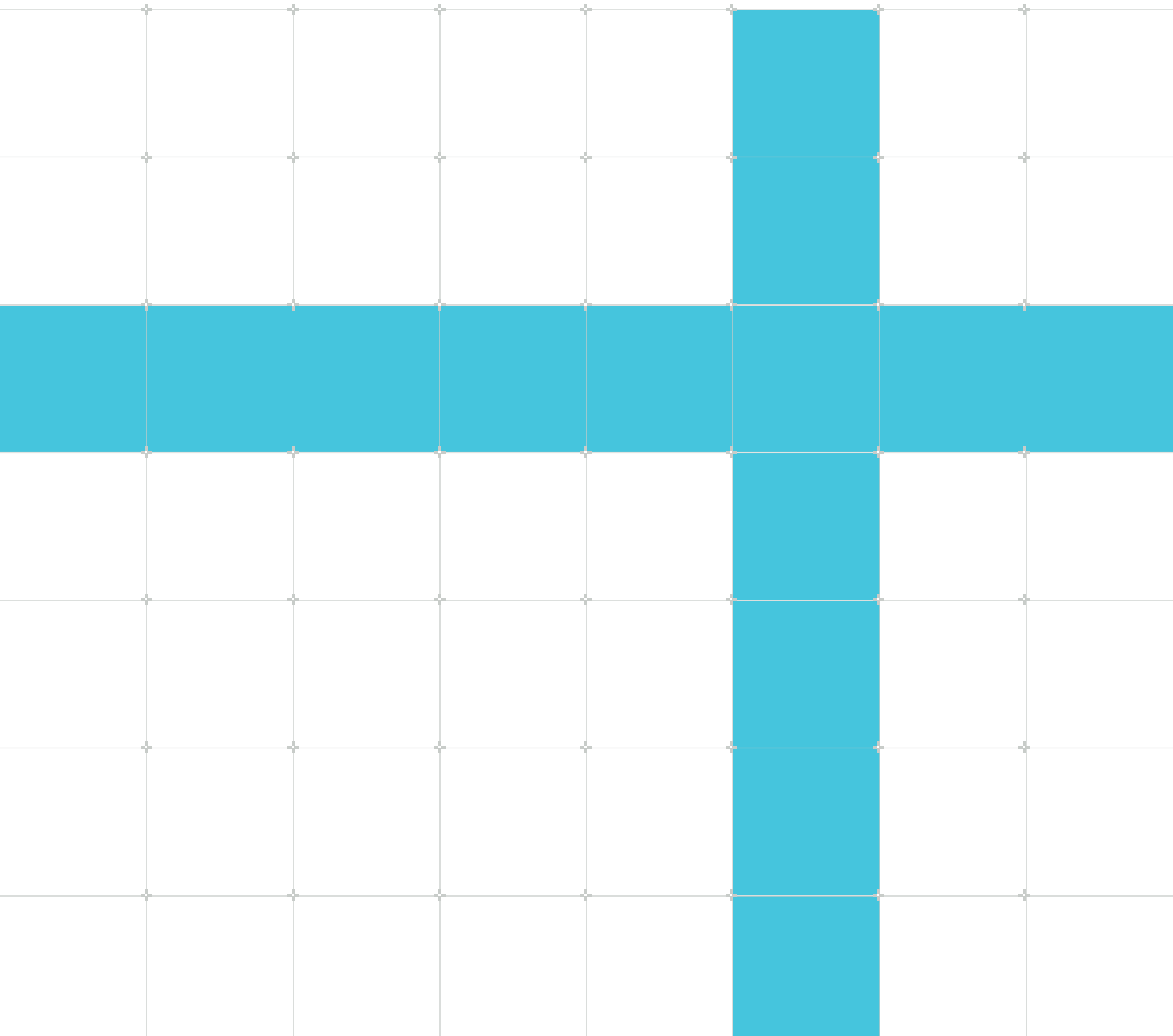
Arm® Architecture Reference Manual for A-profile architecture

Known issues in Issue K.a

Issue 08
102105_K.a_08_en



Arm® Architecture Reference Manual for A-profile architecture

Known issues in Issue K.a

Release information

Document history

Issue	Date	Confidentiality	Change
K.a-08	30 November 2024	Non-Confidential	Known Issues in Arm® Architecture Reference Manual, Issue K.a, as of 30 November 2024
K.a-07	30 October 2024	Non-Confidential	Known Issues in Arm® Architecture Reference Manual, Issue K.a, as of 30 October 2024
K.a-06	4 October 2024	Non-Confidential	Known Issues in Arm® Architecture Reference Manual, Issue K.a, as of 1 October 2024
K.a-05	2 September 2024	Non-Confidential	Known Issues in Arm® Architecture Reference Manual, Issue K.a, as of 2 September 2024
K.a-04	9 August 2024	Non-Confidential	Known Issues in Arm® Architecture Reference Manual, Issue K.a, as of 5 August 2024
K.a-03	3 July 2024	Non-Confidential	Known Issues in Arm® Architecture Reference Manual, Issue K.a, as of 3 July 2024
K.a-02	3 June 2024	Non-Confidential	Known Issues in Arm® Architecture Reference Manual, Issue K.a, as of 3 June 2024
K.a-01	6 May 2024	Non-Confidential	Known Issues in Arm® Architecture Reference Manual, Issue K.a, as of 6 May 2024
K.a-00	18 March 2024	Non-Confidential	Known Issues in Arm® Architecture Reference Manual, Issue K.a, as of 18 March 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

1.1 Conventions

The following subsections describe conventions used in Arm documents.

Glossary

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

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

Typographic conventions

Arm documentation uses typographical conventions to convey specific meaning.

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

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

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

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

This information is important and needs your attention.

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

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

1.2 Useful resources

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

Access to Arm documents depends on their confidentiality:

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

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

1.3 Other information

See the Arm website for other relevant information.

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

2. Known issues

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

Key

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

2.1 D15124

In section **SPSR_EL1, Saved Program Status Register (EL1)"**, under the heading 'When exception taken from AArch64 state:', the text that reads:

> M[3:0], bits [3:0]
>
> AArch64 Exception level and selected Stack Pointer.

M[3:0]	Meaning
0b0000	EL0.
0b0100	EL1 with SP_EL0 (EL1t).
0b0101	EL1 with SP_EL1 (EL1h).

is changed to read:

> M[3:0], bits [3:0]
>
> AArch64 Exception level and selected Stack Pointer.

M[3:0]	Meaning
0b0000	EL0.
0b0100	EL1 with SP_EL0 (EL1t).
0b0101	EL1 with SP_EL1 (EL1h).
0b1000	EL1 with SP_EL0 (EL1t) and either HCR_EL2.{NV, NV1} is {1,0} or HCR_EL2.{NV, NV2} is {1,1}.
0b1001	EL1 with SP_EL1 (EL1h) and either HCR_EL2.{NV, NV1} is {1,0} or HCR_EL2.{NV, NV2} is {1,1}.

In the same section, the text that reads:

> The bits in this field are interpreted as follows:
>
> - M[3:2]: On an exception to EL1:

    - If the Effective value of HCR_EL2.{NV, NV1} is not {1,0} or the exception is not taken from EL1, then M[3:2] is set to the value of PSTATE.EL on taking an exception to EL1.
    - If the Effective value of HCR_EL2.{NV, NV1} is {1,0} and the exception is taken from EL1, then M[3:2] is set to `0b10`.
- M[3:2] is copied to PSTATE.EL on executing a legal exception return operation in EL1.
- M[1] is unused and is 0 for all non-reserved values.
- M[0] is set to the value of PSTATE.SP on taking an exception to EL1 and copied to PSTATE.SP on executing an exception return operation in EL1.

is changed to read:

The bits in this field are interpreted as follows:

- M[3:2]: On an exception to EL1:
    - If the exception is taken from EL0
        - M[3:2] is set to the value of PSTATE.EL on taking an exception to EL1.
    - If the exception is taken from EL1:
        - If the Effective value of HCR_EL2.{NV, NV1} is not {1,0} and the Effective value of HCR_EL2.{NV, NV2} is not {1,1} then M[3:2] is set to the value of PSTATE.EL on taking an exception to EL1.
        - If the Effective value of HCR_EL2.{NV, NV1} is {1,0} or if the Effective value of HCR_EL2.{NV, NV2} is {1,1} then M[3:2] is set to `0b10`.
    - M[3:2] is copied to PSTATE.EL on executing a legal exception return operation in EL1.
- M[1] is unused and is 0 for all non-reserved values.
- M[0] is set to the value of PSTATE.SP on taking an exception to EL1 and copied to PSTATE.SP on executing an exception return operation in EL1.

2.2 D15259

In section **D23.2.2 “ACTLR_EL1, Auxiliary Control Register (EL1)”** under the heading ‘Accessing ACTLR_EL1’, the text that reads:

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TACR == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'1x1'} then
        X[t, 64] = NVMem[0x118];
    else
        X[t, 64] = ACTLR_EL1;
elsif PSTATE.EL == EL2 then
    X[t, 64] = ACTLR_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = ACTLR_EL1;
```

is changed to read:

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TACR == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'1x1'} then
        if EffectiveHCR_EL2_NVx() == '101' && boolean IMPLEMENTATION_DEFINED "IMP
 DEF ACTLR_ELx access implemented" then
            X[t, 64] = ACTLR_EL1;
        else
            X[t, 64] = NVMem[0x118];
    else
        X[t, 64] = ACTLR_EL1;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) && boolean IMPLEMENTATION_DEFINED "IMP DEF ACTLR_ELx access
 implemented" then
        X[t, 64] = ACTLR_EL2;
    else
        X[t, 64] = ACTLR_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = ACTLR_EL1;
```

The equivalent changes are made in the MSR ACTLR_EL1 accessors.

The ACTLR_EL12 accessors are added, with encoding op0:`0b11` op1:`0b101` CRn:`0b0001` CRm:`0b0000` op2:`0b001`, and the condition When boolean IMPLEMENTATION_DEFINED “IMP DEF ACTLR_ELx access implemented”.

The text is added:

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() == '101' then
        X[t, 64] = NVMem[0x118];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        X[t, 64] = ACTLR_EL1;
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL2) then
        X[t, 64] = ACTLR_EL1;
    else
        UNDEFINED;
```

An equivalent MSR ACTLR_EL12 accessor is added.

In section **D23.2.3 “ACTLR_EL2, Auxiliary Control Register (EL2)”** under the heading ‘Accessing ACTLR_EL2’, the ACTLR_EL1 accesses are changed as described above, under the condition ‘When an implementation implements ACTLR_ELx accessor behavior’

2.3 D16137

In section **D23.2.159 "SCTLR_EL1, System Control Register (EL1)"**, in the field description of 'WXN, bit[19]', the text that reads:

> Write permission implies XN (Execute-never). For the EL1&0 translation regime, this bit can force all memory regions that are writable to be treated as XN.

WXN	Meaning
0b0	This control has no effect on memory access permissions.
0b1	Any region that is writable in the EL1&0 translation regime is forced to XN for accesses from software executing at EL1 or EL0.

is changed to read:

WXN	Meaning
0b0	This control has no effect on memory access permissions.
0b1	In the EL1&0 translation regime, any region of memory that is writeable at EL1 is XN at EL1, and any region of memory that is writeable at EL0 is XN at EL0.

In section **D23.2.160 "SCTLR_EL2, System Control Register (EL2)"**, in the field description of 'WXN, bit[19]', the text that reads:

> Write permission implies XN (Execute-never). For the EL2 or EL2&0 translation regime, this bit can force all memory regions that are writable to be treated as XN.

WXN	Meaning
0b0	This control has no effect on memory access permissions.
0b1	Any region that is writable in the EL2 or EL2&0 translation regime is forced to XN for accesses from software executing at EL2.

is changed to read:

> Write permission implies XN (Execute-never). For the EL2 or EL2&0 translation regime, this bit can restrict execute permissions on writeable pages.

WXN	Meaning
0b0	This control has no effect on memory access permissions.
0b1	In the EL2 or EL2&0 translation regime, any region of memory that is writeable at EL2 is XN at EL2. In the EL2&0 translation regime, any region of memory that is writeable at EL0 is XN at EL0.

2.4 D16689

In section **D8.4.5.3 "PSTATE.BTYPE"**, the table in R_{YWFHD} that reads:

Instruction executed	Memory region	Register accessed	PSTATE.BTYPE
...	...	...	...

Instruction executed	Memory region	Register accessed	PSTATE.BTYPE
Any instruction except BR, BRAA, BRAAZ, BRAB, BRABZ, BLR, BLRAA, BLRAAZ, BLRAB, BLRABZ	Any	Any register	0b00

is changed to read:

Instruction executed	Memory region	Register accessed	PSTATE.BTYPE
...	...	...	...
ERET, ERETAA, ERETAB and DRPS	Any	Any register	SPSR_ELx.BTYPE
Any other instruction	Any	Any register	0b00

2.5 D16998

In section **D23.2.160 "SCTLR_EL2, System Control Register (EL2)"**, the text that reads:

0b0	EL0 access to these instructions is disabled, and these instructions are trapped to EL1.

is changed to read:

0b0	EL0 access to these instructions is disabled, and these instructions are trapped to EL2.

2.6 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'.

2.7 C17536

In section **D13.12.3.2 "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.

2.8 D17610

In section **D23.12.2 MPAM1_EL1, MPAM1 Register (EL1)**, in the field description of ‘ALTSP_FRCD, bit [54]’, the text that reads:

Access to this field is RO.

is changed to read:

> Accessing this field has the following behavior:
>
> - Access is **RAO/WI** if all of the following are true:
>   - MPAM3_EL3.ALTSP_HEN == 1
>   - MPAM2_EL2.ALTSP_HFC == 1
> - Access is **RAO/WI** if all of the following are true:
>   - MPAM3_EL3.ALTSP_HEN == 0
>   - MPAM3_EL3.ALTSP_HFC == 1
> - Otherwise, access to this field is **RAZ/WI**.

2.9 D18330

Arm® Architecture Reference Manual for A-profile architecture, Issue K.a is somewhat inconsistent in its use of 'prefetch' and 'preload' to describe the bringing in of items into caches either by hardware prediction or as a result of some prefetch or preload instructions.

In future versions of *Arm® Architecture Reference Manual for A-profile architecture*, this will be cleaned up. The term 'prefetch' will be used for this functionality, with 'hardware prefetch' used where the prefetch is predicted by hardware, and 'software prefetch' used where the prefetch is prompted by particular instructions (such as the AArch64 PRFM or AArch32 PLD instructions).

2.10 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);
```

2.11 D18886

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

```
integer AArch64.PhysicalAddressSize(bit d128, bits(3) encoded_ps, TGx tgx)
    ...
    if !IsFeatureImplemented(FEAT_D128) || d128 == '0' then
        if tgx != TGx_64KB && !IsFeatureImplemented(FEAT_LPA2) then
            max_ps = Min(48, AArch64.PAMax());
        elsif !IsFeatureImplemented(FEAT_LPA) then
            max_ps = Min(48, AArch64.PAMax());
        else
            max_ps = Min(52, AArch64.PAMax());
        end
    else
        max_ps = AArch64.PAMax();
    end

    return Min(ps, max_ps);
```

is changed to read:

```
integer AArch64.PhysicalAddressSize(bit d128, bit ds, bits(3) encoded_ps, TGx tgx)
    ...
    if d128 == '1' then
        max_ps = AArch64.PAMax();
    elsif IsFeatureImplemented(FEAT_LPA) && (tgx == TGx_64KB || ds == '1') then
        max_ps = Min(52, AArch64.PAMax());
```

```
    else
        max_ps = Min(48, AArch64.PAMax());
    end

    return Min(ps, max_ps);
```

In the same section, in the pseudocode function AArch64.S1TTBaseAddress(), the code that reads:

```
bits(56) AArch64.S1TTBaseAddress(S1TTWParams walkparams, Regime regime, bits(N)
 ttbr)
    ...
    if IsFeatureImplemented(FEAT_D128) && walkparams.d128 == '1' then
        tsize = Max(tsize, 5);
        if regime == Regime_EL3 then
            tablebase<55:5> = ttbr<55:5>;
        else
            tablebase<55:5> = ttbr<87:80>:ttbr<47:5>;
    elsif ((IsFeatureImplemented(FEAT_LPA) && walkparams.tgx == TGx_64KB &&
             walkparams.ps == '110') || (walkparams.ds == '1')) then
        tsize = Max(tsize, 6);
    ...
    tablebase = Align(tablebase, 1 << tsize);
    return tablebase;
```

is changed to read:

```
bits(56) AArch64.S1TTBaseAddress(S1TTWParams walkparams, Regime regime, bits(N)
 ttbr)
    ...
    if walkparams.d128 == '1' then
        tsize = Max(tsize, 5);
        if regime == Regime_EL3 then
            tablebase<55:5> = ttbr<55:5>;
        else
            tablebase<55:5> = ttbr<87:80>:ttbr<47:5>;
        end
    elsif walkparams.ds == '1' || (walkparams.tgx == TGx_64KB && walkparams.ps ==
 '110' &&
            (IsFeatureImplemented(FEAT_LPA) ||
             boolean IMPLEMENTATION_DEFINED "BADDR expresses 52 bits for 64KB
 granule")) then
        tsize = Max(tsize, 6);
        tablebase<51:6> = ttbr<5:2>:ttbr<47:6>;
    else
        tablebase<47:1> = ttbr<47:1>;
    end
    ...
    tablebase = Align(tablebase, 1 << tsize);
    return tablebase;
```

An equivalent change is made in the pseudocode function S2TTBaseAddress().

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

```
bits(56) AArch64.NextTableBase(bits(N) descriptor, bit d128, bits(2) skl, bit ds,
 TGx tgx)
   ...
       if IsFeatureImplemented(FEAT_D128) && d128 == '1' then
           tablebase<55:48> = descriptor<55:48>;
           return tablebase;
       if IsFeatureImplemented(FEAT_LPA) && tgx == TGx_64KB then
           tablebase<51:48> = descriptor<15:12>;
```

```
            return tablebase;
        if ds == '1' then
            tablebase<51:48> = descriptor<9:8>:descriptor<49:48>;
            return tablebase;
    return tablebase;
```

is changed to read:

```
bits(56) AArch64.NextTableBase(bits(N) descriptor, bit d128, bits(2) skl, bit ds,
 TGx tgx)
    ...
        if d128 == '1' then
            tablebase<55:48> = descriptor<55:48>;
        elsif tgx == TGx_64KB && (AArch64.PAMax() >= 52 ||
                boolean IMPLEMENTATION_DEFINED "descriptor[15:12] for 64KB granule
 are OA[51:48]") then
            tablebase<51:48> = descriptor<15:12>;
        elsif ds == '1' then
            tablebase<51:48> = descriptor<9:8>:descriptor<49:48>;
    return tablebase;
```

An equivalent change is made in the pseudocode function LeafBase().

2.12 D19428

In section **D23.2.155 "SCR_EL3, Secure Configuration Register"**, in the field description of 'RW, bit [10]', the text that reads:

> If AArch32 state is supported by the implementation at EL1, SCR_EL3.NS == 1 and AArch32 state is not supported by the implementation at EL2, the Effective value of this bit is 1.

is changed to read:

> If AArch32 state is supported by the implementation at EL1, SCR_EL3.NS == 1 and EL2 is implemented and AArch32 state is not supported by the implementation at EL2, the Effective value of this bit is 1.

2.13 D19431

In section **D8.14.3 "MMU faults generated by cache maintenance operations"**, under rule R_{BBLTJ}, the text that reads:

> R_{BBLTJ}
>
> If the Effective value of HCRX_EL2.CMOW is 1, then when executing an IC IVAU, DC CIVAC, DC CIGVAC, or DC CIGDVAC instruction at EL1 or EL0 that has stage 2 read permission, but does not have stage 2 write permission, a stage 2 Permission fault is generated.

is changed to read:

> R_{BBLTJ}
>
> If the Effective value of HCRX_EL2.CMOW is 1, then when executing an IC IVAU, DC CIVAC, DC CIGVAC, or DC CIGDVAC instruction at EL1 or EL0 that does not have stage 2 write permission, a stage 2 Permission fault is generated.

In section **D7.5.9.1 "The instruction cache maintenance instruction (IC)"**, the text that reads:

> The instruction cannot generate a Permission fault, except for:
>
> - The possible generation of a Permission fault by the execution of an IC IVAU instruction at EL0 when the specified address does not have read access at EL0, as described in EL0 accessibility of cache maintenance instructions.
> - When FEAT_CMOW is implemented, the possible generation of a Permission fault by:
>   - The execution of an IC IVAU instruction at EL0 when the specified address has stage 1 read permission, but does not have stage 1 write permission.
>   - The execution of an IC IVAU instruction at EL1 or EL0 when the specified address has stage 2 read permission, but does not have stage 2 write permission.

is changed to read:

> The instruction cannot generate a Permission fault, except for:
>
> - The possible generation of a Permission fault by the execution of an IC IVAU instruction at EL0 when the specified address does not have read access at EL0, as described in EL0 accessibility of cache maintenance instructions.
> - When FEAT_CMOW is implemented, the possible generation of a Permission fault by:
>   - The execution of an IC IVAU instruction at EL0 when the specified address has stage 1 read permission, but does not have stage 1 write permission.
>   - The execution of an IC IVAU instruction at EL1 or EL0 when the specified address does not have stage 2 write permission.

The equivalent changes are made in the following sections:

- **D7.5.9.2 "The data cache maintenance instruction (DC)"**.
- **D23.2.54 "HCRX_EL2, Extended Hypervisor Configuration Register"**, in the description of field 'CMOW, bit[9]'.

2.14 D19549

In section **D13.12.3.2 "Common microarchitectural events"**, for each TRCEXTOUT<n> event, where <n> is 0 to 3, the text that reads:

> This event must be implemented if FEAT_ETE is implemented.

is changed to read:

This event must be implemented if FEAT_ETE is implemented and the ETE implements External output <n>.

2.15 R19582

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

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

is changed to read:

Accesses to translation tables by AT instructions are given the MPAM information specified for translation table accesses by a data load instruction that is issued from an **IMPLEMENTATION DEFINED** choice of:

- The Exception level that the AT instruction was executed from.
- The Exception level that configures the translation regime targeted by the AT instruction.

2.16 D19586

In section **D23.2.155 "SCR_EL3, Secure Configuration Register"**, in the field description of 'HXEn, bit [38]', the text that reads:

When FEAT_HCX is implemented: Enables access to the HCRX_EL2 register at EL2 from EL3.

0b0	Accesses at EL2 to HCRX_EL2 are trapped to EL3. Indirect reads of HCRX_EL2 return 0.
0b1	This control does not cause any instructions to be trapped.

is changed to read:

When FEAT_HCX is implemented: Enables access to the HCRX_EL2 register at EL2 from EL3.

0b0	Accesses at EL2 to HCRX_EL2 are trapped to EL3.
0b1	This control does not cause any instructions to be trapped.

When EL3 is not implemented, the Effective value of this field is 1.

In section **D23.2.54 "HCRX_EL2, Extended Hypervisor Configuration Register"**, under the 'Configuration' heading, the following text is removed:

The bits in this register behave as if they are 0 for all purposes other than direct reads of the register if:

- EL2 is not enabled in the current Security state.

- SCR_EL3.HXEn is 0.

In the same section, in the following field descriptions:

- 'PACMEn, bit [24]'.
- 'EnFPM, bit [23]'.
- 'GCSEn, bit [22]'.
- 'EnIDCP128, bit [21]'.
- 'PTTWI, bit [16]'.
- 'MSCEn, bit [11]'.
- 'EnASR, bit [2]'.
- 'EnALS, bit [1]'.
- 'EnAS0, bit [0]'.

instances of 'the Effective value' are changed to read:

The Effective value of this field is 1 when any of the following are true:

- EL2 is not implemented or not enabled in the current Security state.
- The Effective value of HCR_EL2.{E2H, TGE} is {1, 1}.

The Effective value of this field is 0 when all of the following are true:

- EL2 is implemented and enabled in the current Security state.
- The Effective value of HCR_EL2.{E2H, TGE} is not {1, 1}.
- The Effective value of SCR_EL3.HXEn is 0.

In the same section, in the following field descriptions:

- 'D128En, bit [17]'.
- 'SCTLR2En, bit [15]'.
- 'TCR2En, bit [14]'.

instances of 'the Effective value' are changed to read:

When EL2 is not implemented or not enabled in the current Security state, the Effective value of this field is 1. Otherwise, when the Effective value of SCR_EL3.HXEn is 0, the Effective value of this field is 0.

In the same section, in the following field descriptions:

- 'TMEA, bit [19]'.
- 'CMOW, bit [9]'.
- 'SMPME, bit [5]'.

instances of 'the Effective value' are changed to read:

The Effective value of this field is 0 when any of the following are true:

- The Effective value of SCR_EL3.HXEn is 0.
- EL2 is not implemented or not enabled in the current Security state.
- The Effective value of HCR_EL2.{E2H, TGE} is {1, 1}.

In the same section, in the following field descriptions:

- 'MCE2, bit [10]'.
- 'VFNMI, bit [8]'.
- 'VINMI, bit [7]'.
- 'TALLINT, bit [6]'.
- 'FGTnXS, bit [4]'.
- 'FnXS, bit [3]'.

the following text is added:

The Effective value of this field is 0 when any of the following are true:

- EL2 is not implemented or not enabled in the current Security state.
- The Effective value of SCR_EL3.HXEn is 0.

2.17 R19616

In section **C6.2 "Alphabetical list of A64 base instructions"**, in the pseudocode of the SETG* instruction, the code that reads:

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

is changed to read:

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

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

```
CheckMemCpyParams(...)
    ...
    if MemCpyZeroSizeCheck() || memcpy.cpysize != 0 then
        ...
```

```
        MismatchedMemCpyException(memcpy, options, wrong_option);
    ...
```

is changed to read:

```
CheckMemCpyParams(...)
    ...
    if ((memcpy.stagecpysize != 0 || MemStageCpyZeroSizeCheck()) &&
          (memcpy.cpysize != 0 || MemCpyZeroSizeCheck())) then
        ...
        MismatchedMemCpyException(memcpy, options, wrong_option);
    ...
```

In the same section, in the pseudocde function CheckMemSetParams(), the code that reads:

```
CheckMemSetParams(...)
    ...
    if MemCpyZeroSizeCheck() || memcpy.cpysize != 0 then
        ...
        MismatchedMemSetException(memset, options, wrong_option);
    ...
```

is changed to read:

```
CheckMemSetParams(...)
    ...
    if ((memset.stagesetsize != 0 || MemStageSetZeroSizeCheck()) &&
          (memset.setsize != 0 || MemSetZeroSizeCheck())) then
        ...
        MismatchedMemSetException(memset, options, wrong_option);
   ...
```

2.18 C19984

In section **D13.2.2 "A reasonable degree of inaccuracy"**, the text that reads:

> - In exceptional circumstances, such as a change in Security state or other boundary condition, it is acceptable for the count to be inaccurate.

is changed to read:

> - In exceptional circumstances, such as enabling or disabling the counter, a change in Security state or other boundary condition, it is acceptable for the count to be inaccurate.

In the same section, the text that reads:

> The permitted inaccuracy limits the possible uses of the Performance Monitors. In particular, the architecture does not define the point in a pipeline where the event counter is incremented, relative to the point where a read of the event counters is made. This means that pipelining effects can cause some imprecision, and can affect which events are counted.

is changed to read:

> The permitted inaccuracy limits the possible uses of the Performance Monitors. In particular, the architecture does not define the point in a pipeline where the event counter is incremented, relative to the point where a read of the event counters is made. This means that pipelining effects can cause some imprecision, and can affect which events are counted.
>
> This applies to both architectural and microarchitectural events, and can apply differently to different events. Arm recommends that is applies consistently between counters. For example, any boundary condition effects that apply to an event counter counting the CPU_CYCLES event should apply in the same way to the cycle counter PMCCNTR_EL0. However, this is not required.
>
> Inaccuracies around exceptional events, such as enabling or disabling the counter, should be such that the inaccuracy is reduced by lengthening the sampling period.

2.19 D20310

In section **D1.6.2.1 "WFI wakeup events"**, the following rule I_{ZWCCZ} is removed:

> If a WFI or WFIT instruction put a PE into low-power state, the PE remains in that low power state until it receives a WFE wake-up event.

2.20 C20348

In section **J1.3.3 "shared/functions"**, in the pseudocode function CreateAccDescDCZero(), the code that reads:

```
AccessDescriptor CreateAccDescDCZero(boolean tagaccess, boolean tagchecked)
    ...
    accdesc.tagchecked = tagchecked;
    accdesc.tagaccess = tagaccess;
    ...
```

is changed to read:

```
AccessDescriptor CreateAccDescDCZero(CacheType cachetype)
    ...
    accdesc.tagchecked       = cachetype == CacheType_Data;
    accdesc.tagaccess        = cachetype IN {CacheType_Tag, CacheType_Data_Tag};
    accdesc.devstoreunpred   = cachetype == CacheType_Tag;
    ...
```

In section **J1.1.4 "aarch64/translation"**, in the pseudocode functions AArch64.S1HasAlignmentFault() and AArch64.S2HasAlignmentFault(), the code that reads:

```
    elsif IsFeatureImplemented(FEAT_MTE) && accdesc.tagaccess && accdesc.write then
        return (memattrs.memtype == MemType_Device &&
                ConstrainUnpredictable(Unpredictable_DEVICETAGSTORE) ==
 Constraint_FAULT);
```

is changed to read:

```
     elsif (IsFeatureImplemented(FEAT_MTE) && accdesc.tagaccess && accdesc.write &&
            accdesc.devstoreunpred) then
        return (memattrs.memtype == MemType_Device &&
                ConstrainUnpredictable(Unpredictable_DEVICETAGSTORE) ==
 Constraint_FAULT);
```

In section **C6.2.398 "STZGM"**, the following pseudocode is added:

```
 ...
AccessDescriptor accdesc = CreateAccDescLDGSTG(MemOp_STORE);
accdesc.devstoreunpred = TRUE;
...
```

2.21 D20510

In section **J1.1.4 "aarch64/translation"**, in the pseudocode function AArch64.S1TTWParamsEL3(), the code that reads:

```
S1TTWParams AArch64.S1TTWParamsEL3()
    ...
    walkparams.hpd  = if IsFeatureImplemented(FEAT_HPDS)   then TCR_EL3.HPD  else
 '0';
    ...
```

is changed to read:

```
S1TTWParams AArch64.S1TTWParamsEL3()
    ...
    walkparams.hpd  = if IsFeatureImplemented(FEAT_HPDS)   then TCR_EL3.HPD else
 '0';
    if walkparams.hpd == '0' then
        if walkparams.aie == '1' then walkparams.hpd = '1';
        if walkparams.pie == '1' then walkparams.hpd = '1';
        if AArch64.S1POEnabled(Regime_EL3) then walkparams.hpd = '1';
    ...
```

The equivalent changes are made in the pseudocode functions AArch64.S1TTWParamsEL2(), AArch64.S1TTWParamsEL20(), and AArch64.S1TTWParamsEL10().

2.22 D20634

In section **D23.4.20 "TRCCNTCTLR<n>, Trace Counter Control Register <n>, n = 0 - 3"**, in the field description of 'CNTEVENT_SEL', the following text is added:

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

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

- Section **D23.4.26 "TRCEVENTCTL0R, Trace Event Control 0 Register"**, fields 'EVENT0_SEL', 'EVENT1_SEL', 'EVENT2_SEL', and 'EVENT3_SEL'.
- Section **D23.4.53 "TRCSEQEVR<n>, Trace Sequencer State Transition Control Register <n>, n = 0 - 2"**, fields 'F_SEL' and 'B_SEL'.
- Section **D23.4.54 "TRCSEQRSTEVR, Trace Sequencer Reset Control Register"**, 'RST_SEL' field.
- Section **D23.4.63 "TRCTSCTLR, Trace Timestamp Control Register"**, 'EVENT_SEL' field.
- Section **D23.4.64 "TRCVICTLR, Trace ViewInst Main Control Register"**, 'EVENT_SEL' field.

2.23 D20786

In section **D1.5 "Resets and power domains"**, the rules that read:

> R_{JMTZY}
>
> The PE logic is split into two logical power domains:
>
> - The Core power domain.
> - The Debug power domain.
>
> R_{PDMKB}
>
> It is **IMPLEMENTATION DEFINED** whether the Core power domain and Debug power domain power up and power down separately or together.

are changed to read:

> R_{JMTZY}
>
> The architecture defines the following power domains:
>
> - The Core power domain.
> - The Debug power domain.
> - The system counter power domain.
> - If FEAT_AMUv1 is implemented, the AMU power domain.
> - For MPAM Memory System Components (MSC), a power domain for each MSC.
> - If trace is implemented, a power domain for the Trace Unit.
> - If FEAT_RAS is implemented, there may be one or more IMPLEMENTATION DEFINED error recovery power domains. The relationship between these power domains and other power domains is **IMPLEMENTATION DEFINED**. For error records associated with the PE, this is the same as the Core power domain.
>
> R_{PDMKB}

It is **IMPLEMENTATION DEFINED** whether each power domain powers up and powers down separately or together.

Correspondingly, the information statement that reads:

I_{HDQLB}

The architecture defines the following resets:

- Warm reset.
- Cold reset.
- External debug reset.
- Trace unit reset. See Resetting the trace unit.

is changed to read:

I_{HDQLB}

The architecture defines the following resets:

- Warm reset.
- Cold reset.
- External debug reset.
- Trace unit reset. See Resetting the trace unit.
- If the FEAT_AMUv1 is implemented, AMU reset.
- Timer reset.
- For MPAM Memory System Components (MSC), a reset for each MSC.
- If FEAT_RAS is implemented, there may be one or more IMPLEMENTATION DEFINED error recovery resets. The relationship between this reset and other resets is **IMPLEMENTATION DEFINED**. For error records associated with the PE, the error recovery reset is the same as Warm reset.

In section **D1.5.2 "Reset types"**, the rule that reads:

R_{RPXXYQ}

Mechanisms to assert resets, other than RMR_ELx, are IMPLEMENTATION DEFINED. Any reset-asserting mechanism that software can command, including hardware mechanisms directly exposed to software, must only be accessible at the highest Exception level.

...

R_{PJXVL}

When a Warm reset is asserted, the logic on which the PE executes is reset. The Warm reset does not reset the integrated debug functionality.

> R_{NBKHK}
>
> When an External Debug reset is asserted, the Debug power domain is reset. The Cold reset domain is not reset.

is changed to read:

> R_{RPXXYQ}
>
> Mechanisms to assert resets, other than RMR_ELx, are IMPLEMENTATION DEFINED. One such mechanism is EDPRCR.CWRR. Any reset-asserting mechanism that software can command, including hardware mechanisms directly exposed to software, must only be accessible at the highest Exception level.
>
> ...
>
> R_{PJXVL}
>
> When a Warm reset is asserted, registers and logic in the Core power domain which are affected by the Warm reset are reset to their architecturally-defined reset value. Registers and logic which are only reset by a Cold reset are unaffected by a Warm reset.
>
> R_{NBKHK}
>
> When an External Debug reset is asserted, the External Debug reset domain is reset and affected registers are set to their architecturally-defined External debug reset values. The Cold reset domain is unaffected.

The following rule and info are added to indicate the dependency between Cold and Warm resets:

> R_{X0001}
>
> When a Cold reset is asserted, a Warm reset is asserted.
>
> I_{X0002}
>
> Every register field that has storage belongs to exactly one reset domain. Each system register field definition defines the reset domain and the reset value.

2.24 R20817

In section **B2.13.2 “Alignment of data accesses”**, under the following headings:

- **B2.13.2.1.1 “Load or Store of Single or Multiple registers”**.
- **B2.13.2.1.2 “Load-Exclusive/ Store-Exclusive and Atomic instructions”**.
- **B2.13.2.1.3 “Non-atomic Load-Acquire/Store-Release instructions”**.

instances of 'Normal Inner Write-Back, Outer Write-Back Cacheable memory' are changed to read:

> Normal Inner Write-Back, Outer Write-Back Cacheable, Shareable memory

2.25 D21080

In section **D20.19 "MPAM System registers"**, in Table D20-9, the rows that read:

System register	Controlled from	Supplies PARTID and PMG when Executing in	Notes
MPAM0_EL1	EL3 EL2 EL1	EL0 (Applications)	... MPAM0_EL1 can be controlled from only EL3 if MPAM3_EL3.TRAPLOWER == 1, from only EL2 or EL3 if MPAM3_EL3.TRAPLOWER == 0 and MPAMHCR_EL2.TRAPMPAM0EL1 == 1 or from EL1, EL2 or EL3 if MPAM3_EL3.TRAPLOWER == 0 and MPAMHCR_EL2.TRAPMPAM0EL1 == 0.
MPAM1_EL1	EL3 EL2 EL1	EL1 (Guest OS)	... MPAM1_EL1 can be controlled only from EL3 if MPAM3_EL3.TRAPLOWER == 1, only from EL2 or EL3 if MPAM3_EL3.TRAPLOWER == 0 and MPAMHCR_EL2.TRAPMPAM1EL1 == 1, or from EL1, EL2 or EL3 if MPAM3_EL3.TRAPLOWER == 0 and MPAMHCR_EL2.TRAPMPAM1_EL1 == 0. ...

are changed to read:

System register	Controlled from	Supplies PARTID and PMG when Executing in	Notes
MPAM0_EL1	EL3 EL2 EL1	EL0 (Applications)	... MPAM0_EL1 can be controlled from only EL3 if MPAM3_EL3.TRAPLOWER == 1, from only EL2 or EL3 if MPAM3_EL3.TRAPLOWER == 0 and MPAM2_EL2.TRAPMPAM0EL1 == 1 or from EL1, EL2 or EL3 if MPAM3_EL3.TRAPLOWER == 0 and MPAM2_EL2.TRAPMPAM0EL1 == 0.
MPAM1_EL1	EL3 EL2 EL1	EL1 (Guest OS)	... MPAM1_EL1 can be controlled only from EL3 if MPAM3_EL3.TRAPLOWER == 1, only from EL2 or EL3 if MPAM3_EL3.TRAPLOWER == 0 and MPAM2_EL2.TRAPMPAM1EL1 == 1, or from EL1, EL2 or EL3 if MPAM3_EL3.TRAPLOWER == 0 and MPAM2_EL2.TRAPMPAM1EL1 == 0. ...

2.26 C21090

In section **D23.3.2 "DBGBCR<n>_EL1, Debug Breakpoint Control Registers, n = 0 - 63"**, in the field description of 'MASK, bits [28:24]', the text that reads:

> Indicates the number of masked address bits, from `0b00011` masking 3 address bits (`0x00000007` mask for address) to `0b11111` masking 31 address bits (`0x7FFFFFFF` mask for address).
>
> If DBGBCR<n>_EL1.MASK is programmed with a reserved value, then the breakpoint behaves as if either:
>
> - DBGBCR<n>_EL1.MASK has been programmed with a defined value, which might be `0b00000` (no mask), other than for a direct read of DBGBCR<n>_EL1.
> - The breakpoint is disabled.

is changed to read:

> Indicates the number of masked address bits, from `0b00011` masking 3 address bits (`0x00000007` mask for address) to `0b11111` masking 31 address bits (`0x7FFFFFFF` mask for address).
>
> The behavior of the breakpoint is constrained unpredictable if any of the following are true:
>
> - DBGBCR<n>_EL1.MASK is a reserved value.
> - DBGBCR<n>_EL1.MASK is zero, DBGBCR<n>_EL1.{BT2, BT} is {`0b0`, 0b010x} (address mismatch breakpoint without linking enabled), and AArch32 is not supported at EL1.
> - DBGBCR<n>_EL1.MASK is a valid nonzero value and any of the following apply:
>   - DBGBCR<n>_EL1.BAS is not `0b1111`, and AArch32 is supported at EL0
>   - DBGBCR<n>_EL1[(MASK-1):0] is nonzero.
>   - DBGBCR<n>_EL1.{BT2, BT} is {`0b0`, 0b0x1x} or {`0b0`, 0b1xxx} (Context matching breakpoint).
>
> The constrained unpredictable breakpoint behavior is one of the following:
>
> - DBGBCR<n>_EL1.MASK has been programmed with a defined value, which might be `0b00000` (no mask), other than for a direct read of DBGBCR<n>_EL1.
> - The breakpoint is disabled.

The equivalent changes are made in section **H9.2.2 "DBGBCR<n>_EL1, Debug Breakpoint Control Registers, n = 0 - 63"**.

2.27 C21187

In section **C5.2.8 "FPCR, Floating-point Control Register"**, in the field description of 'DN, bit [25]', the text that reads:

> This bit has no effect on the output of FABS, FMAX*, FMIN*, and FNEG instructions, and a default NaN is never returned as a result of these instructions.

is changed to read:

> This bit has no effect on the output of the FABS and FNEG instructions. This bit has no effect on the output of the FMAX, FMAXP, FMAXV, FMIN, FMINP, and FMINV instructions when FPCR.AH is 1.

2.28 D21219

In section **J1.1.3 "aarch64/functions"** the pseudocode function AArch64.UnalignedAccessFaults() is modified and a new function MemSingleGranule() is introduced.

The code for AArch64.UnalignedAccessFaults() that reads:

```
boolean AArch64.UnalignedAccessFaults(AccessDescriptor accdesc, bits(64) address,
 integer size)
    if AlignmentEnforced() then
    ...
    elsif accdesc.exclusive || accdesc.atomicop then
        return !HaveLSE2Ext() || !AllInAlignedQuantity(address, size, 16);
    elsif accdesc.acqsc || accdesc.acqpc || accdesc.relsc then
        return !HaveLSE2Ext() || (SCTLR[].nAA == '0' && !
AllInAlignedQuantity(address, size, 16));
    ...
```

is changed to read:

```
boolean AArch64.UnalignedAccessFaults(AccessDescriptor accdesc, bits(64) address,
 integer size)
    if AlignmentEnforced() then
    ...
    elsif accdesc.exclusive || accdesc.atomicop then
        return !HaveLSE2Ext() || !AllInAlignedQuantity(address, size, 16);
    elsif accdesc.acqsc || accdesc.acqpc || accdesc.relsc then
        return !HaveLSE2Ext() || (SCTLR[].nAA == '0' && !
AllInAlignedQuantity(address, size, 16));
    ...
```

In all invocations of the function AllInAlignedQuantity(), the hard-coded 16-byte alignment is replaced with the return value of the new function MemSingleGranule().

The code that reads:

```
AllInAlignedQuantity(address, size, 16);
```

is changed to read:

```
integer quantity = MemSingleGranule();
AllInAlignedQuantity(address, size, quantity);
```

where, the new function MemSingleGranule() is defined as:

```
integer MemSingleGranule()
    quantity = integer IMPLEMENTATION_DEFINED "Aligned quantity for atomic access";
    // access is assumed to be within 4096 byte aligned quantity to
    // avoid multiple translations for a single copy atomic access.
    assert 16 <= quantity && quantity <= 4096;
    return quantity;
```

2.29 D21248

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

```
(boolean, bits(2), bits(64), bits(64)) TLBIPRange(Regime regime, bits(128) Xt)
    ...
    constant integer tg_bits = TGBits(tg);
    start_address<55:tg_bits> = Xt<107:64+(tg_bits-12)>;

    if HasUnprivileged(regime) then
        start_address<63:56> = Replicate(Xt<107>, 8);
    else
        start_address<63:56> = Zeros(8);
    ...
```

is changed to read:

```
(boolean, bits(2), bits(64), bits(64)) TLBIPRange(Regime regime, bits(128) Xt)
    ...
    constant integer tg_bits = TGBits(tg);
    // The more-significant bits of the start_address are not updated,
    // as they are not used when performing address matching in TLB
    start_address<55:tg_bits> = Xt<107:64+(tg_bits-12)>;

    ...
```

2.30 D21251

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

```
CheckSMEEnabled()
    boolean disabled;
    // Check if access disabled in CPACR_EL1
    if PSTATE.EL IN {EL0, EL1} && !IsInHost() then
    ...
```

```
    // Check if access disabled in CPTR_EL3
    if HaveEL(EL3) then
        if CPTR_EL3.ESM == '0' then SMEAccessTrap(SMEExceptionType_AccessTrap, EL3);
        if CPTR_EL3.TFP == '1' then AArch64.AdvSIMDFPAccessTrap(EL3);
```

is changed to read:

```
CheckSMEEnabled()
    boolean disabled;

    if HaveEL(EL3) && CPTR_EL3.<ESM,TFP> != '10' && EL3SDDUndefPriority() then
        UNDEFINED;

    // Check if access disabled in CPACR_EL1
    if PSTATE.EL IN {EL0, EL1} && !IsInHost() then
    ...
    // Check if access disabled in CPTR_EL3
    if HaveEL(EL3) then
        if CPTR_EL3.ESM == '0' then
            if EL3SDDUndef() then UNDEFINED;
            SMEAccessTrap(SMEExceptionType_AccessTrap, EL3);

        if CPTR_EL3.TFP == '1' then
            if EL3SDDUndef() then UNDEFINED;
            AArch64.AdvSIMDFPAccessTrap(EL3);
```

Equivalent changes are made to the functions CheckSVEEnabled() and CheckSMEAccess().

2.31 D21341

In section **C6.2.177 “LDIAPP”**, the code segment that reads:

```
constant boolean postindex = opc2<0> == '0';
...
```

is changed to read:

```
constant boolean ispair = TRUE;
constant boolean postindex = opc2<0> == '0';
...
```

In the same section, the code segment that reads:

```
...
AccessDescriptor accdesc = CreateAccDescLDAcqPC(tagchecked);
...
if IsFeatureImplemented(FEAT_LSE2) then
    bits(2*datasize) full_data;
    accdesc.ispair = TRUE;
    full_data = Mem[address, 2*dbytes, accdesc];
    if BigEndian(accdesc.acctype) then
        data2 = full_data<(datasize-1):0>;
        data1 = full_data<(2*datasize-1):datasize>;
    else
        data1 = full_data<(datasize-1):0>;
        data2 = full_data<(2*datasize-1):datasize>;
else
```

```
    address2 = AddressIncrement(address, dbytes, accdesc);
    data1 = Mem[address, dbytes, accdesc];
    data2 = Mem[address2, dbytes, accdesc];
...
```

is changed to read:

```
...
AccessDescriptor accdesc = CreateAccDescLDAcqPC(tagchecked, ispair);
...
bits(2*datasize) full_data;
full_data = Mem[address, 2*dbytes, accdesc];
if BigEndian(accdesc.acctype) then
    data2 = full_data<(datasize-1):0>;
    data1 = full_data<(2*datasize-1):datasize>;
else
    data1 = full_data<(datasize-1):0>;
    data2 = full_data<(2*datasize-1):datasize>;
end
...
```

The equivalent changes are made in section **C6.2.349 “STILP”**.

2.32 D21363

In section **D23.2.39 “DCZID_EL0, Data Cache Zero ID register”**, in the field description of ‘DZP, bit [4]’, the text that reads:

> The value read from this field is governed by the access state and the values of the HCR_EL2.TDZ and SCTLR_EL1.DZE bits.

is changed to read:

> The value read from this field is governed by the current Exception level and the values of the following fields:
>
> - The Effective value of HCR_EL2.TDZ.
> - When the Effective value of HCR_EL2.{E2H, TGE} != ‘11’, SCTLR_EL1.DZE.
> - When the Effective value of HCR_EL2.{E2H, TGE} == ‘11’, SCTLR_EL2.DZE.

2.33 D21503

In section **J1.1.4 “aarch64/translation”**, in the pseudocode function AArch64.S2DirectBasePermissions(), the code that reads:

```
S2AccessControls AArch64.S2DirectBasePermissions(Permissions permissions,
                                                 AccessDescriptor accdesc)
    ....
    s2perms.r_mmu = r;
    s2perms.w_mmu = w;
```

```
    return s2perms;
```

is changed to read:

```
S2AccessControls AArch64.S2DirectBasePermissions(Permissions permissions,
                                                 AccessDescriptor accdesc)
    ....
    s2perms.r_mmu = r;
    s2perms.w_mmu = w;
    s2perms.toplevel0 = '0';
    s2perms.toplevel1 = '0';
    s2perms.overlay = FALSE;

    return s2perms;
```

2.34 C21507

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

> Purpose
>
> Holds the 64-bit physical offset. This is the offset for the AArch64 physical timers and counters when Enhanced Counter Virtualization is enabled.
>
> Configurations
>
> This register is present only when FEAT_ECV is implemented. Otherwise, direct accesses to CNTPOFF_EL2 are **UNDEFINED**.
>
> The CNTPOFF_EL2 offset applies to:
>
> - Direct reads of the physical counter from EL0 or EL1.
> - Indirect reads of the physical counter by the EL1 physical timer.
>
> When EL2 is implemented and enabled in the current Security state, the physical counter uses a fixed physical offset of zero if any of the following are true:
>
> - CNTHCTL_EL2.ECV is 0.
> - SCR_EL3.ECVEn is 0.
> - HCR_EL2.{E2H, TGE} is {1, 1}.

is changed to read:

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

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

The physical offset applies to:

- Direct reads of CNTPCT_EL0 from EL0 or EL1. See CNTPCT_EL0.
- Indirect reads of the physical counter by the EL1 physical timer. See The Generic Timer in AArch64 state.
- Indirect reads of the physical counter for timestamps generating by profiling logic. See The Statistical Profiling Extension and AArch64 Self-hosted trace.

