GPC exceptions due to a fault on an access for a stage 2 translation table walk, an IPA that characterizes the exception is captured in HPFAR_EL2.

For all other exceptions taken to EL2 using AArch64, HPFAR_EL2 is **UNKNOWN**.

is changed to read:

R_{FKLWR}

For Instruction Abort or Data Abort exceptions caused by any of the following faults, when an exception is taken to EL2, an intermediate physical address (IPA) that characterizes the exception is captured i PFAR_EL2:

- A Translation fault, Access Flag fault or Address Size fault on a stage 2 translation not on a stage 1 translation table walk.
- A Translation fault, Access flag fault, Address Size fault or Permission fault on stage 2 translation of an address accessed in a stage 1 translation table walk.
- A Granule Protection Fault (GPF) on an access for a stage 2 translation table.

For GPC exceptions due to a fault on an access for a stage 2 translation table walk, an IPA that characterizes the exception is captured in HPFAR_EL2.

For all other exceptions taken to EL2 using AArch64, HPFAR_EL2 is **UNKNOWN**.

In section D24.2.70 “HPFAR_EL2, Hypervisor IPA Fault Address Register”, under the heading “Configuration” the text that reads:

The HPFAR_EL2 is written for:

- Translation or Access faults in the second stage of translation.
- An abort in the second stage of translation performed during the translation table walk of a first stage translation, caused by a Translation fault, an Access flag fault, or a Permission fault.
- A stage 2 Address size fault.
- If FEAT_RME is implemented, a Granule Protection Check fault in the second stage of translation.

For all other exceptions taken to EL2, this register is **UNKNOWN**.

is changed to read:

The HPFAR_EL2 is written for:

- A Translation fault, Access Flag fault or Address Size fault on a stage 2 translation not on a stage 1 translation table walk.
- A Translation fault, Access flag fault, Address Size fault or Permission fault on stage 2 translation of an address accessed in a stage 1 translation table walk.

- If FEAT_RME is implemented, a Granule Protection Check fault in the second stage of translation.

For all other exceptions taken to EL2, this register is **UNKNOWN**.

2.108 C23701

In section D24.2.205 “VMPIDR_EL2, Virtualization Multiprocessor ID Register”, under the heading ‘Purpose’, the text that reads:

> Holds the value of the Virtualization Multiprocessor ID. This is the value returned by EL1 reads of MPIDR_EL1.

is changed to read:

> Holds the value of the Virtualization Multiprocessor ID. This is the value returned by EL1 reads of MPIDR_EL1, which in a multiprocessor system, provides an additional PE identification system.

Additionally, the text that reads:

> Affinity level 0. This is the affinity level that is most significant for determining PE behavior.
>
> Higher affinity levels are increasingly less significant in determining PE behavior.
>
> The assigned value of the MPIDR.{Aff2, Aff1, Aff0} or MPIDR_EL1.{Aff3, Aff2, Aff1, Aff0} set of fields of each PE must be unique within the system as a whole.

is changed to read:

> Affinity level 0. The value of the MPIDR.{Aff2, Aff1, Aff0} or the MPIDR_EL1.{Aff3, Aff2, Aff1, Aff0} set of fields of each PE must be unique within the system as a whole.

The equivalent changes are made in G8.2.170 “VMPIDR, Virtualization Multiprocessor ID Register”.

Furthermore, in section D13.4 “Attributability”, the note that reads:

> Note
>
> - In an implementation containing multiple PEs, each PE is identified by a unique affinity value reported by MPIDR_EL1.{Aff3, Aff2, Aff1, Aff0}, where the value of affinity level 0 is the most significant for determining the PE behavior, and the values of higher affinity levels are less significant. Affinity level 3 is only supported in AArch64 state.
> - An implementation is described as multithreaded when the lowest level of affinity consists of logical PEs that are implemented using a multithreading type approach. In this section, when referring to a multithreaded implementation, thread is used to mean processing elements with:
>   - MPIDR_EL1.MT or MPIDR.MT set to 1,
>   - Different values for affinity level 0.

- The same values for affinity level 1 and higher.

is changed to read:

Note

An implementation is described as multithreaded when the lowest level of affinity, as reported by MPIDR_EL1.{Aff3, Aff2, Aff1, Aff0}, consists of logical PEs that are implemented using a multithreading type approach.

In this section, when referring to a multithreaded implementation, thread is used to mean processing elements with:

- MPIDR_EL1.MT or MPIDR.MT set to 1.
- Different values for affinity level 0.
- The same values for affinity level 1 and higher.

2.109 D23712

In section D24.1.2.2 “Synchronization requirements for AArch64 System registers”, the text that reads:

Direct writes using the instructions in Table D23-2 require synchronization before software can rely on the effects of changes to the System registers to affect instructions appearing in program order after the direct write to the System register.

is changed to read:

Unless otherwise stated in the System register definition, direct writes to any System register require synchronization before software can rely on the effects of changes to the System registers to affect instructions appearing in program order after the direct write to the System register. This does not apply to Special-purpose registers.

2.110 D23713

In section D14.3 “Common event numbers”, the text that reads:

`0x8074`, SVE_PRED_SPEC, Operation speculatively executed, SVE predicated

The counter counts each speculatively executed SIMD data-processing, load, or store operation due to an instruction with a single Governing predicate operand that determines the Active elements.

> When FEAT_SME is implemented, both operations due to SVE instructions and operations due to SME instructions operating on the SVE Z vectors with at least one Governing predicate operand are counted.
>
> Note:
>
> For outer product instructions which are widening, predication is considered with respect to the input element size.

is changed to read:

> `0x8074`, SVE_PRED_SPEC, Operation speculatively executed, SVE or SME predicated
>
> The counter counts each speculatively executed predicated SVE or SME data-processing, load, or store operation.
>
> All of the following SVE operations are counted:
>
> - Data processing or load operations that write to one or more SVE Z vector destination registers under a Governing predicate using either zeroing or merging predication.
> - Predicated stores of one or more SVE Z vector registers.
>
> It is **IMPLEMENTATION DEFINED** whether data processing operations due to instructions with a single Governing predicate operand that determines the Active elements which do not write to any SVE Z vector destination registers using either zeroing or merging predication are counted. For example, INCP. When FEAT_SPE is implemented, Arm recommends this event is implemented consistently with the PRED field in the SVE data-processing format Operation Type packet.
>
> All speculatively executed SME data-processing, load, or store operations due to instructions with at least one Governing predicate operand that determines the Active elements are counted.
>
> Note:
>
> For outer product instructions which are widening, predication is considered with respect to the input element size.

In the same section, the text that reads:

> `0x8394`, SME_LDST_TILE_SPEC, SME predicated tile load/store
>
> The counter counts each speculatively executed operation that reads from or writes to memory counted by SME_LDST_REG_SPEC that was due to an instruction with at least one governing predicate which is targeting the ZA array.

is changed to read:

> `0x8394`, SME_LDST_TILE_SPEC, SME predicated tile load/store

> The counter counts each speculatively executed operation that reads from or writes to memory counted by SME_INST_SPEC that was due to an instruction with at least one governing predicate which is targeting the ZA array.

Additionally, the text that reads:

> `0x8074`, SVE_PRED_SPEC, Operation speculatively executed, SVE predicated
>
> The counter counts each speculatively executed SIMD data-processing, load, or store operation due to an instruction with a single Governing predicate operand that determines the Active elements.
>
> When FEAT_SME is implemented, both operations due to SVE instructions and operations due to SME instructions operating on the SVE Z vectors with at least one Governing predicate operand are counted.
>
> ...

is changed to read:

> `0x8074`, SVE_PRED_SPEC, Operation speculatively executed, SVE or SME predicated
>
> The counter counts each speculatively executed SVE data-processing, load, or store operation due to an instruction with at least one Governing predicate operand that determines the Active elements.
>
> When FEAT_SME is implemented, any speculatively executed SME data-processing, load, or store operation due to an instruction with at least one Governing predicate operand that determines the Active elements is also counted.
>
> ...

The equivalent changes are made in D14.3 “Common event numbers” in the following event descriptions:

- ‘`0x8075`, SVE_PRED_EMPTY_SPEC, Operation speculatively executed, SVE predicated with no active elements’.
- ‘`0x8076`, SVE_PRED_FULL_SPEC, Operation speculatively executed, SVE predicated with all active elements’.
- ‘`0x8077`, SVE_PRED_PARTIAL_SPEC, Operation speculatively executed, SVE predicated with partially active elements’.
- ‘`0x8078`, SVE_UNPRED_SPEC, Operation speculatively executed, SVE unpredicated’.
- ‘`0x8079`, SVE_PRED_NOT_FULL_SPEC, Operation speculatively executed, SVE predicated with at least one inactive element’.

