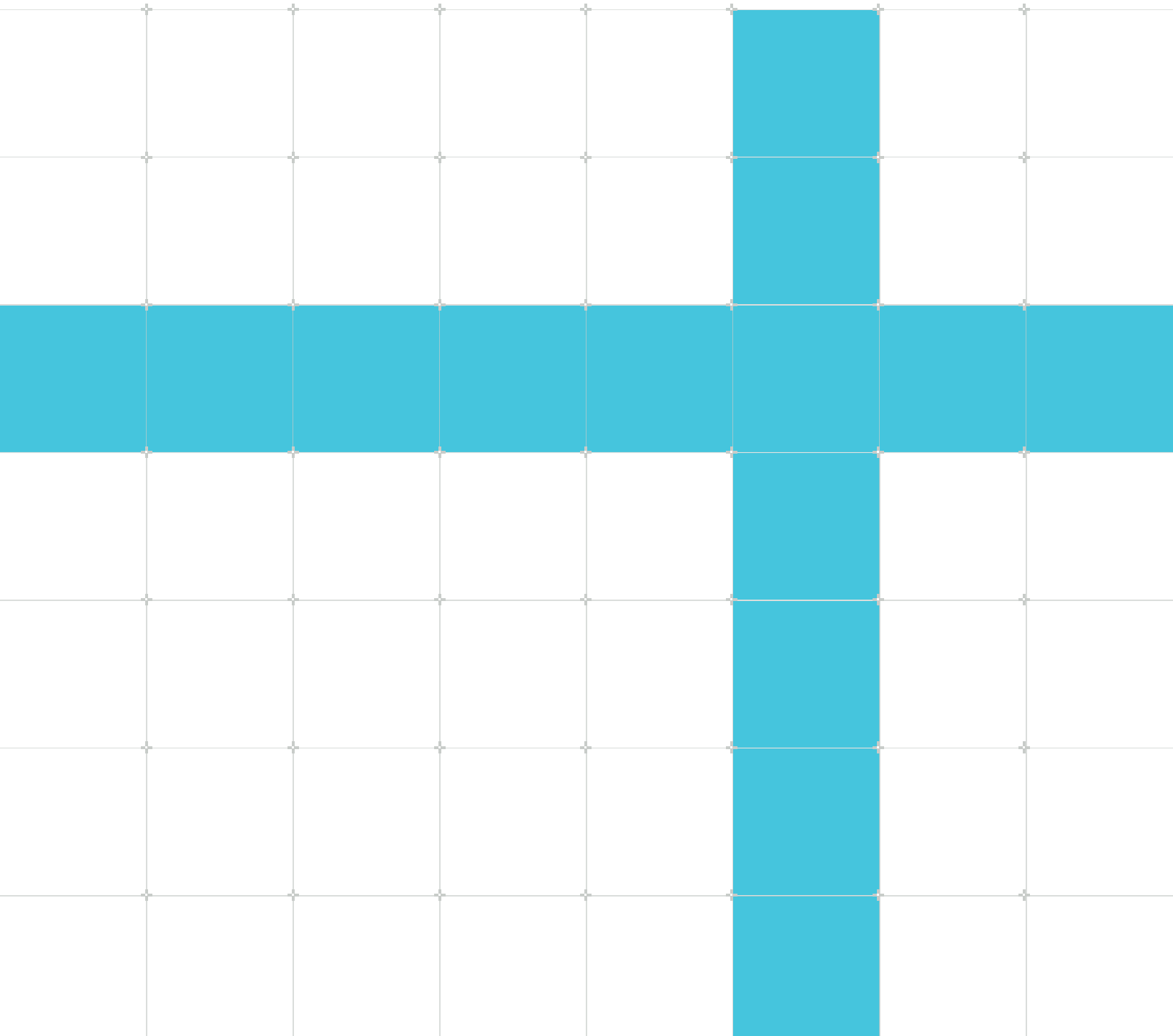
Arm® Architecture Reference Manual for A-profile architecture

Known issues in Issue L.b

Issue 01
102105_L.b_01_en



Arm® Architecture Reference Manual for A-profile architecture

Known issues in Issue L.b

Release information

Document history

Issue	Date	Confidentiality	Change
L.b-01	3 July 2025	Non-Confidential	Known Issues in Arm® Architecture Reference Manual, Issue L.b, as of 1 July 2025
L.b-00	30 May 2025	Non-Confidential	Known Issues in Arm® Architecture Reference Manual, Issue L.b, as of 28 May 2025
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

Issue	Date	Confidentiality	Change
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

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

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

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

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

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....10
1.1 Conventions....10
1.2 Useful resources....11
1.3 Other information....12

2. Known issues....13
2.1 D17119....13
2.2 C17296....13
2.3 C17536....14
2.4 D18847....16
2.5 D19403....17
2.6 C19822....18
2.7 D20234....19
2.8 C20429....20
2.9 D20796....20
2.10 D22004....22
2.11 D22031....23
2.12 E22039....25
2.13 D22105....25
2.14 D22207....29
2.15 C22264....31
2.16 C22430....33
2.17 D22446....34
2.18 C22566....35
2.19 C22584....35
2.20 D22595....35
2.21 D22719....36
2.22 D22806....39
2.23 R22882....40
2.24 D22979....40
2.25 D23002....41
2.26 R23026....41

2.27 D23122 ... 43
2.28 R23144 ... 45
2.29 D23154 ... 46
2.30 C23225 ... 47
2.31 D23239 ... 48
2.32 C23271 ... 48
2.33 D23403 ... 49
2.34 C23431 ... 50
2.35 D23442 ... 50
2.36 D23462 ... 52
2.37 C23479 ... 52
2.38 D23518 ... 59
2.39 D23543 ... 61
2.40 R23545 ... 62
2.41 D23551 ... 63
2.42 C23552 ... 64
2.43 C23586 ... 65
2.44 C23587 ... 69
2.45 D23647 ... 70
2.46 D23659 ... 70
2.47 D23680 ... 72
2.48 D23683 ... 73
2.49 D23689 ... 74
2.50 C23701 ... 74
2.51 D23713 ... 76
2.52 D23733 ... 78
2.53 D23744 ... 79
2.54 C23753 ... 80
2.55 C23766 ... 81
2.56 D23768 ... 82
2.57 D23772 ... 83
2.58 C23775 ... 90
2.59 D23794 ... 92
2.60 D23796 ... 92
2.61 D23797 ... 93
2.62 D23807 ... 94

2.63 C23808....95
2.64 D23812....95
2.65 E23814....95
2.66 R23822....96
2.67 D23823....97
2.68 C23842....97
2.69 C23844....99
2.70 C23845....99
2.71 D23852....100
2.72 D23854....101
2.73 C23859....102
2.74 C23866....102
2.75 D23869....102
2.76 R23907....103
2.77 R23909....104
2.78 R23917....105
2.79 R23928....105
2.80 D23933....108
2.81 D23934....111
2.82 C23935....112
2.83 D23936....112
2.84 D23947....113
2.85 C23956....113
2.86 D23962....114
2.87 D23966....115
2.88 C23967....116
2.89 C23982....119
2.90 D23988....120
2.91 D23995....120
2.92 C23997....120
2.93 D24019....122
2.94 C24031....122
2.95 C24054....123
2.96 C24061....124
2.97 D24062....125
2.98 C24064....125

2.99 C24066 126
2.100 D24086 127
2.101 D24093 127
2.102 C24096 127
2.103 D24099 128
2.104 D24101 129
2.105 C24124 129
2.106 D24129 130
2.107 D24146 131
2.108 C24218 132
2.109 D24149 133
2.110 C24151 133
2.111 D24153 134
2.112 D24156 134
2.113 D24162 135
2.114 D24166 135
2.115 D24180 136
2.116 C24190 138
2.117 C24218 138

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.b	DDI 0487L.b	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.b.

Key

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

2.1 D17119

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

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

2.2 C17296

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

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

is changed to read:

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

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

2.3 C17536

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

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

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

This includes MOV and MVN instructions.

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

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

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

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

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

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

is changed to read:

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

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

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

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

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

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

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

2.4 D18847

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

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

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

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

is changed to read:

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

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

2.5 D19403

In section D2.12 “Synchronization and debug exceptions” and section G2.11 “Synchronization and debug exceptions”, the following text is deleted:

> Some register updates are self-synchronizing. Others require an explicit Context Synchronization event.

In section D24.10.20 “CNTPCTSS_EL0, Counter-timer Self-Synchronized Physical Count Register”, the text that reads:

> This register is a self-synchronised view of the CNTPCT_EL0 counter, and cannot be read speculatively.

is changed to read:

> This register is a view of the CNTPCT_EL0 register for which reads appear to occur in program order relative to other instructions, without the need for any explicit synchronization. Reads of this register return a value consistent with the counter not being read until the read instruction is known to be non-speculative.

Equivalent changes are made in the following sections:

- D24.10.29 “CNTVCTSS_EL0, Counter-timer Self-Synchronized Virtual Count Register”.

- G8.7.20 “CNTPCTSS, Counter-timer Self-Synchronized Physical Count register”.
- G8.7.25 “CNTVCTSS, Counter-timer Self-Synchronized Virtual Count register”.