The physical offset only applies under conditions described by the relevant sections.

In section **D23.10.30 "CNTVOFF_EL2, Counter-timer Virtual Offset register"**, the text that reads:

Purpose

Holds the 64-bit virtual offset. This is the offset between the physical count value visible in CNTPCT_EL0 and the virtual count value visible in CNTVCT_EL0.

Configurations

AArch64 System register CNTVOFF_EL2 bits [63:0] are architecturally mapped to AArch32 System register CNTVOFF[63:0].

If EL2 is not implemented, this register is **RES0** from EL3 and the virtual counter uses a fixed virtual offset of zero.

Note: When EL2 is implemented and enabled in the current Security state, and is using AArch64, the virtual counter uses a fixed virtual offset of zero in the following situations:

- HCR_EL2.E2H is 1, and CNTVCT_EL0 is read from EL2.
- HCR_EL2.{E2H, TGE} is {1, 1}, and either:
  - CNTVCT_EL0 is read from EL0 or EL2.
  - CNTVCT is read from EL0.

is changed to read:

Purpose

Holds the 64-bit virtual offset. This is the offset for the AArch64 virtual timers and counters.

Configurations

AArch64 System register CNTVOFF_EL2 bits [63:0] are architecturally mapped to AArch32 System register CNTVOFF[63:0].

The virtual offset applies to:

- Direct reads of CNTVCT_EL0. See CNTVCT_EL0.

- Indirect reads of the virtual counter by the EL1 virtual timer. See The Generic Timer in AArch64 state.
- Indirect reads of the virtual counter for timestamps generating by profiling logic. See “The Statistical Profiling Extension and AArch64 Self-hosted trace”.

The virtual offset only applies under conditions described by the relevant sections.

In section **D23.10.20 “CNTPCT_EL0, Counter-timer Physical Count register”**, the text that reads:

Purpose

Holds the 64-bit physical count value.

Field descriptions

Bits [63:0] Physical count value.

Reads of CNTPCT_EL0 from EL0 or EL1 return (PhysicalCountInt<63:0> - CNTPOFF_EL2<63:0>) if the access is not trapped, and all of the following are true:

- CNTHCTL_EL2.ECV is 1.
- HCR_EL2.{E2H, TGE} is not {1, 1}.

Where PhysicalCountInt<63:0> is the physical count returned when CNTPCT_EL0 is read from EL2 or EL3.

is changed to read:

Purpose

Reads of CNTPCT_EL0 return the 64-bit physical count value minus a physical offset.

Field descriptions

Bits [63:0] Physical count value.

If the access is not trapped, and all of the following are true, then reads of CNTPCT_EL0 from EL0 or EL1 return (PhysicalCountInt<63:0> - CNTPOFF_EL2<63:0>) :

- EL2 is implemented and enabled in the current Security state.
- CNTHCTL_EL2.ECV is 1.
- SCR_EL3.ECVEn is 1.
- HCR_EL2.{E2H, TGE} is not {1, 1}.

Otherwise, reads of CNTPCT_EL0 return PhysicalCountInt<63:0>.

PhysicalCountInt is defined by The physical counter on page D11-XXXX.

In section **D23.10.29 “CNTVCT_EL0, Counter-timer Virtual Count register”**, the text that reads:

> Purpose
>
> Holds the 64-bit virtual count value. The virtual count value is equal to the physical count value minus the virtual offset visible in CNTVOFF_EL2.
>
> Configurations
>
> AArch64 System register CNTVCT_EL0 bits [63:0] are architecturally mapped to AArch32 System register CNTVCT[63:0].
>
> The value of this register is the same as the value of CNTPCT_EL0 in the following conditions:
>
> - When EL2 is not implemented.
> - When EL2 is implemented, HCR_EL2.E2H is 1, and this register is read from EL2.
> - When EL2 is implemented and enabled in the current Security state, HCR_EL2.{E2H, TGE} is {1, 1}, and this register is read from EL0 or EL2.
>
> All reads to the CNTVCT_EL0 occur in program order relative to reads to CNTVCTSS_EL0 or CNTVCT_EL0.
>
> Field descriptions
>
> Bits [63:0] Virtual count value.

is changed to read:

> Purpose
>
> Reads of CNTVCT_EL0 return the 64-bit physical count value minus a virtual offset.
>
> Configurations
>
> AArch64 System register CNTVCT_EL0 bits [63:0] are architecturally mapped to AArch32 System register CNTVCT[63:0].
>
> All reads to the CNTVCT_EL0 occur in program order relative to reads to CNTVCTSS_EL0 or CNTVCT_EL0.
>
> Field descriptions
>
> Bits [63:0] Virtual count value.
>
> When EL2 is implemented, if the access is not trapped, and any of the following are true, then reads of CNTVCT_EL0 from EL0 and EL1 return (PhysicalCountInt<63:0> - CNTVOFF_EL2<63:0>) :
>
> - All of the following are true:
> - The Effective value of HCR_EL2.{E2H, TGE} is not {1, 1}.
> - CNTVCT_EL0 is read from EL0.

- All of the following are true:
  - The Effective value of HCR_EL2.E2H is 0.
  - CNTVCT_EL0 is read from EL2.
- CNTVCT_EL0 is read from EL1 or EL3.

Otherwise, reads of CNTVCT_EL0 return PhysicalCountInt<63:0>.

PhysicalCountInt is defined by "The physical counter".

Similar changes are also made to section **D23.10.28 "CNTVCTSS_EL0, Counter-timer Self-Synchronized Virtual Count register"**.

In section **D23.10.17 "CNTP_CVAL_EL0, Counter-timer Physical Timer CompareValue register"**, the text that reads:

CompareValue, bits [63:0]

When CNTP_CTL_EL0.ENABLE is 1, the timer condition is met when (CNTPCT_EL0 - CompareValue) is greater than or equal to zero. This means that CompareValue acts like a 64-bit upcounter timer. When the timer condition is met:

- CNTP_CTL_EL0.ISTATUS is set to 1.
- If CNTP_CTL_EL0.IMASK is 0, an interrupt is generated.

When CNTP_CTL_EL0.ENABLE is 0, the timer condition is not met, but CNTPCT_EL0 continues to count.

is changed to read:

CompareValue, bits [63:0]

When CNTP_CTL_EL0.ENABLE is 1, and TimerConditionMet is TRUE for the EL1 physical timer, the timer condition is met and all of the following are true:

- CNTP_CTL_EL0.ISTATUS is set to 1.
- If CNTP_CTL_EL0.IMASK is 0, an interrupt is generated.

TimerConditionMet is defined by Operation of the CompareValue views of the timers. The CompareValue view of the timer acts like a 64-bit upcounter timer. When CNTP_CTL_EL0.ENABLE is 0, the timer condition is not met, but CNTPCT_EL0 continues to count.

Similar changes are also made to the other CNTx_CVAL_ELx registers.

In section **J1.3.3 "shared/functions"**, the pseudocode function AArch64.CheckTimerConditions() that reads:

```
AArch64.CheckTimerConditions()
    ...
```

```
    SecurityState ss = CurrentSecurityState();
    boolean ecv = FALSE;
    if HaveECVExt() then
        ecv = CNTHCTL_EL2.ECV == '1' && SCR_EL3.ECVEn == '1' && EL2Enabled();
    if ecv then
        offset = CNTPOFF_EL2;
    else
        offset = Zeros(64);
    ...
```

is changed to read:

```
AArch64.CheckTimerConditions()
   ...
   SecurityState ss = CurrentSecurityState();
   if (IsFeatureImplemented(FEAT_ECV) && EL2Enabled() && !ELIsInHost(EL0) &&
          CNTHCTL_EL2.ECV == '1' && SCR_EL3.ECVEn == '1') then
       offset = CNTPOFF_EL2;
    else
       offset = Zeros(64);
    ...
```

Similar changes are made to the TestEventCNTP() function.

2.35 D21549

In section section **D23.2.65 “HPFAR_EL2, Hypervisor IPA Fault Address Register”**, in the field description of ‘FIPA, bits [47:4]’, the text that reads:

> When FEAT_MOPS is implemented, the value presented in FIPA on a synchronous exception that set the HPFAR_EL2 from any of the Memory Copy and Memory Set instructions is within the address range of the current stage 2 translation granule, aligned to the size of the current stage 2 translation granule, of the address that generated the Data abort.
>
> Bits[(n-1):0] of the value are unknown, where 2n is the current stage 2 translation granule size in bytes.

is changed to read:

> When FEAT_MOPS is implemented, the value presented in FIPA on a synchronous exception that set the HPFAR_EL2 from any of the Memory Copy and Memory Set instructions is within the address range of the current stage 2 translation granule, aligned to the size of the current stage 2 translation granule, of the address that generated the Data abort. In this case, bits[(n-1):0] of the value are **UNKNOWN**, where 2n is the current stage 2 translation granule size in bytes.

A similar change is made to all instances of the above text.

2.36 D21556

In section **D23.2.73 "ID_AA64ISAR2_EL1, AArch64 Instruction Set Attribute Register 2"**, in the field description of 'RPRES, bits [7:4]', the text that reads:

> Indicates support for 12 bits of mantissa in reciprocal and reciprocal square root instructions in AArch64 state, when FPCR.AH is 1.

is changed to read:

> Indicates support for 12 bits of mantissa in floating-point reciprocal estimate and reciprocal square root estimate instructions in AArch64 state, when FPCR.AH is 1.

The value table that reads:

0b0000	When FPCR.AH == 1: Reciprocal and reciprocal square root estimates give 8 bits of mantissa, when FPCR.AH is 1.
0b0001	When FPCR.AH == 1: Reciprocal and reciprocal square root estimates give 12 bits of mantissa, when FPCR.AH is 1.

is changed to read:

0b0000	Reciprocal and reciprocal square root estimates give 8 bits of mantissa when FPCR.AH is 1.
0b0001	Reciprocal and reciprocal square root estimates give 12 bits of mantissa when FPCR.AH is 1.

The following text is added after the value table:

> When FPCR.AH is 0, the floating-point reciprocal estimate and reciprocal square root estimate instructions give 8 bits of mantissa, regardless of this value.

2.37 C21650

In section **D8.13 "Nested virtualization"**, the rule that reads:

> R_{RHQHT}
>
> If a Guest hypervisor is run with HCR_EL2.E2H set to 0, then the Host hypervisor is required to set HCR_EL2.TVM to 1 and CPTR_EL2.TCPAC to 1 to trap any Guest hypervisor accesses to the EL1 System registers that would be accessed from any Guest OS running under the Guest hypervisor.

is changed to read:

> I_{RHQHT}
>
> If a Guest hypervisor is run with the illusion that HCR_EL2.E2H is set to 0, and the Host hypervisor has therefore set HCR_EL2.NV1 to 1, then the Host hypervisor should additionally set HCR_EL2.TVM to 1 and CPTR_EL2.TCPAC to 1 to trap any Guest hypervisor accesses to the EL1 System registers that would be accessed from any Guest OS running under the Guest hypervisor.

2.38 C21661

In section **C6.2.148 "LD64B"** the following text is added:

> It is **IMPLEMENTATION DEFINED** which memory locations support this instruction. A memory location that supports LD64B also supports ST64B. For more information, including on the memory types accessible and how the accesses are performed, see *Single-copy atomic 64-byte load/store*.

The equivalent change is made in section **C6.2.334 "ST64B"**.

In section **K1.2.17 CONSTRAINED UNPREDICTABLE behavior for A64 instructions**, new sections are added for LD64B and ST64B with the following text:

> If a memory location does not support this instruction, then one of the following behaviors must occur:
>
> - A stage 1 Data Abort, reported using the DFSC code of 110101, is generated.
> - The instruction performs the memory accesses, but the accesses are not single-copy atomic above the byte level.

A cross reference from section **C6.2.148 "LD64B"** and section **C6.2.334 "ST64B"** to the new section in **K1.2.17 CONSTRAINED UNPREDICTABLE behavior for A64 instructions** is also added.

2.39 C21662

In section **C6.2.335 "ST64BV"** the text that reads:

> Single-copy Atomic 64-byte Store with status result stores eight 64-bit doublewords from consecutive registers, Xt to X(t+7), to a memory location, and writes the status result of the store to a register. The data that is stored is atomic and is required to be 64-byte aligned.

is changed to read:

> Single-copy Atomic 64-byte Store with status result stores eight 64-bit doublewords from consecutive registers to a memory location, and writes the status result of the store to a register. The store starts at register Xt, with the data being formed as: Data<511:0> = X(t+7):X(t+6):X(t+5):X(t+4):X(t+3):X(t+2):X(t+1):Xt. The data that is stored is atomic and is required to be 64-byte aligned.
>
> It is **IMPLEMENTATION DEFINED** which memory locations support this instruction.
>
> A memory location that supports ST64BV also supports ST64BV0.
>
> For more information, including on the memory types accessible and how the accesses are performed, see C3.2.12.2 "Single-copy atomic 64-byte load/store".

The equivalent changes are made to section **C6.2.336 “ST64BV0”**.

2.40 D21665

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

```
(boolean, bits(2), bits(64), bits(64)) TLBIRange(Regime regime, bits(64) Xt)
    ...
    if end_address<52> != start_address<52> then
        // overflow, saturate it
        end_address = Replicate(start_address<52>, 64-52) : Ones(52);
    ...
```

is changed to read:

```
(boolean, bits(2), bits(64), bits(64)) TLBIRange(Regime regime, bits(64) Xt)
    ...
    if IsFeatureImplemented(FEAT_LVA3) && end_address<56> != start_address<56> then
        // overflow, saturate it
        end_address = Replicate(start_address<56>, 64-56) : Ones(56);
    elsif end_address<52> != start_address<52> then
        // overflow, saturate it
        end_address = Replicate(start_address<52>, 64-52) : Ones(52)
    ...
```

2.41 D21666

In section **J1.1.1 “aarch64/debug”**, in the CheckWatchpoint() pseudocode function, the code segment that reads:

```
(WatchpointType, boolean) AArch64.WatchpointMatch(integer n, bits(64) vaddress,
 integer size,

    AccessDescriptor accdesc)

...
    if (fault.statuscode == Fault_Debug && HaltOnBreakpointOrWatchpoint() &&
            !accdesc.nonfault && !(accdesc.firstfault && !accdesc.first)) then
...
```

is changed to read:

```
WatchpointInfo AArch64.WatchpointMatch(integer n, bits(64) vaddress, integer size,
 AccessDescriptor accdesc)
...
    fault.vaddress = vaddress;
    fault.watchptinfo.maybe_false_match = rounded_match;
    if (fault.statuscode == Fault_Debug && HaltOnBreakpointOrWatchpoint() &&
            !accdesc.nonfault && !(accdesc.firstfault && !accdesc.first)) then
...
```

The equivalent changes are made to the WatchpointMatch() pseudocode function.

Similarly, in section **J1.3.3 "shared/functions"**, in the FaultRecord() pseudocode function, the code segment that reads:

```
type FaultRecord is (
    Fault            statuscode,        // Fault Status
    AccessDescriptor accessdesc,        // Details of the faulting access
    bits(64)         vaddress,          // Faulting virtual address
```

is changed to read:

```
type FaultRecord is (
    Fault            statuscode,        // Fault Status
    AccessDescriptor accessdesc,        // Details of the faulting access
    bits(64)         vaddress,          // Faulting virtual address
```

2.42 D21712

In section **D23.2.175 "TCR_EL2, Translation Control Register (EL2)"**, in the subsection 'When the Effective value of HCR_EL2.E2H is not 1:' in the field description of 'PS, bits [18:16]', the following text is removed:

0b111	When FEAT_D128 is implemented: 56 bits, 64PB.

In the same section, in the subsection 'When the Effective value of HCR_EL2.E2H is 1:' in the field description of 'IPS, bits [34:32]', the text that reads:

0b110	When FEAT_LPA is implemented: 52 bits, 4PB.

is changed to read:

0b110	When FEAT_LPA is implemented: 52 bits, 4PB.
0b111	When FEAT_D128 is implemented: 56 bits, 64PB.

In the same section, in the subsection 'When the Effective value of HCR_EL2.E2H is 1:' in the field description 'IPS, bits [34:32]', the text that reads:

> If the value of ID_AA64MMFR0_EL1.PARange is 0b0110, and the value of this field is not 0b110 or a value treated as 0b110, then bits[51:48] of every translation table base address for the stage of translation controlled by TCR_EL2 are 0b0000.

is changed to read:

> If the value of ID_AA64MMFR0_EL1.PARange is 0b0110, and the value of this field is not 0b110 or a value treated as 0b110, then bits[51:48] of every translation table base address are 0b0000 for the stage of translation controlled by TCR_EL2.

> If the value of ID_AA64MMFR0_EL1.PARange is `0b0111`, and the value of this field is not `0b111` or a value treated as `0b111`, then bits[55:52] of every translation table base address are `0b0000` for the stage of translation controlled by TCR_EL2.

The equivalent changes are made in the following sections:

- **D23.2.174 TCR_EL1, Translation Control Register (EL1)**, in the field description of 'IPS, bits [34:32]'.
- **D23.2.176 TCR_EL3, Translation Control Register (EL3)**, in the field description of 'PS, bits [18:16]'.

2.43 D21734

In section **D1.4.1 "PSTATE fields that are meaningful in AArch64 state"**, for the PSTATE field 'TCO, Tag Check Override bit', the following text is removed:

> If FEAT_MTE2 is not implemented, it is **CONSTRAINED UNPREDICTABLE** whether the PSTATE.TCO bit is **RES0** or behaves as if FEAT_MTE2 is implemented.

In section **C5.2.27 "TCO, Tag Check Override"**, the text that reads:

> When FEAT_MTE2 is not implemented, it is **CONSTRAINED UNPREDICTABLE** whether this register is **RES0** or behaves as if FEAT_MTE2 is implemented.

is changed to read:

> When FEAT_MTE2 is not implemented, it is **IMPLEMENTATION DEFINED** if accesses to this register access PSTATE.TCO or are **RAZ/WI**

In section **C5.2.19, "SPSR_EL1, Saved Program Status Register (EL1)"**, the following text in the field description of 'TCO, bit[25]' is removed:

> When FEAT_MTE2 is not implemented, it is **CONSTRAINED UNPREDICTABLE** whether this field is **RES0** or behaves as if FEAT_MTE2 is implemented.

The equivalent changes are made in the following sections:

- **C5.2.20 SPSR_EL2, Saved Program Status Register (EL2)**, in the field description of 'TCO, bit[25]'.
- **C5.2.21 SPSR_EL3, Saved Program Status Register (EL3)**, in the field description of 'TCO, bit[25]'.
- **D23.3.14 DSPSR_EL0, Debug Saved Program Status Register**, in the field description of 'TCO, bit[25]'.

In section **C6.2.249, "MSR (immedate)"**, the following text is added:

> When FEAT_MTE is implemented and FEAT_MTE2 is not implemented it is **IMPLEMENTATION DEFINED** if writes to PSTATE.TCO by this instruction are ignored.

2.44 D21753

In section **J1.1.3 “aarch64/functions”**, in the pseudocode for MemSingleNF[] - non-assignment form, the code that reads:

```
(bits(8*size), boolean) MemSingleNF[bits(64) address, integer size, AccessDescriptor
 accdesc_in,
                                    boolean aligned]
    ...
    if IsFeatureImplemented(FEAT_MTE2) && accdesc.tagchecked then
        bits(4) ptag = AArch64.PhysicalTag(address);
        if !AArch64.CheckTag(memaddrdesc, accdesc, ptag) then
            return (bits(8*size) UNKNOWN, TRUE);
    ...
```

is changed to read:

```
(bits(8*size), boolean) MemSingleNF[bits(64) address, integer size, AccessDescriptor
 accdesc_in,
                                    boolean aligned]
    ...
    if IsFeatureImplemented(FEAT_MTE2) && accdesc.tagchecked then
        bits(4) ptag = AArch64.PhysicalTag(address);
        if !AArch64.CheckTag(memaddrdesc, accdesc, ptag) then
            TCFType tcf = AArch64.EffectiveTCF(accdesc.el, accdesc.read);
            case tcf of
                when TCFType_Ignore
                    // Tag Check Faults have no effect on the PE.
                when TCFType_Sync
                    return (bits(8*size) UNKNOWN, TRUE);
                when TCFType_Async
                    return (bits(8*size) UNKNOWN, TRUE);
    ...
```

2.45 D21754

In section **J1.1.3 “aarch64/functions”**, in the pseudocode function AArch64.S1AMECFault(), the code that reads:

```
boolean AArch64.S1AMECFault(...)
    ...
    bit descriptor_amec = if walkparams.d128 == '1' then descriptor<103> else
 descriptor<63>;
```

is changed to read:

```
boolean AArch64.S1AMECFault(...)
    ...
     bit descriptor_amec = if walkparams.d128 == '1' then descriptor<108> else
 descriptor<63>;
```

The equivalent changes are made in the pseudocode functions AArch64.S1OutputMECID() and AArch64.S2OutputMECID().

2.46 D21755

In section **C6.2.134 "GCSPOPX"**, the text that reads:

> GCSPOPX {<Xt>}

is changed to read:

> GCSPOPX

The equivalent changes are made to the **C6.2.132 "GCSPOPCX"** and **C6.2.136 "GCSPUSHX"** instructions.

2.47 C21765

In section **D23.10.5 "CNTHP_TVAL_EL2, Counter-timer Physical Timer TimerValue Register (EL2)"**, the text in the 'TimerValue' field description that reads:

> When CNTHP_CTL_EL2.ENABLE is 0, the timer condition is not met, but CNTPCT_EL0 continues to count, so the TimerValue view appears to continue to count down.

is changed to read:

> When CNTHP_CTL_EL2.ENABLE is 0, the TimerValue cannot be read but continues to decrement. When the timer is enabled, the TimerValue represents the elapsed time whether that time was spent enabled or disabled.

The equivalent changes are made in the following sections:

- **D23.10.8 "CNTHPS_TVAL_EL2, Counter-timer Secure Physical Timer TimerValue Register (EL2)"**.
- **D23.10.11 "CNTHV_TVAL_EL2, Counter-timer Virtual Timer TimerValue Register (EL2)"**.
- **D23.10.14 "CNTHVS_TVAL_EL2, Counter-timer Secure Virtual Timer TimerValue Register (EL2)"**.
- **D23.10.18 "CNTP_TVAL_EL0, Counter-timer Physical Timer TimerValue Register"**.
- **D23.10.24 "CNTPS_TVAL_EL1, Counter-timer Physical Secure Timer TimerValue Register"**.
- **D23.10.27 "CNTV_TVAL_EL0, Counter-timer Virtual Timer TimerValue Register"**.

The equivalent changes are also made in the following AArch32 sections:

- **G8.7.5 "CNTHP_TVAL, Counter-timer Hyp Physical Timer TimerValue register"**.
- **G8.7.8 "CNTHPS_TVAL, Counter-timer Secure Physical Timer TimerValue Register (EL2)"**.

- **G8.7.11 "CNTHV_TVAL, Counter-timer Virtual Timer TimerValue register (EL2)"**.
- **G8.7.14 "CNTHVS_TVAL, Counter-timer Secure Virtual Timer TimerValue Register (EL2)"**.
- **G8.7.18 "CNTP_TVAL, Counter-timer Physical Timer TimerValue register"**.
- **G8.7.23 "CNTV_TVAL, Counter-timer Virtual Timer TimerValue register"**.

2.48 D21767

In section **D23.2.191 "TTBR1_EL2, Translation Table Base Register 1 (EL2)"**, under the heading 'Accessing TTBR1_EL2', the text that reads:

> When FEAT_D128 is implemented MRRS <Xt+1>, <Xt>, TTBR1_EL2name.

is changed to read:

> When FEAT_D128 is implemented MRRS <Xt>, <Xt+1>, TTBR1_EL2

The text that reads:

> When FEAT_D128 is implemented MSRR TTBR1_EL2, <Xt+1>, <Xt>

is changed to read:

> When FEAT_D128 is implemented MSRR TTBR1_EL2, <Xt>, <Xt+1>

The equivalent changes are made in the following sections:

- **D23.2.128 "PAR_EL1, Physical Address Register"**, under the heading 'Accessing PAR_EL1'.
- **D23.2.140 "RCWMASK_EL1, Read Check Write Instruction Mask (EL1)"**, under the heading 'Accessing RCWMASK_EL1'.
- **D23.2.141 "RCWSMASK_EL1, Software Read Check Write Instruction Mask (EL1)"**, under the heading 'Accessing RCWSMASK_EL1'.
- **D23.2.154 "S3_<op1>_<Cn>_<Cm>_<op2>, IMPLEMENTATION DEFINED Registers"**, under the heading 'Accessing S3_<op1>_<Cn>_<Cm>_<op2>'.
- **D23.2.187 "TTBR0_EL1, Translation Table Base Register 0 (EL1)"**, under the heading 'Accessing TTBR0_EL1'.
- **D23.2.188 "TTBR0_EL2, Translation Table Base Register 0 (EL2)"**, under the heading 'Accessing TTBR0_EL2'.
- **D23.2.190 "TTBR1_EL1, Translation Table Base Register 1 (EL1)"**, under the heading 'Accessing TTBR1_EL1'.
- **D23.2.191 "TTBR1_EL2, Translation Table Base Register 1 (EL2)"**, under the heading 'Accessing TTBR1_EL2'.
- **D23.2.203 "VTTBR_EL2, Virtualization Translation Table Base Register"**, under the heading 'Accessing VTTBR_EL2'.

2.49 D21770

In section **J1.1.1 “aarch64/debug”**, the following code is added to the BRBEException() pseudocode function:

```
BRBEException()
...
        case except of
...
            when Exception_LDST64BTrap          branch_type = '100011'; // Trap
            when Exception_GPC
                if iss<20> == '1' then          branch_type = '101011'; // Inst
 fault
                else                            branch_type = '101100'; // Data
 fault
...
```

2.50 D21773

In section **J1.3.1 “shared/debug”**, in the pseudocode function UpdateDbgAuthStatus(), the code that reads:

```
UpdateDbgAuthStatus()
    ...
    if HaveEL(EL3) then
        if ExternalInvasiveDebugEnabled() then
            nsid = '11';          // Non-Secure Invasive debug implemented and
 enabled.
        else
            nsid = '10';          // Non-Secure Invasive debug implemented and
 disabled.
        if IsFeatureImplemented(FEAT_Debugv8p4) || ExternalNoninvasiveDebugEnabled()
 then
            nsnid = '11';         // Non-Secure Non-Invasive debug implemented and
 enabled.
        else
            nsnid = '10';         // Non-Secure Non-Invasive debug implemented and
 disabled.
        if ExternalSecureInvasiveDebugEnabled() then
            sid = '11';           // Secure Invasive debug implemented and enabled.
        else
            sid = '10';           // Secure Invasive debug implemented and disabled.
        if IsFeatureImplemented(FEAT_Debugv8p4) ||
 ExternalSecureNoninvasiveDebugEnabled() then
            snid = '11';            // Field has the same value as
 DBGAUTHSTATUS_EL1.SID
        else
            snid = '10';          // Secure Non-Invasive debug implemented and
 disabled.
    else
        sid   = '00';
        snid  = '00';
        nsid  = '00';
        nsnid = '00';

    DBGAUTHSTATUS_EL1.NSID  = nsid;
    DBGAUTHSTATUS_EL1.NSNID = nsnid;
    DBGAUTHSTATUS_EL1.SID   = sid;
    DBGAUTHSTATUS_EL1.SNID  = snid;
```

```
    return;
```

is changed to read:

```
UpdateDbgAuthStatus()
    ...
    bits(2) rlid, rtid;
    if SecureOnlyImplementation() then
        nsid = '00';
    elsif ExternalInvasiveDebugEnabled() then
        nsid = '11';          // Non-secure Invasive debug implemented and enabled.
    else
        nsid = '10';          // Non-secure Invasive debug implemented and disabled.

    if SecureOnlyImplementation() then
        nsnid = '00';
    elsif ExternalNoninvasiveDebugEnabled() then
        nsnid = '11';         // Non-secure Non-Invasive debug implemented and
 enabled.
    else
        nsnid = '10';         // Non-secure Non-Invasive debug implemented and
 disabled.

    if !HaveSecureState() then
        sid = '00';
    elsif ExternalSecureInvasiveDebugEnabled() then
        sid = '11';           // Secure Invasive debug implemented and enabled.
    else
        sid = '10';           // Secure Invasive debug implemented and disabled.

    if !HaveSecureState() then
        snid = '00';
    elsif ExternalSecureNoninvasiveDebugEnabled() then
        snid = '11';          // Secure Non-Invasive debug implemented and enabled.
    else
        snid = '10';          // Secure Non-Invasive debug implemented and disabled.

    if !IsFeatureImplemented(FEAT_RME) then
        rlid = '00';
    elsif ExternalRealmInvasiveDebugEnabled() then
        rlid = '11';          // Realm Invasive debug implemented and enabled.
    else
        rlid = '10';          // Realm Invasive debug implemented and disabled.

    if !IsFeatureImplemented(FEAT_RME) then
        rtid = '00';
    elsif ExternalRootInvasiveDebugEnabled() then
        rtid = '11';          // Root Invasive debug implemented and enabled.
    else
        rtid = '10';          // Root Invasive debug implemented and disabled.

    DBGAUTHSTATUS_EL1.NSID  = nsid;
    DBGAUTHSTATUS_EL1.NSNID = nsnid;
    DBGAUTHSTATUS_EL1.SID   = sid;
    DBGAUTHSTATUS_EL1.SNID  = snid;
    DBGAUTHSTATUS_EL1.RLID  = rlid;
    DBGAUTHSTATUS_EL1.RLNID = rlid;    // Field has the same value as
 DBGAUTHSTATUS_EL1.RLID.
    DBGAUTHSTATUS_EL1.RTID  = rtid;
    DBGAUTHSTATUS_EL1.RTNID = rtid;    // Field has the same value as
 DBGAUTHSTATUS_EL1.RTID.
    return;
```

In the same section, in the pseudocode function ExternalSecureInvasiveDebugEnabled(), the code that reads:

```
boolean ExternalSecureInvasiveDebugEnabled()
    if !HaveEL(EL3) && !SecureOnlyImplementation() then return FALSE;
    return ExternalInvasiveDebugEnabled() && SPIDEN == Signal_High;
```

is changed to read:

```
boolean ExternalSecureInvasiveDebugEnabled()
    if !HaveSecureState() then return FALSE;
    return ExternalInvasiveDebugEnabled() && SPIDEN == Signal_High;
```

An equivalent change is made in the pseudocode function ExternalSecureNoninvasiveDebugEnabled().

2.51 D21788

In section **D23.2.159 "SCTLR_EL1, System Control Register (EL1)"**, in field 'UCI, bit [26]', the text that reads:

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

is changed to read:

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

Equivalent changes are made in the UCI field in section **D23.2.160 "SCTLR_EL2, System Control Register (EL2)"**.

2.52 D21790

In section **D23.2.159 "SCTLR_EL1, System Control Register (EL1)"**, in the field description of 'TIDCP, bit [63]', the text that reads:

> - In AArch64 state, EL0 access to the encodings in the following reserved encoding spaces are trapped and reported using EC syndrome `0x18`:
>   - **IMPLEMENTATION DEFINED** System instructions, which are accessed using SYS and SYSL, with CRn == {11, 15}.
>   - **IMPLEMENTATION DEFINED** System registers, which are accessed using MRS and MSR with the S3_<op1>_<Cn>_<Cm>_<op2> register name.

is changed to read:

> - In AArch64 state, EL0 access to the encodings in the following reserved encoding spaces are trapped:
>   - **IMPLEMENTATION DEFINED** System instructions, which are accessed using SYS and SYSL, with CRn == {11, 15}, and are reported using EC syndrome `0x18`.
>   - **IMPLEMENTATION DEFINED** System instructions, which are accessed using SYSP, with CRn == {11, 15}, and are reported using EC syndrome `0x14`.
>   - **IMPLEMENTATION DEFINED** System registers, which are accessed using MRS and MSR with the S3_<op1>_<Cn>_<Cm>_<op2> register name, and are reported using EC syndrome `0x18`.
>   - **IMPLEMENTATION DEFINED** System registers, which are accessed using MRRS and MSRR with the S3_<op1>_<Cn>_<Cm>_<op2> register name, and are reported using EC syndrome `0x14`.

The equivalent changes are made in the following sections:

- **D23.2.160 "SCTLR_EL2, System Control Register (EL2)"**, in the field description of 'TIDCP, bit [63]'.
- **D23.2.53 "HCR_EL2, Hypervisor Configuration Register"**, in the field description of 'TIDCP, bit [20]'.

2.53 D21791

In section **J1.1.1 "aarch64/debug"**, in the pseudocode function AArch64.PMUCycle(), the code that reads:

```
if ((!IsFeatureImplemented(FEAT_SEBEP) || PMEVTYPER_EL0[idx].SYNC == '0' ||
                    !IsSupportingPMUSynchronousMode(PMEVTYPER_EL0[idx].evtCount)) &&
```

```
                    CountPMUEvents(idx)) then
```

is changed to read:

```
if ((!IsFeatureImplemented(FEAT_SEBEP) || PMEVTYPER_EL0[idx].SYNC == '0' ||
                    IsSupportingPMUSynchronousMode(PMEVTYPER_EL0[idx].evtCount)) &&
                    CountPMUEvents(idx)) then
```

2.54 D21807

In section **D23.2.75 "ID_AA64MMFR1_EL1, AArch64 Memory Model Feature Register 1"**, in the field description of 'HAFDBS, bits [3:0]', the text that reads:

> As `0b0001`, and adds support for hardware update of the Access flag for Block and Page descriptors. Hardware update of dirty state is supported.

is changed to read:

> As `0b0001`, and hardware update of dirty state is supported.

2.55 D21808

In section **D8.13.1 "Behavior when HCR_EL2.NV is 1"**, in rule I_{TZTZL}, the text that reads:

> Any System register accessed using MRS or MSR named *_EL2, except SP_EL2.

is changed to read:

> Any System register accessed using MRS or MSR named *_EL2, except SP_EL2, MECIDR_EL2, MECID_A0_EL2, MECID_A1_EL2, MECID_P0_EL2, MECID_P1_EL2, VMECID_A_EL2, and VMECID_P_EL2.

2.56 D21809

In section **D1.3.9 "Exception generating instructions"**, the rule that reads:

> R_{HMXQS}
>
> If any of the following are true, the SMC instruction is **UNDEFINED**:
>
> - The PE is executing at EL0.
> - EL3 is not implemented.

is changed to read:

R_{HMXQS}

If any of the following are true, the SMC instruction is **UNDEFINED**:

- The PE is executing at EL0.
- EL3 is not implemented and the SMC instruction is not trapped to EL2 by HCR_EL2.TSC.
- SCR_EL3.SMD is 1 and the SMC instruction is not trapped to EL2 by HCR_EL2.TSC.

2.57 D21820

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

```
if fault then
        if memcpy.stage == MOPSStage_Prologue then
            // Undo any formatting of the input parameters performed in the
 prologue.
            if memcpy.implements_option_a && memcpy.forward then
                // cpysize is negative.
                integer cpysize  = memcpy.cpysize + copied;
                X[memcpy.n, 64]  = (0 - cpysize)<63:0>;
                X[memcpy.d, 64]  = memcpy.toaddress   + cpysize;
                X[memcpy.s, 64]  = memcpy.fromaddress + cpysize;
            elsif !memcpy.implements_option_a && !memcpy.forward then
                integer cpysize  = memcpy.cpysize - copied;
                X[memcpy.n, 64]  = cpysize<63:0>;
                X[memcpy.d, 64]  = memcpy.toaddress   - cpysize;
                X[memcpy.s, 64]  = memcpy.fromaddress - cpysize;
            else
                X[memcpy.n, 64] = (memcpy.cpysize - copied)<63:0>;
                X[memcpy.d, 64] = memcpy.toaddress   + copied;
                X[memcpy.s, 64] = memcpy.fromaddress + copied;
```

is changed to read:

```
if fault then
        if memcpy.stage == MOPSStage_Prologue then
            // Undo any formatting of the input parameters performed in the
 prologue.
            if memcpy.implements_option_a then
                if memcpy.forward then
                    // cpysize is negative.
                    integer cpysize = memcpy.cpysize + copied;
                    X[memcpy.n, 64] = (0 - cpysize)<63:0>;
                    X[memcpy.d, 64] = memcpy.toaddress   + cpysize;
                    X[memcpy.s, 64] = memcpy.fromaddress + cpysize;
                else
                    X[memcpy.n, 64] = (memcpy.cpysize - copied)<63:0>;
            else
                if memcpy.forward then
                    X[memcpy.n, 64] = (memcpy.cpysize - copied)<63:0>;
                    X[memcpy.d, 64] = memcpy.toaddress   + copied;
                    X[memcpy.s, 64] = memcpy.fromaddress + copied;
                else
                    X[memcpy.n, 64] = (memcpy.cpysize - copied)<63:0>;
```

2.58 C21821

In section **D1.3.1.4 "Definition of a precise exception and imprecise exception"**, the following text is added to rule R_{TNVSL}:

- For synchronous Data Abort and Watchpoint exceptions that are generated from load or store instructions executed in AArch64 state, all the following can occur:

...

- If the instruction was an atomic that attempts both a load and a store, and the load is to the base address register, a source register or a compare register, that register is restored to the original value. Otherwise, that destination register becomes **UNKNOWN**.

2.59 E21823

In section **C6.2.43 "CAS, CASA, CASAL, CASL"**, the following text is added after the description:

For a CAS or CASA instruction, when <Ws> or <Xs> specifies the same register as <Wt> or <Xt>, this signals to the memory system that an additional subsequent CAS, CASA, CASAL, or CASL access to the specified location is likely to occur in the near future. The memory system can respond by taking actions that are expected to facilitate success of the subsequent CAS, CASA, CASAL, or CASL access when it does occur. A code sequence starting with a CAS or CASA for which <Ws> or <Xs> specifies the same register as <Wt> or <Xt>, and ending with a subsequent CAS, CASA, CASAL, or CASL to the same location, exhibits the following properties for best performance when the location may be accessed concurrently, on one or more other PEs:

- The sequence does not contain any direct system register writes, address translation instructions, cache or TLB maintenance operations, exception producing instructions, exception returns, or ISB barriers.
- The execution of the sequence includes 32 or fewer instructions.
- The value provided in <Ws> or <Xs> of the first CAS or CASA is a value likely to result in the comparison failing. A failing comparison result may lead to better performance due to the hardware not performing a write to memory.

Note:

For a CASP or CASPA instruction, when <Ws> or <Xs> specifies the same register as <Wt> or <Xt>, the value in memory is not modified, because the CASP or CASPA either fails its compare or writes the same value back to memory.

The equivalent changes are made in the following sections:

- **C6.2.44 "CASB, CASAB, CASALB, CASLB"**
- **C6.2.45 "CASH, CASAH, CASALH, CASLH"**
- **C6.2.46 "CASP, CASPA, CASPAL, CASPL"**

2.60 D21833

In section **J1.1.4 "aarch64/translation"**, in the pseudocode function AArch64.S2CheckPermissions(), the code that reads:

```
(FaultRecord, boolean) AArch64.S2CheckPermissions(...)
    ...
      if accdesc.acctype == AccessType_TTW then
        ...
        elsif walkparams.ptw == '1' && memtype == MemType_Device then
            fault.statuscode = Fault_Permission;
        elsif s2perms.overlay && or == '0' then
            fault.statuscode = Fault_Permission;
            fault.overlay    = TRUE;
        elsif accdesc.write && s2perms.overlay && ow == '0' then
            fault.statuscode = Fault_Permission;
            fault.overlay = TRUE;
    ...
```

is changed to read:

```
(FaultRecord, boolean) AArch64.S2CheckPermissions(...)
   ...
     if accdesc.acctype == AccessType_TTW then
     ...
         elsif s2perms.overlay && or == '0' then
             fault.statuscode = Fault_Permission;
             fault.overlay = TRUE;
        elsif accdesc.write && s2perms.overlay && ow == '0' then
             fault.statuscode = Fault_Permission;
             fault.overlay = TRUE;
        elsif walkparams.ptw == '1' && memtype == MemType_Device then
             fault.statuscode = Fault_Permission;
      ...
```

2.61 D21835

In section **D8.13.4 "Behavior when HCR_EL2.NV is 0 and HCR_EL2.NV1 is 1"**, the third bullet in I_{KZNPS} that reads:

- The implementation behaves as defined when HCR_EL2.NV is 0, with HCR_EL2.NV1 set to 1 having the effect of causing accesses to VBAR_EL1, ELR_EL1, and SPSR_EL1 from EL1 to be trapped to EL2.

is changed to read:

- The implementation behaves as defined when HCR_EL2.NV is 0, with HCR_EL2.NV1 set to 1 having the effect of trapping to EL2 the same set of registers that are trapped when HCR_EL2.{NV, NV1} is {1, 1}. See I_{JKLJK}.

2.62 C21837

In section **J1.1.2 "aarch64/exceptions"**, in the pseudocode function AArch64.TakeReset(), the code that reads:

```
AArch64.TakeReset(boolean cold_reset)
    ...
    // Reset all other PSTATE fields
    PSTATE.SP = '1';                    // Select stack pointer
    PSTATE.<D,A,I,F>  = '1111';         // All asynchronous exceptions masked
    PSTATE.SS = '0';                    // Clear software step bit
    PSTATE.DIT = '0';                   // PSTATE.DIT is reset to 0 when resetting
 into AArch64
    if IsFeatureImplemented(FEAT_PAuth_LR) then
        PSTATE.PACM = '0';              // PAC modifier
    ...
    AArch64.ResetGeneralRegisters();
    AArch64.ResetSIMDFPRegisters();
    ...
```

is changed to read:

```
AArch64.TakeReset(boolean cold_reset)
    ...
    // Reset all other PSTATE fields
    PSTATE.SP = '1';                    // Select stack pointer
    PSTATE.<D,A,I,F>  = '1111';         // All asynchronous exceptions masked
    PSTATE.SS = '0';                    // Clear software step bit
    PSTATE.DIT = '0';                   // PSTATE.DIT is reset to 0 when resetting
 into AArch64
    if IsFeatureImplemented(FEAT_PAuth_LR) then
        PSTATE.PACM = '0';              // PAC modifier
    if IsFeatureImplemented(FEAT_SME) then
        PSTATE.<SM,ZA> = '00';          // Disable Streaming SVE mode & ZA storage
        ResetSMEState('0');
    ...
    AArch64.ResetGeneralRegisters();
    if IsFeatureImplemented(FEAT_SME) || IsFeatureImplemented(FEAT_SVE) then
        ResetSVERegisters();
    else
        AArch64.ResetSIMDFPRegisters();
    ....
```

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

```
ResetSMEState()
    integer vectors = MAX_VL DIV 8;
    for n = 0 to vectors - 1
        _ZA[n] = Zeros(MAX_VL);
    _ZT0 = Zeros(ZT0_LEN);
```

is changed to read:

```
ResetSMEState(bit newenable)
    integer vectors = MAX_VL DIV 8;
    if newenable == '1' then
        for n = 0 to vectors - 1
            _ZA[n] = Zeros(MAX_VL);
```

```
        if IsFeatureImplemented(FEAT_SME2) then
            _ZT0 = Zeros(ZT0_LEN);
    else
        for n = 0 to vectors - 1
            _ZA[n] = bits(MAX_VL) UNKNOWN;
        if IsFeatureImplemented(FEAT_SME2) then
            _ZT0 = bits(ZT0_LEN) UNKNOWN;
```

A new function ResetSVERegisters() is added:

```
ResetSVERegisters()
    for n = 0 to 31
        _Z[n] = bits(MAX_VL) UNKNOWN;
    for n = 0 to 15
        _P[n] = bits(MAX_PL) UNKNOWN;
    _FFR = bits(MAX_PL) UNKNOWN;
```

2.63 D21842

The text in section **C6.2.328 "SETGP, SETGM, SETGE"**, changed by R19616 to read:

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

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

is changed to read:

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

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

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

The equivalent changes are made to the following sections:

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

2.64 D21845

In section **J1.1.3 "aarch64/functions"**, in the pseudocode function ComputePAC2(), the code that reads:

```
bits(64) ComputePAC2(bits(64) data, bits(64) modifier1, bits(64) modifier2,
                     bits(64) key0, bits(64) key1, boolean isgeneric)
    bits(64) concat_modifiers = modifier2<36:5>:modifier1<35:4>;
    if UsePACIMP(isgeneric) then
        return ComputePACIMPDEF(data, concat_modifiers, key0, key1);
    if UsePACQARMA3(isgeneric) then
        boolean isqarma3 = TRUE;
        return ComputePACQARMA(data, concat_modifiers, key0, key1, isqarma3);
    if UsePACQARMA5(isgeneric) then
        boolean isqarma3 = FALSE;
        return ComputePACQARMA(data, concat_modifiers, key0, key1, isqarma3);
```

is changed to read:

```
bits(64) ComputePAC2(bits(64) data, bits(64) modifier1, bits(64) modifier2,
                     bits(64) key0, bits(64) key1, boolean isgeneric)
    if UsePACIMP(isgeneric) then
        return ComputePAC2IMPDEF(data, modifier1, modifer2, key0, key1);
    if UsePACQARMA3(isgeneric) then
        boolean isqarma3 = TRUE;
        return ComputePAC2QARMA(data, modifier1, modifier2, key0, key1, isqarma3);
    if UsePACQARMA5(isgeneric) then
        boolean isqarma3 = FALSE;
        return ComputePAC2QARMA(data, modifier1, modifier2, key0, key1, isqarma3);
```

where ComputePAC2QARMA() is:

```
ComputePAC2QARMA(bits(64) data, bits(64) modifier1, bits(64) modifier2, bits(64)
 key0, bits(64) key1, boolean isqarma3)
     bits(64) concat_modifiers = modifier2<36:5>:modifier1<35:4>;
     return ComputePACQARMA(data, concat_modifiers, key0, key1, isqarma3);
```

2.65 D21851

In section **H9.4.10, "TRBDEVID, Device Configuration Register"**, the following text is removed:

> MPAM, bits [3:0]
>
> MPAM extensions. Indicates support for Memory Partitioning and Monitoring (MPAM) and the Trace Buffer MPAM extensions. The value of this field is an **IMPLEMENTATION DEFINED** choice of:

0b0000	MPAM not implemented by Trace Buffer Unit.
0b0001	MPAM implemented by Trace Buffer Unit, using default PARTID and PMG values.
0b0010	Trace Buffer MPAM extensions implemented.

When FEAT_MPAM is not implemented by the PE, this field reads as `0b0000`. When FEAT_MPAM is implemented by the PE, the value `0b0000` is not permitted. FEAT_TRBE_MPAM implements the functionality identified by the value `0b0010`. Access to this field is RO.

In section **D23.4.2 "TRBIDR_EL1, Trace Buffer ID Register"**, the following text is added:

MPAM, bits [15:12]

MPAM extensions. Indicates support for Memory Partitioning and Monitoring (MPAM) and the Trace Buffer MPAM extensions. The value of this field is an **IMPLEMENTATION DEFINED** choice of:

`0b0000`	MPAM not implemented by Trace Buffer Unit.
`0b0001`	MPAM implemented by Trace Buffer Unit, using default PARTID and PMG values.
`0b0010`	Trace Buffer MPAM extensions implemented.

When FEAT_MPAM is not implemented by the PE, this field reads as `0b0000`. When FEAT_MPAM and FEAT_TRBE_EXT are both implemented by the PE, the value `0b0000` is not permitted. When FEAT_TRBE_EXT is not implemented by the PE, the only permitted value is `0b0000`. FEAT_TRBE_MPAM implements the functionality identified by the value `0b0010`. Access to this field is RO.

The equivalent changes are made in section **H9.4.14 "TRBIDR_EL1, Trace Buffer ID Register"**.

