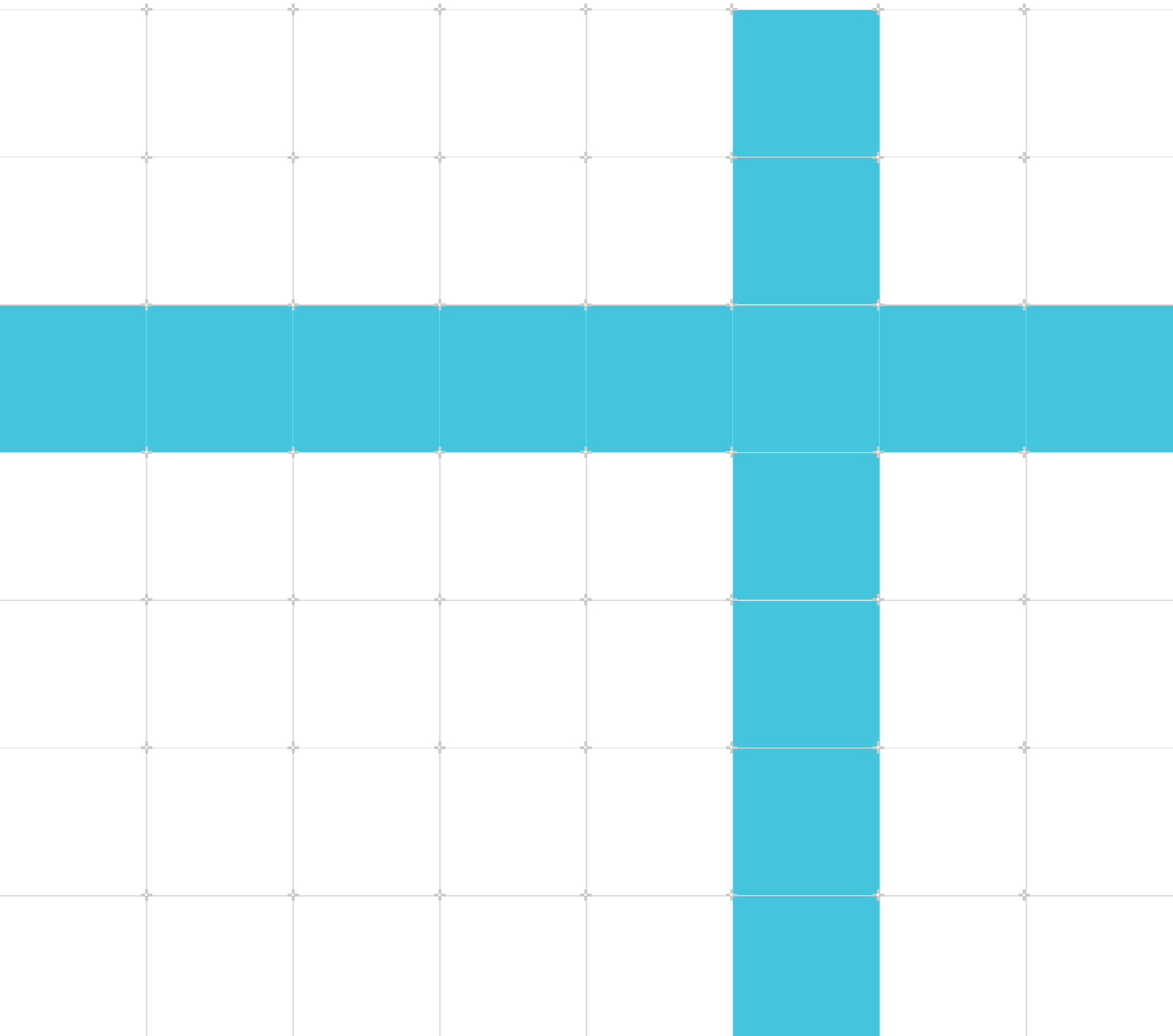
Arm® Architecture Reference Manual for A-profile architecture

Known issues in Issue H.a

Issue 06
102105_H.a_06_en



Arm® Architecture Reference Manual for A-profile architecture

Known issues in Issue H.a

Release information

Document history

Issue	Date	Confidentiality	Change
F.c-04	18 December 2020	Non-Confidential	Known Issues in Arm® Architecture Reference Manual, Issue F.c, as of 18 December 2020
G.b-05	31 January 2022	Non-Confidential	Known Issues in Arm® Architecture Reference Manual, Issue G.b, as of 7 January 2022
H.a-00	31 January 2022	Non-Confidential	Known Issues in Arm® Architecture Reference Manual, Issue H.a, as of 7 January 2022
H.a-01	28 February 2022	Non-Confidential	Known Issues in Arm® Architecture Reference Manual, Issue H.a, as of 18 February 2022
H.a-02	31 March 2022	Non-Confidential	Known Issues in Arm® Architecture Reference Manual, Issue H.a, as of 18 March 2022
H.a-03	29 April 2022	Non-Confidential	Known Issues in Arm® Architecture Reference Manual, Issue H.a, as of 22 April 2022
H.a-04	31 May 2022	Non-Confidential	Known Issues in Arm® Architecture Reference Manual, Issue H.a, as of 20 May 2022
H.a-05	30 June 2022	Non-Confidential	Known Issues in Arm® Architecture Reference Manual, Issue H.a, as of 17 June 2022
H.a-06	22 July 2022	Non-Confidential	Known Issues in Arm® Architecture Reference Manual, Issue H.a, as of 22 July 2022

Proprietary Notice

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**.
Caution	Recommendations. Not following these recommendations might lead to system failure or damage.
Warning	Requirements for the system. Not following these requirements might result in system failure or damage.
Danger	Requirements for the system. Not following these requirements will result in system failure or damage.
Note	An important piece of information that needs your attention.

Convention	Use
Tip	A useful tip that might make it easier, better or faster to perform a task.
Remember	A reminder of something important that relates to the information you are reading.

1.2 Additional reading

This document contains information that is specific to this product. See the following documents for other relevant information:

Table 1-2: Arm publications

Document Name	Document ID	Licensee only
Arm® Architecture Reference Manual for A-profile architecture, Issue H.a	DDI 0487H.a	No

Note

Arm tests its PDFs only in Adobe Acrobat and Acrobat Reader. Arm cannot guarantee the quality of its documents when used with any other PDF reader.

Adobe PDF reader products can be downloaded at http://www.adobe.com

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 H.a.

Key

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

2.1 D14559

In section D6.5 (PE access to Allocation Tags), the text that reads:

> Instructions which load or store Allocation tags are considered to perform the access, irrespective of whether access to Allocation tags in memory is disabled due to Allocation tag access controls in HCR_EL2, SCR_EL3 and SCTLR_ELx, or due to the absence of the Tagged attribute on the locations being accessed, for the purpose of:
>
> - Address translation.
> - Triggering watchpoints.
> - Generating PMU events.
> - Statistical profiling.

is clarified to read:

> Instructions which load or store Allocation tags are considered to perform the access for the purposes of address translation, triggering watchpoints, generating PMU events and Statistical Profiling irrespective of whether access to the Allocation tags in memory is disabled. This includes:
>
> - When FEAT_MTE2 is not implemented.
> - Access to Allocation tags in memory is disabled due to Allocation tag access controls in HCR_EL2, SCR_EL3 and SCTLR_ELx.
> - The Tagged attribute is absent on the locations being accessed.