2.6 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
    ...
```

2.7 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.
...	...

2.8 C20429

In the section D17.8.2 “Buffer Full event”, the text that reads:

> If, after writing a sample record, there is not sufficient space in the Profiling Buffer for a sample record of the size indicated by PMSIDR_EL1.MaxSize, and PMBSR_EL1.S is 0, a Profiling Buffer management event is generated:
>
> - ...
> - PMBPTR_EL1 is set to the first byte after the last complete sample record. PMBSR_EL1.DL is unchanged
> - ...

is changed to read:

> If, after writing a sample record, there is not sufficient space in the Profiling Buffer for a sample record of the size indicated by PMSIDR_EL1.MaxSize, and PMBSR_EL1.S is 0, a Profiling Buffer management event is generated:
>
> - ...
> - PMBPTR_EL1 is set to the first byte after the last complete sample record, which can include any padding that the implementation uses to force alignment. PMBSR_EL1.DL is unchanged.
> - ...

Equivalent changes are made in the following sections: - D17.8.3 “Faults and watchpoints”. - D17.8 “Profiling Buffer management”.

2.9 D20796

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

> A DSB instruction executed by a PE, PEe, completes when all of the following apply:
>
> - All explicit memory effects of the required access types appearing in program order before the DSB are complete for the set of observers in the required shareability domain.
> - If the required access types of the DSB is reads, or reads and writes, all implicit tag memory read effects appearing in program order before the DSB are complete.

- If the required access types of the DSB is reads and writes, the following instructions issued by PEe before the DSB are complete for the required shareability domain:
  - All cache maintenance instructions.
  - All TLB maintenance instructions.
  - All PSB instructions.
- When FEAT_XS is implemented, if the required access types of the DSB is reads and writes, completion of the DSB instruction with the nXS qualifier executed by a PE, PEe, ensures that:
  - All previous TLBInXS maintenance operations generated by AArch64 TLB maintenance instructions with the nXS qualifier executed by PEe are finished for all PEs in the shareability domain of the DSB instruction.
  - All previous TLBInXS maintenance operations generated by AArch32 or AArch64 TLB maintenance instructions executed at EL1 by PEe when HCRX_EL2.FnXS is 1 are finished for all PEs in the shareability domain of the DSB instruction.
- Completion of the DSB instruction with the nXS qualifier executed by a PE, PEe, does not ensure that:
  - All previous TLB maintenance operations generated by AArch32 or AArch64 TLB maintenance instructions executed at EL1 by PEe when HCRX_EL2.FnXS is 0 are finished for all PEs in the shareability domain of the DSB instruction.
  - All previous TLB maintenance operations generated by AArch32 or AArch64 TLB maintenance instructions executed at EL2 or EL3 by PEe are finished for all PEs in the shareability domain of the DSB instruction.

is changed to read:

A DSB instruction executed by a PE completes when all of the following are complete for the set of observers in the required shareability domain:

- If the required access types of the DSB is reads, all of the following:
  - All Explicit Memory Read Effects generated by instructions in program order before the DSB.
  - If FEAT_GCS is implemented, all GCS Memory Read Effects that appear in program order before a GCSB instruction that appears in program order before the DSB.
  - All Implicit Tag Memory Read Effects generated by instructions in program order before the DSB.
- If the required access types of the DSB is writes, all of the following:
  - All Explicit Memory Write Effects generated by instructions in program order before the DSB.
  - If FEAT_GCS is implemented, all GCS Memory Write Effects that appear in program order before a GCSB instruction that appears in program order before the DSB.
- If the required access types of the DSB is reads and writes, all of the following:
  - All Explicit Memory Effects generated by instructions in program order before the DSB.
  - All Implicit Tag Memory Read Effects generated by instructions in program order before the DSB.

- All Implicit TTD Memory Effects generated by instructions in program order before the DSB.
- All Implicit Instruction Memory Read Effects generated by instructions in program order before the DSB.
- All cache maintenance instructions in program order before the DSB.
- If FEAT_SPMU is implemented, all Direct System Register Write Effects to a System PMU register generated by instructions in program order before the DSB.
- If FEAT_SPE is implemented, all PSB CSYNC instructions in program order before the DSB.
- If FEAT_XS is implemented and the DSB has the nXS qualifier, all of the following:
  - All TLBInXS maintenance operations generated by AArch64 TLB maintenance instructions with the nXS qualifier in program order before the DSB.
  - If HCRX_EL2.FnXS is 1, all TLBInXS maintenance operations generated by AArch32 or AArch64 TLB maintenance instructions executed at EL1 in program order before the DSB.
- If FEAT_XS is not implemented or the DSB does not have the nXS qualifier, all of the following:
  - All TLBI maintenance operations generated by TLB maintenance instructions in program order before the DSB.
- If FEAT_HDBSS is implemented and the DSB is executed at EL2 or EL3, all HDBSS Memory Write Effects generated by instructions in program order before the DSB.
- If FEAT_TRBE is implemented, all TSB CSYNC instructions in program order before the DSB.
- All CFP RCTX, COSP RCTX, CPP RCTX, DVP RCTX instructions in program order before the DSB.
- If FEAT_MTE_ASYNC is implemented, all Indirect System Register Write Effects to the Tag Fault Status Register accessible at ELx generated by instructions in program order before the DSB.
- If FEAT_GCS is implemented, all GCS Memory Effects that appear in program order before a GCSB instruction that appears in program order before the DSB.

2.10 D22004

The following text is deleted from section “Glossary”:

> Explicit memory effect
>
> A read from memory, or a write to memory, generated by a load or store instruction executed by the PE. Reads and writes generated by hardware translation table accesses, as well as instruction fetches and SPE writes to the Profiling Buffer, are not explicit memory effects.

The following text is added to section B2.3.1 “Basic definitions”:

Explicit Memory Effect

A Memory Read Effect or a Memory Write Effect that is generated by one of the following:

- a load instruction, store instruction, atomic instruction, Tag and Data store instruction, DC ZVA, DC GZVA, or STZGM; to access the Memory Location(s) addressed by any of the following from the instruction:
  - the Xn|SP operand, plus any applicable offset
  - the PC plus the offset specified by the <label> argument
- a Tag load instruction, Tag store instruction, Tag and data store instruction, Bulk Allocation tag access instruction, DC GZVA, or DC GVA; to access the Tag Location(s) addressed by the Xn|SP operand, plus any applicable offset, of the instruction.
- a Memory Copy or Memory Set instruction; to access the Memory Location(s) addressed by the Xs or Xd operand of the instruction, including all of the bytes within the range specified by the Xn operand of the instruction.
- a Memory Set with tag setting instruction; to access the Tag Location(s) addressed by the Xs or Xd operand of the instruction, including all of the Allocation tags within the range specified by the Xn operand of the instruction.
- an MRS or MSR instruction that is transformed into a Memory Effect due to FEAT_NV2; to access the Memory Locations addressed by VNCR_EL2 plus the offset for the specified register.

Note: Prefetch memory instructions do not generate any Explicit Memory Effects.

2.11 D22031

In section C6.2.287 “PRFUM”, under the “Assembler Symbols” heading, the table for the “<prfop>” symbol that reads:

Rt	<prfop>
00000	PLDL1KEEP
00001	PLDL1STRM
00010	PLDL2KEEP
00011	PLDL2STRM
00100	PLDL3KEEP
00101	PLDL3STRM
01000	PLIL1KEEP
01001	PLIL1STRM
01010	PLIL2KEEP
01011	PLIL2STRM
01100	PLIL3KEEP
01101	PLIL3STRM
10000	PSTL1KEEP

Rt	<prfop>
10001	PSTL1STRM
10010	PSTL2KEEP
10011	PSTL2STRM
10100	PSTL3KEEP
10101	PSTL3STRM

is changed to read:

Rt	<prfop>	Architectural Feature
00000	PLDL1KEEP	-
00001	PLDL1STRM	-
00010	PLDL2KEEP	-
00011	PLDL2STRM	-
00100	PLDL3KEEP	-
00101	PLDL3STRM	-
00110	PLDSLCKEEP	FEAT_PRFMSLC
00111	PLDSLCSTRM	FEAT_PRFMSLC
01000	PLIL1KEEP	-
01001	PLIL1STRM	-
01010	PLIL2KEEP	-
01011	PLIL2STRM	-
01100	PLIL3KEEP	-
01101	PLIL3STRM	-
01110	PLISLCKEEP	FEAT_PRFMSLC
01111	PLISLCSTRM	FEAT_PRFMSLC
10000	PSTL1KEEP	-
10001	PSTL1STRM	-
10010	PSTL2KEEP	-
10011	PSTL2STRM	-
10100	PSTL3KEEP	-
10101	PSTL3STRM	-
10110	PSTSLCKEEP	FEAT_PRFMSLC
10111	PSTSLCSTRM	FEAT_PRFMSLC

In section D24.2.80 “ID_AA64ISAR2_EL1, AArch64 Instruction Set Attribute Register 2”, the field “PRFMSLC, bits [43:40]” that reads:

> Indicates whether the PRFM instructions support a system level cache option. The value of this field is an **IMPLEMENTATION DEFINED** choice of:

PRFMSLC	Meaning
0b0000	The PRFM instructions do not support the SLC target.

PRFMSLC	Meaning
0b0001	The PRFM instructions support the SLC target.

is changed to read:

> Indicates whether the PRFM and PRFUM instructions support a system level cache option. The value of this field is an **IMPLEMENTATION DEFINED** choice of:

PRFMSLC	Meaning
0b0000	The PRFM and PRFUM instructions do not support the SLC target.
0b0001	The PRFM and PRFUM instructions support the SLC target.

Equivalent changes made in section A2.2.10 “The Armv8.9 architecture extension”, under the heading “FEAT_PRFMSLC, SLC target support for PRFM instructions”.

2.12 E22039

In section C6.2.80 “CPYFP, CPYFM, CPYFE”, the text that reads:

> The memory copy performed by these instructions is in the forward direction only, so the instructions are suitable for a memory copy only where there is no overlap between the source and destination locations, or where the source address is greater than the destination address.

is changed to read:

> The memory copy performed by these instructions is in the forward direction only, so the instructions are suitable for a memory copy only where there is no overlap between the source and destination locations, or where the source address is greater than or equal to the destination address.

2.13 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.

FMO	Meaning
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.

2.14 D22207

In section D6.3.5 "The owning translation regime", the text that reads:

R_{RRCNN}

When any of the following is true, the translation of addresses generated by the Trace Buffer Unit is **CONSTRAINED UNPREDICTABLE**:

- The owning Security state is Secure and SCR_EL3.NS is 1.
- When FEAT_RME is not implemented and the owning Security state is Non-secure and SCR_EL3.NS is 0.

- When FEAT_RME is implemented and the owning Security state is Non-secure and SCR_EL3.{NSE,NS} is not {0,1}.
- When FEAT_RME is implemented and the owning Security state is Realm and SCR_EL3.NSE is 0.

For these translations, the PE behaves as if one of the following is true:

- The owning Security state is Secure and SCR_EL3.NS is 0.
- When FEAT_RME is not implemented, the owning Security state is Non-secure and SCR_EL3.NS is 1.
- When FEAT_RME is not implemented, the owning Security state is Non-secure and SCR_EL3.{NSE,NS} is {0,1}.
- When FEAT_RME is not implemented, the owning Security state is Realm and SCR_EL3.{NSE,NS} is {1,1}.

Note:

The behavior might differ within the same translation.

is changed to read:

R_{RRCNN}

If the Effective value of SCR_EL3.{NSE, NS} does not match the owning Security state, it is **CONSTRAINED UNPREDICTABLE** whether any trace data:

- Is written to memory using a virtual address translated using the owning translation regime.
- Is written to memory using a virtual address translated using the translation regime formed from the owning Exception level and the Security state selected by the Effective value of SCR_EL3.{NSE, NS}.
- Is silently discarded and not written to memory.
- Is discarded and not written to memory and a trace buffer management event is generated as follows:
  - TRBSR_EL1.IRQ is set to 1.
  - If TRBSR_EL1.S is 0, then all of the following occur:
    - TRBSR_EL1.S is set to 1, collection is stopped.
    - TRBSR_EL1.EC is set to `0x00`, other buffer management event.
    - TRBSR_EL1.BSC is set to `0b000000`, access not allowed.
    - The other fields in TRBSR_EL1 are unchanged.

A corresponding update is made in section D17.9.1 "**UNPREDICTABLE** behavior", where the text that reads:

If SCR_EL3.{NSE, NS} does not match the owning Security state, then it is **CONSTRAINED UNPREDICTABLE** whether the sample record or records:

- Are written to memory using a virtual address formed using the identity of the owning translation regime.
- Are written to memory using a virtual address formed using the value of SCR_EL3.{NSE, NS}.
- Are silently discarded and not written to memory.
- Are discarded and not written to memory, and an Access Not Allowed Profiling Buffer management event is generated.

is changed to read:

If the Effective value of SCR_EL3.{NSE, NS} does not match the owning Security state, then it is **CONSTRAINED UNPREDICTABLE** whether the profiling data:

- Is written to memory using a virtual address translated using the owning translation regime.
- Is written to memory using a virtual address translated using the translation regime formed from the owning Exception level and the Security state selected by the Effective value of SCR_EL3.{NSE, NS}.
- Is silently discarded and not written to memory.
- Is discarded and not written to memory, and an Access Not Allowed Profiling Buffer management event is generated.

2.15 C22264

In section D19.2.257 FPMR, Floating-point Mode Register, the field description for F8D, bits [8:6], that reads:

F8D, bits [8:6]

Destination result format for instructions that convert other floating-point values to an FP8 format.

F8D	Meaning
`0b000`	OFP8 E5M2 format.
`0b001`	OFP8 E4M3 format.

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.

is changed to read:

F8D, bits [8:6]

> Destination result format for instructions that convert other floating-point values to an FP8 format.

F8D	Meaning
0b000	OFP8 E5M2 format.
0b001	OFP8 E4M3 format.

> All other values are Reserved. Reserved values identify an unsupported format and behave as described in K1.2.19 Reserved values in System and memory-mapped registers and translation table entries. Additionally, FP8 instructions are permitted to set an FP8 result with an unsupported format to 0xFF and signal an Invalid Operation floating-point exception.

The field description for F8S2, bits [5:3], that reads:

> F8S2, bits [5:3]
>
> Second FP8 input data stream format for multiplication instructions with FP8 operands, and the corresponding instructions that convert an FP8 format to other floating-point formats.

F8S2	Meaning
0b000	OFP8 E5M2 format.
0b001	OFP8 E4M3 format.

> 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.

is changed to read:

> F8S2, bits [5:3]
>
> Second FP8 input data stream format for multiplication instructions with FP8 operands, and the corresponding instructions that convert an FP8 format to other floating-point formats.

F8S2	Meaning
0b000	OFP8 E5M2 format.
0b001	OFP8 E4M3 format.

> All other values are reserved. Reserved values identify an unsupported format and behave as described in K1.2.19 Reserved values in System and memory-mapped registers and translation table entries. Additionally FP8 instructions are permitted to treat FP8 input values with an unsupported format as a signaling NaN.

An equivalent change is made to the field F8S1, bits [2:0].

2.16 C22430

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

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

is changed to read:

> R_{BFJJV}
>
> 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_{TGQRC}
>
> 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_{TGQRC}

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_{ZTNQN}

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_{X0001}

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

2.17 D22446

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

The Arm architecture places certain restrictions on the effects of speculation. These are:

...

- A speculative Explicit Memory Read Effect generated by an instruction (I_2) running in a hardware-defined context ($context_2$) will not speculatively read data from a speculative Explicit Memory Write Effect generated by an instruction (I_1) running in a different hardware-

> defined context ($context_1$) if I_1 generates an MMU Fault Effect instead of the Explicit Memory Write Effect.

2.18 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.

2.19 C22584

In section J1.3 “Shared pseudocode”, the function IncrementInstructionCounter() with the code that reads:

```
if old_value<64> != new_value<64> then
    PMOVSSET_EL0.F0 = '1';
    PMOVSCLR_EL0.F0 = '1';
```

is changed to read:

```
if old_value<64> != new_value<64> then
    PMOVSSET_EL0.F0 = '1';
```

Equivalent changes are made to functions CheckForPMUException(), CheckPMUOverflowCondition(), and IncrementEventCounter().

2.20 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.

2.21 D22719

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

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

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

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

- 0b01xx The entry comes from a 4KB translation granule. The level of walk for the leaf level 0bxx is encoded as:
  - `0b00` : If FEAT_LPA2 is implemented, level 0. Otherwise, treat as if TTL[3:2] is `0b00`.
  - ...
- 0b10xx The entry comes from a 16KB translation granule. The level of walk for the leaf level 0bxx is encoded as:
  - `0b00` : Reserved. Treat as if TTL[3:2] is `0b00`.
  - `0b01` : If FEAT_LPA2 is implemented, level 1. Otherwise, treat as if TTL[3:2] is `0b00`.
  - ...

is changed to read:

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

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

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

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

is changed to read:

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

        - The entry is in VMSAv8-32 LPAE or VMSAv8-64 format and matches the granule and level specified in TTL.

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

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

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

is changed to read:

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

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

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

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

- For TLBI instructions, the TTL hint applies for 64-bit descriptors, including VMSAv8-64 and VMSAv8-32LPAE descriptors.

- For TLBIP instructions, the TTL hint applies for 128-bit descriptors.

2.22 D22806

In section D24.2.137 “PFAR_EL1, Physical Fault Address Register (EL1)” and D24.2.138 “PFAR_EL2, Physical Fault Address Register (EL2)”, the description of the field NS, bit [63], that reads:

> NS, bit [63]
>
> ...
>
> When EL3 is implemented:
>
> Non-secure. Reports the physical address space of the access that triggered the exception.

NS	Meaning
0b0	Secure physical address space.
0b1	Non-secure physical address space.

> The reset behavior of this field is:
>
> - On a Warm reset, this field resets to an architecturally **UNKNOWN** value.
>
> Otherwise:
>
> Reserved, **RES0**.

is changed to read:

> NS, bit [63]
>
> ...
>
> When EL3 is implemented or FEAT_Secure is implemented:
>
> Non-secure. Reports the physical address space of the access that triggered the exception.

NS	Meaning
0b0	Secure physical address space.
0b1	Non-secure physical address space.

> The reset behavior of this field is:
>
> - On a Warm reset, this field resets to an architecturally **UNKNOWN** value.
>
> Otherwise:
>
> Reserved, **RES0**.

2.23 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:
>
> - ...

2.24 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.

2.25 D23002

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

Event	Instructions added
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

2.26 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.

2.27 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)

2.28 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.

2.29 D23154

In section J1.1.3 “aarch64/functions”, a new function is added:

```
boolean IsPFARValid(FaultRecord fault);
```

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

```
IssType AArch64.FaultSyndrome(...)
  if d_side then
    ...
  else
    ...
    isstype.iss2<11> = if fault.hdbssf then '1' else '0';
    if IsExternalAbort(fault) then
      isstype.iss<9> = fault.extflag;
```

is changed to read:

```
IssType AArch64.FaultSyndrome(...)
  if d_side then
    ...
  else
    ...
    isstype.iss2<11> = if fault.hdbssf then '1' else '0';
    if (IsFeatureImplemented(FEAT_PFAR) && IsExternalSyncAbort(fault) &&
        !(EL2Enabled() && (HCR_EL2.VM == '1' || HCR_EL2.DC == '1') &&
        target_el == EL1)) then
      isstype.iss<14> = if IsPFARValid(fault) then '1' else '0';
    if IsExternalAbort(fault) then
      isstype.iss<9> = fault.extflag;
```

An equivalent change is made in AArch64.PhysicalSErrorSyndrome().

Additionally, in the function AArch64.TakePhysicalSErrorException(...), the code that reads:

```
AArch64.TakePhysicalSErrorException(...)
  ...
  constant bits(25) syndrome = AArch64.PhysicalSErrorSyndrome(implicit_esb);
  ...
```

is changed to read:

```
AArch64.TakePhysicalSErrorException(...)
  ...
  constant bits(25) syndrome = AArch64.PhysicalSErrorSyndrome(is_esb, implicit_esb);
  ...
  if except.syndrome.iss<14> == '1' then
    except.pavalid = TRUE;
    except.paddress = fault.paddress;
  else
    except.pavalid = FALSE;
    except.paddress = FullAddress UNKNOWN;
  ...
```

Equivalent changes are made in the following functions:

- AArch64.AbortSyndrome().
- HandleExternalTTWAbort().
- HandleExternalAbort().

2.30 C23225

In section B1.2.1 “FFR, First Fault Register” the text that reads:

> I_{IXLZQY}
>
> After a sequence of one or more SVE First-fault or Non-fault loads that follow a SETFFR instruction, the FFR contains a sequence of zero or more TRUE elements, followed by zero or more FALSE elements.

is changed to read:

> I_{IXLZQY}
>
> After a sequence of one or more SVE First-fault or Non-fault loads that follow a SETFFR instruction, without any intervening WRFFR instruction, the FFR contains a monotonic predicate value.

And the following text is added to the same section:

> D_{X0001}
>
> A monotonic predicate value is one that consists of, starting from predicate element 0, a sequence of zero or more TRUE elements, followed only by zero or more FALSE elements.
>
> I_{X0001}
>
> The WRFFR instruction requires that its source predicate contains a monotonic predicate value. If the source is not a monotonic predicate value, then the resulting value in the FFR is **UNKNOWN**.

2.31 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

2.32 C23271

In section D14.3 “Common event numbers”, the description of event “`0x0072`, LDST_SPEC, Operation speculatively executed, load or store” that reads:

> `0x0072`, LDST_SPEC, Operation speculatively executed, load or store
>
> The counter counts each operation counted by INST_SPEC that is a load operation or a store operation. See LD_SPEC and ST_SPEC for these classifications.
>
> When FEAT_PMUv3p8 is not implemented, this is an **IMPLEMENTATION DEFINED** event.

is changed to read:

> `0x0072`, LDST_SPEC, Operation speculatively executed, load or store
>
> The counter counts each operation counted by INST_SPEC that is a load operation or a store operation. See LD_SPEC, PRF_SPEC and ST_SPEC for these classifications.
>
> If the PRF_SPEC event is implemented and the prefetch operations counted by PRF_SPEC are not counted by LD_SPEC, then it is **IMPLEMENTATION DEFINED** whether these operations are counted by LDST_SPEC. Arm recommends that these operations are counted by LDST_SPEC.
>
> When FEAT_PMUv3p8 is not implemented, this is an **IMPLEMENTATION DEFINED** event.

In the same section, the description of event “0x0070, LD_SPEC, Operation speculatively executed, load” that reads:

> `0x0070`, LD_SPEC, Operation speculatively executed, load

> The counter counts each operation counted by LDST_SPEC that is a load operation. Operations due to Memory-reading instructions are counted as load operations.
>
> It is **IMPLEMENTATION DEFINED** whether the prefetch instructions PRFM, PLD, PLDW, and PLI count as integer data-processing operations or load operations. Arm recommends that if the instruction is not implemented as a **NOP** then it is counted as a load operation.
>
> When FEAT_PMUv3p8 is not implemented, this is an **IMPLEMENTATION DEFINED** event.

is changed to read:

> `0x0070`, LD_SPEC, Operation speculatively executed, load
>
> The counter counts each operation counted by LDST_SPEC that is a load operation. Operations due to Memory-reading instructions are counted as load operations.
>
> It is **IMPLEMENTATION DEFINED** whether operations counted by PRF_SPEC count as load operations. Arm recommends that if the instruction is not implemented as a **NOP** and PRF_SPEC is not implemented, then the operation is counted as a load operation. If the operation is counted by neither LD_SPEC nor LDST_SPEC, then it is counted as a data-processing operation.
>
> When FEAT_PMUv3p8 is not implemented, this is an **IMPLEMENTATION DEFINED** event.

The event “0x0080, LD_RETIRED, Operation architecturally executed, load” has the following text added:

> If the prefetch instructions defined to be counted by PRF_SPEC are counted by LD_SPEC then they are also counted by LD_RETIRED.

2.33 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.

2.34 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().

2.35 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”.

2.36 D23462

In section J1.3 “Shared Pseudocode”, in the pseudocode function IsDataAccess(), the code 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};
```

2.37 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 R_{FWRMB} 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.