2.66 D21856

In section **D23.2.76 "ID_AA64MMFR2_EL1, AArch64 Memory Model Feature Register 2"**, in the description of the NV field, bits [27:24], the text that reads:

NV	Meaning
`0b0000`	Nested virtualization is not supported.
`0b0001`	The HCR_EL2.{AT, NV1, NV} bits are implemented.
`0b0010`	The VNCR_EL2 register and the HCR_EL2.{NV2, AT, NV1, NV} bits are implemented.

is changed to read:

NV	Meaning
`0b0000`	If ID_AA64MMFR4_EL1.NV_frac != `0b0000`, support for nested virtualization is described in ID_AA64MMFR4_EL1.NV_frac. Otherwise, nested virtualization is not supported.
`0b0001`	The HCR_EL2.{AT, NV1, NV} bits are implemented.
`0b0010`	The VNCR_EL2 register and the HCR_EL2.{NV2, AT, NV1, NV} bits are implemented.

2.67 D21857

In section **C5.3.10 "DC CIGDPAE, Clean and invalidate of data and allocation tags by PA to PoE"**, the text that reads:

> This instruction is present only when FEAT_MEC is implemented. Otherwise, direct accesses to DC CIGDPAE are **UNDEFINED**.

is changed to read:

> This instruction is present only when FEAT_MEC and FEAT_MTE2 is implemented. Otherwise, direct accesses to DC CIGDPAE are **UNDEFINED**.

In section **C6.2.115 "DC"**, the Assembler symbols table with the entries that read:

> When FEAT_MEC is implemented, the following values are also valid:
>
> CIPAE when op1 = 100, CRm = 1110, op2 = 000
>
> CIGDPAE when op1 = 100, CRm = 1110, op2 = 111

is changed to read:

> When FEAT_MEC is implemented, the following value is also valid:
>
> CIPAE when op1 = 100, CRm = 1110, op2 = 000
>
> When FEAT_MEC and FEAT_MTE2 is implemented, the following value is also valid:
>
> CIGDPAE when op1 = 100, CRm = 1110, op2 = 111

2.68 D21864

In section **D23.2.53 "HCR_EL2, Hypervisor Configuration Register"**, in the field description of 'API, bit [41]', under the heading 'When FEAT_PAuth is implemented:', the text that reads:

> Controls the use of instructions related to Pointer Authentication:
>
> - In EL0, when HCR_EL2.TGE==0 or HCR_EL2.E2H==0, and the associated SCTLR_EL1.En<N><M>==1.
> - In EL1, the associated SCTLR_EL1.En<N><M>==1.
>
> Traps are reported using EC syndrome value `0x09`. The Pointer Authentication instructions trapped are:
>
> - AUTDA, AUTDB, AUTDZA, AUTDZB, AUTIA, AUTIA1716, AUTIASP, AUTIAZ, AUTIB, AUTIB1716, AUTIBSP, AUTIBZ, AUTIZA, AUTIZB.

- PACGA, PACDA, PACDB, PACDZA, PACDZB, PACIA, PACIA1716, PACIASP, PACIAZ, PACIB, PACIB1716, PACIBSP, PACIBZ, PACIZA, PACIZB.
- RETAA, RETAB, BRAA, BRAB, BLRAA, BLRAB, BRAAZ, BRABZ, BLRAAZ, BLRABZ.
- ERETAA, ERETAB, LDRAA, and LDRAB.

is changed to read:

> Controls the use of instructions related to Pointer Authentication:
>
> - PACGA
> - AUTDA, AUTDB, AUTDZA, AUTDZB, AUTIA, AUTIA1716, AUTIASP, AUTIAZ, AUTIB, AUTIB1716, AUTIBSP, AUTIBZ, AUTIZA, AUTIZB.
> - PACGA, PACDA, PACDB, PACDZA, PACDZB, PACIA, PACIA1716, PACIASP, PACIAZ, PACIB, PACIB1716, PACIBSP, PACIBZ, PACIZA, PACIZB.
> - RETAA, RETAB, BRAA, BRAB, BLRAA, BLRAB, BRAAZ, BRABZ, BLRAAZ, BLRABZ.
> - ERETAA, ERETAB, LDRAA, and LDRAB.
>
> This field is ignored if the instruction is disabled as a result of the SCTLR_ELx.{EnIB, EnIA, EnDA, EnDB} fields.

The text in the value `0b0` that reads:

> The instructions related to Pointer Authentication are trapped to EL2, when EL2 is enabled in the current Security state and the instructions are enabled for the EL1&0 translation regime, from:

is changed to read:

> The instructions related to Pointer Authentication are trapped to EL2 and reported using EC syndrome value `0x09`, when EL2 is enabled in the current Security state and the instructions are enabled for the EL1&0 translation regime, from:

2.69 D21870

In section **D23.4.29 "TRCIDR0, Trace ID Register 0"**, in the field description of 'TSMARK, bit [23]', the following text is removed:

> When FEAT_ETEv1p1 is implemented: … Otherwise: Reserved, **RES0**.

The rest of the contents of the field are unchanged.

An equivalent change is made in section **H9.3.29 "TRCIDR0, Trace ID Register 0"**, in the field description of 'TSMARK, bit [23]'.

2.70 D21871

In section **D23.5.28 "PMXEVTYPER_EL0, Performance Monitors Selected Event Type Register"**, under the heading 'Accessing PMXEVTYPER_EL0', the text that reads:

> Permitted writes of PMXEVTYPER_EL0 are ignored if all of the following are true:
> - FEAT_PMUv3p9 is implemented.
> - PSTATE.EL == EL0.
> - PMUSERENR_EL0.UEN == 1.
> - Any of the following are true:
>   - PMSELR_EL0.SEL != 31 and PMUACR_EL1.ER == 1.
>   - PMSELR_EL0.SEL == 31 and PMUACR_EL1.CR == 1.

is changed to read:

> Permitted writes of PMXEVTYPER_EL0 are ignored if all of the following are true:
> - FEAT_PMUv3p9 is implemented.
> - PSTATE.EL == EL0.
> - PMUSERENR_EL0.UEN == 1.
> - Any of the following are true:
>   - PMSELR_EL0.SEL != 31 and PMUSERENR_EL0.ER == 1.
>   - PMSELR_EL0.SEL == 31 and PMUSERENR_EL0.CR == 1.

The equivalent changes are made in section **G8.4.20 "PMXEVTYPER, Performance Monitors Selected Event Type Register"**, under the heading 'Accessing PMXEVTYPER'.

2.71 C21874

In section **J1.3.3 "shared/functions"**, in the pseudocode function EL3SDDUndef(), the text that reads:

```
// EL3SDDUndef()
// =============
// Returns TRUE if in Debug state and EDSCR.SDD is set.

boolean EL3SDDUndef()
    return Halted() && EDSCR.SDD == '1';
```

is changed to read:

```
// EL3SDDUndef()
// =============
// Returns TRUE if in Debug state and EDSCR.SDD is set.
```

```
boolean EL3SDDUndef()
    if Halted() && EDSCR.SDD == '1' then
        assert (PSTATE.EL != EL3  &&
                   (IsFeatureImplemented(FEAT_RME) || CurrentSecurityState() !=
 SS_Secure));
        return TRUE;
    else
        return FALSE;
```

The function EL3SDDUndefPriority() is similarly changed.

2.72 D21882

In section **J1.1.2 “aarch64/exceptions”**, in the pseudocode function AArch64.TakeReset(), the code that reads:

```
AArch64.TakeReset(boolean cold_reset)
    ...
    // Reset all other PSTATE fields
    ...
    if IsFeatureImplemented(FEAT_SME) then
        PSTATE.<SM,ZA> = '00';          // Disable Streaming SVE mode & ZA storage
        ResetSMEState('0');
    PSTATE.IL = '0';
    ...
```

is changed to read:

```
AArch64.TakeReset(boolean cold_reset)
    ...
    // Reset all other PSTATE fields
    ...
    if IsFeatureImplemented(FEAT_SME) then
        PSTATE.<SM,ZA> = '00';          // Disable Streaming SVE mode & ZA storage
        ResetSMEState('0');
    if IsFeatureImplemented(FEAT_SSBS) then
        PSTATE.SSBS = bit IMPLEMENTATION_DEFINED "PSTATE.SSBS bit at reset";
    if IsFeatureImplemented(FEAT_GCS) then
        PSTATE.EXLOCK = '0';            // PSTATE.EXLOCK is reset to 0 when
 resetting into AArch64
    PSTATE.IL = '0';
    ...
```

2.73 R21901

In section **D23.2.49 “GPCCR_EL3, Granule Protection Check Control Register (EL3)”**, in the field description of ‘GPCP, bit [17]’, the text that reads:

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

is changed to read:

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

An implementation is permitted to treat this field as **RES0**, with an Effective value of 0.

2.74 D21902

In section **D1.3.5 "Synchronous exception types"**, the definition of the priority of synchronous exception types places a Branch Target exception above all forms of **UNDEFINED** instruction exception.

However, for some instructions in the pseudocode this order can be interpreted in the opposite order and can imply that an **UNDEFINED** instruction exception takes priority over a Branch Target exception.

For example, the HLT instruction has the following decode code which shows the **UNDEFINED** check before indicating whether the instruction is a compatible branch target:

```
if EDSCR.HDE == '0' || !HaltingAllowed() then UNDEFINED;
if HaveBTIExt() then
    SetBTypeCompatible(TRUE);
```

Other instructions perform a check for an **UNDEFINED** instruction exception in the similar decode phase of the instruction and thus might be interpreted to cause an **UNDEFINED** instruction exception as higher priority than a Branch Target exception.

Arm will communicate the update of these instruction descriptions to ensure this ambiguity is resolved and ensure that a Branch Target exception is higher priority than an **UNDEFINED** instruction exception.

2.75 D21903

In section **J1.1.1 "aarch64/debug"**, in the pseudocode function AArch64.WatchpointByteMatch(), the code that reads:

```
boolean AArch64.WatchpointByteMatch(integer n, bits(64) vaddress)
   ...
   if mask > bottom then
        // If masked bits of DBGWVR_EL1[n] are not zero, the behavior is CONSTRAINED
 UNPREDICTABLE.
        if WVR_match && !IsZero(DBGWVR_EL1[n]<cmpbottom-1:bottombit>) then
            WVR_match = ConstrainUnpredictableBool(Unpredictable_WPMASKEDBITS);
   ...
```

is changed to read:

```
boolean AArch64.WatchpointByteMatch(integer n, bits(64) vaddress)
   ...
   if mask > bottom then
        // If masked bits of DBGWVR_EL1[n] are not zero, the behavior is CONSTRAINED
 UNPREDICTABLE.
        if WVR_match && !IsZero(DBGWVR_EL1[n]<cmplsb-1:bottombit>) then
```

```
            WVR_match = ConstrainUnpredictableBool(Unpredictable_WPMASKEDBITS);
    ...
```

The equivalent changes are made in section **J1.2.1 "aarch32/debug"**, in the pseudocode function AArch32.WatchpointByteMatch().

2.76 C21907

In section **D8.4.5.3 "PSTATE.BTYPE"**, the following rule:

> R_{LJHCL}
>
> If the PSTATE.BTYPE field is not `0b00` and an attempt is made to execute an instruction within a guarded page, a Branch Target exception is generated unless the instruction is one of the following:
>
> - A BTI instruction that is compatible with the PSTATE.BTYPE field.
> - A PACIASP or PACIBSP instruction, and the PSTATE.BTYPE is consistent with implicit BTI behavior of these instructions.
> - A Breakpoint Instruction exception.
> - A Halt Instruction debug event.

is changed to read:

> R_{LJHCL}
>
> If the PSTATE.BTYPE field is not `0b00` and an attempt is made to execute an instruction within a guarded page, a Branch Target exception is generated unless the instruction is one of the following:
>
> - A BTI instruction that is compatible with the PSTATE.BTYPE field.
> - A PACIASP or PACIBSP instruction, and the PSTATE.BTYPE is consistent with implicit BTI behavior of these instructions.

In the same section, the following rule is added:

> I_{X0001}
>
> The Software Breakpoint instruction and, if Halting is enabled and allowed, the Halt instruction always generate a higher priority exception or debug event than a Branch Target exception.

2.77 C21915

In section **D12.1.2 "The system counter"**, the following text that reads:

> Frequency

> From Armv8.0 to Armv8.5 inclusive, increments at a fixed frequency, typically in the range 1-50MHz. It can support one or more alternative operating modes in which it increments by larger amounts at a lower frequency, typically for power-saving. From Armv8.6, increments at a fixed frequency of 1GHz.

is changed to read:

> Effective frequency
>
> This indicates the amount of time each unit of the counter represents. For example:
>
> An effective frequency of 50MHz means each unit of the counter represents 20ns. An effective frequency of 1GHz means each unit of the counter represents 1ns. The effective frequency is indicated in the CNTFRQ_EL0 register.
>
> From Armv8.0 to Armv8.5 inclusive, the effective frequency is a fixed value, typically in the range 1-50MHz.
>
> From Armv8.6, the effective frequency is a fixed value of 1GHz.
>
> The counter might not be driven by a clock running at the effective frequency, for example a counter with an effective frequency of 1GHz might have an actual clock running at 125MHz but the counter increments by 8 on each tick of the clock.
>
> Counter resolution
>
> The counter resolution is a representation of how frequently the counter is updated.
>
> For example, a counter might have an effective frequency of 1GHz, but the actual clock runs at 125MHz and therefore the counter resolution is 125Mhz.
>
> From Armv8.6, Arm recommends the counter resolution is not less than 50MHz in normal running operation.
>
> The counter resolution might vary over time, to accommodate different operating modes of the system such as lower power modes. Such variation might be invisible to the programmer and be system specific, or might be controllable by the programmer through the frequency modes table.

In general, the terms are replaced as follows:

- Clock frequency or counter frequency is replaced with effective frequency
- Update frequency is replaced with counter resolution

For example, the text in section **D12.1.2 “The system counter”** that reads:

> To support lower-power operating modes in architectures from Armv8.0 to Armv8.5, the counter can increment by larger amounts at a lower frequency. For example, a 10MHz system counter might either increment:
>
> - By 1 at 10MHz.

> - By 500 at 20kHz, when the system lowers the clock frequency, to reduce power consumption. In this case, the counter must support transitions between high-frequency, high-precision operation, and lower-frequency, lower-precision operation, without any impact on the required accuracy of the counter.

is changed to read:

> To support lower-power operating modes in architectures from Armv8.0 to Armv8.5, the counter can operate with a lower counter resolution by incrementing by larger amounts at a lower actual clock frequency. For example, a system counter with an effective frequency of 10MHz might either increment:
>
> - By 1 at 10MHz.
> - By 500 at 20kHz, when the system lowers the actual clock frequency, to reduce power consumption. In this case, the counter must support transitions between higher resolution operation and lower resolution operation, without any impact on the required accuracy of the counter.

The equivalent changes are made in the rest of **D12 "The Generic Timer in AArch64 state"** and in **I2 "System Level Implementation of the Generic Timer"**.

In Section **I2.2 "Memory-mapped counter module"**, the text that reads:

> The first entry in the table defines the base frequency of the system counter. This is the maximum frequency at which the counter updates.

is changed to read:

> The first entry in the table defines the base frequency of the system counter. It is **IMPLEMENTATION DEFINED** whether the base frequency indicates the effective frequency of the counter, or the maximum counter resolution. Arm recommends that the base frequency indicates the maximum counter resolution.

In section **I2.2.1.1 "The Frequency modes table"**, the text that reads:

> When the system counter is operating at a lower frequency than the base frequency, the increment applied at each counter update is given by: increment = (base_frequency) / (selected_frequency)

is changed to read:

> When the system counter is operating at a lower frequency than the base frequency, the increment applied at each counter update is increased by the ratio of the base frequency and the selected frequency.
>
> Note: Where the effective frequency and the base frequency are not the same, the increment is not solely defined by the ratio of the base frequency and the selected frequency. For example, a counter with an effective frequency of 1GHz and a base frequency of 125MHz will increment by a value of 8 on each tick of the base frequency. If the Frequency modes table provides an

> additional selected frequency of 31.25MHz, when this frequency is selected then counter increments by 32 on each tick of the selected frequency.

2.78 D21925

In section **D23.2.173 TCR2_EL2, Extended Translation Control Register (EL2)**, under the heading 'When the Effective value of HCR_EL2.E2H is 1:', in the field 'D128, bit [5]', the following text is removed:

> This field is IGNORED when TCR2_EL2.D128 is 0.

2.79 D21933

In section **J1.1.2 "aarch64/exceptions"**, in function AArch64.CheckForSMCUndefOrTrap(), the code that reads:

```
AArch64.CheckForSMCUndefOrTrap(bits(16) imm)
...
    if !HaveEL(EL3) then
        if PSTATE.EL == EL1 && EL2Enabled() then
            if EffectiveHCR_EL2_NVx()<0> == '1' && HCR_EL2.TSC == '1' then
                route_to_el2 = TRUE;
            else
                UNDEFINED;
        else
            UNDEFINED;
...
```

is changed to read:

```
AArch64.CheckForSMCUndefOrTrap(bits(16) imm)
...
    if !HaveEL(EL3) then
        if (PSTATE.EL == EL1 && EL2Enabled() && HCR_EL2.TSC == '1' &&
              (EffectiveHCR_EL2_NVx() IN {'xx1'} ||
              (boolean IMPLEMENTATION_DEFINED "Trap SMC execution at EL1 to EL2")))
 then
            route_to_el2 = TRUE;
        else
            UNDEFINED;
...
```

2.80 D21938

In section **J1.3.3 "shared/functions"**, in the pseudocode function TestEventCNTV(), the code segment that reads:

```
TestEventCNTV(bits(64) prev_physical_count, bits(64) current_physical_count)
    ...
```

```
    if !ELIsInHost(EL0) && CNTKCTL_EL1.EVNTEN == '1' then
        n = UInt(CNTKCTL_EL1.EVNTI);
        ...
        if HaveEL(EL2) && !ELIsInHost(EL0) then
            offset = CNTVOFF_EL2;
        else
            offset = Zeros(64);
        ...
    return;
```

is changed to read:

```
TestEventCNTV(bits(64) prev_physical_count, bits(64) current_physical_count)
    ...
    if (EffectiveHCR_EL2_E2H():EffectiveTGE() != '11' &&
        CNTKCTL_EL1.EVNTEN == '1') then
        n = UInt(CNTKCTL_EL1.EVNTI);
        ...
        offset = if HaveEL(EL2) then CNTVOFF_EL2 else Zeros(64);
        ...
    return;
```

2.81 D21944

In section **D23.2.53 "HCR_EL2, Hypervisor Configuration Register"**, in the field description of 'NV1, bit [43]', the text that reads:

> If HCR_EL2.NV2 is 1, the value of HCR_EL2.NV1 defines which EL1 register accesses are transformed to loads and stores. These transformed accesses have priority over the trapping of registers.
>
> The trapping of EL1 registers caused by other control bits has priority over the transformation of these accesses.

is changed to read:

> If the Effective value of HCR_EL2.NV2 is 1, the Effective value of HCR_EL2.NV1 defines which EL1 register accesses are transformed to loads and stores.
>
> The trapping of EL1 registers caused by other control bits has priority over the transformation of these accesses.

2.82 D21946

In section **D23.2.74 "ID_AA64MMFR0_EL1, AArch64 Memory Model Feature Register 0"**, in the 'PARange' field description, the value table row that reads:

0b0110	When FEAT_LPA is implemented or FEAT_LPA2 is implemented: 52 bits, 4PB.

is changed to read:

0b0110	When FEAT_LPA is implemented: 52 bits, 4PB.

In the following sections:

- **D23.2.187 "TTBR0_EL1, Translation Table Base Register 0 (EL1)"**.
- **D23.2.188 "TTBR0_EL2, Translation Table Base Register 0 (EL2)"**.
- **D23.2.189 "TTBR0_EL3, Translation Table Base Register 0 (EL3)"**.
- **D23.2.190 "TTBR1_EL1, Translation Table Base Register 1 (EL1)"**.
- **D23.2.191 "TTBR1_EL2, Translation Table Base Register 1 (EL2)"**.

In each of the listed TTBRx_ELx, in the field description of 'BADDR[47:1], bits [47:1]', the text that reads:

> If FEAT_LPA is implemented and the value of TCR_ELx.IPS is 0b110, then:
> - Bits A[51:48] of the stage 1 translation table base address bits are in register bits[5:2].
> - Register bit[1] is **RES0**.
> - The smallest permitted value of x is 6.
> - When x>6, register bits[(x-1):6] are **RES0**.

is changed to read:

> The BADDR field represents a 52-bit address if one of the following applies:
> - FEAT_LPA is implemented, the 64KB granule size is in use, and the value of TCR_ELx.IPS is 0b110.
> - FEAT_LPA2 is implemented, the 4KB or 16KB granule size is in use, and the Effective value of TCR_ELx.DS is 1.
> - FEAT_D128 is implemented, 56-bit PAs are supported, the 64KB granule size is in use, and the value of TCR2_EL1.D128 is 0.
>
> When TTBR0_ELx.BADDR represents a 52-bit address, all of the following apply:
> - Bits A[51:48] of the stage 1 translation table base address bits are in register bits[5:2].
> - Register bit[1] is **RES0**.
> - The smallest permitted value of x is 6.
> - When x>6, register bits[(x-1):6] are **RES0**.

In the same field description, the Note that reads:

> If an OA size of more than 48 bits is in use, and the translation table has fewer than eight entries, the table must be aligned to 64 bytes. Otherwise the translation table must be aligned to the size of the table.

is changed to read:

> If BADDR expresses 52-bit address, and the translation table has fewer than eight entries, the table must be aligned to 64 bytes. Otherwise the translation table must be aligned to the size of the table.

In addition, the Note in the field description of 'BADDR[47:1], bits [47:1]' that reads:

> Note:
>
> TCR_ELx.IPS==`0b110` is permitted when:
>
> - FEAT_LPA is implemented and the 64KB translation granule is used.
> - FEAT_LPA2 is implemented and the 4KB or 16KB translation granule is used.
>
> When the value of ID_AA64MMFR0_ELx.PARange indicates that the implementation does not support a 52 bit PA size, if a translation table lookup uses this register when the Effective value of TCR_ELx.IPS is `0b110` and the value of register bits[5:2] is nonzero, an Address size fault is generated. When the value of ID_AA64MMFR0_ELx.PARange indicates that the implementation supports a 56 bit PA size, bits A[55:52] of the stage 1 translation table base address are zero.

is changed to read:

> Note:
>
> For the 64KB granule, if FEAT_LPA is not implemented, and the value of TCR_ELx.IPS is `0b110`, one of the following **IMPLEMENTATION DEFINED** behaviors occur:
>
> - BADDR uses the extended format to represent a 52-bit base address.
> - BADDR does not use the extended format.
>
> When the value of ID_AA64MMFR0_ELx.PARange indicates that the implementation supports a 56-bit PA size, bits A[55:52] of the stage 1 translation table base address are zero.

In the following sections:

- **D23.2.174 "TCR_EL1, Translation Control Register (EL1)"**.
- **D23.2.175 "TCR_EL2, Translation Control Register (EL2)"**.
- **D23.2.176 "TCR_EL3, Translation Control Register (EL3)"**.

In each of the TCR_ELx, in the field descriptions of 'IPS, bits [43:32]' or 'PS, bits [18:16]', the text that reads:

> If the translation granule is not 64KB and FEAT_LPA2 is not implemented, the value `0b110` is treated as reserved.
>
> It is **IMPLEMENTATION DEFINED** whether an implementation that does not implement FEAT_LPA supports setting the value of `0b110` for the 64KB translation granule size or whether setting this value behaves as the `0b101` encoding. "If the value of ID_AA64MMFR0_ELx.PARange is `0b0110`, and the value of this field is not `0b110` or a value treated as `0b110`, then bits[51:48] of every translation table base address are `0b0000` for the stage of translation controlled by TCR_ELx.

is changed to read:

> The value `0b110` represents the following output address sizes:
>
> - For the 64KB translation granule size, if FEAT_LPA is implemented, the value `0b110` represents 52 bits .
> - For the 4KB and 16KB granule sizes, only if both FEAT_LPA and FEAT_LPA2 are implemented, the value `0b110` represents 52 bits .
> - Otherwise, the value `0b110` behaves as the `0b101` value and represents 48 bits.

In section **D23.2.201 "VSTTBR_EL2, Virtualization Secure Translation Table Base Register"**, in the field description of 'BADDR[47:1], bits [47:1]', the text that reads:

> If the value of VTCR_EL2.PS is `0b110`, then:
>
> - Register bits[47:z] hold bits[47:z] of the stage 2 translation table base address, where z is determined as follows:
>   - If x >= 6 then z=x.
>   - Otherwise, z=6.
> - Register bits[5:2] hold bits[51:48] of the stage 2 translation table base address.
> - When z>x register bits[(z-1):x] are **RES0**, and bits[(z-1):x] of the translation table base address are zero.
> - When x>6 register bits[(x-1):6] are **RES0**.
> - Register bit[1] is **RES0**.
> - Bits[5:2] of the stage 2 translation table base address are zero.

is changed to read:

> The BADDR represents a 52-bit address if one of the following applies:
>
> - FEAT_LPA is implemented, the 64KB granule size is in use, and the value of VTCR_EL2.PS is `0b110`.
> - FEAT_LPA2 is implemented, the 4KB or 16KB granule size is in use, and the Effective value of VTCR_EL2.DS is 1.
> - FEAT_D128 is implemented, 56-bit PAs are supported, the 64KB granule size is in use, and the value of VTCR2_EL1.D128 is 0.
>
> When VSTTBR_EL2.BADDR expresses a 52-bit address, all of the following apply:
>
> - Register bits[47:x] hold bits[47:x] of the stage 2 translation table base address, where x is determined by the size of the translation table at the start level.
> - The smallest permitted value of x is 6.
> - Register bits[5:2] hold bits[51:48] of the stage 2 translation table base address.
> - Bits[x:0] of the translation table base address are zero.
> - When x>6 register bits[(x-1):6] are **RES0**.

- Register bit[1] is **RES0**.

In the same field description, the Note that reads:

> If an OA size of more than 48 bits is in use, and the translation table has fewer than eight entries, the table must be aligned to 64 bytes. Otherwise the translation table must be aligned to the size of the table.

is changed to read:

> If BADDR expresses 52-bit address, and the translation table has fewer than eight entries, the table must be aligned to 64 bytes. Otherwise the translation table must be aligned to the size of the table.

In the same field description, the Note that reads:

> When the value of ID_AA64MMFR0_EL1.PARange indicates that the implementation does not support a 52-bit PA size, if a translation table lookup uses this register with the 64KB translation granule when the Effective value of VTCR_EL2.PS is `0b110` and the value of register bits[5:2] is nonzero, an Address size fault is generated.

is changed to read:

> For the 64KB granule, if FEAT_LPA is not implemented, and the value of AArch64-VTCR_EL2.PS is `0b110`, one of the following IMPLEMENTATION DEFINED behaviors occur:
>
> - BADDR uses the extended format to represent a 52-bit base address.
> - BADDR does not use the extended format.
>
> When the value of ID_AA64MMFR0_EL1.PARange indicates that the implementation supports a 56 bit PA size, bits A[55:52] of the stage 2 translation table base address are zero.

In section **D23.2.203 "VTTBR_EL2, Virtualization Translation Table Base Register"**, in the field description of 'BADDR[47:1], bits [47:1]', the text that reads:

> In an implementation that includes FEAT_LPA, if the value of VTCR_EL2.PS is `0b110`, then:
>
> - Register bits[47:z] hold bits[47:z] of the stage 2 translation table base address, where z is determined as follows:
>   - If x >= 6 then z=x.
>   - Otherwise, z=6.
> - Register bits[5:2] hold bits[51:48] of the stage 2 translation table base address.
> - When z>x register bits[(z-1):x] are **RES0**, and bits[(z-1):x] of the translation table base address are zero.
> - When x>6 register bits[(x-1):6] are **RES0**.
> - Register bit[1] is **RES0**.
> - Bits[5:2] of the stage 2 translation table base address are zero.

> - In an implementation that includes FEAT_TTCNP, bit[0] of the stage 2 translation table base address is zero.

is changed to read:

> The BADDR represents a 52-bit address if one of the following applies:
> - FEAT_LPA is implemented, the 64KB granule size is in use, and the value of VTCR_EL2.PS is `0b110`.
> - FEAT_LPA2 is implemented, the 4KB or 16KB granule size is in use, and the Effective value of VTCR_EL2.DS is 1.
> - FEAT_D128 is implemented, 56-bit PAs are supported, the 64KB granule size is in use, and the value of VTCR_EL2.D128 is 0.
>
> When VTTBR_EL2.BADDR represents a 52-bit address, all of the following apply:
> - Register bits[47:x] hold bits[47:x] of the stage 2 translation table base address, where x is determined by the size of the translation table at the start level.
> - The smallest permitted value of x is 6.
> - Register bits[5:2] hold bits[51:48] of the stage 2 translation table base address.
> - Bits[x:0] of the translation table base address are zero.
> - When x>6 register bits[(x-1):6] are **RES0**.
> - Register bit[1] is **RES0**.

In the same field description, the Note that reads:

> If an OA size of more than 48 bits is in use, and the translation table has fewer than eight entries, the table must be aligned to 64 bytes. Otherwise the translation table must be aligned to the size of the table.

is changed to read:

> If BADDR expresses 52-bit address, and the translation table has fewer than eight entries, the table must be aligned to 64 bytes. Otherwise the translation table must be aligned to the size of the table.

In the same field description, the Note that reads:

> When the value of ID_AA64MMFR0_EL1.PARange indicates that the implementation does not support a 52 bit PA size, if a translation table lookup uses this register when the Effective value of VTCR_EL2.PS is `0b110` and the value of register bits[5:2] is nonzero, an Address size fault is generated.

is changed to read:

> For the 64KB granule, if FEAT_LPA is not implemented, and the value of AArch64-VTCR_EL2.PS is `0b110`, one of the following IMPLEMENTATION DEFINED behaviors occur:
> - BADDR uses the extended format to represent a 52-bit base address.

- BADDR does not use the extended format.

When the value of ID_AA64MMFR0_EL1.PARange indicates that the implementation supports a 56 bit PA size, bits A[55:52] of the stage 2 translation table base address are zero.

In section **D23.2.202 "VTCR_EL2, Virtualization Translation Control Register"**, in the field description of 'PS, bits [18:16]', the text that reads:

If the translation granule is not 64KB and FEAT_LPA2 is not implemented, the value `0b110` is treated as reserved. If the translation granule is not 64KB, the value `0b110` is treated as reserved. It is **IMPLEMENTATION DEFINED** whether an implementation that does not implement FEAT_LPA supports setting the value of `0b110` for the 64KB translation granule size or whether setting this value behaves as the `0b101` encoding. In an implementation that supports 52-bit PAs, if the value of this field is not `0b110` or a value treated as `0b110`, then bits[51:48] of every translation table base address for the stage of translation controlled by VTCR_EL2 are `0b0000`.

is changed to read:

The value `0b110` represents the following output address sizes:

- For the 64KB granule size, if FEAT_LPA is implemented, the value `0b110` represents 52 bits .
- For the 4KB and 16KB granule sizes, only if both FEAT_LPA is implemented and the Effective value of VTCR_EL2.DS is `0b1`, then the value `0b110` represents 52 bits .
- Otherwise, the value `0b110` encoding behaves as the `0b101` value and represents 48 bits.

2.83 D21954

In section **J1.1.2 "aarch64/exceptions"**, in the pseudocode function AArch64.CheckForWFxTrap(), the code that reads:

```
AArch64.CheckForWFxTrap(bits(2) target_el, WFxType wfxtype)
    assert HaveEL(target_el);
    ...
    if trap then
        AArch64.WFxTrap(wfxtype, target_el);
```

is changed to read:

```
AArch64.CheckForWFxTrap(bits(2) target_el, WFxType wfxtype)
    assert HaveEL(target_el);
    ...
    if trap then
        if target_el == EL3 && EL3SDDUndef() then
            UNDEFINED;
        AArch64.WFxTrap(wfxtype, target_el);
```

An equivalent change is made to section **J1.2.2 "aarch32/exceptions"**, in the pseudocode function AArch32.CheckForWFxTrap().

In section **J1.3.3 “shared/functions”**, in the pseudocode function Hint_WFE(), the code that reads:

```
Hint_WFE(integer localtimeout, WFxType wfxtype)
    ...
    elsif HaveFeatWFxT() && LocalTimeoutEvent(localtimeout) then
    // No further operation if the local timeout has expired.
        EndOfInstruction();
    else
        bits(2) target_el;
        trap = FALSE;
        if PSTATE.EL == EL0 then
    ...
    if trap && PSTATE.EL != EL3 then
        // Determine if trap delay is enabled and delay amount
        (delay_enabled, delay) = WFETrapDelay(target_el);
        if !WaitForEventUntilDelay(delay_enabled, delay) then
            // Event did not arrive before delay expired so trap WFE
            AArch64.WFxTrap(wfxtype, target_el);
    ...
```

is changed to read:

```
Hint_WFE(integer localtimeout, WFxType wfxtype)
    ...
    elsif HaveFeatWFxT() && LocalTimeoutEvent(localtimeout) then
    // No further operation if the local timeout has expired.
        EndOfInstruction();
    else
        bits(2) target_el;
        trap = FALSE;
        if HaveEL(EL3) && EL3SDDUndefPriority() then
            // Check for traps described by the Secure Monitor.
            // If the trap is enabled, the instruction will be UNDEFINED because
 EDSCR.SDD is 1.
            if isFeatureImplemented(FEAT_TWED) then
                trap      = SCR_EL3.TWE == '1';
                target_el = EL3;
            else
                AArch64.CheckForWFxTrap(EL3, wfxtype);
        if !trap && PSTATE.EL == EL0 then
        ...

        if trap && PSTATE.EL != EL3 then
            // Determine if trap delay is enabled and delay amount
            (delay_enabled, delay) = WFETrapDelay(target_el);
            if !WaitForEventUntilDelay(delay_enabled, delay) then
                // Event did not arrive before delay expired so trap WFE
                if target_el == EL3 && EL3SDDUndef() then
                    UNDEFINED;
                else
                    AArch64.WFxTrap(wfxtype, target_el);
        ...
```

In the function Hint_WFI(), the code that reads:

```
Hint_WFI(integer localtimeout, WFxType wfxtype)
    ...
    if InterruptPending() || (HaveFeatWFxT() && LocalTimeoutEvent(localtimeout))
 then
        // No further operation if an interrupt is pending or the local timeout has
 expired.
        EndOfInstruction();
    else
        if PSTATE.EL == EL0 then
```

```
    ...
```

is changed to read:

```
Hint_WFI(integer localtimeout, WFxType wfxtype)
    ...
    if InterruptPending() || (HaveFeatWFxT() && LocalTimeoutEvent(localtimeout))
 then
        // No further operation if an interrupt is pending or the local timeout has
 expired.
        EndOfInstruction();
    else
        if HaveEL(EL3) && EL3SDDUndefPriority() then
            // Check for traps described by the Secure Monitor.
            // If the trap is enabled, the instruction will be UNDEFINED because
 EDSCR.SDD is 1.
            AArch64.CheckForWFxTrap(EL3, wfxtype);
        if PSTATE.EL == EL0 then
    ...
```

In section **F5.1.300 “WFE”**, the operational pseudocode that reads:

```
if ConditionPassed() then
    EncodingSpecificOperations();
    if isEventRegisterSet() then
        ClearEventRegister();
    else
        if PSTATE.EL == EL0 then
        ...
```

is changed to read:

```
if ConditionPassed() then
        EncodingSpecificOperations();
    if isEventRegisterSet() then
        ClearEventRegister();
    else
        if HaveEL(EL3) && EL3SDDUndefPriority() then
            // Check for traps described by the Secure Monitor.
            // If the trap is enabled, the instruction will be UNDEFINED because
 EDSCR.SDD is 1.
            AArch32.CheckForWFxTrap(EL3, WFxType_WFE);
        if PSTATE.EL == EL0 then
        ...
```

An equivalent change is made in section **F5.1.301 “WFI”**.

2.84 E21957

In section **B2.9 “Restrictions on the effects of speculation”**, the following new subsection B2.9.5 “Restrictions on the effects of speculation from Armv9.5” is added:

If FEAT_BTI is implemented:

- Branches predicted or executed under speculation into a Guarded Page that land on instructions other than intended BTI instructions are also guarded against speculative execution of those instructions.

2.85 D21968

In section **J1.3.1 "shared/debug"**, in the pseudocode function CheckForPMUOverflow(), the code that reads:

```
CheckForPMUOverflow()
    ...
        if pmuirq && HaltingAllowed() && EDECR.PME == '1' then
            Halt(DebugHalt_EDBGRQ);
    ...
```

is changedto read:

```
CheckForPMUOverflow()
    ...
         if pmuirq && EDECR.PME == '1' then
            ExternalDebugRequest();
    ...
```

2.86 D21971

In section **A2.2.3 "The Armv8.2 architecture extension"**, under the heading 'FEAT_PCSRv8p2, PC Sample-based Profiling Extension' the text that reads:

> If FEAT_SEL2 and FEAT_PCSRv8 are implemented, then FEAT_PCSRv8p2 is implemented.

is removed, and the following text is added to the text under the heading 'FEAT_SEL2, Secure EL2':

> If FEAT_SEL2 is implemented, then FEAT_PCSRv8 is not implemented.

In section **A2.3.3 "The Armv9.2 architecture extension"**, under the heading 'FEAT_RME, Realm Management Extension' the text that reads:

> When FEAT_PCSRv8 and FEAT_RME are implemented, FEAT_PCSRv8p2 is implemented.

is changed to read:

> When FEAT_RME is implemented, FEAT_PCSRv8 is not implemented.

2.87 D21978

In **D23.2.105 "LORC_EL1, LORegion Control (EL1)"**, the pseudocode that reads:

```
elsif PSTATE.EL == EL1 then
    ...
    elsif SCR_EL3.NS == '0' then
        UNDEFINED;
...
elsif PSTATE.EL == EL2 then
    if SCR_EL3.NS == '0' then
        UNDEFINED;
```

is changed to read:

```
elsif PSTATE.EL == EL1 then
    ...
    elsif HaveEL(EL3) && SCR_EL3.NS == '0' then
        UNDEFINED;
...
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && SCR_EL3.NS == '0' then
        UNDEFINED;
```

Similar changes are made in the following sections:

- **D23.2.106 "LOREA_EL1, LORegion End Address (EL1)"**.
- **D23.2.108 "LORN_EL1, LORegion Number (EL1)"**.
- **D23.2.109 "LORSA_EL1, LORegion Start Address (EL1)"**.

In **D23.10.27 "CNTV_TVAL_EL0, Counter-timer Virtual Timer TimerValue register"**, the pseudocode that reads:

```
SCR_EL3.NS == '0'
```

is changed to read:

```
IsCurrentSecurityState(SS_Secure)
```

Similarly, pseudocode that reads:

```
SCR_EL3.NS == '1'
```

is changed to read:

```
!IsCurrentSecurityState(SS_Secure)
```

where CurrentSecurityState() handles the effective value of SCR_EL3 in implementations without EL3, or when FEAT_RME adds Realm and Root states.

The equivalent changes have been made in other CNT* registers in chapters **D23 "AArch64 System Register Descriptions"** and **G8 "AArch32 System Register Descriptions"**.

In **G8.2.120 "NSACR, Non-Secure Access Control Register"**, the pseudocode that reads:

```
elsif PSTATE.EL == EL1 then
    ...
    elsif !ELUsingAArch32(EL2) && SCR_EL3.<NS,EEL2> == '01' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif !ELUsingAArch32(EL3) && SCR_EL3.NS == '0' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x03);
```

is changed to read:

```
elsif PSTATE.EL == EL1 then
    ...
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && IsCurrentSecurityState(SS_Secure)
 then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif !ELUsingAArch32(EL3) && IsCurrentSecurityState(SS_Secure) then
        AArch64.AArch32SystemAccessTrap(EL3, 0x03);
```

The equivalent changes are made in the following sections:

- **G8.2.9 "ATS12NSOPR, Address Translate Stages 1 and 2 Non-secure Only PL1 Read"**.
- **G8.2.10 "ATS12NSOPW, Address Translate Stages 1 and 2 Non-secure Only PL1 Write"**.
- **G8.2.11 "ATS12NSOUR, Address Translate Stages 1 and 2 Non-secure Only Unprivileged Read"**.
- **G8.2.12 "ATS12NSOUW, Address Translate Stages 1 and 2 Non-secure Only Unprivileged Write"**.
- **G8.2.115 "MVBAR, Monitor Vector Base Address Register"**.
- **G8.2.125 "RVBAR, Reset Vector Base Address Register"**.
- **G8.2.126 "SCR, Secure Configuration Register"**.
- **G8.3.35 "SDCR, Secure Debug Control Register"**.

2.88 D21994

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

> If FEAT_RME is implemented, then if SCR_EL3.{NSE, NS} is {1, 0}, an exception return from EL3.

is changed to read:

> If FEAT_RME is implemented, then if SCR_EL3.{NSE, NS} is {1, 0}, an exception return from EL3 to a lower Exception level.

2.89 C22003

In section **D8.13.6.2 "Loads and stores generated by transforming register accesses"**, in rule R_{VFMQB}, the text that reads:

> If there is no context synchronizing operation between the register access and a load or store instruction accessing the address of the transformed memory access, then the transformed memory access can be reordered with respect to any reads or writes at EL1 caused by load or store instructions to the same address.

is changed to read:

> The memory access is ordered as if it was generated by a load or store instruction.

2.90 D22008

In section **A1.5.8.1 "Operations that do not generate floating point exceptions"**, the text that reads:

> If FPCR.AH is 1, all of the following instructions do not generate any floating-point exceptions regardless of their input values:
>
> - The BFloat16 instructions defined in A1.5.10 Alternate BFloat16 behaviors.
> - Single-precision, double-precision and half-precision instructions FRECPE, FRECPS, FRECPX, FRSQRTE, and FRSQRTS.
> - Floating-point to integer and floating-point rounding instructions: FCVTMS (scalar), FCVTMS (vector), FCVTMU (scalar), FCVTMU (vector), FCVTNS (scalar), FCVTNS (vector), FCVTNU (scalar), FCVTNU (vector), FCVTPS (scalar), FCVTPS (vector), FCVTPU (scalar), FCVTPU (vector), FCVTZS (scalar, fixed-point), FCVTZS (scalar, integer), FCVTZS (vector, fixed-point), FCVTZS (vector, integer), FCVTZU (scalar, fixed-point), FCVTZU (scalar, integer), FCVTZU (vector, fixed-point), FCVTAS (scalar), FCVTAS (vector), FCVTAU (scalar), FCVTAU (vector), FCVTZS (scalar, fixed-point), FCVTZS (scalar, integer), FCVTZS (vector, fixed-point), FCVTZS (vector, integer), FRINTA (scalar), FRINTA (vector), FRINTZ (scalar), FRINTZ (vector), FRINTM (scalar), FRINTM (vector), FRINTP (scalar), FRINTP (vector), FRINTN (scalar), FRINTN (vector), FRINTX (scalar), FRINTX (vector), FRINTI (scalar), FRINTI (vector), FRINT32X (scalar), FRINT32X (vector), FRINT32Z (scalar), FRINT32Z (vector), FRINT64X (scalar), FRINT64X (vector), FRINT64Z (scalar), and FRINT64Z (vector).

is changed to read:

> If FPCR.AH is 1, all of the following instructions do not generate any floating-point exceptions regardless of their input values:
>
> - BF16 instructions BFMLALB, BFMLALT (by element), BFMLALB, BFMLALT (vector), BFCVT, BFCVTN, BFCVTN2, and BFCVTNT.
> - Single-precision, double-precision and half-precision instructions FRECPE, FRECPS, FRECPX, FRSQRTE, and FRSQRTS.

2.91 D22014

In section **C5.2.19 "SPSR_EL1, Saved Program Status Register (EL1)"**, under the heading 'When exception taken from AArch64 state:', in the field description of 'M[3:0], bits [3:0]', the table that reads:

M[3:0]	Meaning
0b0000	EL0.
0b0100	EL1 with SP_EL0 (ELt).
0b0101	EL1 with SP_EL1 (EL1h).

is changed to read:

M[3:0]	Meaning
0b0000	EL0.
0b0100	EL1 with SP_EL0 (EL1t).
0b0101	EL1 with SP_EL1 (EL1h).

2.92 D22015

In section **C5.2.19 "SPSR_EL1, Saved Program Status Register (EL1)"**, under the heading 'When exception taken from AArch64 state:', in the field description of 'M[3:0], bits [3:0]', the text that reads:

> M[3:2]: On an exception to EL1:
>
> - If the Effective value of HCR_EL2.{NV, NV1} != {1, 0} or the exception is not taken from EL1, then M[3:2] is set to the value of PSTATE.EL on taking an exception to EL1.
> - If the Effective value of HCR_EL2.{NV, NV1} == {1, 0} and the exception is not taken from EL1, then M[3:2] is set to 0b10.
> - M[3:2] is copied to PSTATE.EL on executing a legal exception return operation in EL1.

is changed to read:

> M[3:2]: On an exception to EL1:
>
> - If the Effective value of HCR_EL2.{NV, NV1} != {1, 0} or the exception is not taken from EL1, then M[3:2] is set to the value of PSTATE.EL on taking an exception to EL1.
> - If the Effective value of HCR_EL2.{NV, NV1} == {1, 0} and the exception is taken from EL1, then M[3:2] is set to 0b10.
> - M[3:2] is copied to PSTATE.EL on executing a legal exception return operation in EL1.

2.93 D22021

In section **J1.3.1 “shared/debug”**, in the pseudocode function PCSRSuspended(), the following code segment:

```
// PCSRSuspended()
// ===============
// Returns TRUE if PC Sample-based Profiling is suspended, and FALSE
 otherwise.boolean PCSRSuspended()

boolean PCSRSuspended()
    if PMPCSCTL.IMP == '1' then
        return PMPCSCTL.EN == '0';
    else
        return boolean IMPLEMENTATION_DEFINED "PCSR is suspended";
```

is changed to read:

```
// PCSRSuspended()
// ===============
// Returns TRUE if PC Sample-based Profiling is suspended, and FALSE
 otherwise.boolean PCSRSuspended()

boolean PCSRSuspended()
    if IsFeatureImplemented(FEAT_PCSRv8p9) && PMPCSCTL.IMP == '1' then
        return PMPCSCTL.EN == '0';
    else
        return boolean IMPLEMENTATION_DEFINED "PCSR is suspended";
```

In the same section, in the pseudocode function CreatePCSample(), the following code segment:

```
CreatePCSample()
    // In a simple sequential execution of the program, CreatePCSample is executed
 each time the PE
    // executes an instruction that can be sampled. An implementation is not
 constrained such that
    // reads of EDPCSRlo return the current values of PC, etc.

    if IsFeatureImplemented(FEAT_PCSRv8p9) && PCSRSuspended() then return;
...
```

is changed to read:

```
CreatePCSample()
    // In a simple sequential execution of the program, CreatePCSample is executed
 each time the PE
    // executes an instruction that can be sampled. An implementation is not
 constrained such that
    // reads of EDPCSRlo return the current values of PC, etc.

    if PCSRSuspended() then return;
...
```

In the same section, in the pseudocode function PMUCaptureEvent(), the following code segment:

```
// PMUCaptureEvent()
// =================
```

```
// If permitted and enabled, generate a PMU snapshot Capture event.
...
    if  IsFeatureImplemented(FEAT_PCSRv8p9) && PMPCSCTL.SS == '1' then
        if pc_sample.valid && !debug_state then
            SetPCSample();
        else
            SetPCSRUnknown();
...
```

is changed to read:

```
// PMUCaptureEvent()
// =================
// If permitted and enabled, generate a PMU snapshot Capture event.
...
    if  IsFeatureImplemented(FEAT_PCSRv8p9) && PMPCSCTL.SS == '1' then
        if pc_sample.valid && !debug_state then
            SetPCSRActive();
            SetPCSample();
        else
            SetPCSRUnknown();
...
```

2.94 D22027

In section **J1.3.3 “shared/functions”**, in the pseudocode function WatchpointRelatedSyndrome(), the code that reads:

```
bits(24) WatchpointRelatedSyndrome(FaultRecord fault, bits(64) vaddress)
    ...
    if fault.maybe_false_match then
        syndrome<16> = '1';
    else
        syndrome<16> = bit IMPLEMENTATION_DEFINED "WPF value on TRUE Watchpoint
 match";

    if IsSVEAccess(fault.accessdesc) || IsSMEAccess(fault.accessdesc) then
    ...
```

is changed to read:

```
bits(24) WatchpointRelatedSyndrome(FaultRecord fault, bits(64) vaddress)
    ...
    if fault.maybe_false_match then
        syndrome<16> = '1';                                          // WPF
    elsif IsFeatureImplemented(FEAT_Debugv8p2) then
        syndrome<16> = bit IMPLEMENTATION_DEFINED "WPF value on TRUE Watchpoint
 match";

    if IsSVEAccess(fault.accessdesc) || IsSMEAccess(fault.accessdesc) then
    ...
```

2.95 R22029

In section **K1.2.13 "Crossing a page boundary with different memory types or Shareability attributes"**, the following text is added:

> Memory accesses from a FEAT_MOPS CPY* or SET* instruction that cross 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:
>
> - Each memory access generated by the instruction uses the memory type and Shareability attribute associated with its own address.
> - The instruction generates an Alignment fault caused by the memory type. For the EL1&0 translation regime, when EL2 is enabled in the current Security state:
>   - If the stage 1 translation generated the mismatch, the resulting exception is taken to EL1.
>   - If the stage 2 translation generated the mismatch, the resulting exception is taken to EL2.
>   - If both stages of translation generate the mismatch, the exception can be taken to either EL1 or EL2.