2.111 D23734

In section A2.3.3 “The Armv9.2 architecture extension”, under the heading ‘FEAT_RME, Realm Management Extension’, the text that reads:

> If FEAT_RME is implemented, then FEAT_AA64EL3, FEAT_AA64EL2, FEAT_PMULL, and FEAT_RNG or FEAT_RNG_TRAP are implemented.
>
> When FEAT_RME and FEAT_AES or FEAT_SHA1 are implemented, FEAT_PMULL, FEAT_AES, FEAT_SHA3, FEAT_SHA256, and FEAT_SHA512 are implemented.

is changed to read:

> If FEAT_RME is implemented, then FEAT_AA64EL3, FEAT_AA64EL2, and FEAT_RNG or FEAT_RNG_TRAP are implemented.
>
> When FEAT_RME and FEAT_AES or FEAT_SHA1 are implemented, FEAT_PMULL, FEAT_AES, FEAT_SHA3, FEAT_SHA256, and FEAT_SHA512 are implemented.

2.112 D23741

In section D1.3.2.1 “Synchronous exception entry”, the informational statement I_{YRNGX} that reads:

> If FEAT_RME is implemented, then when GPCCR_EL3.GPC is 0, Granule Protection Checks (GPCs) on accesses to Physical Address (PA) space are enabled and might result in GPC faults.

is changed to read:

> If FEAT_RME is implemented, then when GPCCR_EL3.GPC is 1, Granule Protection Checks (GPCs) on accesses to Physical Address (PA) space are enabled and might result in GPC faults.

In the same section, the informational statement I_{XWVCY} that reads:

> For GPFs at EL2, EL1, or EL0 when SCR_EL3.GPF is 0, EL3 firmware can choose to delegate the resulting GPC exception to a lower Exception level, as an Instruction Abort exception or Data Abort exception.

is changed to read:

> For GPFs at EL2, EL1, or EL0 when SCR_EL3.GPF is 1, EL3 firmware can choose to delegate the resulting GPC exception to a lower Exception level, as an Instruction Abort exception or Data Abort exception.

In the same section, the informational statement I_{YHXKR} that reads:

> When GPCCR_EL3.GPCP is 0, the PE can omit GPCs on fetches of translation table entries that are Table descriptors for stage 2 translation table walks, for a performance optimization. Enabling this optimization is dependent on the security model of the system.

is changed to read:

> When GPCCR_EL3.GPCP is 1, the PE can omit GPCs on fetches of translation table entries that are Table descriptors for stage 2 translation table walks, for a performance optimization. Enabling this optimization is dependent on the security model of the system.

In the same section, the rule R_{RWGJH} that reads:

> If GPCCR_EL3.GPCP is 0 and the PE omits a GPC when fetching a translation table entry that is a Table descriptor for a stage 2 translation table walk, then when the entry is processed: ...

is changed to read:

> If GPCCR_EL3.GPCP is 1 and the PE omits a GPC when fetching a translation table entry that is a Table descriptor for a stage 2 translation table walk, then when the entry is processed: ...

2.113 D23744

In section D24.2.206 “VNCR_EL2, Virtual Nested Control Register”, the text that reads:

> RESS, bits [63:57]
>
> Reserved, Sign extended. If the bits marked as RESS do not all have the same value, then there is a **CONSTRAINED UNPREDICTABLE** choice between:
>
> - Generating an EL2 translation regime Translation abort on use of the VNCR_EL2 register. If FEAT_D128 is implemented:
> - If the virtual address space for EL2 supports 56 bits, bits[63:57] of VNCR_EL2 are treated as the same value as bit[56] for all purposes other than reading back the register.
> - If the virtual address space for EL2 supports 56 bits, bits[63:57] of VNCR_EL2 are treated as the same value as bit[56].
> - If the virtual address space for EL2 supports 52 bits, bits[63:53] of VNCR_EL2 are treated as the same value as bit[52] for all purposes other than reading back the register.
> - If the virtual address space for EL2 supports 52 bits, bits[63:53] of VNCR_EL2 are treated as the same value as bit[52].
> - Bits[63:49] of VNCR_EL2 are treated as the same value as bit[48] for all purposes other than reading back the register.
> - Bits[63:49] of VNCR_EL2 are treated as the same value as bit[48] for all purposes.
>
> Where the EL2 translation regime has upper and lower address ranges, bit[56] is used to select between those address ranges to determine the number of bits supported by the address space.
>
> BADDR, bits [56:12]

> Base Address. If the virtual address space for EL2 does not support more than 48 bits, then bits [56:49] are RESS. If the virtual address space for EL2 does not support more than 52 bits, then bits [56:53] are RESS
>
> When a register read/write is transformed to be a Load or Store, the address of the load/store is to SignOffset(VNCR_EL2.BADDR:Offset<11:0>, 64).
>
> The reset behavior of this field is:
>
> - On a Warm reset, this field resets to an architecturally **UNKNOWN** value.

is changed to read:

> BADDR, bits [63:12]
>
> Base Address. When a register read/write is transformed to be a Load or Store, the address of the load/store is to VNCR_EL2.BADDR:Offset<11:0>.
>
> Bits[63:n] are RESS where n is one of the following:
>
> - If FEAT_LVA3 is implemented, n is 57.
> - If FEAT_LVA is implemented, but FEAT_LVA3 is not implemented, n is 53.
> - If FEAT_LVA is not implemented, n is 49.
>
> If the bits marked as RESS do not all have the same value as bit[n-1], then one of the following **CONSTRAINED UNPREDICTABLE** choices applies:
>
> - When translating the transformed register read/write, a Translation fault is generated.
> - The bits behave as the corresponding RESS for all purposes other than reading back the register.
> - The bits behave as the corresponding RESS for all purposes.
>
> The reset behavior of this field is:
>
> - On a Warm reset, this field resets to an architecturally **UNKNOWN** value.

2.114 C23753

In section D1.3.2 “Exception entry”, under rule R_{DQXFW}, the text that reads:

> When an exception is taken to an Exception level, ELx, that is using AArch64 state, all the following occur:
>
> - The contents of PSTATE immediately before the exception was taken is written to SPSR_ELx.
> - ...

is changed to read:

> When an exception is taken to an Exception level, ELx, that is using AArch64 state, all the following occur:
>
> - Fields in SPSR_ELx are written according to their description in SPSR_ELx, based on the values in PSTATE immediately before the exception was taken.
> - ...

In section D1.3.4.1 “Legal exception returns from AArch64 state”, under rule R_{BWCFK}, the text that reads:

> On a legal exception return from an Exception level, ELx, all of the following occur:
>
> - ...
> - The contents of PSTATE are set to the values held in SPSR_ELx.
> - ...

is changed to read:

> On a legal exception return from an Exception level, ELx, all of the following occur:
>
> - ...
> - Fields in PSTATE that have a corresponding field in SPSR_ELx are written according to their description in SPSR_ELx.
> - ...

Additionally, in section D1.3.2 “Exception entry”, the informational statement I_{X0001} is added:

> I_{X0001}
>
> When an exception is taken to an Exception level that is using AArch64 state, the following PSTATE fields are unchanged:
>
> - If FEAT_DIT is implemented, PSTATE.DIT.
> - If FEAT_SME is implemented, PSTATE.{SM,ZA}.

2.115 C23766

In section B2.3 “Ordering requirements defined by the formal concurrency model”, subsection B2.3.1 “Basic definitions”, the text that reads:

> Same Location An Effect E_1 and an Effect E_2 are to the Same Location if one of the following applies: ...
>
> - All of the following apply:
>   - An Implicit TTD Memory Read Effect E_3 is Translation-intrinsically-before E_1.
>   - The output address of the TTD read by E_3 is the same as the output address of the TTD read by an Implicit TTD Memory Read Effect E_4.