2.38 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.

2.39 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:

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

2.40 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.

2.41 D23551

In section C6.2.96 “CPYP, CPYM, CPYE”, the text that reads:

For CPYP:

- After saturation is performed, the direction of the memory copy is based on the following:
  - If (Xs > Xd) and (Xd + saturated copy size) > Xs, then the direction is forward.
  - If (Xs < Xd) and (Xs + saturated copy size) > Xd, then the direction is backward.
  - Otherwise, the direction is an **IMPLEMENTATION DEFINED** choice between forward and backward.

is changed to read:

For CPYP:

- After saturation is performed, the direction of the memory copy is based on the following:
  - If (Xs <55:0> > Xd <55:0>) and (Xd <55:0> + saturated copy size) > Xs <55:0>, then the direction is forward.
  - If (Xs <55:0> < Xd <55:0>) and (Xs <55:0> + saturated copy size) > Xd <55:0> , then the direction is backward.
  - Otherwise, the direction is an **IMPLEMENTATION DEFINED** choice between forward and backward.

The equivalent changes are made in the following sections:

- C6.2.97 “CPYPN, CPYMN, CPYEN”.
- C6.2.98 “CPYPRN, CPYMRN, CPYERN”.
- C6.2.99 “CPYPRT, CPYMRT, CPYERT”.
- C6.2.100 “CPYPRTN, CPYMRTN, CPYERTN”.
- C6.2.101 “CPYPRTRN, CPYMRTRN, CPYERTRN”.
- C6.2.102 “CPYPRTWN, CPYMRTWN, CPYERTWN”.
- C6.2.103 “CPYPT, CPYMT, CPYET”.
- C6.2.104 “CPYPTN, CPYMTN, CPYETN”.
- C6.2.105 “CPYPTRN, CPYMTRN, CPYETRN”.

- C6.2.106 “CPYPTWN, CPYMTWN, CPYETWN”.
- C6.2.107 “CPYPWN, CPYMWN, CPYEWN”.
- C6.2.108 “CPYPWT, CPYMWT, CPYEWT”.
- C6.2.109 “CPYPWTN, CPYMWTN, CPYEWTN”.
- C6.2.110 “CPYPWTRN, CPYMWTRN, CPYEWTRN”.
- C6.2.111 “CPYPWTWN, CPYMWTWN, CPYEWTWN”.

2.42 C23552

In section C6.2.96 “CPYP, CPYM, CPYE”, the text that reads:

> For the CPYE option B, when PSTATE.C = 1:
>
> - ....
> - ....
> - If the copy is in the backward direction (PSTATE.N == 1), then: - Xs holds the highest address to be copied from. - Xd holds the highest address to be copied to. - On completion of the instruction:

is changed to read:

> For the CPYE option B, when PSTATE.C = 1:
>
> - ....
> - ....
> - If the copy is in the backward direction (PSTATE.N == 1), then: - Xs holds the highest address to be copied from + 1. - Xd holds the highest address to be copied to + 1. - On completion of the instruction:

Equivalent changes are made in the following sections:

- C6.2.97 “CPYPN, CPYMN, CPYEN”.
- C6.2.98 “CPYPRN, CPYMRN, CPYERN”.
- C6.2.99 “CPYPRT, CPYMRT, CPYERT”.
- C6.2.100 “CPYPRTN, CPYMRTN, CPYERTN”.
- C6.2.101 “CPYPRTRN, CPYMRTRN, CPYERTRN”.
- C6.2.102 “CPYPRTWN, CPYMRTWN, CPYERTWN”.
- C6.2.103 “CPYPT, CPYMT, CPYET”.
- C6.2.104 “CPYPTN, CPYMTN, CPYETN”.
- C6.2.105 “CPYPTRN, CPYMTRN, CPYETRN”.
- C6.2.106 “CPYPTWN, CPYMTWN, CPYETWN”.

- C6.2.107 “CPYPWN, CPYMWN, CPYEWN”.
- C6.2.108 “CPYPWT, CPYMWT, CPYETWT”.
- C6.2.109 “CPYPWTN, CPYMWTN, CPYETWTN”.
- C6.2.110 “CPYPWTRN, CPYMWTRN, CPYEWTRN”.
- C6.2.111 “CPYPWTWN, CPYMWTWN, CPYEWTWN”.

2.43 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.

2.44 C23587

In section B2.4.1.1 “Peripherals”, the definition of Out-of-band-ordered-before that reads:

A Read Memory or write effect RW_1 is Out-of-band-ordered-before a Read Memory or write effect RW_2 if either of the following cases apply:

- RW_1 appears in program order before a DSB instruction that begins an **IMPLEMENTATION DEFINED** instruction sequence indirectly leading to the generation of RW_2.
- RW_1 is Ordered-before a Read Memory or write effect RW_3 and RW_3 is Out-of-band-ordered-before RW_2.

If a Memory effect M_1 is Out-of-band-ordered-before a Read Memory or write effect M_2, then M_1 is seen to occur before M_2 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.

2.45 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”.

2.46 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.

2.47 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:

2.48 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.

2.49 D23689

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

> `0x802C`, FP_MUL_SPEC, Floating-point operation speculatively executed, multiply
>
> The counter counts each speculatively executed floating-point multiply operation counted by FP_SPEC due to any of the following A64 instructions:
>
> - Scalar: FMUL or FMULX.
> - Advanced SIMD: FMUL or FMULX.
> - SVE: FMUL, FMULX, or FTSMUL.
> - SVE2: BFMUL (indexed).
>
> It is **IMPLEMENTATION DEFINED** which floating-point multiply operations are counted in AArch32 state.

is changed to read:

> `0x802C`, FP_MUL_SPEC, Floating-point operation speculatively executed, multiply
>
> The counter counts each speculatively executed floating-point multiply operation counted by FP_SPEC due to any of the following A64 instructions:
>
> - Scalar: FMUL or FMULX.
> - Advanced SIMD: FMUL or FMULX.
> - SVE: FMUL, FMULX, or FTSMUL.
> - SVE2: BFMUL.
>
> Such that BFMUL (vectors, predicated) and BFMUL (vectors, unpredicated) are also counted.
>
> It is **IMPLEMENTATION DEFINED** which floating-point multiply operations are counted in AArch32 state.

A corresponding update adds “SVE2: BFMUL.” to the instruction-list for both SVE_FP_MUL_SPEC (0x802E) and ASE_SVE_FP_MUL_SPEC (0x802F).

2.50 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.

2.51 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'.

2.52 D23733

In section C6.2.328 "SETGP, SETGM, SETGE", the pseudocode for SETG when a fault occurs that reads:

```
if memset.implements_option_a then
    while memset.stagesetsize < 0 && !fault do
        (memory_set, memaddrdesc, memstatus) = MemSetBytes(
            memset.toaddress + memset.setsize,
            data, B, accdesc);
        if memory_set != B then
            fault = TRUE;
        else
            tagstep = B DIV TAG_GRANULE;
            tag = AArch64.AllocationTagFromAddress(
                memset.toaddress + memset.setsize);
            while tagstep > 0 do
                tagaddr = memset.toaddress + memset.setsize + (tagstep - 1) *
 TAG_GRANULE;
                AArch64.MemTag[tagaddr, accdesc] = tag;
                tagstep = tagstep - 1;
            end while
            memset.setsize = memset.setsize + B;
        end if
        memset.stagesetsize = memset.stagesetsize + B;
    end while
else
    ...  (similar loop for the reverse direction)
end if
```

is changed to read:

```
if memset.implements_option_a then
    (tags_written, memaddrdesc, memstatus) = MemSetTags(
        memset.toaddress + memset.setsize,  // start address
        data,                               // value to write
        B,                                  // block size
        accdesc);                           // access parameters
    if memstatus.statuscode != Fault_None then
        fault = TRUE;
    else
        memset.setsize      = memset.setsize + B;
        memset.stagesetsize = memset.stagesetsize + B;
    end if
else
    (tags_written, memaddrdesc, memstatus) = MemSetTags(
        memset.toaddress,
        data,
        B,
        accdesc);
    if memstatus.statuscode != Fault_None then
        fault = TRUE;
    else
```

```
        memset.toaddress    = memset.toaddress + B;
        memset.stagesetsize = memset.stagesetsize - B;
    end if
end if
```

2.53 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.

2.54 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}.

2.55 C23766

In section 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.

2.56 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.

2.57 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 “LDAPURSW”.

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”.

2.58 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.

2.59 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)”

2.60 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.

2.61 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.

2.62 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.

2.63 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.

2.64 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.

2.65 E23814

In section C7.2.332 “SRSHR”, under ‘Operational Information', the following text is added:

If PSTATE.DIT is 1, this instruction is required to honor the data-independent timing behaviors described in section B1.5.6 “About PSTATE.DIT”.

The equivalent changes are made in the following sections:

- C7.2.334 “SSHL”.
- C8.2.674 “SRSRA”.
- C8.2.868 “URSHL”.
- C8.2.872 “URSRA”.
- C7.2.406 “URSHR”.

These instructions are also added to the list of instructions in C5.2.4 “DIT, Data Independent Timing”, under ‘Data Processing – Scalar Floating-Point and Advanced SIMD'.

2.66 R23822

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.

2.67 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".

2.68 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.
>
> ...

2.69 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:

2.70 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.

2.71 D23852

In Section J1.1.1.62 “SPEConstructRecord” the pseudocode that reads:

```
                  // Check for overflow
                  constant boolean large_counters = boolean
IMPLEMENTATION_DEFINED "SPE 16bit counters";
                   if SPESampleCounter\[counter_index\] > 0xFFFF && large\_counters
 then
                      SPESampleCounter\[counter_index\] = 0xFFFF;
                elseif SPESampleCounter\[counter_index\] > 0xFFF then
                     SPESampleCounter\[counter_index\] = 0xFFF;
```

is changed to read:

```
                        // Check for overflow
                        if (boolean IMPLEMENTATION_DEFINED "SPE 16bit counters")
then
                     if SPESampleCounter\[counter_index\] > 0xFFFF then
                        SPESampleCounter\[counter_index\] = 0xFFFF;
else
                    if SPESampleCounter\[counter_index\] > 0xFFF then
                       SPESampleCounter\[counter_index\] = 0xFFF;
```

2.72 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.

2.73 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:
>
> ...

2.74 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.