2.96 D22033

In section **J1.3.1 "shared/debug"**, in the pseudocode function UpdateEDHSR(), the code that reads:

```
UpdateEDHSR(bits(6) reason, FaultRecord fault)
    ...
    if reason == DebugHalt_Watchpoint then
       ...
        syndrome<23:0> = WatchpointRelatedSyndrome(fault, EDWAR);
    else
        syndrome = bits(64) UNKNOWN;
    EDHSR = syndrome;
```

is changed to read:

```
 UpdateEDHSR(bits(6) reason, FaultRecord fault)
    ...
    if reason == DebugHalt_Watchpoint then
       ...
        syndrome<23:0> = WatchpointRelatedSyndrome(fault, EDWAR);
        if IsFeatureImplemented(FEAT_Debugv8p9) then
            if fault.write then syndrome<6> = '1'; // WnR
            if fault.accessdesc.acctype IN {AccessType_DC, AccessType_IC,
 AccessType_AT} then
                syndrome<8> = '1';                 // CM
            if IsFeatureImplemented(FEAT_NV2) && fault.accessdesc.acctype ==
 AccessType_NV2 then
                syndrome<13> = '1';                // VNCR
    else
        syndrome = bits(64) UNKNOWN;
    EDHSR = syndrome;
```

2.97 D22035

In section **J1.1.3 "aarch64/functions"**, in function CheckSMEZT0Enabled(), the code that reads:

```
CheckSMEZT0Enabled()
    // Check if ZA and ZT0 are inactive in PSTATE
    if PSTATE.ZA == '0' then
        SMEAccessTrap(SMEExceptionType_InactiveZA, PSTATE.EL);
...
   // Check if all accesses to ZT0 are disabled in SMCR_EL3
    if HaveEL(EL3) then
        if SMCR_EL3.EZT0 == '0' then
            SMEAccessTrap(SMEExceptionType_InaccessibleZT0, EL3);
```

is changed to read:

```
CheckSMEZT0Enabled()
    if HaveEL(EL3) && SMCR_EL3.EZT0 == '0' && EL3SDDUndefPriority() then
        UNDEFINED;

    // Check if ZA and ZT0 are inactive in PSTATE
    if PSTATE.ZA == '0' then
        SMEAccessTrap(SMEExceptionType_InactiveZA, PSTATE.EL);
...
    // Check if all accesses to ZT0 are disabled in SMCR_EL3
    if HaveEL(EL3) then
        if SMCR_EL3.EZT0 == '0' then
            if EL3SDDUndef() then UNDEFINED;
            SMEAccessTrap(SMEExceptionType_InaccessibleZT0, EL3);
```

2.98 D22041

In section **J1.1.1 "aarch64/debug"**, in the pseudocode function PMUExceptionMasked(), the code that reads:

```
boolean PMUExceptionMasked(bits(2) target_el, bits(2) from_el, bit pm)
    assert IsFeatureImplemented(FEAT_EBEP);
    if Halted() then
        return TRUE;
    ...
```

is changed to read:

```
boolean PMUExceptionMasked(bits(2) target_el, bits(2) from_el, bit pm)
    assert IsFeatureImplemented(FEAT_EBEP);
    if Halted() || Restarting() then
        return TRUE;
    ...
```

2.99 D22047

In section **J1.1.3 "aarch64/functions"**, in the function AArch64.AccessIsTagChecked(), the code that reads:

```
boolean AArch64.AccessIsTagChecked(bits(64) vaddr, AccessDescriptor accdesc)
    ...
    if PSTATE.TCO == '1' then
        return FALSE;
    ...
```

is changed to read:

```
boolean AArch64.AccessIsTagChecked(bits(64) vaddr, AccessDescriptor accdesc)
    ...
    if PSTATE.TCO == '1' then
        return FALSE;

    if (Halted() && EDSCR.MA == '1' &&
           ConstrainUnpredictableBool(Unpredictable_NODTRTAGCHK)) then
        return FALSE;
    ...
```

2.100 D22072

In section **J1.1.3 "aarch64/functions"**, in the pseudocode function AArch64.ESBOperation(), the code that reads:

```
AArch64.ESBOperation()
    bits(2) target_el;
    boolean masked;

    (masked, target_el) = AArch64.PhysicalSErrorTarget();
    intdis = Halted() || ExternalDebugInterruptsDisabled(target_el);
    masked = masked || intdis;
    ...
```

is changed to read:

```
AArch64.ESBOperation()
    bits(2) target_el;
    boolean masked;

    (masked, target_el) = AArch64.PhysicalSErrorTarget();
    if !masked then
        boolean intdis = Halted() || ExternalDebugInterruptsDisabled(target_el);
        masked = intdis;
    ...
```

2.101 D22074

In section **H9.2.6 "DBGDTRRX_EL0, Debug Data Transfer Register, Receive"**, the text that reads:

> Writes to this register:
> - If RXfull is set to 1, set DTRRX to **UNKNOWN**.
> - If RXfull is set to 0, update the value in DTRRX.
>
> After the write, RXfull is set to 1.
>
> Reads of this register:
> - If RXfull is set to 1, return the last value written to DTRRX.
> - If RXfull is set to 0, return an **UNKNOWN** value.
>
> After the read, RXfull remains unchanged.

is changed to read:

> Writes to this register:
> - If RXfull is 0, update the value in DTRRX and set RXfull to 1.
> - If RXfull is 1, the written value is ignored and RXO is set to 1.
>
> Reads of this register return the last value written to DTRRX and do not change RXfull.

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

> Reads of this register:
> - If TXfull is set to 1, return the last value written to DTRTX.
> - If TXfull is set to 0, return an **UNKNOWN** value.
>
> After the read, TXfull is cleared to 0.
>
> Writes to this register:
> - If TXfull is set to 1, set DTRTX to **UNKNOWN**.
> - If TXfull is set to 0, update the value in DTRTX.
>
> After the write, TXfull remains unchanged.

is changed to read:

> Reads of this register:
> - If TXfull is 1, return the value in DTRTX and clear TXfull to 0.
> - If TXfull is 0, return an **UNKNOWN** value and set TXU to 1.

> Writes of this register update the value in DTRTX and do not change TXfull.

In section **D23.3.6 "DBGDTR_EL0, Debug Data Transfer Register, half-duplex"**, the text that reads:

> HighWord, bits [63:32]
>
> Writes to this register set DTRRX to the value in this field and do not change RXfull. Reads of this register:
>
> - If RXfull is set to 1, return the last value written to DTRTX.
> - If RXfull is set to 0, return an **UNKNOWN** value. After the read, RXfull is cleared to 0.
>
> LowWord, bits [31:0]
>
> Writes to this register set DTRTX to the value in this field and set TXfull to 1. Reads of this register:
>
> - If RXfull is set to 1, return the last value written to DTRRX.
> - If RXfull is set to 0, return an **UNKNOWN** value. After the read, RXfull is cleared to 0.

is changed to read:

> HighWord, bits [63:32]
>
> Writes to this register set DTRRX to the value in this field and do not change RXfull. Reads of this register:
>
> - If RXfull is 1, return the last value written to DTRTX.
> - If RXfull is 0, return an **UNKNOWN** value. After the read, RXfull is cleared to 0.
>
> LowWord, bits [31:0]
>
> Writes to this register set DTRTX to the value in this field and set TXfull to 1. Reads of this register:
>
> - If RXfull is 1, return the last value written to DTRRX.
> - If RXfull is 0, return an **UNKNOWN** value. After the read, RXfull is cleared to 0.

In section **G8.3.17 "DBGDTRRXint, Debug Data Transfer Register, Receive"**, the text that reads:

> - If RXfull is set to 1, return the last value written to DTRRX.
> - If RXfull is set to 0, return an **UNKNOWN** value.

is changed to read:

> - If RXfull is 1, return the last value written to DTRRX.
> - If RXfull is 0, return an **UNKNOWN** value.

In section **G8.3.19 "DBGDTRTXint, Debug Data Transfer Register, Transmit"**, the text that reads:

- If TXfull is set to 1, set DTRTX to **UNKNOWN**.
- If TXfull is set to 0, update the value in DTRTX.

is changed to read:

- If TXfull is 1, set DTRTX to **UNKNOWN**.
- If TXfull is 0, update the value in DTRTX.

In section **D23.3.7 "DBGDTRRX_EL0, Debug Data Transfer Register, Receive"**, the text that reads:

- If RXfull is set to 1, return the last value written to DTRRX.
- If RXfull is set to 0, return an **UNKNOWN** value.

is changed to read:

- If RXfull is 1, return the last value written to DTRRX.
- If RXfull is 0, return an **UNKNOWN** value.

In section **D23.3.8 "DBGDTRTX_EL0, Debug Data Transfer Register, Transmit"**, the text that reads:

- If TXfull is set to 1, set DTRRX and DTRTX to **UNKNOWN**.
- If TXfull is set to 0, update the value in DTRTX.

is changed to read:

- If TXfull is 1, set DTRRX and DTRTX to **UNKNOWN**.
- If TXfull is 0, update the value in DTRTX.

2.102 R22077

In section **D23.2.43 "FAR_EL1, Fault Address Register"**, the text that reads:

When FEAT_MOPS is implemented, the value in FAR_EL1 on a synchronous exception from any of the Memory Copy and Memory Set instructions represents the first element that has not been copied or set, and is determined as follows:

- ...
- For a Data Abort generated by a Tag Check failure, the value is the lowest address that failed the Tag Check within the block size of the load or store

is changed to read:

When FEAT_MOPS is implemented, the value in FAR_EL1 on a synchronous exception from any of the Memory Copy and Memory Set instructions represents the first element that has not been copied or set, and is determined as follows:

- ...
- For a Data Abort generated by a Tag Check Fault, the value is any address that caused a Tag Check Fault within the block size of the load or store.

2.103 C22079

In section **B2.17.5 "Load-Exclusive and Store-Exclusive instruction usage restrictions"**, the text that reads:

> The Load-Exclusive and Store-Exclusive instructions are intended to work together as a pair, for example a LDXP/STXP pair or a LDXR/STXR pair.

is changed to read:

> The Load-Exclusive and Store-Exclusive instructions are intended to work together as a pair, for example a LDXP/STXP pair or a LDXR/STXR pair executed as part of a loop that contains only a single Load-Exclusive and Store-Exclusive pair.

In the same section, the text that reads:

> Between the Store-Exclusive returning a failing result and the retry of the corresponding Load-Exclusive:
>
> - There are no stores or PRFM or RPRFM instructions to any address within the Exclusives reservation granule accessed by the Store-Exclusive.
> - There are no loads or preloads to any address within the Exclusives reservation granule accessed by the Store-Exclusive that use a different VA alias to that address.
> - There are no direct or indirect System register writes, address translation instructions, cache or TLB maintenance instructions, exception generating instructions, exception returns, indirect branches, or Branch with Link instructions.
> - All loads and stores are to a block of contiguous virtual memory of not more than 512 bytes in size.

is changed to read:

> Between the Store-Exclusive returning a failing result and the retry of the corresponding Load-Exclusive:
>
> - There are no stores or PRFM instructions to any address within the Exclusives reservation granule accessed by the Store-Exclusive. This includes VA aliases to those addresses.
> - There are no loads or preloads to any address within the Exclusives reservation granule accessed by the Store-Exclusive that use a different VA alias to that address.
> - There are no other Store-Exclusive instructions to any other address.
> - There are no direct or indirect System register writes, address translation instructions, cache or TLB maintenance instructions, exception generating instructions, exception returns, or indirect branches.

- All loads and stores are to a block of contiguous virtual memory of not more than 512 bytes in size.

2.104 D22082

In section **D23.2.65 "HPFAR_EL2, Hypervisor IPA Fault Address Register"**, under the heading 'Purpose', the text that reads:

> Holds the faulting IPA for some aborts on a stage 2 translation taken to EL2.

is changed to read:

> Holds the faulting IPA for some aborts on a stage 2 translation taken to EL2 and GPC exceptions due to a fault on an access for a stage 2 translation.

2.105 C22103

In section **D13.12.4 "Cycle event counting"**, under heading "Multithreaded implementations", the text that reads:

> When the PMU implementation supports multithreading, and the Effective value of PMEVTYPER<n>_EL0.MT bit is 0, the CPU_CYCLES event does not count Processor cycles on which the thread was not active.

is changed to read:

> When the PMU implementation supports multithreading, and the Effective value of PMEVTYPER<n>_EL0.MT bit is 0, and the thread is not in WFE or WFI state, the CPU_CYCLES event only counts Processor cycles when the thread is active.

Under the same heading, the text that reads:

> If the Effective value of PMEVTYPER<n>_EL0.MT bit is 1, the CPU_CYCLES event counts each Processor cycle, and can only count a maximum of one each Processor cycle.

is changed to read:

> It is implementation specific whether CPU_CYCLES event counts Processor cycles when the Effective value of PMEVTYPER<n>_EL0.MT bit is 0, the thread is active, and the thread is in WFE or WFI state.
>
> If the Effective value of PMEVTYPER<n>_EL0.MT bit is 1 and at least one thread is not in WFE or WFI state, then the CPU_CYCLES event counts each Processor cycle, and can only count a maximum of one for each Processor cycle.

2.106 C22106

In section **D23.2.79 "ID_AA64PFR0_EL1, AArch64 Processor Feature Register 0"**, the following text is added to the field descriptions of 'EL3, bits [15:12]', 'EL2, bits [11:8]', and 'EL1, bits [7:4]':

> The value `0b0010` is not permitted in Armv9-A implementations.

2.107 D22114

In section **C7.2.258 "SHRN, SHRN2"**, the text that reads:

> The RSHRN instruction writes the vector to the lower half of the destination register and clears the upper half, while the RSHRN2 instruction writes the vector to the upper half of the destination register without affecting the other bits of the register.

is changed to read:

> The SHRN instruction writes the vector to the lower half of the destination register and clears the upper half, while the SHRN2 instruction writes the vector to the upper half of the destination register without affecting the other bits of the register.

2.108 D22115

In section **C3.2.7 "Load-Acquire/Store-Release"**, the text that reads:

> FEAT_LRCPC2 introduces changes to the alignment requirements of Load-Acquire/Store-Release instructions.

is changed to read:

> FEAT_LSE2 introduces changes to the alignment requirements of Load-Acquire/Store-Release instructions.

In section **B2.13.2 (Alignment of data accesses)**, under heading "Load-Exclusive/ Store-Exclusive and Atomic instructions", the text that reads:

> In this case, the architecture does no define the order of the different transactions of the access defined by the single instructions relative to each other.

is changed to read:

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

2.109 D22138

In section **H5.4.4 "Performance Monitors overflow trigger event"**, the text that reads:

> This is an input trigger event to the CTI, and an output trigger event from the PE, asserted each time the PE asserts a new Performance Monitors counter overflow interrupt request. See Chapter D12 The Performance Monitors Extension.
>
> If the CTI supports multicycle trigger events, then the trigger event remains asserted until the overflow is cleared by a write to PMOVSCLR_EL0. Otherwise, the trigger event is asserted when the value of PMOVSCLR_EL0 changes from zero to a nonzero value.

is changed to read:

> This is an input trigger event to the CTI, and an output trigger event from the PE, asserted each time the PE asserts a new Performance Monitors counter overflow interrupt request. See Chapter D12 The Performance Monitors Extension.
>
> If the CTI supports multicycle trigger events, then the trigger event remains asserted while the Performance Monitors counter overflow interrupt request is asserted. Otherwise, the trigger event is asserted when the Performance Monitors counter overflow interrupt request changes from being deasserted to being asserted.

2.110 D22139

In section **D8.5.2.2 "Implications of enabling the dirty state management mechanism"**, the following rule:

> R_{RKMHW}
>
> For the following instructions that require write permission, if the address specified in the instruction is translated by a writeable-clean descriptor, then the descriptor is considered to grant write access:
>
> - Data cache invalidation instruction, DC IVAC.
> - Address translation instructions, AT S12E0W, AT S12E1W, AT S1E0W, AT S1E1W, AT S1E2W, AT S1E3W.

is changed to read:

> R_{RKMHW}
>
> For the following instructions that require write permission, if the address specified in the instruction is translated by a writeable-clean descriptor, then the descriptor is considered to grant write access and the hardware does not perform an update of 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.

2.111 D22142

In section **D23.1.2 "General behavior of accesses to the AArch64 System registers"**, under heading 'Synchronization requirements for AArch64 System registers', the text that reads:

> Since an exception is context synchronizing, registers such as the Exception Syndrome registers that are indirectly written as part of exception entry do not require additional synchronization.

is changed to read:

> Registers such as the Exception Syndrome registers that are indirectly written as part of exception entry do not require additional synchronization. This behavior is independent of the value of SCTLR_ELx.EIS, and the list of affected registers is described in the definition of that field.

2.112 E22161

In section **J1.3.3 "shared/functions"**, in the pseudocode function DecodeSW(), the code that reads:

```
// DecodeSW()
// ==========
// Decode input value into setnum, waynum and level for SW instructions.

(integer, integer, integer) DecodeSW(bits(64) regval, CacheType cachetype)
    level = UInt(regval[3:1]);
    (setnum, waynum, linesize) = GetCacheInfo(level, cachetype);
    return (setnum, waynum, level);
```

is changed to read:

```
// DecodeSW()
// ==========
// Decode input value into setnum, waynum and level for SW instructions.

(integer, integer, integer) DecodeSW(bits(64) regval, CacheType cachetype)
    level = UInt(regval<3:1>);

    (numsets, associativity, linesize) = GetCacheInfo(level, cachetype);

    // For the given level and cachetype, get the number of sets, associativity and
    // cache line size in terms of actual bytes.
    constant integer waybits  = CeilLog2(associativity);
    constant integer setbits  = CeilLog2(numsets);
    constant integer linebits = Log2(linesize);

    constant integer waynum = if associativity == 1 then 0 else UInt(regval<31:32-
waybits>);
```

```
    constant integer setnum = if numsets == 1 then 0 else UInt(regval<linebits
+setbits-1:linebits>);

    return (setnum, waynum, level);
```

2.113 D22162

In section **D8.15.3.1 "Global and process-specific translation table entries"**, the following rule is added under rule R_{JYHZR}:

R_{CJCFJ}

If all of the following apply, then a stage 1 translation is treated as non-global, meaning the Effective value of nG is 1, regardless of the actual value of the Block descriptor or Page descriptor nG bit:

- The translation is for the EL1&0 translation regime.
- Stage 1 and stage 2 translation are enabled.
- Protection is enabled.
- The address is translated via TTBRn_EL1 and TCR2_EL1.FNGNAn is 1.
- The final stage 1 translation using the descriptor does not have the Assured Translation property.

In section **D23.2.172 "TCR2_EL1, Extended Translation Control Register (EL1)"**, the following new fields are added:

FNGNA1, bit [21]

When FEAT_THE is implemented, Force non-global for unassured translations using TTBR1_EL1.

FNGNA0, bit [20]

When FEAT_THE is implemented, Force non-global for unassured translations using TTBR0_EL1.

2.114 D22179

In section **D23.2.64 "HFGWTR_EL2, Hypervisor Fine-Grained Write Trap Register"**, in the field description of 'VBAR_EL1, bit [38]', the text that reads:

Trap MSR or MSRR writes of VBAR_EL1 at EL1 using AArch64 to EL2.

VBAR_EL1	Meaning
0b0	MSR or MSRR writes of VBAR_EL1 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MSR or MSRR writes of VBAR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.

is changed to read:

> Trap MSR writes of VBAR_EL1 at EL1 using AArch64 to EL2.

VBAR_EL1	Meaning
0b0	MSR writes of VBAR_EL1 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MSR writes of VBAR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.

In the same section, in the field description of 'nRCWMASK_EL1, bit [56]', the text that reads:

nRCWMASK_EL1	Meaning
0b0	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MSR writes of RCWMASK_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.
0b1	MSR writes of RCWMASK_EL1 are not trapped by this mechanism.

is changed to read:

nRCWMASK_EL1	Meaning
0b0	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then unless the write generates a higher priority exception: • MSR writes of RCWMASK_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18. • MSRR writes of RCWMASK_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x14.
0b1	MSR or MSRR writes of RCWMASK_EL1 are not trapped by this mechanism.

The equivalent changes are made in the following field descriptions:

- 'TTBR1_EL1, bit [37]'.
- 'TTBR0_EL1, bit [36]'.
- 'PAR_EL1, bit [27]'.

2.115 D22182

In section **D23.2.56 "HDFGRTR_EL2, Hypervisor Debug Fine-Grained Read Trap Register"**, in the field description of 'TRCAUTHSTATUS, bit [34]', the text that reads:

> In an Armv9 implementation, trap MRS reads of TRCAUTHSTATUS at EL1 using AArch64 to EL2.

is changed to read:

> In an Armv9 implementation, trap MRS reads of TRCAUTHSTATUS at EL1 using AArch64 to EL2.
> In an Armv8 implementation, trap MRS reads of ETM TRCAUTHSTATUS at EL1 using AArch64 to EL2.

2.116 D22183

In section **D23.2.56 "HDFGRTR_EL2, Hypervisor Debug Fine-Grained Read Trap Register"**, in the field description of 'PMCCNTR_EL0, bit [15]', the following text is removed:

> PMCCNTR_EL0 is indirectly accessed when PMCR_EL0.C is set to `0b1`. Setting this field to 1 has no effect on accesses to PMCCNTR_EL0 using PMCR_EL0.

In the field description of 'PMEVCNTRn_EL0, bit [12]', the following text is removed:

> PMEVCNTR<n>_EL0 is indirectly accessed when PMCR_EL0.P is set to `0b1`. Setting this field to 1 has no effect on accesses to PMEVCNTR<n>_EL0 using PMCR_EL0.

2.117 C22186

In section **C6.2.43 "CAS, CASA, CASAL, CASL"**, the text that reads:

> Compare and Swap word or doubleword in memory reads a 64-bit doubleword from memory, and compares it against the value held in a first register. If the comparison is equal, the value in a second register is written to memory.

is changed to read:

> Compare and Swap word or doubleword in memory reads a 64-bit doubleword from memory, and compares it against the value held in a first register. If the comparison is equal, the value in a second register is written to memory. If the comparison is not equal, the architecture permits writing the value read from the location to memory.

The equivalent changes are made in the following sections:

- **C6.2.44 "CASB, CASAB, CASALB, CASLB"**.
- **C6.2.45 "CASH, CASAH, CASALH, CASLH"**.
- **C6.2.46 "CASP, CASPA, CASPAL, CASPL"**.

In section **C6.2.275 "RCWCAS, RCWCASA, RCWCASL, RCWCASAL"**, the text that reads:

> Read check write compare and swap doubleword in memory reads a 64-bit doubleword from memory, and compares it against the value held in a register. If the comparison is equal, the value in a second register is conditionally written to memory. Storing back to memory is conditional on RCW Checks. If the write is performed, the read and the write occur atomically such that no other modification of the memory location can take place between the read and the write. This instruction updates the condition flags based on the result of the update of memory.

is changed to read:

> Read check write compare and swap doubleword in memory reads a 64-bit doubleword from memory, and compares it against the value held in a register. If the comparison is equal, the value in a second register is conditionally written to memory. Storing back to memory is conditional on RCW Checks. If the compare fails, or the RCW Checks fail, the architecture permits writing the value read from the location to memory. If the write is performed, the read and the write occur atomically such that no other modification of the memory location can take place between the read and the write. This instruction updates the condition flags based on the result of the update of memory.

The equivalent change is made in the section **C6.2.276 "RCWCASP, RCWCASPA, RCWCASPL, RCWCASPAL"**.

In section **C6.2.279 "RCWSCAS, RCWSCASA, RCWSCASL, RCWSCASAL"**, the text that reads:

> Read check write compare and swap doubleword in memory reads a 64-bit doubleword from memory, and compares it against the value held in a register. If the comparison is equal, the value in a second register is conditionally written to memory. Storing back to memory is conditional on RCW Checks. If the write is performed, the read and the write occur atomically such that no other modification of the memory location can take place between the read and the write. This instruction updates the condition flags based on the result of the update of memory.

is changed to read:

> Read check write compare and swap doubleword in memory reads a 64-bit doubleword from memory, and compares it against the value held in a register. If the comparison is equal, the value in a second register is conditionally written to memory. Storing back to memory is conditional on RCW Checks. If the compare fails, or the RCW Checks fail, the architecture permits writing the value read from the location to memory. If the write is performed, the read and the write occur atomically such that no other modification of the memory location can take place between the read and the write. This instruction updates the condition flags based on the result of the update of memory.

The equivalent change is made in the section **C6.2.280 "RCWSCASP, RCWSCASPA, RCWSCASPL, RCWSCASPAL"**.

2.118 D22188

In section **D13.12.3.1 "Common architectural events"**, in the description of event "`0x400D`, PMU_OVFS, PMU overflow, counters accessible to EL1 and EL0", the following bullet point is added:

> - FEAT_PMUv3_ICNTR is implemented, an instruction is counted by PMICNTR_EL0, PMINTENSET_EL1.F0 is 1, and counting the instruction causes unsigned overflow of PMICNTR_EL0[63:0].

2.119 R22189

In section **B2.3.7 "Ordering relations"**, in the definition of Hardware-required-ordered-before, the text that reads:

> An effect E_1 is Hardware-required-ordered-before an effect E_2 if one of the following applies:
>
> ...
>
> - FEAT_ETS2 is implemented, E_1 is an Explicit Memory effect, E_2 is an Implicit TTD Memory Read effect and all of the following apply:
>   - E_1 is in program-order-before a TLBUncacheable Fault effect E_3.
>   - E_2 is in Translation-intrinsically-before E_3.
>
>   ...

is changed to read:

> An effect E_1 is Hardware-required-ordered-before an effect E_2 if one of the following applies:
>
> ...
>
> - FEAT_ETS2 is implemented, E_1 is an Explicit Memory effect, E_2 is an Implicit TTD Memory Read effect and all of the following apply:
>   - It is not the case that E_1 is generated by a SIMD&FP, SVE, or SME instruction.
>   - E_1 is in program-order-before a TLBUncacheable Fault effect E_3.
>   - E_2 is in Translation-intrinsically-before E_3.
>
>   ...

2.120 C22191

In section **D3.2.1 "Controls to prohibit trace at Exception levels"**, the text that reads:

> If SelfHostedTraceEnabled() == TRUE, no events are exported to the trace unit when tracing is prohibited. If SelfHostedTraceEnabled() == FALSE, no events are exported to the trace unit when the PE is in Secure state and counting in Secure state is prohibited.

is changed to read:

> If FEAT_ETE is not implemented, when SelfHostedTraceEnabled() == TRUE, no events are exported to the trace unit when tracing is prohibited. If FEAT_ETE is not implemented, when SelfHostedTraceEnabled() == FALSE, no events are exported to the trace unit when the PE is in Secure state and counting in Secure state is prohibited. If FEAT_ETE is implemented, see D4.6.13 "External inputs" for the rules governing export of events to the trace unit.

2.121 C22200

In section **C6.2.209 "LDTR"**, the text that reads:

> Memory accesses made by the instruction behave as if the instruction was executed at EL0 if the Effective value of PSTATE.UAO is 0 and either:
>
> - The instruction is executed at EL1.
> - The instruction is executed at EL2 when the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}.

is changed to read:

> Explicit Memory effects produced by the instruction behave as if the instruction was executed at EL0 if the Effective value of PSTATE.UAO is 0 and either:
>
> - The instruction is executed at EL1.
> - The instruction is executed at EL2 when the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}.

The equivalent changes are made in the following sections:

- **C6.2.80 "CPYFPT, CPYFMT, CPYFET"**.
- **C6.2.81 "CPYFPTN, CPYFMTN, CPYFETN"**.
- **C6.2.82 "CPYFPTRN, CPYFMTRN, CPYFETRN"**.
- **C6.2.83 "CPYFPTWN, CPYFMTWN, CPYFETWN"**.
- **C6.2.96 "CPYPT, CPYMT, CPYET"**.
- **C6.2.97 "CPYPTN, CPYMTN, CPYETN"**.
- **C6.2.98 "CPYPTRN, CPYMTRN, CPYETRN"**.
- **C6.2.99 "CPYPTWN, CPYMTWN, CPYETWN"**.
- **C6.2.210 "LDTRB"**.
- **C6.2.211 "LDTRH"**.
- **C6.2.212 "LDTRSB"**.
- **C6.2.213 "LDTRSH"**.
- **C6.2.214 "LDTRSW"**.
- **C6.2.380 "STTR"**.
- **C6.2.381 "STTRB"**.
- **C6.2.382 "STTRH"**.
- **C6.2.140 "GCSSTTR"**.

In section **C6.2.76 "CPYFPRT, CPYFMRT, CPYFERT"**, the following text is added to the instruction description:

> Explicit Memory Read effects produced by the instruction behave as if the instruction was executed at EL0 if the Effective value of PSTATE.UAO is 0 and either:

- The instruction is executed at EL1.
- The instruction is executed at EL2 when the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}.

The same text is added to the instruction descriptions of the following Memory Copy (CPY*) instructions:

- **C6.2.77 CPYFPRTN, CPYFMRTN, CPYFERTN**.
- **C6.2.78 CPYFPRTRN, CPYFMRTRN, CPYFERTRN**.
- **C6.2.79 CPYFPRTWN, CPYFMRTWN, CPYFERTWN**.
- **C6.2.92 CPYPRT, CPYMRT, CPYERT**.
- **C6.2.93 CPYPRTN, CPYMRTN, CPYERTN**.
- **C6.2.94 CPYPRTRN, CPYMRTRN, CPYERTRN**.
- **C6.2.95 CPYPRTWN, CPYMRTWN, CPYERTWN**.

In section **C6.2.85 "CPYFPWT, CPYFMWT, CPYFEWT"**, the following text is added to the instruction description:

Explicit Memory Write effects produced by the instruction behave as if the instruction was executed at EL0 if the Effective value of PSTATE.UAO is 0 and either:

- The instruction is executed at EL1.
- The instruction is executed at EL2 when the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}.

The equivalent changes are made to the following sections:

- **C6.2.86 "CPYFPWTN, CPYFMWTN, CPYFEWTN"**.
- **C6.2.87 "CPYFPWTRN, CPYFMWTRN, CPYFEWTRN"**.
- **C6.2.88 "CPYFPWTWN, CPYFMWTWN, CPYFEWTWN"**.
- **C6.2.101 "CPYPWT, CPYMWT, CPYEWT"**.
- **C6.2.102 "CPYPWTN, CPYMWTN, CPYEWTN"**.
- **C6.2.103 "CPYPWTRN, CPYMWTRN, CPYEWTRN"**.
- **C6.2.104 "CPYPWTWN, CPYMWTWN, CPYEWTWN"**.
- **C6.2.312 "SETGPT, SETGMT, SETGET"**.
- **C6.2.313 "SETGPTN, SETGMTN, SETGETN"**.
- **C6.2.316 "SETPT, SETMT, SETET"**.
- **C6.2.317 "SETPTN, SETMTN, SETETN"**.

2.122 C22201

In section **B2.9 "Restrictions on the effects of speculation"**, under the heading 'Restrictions on the effects of speculation from Armv8.5', the text that reads:

> In this definition, the hardware-defined context is determined by:
>
> - The Exception level.
> - The Security state.
> - When executing at EL1, if EL2 is implemented and enabled in the current Security state, the VMID.
> - When executing at EL0, whether the EL1&0 or the EL2&0 translation regime is in use.
> - When executing at EL0 and using the EL1&0 translation regime, the address space identifier (ASID) and, if EL2 is implemented and enabled in the current Security state, the VMID.
> - When executing at EL0 and using the EL2&0 translation regime, the ASID.
>
> If FEAT_CSV2_2 is implemented, then SCXTNUM_ELx is also part of the hardware-defined context.

is changed to read:

> In this definition, the hardware-defined context is determined by:
>
> - The Exception level.
> - The Security state.
> - When executing at EL1, if EL2 is implemented and enabled in the current Security state, the VMID.
> - When executing at EL0, whether the EL1&0 or the EL2&0 translation regime is in use.
> - When executing at EL0 and using the EL1&0 translation regime, the address space identifier (ASID) and, if EL2 is implemented and enabled in the current Security state, the VMID.
> - When executing at EL0 and using the EL2&0 translation regime, the ASID.
>
> If FEAT_CSV2_2 is implemented, then SCXTNUM_ELx is also part of the hardware-defined context. For unprivileged accesses generated by load or store unprivileged instructions and unprivileged accesses generated by the Memory Copy, Memory Set, or GCSSTTR instructions, the hardware-defined context associated with the data addresses and data values of the unprivileged accesses is determined as if the access was generated by an instruction executed at EL0.

2.123 D22203

In section **J1.3.3 “shared/functions”**, in the pseudocode function EffectiveHCR_EL2_NVx(), the code that reads:

```
bits(3) EffectiveHCR_EL2_NVx()
    if !EL2Enabled() || !IsFeatureImplemented(FEAT_NV) || HCR_EL2.NV == '0' then
        return '000';

    bit nv1 = HCR_EL2.NV1;
    if (!IsFeatureImplemented(FEAT_E2H0) &&
          boolean IMPLEMENTATION_DEFINED "HCR_EL2.NV1 is implemented as RAZ") then
        nv1 = '0';

    if !IsFeatureImplemented(FEAT_NV2) then
        return '0' : nv1 : '1';

    ...
```

is changed to read:

```
bits(3) EffectiveHCR_EL2_NVx()
    if !EL2Enabled() || !IsFeatureImplemented(FEAT_NV) then
        return '000';

    bit nv1 = HCR_EL2.NV1;
    if (!IsFeatureImplemented(FEAT_E2H0) &&
          boolean IMPLEMENTATION_DEFINED "HCR_EL2.NV1 is implemented as RAZ") then
        nv1 = '0';

     if HCR_EL2.NV == '0' then
        if nv1 == '1' then
            case ConstrainUnpredictable(Unpredictable_NVNV1) of
                when Constraint_NVNV1_00 return '000';
                when Constraint_NVNV1_01 return '010';
                when Constraint_NVNV1_11 return '011';
        else
            return '000';

     if !IsFeatureImplemented(FEAT_NV2) then
        return '0' : nv1 : '1';

    ...
```

2.124 D22206

In section **D23.5.22 “PMSELR_EL0, Performance Monitors Event Counter Selection Register”**, under the heading “Purpose”, the text that reads:

> Selects the current event counter PMEVCNTR<n>_EL1 or the cycle counter PMCCNTR.

is changed to read:

> Selects the current event counter PMEVCNTR<n>_EL0 or the cycle counter PMCCNTR_EL0.

2.125 C22213

In section **B2.3.6 "Dependency definitions"** the text that reads:

> There is an Address dependency from an effect E_1 to an effect E_2 if all of the following apply:
> - E_1 is an Explicit Memory effect
> - There is a Basic dependency from E_1 to an effect E_3.
> - E_3 affects the address of the Location accessed by E_2.
> - There is an Intrinsic Data Dependency from E_3 to E_2.
> - E_2 is a Memory effect.

is changed to read:

> There is an Address dependency from an effect E_1 to an effect E_2 if all of the following apply:
> - E_1 is an Explicit Memory effect.
> - There is a Basic dependency from E_1 to an effect E_3.
> - E_3 affects the address of the Location accessed by or the invalidation scope of E_2.
> - There is an Intrinsic Data Dependency from E_3 to E_2.
> - One of the following applies:
>   - E_2 is a Memory effect.
>   - E_2 is a TLBI effect.
>   - E_2 is a DC CVAU effect.
>   - E_2 is an IC IVAU effect.

In the same section, the text that reads:

> There is a Pick Address dependency from an effect E_1 to an effect E_2 if all of the following apply:
> - There is a Pick Basic dependency from E_1 to an effect E_3.
> - E_3 affects the address of the Location accessed by E_2.
> - There exists a chain of Intrinsic data dependency from E_3 to E_2.
> - E_2 is a Memory effect.

is changed to read:

> There is a Pick Address dependency from an effect E_1 to an effect E_2 if all of the following apply:
> - There is a Pick Basic dependency from E_1 to an effect E_3.
> - E_3 affects the address of the Location accessed by or the invalidation scope of E_2.
> - There exists a chain of Intrinsic data dependency from E_3 to E_2.
> - One of the following applies:

- E_2 is a Memory effect.
- E_2 is a TLBI effect.
- E_2 is a DC CVAU effect.
- E_2 is an IC IVAU effect.

In section **B2.3.7 "Ordering relations"** the definition of Locally-hardware-required-ordered-before is extended to include:

An effect E_1 is Locally-hardware-required-ordered-before an effect E_2 if one of the following applies:

...

- All of the following apply:
  - E_1 is an Explicit Memory Read effect.
  - There is a Control dependency from E_1 to E_2.
  - One of the following applies:
    - E_2 is a TLBI effect.
    - E_2 is a DC.CVAU effect
    - E_2 is an IC effect
- All of the following apply:
  - E_1 is an Explicit Memory Read effect.
  - There is a Pick Control dependency from E_1 to E_2.
  - One of the following applies:
    - E_2 is a TLBI effect.
    - E_2 is a DC.CVAU effect
    - E_2 is an IC effect
- All of the following apply:
  - E_1 is an Explicit Memory Read effect.
  - There is an Address dependency from E_1 to E_2.
  - One of the following applies:
    - E_2 is a TLBI effect.
    - E_2 is a DC.CVAU effect
    - E_2 is an IC effect
- All of the following apply:
  - E_1 is an Explicit Memory Read effect.
  - There is a Pick Address dependency from E_1 to E_2.
  - One of the following applies:

- E_2 is a TLBI effect.
- E_2 is a DC.CVAU effect
- E_2 is an IC effect

...

2.126 E22215

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

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

is changed to read:

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

2.127 R22228

In section **D23.8.11 "BRBTS_EL1, Branch Record Buffer Timestamp Register"**, in the field description of 'TS, bits [63:0]', the text that reads:

> The reset behavior of this field is:
>
> - On a Warm reset, this field resets to 0.

is changed to read:

> The reset behavior of this field is:
>
> - On a Cold reset, when FEAT_BRBEv1p1 is implemented, it is **IMPLEMENTATION DEFINED** whether this field resets to an architecturally **UNKNOWN** value.
> - On a Warm reset, when FEAT_BRBEv1p1 is implemented, it is **IMPLEMENTATION DEFINED** whether this field resets to 0 or is preserved.
> - On a Warm reset, when FEAT_BRBEv1p1 is not implemented, this field resets to 0.
>
> When FEAT_BRBEv1p1 is implemented, Arm recommends that this field is preserved on a Warm reset.

2.128 D22232

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

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

is changed to read:

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

2.129 D22241

In section **C6.2.40 "BRB"**, the text that reads:

> BRB <brb_op>{, <Xt>}

is changed to read:

> BRB <brb_op>

2.130 D22243

In section **D23.10.2 "CNTHCTL_EL2, Counter-timer Hypervisor Control Register"**, in subsection 'When the Effective value of HCR_EL2.E2H is 1:' and subsection 'Otherwise:', the field description of 'ECV, bit[12]', the text that reads:

ECV	Meaning
0b0	Enhanced Counter Virtualization functionality is disabled.
0b1	EL0 and EL1 accesses to the specified registers are trapped to EL2.

is changed to read:

ECV	Meaning
0b0	Enhanced Counter Virtualization functionality is disabled.
0b1	Enhanced Counter Virtualization functionality is enabled.

2.131 C22244

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

> FEAT_ECV enhances the Generic Timer architecture.
>
> When executing in AArch64 state or AArch32 state, FEAT_ECV provides:
>
> - Self-synchronizing views of the virtual and physical timers in AArch64 and AArch32 state.
> - The ability to scale the generation of the event stream.
>
> When EL2 is using AArch64 state, FEAT_ECV provides:
>
> - An optional offset between the EL1 or EL0 view of physical time, and the EL2 or EL3 view of physical time.
> - Traps configurable in CNTHCTL_EL2 that trap EL0 and EL1 access to the virtual counter or timer registers, and accesses to the physical timer registers when they are accessed using an EL02 descriptor.
>
> The optional offset to views of physical time, and the configurable traps in CNTHCTL_EL2, both apply to EL1 and EL0 whether EL1 and EL0 are in AArch64 state or AArch32 state.
>
> This feature is supported in both AArch64 and AArch32 states.
>
> FEAT_ECV is OPTIONAL from Armv8.5.
>
> FEAT_ECV is mandatory from Armv8.6.
>
> The following field identifies the presence of FEAT_ECV:

- ID_AA64MMFR0_EL1.ECV.

is changed to read:

FEAT_ECV enhances the Generic Timer architecture. FEAT_ECV provides:

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

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

This feature is supported in both AArch64 and AArch32 states.

FEAT_ECV is OPTIONAL from Armv8.5.

FEAT_ECV is mandatory from Armv8.6.

The following field identifies the presence of FEAT_ECV:

- ID_AA64MMFR0_EL1.ECV.

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

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

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

This feature is supported in both AArch64 and AArch32 states.

FEAT_ECV_POFF is OPTIONAL from Armv8.5.

If FEAT_ECV_POFF is implemented, then FEAT_ECV is implemented.

If FEAT_ECV_POFF is implemented, then FEAT_AA64EL2 is implemented.

If FEAT_RME is implemented, then FEAT_ECV_POFF is implemented.

The following field identifies the presence of FEAT_ECV_POFF:

- ID_AA64MMFR0_EL1.ECV.

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

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

is changed to read:

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

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

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

is changed to read:

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

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

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

is changed to read:

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

In the same field description, the text that reads:

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

is changed to read:

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

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

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

is changed to read:

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

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

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

is changed to read:

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

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

> When FEAT_ECV is implemented:

is changed to read:

> When FEAT_ECV_POFF is implemented:

2.132 C22247

In section **D8.13 "Translation Lookaside Buffers"**, the following rule is added:

> R_{X00001}
>
> For any System register field that is described as being permitted to be cached in a TLB, if that field takes an Effective value as a result of a System register control field at a higher Exception level then the Effective value is also permitted to be cached in a TLB.

2.133 C22259

In section **H2.4.2.1.1 "A64 instructions that are changed in Debug state"**, the text that reads:

> The following A64 instructions are defined in Debug state, but are **UNDEFINED** in Non-debug state:
>
> - DCPS.
>
> Note:
>
> DCPS can be **UNDEFINED** in certain conditions in Debug state. See DCPS<n> on page H2- 11107.
>
> - DRPS.
> - MRS (DLR_EL0), MRS (DSPSR_EL0), MSR (DLR_EL0), MSR (DSPSR_EL0)
> - When FEAT_SVE is implemented, CMPNE (immediate) with byte element size.

> Note:
>
> In Debug state, CMPNE (immediate) with byte element size sets DLR_EL0 and DSPSR_EL0 to **UNKNOWN** values. However, the instruction is unchanged with respect to the SVE vector and SVE predicate source and destination registers.
>
> - When FEAT_SME2 is implemented, MOVT (table to scalar) and MOVT (scalar to table).

is changed to read:

> The following A64 instructions are defined in Debug state, but are **UNDEFINED** in Non-debug state:
>
> - DRPS.
> - MRS (DLR_EL0), MRS (DSPSR_EL0), MSR (DLR_EL0), MSR (DSPSR_EL0)
> - DCPS.
>
> Note:
>
> DCPS can be **UNDEFINED** in certain conditions in Debug state. See DCPS<n> on page H2- 11107.
>
> - When FEAT_SME2 is implemented, MOVT (table to scalar) and MOVT (scalar to table).
>
> The following instruction is defined in both Non-Debug and Debug states and has changed behavior in Debug state
>
> - When FEAT_SVE is implemented, CMPNE (immediate) with byte element size.
>
> Note:
>
> In Debug state, CMPNE (immediate) with byte element size sets DLR_EL0 and DSPSR_EL0 to **UNKNOWN** values. However, the instruction is unchanged with respect to the SVE vector and SVE predicate source and destination registers.

2.134 D22261

In section **I6.3.26 "PMDEVID, Performance Monitors Device ID register"**, under the heading 'Configurations', the text that reads:

> This register is present only when FEAT_PMUv3_EXT32 is implemented. Otherwise, direct accesses to PMDEVID are **RES0**.

is changed to read:

> This register is present only when FEAT_PMUv3_EXT is implemented. Otherwise, direct accesses to PMDEVID are **RES0**.

The equivalent changes are made in the following sections:

- **I6.3.27 "PMDEVTYPE, Performance Monitors Device Type register"**.

- **I6.3.39 “PMITCTRL, Performance Monitors Integration mode Control register”**.
- **I6.3.40 “PMLAR, Performance Monitors Lock Access Register”**.
- **I6.3.41 “PMLSR, Performance Monitors Lock Status Register”**.

2.135 D22263

In section **I6.3.5 “PMCCNTSVR_EL1, Performance Monitors Cycle Count Saved Value Register”**, the text that reads:

> - When !AllowExternalPMSSAccess(), accesses to this register generate an error response.
> - Otherwise, accesses to this register are RO.

is changed to read:

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

Equivalent changes are made in **I6.3.29 “PMEVCNTSVR<n>_EL1, Performance Monitors Event Count Saved Value Register <0-30>“**.

In section **I6.3.34 “PMICNTSVR_EL1, Performance Monitors Instruction Count Saved Value Register”**, the following text is added:

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

In section **I6.3.53 “PMSSCR_EL1, Performance Monitors Snapshot Status and Capture Register”**, the text that reads:

> - When !AllowExternalPMSSAccess(), accesses to this register generate an error response.
> - Otherwise, accesses to this register are RO.

is changed to read:

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

2.136 C22267

In section **D8.14.5 "MMU fault prioritization from a single address translation stage"**, the rule that reads:

> R_{MJLPH}
>
> If all the following apply, the prioritization between the stage 2 Permission fault on the stage 1 translation table walk and the stage 1 abort caused by the stage 1 Block descriptor or Page descriptor is **IMPLEMENTATION DEFINED**:
>
> - Hardware management of the stage 1 access flag or dirty state is enabled.
> - The access flag or dirty state in a stage 1 Block descriptor or Page descriptor needs to be updated.
> - For the location of the stage 1 Block descriptor or Page descriptor, the stage 2 translation has read permission but not write permission.
> - The stage 1 Block descriptor or Page descriptor generates an abort, which can be due to an Address size fault, an Alignment fault caused by memory type, or a Permission fault.

is changed to read:

> R_{MJLPH}
>
> If all the following apply, the prioritization between the stage 2 fault on the stage 1 translation table walk and the stage 1 fault caused by the stage 1 Block descriptor or Page descriptor is IMPLEMENTATION DEFINED:
>
> - Hardware management of the stage 1 Access flag or dirty state is enabled.
> - The Access flag or dirty state in a stage 1 Block descriptor or Page descriptor needs to be updated.
> - For the location of the stage 1 Block descriptor or Page descriptor, the stage 2 translation generates one of the following:
>   - A Permission fault because the descriptor grants read permission but not write permission.
>   - An unsupported atomic hardware update fault.
> - The stage 1 Block descriptor or Page descriptor generates an MMU fault, which can be due to an Address size fault, an Alignment fault caused by memory type, or a Permission fault.

2.137 C22268

In section **D20.20 "Instruction, data, translation table walk, and other accesses"**, the text that reads:

> When a PE generates a memory-system request for an instruction access, the PARTID_I field of an MPAMn_ELx register is used, as shown in Table D20-10.

is changed to read:

> When a PE generates a memory-system request for an instruction fetch, the PARTID_I field of an MPAMn_ELx register is used, as shown in Table D20-10.

In the same section, the text that reads:

> When a PE generates a memory-system request for a data access, the PARTID_D field of an MPAMn_ELx register is used, as shown in Table D20-10.

is changed to read:

> When a PE generates a memory-system request for a data access, the PARTID_D field of an MPAMn_ELx register is used, as shown in Table D20-10. If the memory-system request for data access is due to the execution of an SME (or SVE streaming mode) load or store instruction, the PARTID_D field of MPAMSM_EL1 register is used instead.