2.2 D15075

In section D5.10.1 (General TLB maintenance requirements), under the subsection 'Clearing entries associated with a Conflict abort', the bullet that reads:

- For the EL2&0 translation regime: TLBI VMALLE1, TLBI ALLE1.

is corrected to read:

- For the EL2&0 translation regime: TLBI VMALLE1, TLBI ALLE2.

2.3 R16350

In section K1.2.19 (Reserved values in System and memory-mapped registers and translation table entries), the following text is added:

> Unless otherwise stated, when a direct write of a System register or memory-mapped register writes an unallocated value to a field, if that value is unallocated in all contexts then a subsequent read of the field returns an **UNKNOWN** value.

The same text is added in section K1.1.34 (Reserved values in System and memory-mapped registers and translation table entries).

2.4 D16717

In section D13.2.37 (ESR_EL1, Exception Syndrome Register (EL1)), for the field ISV, bit[24], the following text is removed:

> When FEAT_MTE is implemented, for a synchronous Tag Check Fault abort taken to ELx, ESR_ELx.FnV is 0 and FAR_ELx is valid.

The same correction is made in the following sections:

- D13.2.38 (ESR_EL2, Exception Syndrome Register (EL2)).
- D13.2.39 (ESR_EL3, Exception Syndrome Register (EL3)).

2.5 D17015

Details of traps will be added through the use of new LDC and STC accessibility pseudocode in sections G8.3.17 (DBGDTRRXint) and G8.3.19 (DBGDTRTXint). This accessibility pseudocode is the same as for the equivalent MRC and MCR instructions, except that:

- The reported exception syndrome value, if applicable, is `0x06`.

- For LDC instructions the accessibility pseudocode loads the value to be written to the System register from 'MemA[address, 4]', where 'address' is the virtual address calculated by the LDC instruction.

2.6 D17065

In section D13.2.68 (ID_AA64PFR1_EL1, AArch64 Processor Feature Register 1), the text in the MTE field that reads:

```
From Armv8.7, the value 0b0001 is not permitted.
```

is corrected to:

```
From Armv8.7, the value 0b0010 is not permitted.
```

2.7 D17119

In sections F3.1.10 (Advanced SIMD shifts and immediate generation), subsection 'Advanced SIMD two registers and shift amount' and F4.1.22 (Advanced SIMD shifts and immediate generation), subsection 'Advanced SIMD two registers and shift amount', the following constraints are added to VMOVL:

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

2.8 R17661

In section D6.2 (Allocation Tags), the following Notes are removed:

> Note: The value `0b1111` may incur a higher performance overhead than other Allocation Tag encodings.
>
> Note: Arm recommends that software does not use instructions which write `0b1111` as an Allocation Tag to memory.

2.9 C17811

In section I5.8.32 (ERR<n>STATUS, Error Record Primary Status Register, n = 0 - 65534), under the heading 'Accessing the ERR<n>STATUS', the text that reads:

> To ensure correct and portable operation, when software is clearing the valid fields in the register to allow new errors to be recorded, Arm recommends that software:
>
> - Read ERR<n>STATUS and determine which fields need to be cleared to zero.
> - Write ones to all the W1C fields that are nonzero in the read value.
> - Write zero to all the W1C fields that are zero in the read value.
> - Write zero to all the RW fields.

is clarified to read:

> To ensure correct and portable operation, when software is clearing the valid fields in the register to allow new errors to be recorded, Arm recommends that software:
>
> - Read ERR<n>STATUS and determine which fields need to be cleared to zero.
> - In a single write to ERR<n>STATUS:
>   - Write ones to all the W1C fields that are nonzero in the read value.
>   - Write zero to all the W1C fields that are zero in the read value.
>   - Write zero to all the RW fields.
> - Read back ERR<n>STATUS after the write to confirm no new fault has been recorded.

2.10 D18179

In section D10.6.8 (Controlling the data that is collected), the text that reads:

> The timestamp is a choice between:
>
> - Physical time, which is defined by the value of CNTPCT_EL0.
> - If FEAT_ECVis implemented and enabled, offset physical time, as defined by the value of (CNTPCT_EL0- CNTPOFF_EL2). That is, the physical time minus the physical offset, CNTPOFF_EL2.
> - Virtual time, as defined by the value of CNTVCT_EL0. That is, the physical time minus the virtual offset, CNTVOFF_EL2. However, the virtual offset is treated as zero if a read of CNTVCT_EL0 at the current Exception level would treat the virtual offset as zero.

is replaced by:

> The timestamp is a choice between:
>
> - Physical time, which is defined by the physical count value returned by PhysicalCountInt().

- If FEAT_ECV is implemented, offset physical time, which is defined as the value of (PhysicalCountInt() - CNTPOFF_EL2). However, the physical offset is treated as zero if FEAT_ECV is disabled.
- Virtual time, which is defined as the value of (PhysicalCountInt() - CNTVOFF_EL2). However, the virtual offset is treated as zero if a read of CNTVCT_EL0 at the current Exception level would treat the virtual offset as zero.

In the same section, before Table D10-3 'Recorded timestamp when FEAT_ECV is implemented', the text that reads:

> Virtual
>
> This means the timestamp is offset physical time, as returned by a direct read of CNTVCT_EL0 at the Exception level the sampled operation is executed at.
>
> Physical
>
> This means the timestamp is physical time, given by the value of CNTPCT_EL0 at the Exception level the sampled operation is executed at.
>
> Offset physical
>
> This means the timestamp is offset physical time, as returned by (CNTPCT_EL0 - CNTPOFF_EL2) at the Exception level the sampled operation is executed at. That is, the physical time minus the physical offset. When any of the following are true, the Effective value of CNTPOFF_EL2 is 0 for all profiling purposes:
>
> - EL2 is not implemented.
> - FEAT_ECV is not implemented.
> - CNTHCTL_EL2.ECV is 0.
> - SCR_EL3.ECVEn is 0.

is replaced by:

> Virtual offset
>
> If any of the following are true, the virtual offset is zero, otherwise the virtual offset is the value of CNTVOFF_EL2:
>
> - The Effective value of HCR_EL2.E2H is 1, and the sampled operation was executed at EL2.
> - The Effective values of HCR_EL2.{E2H, TGE} is {1, 1}, and the sampled operation was executed at EL0.
>
> Physical offset
>
> If any of the following are true, the physical offset is zero, otherwise the physical offset is the value of CNTPOFF_EL2:
>
> - EL2 is not implemented.

- FEAT_ECV is not implemented.
- CNTHCTL_EL2.ECV is 0.
- SCR_EL3.ECVEn is 0.

and in the table itself:

- 'Virtual' is replaced by 'PhysicalCountInt() - virtual offset'.
- 'Physical' is replaced by 'PhysicalCountInt()'.
- 'Offset physical' is replaced by 'PhysicalCountInt() - physical offset'.