2.75 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”

2.76 R23907

In section D10.3 “Memory region tagging types”, the text that reads:

> R_{JJRJP}
>
> The memory region tagging type for a memory region is:
> - Tagged if all of the following are true:
> - FEAT_MTE2 is implemented.
> - Allocation Tag Access is enabled for the memory region. For more information, see Allocation Tag Access controls.
> - The stage 1 attributes define the Tagged attribute.
> - The combined effects of stage 1 and stage 2 translations define the memory attributes as Normal memory, Inner and Outer Write-Back, Non-Transient, Read-Allocate and Write-Allocate.
> - Otherwise it is Canonically Tagged if all of the following are true:
> - FEAT_MTE_CANONICAL_TAGS is implemented.
> - Canonical Tagging is enabled for the memory region.
> - The stage 1 attributes do not define the Tagged attribute.
> - Otherwise it is Untagged.

is changed to read:

> R_{JJRJP}
>
> The memory region tagging type for a memory region is:
> - Tagged if all of the following are true:
> - FEAT_MTE2 is implemented.
> - Allocation Tag Access is enabled for the memory region. For more information, see Allocation Tag Access controls.
> - The stage 1 attributes define the Tagged attribute.

    - The combined effects of stage 1 and stage 2 translations define the memory attributes as Normal memory, Inner and Outer Write-Back, Non-Transient, Read-Allocate and Write-Allocate.
- Otherwise it is Canonically Tagged if all of the following are true:
    - FEAT_MTE_CANONICAL_TAGS is implemented.
    - Canonical Tagging is enabled for the memory region.
    - The stage 1 attributes do not define the Tagged attribute.
- Otherwise it is **CONSTRAINED UNPREDICTABLE** if all of the following are true:
    - FEAT_MTE_CANONICAL_TAGS is implemented.
    - Canonical Tagging is enabled for the memory region.
    - The stage 1 attributes define the Tagged attribute.
    - The combined effects of stage 1 and stage 2 translations do not define the memory attributes as Normal memory, Inner and Outer Write-Back, Non-Transient, Read-Allocate and Write-Allocate.
- Otherwise it is Untagged.

2.77 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.

2.78 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.
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.

2.79 R23928

In section B2.4.1.1 “Peripherals”, the text that reads:

Peripheral coherence order

The Peripheral coherence order of a Memory-mapped peripheral is a total order on all reads and writes to that peripheral.

Note:

The Peripheral coherence order for a Memory-mapped peripheral signifies the order in which accesses arrive at the endpoint.

For a Read Memory or write effect RW1 and a Read Memory or write effect RW2 to the same peripheral, then RW1 will appear in the Peripheral coherence order for the peripheral before RW2 if either of the following cases apply:

- RW1 and RW2 are accesses using Non-cacheable or Device attributes and RW1 is Ordered-before RW2.
- RW1 and RW2 are accesses using Device-nGnRE or Device-nGnRnE attributes, with the same XS attribute value, and RW1 appears in program order before RW2.

Note:

When FEAT_XS is implemented, if accesses marked with the Device-nGnRE or Device-nGnRnE attributes are within the same Memory-mapped peripheral, but the XS attribute is not the same on those accesses, the order of arrival at the endpoint is not defined by the architecture.

is changed to read:

Peripheral arrival order

The Peripheral arrival order of a Memory-mapped peripheral is a total order on all Memory Read Effects and Memory Write Effects to that peripheral.

For a Memory Effect E_1 and a Memory Effect E_2 to the same peripheral if any of the following applies:

- All of the following apply:
  - Any of the following applies:
    - E_1 is to Device-nGnRE Memory.
    - E_1 is to Device-nGnRnE Memory.
  - Any of the following applies:
    - E_2 is to Device-nGnRE Memory.
    - E_2 is to Device-nGnRnE Memory.
  - Any of the following applies:
    - FEAT_XS is not implemented.
    - All of the following apply:
      - FEAT_XS is implemented.
      - E_1 has the XS attribute.
      - E_2 has the XS attribute.
    - All of the following apply:
      - FEAT_XS is implemented.
      - E_1 does not have the XS attribute.
      - E_2 does not have the XS attribute.
  - E_1 is in program-order before E_2.

- All of the following apply:
  - Any of the following applies:
    - E_1 is to Device-nGRE Memory.
    - E_1 is to Device-nGnRE Memory.
    - E_1 is to Device-nGnRnE Memory.
  - Any of the following applies:
    - E_2 is to Device-nGRE Memory.
    - E_2 is to Device-nGnRE Memory.
    - E_2 is to Device-nGnRnE Memory.
  - E_1 is Ordered-before E_2.
- All of the following apply:
  - Any of the following applies:
    - E_1 is to Device Memory.
    - E_1 is to Non-cacheable Normal Memory.
  - Any of the following applies:
    - E_2 is to Device Memory.
    - E_2 is to Non-cacheable Normal Memory.
  - E_1 is Barrier-ordered-before E_2.

Then E_1 is inserted in the Peripheral arrival order for the peripheral before E_2 and it is not permitted that E_1 and E_2 are gathered into a single transaction. Otherwise, exactly one of following applies:

- E_1 and E_2 are gathered into a single transaction and appear as one Effect in Peripheral arrival order for the peripheral.
- E_1 is inserted in the Peripheral arrival order for the peripheral before E_2.
- E_2 is inserted in the Peripheral arrival order for the peripheral before E_1.

Note:

- When FEAT_XS is implemented, if E_1 and E_2 are to Device-nGnRE Memory or to Device-nGnRnE Memory and E_1 and E_2 are within the same Memory-mapped peripheral, but E_1 and E_2 use different XS attribute, the order of arrival at the endpoint is not defined by the architecture.

2.80 D23933

In section D24.2.167 “SCTLR_EL1, System Control Register (EL1)”, the description of the field EE, bit [25] is changed to reflect the addition of four new features (FEAT_BigEnd, FEAT_LittleEnd, FEAT_BigEndEL0, FEAT_LittleEndEL0).

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.

2.81 D23934

In section D14.3 “Common event numbers”, in the event description of “`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 of “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.

2.82 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”.

2.83 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.

2.84 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.

2.85 C23956

In section B2.6.9 “Data Synchronization Barrier”, the text that reads:

In addition, no instruction that appears in program order after the DSB instruction can alter any state of the system or perform any part of its functionality until the DSB completes other than:

- Being fetched from memory and decoded.
- Reading the general-purpose, SIMD&FP, SVE vector or predicate, Special-purpose, or System registers that are directly or indirectly read without causing side effects.

- If FEAT_ETS2 is not implemented, having any virtual addresses of loads and stores translated.

is changed to read:

In addition, no instruction that appears in program order after the DSB instruction can alter any state of the system or perform any part of its functionality until the DSB completes other than:

- Being fetched from memory and decoded, or any change to architectural or microarchitectural state resulting from being fetched or decoded.
- Indirectly or directly reading any of the following, so long as the reading of these resources does not cause a change in architectural state:
  - General-purpose registers
  - SIMD&FP registers
  - The Stack Pointer register
  - The Program Counter
  - If FEAT_SVE or FEAT_SME is implemented, SVE scalable vector registers or SVE predicate registers
  - If FEAT_SVE is implemented, the First Fault Register
  - If FEAT_SME is implemented, the ZA storage
  - If FEAT_SME2 is implemented, the ZT0 register
  - Special-purpose registers
  - System registers other than CNTPCTSS_EL0 and CNTVCTSS_EL0
- If FEAT_ETS2 is not implemented, translating any virtual addresses of instructions that generate Explicit Memory Effects.

Equivalent changes are made in section E2.3.5.3 “Data Synchronization Barrier”.

2.86 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

2.87 D23966

In section D8.5.2.3 “Hardware dirty state tracking structure”, the text that reads:

> R_{YHLRY}
>
> If the HDBSS is full and enabled, then the PE does not update any stage 2 descriptors from writable-clean to writable-dirty.
>
> R_{STJMN}
>
> If the HDBSS is full and enabled, or HDBSSPROD_EL2.FSC is set to any value other than `0b000000`, then all of the following apply:
>
> - The PE cannot append entries to the HDBSS.
> - Stage 2 dirty state hardware updates are prevented.
> - A non-speculative access that requires a stage 2 dirty state hardware update generates an MMU fault that is reported as an Instruction Abort or Data Abort taken to EL2.
>   - ESR_EL2 is populated as a stage 2 Permission fault as though VTCR_EL2.HD is 0, indicating the walk level of the stage 2 descriptor that needs to be updated.
>   - ESR_EL2.ISS2.HDBSSF is reported as 1, to indicate that the failed update was a result of the HDBSS being full or in an error state.

is changed to read:

> R_{YHLRY}
>
> If the HDBSS is full and enabled, then the PE does not update any stage 2 descriptors from writable-clean to writable-dirty.
>
> R_{STJMN}
>
> If the HDBSS is full and enabled, or HDBSSPROD_EL2.FSC is set to any value other than `0b000000`, then all of the following apply:
>
> - The PE cannot append entries to the HDBSS.
> - Stage 2 dirty state hardware updates are prevented.
> - Any access that requires a stage 2 dirty state hardware update generates a stage 2 Permission fault that is reported as it would be if VTCR_EL2.HD were 0.
> - If the fault is reported in ESR_EL2, ESR_EL2.ISS2.HDBSSF is reported as 1, to indicate that the failed update was a result of the HDBSS being full or in an error state.
>
> I_{X0001}
>
> MMU faults on speculative accesses are never reported.

If the HDBSS is full and enabled, and this generates a stage 2 Permission fault as part of resolving an AT E1 operation executed at EL2, it is reported in PAR_EL1 as a stage 2 Permission fault on a stage 1 translation table walk, at the level of the stage 2 walk that experienced the fault.

2.88 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.

Additionally, the description of the FT bit with 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

FT	Meaning
	0b0
0b1	Type filtering enabled. Samples not one of the selected operation types will not be recorded

> If this field is set to 1 and the PMSFCR_EL1.{ST, LD, B} bits are all set to zero, it is **CONSTRAINED UNPREDICTABLE** whether no samples are recorded or the PE behaves as if PMSFCR_EL1.FT is set to 0

is changed to read:

> FT, bit [1]
>
> Filter by operation type. The filter is controlled by the {ST, LD, B} and, if FEAT_SPE_EFT is implemented, {SIMD, FP} and {SIMDm, FPm, STm, LDm, Bm} fields.

FT	Meaning
0b0	Type filtering disabled.
0b1	Type filtering enabled. Samples not one of the selected operation types will not be recorded.

> If FEAT_SPE_EFT is not implemented, this field is 1, and the PMSFCR_EL1.{ST, LD, B} bits are all zero, then it is **CONSTRAINED UNPREDICTABLE** whether no samples are recorded or the PE behaves as if PMSFCR_EL1.FT is 0.

2.89 C23982

In section D24.2.197 “TTBR0_EL3, Translation Table Base Register 0 (EL3)”, under the condition “When FEAT_D128 is implemented and TCR_EL3.D128 == 1:”, the description of field CnP, bit [0] the text that reads:

> CnP, bit [0]
>
> Common not Private. In an implementation that includes FEAT_TTCNP, indicates whether each entry that is pointed to by VSTTBR_EL2 is a member of a common set that can be used by every PE in the Inner Shareable domain for which the value of VSTTBR_EL2.CnP is 1.

CnP	Meaning
0b0	The translation table entries pointed to by VSTTBR_EL2 are permitted to differ from the entries for VSTTBR_EL2 for other PEs in the Inner Shareable domain. This is not affected by the value of the current VMID.
0b1	The translation table entries pointed to by VSTTBR_EL2 are the same as the translation table entries for every other PE in the Inner Shareable domain for which the value of VSTTBR_EL2.CnP is 1 and the VMID is the same as the current VMID.