In the same section, the text that reads:

> PARTID_D and PARTID_I fields of an MPAMn_ELx register can be set by software to the same or different PARTIDs, with the following requirements:

is changed to read:

> PARTID_D and PARTID_I fields of an MPAMn_ELx or MPAMSM_EL1 register can be set by software to the same or different PARTIDs, with the following requirements:

2.138 D22283

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

```
elsif (accdesc.exclusive || accdesc.atomicop ||
        accdesc.acqsc || accdesc.acqpc || accdesc.relsc) then
    if !aligned && !ConstrainUnpredictableBool(Unpredictable_MISALIGNEDATOMIC) then
        memaddrdesc.fault = AlignmentFault(accdesc);
        return (value, memaddrdesc, memstatus);
    else
        atomic = TRUE;
```

is changed to read:

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

The equivalent change is made in **J1.1.3 “aarch64/functions”**, in the pseudocode function MemSingleWrite().

In section **J1.1.4 "aarch64/translation"**, the following pseudocode is added to the pseudocode function S1HasAlignmentFault():

```
if accdesc.exclusive || accdesc.atomicop || accdesc.acqsc || accdesc.acqpc ||
 accdesc.relsc then
    if (!aligned && !(IsWBShareable(memattrs) && AArch64.S1DCacheEnabled(regime)) &&
        ConstrainUnpredictableBool(Unpredictable_LSE2_ALIGNMENT_FAULT)) then
    return TRUE;
```

In section **J1.1.4 "aarch64/translation"**, the following pseudocode is added to the pseudocode function S2HasAlignmentFault():

```
if accdesc.exclusive || accdesc.atomicop || accdesc.acqsc || accdesc.acqpc ||
 accdesc.relsc then
    if (!aligned && !(IsWBShareable(memattrs) && S2DCacheEnabled()) &&
        ConstrainUnpredictableBool(Unpredictable_LSE2_ALIGNMENT_FAULT)) then
    return TRUE;
```

2.139 D22294

In section **C3.2.14 "Memory Copy and Memory Set instructions"**, the text that reads:

> The CPY* instructions are guaranteed to make forward progress if none of the following four leaf level translation table entries fault:
>
> - The source leaf level translation table entry held in Xs.
> - The next leaf level translation table entry, as determined by the copy direction, adjacent to the source leaf level translation table entry held in Xs.
> - The destination leaf level translation table entry held in Xd.
> - The next leaf level translation table entry, as determined by the copy direction, adjacent to the destination leaf level translation table entry held in Xd.

is changed to read:

> The CPY* instructions are guaranteed to make forward progress if none of the following four leaf level translation table entries fault:
>
> - The source leaf level translation table entry used to translate the address held in Xs.
> - The next leaf level translation table entry, as determined by the copy direction, adjacent to the source leaf level translation table entry used to translate the address held in Xs.
> - The destination leaf level translation table entry used to translate the address held in Xd.
> - The next leaf level translation table entry, as determined by the copy direction, adjacent to the destination leaf level translation table entry used to translate the address held in Xd.

The note which follows, that reads:

> The forward progress described in this section can be achieved by ensuring that, when a memory management fault is encountered, all bytes leading up to the fault have been operated on.

is changed to read:

> A PE can ensure that the forward progress described in this section can be achieved by ensuring that, when a memory management fault is encountered, all bytes leading up to the fault have been operated on and the operand registers have been updated.
>
> A PE should not report MMU faults other than for the four leaf translation tables entries described in this section.

2.140 C22305

In section **D1.3.5.4 "SVE synchronous memory faults"**, rule R_{SKNTR} that reads:

> When an SVE load or store instruction results in a data memory access, the detection of any of the following conditions is considered to be a Memory fault:
>
> - ...
> - When FEAT_MTE2 is implemented, a Tag Check Fault.

is changed to read:

> When an SVE load or store instruction results in a data memory access, the detection of any of the following conditions is considered to be a Memory fault:
>
> - ...
> - When FEAT_MTE2 is implemented, a synchronous Tag Check Fault.

In section **D1.3.5.4.2 "SVE First-fault and Non-fault loads"**, rule R_{YFTRN} that reads:

> When a memory access performed for any of the following elements detects a Memory fault, or is suppressed for any other reason, the FFR predicate elements starting from that element number, up to and including the highest-numbered element, are set to FALSE:
>
> - Any Active element of an SVE Non-fault vector load.
> - Any Active element of an SVE First-fault vector load except for the First active element.

is changed to read:

> When a memory access performed for any of the following elements detects a Memory fault, an asynchronous Tag check fault, or is suppressed for any other reason, the FFR predicate elements starting from that element number, up to and including the highest-numbered element, are set to FALSE:
>
> - Any Active element of an SVE Non-fault vector load.
> - Any Active element of an SVE First-fault vector load except for the First active element.

A note is added below R_{YFTRN} that reads:

An asynchronous Tag check fault setting elements in FFR to FALSE is required to be synchronous with respect to the instruction stream as per R_{XXMMP}.

2.141 C22308

In sections **D3.1 "About self-hosted trace"** and **G3.1 "About self-hosted trace"**, the text that reads:

If an Armv8.4-compliant PE implements an ETM Architecture trace unit that includes the ETM System register interface, FEAT_TRF must be implemented. If an Armv8.4-compliant PE implements a trace unit that is either not an ETM Architecture trace unit or does not implement the ETM System register interface, Arm recommends that FEAT_TRF is implemented, but this is not mandatory. This is not applicable in Armv9. If an Armv9-compliant PE implements FEAT_ETE, FEAT_TRF must be implemented.

is changed to read:

If an Armv8.4-compliant PE implements an ETM Architecture trace unit that includes the ETM System register interface, FEAT_TRF must be implemented. If an Armv9-compliant PE implements FEAT_ETE, FEAT_TRF must be implemented. If any of the following are true, Arm recommends that FEAT_TRF is implemented, but this is not mandatory:

- An Armv8.4-compliant PE or Armv9-compliant PE implements an **IMPLEMENTATION DEFINED** trace unit.
- An Armv8.4-compliant PE implements an ETM Architecture trace unit but does not implement the ETM System register interface.

2.142 C22310

In section **C6.2.115 "DC"**, under the heading 'Assembler Symbols', the text that reads:

When FEAT_RME is implemented, the following values are also valid:

CIPAPA	when op1 = 110, CRm = 1110, op2 = 001
CIGDPAPA	when op1 = 110, CRm = 1110, op2 = 101

is changed to read:

When FEAT_RME is implemented, the following value is also valid:

CIPAPA	when op1 = 110, CRm = 1110, op2 = 001

When FEAT_RME and FEAT_MTE2 are implemented, the following value is also valid:

CIGDPAPA	when op1 = 110, CRm = 1110, op2 = 101

2.143 D22311

In section In section **J1.3.1 “shared/debug”**, under CountPMUEvents function, the code that reads:

```
boolean CountPMUEvents(integer idx)
...
    if HaveEL(EL3) && (ss == SS_Secure || PSTATE.EL == EL3) then
        if !ELUsingAArch32(EL3) then
            prohibited = (MDCR_EL3.SPME == '0' && IsFeatureImplemented(FEAT_PMUv3p7)
 && MDCR_EL3.MPMX == '0');
...
```

is changed to read:

```
boolean CountPMUEvents(integer idx)
...
    if HaveEL(EL3) && (ss == SS_Secure || PSTATE.EL == EL3) then
        if !ELUsingAArch32(EL3) then
            prohibited = (MDCR_EL3.SPME == '0' && IsFeatureImplemented(FEAT_PMUv3p7)
 && MDCR_EL3.MPMX == '0');
...
```

2.144 D22320

In section **D23.2.53 “HCR_EL2, Hypervisor Configuration Register”**, in the description of field ‘TPU, bit [24]’, the text that reads:

- If EL1 is using AArch32 state, the following instructions are trapped to EL2 and reported with EC syndrome value `0x18`:
  - ICIMVAU, ICIALLU, ICIALLUIS, DCCMVAU.

is changed to read:

- If EL1 is using AArch32 state, the following instructions are trapped to EL2 and reported with EC syndrome value `0x03`:
  - ICIMVAU, ICIALLU, ICIALLUIS, DCCMVAU.

2.145 C22332

In section **D23.2.69 “ID_AA64DFR0_EL1, AArch64 Debug Feature Register 0”**, in the field description of ‘TraceFilt, bits [43:40]’, the text that reads:

> From Armv8.4, if an Embedded Trace Macrocell Architecture trace unit is implemented, the value `0b0000` is not permitted.

is changed to read:

From Armv8.4, if an Embedded Trace Macrocell Architecture trace unit is implemented, the value `0b0000` is not permitted.

If FEAT_ETE is implemented, the value `0b0000` is not permitted.

An equivalent change is made in sections **"D23.2.85 ID_DFR0_EL1, AArch32 Debug Feature Register 0"** and **"G8.2.84 ID_DFR0, Debug Feature Register 0"**

2.146 D22333

In section **D1.4.1.1 "Accessing PSTATE fields"**, the following rule:

R_{BRSMZ}

Software can use the MSR (immediate) instructions to directly write to PSTATE.{D, A, I, F, SP, PAN, UAO, SSBS, TCO, PM, SM, ZA}.

is changed to read:

R_{BRSMZ}

Software can use the MSR (immediate) instructions to directly write to PSTATE.{D, A, I, F, SP, PAN, UAO, DIT, SSBS, TCO, ALLINT, PM, SM, ZA}.

2.147 R22337

In section **D23.2.41 "ESR_EL2, Exception Syndrome Register (EL2)"**, under the heading 'ISS encoding for an exception from MSR, MRS, or System instruction execution in AArch64 state', the field description that reads:

Rt, bits [9:5]

The Rt value from the issued instruction, the general-purpose register used for the transfer.

is changed to read:

Rt, bits [9:5]

The Rt value from the issued instruction, the general-purpose register used for the transfer.

For system instructions which require that the opcode Rt field is set to `0b11111`, but where the trapped instruction has a different value of Rt, an implementation is permitted to return the value `0b11111`, instead of the value of Rt from the trapped instruction.

The equivalent changes are made in the following registers:

- **D23.2.40 "ESR_EL1, Exception Syndrome Register (EL1)"**

- **D23.2.42 “ESR_EL3, Exception Syndrome Register (EL3)”**

2.148 D22342

In section **D8.5.1.2 “Hardware management of the Access flag”**, in the pseudocode function AArch64.S1Translate(), the code that reads:

```
(FaultRecord, AddressDescriptor) AArch64.S1Translate(FaultRecord fault_in, Regime
 regime,
                                                     bits(64) va, boolean aligned,
                                                     AccessDescriptor accdesc)
    ...
        new_desc = descriptor;
        if walkparams.ha == '1' && AArch64.SettingAccessFlagPermitted(fault) then
            // Set descriptor AF bit
            new_desc<10> = '1';
    ...
```

is changed to read:

```
(FaultRecord, AddressDescriptor) AArch64.S1Translate(FaultRecord fault_in, Regime
 regime,
                                                     bits(64) va, boolean aligned,
                                                     AccessDescriptor accdesc)
    ...
        new_desc = descriptor;
        if (walkparams.ha == '1' && AArch64.SettingAccessFlagPermitted(fault) &&
              (accdesc.acctype != AccessType_AT ||
               boolean IMPLEMENTATION_DEFINED "AT updates AF")) then             //
 Set descriptor AF bit
            new_desc<10> = '1';
    ...
```

In the same pseudocode function, the code that reads:

```
(FaultRecord, AddressDescriptor) AArch64.S1Translate(FaultRecord fault_in, Regime
 regime,
                                                     bits(64) va, boolean aligned,
                                                     AccessDescriptor accdesc)
    ...

    until new_desc == descriptor || mem_desc == new_desc;
    ...
```

is changed to read:

```
(FaultRecord, AddressDescriptor) AArch64.S1Translate(FaultRecord fault_in, Regime
 regime,
                                                     bits(64) va, boolean aligned,
                                                     AccessDescriptor accdesc)
    ...

            if fault.statuscode != Fault_None then
                if (accdesc.acctype == AccessType_AT &&
                      !(boolean IMPLEMENTATION_DEFINED "AT reports the HW update
 fault")) then
                    // Mask the fault
```

```
                    fault.statuscode = Fault_None;
                else
                    return (fault, AddressDescriptor UNKNOWN);
    until new_desc == descriptor || mem_desc == new_desc;
    ...
```

An equivalent change is made in the pseudocode function AArch64.S2Translate().

2.149 R22345

In **I5.4.5.2 "Security of error records"** in statement I_{KFNCK}, the following text is removed:

> Access to error records is enabled from reset.

The following text is added to R_{QSFYN}:

> Access to error records from reset is **IMPLEMENTATION DEFINED**, and depends on the security policy of the component implementing this register.

In **I6.9.1" ERRACR, Access Configuration Register"**, in fields 'RLRA, bits [5:4]', 'SRA, bits [3:2]', and 'NSRA, bits [1:0]', the following text:

> The reset behavior of this field is:
>
> - On an Error recovery reset, this field resets to 3.

is changed to read:

> The reset domain and value of this field is **IMPLEMENTATION DEFINED**, and depends on the security policy of the component implementing this register.
>
> The reset behavior of this field is:
>
> - On a Cold reset, this field resets to an **IMPLEMENTATION DEFINED** value.
> - On an Error recovery reset, this field resets to an IMPLEMENTATION DEFINED value.

2.150 D22346

In section **D13.12.3.2 "Common microarchitectural events"**, the event description of FP_FMA_SPEC that reads:

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

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

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

is changed to read:

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

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

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

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

In the same section, the event description of INT_MUL_SPEC that reads:

`0x8048`, INT_MUL_SPEC, Integer operation speculatively executed, multiply

The counter counts each Speculatively executed integer multiply or multiply-accumulate operation counted by INT_SPEC due to any of the following A64 instructions:

- Scalar: ...
- Advanced SIMD: MLA, MLS, MUL, PMUL, PMULL, SMLAL, SMLS, SMLSL, SMULL, SQMLAL, SQMLSL, SQMULH, ...
- SVE: ...
- SVE2: ...

...

is changed to read:

`0x8048`, INT_MUL_SPEC, Integer operation speculatively executed, multiply

The counter counts each Speculatively executed integer multiply or multiply-accumulate operation counted by INT_SPEC due to any of the following A64 instructions:

- Scalar: ...

- Advanced SIMD: MLA, MLS, MUL, PMUL, PMULL, SMLAL, SMLSL, SMULL, SQMLAL, SQMLSL, SQMULH, ...
- SVE: ...
- SVE2: ...

...

In the same section, the event description of SVE_PERM_SPEC that reads:

> 0x8060, SVE_PERM_SPEC, Operation speculatively executed, SVE permute
>
> The counter counts each Speculatively executed vector or predicate permute operation due to any of the following instructions:
>
> - SVE: CLASTA, CLASTB, COMPACT, CPY (SIMD&FP scalar), CPY (scalar), DUP (indexed), DUP (scalar), EXT, INSR, LASTA, LASTB, PUNPKHI, PUNPKLO, REV (vector), REVB, REVH, REVW, SPLICE, SUNPKHI, SUNPKLO, TBL, TRN1 (vectors), TRN2 (vectors), UUNPKHI, UUNPKLO, UZP1 (vectors), UZP2 (vectors), ZIP1 (vectors), or ZIP2 (vectors).
> - SVE2: EXT, SPLICE, TBL, TBX, TRN1, TRN2, UZP1, UZP2, ZIP1, or ZIP2.
>
> ...

is changed to read:

> 0x8060, SVE_PERM_SPEC, Operation speculatively executed, SVE permute
>
> The counter counts each Speculatively executed vector or predicate permute operation due to any of the following instructions:
>
> - SVE: CLASTA, CLASTB, COMPACT, CPY (SIMD&FP scalar), CPY (scalar), DUP (indexed), DUP (scalar), EXT, INSR, LASTA, LASTB, PUNPKHI, PUNPKLO, REV (vector), REVB, REVH, REVW, SPLICE, SUNPKHI, SUNPKLO, TBL, TRN1, TRN2, UUNPKHI, UUNPKLO, UZP1, UZP2, ZIP1, or ZIP2.
> - SVE2: EXT, SPLICE, TBL, or TBX.
>
> ...

Similar changes are made to the following event descriptions:

- "0x8029, ASE_FP_FMA_SPEC, Floating-point operation speculatively executed, Advanced SIMD FMA".
- "0x802B, ASE_SVE_FP_FMA_SPEC, Floating-point operation speculatively executed, Advanced SIMD or SVE FMA".
- "0x8049, ASE_INT_MUL_SPEC, Integer operation speculatively executed, Advanced SIMD multiply".
- "0x804B, ASE_SVE_INT_MUL_SPEC, Integer operation speculatively executed, Advanced SIMD or SVE multiply".

2.151 R22349

In section **D23.2.143 "RGSR_EL1, Random Allocation Tag Seed Register"**, the text that reads:

> TAG, bits [3:0]
>
> Tag generated by the most recent IRG instruction.

is changed to read:

> TAG, bits [3:0]
>
> **IMPLEMENTATION DEFINED**

2.152 D22354

In section **D1.6.1.4 "WFE wakeup events in AArch64 state"**, under the rule 'R_{KBMFC}', the text that reads:

> All of the following are WFE wakeup events:
>
> - The execution of an SEV instruction on any PE in a multiprocessor system.
> - Any physical SError interrupt, physical IRQ interrupt, or physical FIQ interrupt received by the PE that is not disabled by EDSCR.INTdis and either of the following is true:
> - The interrupt is marked as A in R_{SXLWJ} in Asynchronous exception masking, regardless of the value of the corresponding PSTATE.{A, I, F} mask bit.
> - The interrupt is marked as B in R_{SXLWJ} in Asynchronous exception masking, AllIntMask is 0, and any of the following are true:
> - The value of the corresponding PSTATE.{A, I, F} mask bit is 0.
> - An IRQ or FIQ interrupt has Superpriority. See R_{MHWBP} and R_{GFXKY}.

is changed to read:

> All of the following are WFE wakeup events:
>
> - The execution of an SEV instruction on any PE in a multiprocessor system.
> - Any physical SError interrupt, physical IRQ interrupt, or physical FIQ interrupt received by the PE that is not disabled by EDSCR.INTdis and either of the following is true:
> - The interrupt is marked as A in R_{SXLWJ} or R_{JFKMF} in Asynchronous exception masking, regardless of the value of the corresponding PSTATE.{A, I, F} mask bit.
> - The interrupt is marked as B in R_{SXLWJ} or R_{JFKMF} in Asynchronous exception masking, AllIntMask is 0, and any of the following are true:
> - The value of the corresponding PSTATE.{A, I, F} mask bit is 0.
> - An IRQ or FIQ interrupt has Superpriority. See R_{MHWBP} and R_{GFXKY}.

In section **"D1.6.2.1 WFI wakeup events"**, under the rule 'R_{VRLPB}', the text that reads:

> All of the following are WFI wakeup events:
>
> - Any physical SError interrupt, physical IRQ interrupt, or physical FIQ interrupt received by the PE that is marked as A, B or A/B in the tables in Physical interrupt masking, regardless of the value of the corresponding PSTATE.{A, I, F} mask bit.

is changed to read:

> All of the following are WFI wakeup events:
>
> - Any physical SError interrupt, physical IRQ interrupt, or physical FIQ interrupt received by the PE that is marked as A or B in the tables in R_{SXLWJ} or R_{JFKMF} in Asynchronous exception masking, regardless of the value of the corresponding PSTATE.{A, I, F} mask bit.

2.153 D22357

In section **D1.6.1.4 "WFE wakeup events in AArch64 state"**, the rule that reads:

> R_{KBMFC}
>
> Any physical SError interrupt, physical IRQ interrupt, or physical FIQ interrupt received by the PE that is not disabled by EDSCR.INTdis and either of the following is true:
>
> - The interrupt is marked as A in R_{SXLWJ} in Asynchronous exception masking, regardless of the value of the corresponding PSTATE{A, I, F} mask bit.
> - The interrupt is marked as B in R_{SXLWJ} in Asynchronous exception masking, AllIntMask is 0, and any of the following are true:
>   - The value of the corresponding PSTATE mask bit is 0.
>   - An IRQ or FIQ interrupt has Superpriority.
>
> See R_{MHWBP} and R_{GFXKY}.
>
> If all of the following apply, any virtual SError interrupt, virtual IRQ interrupt, or virtual FIQ interrupt received by the PE:
>
> - The PE is executing at EL1 or EL0.
> - The interrupt is not disabled by EDSCR.INTdis.
> - The interrupt is marked as B in R_{BKHXL} in Virtual interrupt masking, AllIntMask is 0, and any of the following are true:
>   - The value of the corresponding PSTATE. {A, I, F} mask bit is 0.
>   - A vIRQ or vFIQ interrupt has Superpriority.
>
> See R_{SNLJH} and R_{XSPSG}.

is changed to read:

R_{KBMFC}

Any physical SError interrupt, physical IRQ interrupt, or physical FIQ interrupt received by the PE that is not disabled by EDSCR.INTdis and either of the following is true:

- The interrupt is marked as A in R_{SXLWJ} in Asynchronous exception masking, regardless of the value of the corresponding PSTATE{A, I, F} mask bit.
- The interrupt is marked as B in R_{SXLWJ} in Asynchronous exception masking, and any of the following are true:
  - The value of the corresponding PSTATE mask bit is 0.
  - AllIntMask is 0 and the IRQ or FIQ interrupt has Superpriority.

See R_{MHWBP} and R_{GFXKY}.

If all of the following apply, any virtual SError interrupt, virtual IRQ interrupt, or virtual FIQ interrupt received by the PE:

- The PE is executing at EL1 or EL0.
- The interrupt is not disabled by EDSCR.INTdis.
- The interrupt is marked as B in R_{BKHXL} in Virtual interrupt masking and any of the following are true:
  - The value of the corresponding PSTATE. {A, I, F} mask bit is 0.
  - AllIntMask is 0 and the vIRQ or vFIQ interrupt has Superpriority.

See R_{SNLJH} and R_{XSPSG}.

2.154 D22362

In section **D11.8 “Guarded Control Stack exceptions”**, the information statement ID I_{SGCVT} is changed to I_{MKLKV}.

In section **D11.4.1 “Pushing and popping exception return state”**, the information statement ID I_{QSVCQ} is changed to I_{DXBZZ}.

In section **K3.1 “Recording the call stacks from the current PE”**, the software usage statement ID S_{KFTHG} is changed to S_{NNHCN}.

In section **K3.3 “Overwriting a Guarded Control Stack record from a higher Exception level”**, the software usage statement ID S_{FFWLL} is changed to S_{KPMKV}.

In section **I5.15.1 “Error record groups”**, the second instance of rule ID R_{GFLXS} is changed to R_{LQBKM}.

In section **D1.3.6.3 “Asynchronous exception masking”**, under the heading ‘Physical interrupt masking’, the Rule ID R_{QZPXL} is changed to R_{ZSRWR}.

In section **D1.3.6.3 "Asynchronous exception masking"**, under the heading 'Physical interrupt masking', the text in R_{JFKMF} that reads:

> SCR_EL3.{NS, EEL2} are not shown, R_{QZPXL} describes Effective values involving Security states.

is changed to read:

> SCR_EL3.{NS, EEL2} are not shown, R_{ZSRWR} describes Effective values involving Security states.

In section **I5.15.2 "Fault injection groups"**, R_{PCXRD} and R_{JFZRW} are removed.

2.155 D22365

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

```
AArch64.EncodePAR(Regime regime, boolean is_ATS1Ex, AddressDescriptor addrdesc)
    PAR_EL1 = Zeros(128);
    paspace = addrdesc.paddress.paspace;

    if AArch64.isPARFormatD128(regime, is_ATS1Ex) then
        PAR_EL1.D128 = '1';
    else
        PAR_EL1.D128 = '0';
    ...

    if !IsFault(addrdesc) then
        PAR_EL1.F = '0';
        ...
        if PAR_EL1.D128 == '1' then
            PAR_EL1<119:76> = addrdesc.paddress.address<55:12>;
        else
            PAR_EL1<55:12> = addrdesc.paddress.address<55:12>;
```

is changed to read:

```
AArch64.EncodePAR(Regime regime, boolean is_ATS1Ex, AddressDescriptor addrdesc)
    paspace = addrdesc.paddress.paspace;

    if IsFeatureImplemented(FEAT_D128) then
        PAR_EL1 = Zeros(128);
        if AArch64.isPARFormatD128(regime, is_ATS1Ex) then
            PAR_EL1.D128 = '1';
        else
            PAR_EL1.D128 = '0';
    else
        PAR_EL1<63:0> = Zeros(64);
    ...

    if !IsFault(addrdesc) then
        PAR_EL1.F = '0';
        ...
        if IsFeatureImplemented(FEAT_D128) && PAR_EL1.D128 == '1' then
            PAR_EL1<119:76> = addrdesc.paddress.address<55:12>;
        else
            PAR_EL1<55:12> = addrdesc.paddress.address<55:12>;
```

2.156 D22366

In section **D8.6.6 "Stage 2 memory type and Cacheability attributes when FWB is enabled"**, in Table D8-95, the text that reads:

Stage 2 MemAttr[1:0]	Stage 1 memory type and Cacheability attribute	Stage 2 MemAttr[1:0]	Resultant memory type and Cacheability attribute
...	...	...	

is changed to read:

Stage 2 MemAttr[3:2]	Stage 1 memory type and Cacheability attribute	Stage 2 MemAttr[1:0]	Resultant memory type and Cacheability attribute
...	...	...	...
01	Normal Write-Back, Tagged	01	Normal Non-cacheable
...	...	...	...

2.157 D22370

In sections **C7.2 "Alphabetical list of A64 Advanced SIMD and floating-point instructions"** and **C8.2 "Alphabetical list of SVE instructions"**, in the SDOT, SMMLA, SUDOT, UDOT, UMMLA, USDOT, and USMMLA instructions, the following text is added:

> Operational information
>
> If PSTATE.DIT is 1:
>
> - The execution time of this instruction is independent of:
> - The values of the data supplied in any of its registers.
> - The values of the NZCV flags.
> - The response of this instruction to asynchronous exceptions does not vary based on:
> - The values of the data supplied in any of its registers.
> - The values of the NZCV flags.

The equivalent changes are made in the following section:

- **C9.2 "Alphabetical list of SME instructions"**, in the SDOT, SUDOT, SUVDOT, SVDOT, UDOT, USDOT, USVDOT and UVDOT instructions.

2.158 R22371

In section **C8.2.106 "CNTP (predicate as counter)"**, under the heading 'Operational Information', the following text is removed:

> When PSTATE.DIT is 1:
> - The execution time of this instruction is independent of:
>   - The values of the data supplied in any of its registers.
>   - The values of the NZCV flags.
> - The response of this instruction to asynchronous exceptions does not vary based on:
>   - The values of the data supplied in any of its registers.
>   - The values of the NZCV flags.

The equivalent changes are made in the following sections:

- **C8.2.107 "CNTP (predicate)"**.
- **C8.2.468 "PUNPKHI, PUNPKLO"**.

2.159 R22375

In section **A2.3.1 "The Armv9.0 architecture extension"**, in the description of FEAT_SVE_SHA3, the text that reads:

> If FEAT_SVE_SHA3 is implemented, then FEAT_SVE2 is implemented.

is changed to read:

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

2.160 D22391

In section **D12.2.2.2 "The virtual offset register"**, the text that reads:

> If EL2 is not implemented and enabled, then the virtual counter uses a fixed offset of zero.

is changed to read:

> If EL2 is not implemented, then the virtual counter uses a fixed offset of zero.

2.161 D22396

In section **J1.3.1 "shared/debug"**, in the pseudocode function PMUCountValue(), the text that reads:

```
// Check if disabled. Note that this function will return the value of Vb when
// the control register fields are all zero, even without this check.
if tc == '000' && te == '0' then
    return Vb;
```

is changed to read:

```
// Check if disabled. Note that this function will return the value of Vb when
// the control register fields are all zero, even without this check.
if tc == '000' && TH == 0 && te == '0' then
    return Vb;
```

2.162 D22399

In section **C5.2.25 "SSBS, Speculative Store Bypass Safe"**, the text that reads:

> A cache timing side channel might be exploited where a load or store uses an address that is derived from a register that is being loaded from memory using a load instruction speculatively read from a memory location. If PSTATE.SSBS is enabled, the address derived from the load instruction might be from earlier in the coherence order than the latest store to that memory location with the same virtual address.

0b0	Hardware is not permitted to load or store speculatively, in a manner that could practically give rise to a cache timing side channel, using an address derived from a register value that has been loaded from memory using a load instruction (L) that speculatively reads an entry from earlier in the coherence order from that location being loaded from than the entry generated by the latest store (S) to that location using the same virtual address as L.
0b1	Hardware is permitted to load or store speculatively, in a manner that could practically give rise to a cache timing side channel, using an address derived from a register value that has been loaded from memory using a load instruction (L) that speculatively reads an entry from earlier in the coherence order fro that location being loaded from than the entry generated by the latest store (S) to that location using the same virtual address as L.

is changed to read:

> A speculative value in a register is used in a potentially speculatively exploitable manner if it is used to form an address, generate condition codes, or generate SVE predicate values to be used by other instructions in the speculative sequence or if the execution timing of any other instructions in the speculative sequence is a function of the data loaded under speculation.

0b0	Hardware is not permitted to use speculative register values in a potentially speculatively exploitable manner if the speculative read that loads the register is from earlier in the coherence order than the entry generated by the latest store to that location using the same virtual address as the load instruction.
0b1	When the value of PSTATE.SSBS is 1, hardware is permitted to use speculative register values in a potentially speculatively exploitable manner if the speculative read that loads the register is from earlier in the coherence order than the entry generated by the latest store to that location using the same virtual address as the load instruction.

The equivalent changes are made in the following section:

- **G8.2.34 "CPSR, Current Program Status Register"**, in the field description of 'SSBS, bit [23]'.

2.163 R22404

In section **A2.3.5 "The Armv9.4 architecture extension"**, the text that reads:

> FEAT_SVE_B16B16, Non-widening BFloat16 to BFloat16 arithmetic for SVE2 and SME2
>
> FEAT_SVE_B16B16 adds SVE2 non-widening BFloat16 floating-point instructions to SVE.
>
> If FEAT_SVE2 is implemented, FEAT_SVE_B16B16 adds the SVE2 non-widening BFloat16 instructions when the PE is not in Streaming SVE mode.
>
> If FEAT_SME2 is implemented, FEAT_SVE_B16B16 adds:
> - The SVE2 non-widening BFloat16 instructions when the PE is in Streaming SVE mode.
> - The SME2 non-widening BFloat16 instructions.
>
> ...
>
> The following fields identify the presence of FEAT_SVE_B16B16:
> - ID_AA64SMFR0_EL1.B16B16.
> - ID_AA64ZFR0_EL1.B16B16.

is changed to read:

> FEAT_SVE_B16B16, Non-widening BFloat16 to BFloat16 arithmetic for SVE2 and SME2
>
> FEAT_SVE_B16B16 adds non-widening BFloat16 floating-point instructions to SVE, and multi-vector Z-targeting non-widening BFloat16 floating-point instructions to SME.
>
> If FEAT_SVE2 is implemented, FEAT_SVE_B16B16 adds the SVE non-widening BFloat16 instructions when the PE is not in Streaming SVE mode.
>
> If FEAT_SME2 is implemented, FEAT_SVE_B16B16 adds:
> - The SVE non-widening BFloat16 instructions when the PE is in Streaming SVE mode.
> - The SME Z-targeting multi-vector non-widening BFloat16 instructions.
>
> ...
>
> The following field identifies the presence of FEAT_SVE_B16B16:
> - ID_AA64ZFR0_EL1.B16B16.

The following text is added:

> FEAT_SME_B16B16, Non-widening BFloat16 to BFloat16 arithmetic for SME (ZA-targeting)
>
> FEAT_SME_B16B16 adds SME ZA-targeting non-widening BFloat16 floating-point instructions to SME.
>
> If FEAT_SME2 is implemented, FEAT_SME_B16B16 adds:
>
> - The SME ZA-targeting non-widening BFloat16 instructions.
>
> This feature is supported in AArch64 state only.
>
> FEAT_SME_B16B16 is OPTIONAL from Armv9.2.
>
> If FEAT_SME_B16B16 is implemented, then all of FEAT_SME2 and FEAT_SVE_B16B16 are implemented.
>
> The following field identifies the presence of FEAT_SME_B16B16:
>
> - ID_AA64SMFR0_EL1.B16B16.

In section **D23.2.83 "ID_AA64ZFR0_EL1, SVE Feature ID Register 0"**, the field description that reads:

> B16B16, bits[27:24]
>
> Indicates support for the following SVE2 non-widening BFloat16 instructions BFADD, BFCLAMP, BFMAX, BFMAXNM, BFMIN, BFMINNM, BFMLA, BFMLS, BFMUL, and BFSUB with BFloat16 operands and results.
>
> ...
>
> FEAT_SVE_B16B16 implements the functionality identified by `0b0001`.
>
> This field must indicate the same level of support as ID_AA64SMFR0_EL1.B16B16.

is changed to read:

> B16B16, bits[27:24]
>
> Indicates support for the SVE non-widening BFloat16 instructions BFADD, BFCLAMP, BFMAX, BFMAXNM, BFMIN, BFMINNM, BFMLA, BFMLS, BFMUL, and BFSUB, and the SME multi-vector Z-targeting non-widening BFloat16 instructions BFCLAMP, BFMAX, BFMAXNM, BFMIN, and BFMINNM, with BFloat16 operands and results.
>
> ...
>
> FEAT_SVE_B16B16 implements the functionality identified by `0b0001`.

In section **D23.2.82 "ID_AA64SMFR0_EL1, SME Feature ID Register 0"**, the field description that reads:

> B16B16, bit[43]
>
> Indicates support for the SME non-widening BFloat16 BFADD, BFCLAMP, BFMAX, BFMAXNM, BFMIN, BFMINNM, BFMLA, BFMLS, BFMOPA, BFMOPS, and BFSUB instructions with BFloat16 operands and results.
>
> ...
>
> FEAT_SVE_B16B16 implements the functionality identified by `0b0001`.
>
> This field must indicate the same level of support as ID_AA64ZFMFR0_EL1.B16B16.

is changed to read:

> B16B16, bit[43]
>
> Indicates support for the SME ZA-targeting non-widening BFloat16 BFADD, BFMLA, BFMLS, BFMOPA, BFMOPS, and BFSUB instructions with BFloat16 operands and results.
>
> ...
>
> FEAT_SME_B16B16 implements the functionality identified by `0b0001`.

2.164 D22411

In section **B2.9.1 "Speculative Store Bypass Safe (SSBS)"**, the following text is added:

> Use of PSTATE.SSBS == 0 is deprecated for performance reasons.

2.165 C22417

In section **D8.16 "TLB maintenance"**, the text that reads:

> R_{NWYRD}
>
> Entries held in a TLB are distinguished by all of the following context information:
> - The Security state and translation regime.
> - If applicable to the translation regime, then the VMID.
> - If applicable to the translation regime, then whether the translation is global or non-global.
> - If the translation is non-global, then the ASID.

is changed to read:

> R_{NWYRD}

Entries held in a TLB are distinguished by all of the following context information:

- The Security state.
- The translation regime, with the exception that hardware is not required to distinguish between EL2 and EL2&0 TLB entries.
- If applicable to the translation regime, then the VMID.
- If applicable to the translation regime, then whether the translation is global or non-global.
- If the translation is non-global, then the ASID.

In section **D8.12.1 "Behavior of HCR_EL2.E2H"**, the following text is added:

I_{X0000}

HCR_EL2.E2H is permitted to be cached in a TLB, and EL2 software is expected to perform a TLBI ALLE2 operation after changing the value of HCR_EL2.E2H.

2.166 C22437

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

If FEAT_CSV2 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 of the following resources:
  - Branch target prediction based on the branch targets used in context1.
    - This applies to both direct and indirect branches, including return instructions, but excludes the prediction of the direction of a conditional branch.
  - Data Value predictions based on data value from execution in context1.
  - Virtual address-based cache prefetch predictions generated as a result of execution in context1, based on, or causing dereference of, data values from memory.
  - Any other prediction mechanisms, other than Branch, Data Value, or Cache Prefetch predictions.

is changed to read:

If FEAT_CSV2 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 of the following resources:
  - Branch target prediction based on the branch targets used in context1.
    - This applies to both direct and indirect branches, including return instructions, but excludes the prediction of the direction of a conditional branch.
  - Data Value predictions based on data value from execution in context1.

- Virtual address-based cache prefetch predictions generated as a result of execution in context1, based on, or causing dereference of, data values from memory.
- Address predictions based on addresses used in context1.
  - This includes any form of memory disambiguation prediction or data forwarding based on partial address match.
- Any other prediction mechanisms, other than Branch, Data Value, Cache Prefetch, or Address predictions.

The following note is added to **C5.6.2, "COSP RCTX, Clear Other Speculative Prediction Restriction by Context"** in the `COSP RCTX` instruction:

Note:

This instruction applies to speculative accesses resulting from address predictions.

2.167 D22438

In section, **J1.1.3 "aarch64/functions"**, in the pseudocode function AArch64.DC(), the following code segment:

```
AArch64.DC(bits(64) regval, CacheType cachetype, CacheOp cacheop, CacheOpScope
 opscope_in)
    ...
    if DCInstNeedsTranslation(opscope) then
        ...
        if opscope IN {
                        ...
                        CacheOpScope_PoC
                        } then
            cache.shareability = memaddrdesc.mematttrs.shareability;
        else
            cache.shareability = Shareability_NSH;
    ...
```

is changed to read:

```
AArch64.DC(bits(64) regval, CacheType cachetype, CacheOp cacheop, CacheOpScope
 opscope_in)
    ...
    if DCInstNeedsTranslation(opscope) then
        ...
        if opscope IN {
                        ...
                        CacheOpScope_PoC,
                        CacheOpScope_PoU
                        } then
            cache.shareability = memaddrdesc.mematttrs.shareability;
        else
            cache.shareability = Shareability_NSH;
    ...
```

2.168 C22441

In section **H9.5.17 "CTIDEVARCH, CTI Device Architecture register"**, in the field description of 'REVISION, bits [19:16]', the text that reads:

> When FEAT_DoPD is implemented:
>
> Revision.
>
> Defines the architecture revision of the component.
>
> The value of this field is an **IMPLEMENTATION DEFINED** choice of:

`0b0000`	First revision.
`0b0001`	As `0b0000`, and also adds support for CTIDEVCTL.

> All other values are reserved.
>
> Access to this field is RO.
>
> Otherwise:
>
> Revision.
>
> Defines the architecture revision of the component.
>
> All other values are reserved.
>
> Reads as `0b0000`.
>
> Access to this field is RO.

is changed to:

> Revision. Defines the architecture revision of the component.
>
> The value of this field is an **IMPLEMENTATION DEFINED** choice of:

`0b0000`	First revision.
`0b0001`	As `0b0000`, and also adds support for CTIDEVCTL.

> All other values are reserved.
>
> When FEAT_DoPD is implemented, the value `0b0000` is not permitted.
>
> Access to this field is RO.

2.169 D22450

In section **I3.1.4 "Access permissions for external views of the Performance Monitors"**, the following row is added in "Table I3-2"Access permissions for the Performance Monitors registers when FEAT_PMUv3_EXT32 is implemented":

Offset	Register	Domain	Off	DLK	OSLK	EPMAD	EPMSSAD	Def	SLK
0x100	PMICNTR_EL0[31:0]	Core	Error	Error	Error	Error	-	RW	RO
0x104	PMICNTR_EL0[63:32]								

2.170 D22464

In section **D2.8.5.1 "Specifying the halfword-aligned address that an Address breakpoint matches on"**, the table that reads:

BAS	Match Instruction at	Constraint for debuggers
0b0011	DBGBCR<n>_EL1	Use for T32 instructions.
0b1100	DBGBCR<n>_EL1 + 2	Use for T32 instructions.
0b1111	DBGBCR<n>_EL1	Use for A64 and A32 instructions.

is changed to read:

BAS	Match Instruction at	Constraint for debuggers
0b0011	DBGBVR<n>_EL1	Use for T32 instructions.
0b1100	DBGBVR<n>_EL1 + 2	Use for T32 instructions.
0b1111	DBGBVR<n>_EL1	Use for A64 and A32 instructions.

In the same section, in the footnote of 'Figure D2-2 "Summary of BAS field meanings for Address Match breakpoints"', the text that reads:

- -2 means ((DBGBVR<n>_EL1[maxAddressSize:2]:00) –
- -1 means ((DBGBVR<n>_EL1[maxAddressSize:2]:00) –

is changed to read:

- -2 means ((DBGBVR<n>_EL1[maxAddressSize:2]:00) – 2)
- -1 means ((DBGBVR<n>_EL1[maxAddressSize:2]:00) – 1)

2.171 D22470

In section **D8.16 "TLB maintenance"**, rule R_{TVTYQ} that reads:

R_{TVTYQ}

> When a TLB maintenance instruction applies only to stage 2 entries, all of the following apply:
>
> - ...
> - If the stage 2 translation information contained in a single block or page has been collectively cached in multiple TLB entries, then all entries containing that stage 1 information are invalidated, regardless of whether the entry would be used to translate the address being invalidated.
> - ...

is changed to read:

> R_{TVTYQ}
>
> When a TLB maintenance instruction applies only to stage 2 entries, all of the following apply:
>
> - ...
> - If the stage 2 translation information contained in a single Block or Page has been collectively cached as one or more smaller TLB entries, for example as the result of splintering, then all entries containing that stage 2 information are invalidated, regardless of whether the entry would be used to translate the address being invalidated.
> - ...

The same clarification, but not the correction, is applied to R_{LGSCG}.

2.172 D22476

In section **I3.1 "About the external interface to the Performance Monitors registers"**, the following note is removed:

> This means that register views in the external debug interface to the Performance Monitors registers are the same size as the register views in the System interface when the PE is using AArch64. The 32-bit external view FEAT_PMUv3_EXT32, is not accessible for Performance Monitors registers which are extended to 64 bits.

2.173 C22477

In section **I4.1 "About the external interface to the Activity Monitors Extension registers"**, the following text is removed:

> - When FEAT_AMU_EXT64 is implemented, all of the following apply:
>   - All AMU registers are 64 bits and are accessed as single 64-bit registers.
>   - Permitted doubleword-aligned 64-bit accesses to 64-bit registers are single-copy atomic at doubleword granularity.

- It is **IMPLEMENTATION DEFINED** whether word-aligned 32-bit accesses to either half of a 64-bit register that is mapped to a doubleword-aligned pair of adjacent 32-bit locations are supported.

2.174 C22479

In section **B2.9.4 "Restrictions on the effects of speculation from Armv8.5"**, the following bullet point is added after the first bullet point:

- Branch type prediction based on the branch types used in context1; in which the hardware predicts a branch of a given type to be a different type of branch, and the type of the branch is used to exploitatively control target prediction of indirect branches within context2.
- Where "branch type" is used to distinguish between: not a branch, direct branch, indirect branch, and branch and link (or call).

In section **B2.9.2 "Definition of exploitative control of speculative execution"**, the following note is added:

Note:

Eviction of entries from one context by another context is not considered exploitative control as long as eviction does not result in other forms of prediction which could be exploitatively controlled.

2.175 D22480

In section **D16.4 "Enabling profiling"**, in Table D16-1, the row that reads:

NSE	NS	NSPBE	NSPB	E2PB	EEL2	TGE	EL3	EL2	EL1	EL0
1	0	1	0b0x	x	x	x	D	D	D	D

is changed to read:

NSE	NS	NSPBE	NSPB	E2PB	EEL2	TGE	EL3	EL2	EL1	EL0
1	1	1	0b0x	x	x	x	D	D	D	D

2.176 C22487

In section **D23.10.2 "CNTHCTL_EL2, Counter-timer Hypervisor Control Register"**, in the accessibility pseudocode statement MRS <Xt>, CNTKCTL_EL1, the code that reads:

```
if ELIsInHost(EL2) then
       X[t, 64] = CNTHCTL_EL2;
```

is changed to read:

```
if ELIsInHost(EL2) then
       X[t, 64] = CNTHCTL_EL2_VHE(CNTHCTL_EL2);
```

Similar changes are made in the accessibility pseudocode statement MSR CNTKCTL_EL1, <Xt>.

2.177 R22488

In section **D16.7.1 "Restrictions on the current write pointer"**, the text that reads:

> If these rules are not followed, the value returned for a direct read of PMBPTR_EL1 is **UNKNOWN**, the behavior is **UNPREDICTABLE**, and the PE might do any of the following at any point after profiling is enabled:
>
> - Write sample records to any virtual address that is writable at the owning Exception level in the owning translation regime.
> - Generate a Profiling Buffer management event, with or without indicating data loss, for one of the following reasons:
>   - The Profiling Buffer is full.
>   - Any MMU Fault.

is changed to read:

> If these rules are not followed, the value returned for a direct read of PMBPTR_EL1 is **UNKNOWN**, the behavior is **UNPREDICTABLE**, and the PE might do any of the following at any point after profiling is enabled:
>
> - Write sample records to any virtual address that is writable at the owning Exception level in the owning translation regime.
> - Generate a Profiling Buffer management event, with or without indicating data loss, for one of the following reasons:
>   - A Buffer full buffer management event [xref D16.8.2].
>   - An Access not allowed buffer management event [xref D16.8.5].
>   - Any MMU Fault [xref D16.8.3].
>   - An IMPLEMENTATATION DEFINED buffer management event [xref D16.8.6].
> - Silently discard all profiling data without writing it to memory.
> - Behave as if profiling is disabled.

In section **D6.2.2 "Restrictions on programming the Trace Buffer Unit"**, the text that reads:

> R_{MGZWR}

> If the current write pointer has an out-of-range value, or a misaligned value that is not a restart value when the Trace Buffer Unit attempts to write to the trace buffer, then any of the following might occur:
>
> - ...

is changed to read:

> R_{MGZWR}
>
> If the current write pointer has an out-of-range value, or a misaligned value that is not a restart value when the Trace Buffer Unit attempts to write to the trace buffer, then any of the following might occur:
>
> - ...
> - Silently discard all trace data without writing it to memory.
> - Behave as if the trace buffer is disabled.

2.178 D22496

In section **D23.10.29 "CNTVCT_EL0, Counter-timer Virtual Count Register"**, the text that reads:

> Bits [63:0] Virtual count value.
>
> When EL2 is implemented, if the access is not trapped, and any of the following are true, then reads of CNTVCT_EL0 return (PhysicalCountInt<63:0> - CNTVOFF_EL2<63:0>):
>
> - All of the following are true:
> - The Effective value of HCR_EL2.{E2H, TGE} is not {1, 1}.
> - CNTVCT_EL0 is read from EL0.
> - All of the following are true:
> - The Effective value of HCR_EL2.E2H is 0.
> - CNTVCT_EL0 is read from EL0.
> - CNTVCT_EL0 is read from EL1 or EL3.

is changed to read:

> Bits [63:0] Virtual count value.
>
> When EL2 is implemented, if the access is not trapped, and any of the following are true, then reads of CNTVCT_EL0 return (PhysicalCountInt<63:0> - CNTVOFF_EL2<63:0>):
>
> - All of the following are true:
> - The Effective value of HCR_EL2.{E2H, TGE} is not {1, 1}.
> - CNTVCT_EL0 is read from EL0.
> - All of the following are true:

    - The Effective value of HCR_EL2.E2H is 0.
    - CNTVCT_EL0 is read from EL2.
- CNTVCT_EL0 is read from EL1 or EL3.

Similar changes are also made to section **D23.10.28 "CNTVCTSS_EL0, Counter-timer Self-Synchronized Virtual Count register"**.

2.179 C22498

In section **D6.3.2 "Behavior when address translation disabled"**, under the rule R_{PBZRZ}, the text that reads:

> R_{PBZRZ}
>
> If TRBLIMITR_EL1.nVM is 1, the Base pointer, Limit pointer, and current write pointer are:
>
> - Physical address in the owning Security state if the owning translation regime has no stage 2 translation.
> - Intermediate physical addresses (IPAs) in the owning Security state if the owning translation regime has stage 2 translations.
>
> These addresses are output directly by stage 1 without any address translation.

is changed to read:

> R_{PBZRZ}
>
> If TRBLIMITR_EL1.nVM is 1, the Base pointer, Limit pointer, and current write pointer are:
>
> - Physical address in the owning Security state if the owning translation regime has no stage 2 translation.
> - Intermediate physical addresses (IPAs) in the owning Security state if the owning translation regime has stage 2 translations.
>
> These addresses are output directly by stage 1 without any address translation. Stage 1 translation is disabled for writes to the trace buffer made by the Trace Buffer Unit.