- An Implicit TTD Memory Read Effect E_4 is Translation-intrinsically-before E_2.
- E_1 and E_2 have the Same Low-order Bits.

is changed to read:

Same Location An Effect E_1 and an Effect E_2 are to the Same Location if one of the following applies: ... - All of the following apply: - E_1 is an MMU Fault Effect. - It is not the case that E_1 is an MMU Translation Fault Effect. - E_3 is Translation-intrinsically-before E_1. - E_3 is an Implicit TTD Memory Read Effect. - E_3 and E_4 have the Same Output Address. - E_4 is an Implicit TTD Memory Read Effect. - E_4 is Translation-intrinsically-before E_2. - E_1 and E_2 have the Same Low-order Bits. - All of the following apply: - E_3 is Translation-intrinsically-before E_1. - E_3 is an Implicit TTD Memory Read Effect. - E_3 and E_4 have the Same Output Address. - E_4 is an Implicit TTD Memory Read Effect. - E_4 is Translation-intrinsically-before E_2. - E_1 and E_2 have the Same Low-order Bits. - E_2 is an MMU Fault Effect. - It is not the case that E_2 is an MMU Translation Fault Effect.

2.116 D23768

In section B1.5.6 “About PSTATE.DIT”, the text that reads:

- If SVE2 is not implemented, the data independent timing control introduced by FEAT_DIT does not affect the timing properties of SVE instructions.
- For SVE and SVE2 predicated instructions, it is the programmer’s responsibility to use a Governing predicate that does not reflect the values of the data being operated on.

is changed to read:

- If FEAT_SVE2 and FEAT_SME are not implemented, the data independent timing control introduced by FEAT_DIT does not affect the timing properties of SVE instructions.
- If FEAT_SVE2 or FEAT_SME is implemented, the data independent timing control introduced by FEAT_DIT affects the timing properties of all SVE and SME instructions that honor PSTATE.DIT.
- For SVE and SME predicated instructions, it is the programmer’s responsibility to use a Governing predicate that does not reflect the values of the data being operated on.

2.117 D23772

In section C6.2.50 “CAS, CASA, CASAL, CASL”, the text that reads:

- CASA and CASAL load from memory with acquire semantics.

is changed to read:

- If the destination register is not one of WZR or XZR, CASA and CASAL load from memory with acquire semantics.

Under the heading "Decode for all variants in this encoding", the pseudocode that reads:

```
...
constant boolean acquire = L == '1';
...
```

is changed to read:

```
...
constant boolean acquire = L == '1' && t != 31;
...
```

Equivalent changes are made in the following sections:

- C6.2.51 "CASB, CASAB, CASALB, CASLB".
- C6.2.52 "CASH, CASAH, CASALH, CASLH".

In section C6.2.291 "RCWCAS, RCWCASA, RCWCASL, RCWCASAL", the text that reads:

- RCWCASA and RCWCASAL load from memory with acquire semantics.

is changed to read:

- If the destination register is not XZR, RCWCASA and RCWCASAL load from memory with acquire semantics.

In the same section, under the heading "Decode for all variants in this encoding", the pseudocode that reads:

```
...
constant boolean acquire = A == '1';
...
```

is changed to read:

```
...
constant boolean acquire = A == '1' && t != 31;
...
```

Equivalent changes are made in the following sections:

- C6.2.293 "RCWCLR, RCWCLRA, RCWCLRL, RCWCLRAL".
- C6.2.295 "RCWSCAS, RCWSCASA, RCWSCASL, RCWSCASAL".
- C6.2.297 "RCWSCLR, RCWSCLRA, RCWSCLRL, RCWSCLRAL".
- C6.2.299 "RCWSET, RCWSETA, RCWSETL, RCWSETAL".
- C6.2.301 "RCWSSET, RCWSSETA, RCWSSETL, RCWSSETAL".

- C6.2.303 “RCWSSWP, RCWSSWPA, RCWSSWPL, RCWSSWPAL”.
- C6.2.305 “RCWSWP, RCWSWPA, RCWSWPL, RCWSWPAL”.

In section C6.2.159 “LDAPR”, the text that reads:

> The instruction has memory ordering semantics as described in Load-Acquire, Load-AcquirePC, and Store-Release, except that:
>
> - There is no ordering requirement, separate from the requirements of a Load-AcquirePC or a Store-Release, created by having a Store-Release followed by a Load-AcquirePC instruction.
> - The reading of a value written by a Store-Release by a Load-AcquirePC instruction by the same observer does not make the write of the Store-Release globally observed.
>
> This difference in memory ordering is not described in the pseudocode.

is changed to read:

> If the destination register is not XZR or WZR, LDAPR loads from memory with AcquirePC semantics.
>
> For more information about memory-ordering semantics, see Load-Acquire, Load-AcquirePC and Store-Release.

Under the heading “Post-index”, subheading “Decode for all variants of this encoding”, the pseudocode that reads:

```
...
constant integer offset = 1 << UInt(size);
constant boolean tagchecked = TRUE;

...
```

is changed to read:

```
...
constant integer offset = 1 << UInt(size);
constant boolean acquirepc = t != 31;
constant boolean tagchecked = TRUE;

...
```

Under the heading “No offset”, subheading “Decode for all variants of this encoding”, the pseudocode that reads:

```
...
constant integer datasize = elsize;
constant boolean tagchecked = n != 31;
```

is changed to read:

```
...
```

```
constant integer datasize = elsize;
constant boolean acquirepc = t != 31;
constant boolean tagchecked = n != 31;
```

Under the heading “Operation”, the pseudocode that reads:

```
...
constant integer dbytes = datasize DIV 8;
constant AccessDescriptor accdesc = CreateAccDescLDAcqPC(tagchecked);
if n == 31 then
...
```

is changed to read:

```
...
constant integer dbytes = datasize DIV 8;
constant AccessDescriptor accdesc = CreateAccDescLDAcqPC(tagchecked, acquirepc);

if n == 31 then
...
```

Equivalent changes are made in the following sections:

- C6.2.160 “LDAPRB”.
- C6.2.161 “LDAPRH”.
- C6.2.162 “LDAPUR”.
- C6.2.163 “LDAPURB”.
- C6.2.164 “LDAPURH”.
- C6.2.165 “LDAPURSB”.
- C6.2.167 “LDAPURW”.

In section C6.2.168 “LDAR”, the text that reads:

> Load-acquire register
>
> This instruction derives an address from a base register value, loads a 32-bit word or 64-bit doubleword from memory, and writes it to a register. The instruction also has memory ordering semantics as described in Load-Acquire, Store-Release. For information about addressing modes, see Load/Store addressing modes.
>
> Note:
>
> For this instruction, if the destination is WZR/XZR, it is impossible for software to observe the presence of the acquire semantic other than its effect on the arrival at endpoints.

is changed to read:

> Load-acquire register

> This instruction derives an address from a base register value, loads a 32-bit word or 64-bit doubleword from memory, and writes it to a register. For information about addressing modes, see Load/Store addressing modes.
>
> If the destination register is not WZR or XZR, LDAR loads from memory with Acquire semantics.
>
> For more information about memory ordering semantics, see Load-Acquire, Load-AcquirePC, Store-Release.

Under the heading “Decode for all variants of this encoding”, the pseudocode that reads:

```
...

constant boolean tagchecked = TRUE;
```

is changed to read:

```
...

constant boolean acquire = t != 31;
constant boolean tagchecked = TRUE;
```

Under the heading “Operation”, the pseudocode that reads:

```
bits(64) address;
constant AccessDescriptor accdesc = CreateAccDescAcqRel(MemOp_LOAD, tagchecked);

if n == 31 then
...
```

is changed to read:

```
bits(64) address;
constant AccessDescriptor accdesc = CreateAccDescAcqRel(MemOp_LOAD, tagchecked,
 acquire);

if n == 31 then
...
```

Equivalent changes are made in the following sections:

- C6.2.169 “LDARB” .
- C6.2.170 “LDARH”.

In section C6.2.171 “LDAXP” the text that reads:

> Load-acquire exclusive pair of registers
>
> This instruction derives an address from a base register value, loads two 32-bit words or two 64-bit doublewords from memory, and writes them to two registers. For information on single-copy atomicity and alignment requirements, see Requirements for single-copy atomicity and Alignment of data accesses. The PE marks the physical address being accessed as an exclusive access.

> This exclusive access mark is checked by Store Exclusive instructions. See Synchronization and semaphores. The instruction also has memory ordering semantics, as described in Load-Acquire, Store-Release. For information about addressing modes, see Load/Store addressing modes.

is changed to read:

> Load-acquire exclusive pair of registers
>
> This instruction derives an address from a base register value, loads two 32-bit words or two 64-bit doublewords from memory, and writes them to two registers. For information on single-copy atomicity and alignment requirements, see Requirements for single-copy atomicity and Alignment of data accesses. The PE marks the physical address being accessed as an exclusive access. This exclusive access mark is checked by Store Exclusive instructions. See Synchronization and semaphores. For information about addressing modes, see Load/Store addressing modes.
>
> If the destination registers are not both WZR or not both XZR, LDAXP loads from memory with Acquire semantics.
>
> For more information about memory ordering semantics, see Load-Acquire, Load-AcquirePC, Store-Release.