In section D3.3 (Self-hosted trace timestamps), the text that reads:

When SelfHostedTraceEnabled() == TRUE, the trace timestamp is one of the following:

- The physical counter value CNTPCT_EL0.
- An offset physical counter value, which is calculated from the physical counter value CNTPCT_EL0, minus an offset CNTPOFF_EL2. When any of the following are true, the Effective value of CNTPOFF_EL2 is 0 for all trace purposes:
  - EL3 is using AArch32.
  - EL2 is not implemented.
  - FEAT_ECV is not implemented.
  - The Effective value of SCR_EL3.{NS,RW} is {1,0}.
  - CNTHCTL_EL2.ECV is 0.
  - SCR_EL3.ECVEn is 0.
- A virtual counter value, which is calculated from the physical counter value CNTPCT_EL0, minus an offset CNTVOFF_EL2.

is replaced by:

When SelfHostedTraceEnabled() == TRUE, the trace timestamp is one of the following:

- Physical time, which is defined by the physical count value returned by PhysicalCountInt().
- If FEAT_ECV is implemented, offset physical time, which is defined as the value of (PhysicalCountInt() - CNTPOFF_EL2). However, the physical offset is treated as zero if FEAT_ECV is disabled.
- Virtual time, which is defined as the value of (PhysicalCountInt() - CNTVOFF_EL2). The virtual offset is always CNTVOFF_EL2, including when a read of CNTVCT_EL0 at the current Exception level would treat the virtual offset as zero.

The following is added before Table D3-2 'Timestamp used for trace':

In Table D3-2, if any of the following are true, the value of physical offset is zero, otherwise the value of physical offset is the value of CNTPOFF_EL2:

- EL3 is using AArch32.
- EL2 is not implemented.

- FEAT_ECV is not implemented.
- The Effective value of SCR_EL3.{NS,RW} is {1,0}.
- CNTHCTL_EL2.ECV is 0.
- SCR_EL3.ECVEn is 0.

and in the table itself:

- 'CNTPCT_EL0' is replaced by 'PhysicalCountInt()'.
- 'CNTPOFF_EL2' is replaced by 'physical offset', and footnote (a) is removed.
- 'CNTVOFF_EL2' is unchanged.

Equivalent changes are made in section G3.3 (Self-hosted trace timestamps) to Table G3-1 'Timestamp used for trace'.

2.11 D18199

In section D8.11.3 (Common event numbers), in the subsection 'Common microarchitectural events', in the following event descriptions:

- `0x8134`, 'DLTB_HWUPD, Data TLB hardware update of translation table'.
- `0x8135`, 'ILTB_HWUPD, Instruction TLB hardware update of translation table'.

The following clarification is added:

> Each attempted hardware update of a translation table entry is counted once. If the PE requires multiple translation table walk accesses to perform an update, this counts as a single update. If the update fails because it would not be atomic and has to be retried, each retry is counted.

2.12 D18206

In section J1.3.1 (shared/debug), the code in the function Halt() that reads:

```
Halt(bits(6) reason)
    ...
    if (HaveBTIExt() &&
        !(reason IN {DebugHalt_Step_Normal, DebugHalt_Step_Exclusive,
 DebugHalt_Step_NoSyndrome,
                     DebugHalt_Breakpoint, DebugHalt_HaltInstruction}) &&
        ConstrainUnpredictableBool(Unpredictable_ZEROBTYPE)) then
        if UsingAArch32() then
            spsr_32<11:10> = '00';
        else
            spsr_64<11:10> = '00';
```

is updated to read as:

```
Halt(bits(6) reason)
    boolean is_async = FALSE;
    Halt(reason, is_async);

Halt(bits(6) reason, boolean is_async)
    ...
    if (HaveBTIExt() && !is_async && !(reason IN {DebugHalt_Step_Normal,
 DebugHalt_Step_Exclusive,
        DebugHalt_Step_NoSyndrome, DebugHalt_Breakpoint, DebugHalt_HaltInstruction})
 &&
        ConstrainUnpredictableBool(Unpredictable_ZEROBTYPE)) then
        if UsingAArch32() then
            spsr_32<11:10> = '00';
        else
            spsr_64<11:10> = '00';
```

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

```
CheckPendingResetCatch()
    if HaltingAllowed() && EDESR.RC == '1' then
        Halt(DebugHalt_ResetCatch);
```

is updated to read as:

```
CheckPendingResetCatch()
    if HaltingAllowed() && EDESR.RC == '1' then
        boolean is_async = TRUE;
```

2.13 D18305

In section D8.11.3 (Common event numbers), in the subsection ‘Common microarchitectural events’, in the following event descriptions, the phrase ‘ALU operation counts’ is corrected to ‘ALU operations’:

- `0x80C0`, FP_SCALE_OPS_SPEC, Scalable floating-point element ALU operation counts Speculatively executed.
- `0x80C1`, FP_FIXED_OPS_SPEC, Non-scalable floating-point element ALU operation counts Speculatively executed.
- `0x80C2`, FP_HP_SCALE_OPS_SPEC, Scalable half-precision floating-point element ALU operation counts Speculatively executed.
- `0x80C3`, FP_HP_FIXED_OPS_SPEC, Non-scalable half-precision floating-point element ALU operation counts Speculatively executed.
- `0x80C4`, FP_SP_SCALE_OPS_SPEC, Scalable single-precision floating-point element ALU operation counts Speculatively executed.
- `0x80C5`, FP_SP_FIXED_OPS_SPEC, Non-scalable single-precision floating-point element ALU operation counts Speculatively executed.

- `0x80C6`, FP_DP_SCALE_OPS_SPEC, Scalable double-precision floating-point element ALU operation counts Speculatively executed.
- `0x80C7`, FP_DP_FIXED_OPS_SPEC, Non-scalable double-precision floating-point element ALU operation counts Speculatively executed.
- `0x80C8`, INT_SCALE_OPS_SPEC, Scalable integer element ALU operation counts Speculatively executed.
- `0x80C9`, INT_FIXED_OPS_SPEC, Non-scalable integer element ALU operation counts Speculatively executed.

2.14 D18330

The ArmARM 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 the ArmARM, 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).

2.15 D18354

In section C5.6.2 (CPP RCTX, Cache Prefetch Prediction Restriction by Context), the line that currently reads:

> Cache prefetch predictions determined by the actions of code in the target execution context or contexts appearing in program order before the instruction cannot influence speculative execution occurring after the instruction is complete and synchronized.

is corrected to read:

> The actions of code in the target execution context or contexts appearing in program order before the instruction cannot exploitatively control cache prefetch predictions occurring after the instruction is complete and synchronized.

Equivalent clarifications are made in sections C6.2.67 (CPP) and G8.2.34 (CPPRCTX, Cache Prefetch Prediction Restriction by Context).

2.16 C18405

In section D6.7 (PE handling of Tag Check Fault), the following text is added:

> If a Tag check fault configured to asynchronously set a bit in TFSR_ELx or TFSRE0_EL1 has not been observed in TFSR_ELx or TFSRE0_EL1, and has not been synchronized by a DSB, or by the effects of SCTLR_ELx.ITSB, when a write to a System register affecting Tag check fault handling occurs, the effect of the Tag check fault is **CONSTRAINED UNPREDICTABLE** between the effect of the old and new value of the System register.

2.17 C18433

In section D8.3 (Behavior on overflow), before the paragraph which starts 'For all 64-bit counters ...', the following text is added:

> The update of PMOVSCLR occurs synchronously with the update of the counter.