> This bit is permitted to be cached in a TLB.
>
> Note:
>
> If the value of VSTTBR_EL2.CnP bit is 1 on multiple PEs in the same Inner Shareable domain and those VSTTBR_EL2s do not point to the same translation table entries when using the current VMID, then the results of translations using VSTTBR_EL2 are **CONSTRAINED UNPREDICTABLE**, see ‘**CONSTRAINED UNPREDICTABLE** behaviors due to caching of control or data values'.
>
> When this register has an architecturally-defined reset value, this field resets to a value that is architecturally **UNKNOWN**.
>
> The reset behavior of this field is:
>
> - On a Warm reset, this field resets to an architecturally **UNKNOWN** value.

is changed to read:

> CnP, bit [0]
>
> Common not Private. This bit indicates whether each entry that is pointed to by TTBR0_EL3 is a member of a common set that can be used by every PE in the Inner Shareable domain for which the value of TTBR0_EL3.CnP is 1.

CnP	Meaning
0b0	The translation table entries pointed to by TTBR0_EL3, for the current translation regime, are permitted to differ from corresponding entries for TTBR0_EL3 on other PEs in the Inner Shareable domain.
0b1	The translation table entries pointed to by TTBR0_EL3 are the same as the translation table entries for every other PE in the Inner Shareable domain for which the value of TTBR0_EL3.CnP is 1.

> This bit is permitted to be cached in a TLB.

Note:

If the value of the TTBR0_EL3.CnP bit is 1 on multiple PEs in the same Inner Shareable domain and those TTBR0_EL3s do not point to the same translation table entries, then the results of translations using TTBR0_EL3 are **CONSTRAINED UNPREDICTABLE**, see '**CONSTRAINED UNPREDICTABLE** behaviors due to caching of control or data values'.

The reset behavior of this field is:

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

2.90 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.

2.91 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".

2.92 C23997

In section B2.2.1 "Requirements for single-copy atomicity", the statements that read:

For explicit memory effects generated from an Exception level the following rules apply:

...

- Reads to SIMD and floating-point registers of a single 64-bit or smaller quantity that is aligned to the size of the quantity being loaded are treated as single-copy atomic reads.
- Writes from SIMD and floating-point registers of a single 64-bit or smaller quantity that is aligned to the size of the quantity being stored are treated as single-copy atomic writes.
- Element or Structure Reads to SIMD and floating-point registers of 64-bit or smaller elements, where each element is aligned to the size of the element being loaded, have each element treated as a single-copy atomic read.
- Element or Structure Writes from SIMD and floating-point registers of 64-bit or smaller elements, where each element is aligned to the size of the element being stored, have each element treated as a single-copy atomic store.
- Reads to SIMD and floating-point registers of a 128-bit value that is 64-bit aligned in memory are treated as a pair of single-copy atomic 64-bit reads.
- Writes from SIMD and floating-point registers of a 128-bit value that is 64-bit aligned in memory are treated as a pair of single-copy atomic 64-bit writes.
- Atomicity rules for SIMD load and store instructions also apply to SVE load and store instructions.
- SVE predicated load and store instructions are performed as a sequence of memory element accesses.

...

is changed to read:

For explicit memory effects generated from an Exception level the following rules apply:

...

- Reads to SIMD and floating-point registers of a single 64-bit or smaller quantity that is aligned to the size of the quantity being loaded are treated as single-copy atomic reads.
- Writes from SIMD and floating-point registers of a single 64-bit or smaller quantity that is aligned to the size of the quantity being stored are treated as single-copy atomic writes.
- Element or Structure Reads to SIMD and floating-point registers of 64-bit or smaller elements, where each element is aligned to the size of the element being loaded, have each element treated as a single-copy atomic read.
- Element or Structure Writes from SIMD and floating-point registers of 64-bit or smaller elements, where each element is aligned to the size of the element being stored, have each element treated as a single-copy atomic store.
- Reads to SIMD and floating-point registers of a 128-bit value that is 64-bit aligned in memory are treated as a pair of single-copy atomic 64-bit reads.
- Writes from SIMD and floating-point registers of a 128-bit value that is 64-bit aligned in memory are treated as a pair of single-copy atomic 64-bit writes.
- SVE predicated load and store instructions have the same single-copy atomicity guarantees as for SIMD load and store instructions for 64-bit or smaller elements.
- An SVE predicated load or store of a 128-bit element that is 64-bit aligned is treated as a pair of 64-bit single-copy atomic accesses.

- SVE unpredicated load and store instructions are performed as a sequence of byte accesses.
- SVE unpredicated load and store instructions do not guarantee that any access larger than a byte will be performed as a single-copy atomic access.

...

2.93 D24019

In section J1.2 “Pseudocode for AArch32 operation”, the pseudocode for function AArch32.S1TTWParamsEL2() that reads:

```
S1TTWParams AArch32.S1TTWParamsEL2()
    S1TTWParams walkparams;
    ...
    walkparams.irgn = HTCR.SH0;
    walkparams.orgn = HTCR.IRGN0;
    walkparams.sh   = HTCR.ORGN0;
    ...
    return walkparams;
```

is changed to read:

```
S1TTWParams AArch32.S1TTWParamsEL2()
    S1TTWParams walkparams;
    ...
    walkparams.irgn = HTCR.IRGN0;
    walkparams.orgn = HTCR.IRGN0;
    walkparams.sh   = HTCR.SH0;
    ...
    return walkparams;
```

2.94 C24031

In section D18.2.3 “Counter packet”, the counter definitions that read:

0b00001	Issue latency. Cycle count from the operation being dispatched for issue to the operation being issued for execution. This counts any delay in waiting the operation being ready to issue. Included for all operations.
0b00010	Translation latency. Cycle count from a virtual address being passed to the MMU for translation to the result of the translation being available. Included for all load, store, and atomic operations.

are changed to read:

0b00001	Issue latency. The count of cycles when the operation is waiting to be issued.
0b00010	Translation latency. The count of cycles when at least one part of the operation is waiting for the MMU to complete an address translation. It is **IMPLEMENTATION DEFINED** whether a cycle is counted if a part of the operation is accessing memory.

Equivalent changes made in section D17.6 “The profiling data” and section D17.6.4 “Additional information for each profiled memory access operation”.

2.95 C24054

In section D8.2.12.1 “Behavior when stage 1 address translation is disabled”, under rule RPPDBS, the rule that reads:

> R_{RPPDBS}
>
> If stage 1 address translation is disabled, then all of the following apply to memory accesses that would otherwise be translated at stage 1:
>
> - The stage 1 IA is flat mapped to the OA.
> - No Translation faults, Access flag faults, or Permission faults can be generated.
> - Address size faults and Alignment faults can be generated.
> - ...

is changed to read:

> R_{PPDBS}
>
> If stage 1 address translation is disabled, then all of the following apply to memory accesses that would otherwise be translated at stage 1:
>
> - The stage 1 IA is flat mapped to the OA.
> - No Translation faults, Access flag faults, or Permission faults can be generated.
> - Address size faults and Alignment faults can be generated.
> - An **IMPLEMENTATION DEFINED** fault for an Unsupported Exclusive or Atomic access can be generated if one of the following applies:
>   - The translation regime is not the EL1&0 regime.
>   - The translation regime is the EL1&0 regime and the Effective value of HCR_{EL2}.DC is 0.
> - ...

The pseudocode function AArch64.S1DisabledOutput() with code that reads:

```
if !IsZero(va<addrtop:pamax>) then
    fault.statuscode = Fault_AddressSize;
elsif AArch64.S1HasAlignmentFaultDueToMemType(regime, accdesc, aligned,
 walkparams.ntlsmd, memattrs) then
    fault.statuscode = Fault_Alignment;
```

is changed to read:

```
if !IsZero(va<addrtop:pamax>) then
    fault.statuscode = Fault_AddressSize;
elsif AArch64.S1HasAlignmentFaultDueToMemType(regime, accdesc, aligned,
 walkparams.ntlsmd, memattrs) then
    fault.statuscode = Fault_Alignment;
elsif ((accdesc.exclusive || accdesc.atomicop) &&
```

```
        !(regime == Regime_EL10 && EL2Enabled() && HCR_EL2.DC == '1') &&
        ConstrainUnpredictableBool(Unpredictable_Atomic_MMU_IMPDEF_FAULT)) then
    fault.statuscode = Fault_Exclusive;
```

2.96 C24061

In section D23.3.1 “Instructions for accessing non-debug System registers”, the Note that reads:

> Note:
>
> ...
>
> All unused encodings in the range op0 == `0b11`, op1 == `0b000`, CRn == `0b0000`, CRm == {`0b0010-0b0111`}, op2 == {`0b000-0b111`} are defined to be accessible as Reserved, **RAZ** to ensure correct behavior if the encodings are used for ID registers in the future.

is changed to read:

> Direct accesses to some regions of the system register encoding space have a defined behavior even if no register is currently allocated or implemented for the encodings in those regions. The Feature ID space is the System register space in AArch64 with op0==3, op1=={0, 1, 3}, CRn==0, CRm=={0-7}, op2=={0-7}. The properties for this space are that:
>
> - For unused encodings in the part of the space that has op1==0 and CRm=={2-7}, the following apply:
>   - If FEAT_FGT is not implemented, it is **IMPLEMENTATION DEFINED** whether direct reads can be trapped by HCR_EL2.TID3.
>   - If FEAT_FGT is implemented, direct reads can be trapped by HCR_EL2.TID3.
>   - If FEAT_IDTE3 is implemented, direct reads can be trapped by SCR_EL3.TID3.
>   - Direct reads that are not trapped behave as Reserved, **RES0**, to ensure correct behavior if the encodings are used for ID registers in the future.
> - For unused encodings in the part of the space that has either op1==0 and CRm==0, or op1=={1, 3}, the following apply:
>   - If FEAT_IDST is not implemented, direct reads are **UNDEFINED** and therefore generate exceptions reported with EC syndrome value `0x00`.
>   - If FEAT_IDST is implemented, direct reads generate exceptions reported with EC syndrome value `0x18`.
>   - The behavior for direct reads is consistent with the pseudocode function UnimplementedIDRegister().