Under the heading “Decode for all variants of this encoding”, the pseudocode that reads:

```
...

constant boolean acqrel = TRUE;
constant boolean tagchecked = n != 31;

...
```

is changed to read:

```
...

constant boolean acqrel = t != 31 && t1 != 31;
constant boolean tagchecked = n != 31;

...
```

Equivalent change is made to section C6.2.184 “LDIAPP”.

In section C6.2.172 “LDAXR”, the text that reads:

> Load-acquire exclusive register
>
> This instruction derives an address from a base register value, loads a 32-bit word or 64-bit doubleword from memory, and writes it to a register. The memory access is atomic. The PE marks the physical address being accessed as an exclusive access. This exclusive access mark is checked by Store Exclusive instructions. See Synchronization and semaphores. The instruction also has memory ordering semantics as described in Load-Acquire, Store-Release. For information about addressing modes, see Load/Store addressing modes.

is changed to read:

> Load-acquire exclusive register
>
> This instruction derives an address from a base register value, loads a 32-bit word or 64-bit doubleword from memory, and writes it to a register. The memory access is atomic. The PE marks the physical address being accessed as an exclusive access. This exclusive access mark is checked by Store Exclusive instructions. See Synchronization and semaphores. For information about addressing modes, see Load/Store addressing modes.
>
> If the destination register is not WZR or XZR, LDAXR loads from memory with Acquire semantics.
>
> For more information about memory ordering semantics, see Load-Acquire, Load-AcquirePC, Store-Release.

Under the heading “Decode for all variants of this encoding”, the pseudocode that reads:

```
...

constant boolean acqrel = TRUE;
constant boolean tagchecked = n != 31;
```

is changed to read:

```
...

constant boolean acqrel = t != 31;
constant boolean tagchecked = n != 31;
```

Equivalent changes are made in the following sections:

- C6.2.173 “LDAXRB”.
- C6.2.174 “LDAXRH”.

In section C6.2.185 “LDLAR”, the text that reads:

> Load LOAcquire register
>
> This instruction loads a 32-bit word or 64-bit doubleword from memory, and writes it to a register. The instruction also has memory ordering semantics as described in Load LOAcquire, Store LORelease. For information about addressing modes, see Load/Store addressing modes.
>
> Note:
>
> For this instruction, if the destination is WZR/XZR, it is impossible for software to observe the presence of the acquire semantic other than its effect on the arrival at endpoints.

is changed to read:

> Load LOAcquire register

> This instruction loads a 32-bit word or 64-bit doubleword from memory, and writes it to a register. For information about addressing modes, see Load/Store addressing modes.
>
> If the destination register is not WZR or XZR, LDLAR loads from memory with Acquire semantics.
>
> For more information about memory ordering semantics, see Load LOAcquire, Store LORelease and Load-Acquire, Load-AcquirePC, Store-Release.

In addition, under the heading “Decode for all variants of this encoding”, the pseudocode that reads:

```
...

constant boolean acqrel = TRUE;
constant boolean tagchecked = n != 31;
```

is changed to read:

```
...

constant boolean acqrel = t != 31;
constant boolean tagchecked = n != 31;
```

Under the heading “Operation”, the pseudocode that reads:

```
bits(64) address;
constant integer dbytes = elsize DIV 8;

constant AccessDescriptor accdesc = CreateAccDescLOR(MemOp_LOAD, tagchecked);

if n == 31 then
```

is changed to read:

```
bits(64) address;
constant integer dbytes = elsize DIV 8;

constant AccessDescriptor accdesc = CreateAccDescLOR(MemOp_LOAD, tag checked,
 acqrel);

if n == 31 then
```

Equivalent changes are made to the following sections:

- C6.2.186 “LDLARB”.
- C6.2.187 “LDLARH”.

2.118 C23775

The following text is added to B2.3.7 “Ordering relations”:

> An Effect E_1 is ETS-ordered-before an Effect E_2 if one of the following applies:
>
> - All of the following apply:
>   - FEAT_ETS2 is implemented.
>   - E_1 is an Explicit Memory Effect.
>   - E_1 appears in program order before E_3.
>   - E_3 is a TLBUncacheable Fault Effect.
>   - E_2 is Translation-intrinsically-before E_3.
>   - E_2 is an Implicit TTD Memory Read Effect.
> - All of the following apply:
>   - FEAT_ETS3 is implemented.
>   - E_1 is an Explicit Memory Effect.
>   - E_1 appears in program order before E_3.
>   - E_3 is an MMU Fault Effect.
>   - E_2 is Translation-intrinsically-before E_3.
>   - E_2 is an Implicit TTD Memory Read Effect.
>
> Note:
>
> The ETS-ordered-before relation does not apply when E_1 is generated by an Advanced SIMD, floating-point, SVE, or SME instruction. It also does not apply when E_2 is an Implicit TTD Memory Read Effect generated for an instruction fetch.

The following text is added to D8.2.6.1 “Ordering of memory accesses from translation table walks”, under the rule R_{NNFPF}:

> If FEAT_ETS2 is implemented, then an Effect E_1 is Ordered-before an Effect E_2 when all of the following apply:
>
> - E_1 is not generated by an Advanced SIMD, floating-point, SVE, or SME instruction.
> - E_2 is not generated for an instruction fetch.
> - E_1 is an Explicit Memory Effect.
> - E_1 appears in program order before E_3.
> - E_3 is a TLBUncacheable Fault Effect.
> - E_2 is Translation-intrinsically-before E_3.
> - E_2 is an Implicit TTD Memory Read Effect.

The following text is added to the same section under the rule R_{QVWHP}:

If FEAT_ETS3 is implemented, then an Effect E_1 is Ordered-before an Effect E_2 when all of the following apply:

- E_1 is not generated by an Advanced SIMD, floating-point, SVE, or SME instruction.
- E_1 is an Explicit Memory Effect.
- E_2 is not generated for an instruction fetch.
- E_1 appears in program order before E_3.
- E_3 is an MMU Fault Effect.
- E_2 is Translation-intrinsically-before E_3.
- E_2 is an Implicit TTD Memory Read Effect.

The following text is added to E2.4 “Ordering of translation table walks”:

If FEAT_ETS2 is implemented, then an Effect E_1 is Ordered-before an Effect E_2 when all of the following apply:

- E_1 is not generated by an Advanced SIMD or floating-point instruction.
- E_2 is not generated for an instruction fetch.
- E_1 is an Explicit Memory Effect.
- E_1 appears in program order before E_3.
- E_3 is a TLBUncacheable Fault Effect.
- E_2 is Translation-intrinsically-before E_3.
- E_2 is an Implicit TTD Memory Read Effect.

The following text is added to the same section:

If FEAT_ETS3 is implemented, then an Effect E_1 is Ordered-before an Effect E_2 when all of the following apply:

- E_1 is not generated by an Advanced SIMD, floating-point, SVE, or SME instruction.
- E_2 is not generated for an instruction fetch.
- E_1 is an Explicit Memory Effect.
- E_1 appears in program order before E_3.
- E_3 is an MMU Fault Effect.
- E_2 is Translation-intrinsically-before E_3.
- E_2 is an Implicit TTD Memory Read Effect.

2.119 D23778

In section B2.2.1 “Requirements for single-copy atomicity”, the text that reads:

> For explicit memory effects generated from an Exception level the following rules apply:
>
> ...
>
> - The reads and writes of the two words or two doublewords accessed by CASP instructions are single-copy atomic at the size of the two words or doublewords.

is changed to read:

> For explicit memory effects generated from an Exception level the following rules apply:
>
> ...
>
> - The read and write generated by an Atomic instruction is single-copy atomic at the overall size of the data access.

In section B2.8.2.1.2 “Load-Exclusive/ Store-Exclusive and Atomic instructions”, 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 overall size of the access.

2.120 D23794

In section D24.2.164 “SCTLR2_EL1, System Control Register (EL1)”, the text that reads:

> If FEAT_SVE is implemented, SCTLR2_EL1.EnADERR is 0, and the access generating the External abort is due to any Active element of an SVE Non-fault vector load instruction or an Active element that is not the first Active element of an SVE First-fault vector load instruction, then no exception is generated and the External abort is reported in the FFR.

is changed to read:

> If FEAT_SVE is implemented, SCTLR2_EL1.EnANERR is 0, and the access generating the External abort is due to any Active element of an SVE Non-fault vector load instruction or an Active element that is not the first Active element of an SVE First-fault vector load instruction, then no exception is generated and the External abort is reported in the FFR.