Also, in section D8.2.2 (A reasonable degree of inaccuracy), the text that reads:

> Where a direct write to a Performance Monitors control register disables a counter, and is followed by a Context synchronization event, any subsequent indirect read of the control register by the Performance Monitors to determine whether the counter is enabled will return the updated value. Any subsequent direct read of the counter will return the value at the point the counter was disabled.

is extended to read:

> Where a direct write to a Performance Monitors control register disables a counter, and is followed by a Context synchronization event, any subsequent indirect read of the control register by the Performance Monitors to determine whether the counter is enabled will return the updated value. Any subsequent direct read of the counter or counter overflow status flags will return the value at the point the counter was disabled.

2.18 D18456

The following function EL3SDDTrapPriority() is added to section J1.3.3 (shared/functions):

```
boolean EL3SDDTrapPriority()
    return (Halted() && EDSCR.SDD == '1' &&
            boolean IMPLEMENTATION_DEFINED "EL3 trap priority when SDD == '1'");
```

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

```
    if Enable == '0' then return X;
    elsif TrapEL2 then TrapPACUse(EL2);
    elsif TrapEL3 then TrapPACUse(EL3);
```

```
    else return AddPAC(X, Y, APDAKey_EL1, TRUE);
```

is updated to read:

```
    if Enable == '0' then
        return X;
    elsif TrapEL3 && EL3SDDTrapPriority() then
        UNDEFINED;
    elsif TrapEL2 then
        TrapPACUse(EL2);
    elsif TrapEL3 then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            TrapPACUse(EL3);
    else
        return AddPAC(X, Y, APDAKey_EL1, TRUE);
```

An equivalent change is made for AddPACDB(), AddPACGA(), AddPACIA(), AddPACIB(), AuthDA(), AuthDB(), AuthIA(), and AuthIB().

In section J1.1.2 (aarch64/exceptions), in the function CheckST64BV0Enabled(), the code that reads:

```
CheckST64BV0Enabled()
    boolean trap = FALSE;
    bits(25) iss = ZeroExtend('1');  // 0x1
    bits(2) target_el;
    ...
    if !trap && PSTATE.EL != EL3 then
        trap = HaveEL(EL3) && SCR_EL3.EnAS0 == '0';
        target_el = EL3;

    if trap then LDST64BTrap(target_el, iss);
```

is updated to read:

```
CheckST64BV0Enabled()
    boolean trap = FALSE;
    bits(25) iss = ZeroExtend('1');  // 0x1
    bits(2) target_el;

    if (PSTATE.EL != EL3 && HaveEL(EL3) &&
        SCR_EL3.EnAS0 == '0' && EL3SDDTrapPriority()) then
        UNDEFINED;
   ...
    if !trap && PSTATE.EL != EL3 then
        trap = HaveEL(EL3) && SCR_EL3.EnAS0 == '0';
        target_el = EL3;

    if trap then
        if target_el == EL3 && Halted() && EDSCR.SDD == '1' then
        UNDEFINED;
    else
        LDST64BTrap(target_el, iss);
```

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

```
CheckNormalSVEEnabled()
    boolean disabled;
    ...
    // Check if access disabled in CPTR_EL3
    if HaveEL(EL3) then
        if CPTR_EL3.EZ == '0' then SVEAccessTrap(EL3);
        if CPTR_EL3.TFP == '1' then AArch64.AdvSIMDFPAccessTrap(EL3);
```

is updated to read:

```
CheckNormalSVEEnabled()
    boolean disabled;
    if (HaveEL(EL3) && (CPTR_EL3.EZ == '0' || CPTR_EL3.TFP == '1') &&
        EL3SDDTrapPriority()) then
        UNDEFINED;
    ...
    // Check if access disabled in CPTR_EL3
    if HaveEL(EL3) then
        if CPTR_EL3.EZ == '0' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                SVEAccessTrap(EL3);

        if CPTR_EL3.TFP == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.AdvSIMDFPAccessTrap(EL3);
```

An equivalent check for CPTR_EL3.TFP is added in section J1.1.2 (aarch64/exceptions) for AArch64.CheckFPAdvSIMDTrap(), and in section J1.2.2 (aarch32/exceptions) for AArch32.CheckFPAdvSIMDTrap().

2.19 D18465

In section D13.2.119 (SCTLR_EL2, System Control Register (EL2)), for all of the bits that are described as having a function when HCR_EL2.E2H==1 && HCR_EL2.TGE==1 and being **RES0** otherwise, it is clarified that these bits:

- Are **RES0** when HCR_EL2.E2H==0, so software should write the value 0.
- Are ignored by hardware when HCR_EL2.E2H==1 && HCR_EL2.TGE==0, but software doesn't have to set the value 0.
- Have their described effect when HCR_EL2.E2H==1 && HCR_EL2.TGE==1.

2.20 E18471

In section D13.2.26 (CCSIDR_EL1, Current Cache Size ID Register), in the LineSize, bits [2:0] field description, the following Note is added:

> Note: The C++ 17 specification has two defined parameters relating to the granularity of memory that does not interfere. For generic software and tools, Arm will set the hardware_destructive_interference_size parameter to 256 bytes and the hardware_constructive_interference_size parameter to 64 bytes.

The equivalent change is made in section G8.2.24 (CCSIDR, Current Cache Size ID Register).

2.21 R18485

In section I5.8.8 (ERRDEVAFF, Device Affinity Register), the following text is added to the end of the Purpose section:

> Depending on the **IMPLEMENTATION DEFINED** nature of the system, it might be possible that ERRDEVAFF is read before system firmware has configured the group of error records and/or the PE or group of PEs that the group of error records has affinity with. When this is the case, ERRDEVAFF reads as zero.

2.22 D18520

In section I5.8.31 (ERR<n>PFGF, Pseudo-fault Generation Feature Register, n = 0 - 65534), the text in MV, bit [12] that reads:

> `0b0` When an injected error is recorded, the node might update ERR<n>MISC<m>. If any syndrome is recorded by the node in ERR<n>MISC<m>, then ERR<n>STATUS.MV is set to `0b1`. ERR<n>PFGCTL.MV is **RES0**.

is updated to read:

> `0b0` ERR<n>PFGCTL.MV not supported. When an injected error is recorded, the node might update ERR<n>MISC<m>. If any syndrome is recorded by the node in ERR<n>MISC<m>, then ERR<n>STATUS.MV is set to `0b1`. If the node always sets ERR<n>.STATUS.MV to `0b1` when recording an injected error, then ERR<n>PFGCTL.MV might be RAO/WI. Otherwise, ERR<n>PFGCTL.MV is **RES0**.

Corresponding updates are made to section I5.8.30 (ERR<n>PFGCTL, Pseudo-fault Generation Control Register, n = 0 - 65534), for bit [12] 'when the node supports this control'. Similar corrections are made for the ERR<n>PFGF.AV and ERR<n>PFGCTL.AV controls.