In section **D6.3.3 "Effects of stage 2 translation"**, under the rule I_{ZSDMR}, the text that reads:

> For example:
>
> - The intermediate physical addresses are translated to physical addresses by stage 2 translation, and checked for stage 2 MMU faults.
> - The attributes from stage 1 are combined with the attributes from the stage 2 translation to generate the physical memory attributes.
> - If the Effective value of HCR_EL2.DC in the owning translation regime is 1, then stage 1 translation is disabled and the memory type produced by stage 1 is Normal Non-shareable,

Inner Write-Back Cacheable Read-Allocate Write-Allocate, Outer Write-Back Cacheable Read-Allocate Write-Allocate, regardless of the values of SCTLR_EL1.C and TRBMAR_EL1.

- If the Effective value of HCR_EL2.MIOCNCE in the owning translation regime is 0, then for permitted accesses to a memory location that use a common definition of the Shareability and Cacheability of the location, there is no loss of coherency if the Inner Cacheability attribute for those accesses differs from the Outer Cacheability attribute.

is changed to read:

Example:

- The intermediate physical addresses are translated to physical addresses by stage 2 translation, and checked for stage 2 MMU faults.
- If the Effective value of TRBLIMITR_EL1.nVM is 1, meaning stage 1 translation is disabled, and the resulting IPA is translated by a stage 2 Block or Page descriptor with the AssuredOnly attribute set to 1, then the access translated by that descriptor generates a stage 2 Permission fault
- The attributes from stage 1 are combined with the attributes from the stage 2 translation to generate the physical memory attributes.
- If the Effective value of HCR_EL2.DC in the owning translation regime is 1, then stage 1 translation is disabled and the memory type produced by stage 1 is Normal Non-shareable, Inner Write-Back Cacheable Read-Allocate Write-Allocate, Outer Write-Back Cacheable Read-Allocate Write-Allocate, regardless of the values of SCTLR_EL1.C and TRBMAR_EL1.
- If the Effective value of HCR_EL2.MIOCNCE in the owning translation regime is 0, then for permitted accesses to a memory location that use a common definition of the Shareability and Cacheability of the location, there is no loss of coherency if the Inner Cacheability attribute for those accesses differs from the Outer Cacheability attribute.

2.180 C22499

In section **D1.3.2.1 “Synchronous exception entry”**, the following rule is added:

I_{X0001}

For synchronous exceptions which result in the exception handler emulating the instruction and returning to the following instruction, the handler typically completes by ensuring the state of the PE on returning from the exception is consistent with the emulated instruction having been executed. This usually includes ensuring the value of SPSR_ELx.BTYPE is consistent with the instruction being emulated having executed.

For SVC, HVC, and SMC instructions, when these instructions execute without being trapped, the value of ELR_ELx already points to the following instruction. However SPSR_ELx.BTYPE is not guaranteed to be `0b00` and so might not be consistent with the SVC/HVC/SMC instruction having executed. This might subsequently cause a Branch Target exception at the instruction after the SVC/HVC/SMC instruction. This is unlikely to be encountered in real code since it

requires the SVC/HVC/SMC to be executed in a non-Guarded page with the instruction after the SVC/HVC/SMC executed in a Guarded page.

2.181 D22502

In section **D8.2.7.1 "Stage 1 Protected Attribute"**, rules R_{VDPDW} and R_{NYFNK} that read:

R_{VDPDW}

For a stage 1 descriptor, if Protection is enabled and any of the following checks fail, then RCW and RCWS instructions do not update that descriptor:

- An RCW State check in all of the following cases:
  - For a Protected descriptor, there is an attempt to change the protection or validity of that descriptor.
  - For a Non-protected descriptor, there is an attempt to change the protection of that descriptor.
- An RCW Mask check in all of the following cases:
  - For a Protected and valid descriptor, when an attempt is made to update a bit in that descriptor and the Effective value of the corresponding RCWMASK_EL1 bit is 0.

R_{NYFNK}

For a stage 1 descriptor, if any of the following checks fail, then RCWS instructions do not update that descriptor:

- An RCWS State check when there is an attempt to change the validity of that descriptor.
- For a valid descriptor, an RCWS Mask check when an attempt is made to update a bit in that descriptor and the Effective value of the corresponding RCWSMASK_EL1 bit is 0.

are changed to read:

R_{VDPDW}

For a stage 1 descriptor, if Protection is enabled and the RCW checks fail, then RCW and RCWS instructions do not update that descriptor. The RCW checks are:

RCW State Checks:

- For a Protected descriptor, there is an attempt to change the protection or validity of that descriptor.
- For a Non-protected descriptor, there is an attempt to change the protection of that descriptor.

RCW Mask Check:

- For a Protected and valid descriptor, there is an attempt to update a bit in that descriptor and the Effective value of the corresponding RCWMASK_EL1 bit is 0.

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

In addition, the descriptions of RCW* instructions in Part C are changed to say that these instructions are for atomic updates to translation table entries and not for general use.

2.182 C22510

In section **B2.3.2.1 "Intrinsic data, control and order dependencies"**, the following subsection is added:

> B2.3.2.1.5 Branch with Link to Register instructions
>
> All of the following apply to the effects generated by the BLR X0 instruction:
>
> - D1: There is an Intrinsic data dependency from the Register Read effect of X0 to the Branching effect of the instruction.

2.183 C22511

In section **B2.3.7 "Ordering relations"**, the text that reads:

> Explicitly-Hazard-ordered-before
>
> An effect E_1 is Explicitly-Hazard-ordered-before an effect E_2 if all of the following apply:
>
> - E_1 is an Explicit Memory Read effect.
> - E_1 appears in program order before E_3.
> - E_1 and E_3 are to the Same Location.
> - E_3 is an Explicit Memory Read effect.
> - E_3 is Coherence-before E_2.
> - E_3 and E_2 are from different Processing Elements.
> - E_3 is an Explicit Memory Write effect.
>
> Note:

> The Explicitly-Hazard-ordered-before relation does not apply when either E_1 or E_3 or both are generated by SVE instructions.

is changed to read:

> Explicitly-Hazard-ordered-before
>
> An effect E_1 is Explicitly-Hazard-ordered-before an effect E_2 if all of the following apply:
> - E_1 is an Explicit Memory Read effect.
> - E_1 appears in program order before E_3.
> - E_1 and E_3 are to the Same Location.
> - E_3 is an Explicit Memory Read effect.
> - E_3 is Coherence-before E_2.
> - E_3 and E_2 are from different Processing Elements.
> - E_2 is an Explicit Memory Write effect.
>
> Note:
>
> The Explicitly-Hazard-ordered-before relation does not apply when either E_1 or E_2 or both are generated by SVE instructions.

2.184 C22521

In section **J1.3.3 "shared/functions"**, in the function EffectiveHCR_EL2_E2H(), the code that reads:

```
bit EffectiveHCR_EL2_E2H()
    if !EL2Enabled() then
        return '0';

    if !IsFeatureImplemented(FEAT_VHE) then
        return '0';

    if !IsFeatureImplemented(FEAT_E2H0) then
        return '1';

    return HCR_EL2.E2H;
```

is changed to read:

```
bit EffectiveHCR_EL2_E2H()
    if !IsFeatureImplemented(FEAT_VHE) then
        return '0';

    if !IsFeatureImplemented(FEAT_E2H0) then
        return '1';

    return HCR_EL2.E2H;
```

In the same section, in function ELIsInHost(), the code that reads:

```
boolean ELIsInHost(bits(2) el)
    ...
    case el of
        ...
        when EL2
            return EffectiveHCR_EL2_E2H() == '1';
        ...
        when EL0
            return EffectiveHCR_EL2_E2H():HCR_EL2.TGE == '11';
        ...
```

is changed to read:

```
boolean ELIsInHost(bits(2) el)
    ...
    case el of
        ...
        when EL2
            return EL2Enabled() && EffectiveHCR_EL2_E2H() == '1';
        ...
        when EL0
            return EL2Enabled() && EffectiveHCR_EL2_E2H():HCR_EL2.TGE == '11';
        ...
```

2.185 R22529

In section **D23.4.7 "TRBSR_EL1, Trace Buffer Status/syndrome Register"**, in the field description of 'DAT, bit[23]', the text that reads:

> DAT, bit [23]
>
> When FEAT_TRBE_EXT is implemented:
>
> Data. Indicates when the Trace Buffer Unit has trace data that has not yet been written to memory.

0b0	Internal buffers are empty. All Trace operations Accepted by the Trace Buffer Unit will Complete in finite time.
0b1	Internal buffers are not empty.

> When TRBSR_EL1.{DAT, S} is {0, 1}, meaning Collection is stopped and the Trace Buffer Unit internal buffers are empty, then all trace data has been written to memory. An additional Data Synchronization Barrier may be required to ensure that the writes are Complete. When TRBSR_EL1.DAT is 0 and Collection is not stopped, there may still be trace data held by the trace unit that the Trace Buffer Unit has not Accepted.
>
> That is, TRBSR_EL1.DAT reads as 1 when the Trace Buffer Unit has Accepted trace data from the trace unit, but has not yet written it to memory.
>
> The reset behavior of this field is:

> - On a Cold reset, this field resets to an architecturally **UNKNOWN** value.
>
> Otherwise:
>
> Reserved, **RES0**.

is changed to read:

> Bit [23]
>
> When FEAT_TRBE_EXT is implemented:
>
> This field reads as an **UNKNOWN** value.
>
> Otherwise:
>
> Reserved, **RES0**.

2.186 R22532

In section **H2.5 "Exiting Debug state"**, the text that reads:

> The PE exits Debug state when it receives a Restart request trigger event. If EDSCR.ITE == 0 the behavior of any instruction issued through the ITR in Normal access mode or an operation issued by a DTR access in memory access mode that has not completed execution is CONSTRAINED **UNPREDICTABLE**, and must do one of the following:
>
> - It must complete execution in Debug state before the PE executes the restart sequence.
> - It must complete execution in Non-debug state after the PE executes the restart sequence.
> - It must be abandoned. This means that the instruction does not execute. Any registers or memory accessed by the instruction are left in an **UNKNOWN** state.

is changed to read:

> The PE exits Debug state when it receives a Restart request trigger event. If EDSCR.ITE == 0, when the PE receives a Restart Request trigger event, the behavior of any instruction issued through the ITR in Normal access mode or an operation issued by a DTR access in memory access mode that has not completed execution is one of the following:
>
> - The instruction completes execution in Debug state and the PE ignores the restart request and remains in Debug state.
> - The instruction completes execution in Debug state before the PE executes the restart sequence.
> - The instruction completes execution in Non-debug state after the PE executes the restart sequence.
> - The instruction is abandoned. This means that the instruction does not execute. Any registers or memory accessed by the instruction are left in an **UNKNOWN** state.

2.187 D22546

In section **D10.3 "Memory region tagging type"**, the text that reads:

> R_{YSLYC}
>
> If the memory region tagging type for a memory region is:
>
> - Tagged then all of the following are true:
> - If the memory region has the Non-shareable attribute, it is **IMPLEMENTATION DEFINED** whether the memory region is treated as Tagged or Untagged.
> - If SCTLR_ELx.C is 0 for a memory access, it is CONSTRAINED **UNPREDICTABLE** whether the memory region is treated as Tagged or Untagged.
> - Canonically Tagged then all of the following are true:
> - If the memory region has the Non-shareable attribute, it is **IMPLEMENTATION DEFINED** whether the memory region is treated as Canonically Tagged or Untagged.
> - If SCTLR_ELx.C is 0 for a memory access, it is CONSTRAINED **UNPREDICTABLE** whether the memory region is treated as Canonically Tagged or Untagged.

is changed to read:

> R_{YSLYC}
>
> If the memory region tagging type for a memory region is:
>
> - Tagged then all of the following are true:
> - For the EL1&0 translation regime
> - If HCR_EL2.DC=0 and the stage 1 of translation assigns the Non-shareable attribute it is **IMPLEMENTATION DEFINED** whether the memory region is treated as Tagged or Untagged.
> - If the combined stage 1 and stage 2 Shareability attributes are Non-shareable it is **IMPLEMENTATION DEFINED** whether the memory region is treated as Tagged or Untagged.
> - For a translation regime other than EL1&0
> - If the stage 1 of translation assigns the Non-shareable attribute it is **IMPLEMENTATION DEFINED** whether the memory region is treated as Tagged or Untagged.
> - If SCTLR_ELx.C is 0 for a memory access, it is CONSTRAINED **UNPREDICTABLE** whether the memory region is treated as Tagged or Untagged.
> - Canonically Tagged then all of the following are true:
> - For the EL1&0 translation regime
> - If HCR_EL2.DC=0 and the stage 1 of translation assigns the Non-shareable attribute it is **IMPLEMENTATION DEFINED** whether the memory region is treated as Canonically tagged or Untagged.

        - If the combined stage 1 and stage 2 Shareability attributes are Non-shareable it is **IMPLEMENTATION DEFINED** whether the memory region is treated as Canonically tagged or Untagged.
    - For a translation regime other than EL1&0
        - If the stage 1 of translation assigns the Non-shareable attribute it is **IMPLEMENTATION DEFINED** whether the memory region is treated as Canonically tagged or Untagged
    - If SCTLR_ELx.C is 0 for a memory access, it is CONSTRAINED **UNPREDICTABLE** whether the memory region is treated as Canonically Tagged or Untagged.

2.188 D22547

In section **D23.5.8 "PMCR_EL0, Performance Monitors Control Register"**, in the field description of "PTR, bits [63:0]", the text that reads:

> Resetting PMCCNTR_EL0 does not change the cycle counter overflow bit. If FEAT_PMUv3p5 is implemented, the value of PMCR_EL0.LC is ignored, and bits [63:0] of the cycle counter are reset.

is changed to read:

> Resetting PMCCNTR_EL0 does not change the cycle counter overflow bit. The value of PMCR_EL0.LC is ignored, and bits [63:0] of the cycle counter are reset.

The equivalent changes are made in the following sections:

- **G8.4.9 "PMCR, Performance Monitors Control Register"**.
- **I6.3.21 PMCR_EL0, "Performance Monitors Control Register"**.

2.189 R22550

In section **D16.6.3 "Additional information for each profiled memory access operation"**, the following text is added:

> It is **IMPLEMENTATION DEFINED** whether a sampled DC ZVA, DC GVA, and DC GZVA operation is recorded as a store operation targeting unspecified registers or a cache maintenance operation. The behavior if the sampled operation is recorded as a data cache maintenance operation is described in Additional information for other operations. Arm recommends that these operations are recorded as store operations.

In addition, a cross-reference is added to section D16.6.7 "Additional information for other operations."

In section **D16.6.7 "Additional information for other operations"**, the text that reads:

> For cache maintenance operations by virtual address, cache prefetch, other than SVE cache prefetch, or address translation instructions, the profiling operation:
>
> - Captures an **IMPLEMENTATION DEFINED** subset of the information captured for a load instruction

is changed to read:

> For cache maintenance operations by virtual address, cache prefetch, other than SVE cache prefetch, or address translation instructions, the profiling operation:
>
> - Captures an **IMPLEMENTATION DEFINED** subset of the information captured for a load or store instruction

2.190 D22551

In section **D23.5.9 "PMECR_EL1, Performance Monitors Extended Control Register (EL1)"**, under the heading 'PMEE, bits [1:0]', the text that reads:

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

Is changed to read:

> The reset behavior of this field is:
>
> - On a Warm reset:
>   - When the highest implemented Exception level is EL1, this field resets to `0b00`.
>   - Otherwise, this field resets to an architecturally **UNKNOWN** value.

In section **D23.3.17 "MDCR_EL2, Monitor Debug Configuration Register (EL2)"**. under the heading 'PMEE, bits [41:40]', the text that reads:

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

is changed to read:

> The reset behavior of this field is:
>
> - On a Warm reset:
>   - When the highest implemented Exception level is EL2, this field resets to `0b00`.
>   - Otherwise, this field resets to an architecturally **UNKNOWN** value.

In section **D23.3.18 "MDCR_EL3, Monitor Debug Configuration Register (EL3)"**, under the heading 'PMEE, bits [41:40]', the text that reads:

> The reset behavior of this field is:

- On a Warm reset, 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 `0b00`.

2.191 C22556

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

Offset	Register	Domain	Off	DLK	OSLK	EPMAD	EPMSSAD	Def
0x800+8xn	PMEVFILT2R<n>	Core	Error	Error	Error	--	-	RW

is changed to read:

Offset	Register	Domain	Off	DLK	OSLK	EPMAD	EPMSSAD	Def
0x800+8xn	PMEVFILT2R<n>	Core	Error	Error	Error	Error	-	RW

In the same section, in Table I3-2 'Access permissions for the Performance Monitors registers when FEAT_PMUv3_EXT32 is implemented', the following entry is added:

Offset	Register	Domain	Off	DLK	OSLK	EPMAD	EPMSSAD	Def
0x800+4xn	PMEVFILT2R<n>	Core	Error	Error	Error	Error |-		RW

2.192 D22558

In section **D1.3.5.5 "Prioritization of Synchronous exceptions taken to AArch64 state"**, the prioritization list in R_{ZFGJP} that reads:

Priority	Synchronous exception type
...	...
8	Instruction Abort exceptions
9	Breakpoint exceptions
10	Illegal Execution state exceptions
...	...

is changed to read:

Priority	Synchronous exception type
...	...

Priority	Synchronous exception type
8	Illegal Execution state exceptions. It is **IMPLEMENTATION DEFINED** whether this is prioritized here or at 11.
9	Instruction Abort exceptions
10	Breakpoint exceptions
11	Illegal Execution state exceptions. It is **IMPLEMENTATION DEFINED** whether this is prioritized here or at 8.
...	...

The Priority number in subsequent rows of the table are incremented accordingly.

2.193 D22564

In section **D23.4.6 "TRBPTR_EL1, Trace Buffer Write Pointer Register"**, in the field description of "PTR, bits [63:0]", the text that reads:

> If PMBIDR_EL1.Align is not zero, then it is **IMPLEMENTATION DEFINED** whether bits [M-1:0] are **RES0** or read/write, where M is an integer between 1 and PMBIDR_EL1.Align inclusive.

is changed to read:

> If TRBIDR_EL1.Align is not zero, then it is **IMPLEMENTATION DEFINED** whether bits [M-1:0] are **RES0** or read/write, where M is an integer between 1 and TRBIDR_EL1.Align inclusive.

The equivalent changes are made in **H9.4.29 "TRBPTR_EL1, Trace Buffer Write Pointer Register"**.

2.194 D22568

In section **A2.2.10 "The Armv8.9 architecture extension"**, under the heading 'FEAT_CSSC, Common Short Sequence Compression instructions', the text that reads:

> In an Armv8.9 implementation, if FEAT_AdvSIMD is implemented, FEAT_CSSC is implemented.

is changed to read:

> FEAT_CSSC is mandatory from Armv8.9.

2.195 E22575

In section **D23.2.169 "SMIDR_EL1, Streaming Mode Identification Register"**, the following text is added:

> NSMC, bits [59:56]
>
> The number of shared SMCUs usable by this PE minus 1, which correspond to the same values of SMIDR_EL1.Affinity and SMIDR_EL1.Affinity2. This field is `0b0000` if the implementation of

Streaming SVE mode associated with this PE is not shared with other PEs. The value `0b1111` is reserved for future expansion.

The field has an **IMPLEMENTATION DEFINED** value.

Access to this field is RO.

2.196 R22577

In section **D23.5.12 "PMEVTYPER<n>_EL0, Performance Monitors Event Type Registers, n = 0 - 30"**, in the field description of 'TC, bits [63:61]', 'TE, bit [60]', and 'TH, bits [43:32]', the text that reads:

The reset behavior of this field is:

- On a Warm reset:
  - When AArch32 is supported, this field resets to 0.
  - Otherwise, this field resets to an architecturally **UNKNOWN** value.

is changed to read:

The reset behavior of this field is:

- On a Warm reset:
  - When AArch32 is supported at EL1, this field resets to `0b000`.
  - Otherwise, this field resets to an architecturally **UNKNOWN** value.

2.197 C22583

In **D13.12.3.2 "Common microarchitectural events"**, under heading '`0x80CB`, LDST_FIXED_OPS_SPEC, Non-scalable load or store element Operation speculatively executed', the text that reads:

The counter counts each Speculatively executed Memory-read operation or Memory-write operation due to any of:

- Any load, store, or atomic operation, other than loads and stores of the SVE P and Z registers, and the SME ZA registers.
- Any SVE non-vector load or store operation.
- An SVE replicating LD1R or LD1RQ instruction.

See ALU operation counts for information on the counter increment for different types of instruction.

is changed to read:

> The counter counts each Speculatively executed Memory-read operation or Memory-write operation due to any of:
>
> - Any load, store, or atomic operation, other than vector loads and stores of the SVE P and Z registers, and the SME ZA registers.
> - An SVE replicating load instructions, including LD1RB, LD1RD, LD1RH, LD1ROB, LD1ROD, LD1ROH, LD1ROW, LD1RQB, LD1RQD, LD1RQH, LD1RQW, LD1RSB, LD1RSH and LD1RSW.
>
> Note that the instruction is counted even when it is a replicating load instruction loading a single element of the same size of the vector length, meaning the loaded data is not replicated.

In **D13.12.3.2 "Common microarchitectural events"**, under heading '`0x8004`, SIMD_INST_SPEC, Operation speculatively executed, SIMD', the text that reads:

> The counter counts each Speculatively executed operation due to any of:
>
> - An SVE instruction that is not a non-SIMD SVE instruction.
> - An A64 Advanced SIMD instruction that is not an Advanced SIMD scalar instruction.
> - An SME instruction that is not a non-SIMD SME instruction.
>
> It is **IMPLEMENTATION DEFINED** which Advanced SIMD operations are counted in AArch32 state.

is changed to read:

> The counter counts each Speculatively executed operation due to any of:
>
> - An Advanced SIMD, SVE, or SME SIMD data-processing operation. Non-SIMD data-processing operations are not counted.
> - A structure load/store of one or more SIMD&FP registers.
> - A load and replicate to one or more SIMD&FP registers.
> - A scalar load/store of a SIMD&FP Q register or pair of Q registers.
> - An SVE or SME load/store.
>
> It is **IMPLEMENTATION DEFINED** which Advanced SIMD operations 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. From Armv9.5, other scalar loads and stores to SIMD&FP registers are not counted.

An equivalent change is made under heading '`0x8000`, SIMD_INST_RETIRED, Instruction architecturally executed, SIMD'.

2.198 D22586

In section **J.1.3.3 "shared/functions"**, in the pseudocode function AArch64.CheckTimerConditions(), the code segment that reads:

```
AArch64.CheckTimerConditions()
   ...
   if CNTPS_CTL_EL1.ENABLE == '1' then
       status = IsTimerConditionMet(offset, CNTPS_CVAL_EL1,
                                    CNTPS_CTL_EL1.IMASK, InterruptID_CNTPS);
      CNTPS_CTL_EL1.ISTATUS = if status then '1' else '0';
   end
  ...
```

is changed to read:

```
AArch64.CheckTimerConditions()
   ...
   if CNTPS_CTL_EL1.ENABLE == '1' then
       status = IsTimerConditionMet(Zeros(64), CNTPS_CVAL_EL1,
                                    CNTPS_CTL_EL1.IMASK, InterruptID_CNTPS);
       CNTPS_CTL_EL1.ISTATUS = if status then '1' else '0';
   end
   ...
```

2.199 R22594

In section **D23.2.157 "SCTLR2_EL2, System Control Register (EL2)"**, in the field description of 'EnANERR, bit [4]', the list of excluded memory access types is extended:

> It is implementation-specific whether this field applies to memory reads generated by each of the following:
>
> ...
>
> - When FEAT_NV2 is implemented, a memory read that is generated as a result of HCR_EL2.NV2 transforming an MRS or MRRS instruction to a load, and the target memory type is not Normal Inner and Outer Write-back cacheable.

In the same section, in the field description of 'EnADERR, bit [3]', the list of excluded memory access types is extended:

> It is implementation-specific whether this field applies to memory reads generated by each of the following:
>
> ...
>
> - When FEAT_NV2 is implemented, a memory read that is generated as a result of HCR_EL2.NV2 transforming an MRS or MRRS instruction to a load.

2.200 D22595

In section **D13.12.3.2 "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.

2.201 D22596

In section **D13.12.3.2 "Common microarchitectural events"**, the text for the PMU event PRF_SPEC that reads:

> `0x8087`, PRF_SPEC, Operation speculatively executed, prefetch
>
> The counter counts each Speculatively executed prefetch operation due to any of the following A64 instructions:
>
> - Scalar: PRFM.
> - SVE: PRFB, PRFD, PRFH, or PRFW.
>
> It is **IMPLEMENTATION DEFINED** which prefetch operations are counted in AArch32 state.

is changed to read:

> `0x8087`, PRF_SPEC, Operation speculatively executed, prefetch
>
> The counter counts each Speculatively executed prefetch operation due to any of the following A64 instructions:
>
> - Scalar: PRFM, PRFUM, or RPRFM.
> - SVE: PRFB, PRFD, PRFH, or PRFW.
>
> It is **IMPLEMENTATION DEFINED** which prefetch operations are counted in AArch32 state.

The equivalent change is made in BASE_PRF_SPEC.

2.202 D22598

In section **D21.4.1 Streaming execution priority for shared implementations** the text that reads:

> R_{WPVQK}
>
> All PEs in a Priority domain have the same value of SMPRI_EL1.Affinity and SMPRI_EL1.Affinity2.
>
> R_{CVLSF}
>
> PEs in different Priority domains have different values of SMPRI_EL1.Affinity or SMIDR_EL1.Affinity 2.

is changed to read:

> R_{WPVQK}
>
> All PEs in a Priority domain have the same value of SMIDR_EL1.Affinity and SMIDR_EL1.Affinity2.
>
> R_{CVLSF}
>
> PEs in different Priority domains have different values of SMIDR_EL1.Affinity or SMIDR_EL1.Affinity 2.

2.203 C22614

In section, **D23.2.47 "GCR_EL1, Tag Control Register"**, under the heading 'RRND, bit [16]', the field description that reads:

> Controls generation of tag values by the IRG instruction.

`0b0`	IRG generates a tag value as defined by RandomTag().

0b1	IRG generates an implementation-specific tag value with a distribution of tag values no worse than generated with GCR_EL1.RRND == 0.

is changed to read:

> Controls generation of tag values by the IRG instruction.

0b0	IRG generates a tag value as defined by RandomTag() and ChooseNonExcludedTag(). This mode does not provide strong guarantees for randomness and should only be used for debugging purposes.
0b1	IRG generates an implementation-specific tag value with a distribution of tag values no worse than generated with GCR_EL1.RRND == 0.

> Note:
>
> Arm recommends that **IMPLEMENTATION DEFINED** algorithms minimize the risk of a bias by selecting tags from a uniform distribution.

In section, **J1.1.3 "aarch64/functions"**, in the pseudocode function ChooseRandomNonExcludedTag(), the text that reads:

```
// This function is permitted to generate a non-deterministic selection from the
set of non-excluded
// Allocation Tags. A reasonable implementation is described by the Pseudocode used
when
// GCR_EL1.RRND is 0, but with a non-deterministic implementation of
NextRandomTagBit().
// Implementations may choose to behave the same as GCR_EL1.RRND=0.
```

is changed to read:

```
// This function is permitted to generate a non-deterministic selection from the
set of non-excluded
// Allocation Tags. A reasonable implementation should select a tag from a uniform
distribution and avoid common pitfalls such as modulo bias.
```

2.204 D22620

In section **D23.2.173 "TCR2_EL2, Extended Translation Control Register (EL2)"**, under the heading 'When the Effective value of HCR_EL2.E2H is 1:', the following field description of 'SKL1, bits [9:8]' is removed:

> When FEAT_D128 is implemented:
>
> Skip Level associated with translation table walks using TTBR1_EL2.
>
> This determines the number of levels to be skipped in the regular start level of the stage 1 EL2&0 translation table walks using TTBR1_EL2.

0b00	Skip 0 level in the regular start level.
0b01	Skip 1 level in the regular start level.

0b10	Skip 2 levels in the regular start level.
0b11	Skip 3 levels in the regular start level.

This field is IGNORED when TCR2_EL2.D128 is 0.

This field is ignored by the PE and treated as zero when EL3 is implemented and SCR_EL3.TCR2En == 0.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to 0.
  - Otherwise, this field resets to an architecturally **UNKNOWN** value.

Otherwise: Reserved, **RES0**.

In the same section, under the heading 'When the Effective value of HCR_EL2.E2H is 1:', the following field description of 'SKL0, bits [7:6]' is removed:

When FEAT_D128 is implemented:

Skip Level associated with translation table walks using TTBR0_EL2.

This determines the number of levels to be skipped in the regular start level of the stage 1 EL2&0 translation table walks using TTBR0_EL2.

0b00	Skip 0 level in the regular start level.
0b01	Skip 1 level in the regular start level.
0b10	Skip 2 levels in the regular start level.
0b11	Skip 3 levels in the regular start level.

This field is ignored by the PE and treated as zero when EL3 is implemented and SCR_EL3.TCR2EN == 0.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to 0.
  - Otherwise, this field resets to an architecturally **UNKNOWN** value.

Otherwise: Reserved, **RES0**.

2.205 D22621

In section **D2.8.3 "Breakpoint types and linking of breakpoints"**, the text that reads:

> When more than 16 context-aware breakpoints are implemented, the number of context-aware breakpoints is identified to software by ID_AA64DFR1_EL1.CTX_CMPs and EDDFR1.CTX_CMPs. Otherwise ID_AA64DFR0_EL1.CTX_CMPs shows how many are implemented. The number of context-aware breakpoints cannot be more than the number of implemented breakpoints, but at least one implemented breakpoint must be context-aware.
>
> If the number of implemented breakpoints is less than or equal to 16, then the context-aware breakpoints are the highest numbered breakpoints.
>
> If the number of implemented breakpoints is greater than 16, then the context-aware breakpoints are numbered consecutively beginning with 0.

is changed to read:

> If more than 16 context-aware breakpoints are implemented, the number of context-aware breakpoints is identified to software by ID_AA64DFR1_EL1.CTX_CMPs and EDDFR1.CTX_CMPs. Otherwise ID_AA64DFR0_EL1.CTX_CMPs indicates how many are implemented. At least one breakpoint must be context-aware.
>
> If 16 or fewer breakpoints are implemented, the highest numbered are context-aware.
>
> If more than 16 breakpoints are implemented:
>
> - If 16 or fewer are context-aware, these are the highest numbered breakpoints up to breakpoint number 15.
> - Otherwise, context-aware breakpoints are numbered consecutively beginning at breakpoint number 0.

2.206 D22625

In section **A2.2.1 "The Armv8.0 architecture extension"**, under the heading 'FEAT_CSV2_1p1, Cache Speculation Variant 2', the text that reads:

> The following fields identify the presence of FEAT_CSV2_1p1:
>
> - ID_AA64PFR1_EL1.CSV2_frac.
> - ID_AA64PFR0_EL1.CSV2.

is changed to read:

> The following fields identify the presence of FEAT_CSV2_1p1:
>
> - ID_AA64PFR1_EL1.CSV2_frac.
> - ID_AA64PFR0_EL1.CSV2.

- ID_PFR0_EL1.CSV2.
- ID_PFR0.CSV2.

2.207 D22636

In section **J1.1.1 "aarch64/debug"**, in the pseudocode function AArch64.TakeExceptionInDebugState(), the following code segment is removed:

```
if IsFeatureImplemented(FEAT_TME) && TSTATE.depth > 0 then
     TMFailure cause;
     case except.exceptype of
         when Exception_SoftwareBreakpoint cause = TMFailure_DBG;
         when Exception_Breakpoint         cause = TMFailure_DBG;
         when Exception_Watchpoint         cause = TMFailure_DBG;
         when Exception_SoftwareStep       cause = TMFailure_DBG;
         otherwise                         cause = TMFailure_ERR;
     FailTransaction(cause, FALSE);

 boolean brbe_source_allowed = FALSE;
 bits(64) brbe_source_address = Zeros(64);
 if IsFeatureImplemented(FEAT_BRBE) then
     brbe_source_allowed = BranchRecordAllowed(PSTATE.EL);
     brbe_source_address = bits(64) UNKNOWN;
```

2.208 D22637

In section **I6.9.33 "ERR<n>STATUS, Error Record <n> Primary Status Register, n = 0 - 65534"**, the text that reads:

> Bits [17:16]
>
> Reserved, **RES0**.

is changed to read:

> RV2, bit [17]
>
> When RAS System Architecture v2 is implemented: Reset Valid 2. When ERR<n>STATUS.{V, RV} is {1, 1}, indicating the error record is valid and one or more errors were recorded before the last Error Recovery reset, this field indicates whether any lower severity errors have been recorded after the Error Recovery reset that did not overwrite the syndrome.

RV2	Meaning
0b0	If the error record is valid then one or more errors were recorded after the last Error Recovery reset that did not overwrite the error syndrome. This includes errors that did not overwrite a previously recorded error syndrome.
0b1	If the error record is valid then one or more errors were recorded before the last Error Recovery reset.

> This field is set to 0 when an error is recorded, including when the fault does not overwrite a previously recorded syndrome.

> The reset behavior of this field is:
>
> On an Error recovery reset, this field resets to ‘1’.
>
> Access to this field is W1C.
>
> Otherwise:
>
> Reserved, **RES0**.
>
> Bit [16]
>
> Reserved, **RES0**.

2.209 D22640

In the “Product Status” section on page iii, the text that reads:

> The information in this document is final, that is for a developed product. The information in this manual is at EAC quality, which means that all features of the specification are described in the manual.
>
> This document includes the A-profile system registers, instructions and pseudocode corresponding to the 2022-12 version of the A-profile XML published on developer.arm.com. The register descriptions relating to feature FEAT_MEC are at Alpha quality.
>
> Alpha quality means that most major features of the specification are described in the manual, some features and details might be missing.

is changed to read:

> The information in this document is final, that is for a developed product. The information in this manual is at EAC quality, which means that all features of the specification are described in the manual.
>
> This document includes the A-profile system registers, instructions and pseudocode corresponding to the 2023-09 version of the A-profile XML published on developer.arm.com.
>
> For more recent changes to the instruction and register XML content, see https://developer.arm.com/architectures/cpu-architecture/a-profile/exploration-tools

2.210 D22647

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

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

is changed to read:

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

2.211 D22658

In section **A2.2.5 "The Armv8.4 architecture extension"**, under the heading "FEAT_SEL2, Secure EL2", the text that reads:

> In an Armv8.4 implementation, if FEAT_AA64EL2 is implemented, FEAT_SEL2 is implemented.

is changed to read:

> In an Armv8.4 implementation, if FEAT_AA64EL2 and Secure state are implemented, FEAT_SEL2 is implemented.

2.212 D22660

In section **D23.3.8 "DBGDTRTX_EL0, Debug Data Transfer Register, Transmit"**, in field 'Bits [31:0]', the text that reads:

> Return DTRTX. Writes to this register:
>
> - If TXfull is set to 1, set DTRRX and DTRTX to **UNKNOWN**.
> - If TXfull is set to 0, update the value in DTRTX.
>
> After the write, TXfull is set to 1.

is changed to read:

> DTRTX. Writes to this register:
>
> - If TXfull is 1, DTRTX is **UNKNOWN**.
> - If TXfull is 0, update the value in DTRTX.

After the write, TXfull is set to 1.

2.213 D22665

In section **D8.15 "Translation Lookaside Buffers"**, the following text is added:

R_{X0001}

A translation table entry is not permitted to be cached in a TLB or intermediate caching structure if a synchronous external abort on the translation table walk prevents the PE from determining if the translation table entry would otherwise cause a Translation fault, an Address size fault, or an Access flag fault.

In section **B2.3 "Definition of the Arm memory model"**, under the definition for 'TLBUncacheable', the following bullet point is added:

- A synchronous External Abort on translation table walk, such that it cannot be determined whether the TTD would otherwise generate a Translation Fault, Access Flag Fault, or Address Size Fault effect.

2.214 D22667

In section **"D23.2.41 ESR_EL2, Exception Syndrome Register (EL2)"**, in the field description 'ISS2, bits [55:32]' under the heading 'ISS2 encoding for an exception from an Instruction Abort', the text that reads:

Bits [5:0]

Reserved, **RES0**.

is changed to read:

DirtyBit, bit [5] When FEAT_S1PIE is implemented or FEAT_S2PIE is implemented: DirtyBit flag.

If a write access to memory generates an Instruction Abort for a Permission fault using Indirect Permission, this field holds information about the fault.

0b0	Permission Fault is not due to dirty state.
0b1	Permission Fault is due to dirty state.

For any other fault or Access, this field is **RES0**.

The reset behavior of this field is:

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

Otherwise: Reserved, **RES0**.

Bits [4:0] Reserved, **RES0**.

Note:

This change is not made to ESR_EL1 and ESR_EL3

2.215 D22675

In section **D20.26.2 "Alternative PARTID spaces and selection"**, the text that reads:

The alternative PARTID Space feature, ALTSP, defines alternative PARTID spaces for each of the Security states.

MPAM3_EL3 and MPAM2_EL2 have fields to control whether the primary or alternative PARTID space is used at those Exception levels and lower Exception levels.

is changed to read:

The alternative PARTID Space feature, ALTSP, defines alternative PARTID spaces for each of the Security states.

MPAM3_EL3 and MPAM2_EL2 have fields to control whether the primary or alternative PARTID space is used at those Exception levels and lower Exception levels.

When MPAM is disabled from EL3 (MPAM3_EL3.MPAMEN==0, see MPAM enable), fields MPAM3_EL3.RT_ALTSP_NS, MPAM3_EL3.ALTSP_EL3, MPAM3_EL3.ALTSP_HFC and MPAM3_EL3.ALTSP_HEN are effective, and fields MPAM2_EL2.ALTSP_EL2 and MPAM2_EL2.ALTSP_HFC are treated as 0.

In section **D23.12.2 MPAM1_EL1, MPAM1 Register (EL1)**, in the field description of 'ALTSP_FRCD, bit [54]', the text that reads:

Access to this field is RO.

is changed to read:

Accessing this field has the following behavior:

- When !UsePrimarySpaceEL10(), access to this field is **RAO/WI**.
- Otherwise, access to this field is **RAZ/WI**.

In section **D23.12.3 MPAM2_EL2, MPAM2 Register (EL2)**, the text that reads:

Access to this field is RO.

is changed to read:

Accessing this field has the following behavior:

- When !UsePrimarySpaceEL2(), access to this field is **RAO/WI**.

- Otherwise, access to this field is **RAZ/WI**.

2.216 D22677

In **D23.2.135 "PIRE0_EL2, Permission Indirection Register 0 (EL2)"**, in the pseudocode for the 'MRS <Xt> PIRE0_EL2' EL1 accessor, the text that reads:

```
if EffectiveHCR_EL2_NVx() IN {'1x1'} then
    X[t, 64] = NVMem[0x298];
elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
    AArch64.SystemAccessTrap(EL2, 0x18);
else
    UNDEFINED;
```

is changed to read:

```
if EffectiveHCR_EL2_NVx() IN {'xx1'} then
     AArch64.SystemAccessTrap(EL2, 0x18);
else
     UNDEFINED;
```

The equivalent change is made in the pseudocode for the 'MSR PIRE0_EL2, <Xt>' EL1 accessor.

In **D8.13.6.2 "Loads and stores generated by transforming register accesses"**, in table 'D8-107 Memory address offset associated with transformed register access', the following line is removed:

PIRE0_EL1 PIRE0_EL2 `0x298`

2.217 C22688

In section **B2.13.2 "Alignment of data accesses"**, the text that reads:

> For an access where the address that is accessed is not aligned to the size of the data element being accessed, if the value of SCTLR_ELx.A applicable to the current Exception level is 1, an Alignment fault is generated.
>
> If FEAT_LSE2 is implemented, Load-Acquire/Store-Release instructions without exclusive or atomic behavior generate an Alignment fault if all of the following apply:
>
> - Not all bytes of the memory access lie within a 16-byte quantity aligned to 16 bytes.
> - The value of SCTLR_ELx.nAA applicable to the current Exception level is 0.

is added at the beginning of the section:

> For an unaligned access to any type of Device memory: If the memory location cannot support unaligned accesses then an Alignment fault is generated. If the memory location supports unaligned accesses, then it is **IMPLEMENTATION DEFINED** whether the access generates an

> Alignment fault if the location would not generate an Alignment fault if the same access were made to Normal memory.

In section **B2.13.2.1.3 "Non-atomic Load-Acquire/Store-Release instructions"**, the text that reads:

> For Load-Acquire/Store-Release instructions that do not have exclusive or atomic behaviors: When the value of SCTLR_ELx.A applicable to the current Exception level is 1, an Alignment fault is generated. When the value of SCTLR_ELx.A applicable to the current Exception level is 0:
>
> If FEAT_LSE2 is not implemented, these instructions generate an Alignment fault if the address being accessed is not aligned to the size of the data structure being accessed.
>
> If FEAT_LSE2 is implemented, then:
>
> - If the memory access is not to Normal Inner Write-Back or Outer Write-Back Cacheable memory, then it is a **CONSTRAINED UNPREDICTABLE** choice of either of the following:
>   - An unaligned access is performed meeting all of the semantics of the instruction.
>   - An Alignment fault is generated. If all of the bytes of the memory access do not lie within a 16-byte quantity aligned to 16 bytes then the following applies:
>   - If SCTLR_ELx.nAA applicable to the current Exception level is 0, an Alignment fault is generated.
>   - If SCTLR_ELx.nAA applicable to the current Exception level is 1, then an unaligned access is performed which is not guaranteed to be single-copy atomic except at the byte access level.
>
> In this case, the architecture does no define the order of the different transactions of the access defined by the single instructions relative to each other.

is changed to read:

> For Load-Acquire/Store-Release instructions that do not have exclusive or atomic behaviors, when the address being accessed is not aligned to the size of the data structure being accessed and the value of SCTLR_ELx.A applicable to the current Exception level is 0:
>
> If FEAT_LSE2 is not implemented, these instructions generate an Alignment fault.
>
> If FEAT_LSE2 is implemented, then:
>
> - If the memory access is not to Normal Inner Write-Back or Outer Write-Back Cacheable memory, then it is a **CONSTRAINED UNPREDICTABLE** choice of either of the following:
>   - An unaligned access is performed meeting all of the semantics of the instruction.
>   - An Alignment fault is generated.
> - If all of the bytes of the memory access do not lie within a 16-byte quantity aligned to 16 bytes and SCTLR_ELx.nAA applicable to the current Exception level is 1, then an unaligned access is performed which is not guaranteed to be single-copy atomic except at the byte access level.

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

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

> For Load-Exclusive/Store-Exclusive, and Atomic instructions including those with acquire or acquire-release semantics:
>
> When the value of SCTLR_ELx.A applicable to the current Exception level is 1, an Alignment fault is generated.
>
> When the value of SCTLR_ELx.A applicable to the current Exception level is 0:
>
> If FEAT_LSE2 is not implemented, then these instructions generate an Alignment fault if the address being accessed is not aligned to the size of the data structure being accessed

is changed to read:

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

2.218 E22689

In **D23.2.69 "ID_AA64DFR0_EL1, AArch64 Debug Feature Register 0"**, in the field description of 'BRPs, bits [15:12]', the text that reads:

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

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

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

is changed to read:

> Index of the highest numbered context-aware breakpoint. Identifies which of the implemented breakpoints are context-aware breakpoints.
>
> The value of this field is an **IMPLEMENTATION DEFINED** choice of:

`0b0001...0b1110`	Index of the highest numbered context-aware breakpoint.

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

> If ID_AA64DFR1_EL1.BRPs is zero, then this is also the number of implemented breakpoints, minus 1, meaning the context-aware breakpoints are the highest numbered breakpoints.
>
> The value of this field is always less than the number of implemented breakpoints.

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

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

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

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

is changed to read:

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

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

> Values greater than ID_AA64DFR0_EL1.BRPs are not permitted.

In **D23.2.70 "ID_AA64DFR1_EL1, AArch64 Debug Feature Register 1"**, in the field description of 'BRPs, bits [15:8]', the following text is added:

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

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

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

is changed to read:

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

2.219 C22691

In section **D23.2.53 "HCR_EL2, Hypervisor Configuration Register"**, in the field description of bit [23], the text that reads:

> Bit [23]
>
> When FEAT_DPB is implemented:
>
> TPCP
>
> ...
>
> Otherwise:
>
> TPC
>
> ...

is changed to read:

> TPCP, bit [23]
>
> ...
>
> Note:
>
> This field was previously referred to as TPC.

2.220 C22693

In section **D1.3.5.4.2 "SVE First-fault and Non-fault loads"**, the rule R_{WCHSR} that reads:

> In the previous table, watchpoints are not a mechanism for preventing access to memory.

is changed to read:

> In the previous table, watchpoints and MTE Tag check faults are not a mechanism for preventing access to memory.

2.221 D22703

In section **J1.1.3 "aarch64/functions"**, in the pseudocode function Arch64.FaultSyndrome(), the code segment that reads:

```
        if exceptype == Exception_Watchpoint then
           iss<23:0> = WatchpointRelatedSyndrome(fault, vaddress);
```

Is changed to read:

```
        if exceptype IN {Exception_Watchpoint, Exception_NV2Watchpoint} then
           iss<23:0> = WatchpointRelatedSyndrome(fault, vaddress);
```

2.222 D22719

In **C5.5.89 "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 **C5.5.83 "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 **C5.5.125 "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 **D8.16.5.3 "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.

2.223 D22725

In section **D23.2.54 "HCRX_EL2, Extended Hypervisor Configuration Register"** in the definition of field 'EnFPM, bit [23]', the text that reads:

> Enables the following accesses to FPMR from EL0 and EL1 to EL2:
>
> - Direct accesses reported with EC syndrome value `0x18`.
> - Indirect reads as a result of execution of an FP8 instruction reported with EC syndrome value `0x00`.

EnFPM	Meaning 2
`0b0`	EL1 and EL0 accesses to FPMR are disabled and trapped to EL2.
`0b1`	This control does not cause any instructions to be trapped.

> Traps are not taken if there is a higher priority exception generated by the access.

is changed to read:

> Enables direct and indirect accesses to FPMR from EL0 and EL1. When accesses to FPMR are disabled by this control:
>
> - Direct accesses to FPMR from EL0 or EL1 are trapped to EL2 and reported with EC syndrome value `0x18`.
> - Execution of FP8 data-processing instructions that indirectly access FPMR is **UNDEFINED** at EL0 or EL1.

EnFPM	Meaning 2
`0b0`	Direct and indirect accesses to FPMR are disabled at EL1 and EL0.
`0b1`	This control does not cause any instructions to be trapped.

> Traps are not taken if there is a higher priority exception generated by the access.

2.224 D22736

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

```
constant integer ovflw = if HaveAArch32() && lc == '1' then 64 else 32; if
 old_value<64:ovflw> != new_value<64:ovflw> then
...
```

is changed to read:

```
constant integer ovflw = if HaveAArch32() && lc == '0' then 32 else 64; if
 old_value<64:ovflw> != new_value<64:ovflw> then
...
```

2.225 D22739

In section **C5.5.38 "TLBI RVAE2, TLBI RVAE2NXS, TLB Range Invalidate by VA, EL2"**, under the heading 'TLBI RVAE2{, <Xt>}', all occurrences of VMID[] are replaced with VMID_NONE, therefore the code for operations in EL2 and EL3 that reads:

> AArch64.TLBI_RVA(SecurityStateAtEL(EL2), Regime_EL2, VMID[], Broadcast_NSH, TLBILevel_Any, TLBI_AllAttr, X[t, 64]);

is changed to read:

> AArch64.TLBI_RVA(SecurityStateAtEL(EL2), Regime_EL2, VMID_NONE, Broadcast_NSH, TLBILevel_Any, TLBI_AllAttr, X[t, 64]);

In the same section, under the heading 'TLBI RVAE2NXS{, <Xt>}', the operation pseudocode that reads:

> AArch64.TLBI_RVA(SecurityStateAtEL(EL2), Regime_EL2, VMID[], Broadcast_NSH, TLBILevel_Any, TLBI_ExcludeXS, X[t, 64]);

is changed to read:

> AArch64.TLBI_RVA(SecurityStateAtEL(EL2), Regime_EL2, VMID_NONE, Broadcast_NSH, TLBILevel_Any, TLBI_ExcludeXS, X[t, 64]);

The equivalent changes are made in:

- TLBI RVAE2IS, TLBI RVAE2ISNXS, TLBI RVAE2OS, TLBI RVAE2OSNXS.
- TLBI RVALE2, TLBI RVALE2NXS, TLBI RVALE2IS, TLBI RVALE2ISNXS, TLBI RVALE2OS, TLBI RVALE2OSNXS.
- TLBIP RVAE2, TLBIP RVAE2NXS, TLBIP RVAE2IS, TLBIP RVAE2ISNXS, TLBIP RVAE2OS, TLBIP RVAE2OSNXS.
- TLBIP RVALE2, TLBIP RVALE2NXS, TLBIP RVALE2IS, TLBIP RVALE2ISNXS, TLBIP RVALE2OS, TLBIP RVALE2OSNXS.

and in the EL3 accessors for:

- TLBI RVAE3, TLBI RVAE3NXS, TLBI RVAE3IS, TLBI RVAE3ISNXS, TLBI RVAE3OS, TLBI RVAE3OSNXS.
- TLBI RVALE3, TLBI RVALE3NXS, TLBI RVALE3IS, TLBI RVALE3ISNXS, TLBI RVALE3OS, TLBI RVALE3OSNXS.
- TLBIP RVAE3, TLBIP RVAE3NXS, TLBIP RVAE3IS, TLBIP RVAE3ISNXS, TLBIP RVAE3OS, TLBIP RVAE3OSNXS.
- TLBIP RVALE3, TLBIP RVALE3NXS, TLBIP RVALE3IS, TLBIP RVALE3ISNXS, TLBIP RVALE3OS, TLBIP RVALE3OSNXS.