The equivalent changes are made in the following sections:

- D24.2.165 “SCTLR2_EL2, System Control Register (EL2)”
- D24.2.166 “SCTLR2_EL3, System Control Register (EL3)”

2.121 D23796

In section D24.7.9 “PMSFCR_EL1, Sampling Filter Control Register”, in the description of PMSFCR_EL1.FT, the text that reads:

> FT, bit [1]
>
> Filter by operation type. The filter is the logical OR of the ST, LD and B bits. For example, if LD and ST are both set, both load and store operations are recorded.
>
> The full description of filtering is provided in the section D17.5.1 “Filtering by operation type,” and does not need to be repeated here.

is changed to read:

> FT, bit [1]
>
> Filter by type. The filter is controlled by the {SIMD, FP, ST, LD, B} and {SIMDm, FPm, STm, LDm, Bm} fields.
>
> For more information, see Filtering by operation type.

2.122 D23797

In section D1.3.5.3 “Granule Protection Check faults”, under the rule R_{JXSRX}, the text that reads:

> When the PE is in Debug state and EDSCR.SDD is 0, SCR_EL3.GPF is treated as 0, and the following GPC faults are treated as a GPF for the purposes of causing an exception:
>
> - GPT walk faults.
> - GPT address size faults.
> - Synchronous External abort on GPT fetch.

is changed to read:

When the PE is in Debug state and EDSCR.SDD is 1, SCR_EL3.GPF is treated as 0, and the following GPC faults are treated as a GPF for the purposes of causing an exception:

- GPT walk faults.
- GPT address size faults.
- Synchronous External abort on GPT fetch.

In section J1.3 “Shared pseudocode”, in the function `ReportAsGPCException()`, the code that reads:

```
boolean ReportAsGPCException(FaultRecord fault)
  ...
  case fault.gpcf.gpf of
    when GPCF_Walk return TRUE;
    when GPCF_AddressSize return TRUE;
    when GPCF_EABT return TRUE;
    when GPCF_Fail return SCR_EL3.GPF == '1' && PSTATE.EL != EL3;
```

is changed to read:

```
boolean ReportAsGPCException(FaultRecord fault)
  ...
  if Halted() && EDSCR.SDD == '1' then
    return FALSE;
  case fault.gpcf.gpf of
    when GPCF_Walk return TRUE;
    when GPCF_AddressSize return TRUE;
    when GPCF_EABT return TRUE;
    when GPCF_Fail return SCR_EL3.GPF == '1' && PSTATE.EL != EL3;
```

In the function `AArch64.DataAbort()`, the code that reads:

```
else
  route_to_el2 = (EL2Enabled() && PSTATE.EL IN {EL0, EL1} &&
    (HCR_EL2.TGE == '1' ||
     (IsFeatureImplemented(FEAT_RME) &&
      fault.gpcf.gpf == GPCF_Fail &&
      HCR_EL2.GPF == '1') ||
    ...
```

is changed to read:

```
else
  route_to_el2 = (EL2Enabled() && PSTATE.EL IN {EL0, EL1} &&
    (HCR_EL2.TGE == '1' ||
     (IsFeatureImplemented(FEAT_RME) &&
      fault.gpcf.gpf != GPCF_None &&
      HCR_EL2.GPF == '1') ||
    ...
```

The equivalent change is made in the function `AArch64.InstructionAbort()` for consistency.

In section H2.4.7.1 “Generating exceptions when in Debug state”, the text that reads:

In Debug state:

...

- If FEAT_RME is implemented and EDSCR.SDD is 1, SCR_EL3.GPF is treated as 0, regardless of its actual state, other than for the purpose of a direct read.

is changed to read:

In Debug state:

...

- If FEAT_RME is implemented and EDSCR.SDD is 1:
  - SCR_EL3.GPF is treated as 0, other than for the purpose of a direct read.
  - All GPC faults are treated as a GPF for the purposes of causing an exception.

2.123 D23807

In section D20.1 “About the RAS Extension”, in the example D20-1 “Minimal implementation of RAS Extension”, the text that reads:

Any error signaled to the PE generates an asynchronous SError exception, and on taking the SError exception:

- The PE error state is recorded as Uncontainable (UC).
- ESR_ELx.FnV is set to 1, meaning FAR_ELx is **UNKNOWN** and not valid.

FEAT_PFAR is not implemented. See Taking error exceptions.

is changed to read:

Any error signaled to the PE generates an asynchronous SError exception, and on taking the SError exception:

- The PE error state is recorded as Uncontainable (UC).
  - If FEAT_RASv2 is implemented, then ESR_ELx.VFV is set to 0, meaning FAR_ELx is **UNKNOWN** and not valid. If FEAT_RASv2 is not implemented, then FAR_ELx is always **UNKNOWN** and not valid on taking an SError exception.
  - FEAT_PFAR is not implemented, meaning ESR_ELx.PFV is set to 0.

See Taking error exceptions.

2.124 C23808

In section D17.3.1 “Operation sampling”, the text that reads:

1. A sampling interval is written to PMSICR_EL1.COUNT by software. The interval is measured in operations.

is changed to read:

> 1. Software writes a sampling interval to PMSIRR_EL1.COUNT, and sets PMSICR_EL1 to zero. The interval is measured in operations.

2.125 D23812

In section D1.3.5.7 “Memory Copy and Memory Set exceptions”, under the rule I_{PRXVQ}, the text that reads:

> The information in the ESR_ELx.ISS field is sufficient to allow the diagnosis of the reason for a Memory Copy or Memory Set exception being generated and to allow a generic emulation of the memory copy or memory set.

is changed to read:

> The information in SPSR_ELx.C (or, equivalently, the ESR_ELx.ISS.FormatOption field), SPSR_ELx.N and the sign of the value in the Xn register is sufficient to determine the current format of the Xd, Xs and Xn registers, and perform any required reformatting, before restarting the memory copy or memory set. However, it is not required to use FEAT_MOPS instructions and the memory copy or memory set could alternatively be restarted using a software implementation.

2.126 R23867

In section D18.2.7 “Operation Type packet”, in the description of the PRED field, the text that reads:

PRED	Description
0b0	Not predicated.
0b1	Predicated SVE operation. The operation is an SVE operation that writes to a vector destination register under a Governing predicate using either zeroing or merging predication.

is changed to read:

PRED	Description
0b0	Not predicated.
0b1	Predicated SVE data-processing operation.

> Predicated SVE data-processing operation means an SVE data-processing operation that writes to one or more SVE Z vector destination registers under a Governing predicate using either zeroing or merging predication. It is **IMPLEMENTATION DEFINED** whether this includes data-processing operations due to instructions with a single Governing predicate operand that determines the Active elements which do not write to any SVE Z vector destination registers using either zeroing or merging predication are counted. For example, INCP.

In section D18.2.6 “Events packet” (issue L.a), in the description of the E[18] field, the text that reads:

> E[18], byte 2 bit [2], when SZ == `0b10` or SZ == `0b11`
>
> Empty predicate.
>
> ...

is changed to read:

> E[18], byte 2 bit [2], when SZ == `0b10` or SZ == `0b11`
>
> Empty predicate.
>
> ...
>
> For a sampled SVE operation, it is **IMPLEMENTATION DEFINED** whether this field is valid or reads as zero when the PRED field in the Operation Type field is 0.

The equivalent change is made to the E[17] “Partial or empty predicate”field.

2.127 D23823

In section D18.2.6 “Events packet”, the following text is added to the description of E[19] “Level 2 Data cache access”:

> This event is optional. If this event is implemented, then it is further **IMPLEMENTATION DEFINED** and might be **UNPREDICTABLE** whether a store can finish execution before this event is generated, meaning that, for stores, this event is not recorded and this bit reads-as-zero. If this event is not implemented, then this bit reads-as-zero.

Similar text is added to each of the following events: - E[20] “Level 2 Data cache miss”. - E[21] “Cached data modified”. - E[22] “Recently fetched”. - E[23] “Data snooped”.