2.23 D18530

In section D2.10.5 (Determining the memory location that caused a Watchpoint Exception), in the subsection 'Address recorded for Watchpoint exceptions generated by instructions other than data cache maintenance instructions', the text that reads:

> The address recorded must be both:

is changed to read:

> For Watchpoint exceptions generated by a DC ZVA, DC GVA, or DC GZVA instruction, the address recorded is an address accessed by the instruction that triggered the watchpoint.
>
> Otherwise, the address recorded must be both:

2.24 D18557

In section I3.1.4 (Access permissions for external views of the Performance Monitors), in tables I3-1 and I3-2, the rows that read:

Offset	Register	Domain	Off	DLK	OSLK	EPMAD	Default	SLK
0x600-0x6FC	**IMPLEMENTATION DEFINED** registers	-	-	-	-	-	-	-

are changed to read:

Offset	Register	Domain	Off	DLK	OSLK	EPMAD	Default	SLK
0x600-0x7FC	**IMPLEMENTATION DEFINED** registers	-	-	-	-	-	-	-

and the following row is added to both tables:

Offset	Register	Domain	Off	DLK	OSLK	EPMAD	Default	SLK
0xE30-0xE3C	**IMPLEMENTATION DEFINED** registers	-	-	-	-	-	-	-

Similarly, in section I5.2.1 (Performance Monitors external register views), in tables I5-1 and I5-2, the entries that read:

Name	Type	Description	Offset
-	-	**IMPLEMENTATION DEFINED**	0x600-0x6FC

are changed to read:

Name	Type	Description	Offset
-	-	**IMPLEMENTATION DEFINED**	0x600-0x7FC

and the following row is added to both tables:

Name	Type	Description	Offset
-	-	**IMPLEMENTATION DEFINED**	0xE30-0xE3C

2.25 C18578

In the following sections:

- G8.4.10 (PMEVCNTR<n>, Performance Monitors Event Count Registers, n = 0 - 30).
- 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).

The accessibility pseudocode for accesses at EL1 using AArch32 is updated to indicate that FEAT_FGT causes a Trap exception to EL2.

2.26 R18641

In section H9.4.19 (CTIDEVID, CTI Device ID register 0), in the NUMTRIG, bits [13:8] definition, the text that reads:

> Number of triggers implemented. IMPLEMENTATION DEFINED. This is one more than the index of the largest trigger, rather than the actual number of triggers.

is updated to read:

> Upper bound for number of triggers. The indices of all implemented input and output triggers are less than this value. The value of this field is IMPLEMENTATION DEFINED.

and the text that reads:

> There is no guarantee that any of the implemented triggers, including the highest numbered, are connected to any components.

is updated to read:

> There is no guarantee that all of the input and output triggers, including the highest numbered, are connected to any components, or that the implementation of input and output triggers is symmetrical.

In section H9.4.24 (CTIINEN<n>, CTI Input Trigger to Output Channel Enable registers, n = 0 - 31), the text in ‘Configurations’ that reads:

> If input trigger n is not connected, the behavior of CTIINEN<n> is IMPLEMENTATION DEFINED.

is changed to read:

> If input trigger n is not implemented or connected, CTIINEN<n> is **RES0**.

The equivalent change is made in section H9.4.29 (CTIOUTEN<n>, CTI Input Channel to Output Trigger Enable registers, n = 0 - 31).

Similarly, in section H9.4.25 (CTIINTACK, CTI Output Trigger Acknowledge register), in the ACK<n>, bit [n], for n = 31 to 0 definition, the text that reads:

> Otherwise, if any of the following are true, it is IMPLEMENTATION DEFINED whether writes to ACK<n> are ignored:

is relaxed to read:

> Otherwise, if any of the following are true, ACK<n> is **RES0**:

2.27 E18645

In section D7.2.1 (The physical counter), after the text that currently reads:

> Neither view of the physical counter ensures that:
>
> - Context changes occurring in program order before the read of the counter have been synchronized.
> - Accesses to memory appearing in program order after the read of the counter are executed before the counter has been read.

the following text is added:

> Where there is a dependency through registers dependency from the read of the physical counter to a Register effect generated by a read or write, the read or write will be executed after the read of the counter.

The equivalent change is made in section D7.2.2 (The virtual counter).

Also in section D7.2.1 (The physical counter), example D7-2 ‘Ensuring reads of the physical counter occur after previous memory accesses’ is changed to read:

> To ensure that all previous memory accesses have completed and all previous context changes have been synchronized before the read of the counter, one of the following sequences should be used:
>
> either:
>
> DSB
>
> ISB
>
> MRS Xn, CNTPCT{SS}_EL0 ; either view of the physical counter has the same effect in this example

> or
>
> DMB
>
> LDR Xa, [Xd] ; this could be any memory location, for example the stack pointer
>
> CBZ Xa, next
>
> next
>
> ISB ; this ISB is not needed if the MRS is accessing CNTPCTSS_EL0
>
> MRS Xn, CNTPCT{SS}_EL0
>
> To ensure that a memory access occurs only after a read of the counter, then either of the following sequences should be used:
>
> MRS Xn, CNTPCT{SS}_EL0 ; either view of the physical counter has the same effect in this example
>
> ISB
>
> LDR Xa, [Xb] ; this load will be executed after the counter has been read
>
> or
>
> MRS Xn, CNTPCT{SS}_EL0 ; either view of the physical counter has the same effect in this example
>
> EOR Xm, Xn, Xn ;
>
> LDR Xa, [Xb, Xm] ; this load will be executed after the counter has been read

In section D7.2.2 (The virtual counter), the equivalent change is made to Example D7-4 ‘Ensuring reads of the virtual counter occur after previous memory accesses’.

Equivalent changes are also made in sections G6.2.1 (The physical counter) in Example G6-2 and G6.2.2 (The virtual counter) in Example G6-4.

2.28 D18685

In section H6.3 (Core power domain power states), the text preceding the definition of the ‘Normal’ power state is changed to read:

> The Arm architecture does not define the power states of the PE as these are not normally visible to software. However, they are visible to the external debugger. Arm A-profile external debug uses a four logical power states model for the Core power domain. Mechanisms for entering a

> low-power state on page D1-4639 describes the architectural mechanisms for entering low-power states.
>
> When the PE enters a low-power state other than Powerdown by executing a WFI, WFIT, WFE, or WFET instruction, it remains in that low-power state until it receives a *wake-up event*. See Mechanisms for entering a low-power state on page D1-4639 for the definition of wake-up events. When halting is allowed, an External Debug Request is a wake-up event. In addition, if the PE enters the Standby state for any reason, it will leave that state to service an External Debug Request debug event.
>
> The four logical power states are as follows:

Correspondingly, the text in the description of the 'Standby' state that reads:

> In a typical implementation, the PE enters standby by executing a WFI or WFE instruction, and exits on a wake-up event.

and the text in the description of the 'Retention' state that reads:

> The OS takes some measures, including **IMPLEMENTATION DEFINED** code sequences and registers, to reduce energy consumption.

and the text in the description of the 'Powerdown' state that reads:

> The OS takes some measures to reduce energy consumption by turning the Core power domain off.

are removed. Also in the description of the 'Standby' state, the text that reads:

> - An External Debug Request debug event is a wake-up event when halting is allowed. This means that the PE must exit standby to handle the debug event. If the PE executed a WFE or a WFI instruction to enter standby, then it retires that instruction.

and the text in the description of the 'Retention' state that reads:

> External Debug Request debug events stay pending and registers in the Core power domain cannot be accessed.

are removed.