2.97 D24062

In section J1.3.1.99 “PMUCaptureEvent”, the code that reads:

```
PMUCaptureEvent()
    ...
    if  IsFeatureImplemented(FEAT_PCSRv8p9) && PMPCSCTL.SS == '1' then
        if pc_sample.valid && !debug_state then
            ...
        else
            SetPCSRUnknown();
    ...
```

is changed to read:

```
PMUCaptureEvent()
    ...
    if  IsFeatureImplemented(FEAT_PCSRv8p9) && PMPCSCTL.SS == '1' then
        if pc_sample.valid && !debug_state then
            ...
        else
            PMPCSR<31:0> = Ones(32);
    ...
```

2.98 C24064

In section D17.6.9 “Controlling the data that is collected”, the text that reads:

> Physical data addresses are collected only if one of the following is true:
>
> - PMSCR_EL1.PA is set to 1 and the Profiling Buffer is owned by Secure EL1, and Secure EL2 is disabled or is not implemented.
> - PMSCR_EL2.PA is set to 1 and the Profiling Buffer is owned by Secure or Non-secure EL2.
> - PMSCR_EL1.PA is set to 1 and PMSCR_EL2.PA is set to 1 and either the Profiling Buffer is owned by Non-secure EL1, or the Profiling Buffer is owned by Secure EL1 and Secure EL2 is implemented and enabled.
>
> If EL2 is not implemented or is disabled for the current Security state, the PE behaves as if PMSCR_EL2.PA is set to 1, other than for a direct read of the register.

is changed to read:

> Physical data addresses are collected only if one of the following is true:
>
> - The Effective value of PMSCR_EL1.PA is 1, the Effective value of PMSCR_EL2.PA is 1, and the Profiling Buffer is owned by EL1.
> - The Effective value of PMSCR_EL2.PA is 1 and the Profiling Buffer is owned by EL2.
>
> If EL2 is not implemented or is disabled for the owning Security state, then the Effective value of PMSCR_EL2.PA is 1.

A similar update is made in section D24.7.6 “PMSCR_EL2, Statistical Profiling Control Register (EL2)”.

2.99 C24066

In section D14.3.1 “Common architectural events”, in event “0x8000, SIMD_INST_RETIRED, Instruction architecturally executed, SIMD”, the text that reads:

> 0x8000, SIMD_INST_RETIRED, Instruction architecturally executed, SIMD
>
> The counter counts each architecturally executed SIMD instruction.
>
> It is **IMPLEMENTATION DEFINED** which architecturally executed SIMD instructions are counted in AArch32 state.
>
> When Armv9.5 is not implemented, it is **IMPLEMENTATION DEFINED** whether scalar loads and stores to SIMD&FP registers other than those listed above are counted.
>
> When Armv9.5 is implemented, other scalar loads and stores to SIMD&FP registers are not counted.

is changed to read:

> 0x8000, SIMD_INST_RETIRED, Instruction architecturally executed, SIMD
>
> The counter counts each architecturally executed instruction counted by INST_RETIRED that is a SIMD instruction.
>
> That is, the counter counts:
>
> - Advanced SIMD data-processing instructions. Advanced SIMD scalar data-processing instructions are not counted.
> - SVE and SME SIMD data-processing instructions. Non-SIMD SVE and SME data-processing instructions are not counted.
> - Advanced SIMD structure load/store of one or more SIMD&FP registers instructions.
> - Advanced SIMD load and replicate to one or more SIMD&FP registers instructions.
> - Scalar load/store of a SIMD&FP Q register or pair of Q registers instructions.
> - SVE and SME load/store instructions.
>
> It is **IMPLEMENTATION DEFINED** which architecturally executed SIMD instructions are counted in AArch32 state.
>
> When Armv9.5 is not implemented, it is **IMPLEMENTATION DEFINED** whether scalar loads and stores to SIMD&FP registers other than those listed above are counted.

When Armv9.5 is implemented, other scalar loads and stores to SIMD&FP registers are not counted.

2.100 D24086

In section D24.7.11 “PMSIDR_EL1, Sampling Profiling ID Register”, in the description of the field FDS, the text that reads:

From Armv8.9, if FEAT_SPE is implemented, the value 0 is not permitted.

is changed to read:

From Armv8.9, if FEAT_SPE_LDS is implemented, the value 0 is not permitted.

2.101 D24093

FEAT_TME is withdrawn from all future versions of *Arm® Architecture Reference Manual for A-profile architecture*.

2.102 C24096

In section D8.15.1.6 TLB conflict abort, the rule R_{QWTKL} that reads:

R_{QWTKL}

When a TLB conflict abort is generated, it is **IMPLEMENTATION DEFINED** whether it is a stage 1 abort or a stage 2 abort.

is changed to read:

R_{QWTKL}

For the EL1&0 regime, when a TLB conflict abort is generated and both stages of translation are enabled, then one of the following applies:

- If FEAT_BBML1 is implemented and the TLB conflict abort is generated due to changing the table or block size or Contiguous bit, then the abort is reported to EL2.
- Otherwise, it is **IMPLEMENTATION DEFINED** whether it is reported to EL1 or EL2.

The rule in section D8.17.2 Support levels for changing table or block size, R_{FWRMB}, is removed:

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.

Additionally, the following information statement in the same section, I_{GQTBJ}, is removed:

> I_{GQTBJ}
>
> When a TLB conflict abort is generated, it is **IMPLEMENTATION DEFINED** whether it is a stage 1 abort or a stage 2 abort.

For rule R_{ZMVZT} the text that reads:

> R_{ZMVZT}
>
> If EL2 is enabled and stage 2 of the EL1&0 translation regime is disabled, then a stage 2 abort cannot be generated.

is changed to read:

> R_{ZMVZT}
>
> A TLB Conflict abort cannot be generated for a disabled stage of translation.

2.103 D24099

In section J1.3.5 “shared/translation”, in the pseudocode function `GPCRegistersConsistent()`, the following code segment that reads:

```
boolean GPCRegistersConsistent()
  if IsFeatureImplemented(FEAT_RME_GPC3) then
    ...
    if GPCCR_EL3.GPCBW == '1' then
      ...
      if !GPCBW_EL3BWSTRIDEValid() then
        return FALSE;

      if !IsAligned(GPCBW_EL3.BWADDR, 1 << UInt(GPCBW_EL3.BWSIZE)) then
        return FALSE;
    else
      ...
```

is changed to read:

```
boolean GPCRegistersConsistent()
  if IsFeatureImplemented(FEAT_RME_GPC3) then
    ...
    if GPCCR_EL3.GPCBW == '1' then
      ...
      if !GPCBW_EL3BWSTRIDEValid() then
        return FALSE;
```

```
      constant integer bwstride = 1 << (40 + UInt(GPCBW_EL3.BWSTRIDE));
      constant integer bwaddr = UInt(GPCBW_EL3.BWADDR) << 30;
      if bwstride <= bwaddr then
        return FALSE;

      if !IsAligned(GPCBW_EL3.BWADDR, 1 << UInt(GPCBW_EL3.BWSIZE)) then
        return FALSE;
    else
      ...
```

2.104 D24101

In section D24.2.40 “ESR_EL1, Exception Syndrome Register (EL1)”, the description of the ISS2.GCS field that reads:

GCS	Meaning
0b0	The Data Abort is not due to a Guarded control stack data access.
0b1	The Data Abort is due to a Guarded control stack data access.

is changed to read:

GCS	Meaning
0b0	The Data Abort is not due to a Guarded control stack data access.
0b1	The Data Abort is due to a Guarded control stack data access. The ISV field is 0.

A similar update is made in sections D24.2.41 “ESR_EL2, Exception Syndrome Register (EL2)” and D24.2.42 “ESR_EL3, Exception Syndrome Register (EL3)”.

2.105 C24124

In section D20.4.2 PE error state classification, under the rule R_{LFXRD}, the text that reads:

> R_{LFXRD}
>
> If and only if all of the following are true, then on taking an Error exception the PE error state is recorded as Corrected (CE):
>
> - The error has been corrected and not silently propagated by the PE.
> - The Error exception is taken as an SError exception taken to AArch64 state.
> - Software can recover execution from the preferred return address of the exception. Because the error has been corrected, software does not need to take action to locate and repair the error.
> - The implementation has not elected to record the PE error state as any other type.

is changed to read:

> R_{LFXRD}

If and only if all of the following are true, then on taking an Error exception the PE error state is recorded as Corrected (CE):

- The error has been corrected and not silently propagated by the PE.
- The Error exception is taken as an SError exception taken to AArch64 state.
- Software can recover execution from the preferred return address of the exception. Because the error has been corrected, software does not need to take action to locate and repair the error.
- The implementation has not elected to record the PE error state as any other type.

Arm strongly recommends that a corrected error that is not silently propagated does not generate any error exception at the PE. The producer of the data that corrected the error might still record the error in a RAS error record, and generate a fault handling interrupt, as described in the Arm® Reliability, Availability, and Serviceability (RAS) System Architecture for A-profile architecture.

Additionally, the text that reads:

I_{VKMZB}

The PE error states are summarized by Figure D20-1. Figure D20-1 assumes the type of Error exception supports the resulting PE error state, never elects to record an error as a different PE error state when permitted, and does not show Uncategorized error or **IMPLEMENTATION DEFINED** syndrome.

is changed to read:

I_{VKMZB}

The PE error states for an Error exception generated when the PE consumes an uncorrected error are summarized by Figure D20-1. Figure D20-1 assumes the type of Error exception supports the resulting PE error state, that the error was neither corrected nor deferred, and the PE never elects to record an error as a different PE error state when permitted, an Uncategorized error syndrome, or an **IMPLEMENTATION DEFINED** syndrome.

2.106 D24129

In section B1.5.6 “About PSTATE.DIT” the following text is deleted:

- 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.

In section C8.2 “Alphabetical list of SVE instructions”, in the “Operational information” subsection of SVE instructions which were defined prior to FEAT_SVE2, but which are required to honor PSTATE.DIT when FEAT_SVE2 or FEAT_SME is implemented, the text that reads:

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

is changed to:

> If PSTATE.DIT is 1:
>
> ...

2.107 D24146

In section D5.5.2 Common microarchitectural events, the text that describes the PMU event `0x8056, SVE_SPEC, Operation speculatively executed, SVE data processing` that reads:

> The counter counts each operation counted by INST_SPEC that is an SVE data-processing operation.
>
> - Data-processing operations involving SVE scalable vector and predicate registers. This includes operations added by FEAT_SME which involve the SVE registers but do not involve any ZA or ZT registers. This does not include non-SIMD SVE instructions.
> - If the Cryptographic Extension and FEAT_SVE2 are implemented, the PMULLB, PMULLT (Q variants) instructions.
>
> This includes all operations that operate on the SVE registers, except those that are counted as one of the following:
>
> - Load or store operations.
> - Cryptographic data-processing operations other than those included above.
> - When FEAT_SME is implemented, SME data-processing operations.

is changed to read:

> The counter counts each operation counted by SE_SPEC that is an SVE data-processing operation.
>
> All operations that operate on the SVE scalable vector and predicate registers are counted as SVE data-processing operations, other than all of the following:
>
> - Load or store operations.
> - If the Cryptographic Extension is implemented, operations counted by CRYPTO_SPEC as Cryptographic data-processing operations.
> - When FEAT_SME is implemented, operations counted by SME_SPEC as SME data-processing operations.
> - Non-SIMD SVE instructions.

> If the Cryptographic Extension and FEAT_SVE2 are implemented, operations due to the PMULLB, PMULLT (Q variants) instructions are not counted by CRYPTO_SPEC and are counted as SVE data-processing operations. Operations due to instructions defined by FEAT_SME which involve the SVE registers but do not involve any ZA or ZT registers are counted as SVE data-processing operations.