2.128 C23842

In section B2.3.6 “Basic definitions”, the definition of *Local write or MMU Fault successor* that reads:

> Local write or MMU Fault successor
>
> An Effect E_2 is a Local write or MMU Fault successor of an Effect E_1 if one of the following applies:
>
> - All of the following apply:
> - One of the following applies:

        - E_1 is an Explicit Memory Effect.
        - E_1 is an Implicit Tag Memory Read Effect.
    - E_1 appears in program order before E_2.
    - E_1 and E_2 are to the Same Location.
    - E_2 is an Explicit Memory Write Effect.
- All of the following apply:
    - E_1 is an Explicit Memory Effect.
    - E_1 appears in program order before E_2.
    - E_1 and E_2 are to the Same Location.
    - E_2 is an MMU Fault Effect.

is changed to read:

Local write successor

An Effect E_2 is a Local write successor of an Effect E_1 if one of the following applies:

- All of the following apply:
    - One of the following applies:
        - E_1 is an Explicit Memory Effect.
        - E_1 is an Implicit Tag Memory Read Effect.
    - E_1 appears in program order before E_2.
    - E_1 and E_2 are to the Same Location.
    - E_2 is an Explicit Memory Write Effect.
- All of the following apply:
    - E_1 is an Implicit TTD Memory Read Effect.
    - E_1 appears in program order before E_2.
    - E_1 and E_2 are to the Same Location.
    - One of the following applies:
        - E_2 is an Explicit Memory Write Effect.
        - E_2 is a Hardware Update Effect.

Local MMU Fault successor

An Effect E_2 is a Local MMU Fault successor of an Effect E_1 if all of the following apply:

- E_1 is an Explicit Memory Effect.
- E_1 appears in program order before E_2.
- E_1 and E_2 are to the Same Location.
- E_2 is an MMU Fault Effect.

Additionally, in section B2.3.6, the definition of 'Locally-ordered-before' that reads:

> Locally-ordered-before
>
> An Effect E_1 is Locally-ordered-before an Effect E_2 if one of the following applies:
>
> ...
> - E_2 is a Local write or MMU Fault successor of E_1.
> - All of the following apply:
>   - E_3 is a Local write or MMU Fault successor of E_1.
>   - E_3 belongs to the same single-copy-atomic class as E_2. ...

is changed to read:

> Locally-ordered-before
>
> An Effect E_1 is Locally-ordered-before an Effect E_2 if one of the following applies:
>
> ...
> - E_2 is a Local write successor of E_1.
> - All of the following apply:
>   - E_3 is a Local write successor of E_1.
>   - E_3 belongs to the same single-copy-atomic class as E_2.
> - E_2 is a Local MMU Fault successor of E_1.
>
> ...

2.129 C23844

In section D13.1 "About the Performance Monitors", the text that reads:

> FEAT_PMUv3 requires that an implementation includes the following common events:

is changed to read:

> When at least one event counter is implemented, FEAT_PMUv3 requires that an implementation includes the following common events:

2.130 C23845

In section D24.2.167 “SCTLR_EL1, System Control Register (EL1)”, in the description of the field nAA, bit [6], the text that reads:

> Non-aligned access. This bit controls generation of Alignment faults at EL1 and EL0 under certain conditions.
>
> The following instructions generate an Alignment fault if all bytes being accessed are not within a single 16-byte quantity, aligned to 16 bytes for access:
>
> - LDAPR, LDAPRH, LDAPUR, LDAPURH, LDAPURSH, LDAPURSW, LDAR, LDARH, LDLAR, LDLARH.
> - STLLR, STLLRH, STLR, STLRH, STLUR, and STLURH.
>
> If FEAT_LRCPC3 is implemented, the following instructions generate an Alignment fault if all bytes being accessed for a single register are not within a single 16-byte quantity, aligned to 16 bytes for access:
>
> - LDIAPP, STILP, the post index versions of LDAPR and the pre index versions of STLR.
> - If Advanced SIMD and floating-point instructions are implemented, LDAPUR (SIMD&FP), LDAP1 (SIMD&FP), STLUR (SIMD&FP), and STL1 (SIMD&FP).

is changed to read:

> Non-aligned access. This bit controls generation of Alignment faults at EL1 and EL0 under certain conditions.
>
> The following instructions generate an Alignment fault if all bytes being accessed are not within a single naturally aligned 16-byte quantity, for access:
>
> - LDAPR, LDAPRH, LDAPUR, LDAPURH, LDAPURSH, LDAPURSW, LDAR, LDARH, LDLAR, LDLARH.
> - STLLR, STLLRH, STLR, STLRH, STLUR, and STLURH.
> - If FEAT_LRCPC3 is implemented, the post index versions of LDAPR and the pre index versions of STLR.
> - If FEAT_LRCPC3 and Advanced SIMD and floating-point instructions are implemented, LDAPUR (SIMD&FP), LDAP1 (SIMD&FP), STLUR (SIMD&FP), and STL1 (SIMD&FP).
>
> If FEAT_LRCPC3 is implemented, the following instructions generate an Alignment fault if all bytes being accessed for a single register are not within a single naturally aligned 16-byte quantity, for access:
>
> - LDIAPP, STILP

The equivalent changes are made in the following sections:

- D24.2.168 “SCTLR_EL2, System Control Register (EL2)”, field nAA.
- D24.2.169 “SCTLR_EL3, System Control Register (EL3)”, field nAA.

2.131 D23854

In section D24.2.56 “HCR_EL2, Hypervisor Configuration Register”, in the description of the field FB, bit [9], the text that reads:

> Force broadcast. Causes the following instructions to be broadcast within the Inner Shareable domain when executed from EL1:
>
> AArch32: BPIALL, TLBIALL, TLBIMVA, TLBIASID, DTLBIALL, DTLBIMVA, DTLBIASID, ITLBIALL, ITLBIMVA, ITLBIASID, TLBIMVAA, ICIALLU, TLBIMVAL, and TLBIMVAAL.
>
> AArch64: TLBI VMALLE1, TLBI VAE1, TLBI ASIDE1, TLBI VAAE1, TLBI VALE1, TLBI VAALE1, IC IALLU, TLBI RVAE1, TLBI RVAAE1, TLBI RVALE1, and TLBI RVAALE1.

is changed to read:

> Force broadcast. Causes the following Non-shareable invalidate instructions to be broadcast within the Inner Shareable domain when executed from EL1:
>
> - In AArch32: BPIALL, TLBIALL, TLBIMVA, TLBIASID, DTLBIALL, DTLBIMVA, DTLBIASID, ITLBIALL, ITLBIMVA, ITLBIASID, TLBIMVAA, ICIALLU, TLBIMVAL, and TLBIMVAAL.
> - In AArch64:
>   - TLBI instructions that are executable at EL1 and do not specify IS or OS.
>   - TLBIP instructions that are executable at EL1 and do not specify IS or OS.
>   - IC IALLU.
>
> See also:
>
> - A64 System instructions for TLB maintenance.

2.132 C23859

In section K1.2.13 “Crossing a page boundary with different memory types or Shareability attributes”, the text that reads:

> A memory access from a load or store instruction that crosses a page boundary to a memory location that has a different memory type or Shareability attribute results in **CONSTRAINED UNPREDICTABLE** behavior. In this case, the implementation must perform one of the following behaviors:
>
> ...

is changed to read:

> If a single load or store instruction generates multiple memory accesses, such that the total set of accesses crosses a page boundary to a memory location that has a different memory type,

> Normal or Device, or Shareability attribute, the result is **CONSTRAINED UNPREDICTABLE** behavior. In this case, the implementation must perform one of the following behaviors:
>
> ...

2.133 C23866

In section A2.3.6 "The Armv9.5 architecture extension", under the description of 'FEAT_ETS3, Enhanced Translation Synchronization', the following text is added:

> If FEAT_ETS3 is implemented, then FEAT_ETS2 is implemented.

2.134 D23869

In section D24.2.168 "SCTLR_EL2, System Control Register (EL2)", the text that reads:

> The reset behavior of this field is:
>
> - On a Warm reset:
> - When the highest implemented Exception level is EL2, this field resets to an architecturally **UNKNOWN** value.

is changed to read:

> The reset behavior of this field is:
>
> - On a Warm reset, this field resets to an architecturally **UNKNOWN** value.