Additionally, in section H3.5.1 (Synchronization and External Debug Request debug events), the text that reads:

> Otherwise, when halting is allowed, External Debug Request debug events must be taken in finite time, without requiring the synchronization of any necessary change to the external authentication interface.

is clarified to read:

Otherwise, when all of the following are true, External Debug Request debug events must be taken in finite time, without requiring the synchronization of any necessary change to the external authentication interface:

- Halting is allowed.
- The PE is not in a low-power state that is not required to exit on an External Debug Request. See Core power domain power states on page H6-10012.

External Debug Request is a wake-up event for WFI, WFIT, WFE, or WFET instructions. See Mechanisms for entering a low-power state on page D1-4639.

2.29 D18689

In section F5.1.268 (UBFX), the **UNPREDICTABLE** checks are moved to the operational pseudocode for UBFX.

The decode pseudocode for UBFX A1 that reads:

```
d = UInt(Rd); n = UInt(Rn);
lsbit = UInt(lsb); widthminus1 = UInt(widthm1);
if d == 15 || n == 15 then UNPREDICTABLE;
```

is updated to read as:

```
d = UInt(Rd); n = UInt(Rn);
lsbit = UInt(lsb); widthminus1 = UInt(widthm1);
msbit = lsbit + widthminus1;
if d == 15 || n == 15 then UNPREDICTABLE;
if msbit > 31 then UNPREDICTABLE;
```

The decode pseudocode for UBFX T1 that reads:

```
d = UInt(Rd); n = UInt(Rn);
lsbit = UInt(imm3:imm2); widthminus1 = UInt(widthm1);
if d == 15 || n == 15 then UNPREDICTABLE; // Armv8-A removes UNPREDICTABLE for R13
```

is updated to read as:

```
d = UInt(Rd); n = UInt(Rn);
lsbit = UInt(imm3:imm2); widthminus1 = UInt(widthm1);
msbit = lsbit + widthminus1;
if d == 15 || n == 15 then UNPREDICTABLE; // Armv8-A removes UNPREDICTABLE for R13
if msbit > 31 then UNPREDICTABLE;
```

The operational pseudocode for UBFX that reads:

```
if ConditionPassed() then
    EncodingSpecificOperations();
    msbit = lsbit + widthminus1;
    if msbit <= 31 then
```

```
        R[d] = ZeroExtend(R[n]<msbit:lsbit>, 32);
    else
        UNPREDICTABLE;
```

is updated to read as:

```
if ConditionPassed() then
    EncodingSpecificOperations();
    R[d] = ZeroExtend(R[n]<msbit:lsbit>, 32);
```

Equivalent changes are made to decode and operational pseudocode in the following sections:

- F5.1.179 (SBFX).
- F5.1.20 (BFI).
- F5.1.19 (BFC).

2.30 D18698

In section D10.6.2 (Additional information for each profiled branch or exception return), the text that currently reads:

> If the optional behavior in FEAT_SPEv1p2 is implemented, the target address of the most recently executed sampled branch that was taken and retired in program order before the sampled operation.

is corrected to read:

> If the optional behavior in FEAT_SPEv1p2 is implemented, the target address of the most recently executed branch that was taken and retired in program order before the sampled operation.

In the same section, the subsection ‘Last Branch Target’ is renamed to ‘Previous Branch Target’, and the text in this subsection that reads:

> If FEAT_SPEv1p2 is implemented, PMSIDR_EL1.PBT optionally adds the capability to record a packet for each event that provides the target address of the previous taken branch.
>
> If enabled, the profiling operation records the target address of the most recently sampled branch that was taken and retired in program order before the sampled operation.

is corrected to read:

> FEAT_SPEv1p2 adds an optional capability to record a packet for each event that provides the target address of the previous taken branch. PMSIDR_EL1.PBT describes whether this feature is implemented.
>
> When implemented, the profiling operation records the target address of the most recent branch that was taken and retired in program order before the sampled operation.

2.31 C18700

In section D2.4 (Enabling debug exceptions from the current Exception level), Table D2-6 'Whether debug exceptions are enabled from the current Exception level' is replaced with the following text:

> Debug exceptions are enabled from the current Exception level, where EL_D is the Exception level that Table D2-3 on page D2-4662 defines, as follows:
>
> When executing at any Exception level that is higher than EL_D:
>
> - Breakpoint Instruction exceptions are enabled.
> - All other debug exceptions are disabled.
>
> When executing at EL_D:
>
> - Breakpoint Instruction exceptions are enabled.
> - All other debug exceptions are disabled if either of the following is true:
>   - The Local (kernel) Debug Enable bit, MDSCR_EL1.KDE, is 0.
>   - The Debug exception mask bit, PSTATE.D, is 1.
>
>   Otherwise enabled. This means that a debugger must explicitly enable these debug exceptions from EL_D by setting MDSCR_EL1.KDE to 1 and PSTATE.D to 0.
>
> When executing at EL1, EL_D is EL2, and FEAT_NV2 is implemented:
>
> - Watchpoint debug exceptions generated by a System register access converted to a memory access because HCR_EL2.NV2 is 1 are disabled if MDSCR_EL1.KDE is 0 and enabled if MDSCR_EL1.KDE is 1. The value of PSTATE.D is ignored.
> - All other debug exceptions are enabled.
>
> Otherwise:
>
> - All debug exceptions are enabled.

Additionally, in section D2.3 (Routing debug exceptions), the text that reads:

> The routing of debug exceptions is as follows:

is moved to after Table D2-2, 'Whether debug exceptions are enabled from the current Security state'.

2.32 C18704

In section D1.3.8 (Configurable instruction controls), the following Note is added after rule IBJBMF:

> Note: Many **CONSTRAINED UNPREDICTABLE** behaviors for instructions include an allowance that the **CONSTRAINED UNPREDICTABLE** instruction behaves the same way as a closely related instruction

that is not **CONSTRAINED UNPREDICTABLE**. In those cases, the instruction enable, disable or trap control that causes in exception on the closely related instruction will cause the same exception on the **CONSTRAINED UNPREDICTABLE** instruction.

2.33 D18705

The change that was communicated under this issue is updated in D18731.

2.34 D18712

In section H8.6 (External debug interface registers), in Table H8-2 'External debug interface register map', the following rows:

Offset	Mnemonic	Register, or additional information
`0xFC0`	EDDEVID2	EDDEVID, External Debug Device ID register 0 on page H9-10049
`0xFC4`	EDDEVID1	EDDEVID1, External Debug Device ID register 1 on page H9-10051
`0xFC8`	EDDEVID	EDDEVID2, External Debug Device ID register 2 on page H9-10053

are changed to read:

Offset	Mnemonic	Register, or additional information
`0xFC0`	EDDEVID2	EDDEVID2, External Debug Device ID register 2 on page H9-10105
`0xFC4`	EDDEVID1	EDDEVID1, External Debug Device ID register 1 on page H9-10103
`0xFC8`	EDDEVID	EDDEVID, External Debug Device ID register 0 on page H9-10101

2.35 E18715

In section C5.2.4 (DIT, Data Independent Timing), in the description of DIT, bit [24], the instructions SQRDMULH, SQDMULH, and SQRDMLAH are added to the bullet list that begins 'A subset of those instructions which use the SIMD&FP register file. These instructions are:'.