In section **C5.5.80 "TLBI VMALLS12E1, TLBI VMALLS12E1NXS, TLB Invalidate by VMID, All at Stage 1 and 2, EL1**, under the heading 'TLBI VMALLS12E1{, <Xt>}' the following code segment that reads:

```
elsif PSTATE.EL == EL2 then
    AArch64.TLBI_VMALLS12(SecurityStateAtEL(EL1), Regime_EL10, VMID[],
 Broadcast_NSH, TLBI_AllAttr, X[t, 64]);
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        AArch64.TLBI_VMALL(SecurityStateAtEL(EL1), Regime_EL10, VMID[],
 Broadcast_NSH, TLBI_AllAttr, X[t, 64]);
    else
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL1) then
            return;
        else
            AArch64.TLBI_VMALLS12(SecurityStateAtEL(EL1), Regime_EL10, VMID[],
 Broadcast_NSH, TLBI_AllAttr, X[t, 64]);
```

is changed to read:

```
elsif PSTATE.EL == EL2 then
    AArch64.TLBI_VMALLS12(SecurityStateAtEL(EL1), Regime_EL10, VMID[],
 Broadcast_NSH, TLBI_AllAttr, X[t, 64]);
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        AArch64.TLBI_VMALL(SecurityStateAtEL(EL1), Regime_EL10, VMID_NONE,
 Broadcast_NSH, TLBI_AllAttr, X[t, 64]);
    else
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL1) then
            return;
        else
            AArch64.TLBI_VMALLS12(SecurityStateAtEL(EL1), Regime_EL10, VMID[],
 Broadcast_NSH, TLBI_AllAttr, X[t, 64]);
```

The equivalent changes are made under the heading 'TLBI VMALLS12E1NXS{, <Xt>}' and under the following:

- TLBI VMALLS12E1IS, TLBI VMALLS12E1ISNXS.
- TLBI VMALLS12E1OS, TLBI VMALLS12E1OSNXS.

2.226 C22740

In section **D23.7.11 "PMSIDR_EL1, Sampling Profiling ID Register"**, under the heading 'FDS, bit [7]', the text that reads:

> Filter by data source. The value of this field is an IMPLEMENTATION DEFINED choice of:

0b0	PMSDSFR_EL1 is not implemented and PMSFCR_EL1.FDS is **RES0**.
0b1	PMSDSFR_EL1 and PMSFCR_EL1.FDS are implemented.

> FEAT_SPE_FDS implements the functionality identified by the value 1.
>
> Access to this field is RO.

is changed to read:

> When FEAT_SPEv1p4 is implemented
>
> Filter by data source. The value of this field is an IMPLEMENTATION DEFINED choice of:

0b0	PMSDSFR_EL1 is not implemented and PMSFCR_EL1.FDS is **RES0**.
0b1	PMSDSFR_EL1 and PMSFCR_EL1.FDS are implemented.

> FEAT_SPE_FDS implements the functionality identified by the value 1.
>
> Access to this field is RO.
>
> Otherwise
>
> **RES0**.

In section **D23.7.11 "PMSIDR_EL1, Sampling Profiling ID Register"**, under the heading 'FnE', the text that reads:

> Filtering by events, inverted.
>
> The value of this field is an **IMPLEMENTATION DEFINED** choice of:

0b0	PMSNEVFR_EL1 is not implemented and PMSFCR_EL1.FnE is **RES0**.
0b1	PMSNEVFR_EL1 and PMSFCR_EL1.FnE are implemented.

> FEAT_SPEv1p2 implements the functionality identified by the value 1.
>
> Access to this field is RO.

is changed to read:

> When FEAT_SPEv1p2 is implemented
>
> Filtering by events, inverted.
>
> The value of this field is an **IMPLEMENTATION DEFINED** choice of:

0b0	PMSNEVFR_EL1 is not implemented and PMSFCR_EL1.FnE is **RES0**.
0b1	PMSNEVFR_EL1 and PMSFCR_EL1.FnE are implemented.

> FEAT_SPE_FnE implements the functionality identified by the value 1.
>
> Access to this field is RO.

In section **A2.2.8 "The Armv8.7 architecture extension"**, under the heading 'FEAT_SPEv1p2, Statistical Profiling Extensions version 1.2', the text that reads:

> FEAT_SPEv1p2 adds the following features to the Statistical Profiling Extension:

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

FEAT_SPEv1p2 optionally enables support for a packet for each taken branch that provides the target address for the previous taken branch.

If FEAT_SPEv1p2 is implemented, PMSIDR_EL1.PBT indicates support for the previous branch target packet.

This feature is supported in AArch64 state only.

FEAT_SPEv1p2 is OPTIONAL from Armv8.6.

If FEAT_SPEv1p2 is implemented, then FEAT_SPEv1p1 is implemented.

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

The following fields identify the presence of FEAT_SPEv1p2:

- ID_AA64DFR0_EL1.PMSVer.
- PMSIDR_EL1.FnE.

For more information, see:

- Freezing PMU counters.
- Common event numbers.
- Filtering sample records.
- Previous branch target.
- About the Statistical Profiling Extension sample records.
- Address packet

is changed to read:

FEAT_SPEv1p2 adds the following features to the Statistical Profiling Extension:

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

This feature is supported in AArch64 state only.

FEAT_SPEv1p2 is OPTIONAL from Armv8.6.

If FEAT_SPEv1p2 is implemented, then FEAT_SPEv1p1 is implemented.

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

The ID_AA64DFR0_EL1.PMSVer field identifies the presence of FEAT_SPEv1p2.

For more information, see:

- Freezing PMU counters.
- Common event numbers.
- About the Statistical Profiling Extension sample records.

FEAT_SPE_FnE, Statistical Profiling inverse event filter control

FEAT_SPE_FnE adds the PMSFCR_EL1.FnE field and PMSNEVFR_EL1 register that enable filtering of sample records by the inverse values of bits in the sampled Events packets.

This feature is supported in AArch64 state only.

FEAT_SPE_FnE is OPTIONAL.

FEAT_SPE_FnE is implemented if and only if FEAT_SPEv1p2 is implemented.

The PMSIDR_EL1.FnE field identifies the presence of FEAT_SPE_FnE.

For more information, see Filtering sample records.

FEAT_SPE_PBT, Statistical Profiling previous branch target

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

This feature is supported in AArch64 state only.

FEAT_SPE_PBT is OPTIONAL.

If FEAT_SPE_PBT is implemented, then FEAT_SPEv1p2 is implemented.

The PMSIDR_EL1.PBT field identifies the presence of FEAT_SPE_PBT.

For more information, see:

- Previous branch target.
- Address packet.

2.227 D22755

In the section **C5.2 "Special-purpose registers"**, the list of Special-purpose registers is changed to include a new entry:

This section describes the following Special-purpose registers:

- ...
- PM, that provides access to the Profiling exception mask bit.

2.228 C22762

In section **A2.2.7 "The Armv8.6 architecture extension"**, under the heading 'FEAT_BF16, AArch64 BFloat16 instructions', the text that reads:

> FEAT_BF16 is OPTIONAL from Armv8.2.
>
> FEAT_BF16 is mandatory from Armv8.6.

is changed to read:

> FEAT_BF16 is OPTIONAL from Armv8.2.
>
> In an Armv8.6 implementation, if FEAT_FP is implemented, FEAT_BF16 is implemented.

2.229 C22763

In section **D23.2.74 "ID_AA64MMFR0_EL1, AArch64 Memory Model Feature Register 0"**, under the heading 'TGran4_2, bits [43:40]', the text that reads:

> The `0b0000` value is deprecated.

is changed to read:

> If EL2 is not implemented, this field is `0b0000`. Otherwise, the value `0b0000` is deprecated.

The equivalent changes are made in the following sections:

- **D23.2.74 "ID_AA64MMFR0_EL1, AArch64 Memory Model Feature Register 0"**, under the heading 'TGran64_2, bits [39:36]'
- **D23.2.74 "ID_AA64MMFR0_EL1, AArch64 Memory Model Feature Register 0"**, under the heading 'TGran16_2, bits [35:32]'

2.230 C22764

In section **J1.1.3 "aarch64/functions"**, under the heading 'aarch64/functions/tlbi/TLBIMatch', the code that reads:

```
boolean TLBIMatch(TLBIRecord tlbi, TLBRecord tlb_entry)
    …
    case tlbi.op of
       …
        when TLBIOp_VA, TLBIPOp_VA
            constant integer addr_lsb = tlb_entry.blocksize;
            match = (tlb_entry.context.includes_s1 &&
                     …
```

```
                    (!UseVMID(tlb_entry.context) || tlbi.vmid ==
 tlb_entry.context.vmid) &&
                    (!UseASID(tlb_entry.context) || tlbi.asid ==
 tlb_entry.context.asid ||
                      …
                    (tlbi.level == TLBILevel_Any || !tlb_entry.walkstate.istable));
        ...
    ...
```

is changed to read:

```
boolean TLBIMatch(TLBIRecord tlbi, TLBRecord tlb_entry)
    …
    case tlbi.op of
       …
        when TLBIOp_VA, TLBIPOp_VA
            constant integer addr_lsb = tlb_entry.blocksize;
            match = (tlb_entry.context.includes_s1 &&
                     …
                     tlbi.use_vmid == tlb_entry.context.use_vmid &&
                     (!tlb_entry.context.use_vmid || tlbi.vmid ==
 tlb_entry.context.vmid) &&
                     (!UseASID(tlb_entry.context) || tlbi.asid ==
 tlb_entry.context.asid ||
                       …
                     (tlbi.level == TLBILevel_Any || !tlb_entry.walkstate.istable));
        ...
    ...
```

Similar changes are made for the following cases of tlbi.op in the TLBIMatch function:

- TLBIOp_DASID, TLBIOp_IASID
- TLBIOp_DVA, TLBIOp_IVA
- TLBIOp_ASID
- TLBIOp_VAA, TLBIPOp_VAA
- TLBIOp_VA, TLBIPOp_VA
- TLBIOp_VMALL
- TLBIOp_RVAA, TLBIPOp_RVAA
- TLBIOp_RVA, TLBIPOp_RVA

In section **J1.1.3 “aarch64/functions”**, under the heading ‘aarch64/functions/tlbi/AArch64.TLBI_VA’, the code that reads:

```
AArch64.TLBI_VA(SecurityState security, Regime regime, bits(16) vmid,
                Broadcast broadcast, TLBILevel level,  TLBIMemAttr attr, bits(64)
 Xt)
    …
    r.vmid         = vmid;
    …
```

is changed to read:

```
AArch64.TLBI_VA(SecurityState security, Regime regime, bits(16) vmid,
```

```
                Broadcast broadcast, TLBILevel level,  TLBIMemAttr attr, bits(64)
Xt)
    …
    r.vmid         = vmid;
    r.use_vmid     = UseVMID(regime);
    …
```

Similar changes are done to

- AArch64.TLBI_VMALL()
- AArch64.TLBI_ASID()
- AArch64.TLBI_VA()
- AArch64.TLBIP_VA()
- AArch64.TLBI_VAA()
- AArch64.TLBIP_VAA()
- AArch64.TLBI_IPAS2()
- AArch64.TLBIP_IPAS2()
- AArch64.TLBI_RVA()
- AArch64.TLBIP_RVA()
- AArch64.TLBI_RVAA()
- AArch64.TLBIP_RVAA()
- AArch64.TLBI_RIPAS2()
- AArch64.TLBIP_RIPAS2()
- AArch32.DTLBI_ASID()
- AArch32.DTLBI_VA()
- AArch32.ITLBI_ASID()
- AArch32.ITLBI_VA()
- AArch32.TLBI_VMALL()
- AArch32.TLBI_VMALLS12()
- AArch32.DTLBI_ASID()
- AArch32.DTLBI_VA()
- AArch32.ITLBI_ASID()
- AArch32.ITLBI_VA()
- AArch32.TLBI_VMALL()
- AArch32.TLBI_ASID()
- AArch32.TLBI_VA()
- AArch32.TLBI_VAA()
- AArch32.TLBI_IPAS2()

In section **J1.1.4 "aarch64/translation"**, under the heading 'aarch64/translation/vmsa_tlbcontext/AArch64.GetS1TLBContext', the code that reads:

```
TLBContext AArch64.GetS1TLBContext(Regime regime, SecurityState ss, bits(64) va, TGx
 tg)
    TLBContext tlbcontext;
    …
    tlbcontext.includes_s2  = FALSE;
    …
    return tlbcontext;
end
```

is changed to read:

```
TLBContext AArch64.GetS1TLBContext(Regime regime, SecurityState ss, bits(64) va, TGx
 tg)
    TLBContext tlbcontext;
    …
    tlbcontext.includes_s2  = FALSE;
    tlbcontext.use_vmid     = UseVMID(regime);
    …
    return tlbcontext;
end
```

A similar change is done under the heading 'aarch64/translation/vmsa_tlbcontext/AArch64.GetS2TLBContext'.

2.231 D22765

In section **B2.3.2.1 "Intrinsic data, control and order dependencies"**, under the heading B2.3.2.1.1 'Conditional Selection Instructions', the text that reads:

> CSEL instruction success case.
>
> If the condition cond is true all of the following apply to the effects generated by CSEL X_0, X_1, X_2, cond instruction:
>
> - Ct_1: There is an Intrinsic control dependency from the Register Read effect of PSTATE.NZCV to the Register Read effect of X_1.
> - Dt_1: There is an Intrinsic data dependency from the Register Read effect of X_0 to the Register Write effect of X_0.

is changed to read:

> CSEL instruction success case.
>
> If the condition cond is true all of the following apply to the effects generated by CSEL X_0, X_1, X_2, cond instruction:
>
> - Ct_1: There is an Intrinsic control dependency from the Register Read effect of PSTATE.NZCV to the Register Read effect of X_1.

- Dt_1: There is an Intrinsic data dependency from the Register Read effect of X_1 to the Register Write effect of X_0.

2.232 D22775

In section **D23.3.20 "MDSCR_EL1, Monitor Debug System Control Register"**, the reset behavior for the following fields is changed to match their corresponding fields in H9.2.44 "EDSCR, External Debug Status and Control Register":

- TFO, bit [31].
- RXfull, bit [30].
- TXfull, bit [29].
- RXO, bit [27].
- TXU, bit [26].
- INTdis, bits [23:22].
- TDA, bit [21].
- SC2, bit [19].
- HDE, bit [14].
- ERR, bit [6].

The equivalent changes are made in section **G8.3.14 "DBGDSCRext, Debug Status and Control Register, External View"** for the TFO and SC2 fields.

2.233 R22782

In section **D23.8.7 "BRBSRC<n>_EL1, Branch Record Buffer Source Address Register <n>, n = 0 - 31"**, under the heading 'ADDRESS, bits [63:0]', the text that reads:

> Source virtual address of the Branch record. When an indirect write occurs with a value with ADDRESS bits [63:P] being other than all zeroes or all ones, an **UNKNOWN** value which is not all zeroes or all ones is written to bits [63:P]. P is defined as:
>
> - 56, when FEAT_LVA3 is implemented.
> - 52, when FEAT_LVA is implemented.
> - 48, otherwise.
>
> The value in bits [P-1:0] is the value written.
>
> When an indirect write occurs with a value with ADDRESS bits [63:P] being all zeroes or all ones, the written value is written to bits [63:0], and a read of the register returns the written value.

is changed to read:

> Source virtual address of the Branch record. When an indirect write occurs with a value with ADDRESS bits [63:P] indicating an invalid address, an **UNKNOWN** value which indicates an invalid address is written to bits [63:P]. An invalid address is:
>
> - For an address in a translation regime with 2 VA ranges, with bits [63:P] not all zeroes or all ones.
> - For an address in a translation regime with a single VA range, with bits [63:P] not all zeroes.
>
> P is defined as:
>
> - 56, when FEAT_LVA3 is implemented.
> - 52, when FEAT_LVA is implemented.
> - 48, otherwise.
>
> The value in bits [P-1:0] is the value written. When an indirect write occurs with a value with ADDRESS bits [63:P] indicating a valid address, the written value is written to bits [63:0], and a read of the register returns the written value.

The equivalent changes are made in the following sections:

- **D23.8.9 "BRBTGT<n>_EL1, Branch Record Buffer Target Address Register <n>, n = 0 - 31"**, under the heading 'ADDRESS, bits [63:0]'
- **D23.8.8 "BRBSRCINJ_EL1, Branch Record Buffer Source Address Injection Register"**, under the heading 'ADDRESS, bits [63:0]'
- **D23.8.10 "BRBTGTINJ_EL1, Branch Record Buffer Target Address Injection Register"**, under the heading 'ADDRESS, bits [63:0]'

2.234 D22789

In section **C5.2.25 "SSBS, Speculative Store Bypass Safe"**, under the heading 'Configurations', the text that reads:

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

is changed to read:

> This register is present only when FEAT_SSBS2 is implemented. Otherwise, direct accesses to SSBS are **UNDEFINED**.

2.235 D22801

In section **D23.12.2 "MPAM1_EL1, MPAM1 Register (EL1)"**, under the heading 'MRS <Xt>, MPAM1_EL12', the following pseudocode segment that reads:

```
elsif PSTATE.EL == EL1 then
```

```
    if EffectiveHCR_EL2_NVx() == '101' then
        X[t, 64] = NVMem[0x900];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        if HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
```

is changed to read:

```
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() == '101' then
        X[t, 64] = NVMem[0x900];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        if HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
```

A similar change is made under the heading 'MSR MPAM1_EL12, <Xt>'

2.236 D22802

In section **D23.2.41 "ESR_EL2, Exception Syndrome Register (EL2)"**, under the heading 'ISS2 encoding for an exception from a Data Abort', in the field description for "AssuredOnly, bit [7]", the text that reads:

> AssuredOnly flag.
>
> If a memory access generates a stage 2 Data Abort for a Permission fault, this field holds information about the fault.

is changed to read:

> AssuredOnly flag.
>
> If a memory access generates a Data Abort for a stage 2 Permission fault, this field holds information about the fault.
>
> If this field is 1 and ESR_EL2.GCS is also 1 then the AssuredOnly check might have been the result of VTCR_EL2.GCSH configuration.

Equivalent changes are made to section **D23.2.40 "ESR_EL1, Exception Syndrome Register (EL1)"**.

2.237 D22811

In the section **A1.5.8.2 "Input Denormal Exceptions"**, the text that reads:

> The cumulative floating-point exception bit FPSR.IDC, and the trap enable bit FPCR.IDE both relate to Input Denormal exceptions.
>
> If a denormalized input is flushed to zero, the occurrence of the Input Denormal exception is determined using the value before flushing.
>
> If a denormalized input is flushed to zero, and FPCR.AH is 0, all floating-point exceptions, except Input Denormal, are determined by treating the input value that is flushed to zero as zero.
>
> If a denormalized input is flushed to zero, and FPCR.AH is 1, an Input Denormal exception is not generated, and other floating-point exceptions are determined by treating the input value that is flushed to zero as zero.
>
> When a half-precision floating-point or BFloat16 value is flushed to zero, an Input Denormal exception is not generated.
>
> If FPCR.AH is 0, when a single-precision or double-precision floating-point input is flushed to zero, an Input Denormal exception is generated.
>
> If FPCR.AH is 1, and FPCR.FIZ is 0, if and only if none of the following apply, any operation with a denormalized floating-point input generates an Input Denormal exception:
>
> - One of the other operands of the instruction is a NaN.
> - The operation generates an Invalid Operation floating-point exception.
> - The operation generates a Divide-by-Zero floating-point exception.
> - The instruction that generated the operation was one of: BFCVT, BFCVTN, BFCVTN2, and BFCVTNT.
> - The denormalized floating-point input is BFloat16.
> - The denormalized floating-point input is half-precision.
> - The denormalized floating-point input is flushed to zero.
>
> If FPCR.AH is 1, or FPCR.FZ is 0, when FPCR.FIZ causes flushing of a denormalized number, an Input Denormal Exception is not generated.

is changed to read:

> The cumulative floating-point exception bit FPSR.IDC, and the trap enable bit FPCR.IDE both relate to Input Denormal exceptions.
>
> Input Denormal exceptions are not generated for denormalized half-precision or denormalized BFloat16 inputs. Otherwise, for single-precision and double-precision inputs:
>
> - When FPCR.{AH, FZ, FIZ} is {0, 0, x}, Input Denormal exceptions are not generated.

- When FPCR.{AH, FZ, FIZ} is {0, 1, x}, an Input Denormal exception is generated when an input is flushed to zero.
- When FPCR.AH is 1, an operation with a denormalized input generates an Input Denormal exception unless any of the following are true:
  - The denormalized input is flushed to zero, when FPCR.FIZ is 1.
  - Another operand to the operation is a NaN.
  - The operation generates a Divide-by-Zero exception.
  - The operation generates an Invalid Operation exception.
  - The instruction that generated the operation was one of:
    - Advanced SIMD&FP convert from single-precision to BFloat16: BFCVT, BFCVTN, BFCVTN2.
    - Advanced SIMD&FP convert to integer: FCVTAS, FCVTAU, FCVTMS, FCVTMU, FCVTNS, FCVTNU, FCVTPS, FCVTPU, FCVTZS, FCVTZU.
    - Advanced SIMD&FP round to integral: FRINT32X, FRINT32Z, FRINT64X, FRINT64Z, FRINTA, FRINTI, FRINTM, FRINTN, FRINTP, FRINTX, FRINTZ.
    - Advanced SIMD&FP JavaScript convert to signed fixed-point: FJCVTZS.
    - SVE convert from single-precision to BFloat16: BFCVT, BFCVTNT.
    - SVE convert to integer: FCVTZS, FCVTZU.
    - SVE round to integral: FRINTA, FRINTI, FRINTM, FRINTN, FRINTP, FRINTX, FRINTZ.
    - SME convert from single-precision to BFloat16: BFCVT, BFCVTN.
    - SME convert to integer: FCVTZS, FCVTZU.
    - SME round to integral: FRINTA, FRINTM, FRINTN, FRINTP.

When a denormalized input is flushed to zero, the occurrence of floating-point exceptions other than Input Denormal Exception are determined by treating the input value that is flushed to zero as zero.

This change is consistent with the pseudocode functions that describes the flush-to-zero of denormalized inputs and Input Denormal exception detection. This update does not modify behaviors exhibited by any floating-point instruction.

2.238 D22813

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

> R_{YSLYC}
>
> If the memory region tagging type for a memory region is:
>
> - Tagged then all of the following are true:

- If the memory region has the Non-shareable attribute, it is **IMPLEMENTATION DEFINED** whether the memory region is treated as Tagged or Untagged.
  - If SCTLR_ELx.C is 0 for a memory access, it is CONSTRAINED **UNPREDICTABLE** whether the memory region is treated as Tagged or Untagged.
- Canonically Tagged then all of the following are true:
  - If the memory region has the Non-shareable attribute, it is **IMPLEMENTATION DEFINED** whether the memory region is treated as Canonically Tagged or Untagged.
  - If SCTLR_ELx.C is 0 for a memory access, it is CONSTRAINED **UNPREDICTABLE** whether the memory region is treated as Canonically Tagged or Untagged.

is changed to read:

R_{YSLYC}

If the memory region tagging type for a memory region is:

- Tagged then all of the following are true:
- If the memory region has the Non-shareable attribute, it is **IMPLEMENTATION DEFINED** whether the memory region is treated as Tagged or Untagged.
  - If SCTLR_ELx.C is 0 for a memory access, it is CONSTRAINED **UNPREDICTABLE** whether the memory region is treated as Tagged or Untagged.
- Canonically Tagged then all of the following are true:
  - If the memory region has the Non-shareable attribute, it is **IMPLEMENTATION DEFINED** whether the memory region is treated as Canonically Tagged or Untagged.
  - SCTLR_ELx.C has no effect on whether the memory region is treated as Canonically Tagged.

2.239 C22814

In section **C8.2.465 "PRFB (scalar plus immediate)"**, the text in the instruction description that reads:

The predicate may be used to suppress prefetches from unwanted addresses.

is changed to a 'Note' as follows:

Arm strongly recommends the following for this instruction:

- A PE does not perform a prefetch as a result of Inactive elements.
- Software uses the predicate operand to suppress prefetches from unwanted addresses.

The equivalent changes are made in the following sections:

- **C8.2.466 "PRFB (scalar plus scalar)"**.
- **C8.2.469 "PRFD (scalar plus immediate)"**.

- **C8.2.470 “PRFD (scalar plus scalar)”**.
- **C8.2.473 “PRFH (scalar plus immediate)”**.
- **C8.2.474 “PRFH (scalar plus scalar)”**.
- **C8.2.477 “PRFW (scalar plus immediate)”**.
- **C8.2.478 “PRFW (scalar plus scalar)”**.

2.240 D22816

In section **I5.11.1 “Error record reset flag”**, the text that reads:

> D_{CPRWG}
>
> FEAT_RASSA_RV is an OPTIONAL node feature from FEAT_RASSAv2. FEAT_RASSA_RV is permitted from FEAT_RASSAv2.

is changed to read:

> D_{CPRWG}
>
> FEAT_RASSA_RV is an OPTIONAL node feature from FEAT_RASSAv1p1.

In section **I5.1 “About the RAS System Architecture”** the text that reads:

> D_{TGKPM}
>
> In this chapter, permitted from refers to the earliest version of the RAS System Architecture in which the feature is permitted to be implemented. A system must be FEAT_RASSAv1p1 compliant to include a feature permitted from FEAT_RASSAv1p1, and must be FEAT_RASSAv2 compliant to include a feature permitted from FEAT_RASSAv2.

is changed to read:

> D_{TGKPM}
>
> In this chapter, OPTIONAL from or permitted from refers to the earliest version of the RAS System Architecture in which the feature is permitted to be implemented. A system must be FEAT_RASSAv1p1 compliant to include a feature permitted from FEAT_RASSAv1p1, and must be FEAT_RASSAv2 compliant to include a feature permitted from FEAT_RASSAv2. If no version is indicated, this means the feature is OPTIONAL from or permitted from FEAT_RASSAv1.

In section **I5.4.5.2 “Security of error records”**, the text that reads:

> D_{VKJVV}
>
> FEAT_RASSA_ACR is an OPTIONAL Error record group feature from FEAT_RASSAv2. FEAT_RASSA_ACR is permitted from FEAT_RASSAv1p1.

is changed to read:

> D_{VKJVV}
>
> FEAT_RASSA_ACR is an OPTIONAL Error record group feature from FEAT_RASSAv1p1.

The equivalent changes are in the following:

- In section **I5.6 "Fault handling interrupt"**, rules D_{ZTFFN} and D_{NDSNV}.
- In section **I5.10 "Error recovery, fault handling, and critical error signaling"**, rules D_{JWTHK} and D_{PRGRZ}.
- In section **I5.11 "Error record reset"**, rule D_{MCHNV}
- In section **I5.15.2 "Fault injection groups"**, rule D_{FQHBV}.

2.241 C22819

In section **D23.2.54 "HCRX_EL2, Extended Hypervisor Configuration Register"**, in the description of field 'FnXS, bit [3]', the text that reads:

> An AArch64 DSB instruction executed at EL1 or EL0 behaves in the same way as the corresponding DSB instruction with the nXS qualifier executed at EL1 or EL0.

is changed to read:

> An AArch64 DSB instruction that applies to both loads and stores executed at EL1 or EL0 behaves in the same way as the corresponding DSB instruction with the nXS qualifier executed at EL1 or EL0.

2.242 C22825

In section **D8.10 "Pointer authentication"**, the information statement I_{PKPFN} that reads:

> I_{PKPFN}
>
> If the value of one or more of the ID_AA64ISAR1_EL1.{GPI, GPA, API, APA} or ID_AA64ISAR2_EL1.{GPA3,APA3} fields is nonzero, then pointer authentication is implemented.

is changed to read:

> I_{PKPFN}
>
> If Pointer authentication is implemented then both of the following are true:
>
> - Exactly one of ID_AA64ISAR1_EL1.GPI, ID_AA64ISAR1_EL1.GPA, or ID_AA64ISAR2_EL1.GPA3 fields is nonzero.

- Exactly one of ID_AA64ISAR1_EL1.API, ID_AA64ISAR1_EL1.APA, or ID_AA64ISAR2_EL1.APA3 fields is nonzero.

Otherwise, all of these ID fields are zero.

2.243 D22830

In section **D8.2.6.1 "Ordering of memory accesses from translation table walks"**, the text that reads:

R_{NNFPF}

If FEAT_ETS2 is implemented, E_1 is an Explicit Memory Effect, E_2 is an Implicit Read of a TTD, and all of the following apply, then E_1 is Ordered-before E_2:

- E_1 is program-order-before a Fault Effect E_3.
- E_2 is Translation-intrinsically-before E_3.

is changed to read:

R_{NNFPF}

If FEAT_ETS2 is implemented, E_1 is an Explicit Memory Effect, E_2 is an Implicit Read of a TTD, and all of the following apply, then E_1 is Ordered-before E_2:

- E_1 is program-order-before a TLBUncacheable Fault Effect E_3.
- E_2 is Translation-intrinsically-before E_3.

I_{X0001}

The full definition of the ordering implications of ETS2 is covered formally in section B2.3 "Definition of the Arm memory model", and the rules in this section are not intended to contradict that section. For detailed information about all ordering requirements to FEAT_ETS2, see also:

- The DSB-ordered-before requirements in section B2.3.7 "Ordering relations".
- The Locally-hardware-required-ordered-before requirements in section B2.3.7 "Ordering relations".

Similarly in section **E2.4 "Ordering of translation table walks"**, the text that reads:

If FEAT_ETS2 is implemented, E_1 is an Explicit Memory Effect, E_2 is an Implicit Read of a TTD and all of the following apply, then E_1 is Ordered-before E_2:

- E_1 is program-order-before a Fault Effect E_3.
- E_2 is Translation-intrinsically-before E_3.

is changed to read:

If FEAT_ETS2 is implemented, E_1 is an Explicit Memory Effect, E_2 is an Implicit Read of a TTD and all of the following apply, then E_1 is Ordered-before E_2:

- E_1 is program-order-before a TLBUncacheable Fault Effect E_3.
- E_2 is Translation-intrinsically-before E_3.

The full definition of the ordering implications of ETS2 is covered formally in section B2.3 "Definition of the Arm memory model", and the rule in this section is not intended to contradict that section. For detailed information about all ordering requirements to FEAT_ETS2, see also:

- The DSB-ordered-before requirements in section B2.3.7 "Ordering relations".
- The Locally-hardware-required-ordered-before requirements in section B2.3.7 "Ordering relations".

2.244 C22833

In section **D8.15.3 "Use of ASIDs and VMIDs to reduce TLB maintenance requirements"**, the following rule is added:

I_{CNTNV}

For the EL1&0 translation regime, TLB entries are associated with a VMID if EL2 is enabled, regardless of whether stage 2 translation is enabled.

2.245 C22838

In section **D23.2.131 "PIR_EL1, Permission Indirection Register 1 (EL1)"**, under the heading 'Perm<m>, bits [4m+3:4m], for m = 15 to 0', the text that reads:

0b0110	Read, Write and Execute, Overlay applied.
0b0111	Read, Write and Execute, Overlay applied.

is changed to read:

0b0110	Read, Write and Execute, Overlay applied. WXN control applied.
0b0111	Read, Write and Execute, Overlay applied.

Unless otherwise specified, the WXN control is not applied.

The equivalent changes are made in the following sections:

- **D23.2.132 "PIR_EL2, Permission Indirection Register 2 (EL2)"**.
- **D23.2.133 "PIR_EL3, Permission Indirection Register 3 (EL3)"**.
- **D23.2.134 "PIRE0_EL1, Permission Indirection Register 0 (EL1)"**.
- **D23.2.135 "PIRE0_EL2, Permission Indirection Register 0 (EL2)"**.

2.246 C22842

In section **C3.2.12.3 "Swap"**, the following text is added:

> If the destination register Rt is WZR or XZR, this is not regarded as doing a read for the purpose of a DMB LD barrier.

Additionally, in section **C3.2.12.4 "Compare and Swap":**, the following text is added:

> If the destination register Rs is WZR or XZR, this is not regarded as doing a read for the purpose of a DMB LD barrier.

2.247 D22844

In section **D23.2.72 "ID_AA64ISAR1_EL1, AArch64 Instruction Set Attribute Register 1"**, under the heading 'APA, bits [7:4]', the text that reads:

> In Armv8.3, the permitted values are `0b0001`, `0b0010`, `0b0011`, `0b0100`, and `0b0101`.
>
> From Armv8.6, the permitted values are `0b0011`, `0b0100`, and `0b0101`.
>
> From Armv9.5, the permitted values are `0b0011`, `0b0100`, `0b0101`, and `0b0110`.

is changed to read:

> In Armv8.3, the permitted values are `0b0000`, `0b0001`, `0b0010`, `0b0011`, `0b0100`, and `0b0101`.
>
> From Armv8.6, the permitted values are `0b0000`, `0b0011`, `0b0100`, and `0b0101`.
>
> From Armv9.5, the permitted values are `0b0000`, `0b0011`, `0b0100`, `0b0101`, and `0b0110`.

The equivalent changes are made in the following sections:

- **D23.2.72 "ID_AA64ISAR1_EL1, AArch64 Instruction Set Attribute Register 1"**, under the heading 'API, bits [11:8]'.
- **D23.2.73 "ID_AA64ISAR2_EL1, AArch64 Instruction Set Attribute Register 2"**, under the heading 'APA3, bits [15:12]'.

2.248 D22852

In section **D8.7.1 "The Contiguous Bit"**, in the final row in Table D8-98 'Permitted properties of caching a translation in a TLB when Effective value of Contiguous bit is 1, VMSAv8-64 translation system', the text that reads:

Translation granule	Block or Page lookup level	Number of adjacent translation table entries	Alignment boundary of adjacent translation table entries within the translation table	Alignment of contiguous OA range
4KB	1	16	128 bytes	16GB
	2	16	128 bytes	32MB
	3	16	128 bytes	64KB
16KB	2	32	256 bytes	1GB
	3	128	1KB	2MB
64KB	2	32	256 bytes	16GB
	3	32	256 bytes	512KB

is changed to read:

Translation granule	Block or Page lookup level	Number of adjacent translation table entries	Alignment boundary of adjacent translation table entries within the translation table	Alignment of contiguous OA range
4KB	1	16	128 bytes	16GB
	2	16	128 bytes	32MB
	3	16	128 bytes	64KB
16KB	2	32	256 bytes	1GB
	3	128	1KB	2MB
64KB	2	32	256 bytes	16GB
	3	32	256 bytes	2MB

2.249 D22855

In section **D19.5.2.1 "Synchronization on exception entry"**, the text that reads:

> IMMVJW
>
> The prioritization of asynchronous exceptions is IMPLEMENTATION DEFINED. This means that an implementation might choose to behave as if the SError exception was taken before the implicit Error synchronization event, if the SError exception was not masked, taking the SError exception in place of the original exception.

is changed to read:

IMMVJW

The prioritization of asynchronous exceptions is IMPLEMENTATION DEFINED. This means that an implementation might choose to behave as if the SError exception made pending by the Error synchronization event was taken before the implicit Error synchronization event, if the SError exception was not masked, taking the SError exception in place of the original exception, unless the original exception is also an SError exception.

In section **J1.1.2 "aarch64/exceptions"**, in the pseudocode function AArch64.TakeException(), the following code segment that reads:

```
if sync_errors && InsertIESBBeforeException(target_el) then
    SynchronizeErrors();
    iesb_req = FALSE;
    sync_errors = FALSE;
    TakeUnmaskedPhysicalSErrorInterrupts(iesb_req);
```

is changed to read:

```
if sync_errors && InsertIESBBeforeException(target_el) then
    SynchronizeErrors();
    if excep.exceptype != Exception_SError then
        iesb_req = FALSE;
        sync_errors = FALSE;
        TakeUnmaskedPhysicalSErrorInterrupts(iesb_req);
```

2.250 D22868

In section **D23.2.160 "SCTLR_EL2, System Control Register (EL2)"**, in the field description of 'ITD, bit [7]', the text that reads:

When EL0 can only use AArch64 and the Effective value of HCR_EL2.E2H is not 1:

If HCR_EL2.TGE == `0b0`, the field is IGNORED for all purposes other than direct reads and writes of the register.

Otherwise:

Reserved, **RES0**.

is changed to read:

When EL0 can only use AArch64 and the Effective value of HCR_EL2.E2H is 1:

Reserved, **RES1**.

Otherwise:

Reserved, **RES0**.

2.251 C22874

In section **K1.2.9 "Out of range virtual address",**, the text that reads:

> If the PE executes a load or store instruction with tagged addressing disabled in the current translation regime, and where the computed virtual address, total access size, and alignment mean that it accesses the bytes at `0xFFFFFFFFFFFFFFFF` and `0x0000000000000000`, then the bytes that appear to be from `0x0000000000000000` onwards are accessed at an **UNKNOWN** address.
>
> If the PE executes a load or store instruction with tagged addressing enabled in the current translation regime, and where the computed address, total access size, and alignment mean that it accesses the bytes at `0xFFFFFFFFFFFFFFFF` and `0x0000000000000000`, then the bytes that appear to be from `0x0000000000000000` onwards are accessed at an unknown address and the tags associated with address also become unknown.

is changed to read:

> If the PE executes a load or store instruction with tagged addressing disabled in the current translation regime, and where the computed virtual address, total access size, and alignment mean that it accesses the bytes at `0xFFFFFFFFFFFFFFFF` and `0x0000000000000000`, then the bytes that appear to be from `0x0000000000000000` onwards are accessed at an **UNKNOWN** address.
>
> It is permitted for the **UNKNOWN** address used to be a fixed value such that it always generates an MMU fault.
>
> If the PE executes a load or store instruction with tagged addressing enabled in the current translation regime, and where the computed address, total access size, and alignment mean that it accesses the bytes at `0xFFFFFFFFFFFFFFFF` and `0x0000000000000000`, then the bytes that appear to be from `0x0000000000000000` onwards are accessed at an unknown address and the tags associated with address also become unknown.
>
> It is permitted for the **UNKNOWN** address used to be a fixed value such that it always generates an MMU fault.

2.252 D22877

In section **C8.2.169 "FEXPA"**, the text that reads:

> The FEXPA instruction accelerates the polynomial series calculation of the EXP(X) function.
>
> The double-precision variant copies the low 52 bits of an entry from a hard-wired table of 64-bit coefficients, indexed by the low 6 bits of each element of the source vector, and prepends to that the next 11 bits of the source element (src<16:6>), setting the sign bit to zero.

> The single-precision variant copies the low 23 bits of an entry from hard-wired table of 32-bit coefficients, indexed by the low 6 bits of each element of the source vector, and prepends to that the next 8 bits of the source element (src<13:6>), setting the sign bit to zero.
>
> The half-precision variant copies the low 10 bits of an entry from hard-wired table of 16-bit coefficients, indexed by the low 5 bits of each element of the source vector, and prepends to that the next 5 bits of the source element (src<9:5>), setting the sign bit to zero.
>
> A coefficient table entry with index M holds the floating-point value $2^{(m/64)}$, or for the half-precision variant $2^{(m/32)}$. This instruction is unpredicated.
>
> This instruction is illegal when executed in Streaming SVE mode, unless FEAT_SME_FA64 is implemented and enabled.

is changed to read:

> The FEXPA instruction computes an exponentiation acceleration operation on each floating-point element in the source vector, where the result sign is zero, the result exponent field is copied from a set of significant bits of the input fraction, and the result fraction is inserted from a lookup table indexed by the least-significant input fraction bits, and returns each result in the corresponding element of the destination vector.
>
> This instruction is fully defined by its bit-manipulation semantics, does not generate floating-point exceptions, and does not guarantee NaN propagation.
>
> For double-precision variants, the result element exponent is copied from the source element bits<16:6>, and the result fraction is set based on the source element to the rounded value of $2^{52} \times (2^{\text{bits<5:0>}/64} - 1)$.
>
> For single-precision variants, the result element exponent is copied from the source element bits<13:6>, and the result fraction is set based on the source element to the rounded value of $2^{23} \times (2^{\text{bits<5:0>}/64} - 1)$.
>
> For half-precision variants, the result element exponent is copied from the source element bits<9:5>, and the result fraction is set based on the source element to the rounded value of $2^{10} \times (2^{\text{bits<4:0>}/32} - 1)$.
>
> This instruction is unpredicated.
>
> This instruction is illegal when executed in Streaming SVE mode, unless FEAT_SME_FA64 is implemented and enabled.
>
> Note:
>
> For a double-precision floating-point input value x in the range $70{,}368{,}744{,}177{,}655 \le x < 70{,}368{,}744{,}179{,}711$, the operation performed by this instruction is equivalent to computing $2^{x-70{,}368{,}744{,}178{,}687}$.

> For a single-precision floating-point input value x in the range 131,073 ≤ x < 131,327, the operation performed by this instruction is equivalent to computing $2^{x-131,199}$.
>
> For a half-precision floating-point input value x in the range 33 ≤ x < 63, the operation performed by this instruction is equivalent to computing 2^{x-47}.

2.253 R22888

In section **D23.2.159 "SCTLR_EL1, System Control Register (EL1)"**, under the heading 'ITFSB, bit[37]', the text that reads:

> When FEAT_MTE2 is implemented

is changed to read:

> When FEAT_MTE_ASYNC is implemented:

The equivalent changes are made in the following sections:

- **D23.2.160 "SCTLR_EL2, System Control Register (EL1)"**, under heading 'ITFSB, bit[37]'.
- **D23.2.161 "SCTLR_EL3, System Control Register (EL1)"**, under heading 'ITFSB, bit[37]'.

In section **D23.2.177 "TFSRE0_EL1, Tag Fault Status Register (EL0)"**, under the heading 'TF1, bit[1]', the text that reads:

> Tag Check Fault. Asynchronously set to 1 when a Tag Check Fault using a virtual address with bit[55] == `0b1` occurs.
>
> The reset behavior of this field is:
>
> - On a Warm reset, this field resets to an architecturally **UNKNOWN** value.

is changed to read:

> When FEAT_MTE_ASYNC is implemented: Tag Check Fault. Asynchronously set to 1 when a Tag Check Fault using a virtual address with bit[55] == `0b1` occurs.
>
> The reset behavior of this field is:
>
> - On a Warm reset, this field resets to an architecturally **UNKNOWN** value.
>
> Otherwise: Reserved, **RES0**.

The equivalent changes are made in the following sections:

- **D23.2.177 "TFSRE0_EL1, Tag Fault Status Register (EL0)"**, under the heading 'TF0, bit[0]'.
- **D23.2.178 "TFSR_EL1, Tag Fault Status Register (EL1)"**, under the heading 'TF0, bit[0]'.
- **D23.2.178 "TFSR_EL1, Tag Fault Status Register (EL1)"**, under the heading 'TF1, bit[1]'.
- **D23.2.179 "TFSR_EL2, Tag Fault Status Register (EL2)"**, under the heading 'TF0, bit[0]'.

- **D23.2.179 "TFSR_EL2, Tag Fault Status Register (EL2)"**, under the heading 'TF1, bit[1]'.
- **D23.2.180 "TFSR_EL3, Tag Fault Status Register (EL3)"**, under the heading 'TF0, bit[0]'.

2.254 C22897

The previously communicated change in this issue is being revised and will be made available with the next release of the *Arm Architecture Reference Manual for A-profile architecture: Known issues*.

2.255 D22901

In section **G1.16.4 "Asynchronous exception routing and masking with higher Exception levels using AArch64"**, the table 'G1-19 Routing of physical asynchronous exceptions' that reads:

SCR.NS	SCR.{FIQ, IRQ, EA}	HCR.TGE	HCR.{FMO IMO AMO}	Target when taken from EL0	Target when taken from EL1	Target when taken from EL2	Target when taken from EL3
0	X	X	X	FIQ IRQ Abt	n/a	n/a	FIQ IRQ Abt
...	...	...	...	...	...	...	...

is changed to read:

SCR.NS	SCR.{FIQ, IRQ, EA}	HCR.TGE	HCR.{FMO IMO AMO}	Target when taken from EL0	Target when taken from EL1	Target when taken from EL2	Target when taken from EL3
0	0	X	X	FIQ IRQ Abt	n/a	n/a	FIQ IRQ Abt
0	1	X	X	Mon	n/a	n/a	Mon

In section **G1.14 "Routing of aborts taken to AArch32 state"**, under the heading 'Memory aborts taken to Secure Abort mode', the text that reads:

> If an implementation includes EL3, when the PE is executing in Secure state, all memory aborts that are not routed to EL3 are taken to Secure Abort mode.

is changed to read

> If an implementation includes EL3, when the PE is executing in Secure state, all memory aborts that are not routed to Monitor mode are taken to Secure Abort mode.

In section **G1.16.2 "Asynchronous exception routing controls"**, the text that reads:

> In an implementation that includes EL3 the following bits in the SCR control the routing of asynchronous exceptions:

SCR.EA	When the value of this bit is 1, any SError interrupt is taken to EL3.
Note	Although this section describes the asynchronous exception routing controls, SCR.EA also controls the routing of synchronous External aborts, see Routing of aborts taken to AArch32 state.
SCR.FIQ	When the value of this bit is 1, any FIQ exception is taken to EL3.

SCR.EA	When the value of this bit is 1, any SError interrupt is taken to EL3.
SCR.IRQ	When the value of this bit is 1, any IRQ exception is taken to EL3.

> When EL3 is using AArch32 and the value of one of the SCR.{EA, FIQ, IRQ} bits is 1, the exception is taken to Monitor mode.

is changed to read:

> In an implementation that includes EL3 the following bits in the SCR control the routing of asynchronous exceptions:

SCR.EA	When the value of this bit is 1, any SError interrupt is taken to Monitor mode.
Note	Although this section describes the asynchronous exception routing controls, SCR.EA also controls the routing of synchronous External aborts, see Routing of aborts taken to AArch32 state.
SCR.FIQ	When the value of this bit is 1, any FIQ exception is taken to Monitor mode.
SCR.IRQ	When the value of this bit is 1, any IRQ exception is taken to Monitor mode.

2.256 R22902

In section **D8.5.2 “The dirty state”**, under the heading ‘Implications of enabling the dirty state management mechanism’, the text that reads:

> R_{QXXXL}
>
> For all of the following cases, if the stage 1 descriptor is writable-clean, then it is **IMPLEMENTATION DEFINED** whether hardware updates the dirty state:
>
> - ...

is changed to read:

> R_{QXXXL}
>
> For all of the following cases, if the stage 1 descriptor is writable-clean, then it is **IMPLEMENTATION DEFINED** whether hardware updates the dirty state:
>
> - ...
> - A synchronous Tag check fault is generated on a write to the memory location.

The same additional entry is added to R_{SJQGP}.

2.257 R22905

In section **D13.1 "About the Performance Monitors"**, under 'Note', the text that reads:

> The Performance Monitors Extension permits an implementation with no event counters (PMCR_EL0.N==0). However, Arm recommends that at least two event counters are implemented, and that hypervisors provide at least this many event counters to guest operating systems.

is changed to read:

> The Performance Monitors Extension permits an implementation with no event counters (PMCR_EL0.N==0) if one of the following is true:
>
> - EL2 is not implemented.
> - FEAT_HPMN0 is implemented.
>
> This means only the cycle counter is implemented. Optionally, the instruction counter can also be implemented. However, Arm recommends that at least two event counters are implemented, and that hypervisors provide at least this many event counters to guest operating systems.