Equivalent changes are made in the following sections:

- D24.2.167 "SCTLR_EL1, System Control Register (EL1)"
- D24.2.169 "SCTLR_EL3, System Control Register (EL3)"
- G8.2.73 "HSCTLR, Hyp System Control Register"
- G8.2.120 "NSACR, Non-Secure Access Control Register"
- G8.2.173 "VTTBR, Virtualization Translation Table Base Register"
- G8.3.32 "HDCR, Hyp Debug Control Register"
- G8.3.35 "SDCR, Secure Debug Control Register"

2.135 R23909

In section D17.6.5 “Additional information for each profiled conditional instruction”, the text that reads:

> A conditional compare operation is treated as a conditional operation.
>
> It is **IMPLEMENTATION DEFINED** which conditional select operations (both integer and floating-point), including general-purpose, SIMD&FP, and SVE operations, are treated as conditional:
>
> - Conditional select.
> - Conditional select increment.
> - Conditional select negate.
> - Conditional select invert operations.
>
> Predicated SVE operations are not conditional operations, as the conditionality of the operation is controlled by a predicate.

is changed to read:

> Only operations where the conditionality is controlled by PSTATE.{N, Z, C, V} are treated as conditional operations. Predicated SVE operations are not treated as conditional operations, because the conditionality of the operation is controlled by a predicate.
>
> For each of the following operations, it is **IMPLEMENTATION DEFINED** whether it is treated as a conditional operation:
>
> - Conditional select, both integer and floating-point.
> - Conditional select increment.
> - Conditional select negation.
> - Conditional select inversion.
> - Conditional compare, both integer and floating-point.
> - Conditional compare negative.
>
> Arm recommends that all operations where the conditionality is controlled by PSTATE.{N, Z, C, V} are treated as conditional operations.

2.136 R23917

In section D24.2.163 “SCR_EL3, Secure Configuration Register”, in the description of the field PIEn, bit [45], the table that reads:

PIEn	Meaning
0b0	EL0, EL1 and EL2 accesses to the specified registers are disabled, and trapped to EL3. The values in these registers are treated as 0.

PIEn	Meaning
0b1	This control does not cause any instructions to be trapped.

is changed to read:

PIEn	Meaning
0b0	EL0, EL1 and EL2 accesses to the specified registers are disabled, and trapped to EL3.
0b1	This control does not cause any instructions to be trapped.

> If this field is 0, it is **IMPLEMENTATION SPECIFIC** whether the values of the named registers are treated as zero.

2.137 D23919

In section B2.10.2.1 “Gathering”, the following text:

> Gathering between the memory accesses generated by one Load-Acquire or Store-Release instruction, and the memory accesses generated by another Load-Acquire or Store-Release instruction is not permitted.
>
> Gathering between two memory accesses generated by a Load-Acquire/Store-Release is not permitted.
>
> Gathering between memory accesses separated by a memory barrier that affects those memory accesses is not permitted.

is changed to read:

> Gathering between two accesses, where one access is barrier-ordered-before the other access, is not permitted.

2.138 D23933

In section D24.2.167 “SCTLR_EL1, System Control Register (EL1)”, in the description of the field EE, bit [25], the text that reads:

> When FEAT_MixedEnd is implemented: Endianness of data accesses at EL1, and stage-1 translation table walks in the EL1&0 translation regime.

EE	Meaning
0b0	Explicit data accesses at EL1, and stage-1 translation table walks in the EL1&0 translation regime are little-endian.
0b1	Explicit data accesses at EL1, and stage-1 translation table walks in the EL1&0 translation regime are big-endian.

> The EE bit is permitted to be cached in a TLB.

When the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}, this bit has no effect on the PE.

The reset behavior of this field is:

- On a Warm reset, this field resets to an **IMPLEMENTATION DEFINED** value.

Otherwise:

Endianness of data accesses at EL1, and stage-1 translation table walks in the EL1&0 translation regime.

EE	Meaning
0b0	Explicit data accesses at EL1, and stage-1 translation table walks in the EL1&0 translation regime are little-endian.
0b1	Explicit data accesses at EL1, and stage-1 translation table walks in the EL1&0 translation regime are big-endian.

If an implementation does not provide Big-endian support at Exception levels higher than EL0, this bit is **RES0**.

If an implementation does not provide Little-endian support at Exception levels higher than EL0, this bit is **RES1**.

The EE bit is permitted to be cached in a TLB.

When the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}, this bit has no effect on the PE.

The reset behavior of this field is:

- On a Warm reset, this field resets to an **IMPLEMENTATION DEFINED** value.

is changed to read:

When FEAT_MixedEnd is implemented:

Endianness of data accesses at EL1, and stage-1 translation table walks in the EL1&0 translation regime.

EE	Meaning
0b0	Explicit data accesses at EL1, and stage-1 translation table walks in the EL1&0 translation regime are little-endian.
0b1	Explicit data accesses at EL1, and stage-1 translation table walks in the EL1&0 translation regime are big-endian.

The EE bit is permitted to be cached in a TLB.

When the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}, this bit has no effect on the PE.

The reset behavior of this field is:

- On a Warm reset, this field resets to an **IMPLEMENTATION DEFINED** value.

When FEAT_BigEnd is implemented

> Explicit data accesses at EL1, and stage-1 translation table walks in the EL1&0 translation regime are big-endian.
>
> This field is **RES1**.
>
> Otherwise
>
> Explicit data accesses at EL1, and stage-1 translation table walks in the EL1&0 translation regime are little-endian.
>
> This field is **RES0**.

Equivalent changes are made in the following sections:

- D24.2.168 “SCTLR_EL2, System Control Register (EL2)” – field EE
- D24.2.169 “SCTLR_EL3, System Control Register (EL3)” – field EE

In section D24.2.167 “SCTLR_EL1, System Control Register (EL1)”, in the description of the field E0E, bit [24], the text that reads:

> E0E, bit [24]
>
> When FEAT_MixedEndEL0 is implemented:
>
> Endianness of data accesses at EL0.

E0E	Meaning
0b0	Explicit data accesses at EL0 are little-endian.
0b1	Explicit data accesses at EL0 are big-endian.

> This bit has no effect on the endianness of LDTR, LDTRH, LDTRSH, LDTRSW, STTR, and STTRH instructions executed at EL1.
>
> When the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}, this bit has no effect on execution at EL0.
>
> The reset behavior of this field is:
>
> - On a Warm reset, this field resets to an architecturally **UNKNOWN** value.
>
> Otherwise:
>
> Endianness of data accesses at EL0.

E0E	Meaning
0b0	Explicit data accesses at EL0 are little-endian.
0b1	Explicit data accesses at EL0 are big-endian.

> If an implementation only supports Little-endian accesses at EL0, this bit is **RES0**.

If an implementation only supports Big-endian accesses at EL0, this bit is **RES1**.

This bit has no effect on the endianness of LDTR, LDTRH, LDTRSH, LDTRSW, STTR, and STTRH instructions executed at EL1.

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

The reset behavior of this field is:

- On a Warm reset, this field resets to an **IMPLEMENTATION DEFINED** value.

is changed to read:

When FEAT_MixedEndEL0 is implemented: Endianness of data accesses at EL0.

E0E	Meaning
0b0	Explicit data accesses at EL0 are little-endian.
0b1	Explicit data accesses at EL0 are big-endian.

This bit has no effect on the endianness of LDTR, LDTRH, LDTRSH, LDTRSW, STTR, and STTRH instructions executed at EL1.

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

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally **UNKNOWN** value.

When FEAT_BigEndEL0 is implemented

Explicit data accesses at EL0 are big-endian.

This field is **RES1**.

Otherwise

Explicit data accesses at EL0 are little-endian.

This field is **RES0**.

Equivalent changes are made in the following section:

- D24.2.168 “SCTLR_EL2, System Control Register (EL2)” – field E0E

2.139 D23934

In section D14.3 “Common event numbers”, in the event description “`0x80F1`, ASE_FP_DOT_SPEC, Floating-point operation speculatively executed, Advanced SIMD dot-product”, the text that reads:

> - When FEAT_BF8DOT2 is implemented, Advanced SIMD: FDOT (2-way, by element) or FDOT (2-way, vector).

is changed to read:

> - When FEAT_FP8DOT2 is implemented, Advanced SIMD: FDOT (2-way, by element) or FDOT (2-way, vector).

Equivalent changes are made in the following event description:

- `0x80F3`, ASE_SVE_FP_DOT_SPEC, Floating-point operation speculatively executed, Advanced SIMD or SVE dot-product.

In the same section, in the event description “`0x8028`, FP_FMA_SPEC, Floating-point operation speculatively executed, FMA”, the text that reads:

> - When FEAT_BF8FMA is implemented, Advanced SIMD: FMLALB, FMLALLBB, FMLALLBT, FMLALLTB, FMLALLTT, or FMLALT.

is changed to read:

> - When FEAT_FP8FMA is implemented, Advanced SIMD: FMLALB, FMLALLBB, FMLALLBT, FMLALLTB, FMLALLTT, or FMLALT.