Additionally, in the following sections:

- C7.2.288 (SQDMULH (by element)).
- C7.2.289 (SQDMULH (vector)).
- C7.2.293 (SQRDMLAH (by element)).
- C7.2.294 (SQRDMLAH (vector)).
- C7.2.297 (SQRDMULH (by element)).
- C7.2.298 (SQRDMULH (vector)).

The following is added to the instruction descriptions:

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.

2.36 C18720

In section D10.8.3 (Faults and watchpoints), the following entry is added to Table D10-5 'Faults':

Fault	Conditions
External abort on translation table walk or translation table update	The translation of a virtual address to a physical address might generate an External abort on the translation table walk or translation table update, and be treated as a synchronous MMU fault. See also External aborts on page D10-5179.

In the same section, following the list starting 'If a write to the Profiling Buffer generates a fault and PMBSR_EL1.S is 0, then a Profiling Buffer management event is generated:', the following text is added:

If a write to the Profiling Buffer generates an External abort on a translation table walk or translation table update, it is **IMPLEMENTATION DEFINED** whether PMBSR_EL1.EA is set to 1 or unchanged.

In section D10.8.4 (External aborts), following the list of options starting 'When a write to the Profiling Buffer generates an External abort, including an External abort on a translation table walk or translation table update, the permitted **IMPLEMENTATION DEFINED** behaviors are:', the following text is added:

The choice of **IMPLEMENTATION DEFINED** behavior is not required to be the same for each of:

- An External abort on the write to the Profiling Buffer.
- An External abort on a translation table walk.
- An External abort on a translation table update.

An External abort on a translation table walk or translation table update might be treated as a synchronous MMU fault, as described by Faults and watchpoints on page D10-5178.

2.37 R18721

In section D13.2.52 (HFGITR_EL2, Hypervisor Fine-Grained Instruction Trap Register), the following changes are made:

In the definitions of DCCVAU, bit [7], ICIVAU, bit [2], ICIALLU, bit [1], and ICIALLUIS, bit [0], the following text is added:

> If the Point of Unification is before any level of data cache, it is **IMPLEMENTATION DEFINED** whether the execution of the affected instruction is trapped when the value of this control is 1.

In the definitions of DCCVAC, bit [54], DCCIVAC, bit [10], and DCIVAC, bit [3], the following text is added:

> If the Point of Coherence is before any level of data cache, it is **IMPLEMENTATION DEFINED** whether the execution of the affected instruction is trapped when the value of this control is 1.

In the definition of DCCVAP, bit [8], the following text is added:

> If the Point of Persistence is before any level of data cache, it is **IMPLEMENTATION DEFINED** whether the execution of the affected instruction is trapped when the value of this control is 1.

In the definition of DCCVADP, bit [9], the following text is added:

> If the Point of Deep Persistence is before any level of data cache, it is **IMPLEMENTATION DEFINED** whether the execution of the affected instruction is trapped when the value of this control is 1.

2.38 D18731

In sections D13.3.27 (SDER32_EL2, AArch32 Secure Debug Enable Register) and G8.3.35 (SDER, Secure Debug Enable Register), the following condition is added to the SUIDEN, bit [0] description:

> When EL3 is implemented:
>
> Otherwise:
>
> Reserved, **RES0**.

Also in section D13.3.27 (SDER32_EL2, AArch32 Secure Debug Enable Register), SUNIDEN, bit [1] is updated to read:

> Secure User Non-Invasive Debug Enable.
>
> `0b0` This bit has no effect on non-invasive debug. `0b1` Non-invasive debug is allowed in Secure EL0 using AArch32.
>
> When Secure EL1 is using AArch32, the forms of non-invasive debug affected by this control are:

- The PC Sample-based Profiling Extension. See About the PC Sample-based Profiling Extension on page H7-10028.
- When SelfHostedTraceEnabled() is FALSE, processor trace.
- When EL3 is implemented, Performance Monitors.

When Secure EL1 is using AArch64, this bit has no effect.

The equivalent changes are made in the following locations:

- Section D13.3.28 (SDER32_EL3, AArch32 Secure Debug Enable Register), to SUNIDEN, bit [1].
- Section G8.3.35 (SDER, Secure Debug Enable Register), to SUNIDEN, bit [1].

In section H7.1.1 (Controlling the PC Sample-based Profiling Extension), the text that reads:

- EL3 is not implemented.
- EL3 is implemented, the PE is executing in Secure state, and non-invasive debug is allowed by the IMPLEMENTATION DEFINED authentication interface ExternalSecureNoninvasiveDebugEnabled().
- EL3 is implemented, EL3 or EL1 is using AArch32, the PE is executing at EL0 in Secure state, and the value of SDER.SUNIDEN is 1.

is corrected to:

- The PE is executing in Secure state, and non-invasive debug is allowed by the IMPLEMENTATION DEFINED authentication interface ExternalSecureNoninvasiveDebugEnabled().
- EL3 or EL1 is using AArch32, the PE is executing at EL0 in Secure state, and the value of SDER.SUNIDEN is 1.

In section J1.3.4 (shared/trace), the pseudocode in the function TraceAllowed() that reads:

```
case ss of
    when SS_NonSecure return ExternalNoninvasiveDebugEnabled();
    when SS_Secure    return ExternalSecureNoninvasiveDebugEnabled();
```

is corrected to read:

```
return ExternalNoninvasiveDebugAllowed();
```

In section D3.1.2 (Register controls to enable self-hosted trace), the text that reads:

While SelfHostedTraceEnabled() == FALSE, ExternalSecureNoninvasiveDebugEnabled() and ExternalNoninvasiveDebugEnabled() control whether tracing is prohibited or allowed in each Security state.

is corrected to read:

> While SelfHostedTraceEnabled() is FALSE, ExternalNoninvasiveDebugAllowed() controls whether tracing is prohibited or allowed in each Security state.

The equivalent change is made in section G3.1.2 (Register controls to enable self-hosted trace).

In section D3.2 (Prohibited regions in self-hosted trace), the following text is deleted:

> If FEAT_TRF is implemented but not enabled, tracing is prohibited in Secure state when ExternalSecureNoninvasiveDebugEnabled() == FALSE.

The equivalent text is deleted in section G3.2 (Prohibited regions in self-hosted trace).

In section G2.6 (Summary of permitted routing and enabling of debug exceptions), in Table G2-8 'Breakpoint, Watchpoint, and Vector Catch exceptions', footnote 'b' is added to the SUIDEN column header, and is changed to read:

> If EL3 is not implemented, behavior is as if SPD and `0b11` and SUIDEN is `0b0`.

2.39 C18732

In section I5.5.34 (PMPCSR, Program Counter Sample Register), under the heading 'Accessing the PMPCSR', the following clarification is added:

> Note: A 32-bit access to PMPCSR[63:32] does not update the PC sample registers. Only a 64-bit access to PMPCSR[63:0] or a 32-bit access to PMPCSR[31:0] updates the PC sample registers. This includes the value a subsequent 32-bit read of PMPCSR[63:32] will return.

2.40 D18735

In section D5.9.1 (Use of ASIDs and VMIDs to reduce TLB maintenance requirements), under 'ASID', the following text is deleted:

> For a symmetric multiprocessor cluster where a single operating system is running on the set of processing elements, the Arm architecture requires all ASID values to be assigned uniquely within any single Inner Shareable domain. In other words, each ASID value must have the same meaning to all processing elements in the system.