2.258 E22917

In section **J1.3.3 "shared/functions"**, in the pseudocode function LocalTimeoutEvent(), the code segment that reads:

```
// LocalTimeoutEvent()
// ===================
// Returns TRUE if CNTVCT_EL0 equals or exceeds the localtimeout value.

boolean LocalTimeoutEvent(integer localtimeout);
```

is changed to read:

```
boolean IsLocalTimeoutEventPending;
bits(64) LocalTimeoutVal;           // Value to compare against the Virtual Counter
 Timer
                                    // to generate the local timeout event.

// LocalTimeoutEvent()
// ===================
// Returns TRUE if CNTVCT_EL0 equals or exceeds the localtimeout value.

boolean LocalTimeoutEvent(integer localtimeout)
    assert localtimeout >= 0;

    constant bits(64) cntvct = VirtualCounterTimer();
    if UInt(cntvct) >= localtimeout then
        return TRUE;

    IsLocalTimeoutEventPending = TRUE;
    LocalTimeoutVal = localtimeout<63:0>;   // Store value to compare against
                                            // Virtual Counter Timer at subsequent
 clock ticks
```

```
    return FALSE;

// VirtualCounterTimer()
// =====================
// Returns the Counter-Timer Virtual Count value, the value is as read by CurrentEL
 to CNTVCT_EL0.

bits(64) VirtualCounterTimer()
    bits(64) cntvct;

    if PSTATE.EL != EL3 then
        if HaveEL(EL2) && !ELIsInHost(PSTATE.EL) then
            cntvct = PhysicalCountInt() - CNTVOFF_EL2;
        else
            cntvct = PhysicalCountInt();
    else
        if HaveEL(EL2) && !ELUsingAArch32(EL2) then
            cntvct = PhysicalCountInt() - CNTVOFF_EL2;
        elsif HaveEL(EL2) && ELUsingAArch32(EL2) then
            cntvct = PhysicalCountInt() - CNTVOFF;
        else
            cntvct = PhysicalCountInt();
    return cntvct;
```

In **C6.2.448 "WFET"**, under the heading 'Operation', the text that reads:

```
Hint_WFE(localtimeout, WFxType_WFET);
```

is changed to read:

```
Hint_WFET(localtimeout);
```

In **C6.2.447 "WFE"**, under the heading 'Operation', the text that reads:

```
 constant integer localtimeout = 1 << 64; // No local timeout event is generated
Hint_WFE(localtimeout, WFxType_WFE);
```

is changed to read:

```
Hint_WFE();
```

The equivalent changes are made in sections **C6.2.449 "WFI"** and **C6.2.450 "WFIT"**.

2.259 D22918

In section **I6.9.1 "ERRACR, Access Configuration Register"**, under the heading 'Accessing the ERRACR:', the text that reads:

> ERRACR can be accessed through the external debug interface:

is changed to read:

> ERRACR can be accessed through its memory-mapped interface:

2.260 D22929

In section **D1.3.4.2 "Illegal exception returns from AArch64 state"**, under the rule R_{TYTWB}, the text that reads:

> If in AArch64 state, any of the following situations can cause an illegal exception return:
> - A return is made to an Exception level higher than the current Exception level.
> - A return is made to an Exception level that is not implemented.
> - If FEAT_RME is implemented, then if SCR_EL3.{NSE, NS} is {1, 0}, an exception return from EL3.
>
> ...

is changed to read:

> If in AArch64 state, any of the following situations can cause an illegal exception return:
> - A return is made to an Exception level higher than the current Exception level.
> - A return is made to an Exception level that is not implemented.
> - If FEAT_RME is implemented, then if SCR_EL3.{NSE, NS} is {1, 0}, an exception return from EL3 to a lower Exception level.
>
> ...

2.261 R22934

In section **C6.2.275 "RCWCAS, RCWCASA, RCWCASL, RCWCASAL"**, the heading 'Operational information' and the following text is removed:

> If PSTATE.DIT is 1, the timing of this instruction is insensitive to the value of the data being loaded or stored.

The equivalent changes are made in the following sections:

- **C6.2.276 "RCWCASP, RCWCASPA, RCWCASPL, RCWCASPAL"**.
- **C6.2.277 "RCWCLR, RCWCLRA, RCWCLRL, RCWCLRAL"**.
- **C6.2.278 "RCWCLRP, RCWCLRPA, RCWCLRPL, RCWCLRPAL"**.
- **C6.2.279 "RCWSCAS, RCWSCASA, RCWSCASL, RCWSCASAL"**.
- **C6.2.280 "RCWSCASP, RCWSCASPA, RCWSCASPL, RCWSCASPAL"**.
- **C6.2.281 "RCWSCLR, RCWSCLRA, RCWSCLRL, RCWSCLRAL"**.

- **C6.2.282 "RCWSCLRP, RCWSCLRPA, RCWSCLRPL, RCWSCLRPAL"**.
- **C6.2.283 "RCWSET, RCWSETA, RCWSETL, RCWSETAL"**.
- **C6.2.284 "RCWSETP, RCWSETPA, RCWSETPL, RCWSETPAL"**.
- **C6.2.285 "RCWSSET, RCWSSETA, RCWSSETL, RCWSSETAL"**.
- **C6.2.286 "RCWSSETP, RCWSSETPA, RCWSSETPL, RCWSSETPAL"**.
- **C6.2.287 "RCWSSWP, RCWSSWPA, RCWSSWPL, RCWSSWPAL"**.
- **C6.2.288 "RCWSSWPP, RCWSSWPPA, RCWSSWPPL, RCWSSWPPAL"**.
- **C6.2.289 "RCWSWP, RCWSWPA, RCWSWPL, RCWSWPAL"**.
- **C6.2.290 "RCWSWPP, RCWSWPPA, RCWSWPPL, RCWSWPPAL"**.

2.262 D22937

In section **D8.14.5 "MMU fault prioritization from a single address translation stage"**, the text that reads:

> R_{CSFKV}
>
> For an **IMPLEMENTATION DEFINED** MMU fault caused by a Load-Exclusive or Store-Exclusive to an unsupported memory type, the priority is **IMPLEMENTATION DEFINED**.

is changed to read:

> R_{CSFKV}
>
> For an **IMPLEMENTATION DEFINED** MMU fault caused by an unsupported combination of memory access type and memory type and reported with ESR_ELx.DFSC value `0b110101`, the priority is **IMPLEMENTATION DEFINED**.

2.263 D22938

In section **A1.5.8.1 "Operations that do not generate floating-point exceptions"**, the text that reads:

> The BFloat16 instructions defined in BFloat16 behaviors for instructions that compute sum-of-products do not generate floating-point exceptions.
>
> The SME instructions defined in Floating-point behaviors for instructions that target the SME ZA array do not generate floating-point exceptions.

is changed to read:

> The following instructions do not generate any floating-point exceptions:

- SVE exponential accelerator: FEXPA.
- SVE trigonometric select coefficient: FTSSEL.
- The BFloat16 instructions defined in BFloat16 behaviors for instructions that compute sum-of-products.
- The SME instructions defined in Floating-point behaviors for instructions that target the SME ZA array.

2.264 D22943

Section **C3.2.12.2 "Single-copy atomic 64-byte load/store"** is removed from C3.2.12 "Atomic instructions", and is moved under C3.2 "Loads and stores".

2.265 C22945

In section **B2.3.1 "Basic definitions"**, under the heading 'Context Synchronization effects', the text that reads:

> Context Synchronization effects
>
> A Context Synchronization event generates a Context synchronization effect (CSE effect). For example, an ISB instruction generates an ISB effect, which is a CSE effect. An Exception Entry effect is a CSE effect.
>
> If FEAT_ExS is not implemented, or if FEAT_ExS is implemented and the SCTLR ELx.EOS field is set, an Exception Return effect is a CSE effect.

is changed to read:

> Instruction Fetch Barrier effects
>
> An Instruction Fetch Barrier effect (IFBE) is one of:
>
> - A Context Synchronizing event.
> - An exception entry; regardless of the value of SCTLR_ELx.EIS.
>
> Note:
>
> If FEAT_ExS is not implemented, or if FEAT_ExS is implemented and the SCTLR ELx.EOS field is set, an Exception Return effect is an IFBE.

References to 'Context Synchronization effects' in B2.3.7 "Ordering relations" are changed to refer to 'Instruction Fetch Barrier' effects.

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

> The following are not affected by the value of SCTLR_EL1.EIS:
>
> - ...
> - The memory model requirement for exception entry to be an Instruction Fetch Barrier effect.

The equivalent changes are made to the 'EIS' field descriptions in the following sections:

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

In section **D1.3.2 "Exception entry"**, the following informational statement is added after R_{BBSRF}:

> I_{00001}
>
> For the purposes of the memory model, exception entry is an Instruction Fetch Barrier effect even if this SCTLR_ELx.EIS is 0.

This is superseded R23103.

2.266 C22949

In section **A2.2.1 "The Armv8.0 architecture extension"**, under the heading 'FEAT_MixedEnd, Mixed-endian support', the following text is added:

> If FEAT_MixedEnd is implemented, then FEAT_MixedEndEL0 is implemented.

In the same section, under the heading 'FEAT_MixedEndEL0, Mixed-endian support at EL0', the text that reads:

> The following field identifies the presence of FEAT_MixedEndEL0:
>
> - ID_AA64MMFR0_EL1.BigEndEL0.

is changed to read:

> The following fields identify the presence of FEAT_MixedEndEL0:
>
> - ID_AA64MMFR0_EL1.BigEndEL0.
> - ID_AA64MMFR0_EL1.BigEnd.

2.267 D22977

In section **C7.2.15 "BFDOT (by element)"**, the text that reads:

> If FEAT_EBF16 is not implemented or FPCR.EBF is 0, this instruction:
>
> - ...
> - Disables alternative floating point behaviors, as if FPCR.AH is 0.

is changed to read:

> If FEAT_EBF16 is not implemented or FPCR.EBF is 0, this instruction:
>
> - ...
> - Honors FPCR.AH when generating a default NaN result, otherwise disables alternative floating point behaviors as if FPCR.AH is 0.

The equivalent changes are made in the following sections:

- C7.2.16 "BFDOT (vector)".
- C7.2.19 "BFMMLA".
- C8.2.40 "BFDOT (indexed)".
- C8.2.41 "BFDOT (vectors)".
- C8.2.58 "BFMMLA".

This behavior is consistent with the BFDotAdd() pseudocode function.

2.268 C22980

In section **D8.5.2.2 "Implications of enabling the dirty state management mechanism"**, the rule that reads:

> R_{KTXCF}
>
> For a stage 2 writable-clean descriptor with the S2AP[1:0] bits set to `0b00`, if an atomic instruction to the address translated by the descriptor generates a stage 2 Permission fault as a result of not having read permission, then it is **CONSTRAINED UNPREDICTABLE** whether hardware updates the dirty state.

is changed to read:

> R_{KTXCF}
>
> For a stage 2 writable-clean descriptor, if an atomic instruction to the address translated by the descriptor generates a stage 2 Permission fault only as a result of not having read permission, then it is **CONSTRAINED UNPREDICTABLE** whether hardware updates the dirty state.

2.269 D22981

In section **B2.3.9.2 "External completion requirement"**, the text that reads:

> External completion requirement

> The External completion requirement requires that a Memory effect E_1 Completes-before a Memory effect E_2 if all of the following statements are true:
>
> - E_1 is Locally-hardware-required-ordered-before E_2.
> - E_1 is a Memory Read effect, E_2 is a Memory Read effect, and E_1 is single-copy-atomic-ordered before E_2.

is changed to read:

> External completion requirement
>
> The External completion requirement requires that a Memory effect E_1 Completes-before a Memory effect E_2 if any of the following statements are true:
>
> - E_1 is Locally-hardware-required-ordered-before E_2.
> - E_1 is a Memory Read effect, E_2 is a Memory Read effect, and E_1 is single-copy-atomic-ordered before E_2.

In section **B2.3.9.3 "External global completion requirement"**, the text that reads:

> External global completion requirement
>
> The External global completion requirement requires that a Memory effect E_1 Globally-completes-before a Memory effect E_2 if all of the following statements are true:
>
> - E_1 is Locally-hardware-required-ordered-before E_2 and one of the following applies:
>   - E_1 is a Memory Write effect.
>   - E_1 is a Memory Read effect, and one of the following applies:
>   - E_1 Reads-from-memory a Memory Write effect E3, and E_1 and E3 are from different Processing Elements, or
>   - E_1 Reads-from-memory a Memory Write effect E3, E_1, and E3 are from the same Processing Element, and E3 is Locally-hardware-required-ordered-before E_2.
> - E_1 is a Memory Read effect, E_2 is a Memory Read effect, and E_1 is single-copy-atomic-ordered before E_2.

is changed to read:

> External global completion requirement
>
> The External global completion requirement requires that a Memory effect E_1 Globally-completes-before a Memory effect E_2 if any of the following statements are true:
>
> - E_1 is Locally-hardware-required-ordered-before E_2 and one of the following applies:
>   - E_1 is a Memory Write effect.
>   - E_1 is a Memory Read effect, and one of the following applies:
>   - E_1 Reads-from-memory a Memory Write effect E3, and E_1 and E3 are from different Processing Elements, or

    - E_1 Reads-from-memory a Memory Write effect E3, E_1, and E3 are from the same Processing Element, and E3 is Locally-hardware-required-ordered-before E_2.
- E_1 is a Memory Read effect, E_2 is a Memory Read effect, and E_1 is single-copy-atomic-ordered before E_2.

2.270 D22986

In section **J1.3.1 "shared/debug"**, in the pseudocode function PMUOverflowCondition(), the code segment that reads:

```
boolean PMUOverflowCondition(boolean check_e, boolean check_cnten,
                             boolean check_inten, boolean include_hi, boolean
 include_lo,
                             boolean exclude_cyc, boolean exclude_sync)
...
            constant bit sync = (PMCNTENCLR_EL0<idx> AND PMEVTYPER_EL0[idx].SYNC);
...
            constant bit sync = (PMCNTENCLR_EL0.F0 AND PMICFILTR_EL0.SYNC);
...
```

is changed to read:

```
boolean PMUOverflowCondition(boolean check_e, boolean check_cnten,
                             boolean check_inten, boolean include_hi, boolean
 include_lo,
                             boolean exclude_cyc, boolean exclude_sync)
...
            constant bit sync = PMEVTYPER_EL0[idx].SYNC;
...
            constant bit sync = PMICFILTR_EL0.SYNC;
...
```

2.271 C22995

In section **D23.7.12 "PMSIRR_EL1, Sampling Interval Reload Register"**, under the heading 'INTERVAL, bits [31:8]', the text that reads:

> Software should set this to a value greater than the minimum indicated by PMSIDR_EL1.Interval.

is changed to read:

> Arm recommends that software sets this field to a value greater than or equal to the minimum indicated by PMSIDR_EL1.Interval. Setting this field to a smaller nonzero value is likely to cause a large number of sample collisions. See D16.3.7 'Sample collisions' .

2.272 D22998

In section **D1.3.5.5 "Prioritization of Synchronous exceptions taken to AArch64 state"**, the row in rule R_{ZFGJP} that reads:

Priority	Synchronous exception type
29	Other than an exception defined by priorities 4-28, inclusive, any exception taken to EL3 because of a configurable access to an instruction. It is **IMPLEMENTATION DEFINED** whether the exception is prioritized as an **UNDEFINED** instruction or has the priority of the original Trap exception.

is changed to read:

Priority	Synchronous exception type
29	Other than an exception defined by priorities 4-28, inclusive, any exception taken to EL3 because of a configurable access to an instruction. If an exception generated by a configurable instruction control is treated as **UNDEFINED** as a result of EDSCR.SDD configuration, then it is **IMPLEMENTATION DEFINED** whether the exception is prioritized as an **UNDEFINED** instruction or has the priority of the original Trap exception.

2.273 D23002

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

Event	Instructions added
ASE_FP_CVT_SPEC	BF1CVTL, BF1CVTL2, BF2CVTL, BF2CVTL2, F1CVTL, F1CVTL2, F2CVTL, F2CVTL2, FCVTN, FCVTN2, FCVTNS
ASE_FP_DOT_SPEC	FDOT (2-way, by element), FDOT (2-way, vector), FDOT (4-way, by element), FDOT (4-way, vector)
ASE_FP_FMA_SPEC	FMLALB, FMLALLBB, FMLALLBT, FMLALLTT, FMLALT
ASE_FP_PREDUCE_SPEC	FMAXNMP, FMAXP, FMINNMP, FMINP
ASE_INT16_SPEC	LUTI2 (halfword), LUTI4 (halfword)
ASE_INT8_SPEC	LUTI2 (byte), LUTI4 (byte)
ASE_INT_MUL_SPEC	PMULL2, SMLAL2, SMLSL2, SMULL2, SQDMLAL2, SQDMLSL2, SQDMULL2, UMLAL2, UMLSL2, UMULL2
ASE_INT_VREDUCE_SPEC	ADDP, ADDV, SADALP, UADALP, UADDLP, UADDLV, UMAX, UMAXP, UMIN, UMINP
ASE_SVE_FP_CVT_SPEC	BF1CVTL, BF1CVTL2, BF2CVTL, BF2CVTL2, BFCVTN, BFCVTN2, F1CVTL, F1CVTL2, F2CVTL, F2CVTL2, FCVTN, FCVTN2, FCVTNS, FCVTNU, FCVTPS, FCVTPU, FCVTZS, FCVTZU
ASE_SVE_FP_DOT_SPEC	FDOT (2-way, by element), FDOT (2-way, vector), FDOT (4-way, by element), FDOT (4-way, vector)
ASE_SVE_FP_FMA_SPEC	FMLALLBB, FMLALLBT, FMLALLTB, FMLALLTT, FMLALT
ASE_SVE_FP_VREDUCE_SPEC	FAMAX, FAMIN, FMINNMP
ASE_SVE_INT_MUL_SPEC	PMULL2, SMLAL2, SMLSL2, SMULL2, SQDMLAL2, SQDMLSL2, SQDMULL2, UMLAL2, UMLSL2, UMULL2
ASE_SVE_INT_VREDUCE_SPEC	ADDP, ADDV, SADALP, UADALP, UADDLP, UADDLV, UMAX, UMAXP, UMIN, UMINP.
FPASE_LDST_REG_SPEC	LDAP1, LDAPUR, LDNP, LDP, LDR, LDUR, STL1, STLUR, STNP, STP, STR, STUR
FPASE_LD_REG_SPEC	LDAP1, LDAPUR, LDNP, LDP, LDR, LDUR

Event	Instructions added
FPASE_ST_REG_SPEC	STL1, STLUR, STNP, STP, STR, STUR
FP_FMA_SPEC	FMLALB, FMLALLBB, FMLALLBT, FMLALLTB, FMLALLTT, FMLALT
INT_MUL_SPEC	PMULL2, SMLAL2, SMLSL2, SMULL2, SQDMLAL2, SQDMLSL2, SQDMULL2, UMLAL2, UMLSL2, UMULL2

In the same section, the following instructions are removed:

Event	Instructions removed
ASE_INT_VREDUCE_SPEC	UADDVL
ASE_SVE_INT_VREDUCE_SPEC	UADDVL

In the same section, the following Advanced SIMD instructions:

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

are changed to read:

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

2.274 C23004

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

> FEAT_IVIPT is OPTIONAL from Armv8.0.

is changed to read:

> FEAT_IVIPT is mandatory from Armv8.0.

2.275 C23006

In section **D7.5.9.13 "Ordering and completion of data and instruction cache instructions"**, the text that reads:

> In all cases, where the text in this section refers to a DMB or a DSB, this means a DMB or DSB whose required access type is both loads and stores.

is changed to read:

> In all cases, where the text in this section refers to a DMB or a DSB, this means a DMB or DSB whose required access type is both loads and stores.
>
> Note:
>
> This section specifies the memory ordering requirements of DMB and DSB based on the required access type as specified by the <option> parameter. DSB has additional requirements with respect to execution of instructions appearing in program order after the DSB as described in section B2.10.9 "Data Synchronization Barrier (DSB)". These additional requirements create the effect of additional memory ordering beyond what is specified in this section.

2.276 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.

2.277 R23045

In section **D8.14.5 “MMU fault prioritization from a single address translation stage”**, the following text:

1. Alignment fault caused by the memory type.

is changed to read:

1. Alignment fault caused by the memory type.

> For an Alignment fault generated as a result of the conditions specified in K1.2.13 Crossing a page boundary with different memory types or Shareability attributes or K1.2.14 Crossing a peripheral boundary with a Device access, the prioritization of the Alignment fault is **IMPLEMENTATION DEFINED**, at a point between here and above Synchronous External abort on the memory access.

2.278 C23071

In section **D8.5.1.2 "Hardware management of the Access flag"**, the following text is added:

> I_{X0001}
>
> If an implementation does not generate Access flag faults for cache maintenance by VA instructions, it is not required to perform hardware updates of AF for those cache maintenance instructions.
>
> See also D8.14.3 MMU faults generated by cache maintenance operations.

2.279 D23074

In known issue D22811, concerning section **A1.5.8.2 "Input Denormal Exceptions"**, the changed text that reads:

> Input Denormal exceptions are not generated for denormalized half-precision or denormalized BFloat16 inputs. Otherwise, for single-precision and double-precision inputs:

is changed to read:

> Input Denormal exceptions are not generated for denormalized half-precision inputs. For BFloat16 denormalized inputs, see Summary of BFloat16 instruction behaviors. For single-precision and double-precision inputs:

2.280 D23088

In section **G8.4.10 "PMEVCNTR<n>, Performance Monitors Event Count Registers, n = 0 - 30"**, subsection 'Accessing PMEVCNTR<n>', the part of the EL0 access pseudocode that reads:

```
    elsif EL2Enabled() && m >= GetNumEventCountersAccessible() then
        if !IsFeatureImplemented(FEAT_FGT) then
            ConstrainUnpredictableProcedure(Unpredictable_PMUEVENTCOUNTER);
        elsif ELUsingAArch32(EL1) then
            AArch32.TakeHypTrapException(0x03);
        else
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
```

is changed to read:

```
    elsif EL2Enabled() && m >= GetNumEventCountersAccessible() then
        if !IsFeatureImplemented(FEAT_FGT) then
            ConstrainUnpredictableProcedure(Unpredictable_PMUEVENTCOUNTER);
        elsif ELUsingAArch32(EL2) then
            AArch32.TakeHypTrapException(0x03);
        else
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
```

The equivalent changes are made in the following sections:

- **G8.4.11 “PMEVTYPER<n>, Performance Monitors Event Type Registers, n = 0 - 30”**.
- **G8.4.19 “PMXEVCNTR, Performance Monitors Selected Event Count Register”**.
- **G8.4.20 “PMXEVTYPER, Performance Monitors Selected Event Type Register”**.

2.281 R23103

In section **B2.3.1 “Basic definitions”**, under the heading ‘Instruction Fetch Barrier effects’, the text that reads:

> An Instruction Fetch Barrier effect (IFBE) is one of:
>
> - A Context Synchronizing event.
> - An exception entry; regardless of the value of SCTLR_ELx.EIS.
>
> Note: If FEAT_ExS is not implemented, or if FEAT_ExS is implemented and the SCTLR ELx.EOS field is set, an Exception Return effect is an IFBE.

is changed to read:

> An Instruction Fetch Barrier effect (IFBE) is one of:
>
> - A Context Synchronization event.
> - An exception entry to ELx, regardless of the value of SCTLR_ELx.EIS, due to an exception generated for any reason other than any of the following:
>   - SVC instruction execution that is not trapped.
>   - HVC instruction execution that is not disabled.
>   - SMC instruction execution that is not trapped or disabled.
>   - BKPT instruction execution.
>   - BRK instruction execution.
>
> Note: If FEAT_ExS is not implemented, or if FEAT_ExS is implemented and the SCTLR ELx.EOS field is 1, an Exception Return effect is an IFBE due to it being a Context synchronization event.

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

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

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

The equivalent changes are made in the following sections:

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

In section **D1.3.2 "Exception entry"**, the new information statement I_{00001} added after R_{BBSRF} by C22945, that reads:

> I_{00001}
>
> For the purposes of the memory model, exception entry is an Instruction Fetch Barrier effect even if this SCTLR_ELx.EIS is 0.

is changed to read:

> I_{00001}
>
> For the purposes of the memory model, some exception entries are Instruction Fetch Barrier effects, regardless of the value of SCTLR_ELx.EIS. See B2.3.1 "Basic definitions" for the list of exception entries.

2.282 D23126

In section **D10.7.1 "Asynchronous Tag Check Faults"**, under the rule R_{CFWVW}, the text that reads:

> If SVE is implemented and a load of an element in a SVE Non-fault or SVE First-fault load instruction causes an asynchronous Tag Check Fault:
>
> - A bit in a Tag Fault Status Register is not set if the element is not the First active element in a SVE First-fault load.
> - A bit in a Tag Fault Status Register is set if the element is the First active element.

is changed to read:

> If SVE is implemented and a load of an element in a SVE Non-fault or SVE First-fault load instruction causes an asynchronous Tag Check Fault:
>
> - A bit in a Tag Fault Status Register is not set if the element is not the First active element in a SVE First-fault load instruction or any element of an SVE Non-fault load instruction.
> - A bit in a Tag Fault Status Register is set if the element is the First active element of an SVE First-fault load instruction.

2.283 C23130

In section **C3.2.14 "Memory Copy and Memory Set instructions"**, the following text is added:

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

2.284 R23151

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

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

is changed to read:

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

2.285 D23153

In section **D23.10.18 "CNTP_TVAL_EL0, Counter-timer Physical Timer TimerValue Register"**, under the heading 'MRS <Xt>, CNTP_TVAL_EL0' the text that reads:

```
   elsif ELIsInHost(EL0) && IsCurrentSecurityState(SS_Secure) &&
IsFeatureImplemented(FEAT_SEL2) then
       if CNTHPS_CTL_EL2.ENABLE == '0' then
           X[t, 64] = bits(64) UNKNOWN;
       else
           X[t, 64] = CNTHPS_CVAL_EL2 - PhysicalCountInt();
```

is changed to read:

```
    elsif ELIsInHost(EL0) && IsCurrentSecurityState(SS_Secure) &&
 IsFeatureImplemented(FEAT_SEL2) then
        if CNTHPS_CTL_EL2.ENABLE == '0' then
            X[t, 64] = bits(64) UNKNOWN;
        else
            X[t, 64] = ZeroExtend((CNTHPS_CVAL_EL2 - PhysicalCountInt())<31:0>, 64);
```

The equivalent changes are made in the following sections:

D23.10.5 “CNTHP_TVAL_EL2, Counter-timer Physical Timer TimerValue Register (EL2)” D23.10.8 “CNTHPS_TVAL_EL2, Counter-timer Secure Physical Timer TimerValue Register (EL2)” D23.10.11 “CNTHV_TVAL_EL2, Counter-timer Virtual Timer TimerValue Register (EL2)” D23.10.14 “CNTHVS_TVAL_EL2, Counter-timer Secure Virtual Timer TimerValue Register (EL2)” D23.10.24 “CNTPS_TVAL_EL1, Counter-timer Physical Secure Timer TimerValue Register” D23.10.27 “CNTV_TVAL_EL0, Counter-timer Virtual Timer TimerValue Register”

2.286 D23157

In section **D23.10.23 “CNTPS_CVAL_EL1, Counter-timer Physical Secure Timer CompareValue Register”**, under the heading ‘MRS <Xt>, CNTPS_CVAL_EL1’, the pseudocode fragment that reads:

```
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && SCR_EL3.NS == '0' then
        if SCR_EL3.EEL2 == '1' then
            UNDEFINED;
        elsif SCR_EL3.ST == '0' then
            AArch64.SystemAccessTrap(EL3, 0x18);
```

is changed to read:

```
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && SCR_EL3.NS == '0' then
        if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.ST == '0' then
            UNDEFINED;
        elsif SCR_EL3.EEL2 == '1' then
            UNDEFINED;
        elsif SCR_EL3.ST == '0' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
```

Equivalent changes are made under the heading ‘MSR CNTPS_CVAL_EL1, <Xt>,’, and in **D23.10.24 “CNTPS_TVAL_EL1, Counter-timer Physical Secure Timer TimerValue Register”** and **D23.10.25 “CNTV_CTL_EL0, Counter-timer Virtual Timer Control Register”**.

2.287 D23178

In section **D8.1.3 "Relationship between translation regimes and implemented Exception levels"**, in the table 'D8-2 Translation regimes, translation stages, and associated controls', the row that reads:

Translation Stage	Requires
...	...
Non-secure EL2&0 stage 1	EL2 is implemented and EL2 uses AArch64.
	The Effective value of SCR_EL3.NS is 1.
	The Effective value of HCR_EL2.E2H is 1.
...	...

is changed to read:

Translation Stage	Requires
...	...
Non-secure EL2&0 stage 1	EL2 is implemented and EL2 uses AArch64.
	The Effective value of SCR_EL3.{NSE, NS} is {0,1}.
	The Effective value of HCR_EL2.E2H is 1.
...	...

2.288 D23206

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

> FEAT_PAN3 is OPTIONAL from Armv8.0.

is changed to read:

> FEAT_PAN3 is OPTIONAL from Armv8.1.

2.289 D23239

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

> I32

is changed to read:

> I16

and the text:

> I16

is changed to read:

> I32

2.290 C23281

In section **D11.7 "Guarded Control Stack switching"**, the text that reads:

> R_{QHSGP}
>
> The GCSSS1 Xn instruction performs the following functionalities in order:
>
> ...
>
> 1. If the loaded value is not a Valid cap entry, a GCS Data Check exception is generated.
>
> ...

is changed to read:

> R_{QHSGP}
>
> The GCSSS1 Xn instruction performs the following functionalities in order:
>
> ... 3. If the loaded value is not a Valid cap entry, a GCS Data Check exception is generated and no write is required to be performed.
>
> ...

2.291 D23305

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

> R_{JSXRX}
>
> If FEAT_MTE_PERM is implemented, all of the following apply:
>
> - If either translation stage assigns a Device, Non-cacheable, or Write-through memory type, then the stage 1 memory type is treated as not having the Tagged attribute and the resultant memory type is as defined in Stage 2 memory type and Cacheability attributes when FWB is enabled.
> - Otherwise, the stage 1 and stage 2 attributes are combined as shown in Table D8-93.
>
> Table D8-93 Combined stage 1 and stage 2 attributes if FEAT_MTE_PERM is implemented

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

is changed to read:

> R_{JSXRX}
>
> If FEAT_MTE_PERM is implemented, all of the following apply:
>
> - If the stage 1 memory type is not Tagged, the stage 2 NoTagAccess attribute is ignored.
> - Otherwise, the stage 1 and stage 2 attributes are combined as shown in Table D8-93.
>
> Table D8-93 Combined stage 1 and stage 2 attributes if FEAT_MTE_PERM is implemented

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

2.292 D23306

In section **D19.6 "Virtual SError exceptions"**,the text that reads:

> I_{LSSCN}
>
> When implemented, EL2 provides a virtual SError exception.
>
> Virtual SError exceptions are generated by one of the following:
>
> - Software sets HCR_EL2.AMO to `0b1` to enable the virtual SError exception mechanism and HCR_EL2.VSE to `0b1` to inject a virtual SError exception. In AArch32 state, these are the HCR.AMO and HCR.VA bits respectively.
> - An **IMPLEMENTATION DEFINED** source of virtual SError exceptions.

is changed to read:

> I_{LSSCN}
>
> When EL2 is implemented, the PE supports a virtual SError exception. For more information, see:
>
> - D1.3.6.1 Virtual interrupts.
> - G1.16.1 Virtual exceptions when an implementation includes EL2.
>
> I_{X0001}

FEAT_RAS provides:

- Mechanisms to allow a hypervisor to specify the syndrome value reported to a guest Operating System on taking a virtual SError exception.
- Mechanisms to allow firmware to specify the syndrome value reported to a hypervisor or guest Operating System on taking a delegated SError exception.
- Support for EL0 or EL1 to isolate a virtual SError exception and for EL0, EL1, or EL2 to isolate a delegated SError exception as if it was a physical SError exception. See D19.5.1 "ESB and Virtual SError exceptions".

In section **D19.5.1 "ESB and Virtual SError exceptions"**,the following text is removed:

R_{YVBSH}

If all of the following are true, then it is **IMPLEMENTATION DEFINED** whether or not an ESB instruction executed at EL0 or EL1 synchronizes a pending virtual SError exception from an **IMPLEMENTATION DEFINED** source:

- EL2 is implemented and enabled in the current Security state.
- Any of the following are true:
  - EL2 is using AArch64, HCR_EL2.AMO is `0b1`, and HCR_EL2.TGE is `0b0`.
  - EL2 is using AArch32, HCR.AMO is `0b1`, and HCR.TGE is `0b0`.

If a virtual SError exception from an **IMPLEMENTATION DEFINED** source that is a synchronizable error is pending, then the following occur when an ESB instruction is executed at EL0 or EL1:

- If the virtual SError exception is unmasked at the current Exception level, then the exception is taken before the completion of the ESB instruction.
- If the virtual SError exception is masked at the current Exception level, then all the following occur:
  - The pending state of the virtual SError exception is cleared.
  - The virtual SError exception syndrome is set to the IMPLEMENTATION DEFINED syndrome for the virtual SError exception. See R_{YZCYX} and R_{JQGXD}.
- VDISR_EL2.A or VDISR.A is set to `0b1` to indicate the SError exception was pending prior to the execution of the ESB instruction.

If a virtual SError exception from an **IMPLEMENTATION DEFINED** source that is not a synchronizable error is pending, then an ESB instruction executed at EL0 or EL1 ignores the pending virtual SError exception and the virtual SError exception stays pending.

All other mentions of virtual SError exceptions are similarly removed.

In section **G5.12.2.2 "Data Abort exceptions, taken to a PL1 mode"**, the text that reads:

If the FEAT_RAS is implemented, and the exception is a virtual SError interrupt exception, the classification reported in DFSR is taken from VDFSR or VSESR_EL2.

is changed to read:

If the FEAT_RAS is implemented, and the exception is a virtual SError interrupt exception, then DFSR.{AET, ExT} are set to values from VSESR_EL2 or VDFSR.

2.293 D23319

In section **B2.3.1 "Basic definitions"**, the text that reads:

An effect E_1 and an effect E_2 are to the Same Location if one of the following applies:

...

- All of the following applies:
  - An effect E_3 is a Translation-intrinsically-before E_1.
  - An effect E_4 is a Translation-intrinsically-before E_1.
  - The output address of E_3 is the same as the output address of E_4.
  - E_1 and E_2 have the Same Low-order Bits.

is changed to read:

An effect E_1 and an effect E_2 are to the Same Location if one of the following applies:

...

- All of the following applies:
  - An effect E_3 is a Translation-intrinsically-before E_1.
  - An effect E_4 is a Translation-intrinsically-before E_2.
  - The output address of E_3 is the same as the output address of E_4.
  - E_1 and E_2 have the Same Low-order Bits.

2.294 D23325

In section **C5.1.3.2 "op0==0b00, architectural hints, barriers and CLREX, and PSTATE access"**, under the heading "Barriers and CLREX", the text that reads:

The value of op2 determines the instruction, as follows.

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

is changed to read:

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

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

2.295 R23333

In section **D19 "RAS PE Architecture"**, under 'Example D19-1 Minimal implementation of RAS Extension', the text that reads:

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

is changed to read:

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

2.296 D23338

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

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

is changed to read:

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

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

The equivalent change is made in **J1.1 "Pseudocode for AArch64 operation"**, in the pseudocode function MemAtomicRCW().

2.297 D23339

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

> R_{00000}
>
> If MemAttr[1:0] bits define Device memory attributes, then stage 2 Device memory attributes are combined with stage 1 memory attributes.

2.298 R23355

In section **D13.12.3 "Common event numbers"**, the text that reads:

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

is changed to read:

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

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

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

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

2.299 R1610: SME

In section **C9.2.126 "MOVA (tile to vector, four registers)"**, in the '64-bit' encoding variant, the decode pseudocode that reads:

```
if !HaveSME2() then UNDEFINED;
...
```

is changed to read:

```
if !HaveSME2() then UNDEFINED;
if MaxImplementedSVL() < 256 then UNDEFINED;
...
```

Similarly, in section **C9.2.271 "UZP (four registers)"**, in the '128-bit' encoding variant, the decode pseudocode that reads:

```
if !HaveSME2() then UNDEFINED;
constant integer esize = 128;
...
```

is changed to read:

```
if !HaveSME2() then UNDEFINED;
if MaxImplementedSVL() < 512 then UNDEFINED;
constant integer esize = 128;
...
```

Equivalent changes are made in the following sections:

- **C9.2.131 "MOVA (vector to tile, four registers)"**, when esize == 64.
- **C9.2.136 "MOVAZ (tile to vector, four registers)"**, when esize == 64.
- **C9.2.272 "UZP (two registers)"**, when esize >= 64.
- **C9.2.278 "ZIP (four registers)"**, when esize >= 64.
- **C9.2.279 "ZIP (two registers)"**, when esize == 128.

2.300 R1611: SME

In section **C8.2 "Alphabetical list of SVE instructions"**, under the heading 'Operational Information' in all PEXT instruction pages, the following text is removed:

> When PSTATE.DIT is 1:
> - The execution time of this instruction is independent of:
>   - The values of the data supplied in any of its registers.
>   - The values of the NZCV flags.
> - The response of this instruction to asynchronous exceptions does not vary based on:
>   - The values of the data supplied in any of its registers.
>   - The values of the NZCV flags

2.301 D558: SVE2

In section **C8 "SVE Instruction Descriptions"**, in the predicated SVE instructions that can be prefixed by MOVPRFX, the 'Operational information' subsections that read:

> This instruction might be immediately preceded in program order by a MOVPRFX instruction. The MOVPRFX instruction must conform to all of the following requirements, otherwise the behavior of the MOVPRFX and this instruction is **UNPREDICTABLE**:
> - The MOVPRFX instruction must be unpredicated, or be predicated using the same governing predicate register and source element size as this instruction.
> - The MOVPRFX instruction must specify the same destination register as this instruction.
> - The destination register must not refer to architectural register state referenced by any other source operand register of this instruction.

are changed to read:

> This instruction might be immediately preceded in program order by a MOVPRFX instruction. The MOVPRFX instruction must conform to all of the following requirements, otherwise the behavior of the MOVPRFX and this instruction is **CONSTRAINED UNPREDICTABLE**:
> - The MOVPRFX must specify the same destination register as this instruction.
> - The MOVPRFX can be predicated or unpredicated.
> - A predicated MOVPRFX must use the same governing predicate register as this instruction.
> - A predicated MOVPRFX must use the larger of the destination element size and first source element size generated by the preferred disassembly of this instruction.
> - The destination register must not refer to architectural register state referenced by any other source operand register of this instruction.

2.302 D568: SVE2

In section **C8.2.233 "FTMAD"**, the instruction description that reads:

> The FTMAD instruction calculates the series terms for either SIN(X) or COS(X), where the argument x has been adjusted to be in the range $-\pi/4 < X \leq \pi/4$.
>
> To calculate the series terms of SIN(X) and COS(X) the initial source operands of FTMAD should be zero in the first source vector and X2 in the second source vector. The FTMAD instruction is then executed eight times to calculate the sum of eight series terms, which gives a result of sufficient precision.
>
> The FTMAD instruction multiplies each element of the first source vector by the absolute value of the corresponding element of the second source vector and performs a fused addition of each product with a value obtained from a table of hard-wired coefficients, and places the results destructively in the first source vector.
>
> The coefficients are different for SIN(X) and COS(X), and are selected by a combination of the sign bit in the second source element and an immediate index in the range 0 to 7.

is changed to read:

> The FTMAD instruction multiplies each element of the first source vector by the absolute value of the corresponding element of the second source vector and performs a fused addition of each product with a value obtained from a table of hard-wired coefficients, and places the results destructively in the first source vector. This instruction is unpredicated.
>
> The coefficients are selected by a combination of the sign bit in the second source element and an immediate index in the range 0 to 7.
>
> FTMAD can be combined with FTSMUL and FTSSEL to compute values for SIN(X) and COS(X), see FTSMUL. The coefficients are intended to provide accurate results for FTSMUL inputs in the range $-\pi/4 < X \leq \pi/4$.

The coefficient tables in the FTMAD instruction description are changed as follows:

- The rounded decimal values are replaced by the approximate fractional representation, the actual values used can be found in the pseudocode.
- The references in the table headings to sin(x) and cos(x) depending on the sign bit of the second source element are removed.

In section **C8.2.234 "FTSMUL"**, the instruction description that reads:

> The FTSMUL instruction calculates the initial value for the FTMAD instruction. The instruction squares each element in the first source vector and then sets the sign bit to a copy of bit 0 of the corresponding element in the second source register, and places the results in the destination vector. This instruction is unpredicated.

> To compute SIN(X) or COS(X) the instruction is executed with elements of the first source vector set to X, adjusted to be in the range $-\pi/4 < X \leq \pi/4$.
>
> The elements of the second source vector hold the corresponding value of the quadrant q number as an integer not a floating-point value. The value q satisfies the relationship $(2q-1) \times \pi/4 < X \leq (2q+1) \times \pi/4$.

is changed to read:

> The FTSMUL instruction multiplies each element of the first source vector by itself, replaces the sign-bit of each product with the least-significant bit of the corresponding element of the second source vector, and places the results in the destination vector. This instruction is unpredicated.
>
> FTSMUL can be combined with FTMAD and FTSSEL to compute values for SIN(X) and COS(X). The use of the second operand is consistent with it holding an integer corresponding to the desired sine-wave quadrant when used in conjunction with FTMAD.
>
> Note:
>
> FTSMUL, FTMAD and FTSSEL can be used to compute values for SIN(K) and correspondingly COS(K-$\pi/2$) via an intermediate value X, in the range $-\pi/4 < X \leq \pi/4$, and a quadrant Q where $K = Q\pi/2 + X$, using a Taylor Series approximation. FTSMUL can be used to compute X2, and to insert Q<0> into the most-significant bit, indicating the desired odd vs even Taylor Series to be used in FTMAD. Repeated uses of FTMAD can be performed on this value with decreasing immediate index operands, to produce a single accumulated value approximating the Taylor Series result with a single outstanding factor. FTSSEL can be used to apply the final factor of X, 1.0, -X, or -1.0, dependent on the corresponding sine-wave quadrant Q<1:0>, to produce a final SIN() or COS() value.

In section **C8.2.235 "FTSSEL"**, the instruction description that reads:

> The FTSSEL instruction selects the coefficient for the final multiplication in the polynomial series approximation. The instruction places the value 1.0 or a copy of the first source vector element in the destination element, depending on bit 0 of the quadrant number Q held in the corresponding element of the second source vector. The sign bit of the destination element is copied from bit 1 of the corresponding value of Q. This instruction is unpredicated.
>
> To compute SIN(X) or COS(X) the instruction is executed with elements of the first source vector set to X, adjusted to be in the range $-\pi/4 < X \leq \pi/4$.
>
> The elements of the second source vector hold the corresponding value of the quadrant Q number as an integer not a floating-point value. The value Q satisfies the relationship $(2Q-1) \times \pi/4 < X \leq (2Q+1) \times \pi/4$.

is changed to read:

> The FTSSEL instruction selects either the element of the first source vector, or the value 1.0, based on the bit<0> of the corresponding element of the second source element, negates the result if bit<1> of the corresponding element of the second source element is set, and places the results in the destination vector. This instruction is unpredicated.

> FTSSEL may be combined with FTSMUL and FTMAD to compute values for SIN(K) and COS(K), see FTSMUL. The use of the second operand is consistent with it holding an integer corresponding to the desired sine-wave quadrant.

2.303 C573: SVE2

In section **C8.2.204 "FMMLA"**, the text in the instruction description that reads:

> The double-precision variant requires that the current vector length is at least 256 bits, and if the current vector length is not an integer multiple of 256 bits then the trailing bits are set to zero.

is changed to read:

> The double-precision variant requires that the Effective SVE vector length is at least 256 bits.

Equivalent changes are made in the following sections:

- **C8.2.274 "LD1ROB (scalar plus immediate)"**.
- **C8.2.275 "LD1ROB (scalar plus scalar)"**.
- **C8.2.276 "LD1ROD (scalar plus immediate)"**.
- **C8.2.277 "LD1ROD (scalar plus scalar)"**.
- **C8.2.278 "LD1ROH (scalar plus immediate)"**.
- **C8.2.279 "LD1ROH (scalar plus scalar)"**.
- **C8.2.280 "LD1ROW (scalar plus immediate)"**.
- **C8.2.281 "LD1ROW (scalar plus scalar)"**.
- **C8.2.745 "TRN1, TRN2 (vectors)"**.
- **C8.2.853 "UZP1, UZP2 (vectors)"**.
- **C8.2.885 "ZIP1, ZIP2 (vectors)"**.

2.304 R598: SVE2

In section **C8.2.498 "REVD"**, under the heading 'Operational Information', the following text is removed:

> This instruction might be immediately preceded in program order by a MOVPRFX instruction. The MOVPRFX instruction must conform to all of the following requirements, otherwise the behavior of the MOVPRFX and this instruction is unpredictable:
>
> - The MOVPRFX instruction must be unpredicated.
> - The MOVPRFX instruction must specify the same destination register as this instruction.
> - The destination register must not refer to architectural register state referenced by any other source operand register of this instruction.

2.305 D1273: RME

In section **D13.5.4 "Prohibiting event and cycle counting"**, the text that reads:

> The cycle counter, PMCCNTR, counts unless any of the following is true:
>
> ...
>
> - FEAT_PMUv3p5 is implemented, EL3 is implemented, the PE is in Secure state, and SDCR.SCCD is 1.
> - FEAT_PMUv3p7 is implemented, the PE is at EL3, EL3 is using AArch64, and MDCR_EL3.MCCD is 1.

is changed to read:

> The cycle counter, PMCCNTR, counts unless any of the following is true:
>
> ...
>
> - FEAT_PMUv3p5 is implemented, EL3 is implemented, the PE is in Secure state, and SDCR.SCCD is 1.
> - FEAT_PMUv3p7 is implemented, the PE is at EL3, EL3 is using AArch64, and MDCR_EL3.{SCCD, MCCD} is not {0, 0}.

In section **J1.2.1 "shared/debug"**, in the pseudocode function CountPMUEvents(), the code that reads:

```
// Event counting in Secure state is prohibited if all of:
...
if HaveEL(EL3) && ss == SS_Secure then
    if !ELUsingAArch32(EL3) then
        prohibited = MDCR_EL3.SPME == '0' && HavePMUv3p7() && MDCR_EL3.MPMX == '0';
    else
        prohibited = SDCR.SPME == '0';
...
// If FEAT_PMUv3p5 is implemented, cycle counting can be prohibited.
// This is not overridden by PMCR_EL0.DP.
if HavePMUv3p5() && idx == CYCLE_COUNTER_ID then
     if HaveEL(EL3) && ss == SS_Secure then
        sccd = if HaveAArch64() then MDCR_EL3.SCCD else SDCR.SCCD;
        if sccd == '1' then
            prohibited = TRUE;
```

is changed to read:

```
// Event counting in Secure state and EL3 is prohibited if all of:
...
if HaveEL(EL3) && (ss == SS_Secure || PSTATE.EL == EL3) then
    if !ELUsingAArch32(EL3) then
        prohibited = MDCR_EL3.SPME == '0' && HavePMUv3p7() && MDCR_EL3.MPMX == '0';
    else
        prohibited = SDCR.SPME == '0';
...
// If FEAT_PMUv3p5 is implemented, cycle counting can be prohibited.
// This is not overridden by PMCR_EL0.DP.
```

```
if HavePMUv3p5() && idx == CYCLE_COUNTER_ID then
     if HaveEL(EL3) && (ss == SS_Secure || PSTATE.EL == EL3) then
        sccd = if HaveAArch64() then MDCR_EL3.SCCD else SDCR.SCCD;
        if sccd == '1' then
            prohibited = TRUE;
```

In section **D23.3.18 “MDCR_EL3, Monitor Debug Configuration Register (EL3)”**, the text that reads:

> SCCD, bit [23]
>
> When FEAT_PMUv3p5 is implemented:
>
> Secure Cycle Counter Disable. Prohibits PMCCNTR_EL0 from counting in Secure state.

0b0	Cycle counting by PMCCNTR_EL0 is not affected by this mechanism.
0b1	Cycle counting by PMCCNTR_EL0 is prohibited in Secure state.

is changed to read:

> SCCD, bit [23]
>
> When FEAT_PMUv3p5 is implemented:
>
> Secure Cycle Counter Disable. Prohibits PMCCNTR_EL0 from counting in Secure state and EL3.

0b0	Cycle counting by PMCCNTR_EL0 is not affected by this mechanism.
0b1	Cycle counting by PMCCNTR_EL0 is prohibited in Secure state and EL3.