Equivalent changes are made in 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.

2.140 C23935

In section D24.5.8 “PMCR_EL0, Performance Monitors Control Register”, in the description of the field X, bit [4], the following text is added:

> If FEAT_ETE is implemented, this field does not affect the use of PMU events as an External Input by the trace unit. If FEAT_ETM4 is implemented, this field does affect the use of PMU events as an External Input by the trace unit.

The same clarification is added in section G8.4.9 “PMCR, Performance Monitors Control Register”.

2.141 D23936

In section D13.1.4 “Interaction with trace”, the text that reads:

> D13.1.4 Interaction with trace
>
> It is **IMPLEMENTATION DEFINED** whether the implementation exports counter events to a trace unit, or other external monitoring agent, to provide triggering information. The form of any exporting is also **IMPLEMENTATION DEFINED**. If implemented, this exporting might be enabled as part of the performance monitoring control functionality.
>
> Arm recommends system designers include a mechanism for importing a set of external events to be counted, but such a feature is **IMPLEMENTATION DEFINED**. When implemented, this feature enables the trace unit to pass in events to be counted.
>
> Exporting PMU events to the ETM is prohibited for some Exception levels when SelfHostedTraceEnabled() == TRUE. For more information, see Controls to prohibit trace at Exception levels.

is changed to read:

> D13.1.4 Interaction with trace
>
> When FEAT_ETE is implemented, the trace unit uses the PMU events as External Inputs. See External inputs.
>
> When FEAT_ETMv4 is implemented, it is **IMPLEMENTATION DEFINED** whether the implementation exports PMU events to the ETM trace unit, and the form of any exporting is also **IMPLEMENTATION DEFINED**. If implemented, this exporting is controlled by PMCR.X.
>
> For more information about when the trace unit is permitted to observe PMU events, see Controls to prohibit trace at Exception levels and External inputs.
>
> When FEAT_ETE is implemented, the PMU implements a set of events for importing external events from the trace unit and the cross-trigger interface. See Required events. Arm recommends similar functionality is implemented when FEAT_ETMv4 is implemented, and further recommends re-using the events defined for FEAT_ETE. When implemented, this functionality enables the trace unit to pass in events to be counted.
>
> D13.1.5 Export of PMU events
>
> It is **IMPLEMENTATION DEFINED** whether the implementation exports counter events to any other external monitoring agent to provide triggering information. The form of any exporting is also **IMPLEMENTATION DEFINED**. If implemented, this exporting is controlled by PMCR.X.

2.142 D23947

In section D13.8 “Event threshold and edge counting”, the rule that reads:

> R_{HBWBB}
>
> The value of V[n] for event counter <n> is defined as follows, where TC, TH, and TE are the Effective values of PMEVTYPER<n>_EL0.{TC, TH, TE} respectively:

```
        / V_B[n], if TC = 0b000 and TH = 0 (disabled).
V [n] = { V_T[n], if TC != 0b000 and TH != 0 and TE = 0b0 (threshold).
        \ V_E[n], if TE = 0b1 (edge).
```

> The pseudocode function PMUCountValue() in Armv8 Pseudocode describes this.

is changed to read:

> R_{HBWBB}
>
> The value of V[n] for event counter <n> is defined as follows, where TC, TH, and TE are the Effective values of PMEVTYPER <n>_EL0.{TC, TH, TE} respectively:

```
        / V_B[n], if TC = 0b000 and TH = 0 (disabled).
V [n] = { V_T[n], if (TC != 0b000 or TH != 0) and TE = 0b0 (threshold).
        \ V_E[n], if TE = 0b1 (edge).
```

> The pseudocode function PMUCountValue() in Armv8 Pseudocode describes this.

2.143 D23962

In section D14.3.2 “Common microarchitectural events”, the text that reads:

> 0x8432, SVE_FP_PREDUCE_SPEC, Floating-point operation_speculatively_executed, Advanced SIMD pairwise add step or pairwise reduce step

is changed to read:

> 0x8432, SVE_FP_PREDUCE_SPEC, Floating-point operation speculatively executed, SVE pairwise add step or pairwise reduce step

2.144 C23967

In section D24.7.9 “PMSFCR_EL1, Sampling Filter Control Register”, the text that reads:

> R_{YGCFK}

When FEAT_SPE_EFT is implemented and PMSFCR_EL1.FT is 1, all of the following apply:

- The sampled operation is discarded if the following is true:

```
(!IsZero(ctrl AND NOT mask) && IsZero(flags AND (ctrl AND NOT mask)))
```

That is, the SPU applies a Boolean-OR filter based on each of the operation type filter controls where the corresponding operation type filter mask bit is 0:

- If all these bits are zero, or all the mask bits are one, then no operations are discarded by this part of the filter.
- Otherwise, operations that match any of these bits are selected, and other operations are discarded.
- ...

is changed to read:

R_{YGCFK}

When FEAT_SPE_EFT is implemented and PMSFCR_EL1.FT is 1, all of the following apply:

- The sampled operation is discarded if the following is true:

```
(!IsZero(ctrl AND NOT mask) && IsZero(flags AND (ctrl AND NOT mask)))
```

That is, the SPU applies a Boolean-OR filter based on each of the operation type filter controls where the corresponding operation type filter mask bit is 0:

- If all the operation type filter mask bits are one, then no operations are discarded by this part of the filter.
- Otherwise, for the operation type bits for which corresponding mask bit is 0:
  - If all these bits are 0, then no operations are discarded by this part of the filter.
  - Otherwise, operations that match any of these bits are selected, and other operations are discarded.
- ...

In the same section, the description of the LD, bit [17], which reads:

LD, bit [17]

Load filter enable.

LD	Meaning
0b0	If FEAT_SPE_EFT is implemented and PMSFCR_EL1.LDm is 1, then record only operations that are not load operations. Otherwise, do not record load operations, unless enabled by another filter.
0b1	If FEAT_SPE_EFT is implemented and PMSFCR_EL1.LDm is 1, then record only operations that are load operations. Otherwise, record all load operations.

This field is ignored by the PE when PMSFCR_EL1.FT == 0.

For filtering purposes, load operations include vector loads and atomic operations that return a value to the PE.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally **UNKNOWN** value.

is changed to read:

LD, bit [17]

When FEAT_SPE_EFT is implemented and PMSFCR_EL1.LDm is 1

Load filter enable.

LD	Meaning
0b0	Record only operations that are not load operations.
0b1	Record only operations that are load operations.

This field is ignored by the PE when PMSFCR_EL1.FT == 0.

For filtering purposes, load operations include vector loads and atomic operations that return a value to the PE.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally **UNKNOWN** value.

Otherwise

Load filter enable.

LD	Meaning
0b0	Do not record load operations, unless enabled by another filter.
0b1	Record all load operations.

This field is ignored by the PE and no records are removed by this filter when any of the following apply:

- PMSFCR_EL1.FT == 0.
- FEAT_SPE_EFT is implemented and the values of all of PMSFCR_EL1.{SIMDm, FPm, STm, LDm, Bm} are zero for which the corresponding PMSFCR_EL1.{SIMD, FP, ST, LD, B} bit is zero.

If FEAT_SPE_EFT is not implemented and the values of PMSFCR_EL1.{ST, LD, B} are all zero, then it is **CONSTRAINED UNPREDICTABLE** whether any records are removed by this filter.

For filtering purposes, load operations include vector loads and atomic operations that return a value to the PE.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally **UNKNOWN** value.

Similar updates are made to each of the other fields {SIMD, FP, ST, B}, noting that PMSFCR_EL1.{SIMD, FP} are only implemented when FEAT_SPE_EFT is implemented.

2.145 D23988

In section D1.3.5.5 “Prioritization of Synchronous exceptions taken to AArch64 state”, under rule R_{ZFGJP}, the following text is added to the priority table between the existing priorities 14 and 15:

Priority	Synchronous exception type
14	...
...	Exceptions taken to EL3 due to the configuration of MPAM3_EL3.TRAPLOWER.

2.146 D23995

In section D24.2.52 “HACDBSBR_EL2, Hardware Accelerator for Cleaning Dirty State Base Register”, the following encoding is added:

SZ, bits [3:0]

Size of the HACDBS.

SZ	Meaning
0b0000	4 KB
...	...

The equivalent change is made in:

- D24.2.58 “HDBSSBR_EL2, Hardware Dirty State Tracking Structure Base Register”.