Additionally, the text that describes the PMU event `0x835C, SME_SPEC, Operation speculatively executed, SME data processing` that reads:

> The counter counts each operation counted by SE_SPEC that is an SME data-processing operation.
>
> - Data-processing operations involving the ZA and ZT registers.
>
> This includes all operations that operate on the ZA or ZT registers, except those that are counted as load and store operations. Operations due to instructions added by FEAT_SME which involve the SVE registers but do not involve any ZA or ZT registers are counted as SVE data-processing operations. This does not include non-SIMD SME operations.

is changed to read:

> The counter counts each operation counted by SE_SPEC that is an SME data-processing operation.
>
> All operations that operate on the ZA or ZT registers are counted as SME data-processing operations, other than all of the following:
>
> - Load and store operations.
> - Non-SIMD SME operations.
>
> Operations due to instructions defined by FEAT_SME which involve the SVE registers but do not involve any ZA or ZT registers are counted as SVE data-processing operations.

2.108 C24218

In section C1.2.5 “SVE Condition code aliases” the text that reads:

> The SVE assembler syntax defines an alternative set of SVE condition code aliases for use with AArch64 conditional instructions, as follows:

is changed to read:

> The SVE and SME assembler syntax defines an alternative set of condition code aliases for use with AArch64 conditional instructions, as follows:

2.109 D24149

In section D1.3.4.2 “Illegal exception returns from AArch64 state”, the text that reads:

> R_{VWJHB}
>
> On an illegal exception return from an Exception level, ELx, all of the following occur:
>
> ...
>
> - All of the following are **UNKNOWN**:
>   - If FEAT_UAO is implemented, then PSTATE.UAO.
>   - If FEAT_DIT is implemented, then PSTATE.DIT.
>   - If FEAT_MTE is implemented, then PSTATE.TCO.
>   - If FEAT_SSBS is implemented, then PSTATE.SSBS.
>   - If FEAT_PAuth_LR is implemented, then PSTATE.PACM.
>
> ...

is changed to read:

> R_{VWJHB}
>
> On an illegal exception return from an Exception level, ELx, all of the following occur:
>
> ...
>
> - All of the following are **UNKNOWN**:
>   - If FEAT_UAO is implemented, then PSTATE.UAO.
>   - If FEAT_DIT is implemented, then PSTATE.DIT.
>   - If FEAT_MTE is implemented, then PSTATE.TCO.
>   - If FEAT_SSBS is implemented, then PSTATE.SSBS.
>   - If FEAT_BTI is implemented, then PSTATE.BTYPE.
>   - If FEAT_PAuth_LR is implemented, then PSTATE.PACM.
>
> ...

2.110 C24151

In section K14.6.1.2.3 “Resolving by the use of barriers and address dependency”, the following text is deleted:

> Where the value returned by a read is used for computation of the virtual address of a subsequent read or write, then these two memory accesses are observed in program order.

> Where the value returned by a read is used for computation of the virtual address of a subsequent read or write, this is called an address dependency. An address dependency exists even if the value returned by the first read has no effect on the virtual address. This might occur if the value returned is masked off before it is used, or if it confirms a predicted address value that it might have changed.
>
> This restriction applies only when the data value returned by a read is used as a data value to calculate the address of a subsequent read or write. It does not apply if the data value returned by a read determines the condition flags values, and the values of the flags are used for condition code evaluation to determine the address of a subsequent read, either through conditional execution or the evaluation of a branch. This is called a control dependency.
>
> Where both a control and address dependency exist, the ordering behavior is consistent with the address dependency.

2.111 D24153

In section K7.3 “Recommended authentication interface”, the text that reads:

> Arm recommends that any Trace extension has the same authentication interface as the PE it is connected to.

is changed to read:

> Any Trace extension has the same authentication interface as the PE it is connected to.

2.112 D24156

In section H9.2.7 “DBGDTRTX_EL0, Debug Data Transfer Register, Transmit” the text that reads:

> When DoubleLockStatus(), or !IsCorePowered(), or OSLockStatus(), or ! AllowExternalDebugAccess(), accesses to this register return an ERROR.

is changed to read:

> When DoubleLockStatus(), or !IsCorePowered(), or OSLockStatus(), accesses to this register return an ERROR.

In section J1.3.1 shared/debug, the pseudocode that reads:

```
// AllowExternalDebugAccess()
// ==========================
// Returns TRUE if an external debug interface access to the External debug
// registers
// is allowed for the given Security state, FALSE otherwise.
```

is changed to read:

```
// AllowExternalDebugAccess()
// ==========================
// Returns TRUE if an external debug interface access to the External
// DBGBVR<n>_EL1,
// DBGBCR<n>_EL1, DBGWVR<n>_EL1, DBGWCR<n>_EL1 registers, and, from Armv8.2, the
// OSLAR_EL1 register is allowed for the access. Returns FALSE otherwise.
```

2.113 D24162

In section D8.1.2.3 “Realm EL1&0 translation regime” the rule R_{BTYVG} with text that reads:

> If all of the following apply, then the Realm EL1&0 translation regime is used to translate addresses:
>
> - The memory accesses occur in Realm state.
> - Memory is accessed from EL1 or EL0.
> - The Effective value of HCR_EL2.TGE is 0.

is changed to read:

> If all of the following apply, then the Realm EL1&0 translation regime is used to translate addresses:
>
> - The memory accesses occur in Realm state.
> - One of the following applies:
>   - Memory is accessed from EL1.
>   - Memory is accessed from EL0 and the Effective value of HCR_EL2.{E2H,TGE} is not {1,1}.

Equivalent changes are made to:

- R_{QJSCR} (for the Non-secure EL1&0 translation regime).
- R_{DKYXN} (for the Secure EL1&0 translation regime).

2.114 D24166

In section J1.1 “Pseudocode for AArch64 operation”, in the function AArch64.S1ComputePermissions(), the code that reads:

```
if s1perms.overlay && s1perms.wxn == '1' && s1perms.ox == '1' then
    s1perms.ow = '0';
elsif s1perms.wxn == '1' then
    s1perms.x = '0';
```

is changed to read:

```
if s1perms.overlay then
     // If WXN and the overlay X permission is present, the overlay W permission is
 removed.
    s1perms.ow = s1perms.ow AND NOT(s1perms.wxn AND s1perms.ox);
else
    // If WXN and the W and X permissions are present, the X permission is removed.
    // The W and X permissions are factored into the WXN computation.
    s1perms.x  = s1perms.x AND NOT(s1perms.wxn);
```

2.115 D24180

In section D1.3.6.4.2 “Virtual interrupt masking”, the text which reads:

B	When the interrupt is pending, it might be subject to masking, as defined in R_{SNLJH} and R_{XSPSG}. If the interrupt is masked, it is not taken. If the interrupt is not masked, it is taken.

is changed to read:

B	When the interrupt is pending, it might be subject to masking, as defined in R_{SNLJH}, R_{XSPSG}, and R_{MMYBS}. If the interrupt is masked, it is not taken. If the interrupt is not masked, it is taken.

In the same section, the text that reads:

> R_{SNLJH}
>
> If the target Exception level of a virtual IRQ interrupt is the current Exception level, ELx, the following controls determine whether the interrupt is masked:

PSTATE.I	SCTLR_ELx.NMI	AllIntMask	vIRQ	vIRQ with Superpriority
0	0	x	Not Masked	Not Masked
0	1	0	Not Masked	Not Masked
0	1	1	Masked	Masked
1	0	x	Masked	Masked
1	1	0	Masked	Not Masked
1	1	1	Masked	Masked

> R_{XSPSG}
>
> If the target Exception level of a virtual FIQ interrupt is the current Exception level, ELx, the following controls determine whether the interrupt is masked:

PSTATE.F	SCTLR_ELx.NMI	AllIntMask	vFIQ	vFIQ with Superpriority
0	0	x	Not Masked	Not Masked
0	1	0	Not Masked	Not Masked
0	1	1	Masked	Masked
1	0	x	Masked	Masked

PSTATE.F	SCTLR_ELx.NMI	AllIntMask	vFIQ	vFIQ with Superpriority
1	1	0	Masked	Not Masked
1	1	1	Masked	Masked

> R_{MMYBS}
>
> If the target Exception level of a virtual SError exception is the current Exception level, ELx, the PSTATE.A control determines whether the interrupt is masked. However, if all of the following are true, the PSTATE.A control is ignored, and the interrupt is taken:
>
> - The target Exception level is the current Exception level.
> - FEAT_DoubleFault is implemented.
> - SCR_EL3.NMEA is 1.

is changed to read:

> R_{SNLJH}
>
> The following controls determine whether a virtual IRQ interrupt is masked:

PSTATE.I	SCTLR_EL1.NMI	PSTATE.EL	AllIntMask	vIRQ	vIRQ with Superpriority
0	0	xx	x	Not Masked	Not Masked
0	1	EL0	x	Not Masked	Not Masked
0	1	EL1	0	Not Masked	Not Masked
0	1	EL1	1	Masked	Masked
1	0	xx	x	Masked	Masked
1	1	EL0	x	Masked	Not Masked
1	1	EL1	0	Masked	Not Masked
1	1	EL1	1	Masked	Masked

> R_{XSPSG}
>
> The following controls determine whether a virtual FIQ interrupt is masked:

PSTATE.F	SCTLR_EL1.NMI	PSTATE.EL	AllIntMask	vFIQ	vFIQ with Superpriority
0	0	xx	x	Not Masked	Not Masked
0	1	EL0	x	Not Masked	Not Masked
0	1	EL1	0	Not Masked	Not Masked
0	1	EL1	1	Masked	Masked
1	0	xx	x	Masked	Masked
1	1	EL0	x	Masked	Not Masked
1	1	EL1	0	Masked	Not Masked
1	1	EL1	1	Masked	Masked

> R_{MMYBS}

The following controls determine whether a virtual SError exception is masked:

PSTATE.A	PSTATE.EL	vSError
0	xx	Not Masked
1	xx	Masked

Additionally, and info statement is added to the same section:

I$_{X0001}$

The ability to execute at EL0 with virtual interrupts taken to EL1 masked is required by some user level driver code.

In section D1.3.6.4.3 “Delegated SError exception masking”, the text that reads:

B	When the exception is pending, it might be subject to masking. If the exception is masked, it is not taken. If the exception is not masked, it is taken.

is changed to read:

B	When the exception is pending, it might be subject to masking. If PSTATE.A is 1, then the exception is masked and is not taken. If PSTATE.A is 0, then the exception is not masked and is taken.

2.116 C24190

In section A1.5.9.9 “Combinations of floating-point exceptions” the following text is added:

If FPCR.AH is 1 and the result of a floating-point operation is a denormalized number that is flushed to zero, both an Inexact floating-point exception and an untrapped Underflow floating-point exception are generated, which sets the cumulative Underflow exception flag FPSR.UFC to 1 regardless of whether or not the Inexact exception is trapped.

2.117 C24218

In section J1.1.1 “aarch/debug”, in the pseudocode function SPECollectRecord(), the code that reads:

```
constant boolean is_lat = (total_latency < UInt(PMSLATFR_EL1.MINLAT));
```

is changed to read:

```
constant boolean is_lat = (total_latency >= UInt(PMSLATFR_EL1.MINLAT));
```