2.41 D18736

In section I5.8.5 (ERRCRIC0, Critical Error Interrupt Configuration Register 0), under the heading 'Accessing the ERRCRICR0', the following text is added:

> If the implementation does not use the recommended layout for the ERRIRQCR<n> registers, accesses to ERRCRICR0 are IMPLEMENTATION DEFINED.

> ERRCRICR0 ignores writes if all of the following are true:
>
> - The implementation uses the recommended layout for the ERRIRQCR<n> registers.
> - ERRCRICR2.NSMSI configures the physical address space for message signaled interrupts as Secure.
> - Accessed as a Non-secure access.

The equivalent changes are made in the following sections:

- I5.8.6 (ERRCRICR1, Critical Error Interrupt Configuration Register 1).
- I5.8.7 (ERRCRICR2, Critical Error Interrupt Configuration Register 2).
- I5.8.11 (ERRERICR0, Error Recovery Interrupt Configuration Register 0).
- I5.8.12 (ERRERICR1, Error Recovery Interrupt Configuration Register 1).
- I5.8.13 (ERRERICR2, Error Recovery Interrupt Configuration Register 2).
- I5.8.14 (ERRFHICR0, Fault Handling Interrupt Configuration Register 0).
- I5.8.15 (ERRFHICR1, Faulting Handling Interrupt Configuration Register 1).
- I5.8.16 (ERRFHICR2, Faulting Handling Interrupt Configuration Register 2).

In section I5.8.7 (ERRCRICR2, Critical Error Interrupt Configuration Register 2), the text that reads:

> When the component supports configuring the Security attribute for messaged signaled interrupts and the component does not allow Non-secure writes to ERRCRICR2:
>
> Security attribute. Defines the physical address space for message signaled interrupts.
>
> `0b0` Secure.
>
> `0b1` Non-secure.
>
> The reset behavior of this field is:
>
> - On a Error recovery reset,this field resets to an IMPLEMENTATION DEFINED VALUE.
>
> When the component allows Non-secure writes to ERRCRICR2:
>
> Reserved, **RES0**.
>
> Security attribute. Defines the physical address space for message signaled interrupts.
>
> The Security attribute used for message signaled interrupts is Non-secure.

is changed to read:

> When the component supports configuring the physical address space for message signaled interrupts:
>
> Non-secure message signaled interrupt. Defines the physical address space for message signaled interrupts.

`0b0` Secure physical address space.

`0b1` Non-secure physical address space.

The reset behavior of this field is:

- On an Error recovery reset, this field resets to an IMPLEMENTATION DEFINED VALUE.

Accessing this field has the following behavior:

- If accessed as a Non-secure access, access to this field is RES1.
- Otherwise, access to this field is RW.

The equivalent changes are made in the following sections:

- I5.8.13 (ERRERICR2, Error Recovery Interrupt Configuration Register 2).
- I5.8.16 (ERRFHICR2, Faulting Handling Interrupt Configuration Register 2).

2.42 C18738

In section D13.2.40 (FAR_EL1, Fault Address Register (EL1)), the text in the Bits [63:0] description that reads:

- For a Data abort generated by the MMU, the value is within the address range of the current translation granule, aligned to the size of the current translation granule of the address that generated the Data Abort. Bits [n-1:0] of the value are **UNKNOWN**, where 2^n is the translation granule size in bytes.

is updated to read:

- For a Data Abort generated by the MMU, the value is within the address range of the relevant translation granule, aligned to the size of the relevant translation granule of the address that generated the Data Abort. Bits[(n-1):0] of the value are **UNKNOWN**, where 2^n is the relevant translation granule size in bytes. For the purpose of calculating the relevant translation granule, if the MMU is disabled for stage 1 translation, then the stage 1 granule size is equal to 2^{64}. The relevant translation granule is:
  - For MMU faults generated at stage 1, the stage 1 translation granule.
  - If FEAT_RME is implemented, for a synchronous data abort generated as the result of a GPF, the smaller of the stage 1 translation granule and the configured granule size in GPCCR_EL3.PGS.
- For a Data Abort generated by a Tag Check failure, the value is the lowest address that failed the Tag Check within the block size of the load or store.
- For a Watchpoint exception, the value is an address range of the size defined by the DCZID_EL0.BS field. This address does not need to be the element with a watchpoint, but can be some earlier element.
- Otherwise, the value is the lowest address in the block size of the load or store.

The equivalent changes are made in the following sections:

- D13.2.41 (FAR_EL2, Fault Address Register (EL2)).
- D13.2.42 (FAR_EL3, Fault Address Register (EL3)).

In section D13.2.55 (HPFAR_EL2, Hypervisor IPA Fault Address Register), the following text is added to the FIPA, bits [38:0] and FIPA, bits [34:0] descriptions:

> When FEAT_MOPS is implemented, the value in FIPA on a synchronous exception from any of the Memory Copy and Memory Set instructions represents the first element that has not been copied or set, and is determined as follows:
>
> - For a Data Abort generated by the MMU, the value is within the address range of the relevant translation granule, aligned to the size of the relevant translation granule of the address that generated the Data Abort. Bits[(n-1):0] of the value are **UNKNOWN**, where 2^n is the relevant translation granule size in bytes. For the purpose of calculating the relevant translation granule, if the MMU is disabled for a stage of translation, then the relevant translation granule size is equal to 2^{64} for stage 1, and the PARange for stage 2. The relevant translation granule is:
>   - For MMU faults generated at stage 2, the smaller of the stage 1 translation granule and the stage 2 translation granule.
>   - If FEAT_RME is implemented, for a synchronous data abort generated as the result of a GPF, the smallest of the stage 1 translation granule, the stage 2 translation granule, and the configured granule size in GPCCR_EL3.PGS.
> - Otherwise, the value is the lowest address in the block size of the load or store.

2.43 D18739

In section J1.1.1 (aarch64/debug), in the CollectTimeStamp() function, each case of the following code:

```
    return TimeStamp_Virtual;
```

is changed to read:

```
    return if IsInHost() then TimeStamp_Physical else TimeStamp_Virtual;
```

2.44 D18741

In section J1.3.1 (shared/debug), the code in the function ExitDebugState() that reads as:

```
ExitDebugState()
    ...
    // If this is an illegal return, SetPSTATEFromPSR() will set PSTATE.IL.
    ...
```

```
    if UsingAArch32() then
        if ConstrainUnpredictableBool(Unpredictable_RESTARTALIGNPC) then new_pc<0> =
'0';
        // AArch32 branch
        BranchTo(new_pc<31:0>, BranchType_DBGEXIT, branch_conditional);
    else
        // If targeting AArch32 then possibly zero the 32 most significant bits of
the target PC
        if spsr<4> == '1' &&
ConstrainUnpredictableBool(Unpredictable_RESTARTZEROUPPERPC) then
            new_pc<63:32> = Zeros();
```

is updated to read:

```
ExitDebugState()
    ...
    boolean illegal_psr_state = IllegalExceptionReturn(spsr);
    // If this is an illegal return, SetPSTATEFromPSR() will set PSTATE.IL.
    ...
    if UsingAArch32() then
        if ConstrainUnpredictableBool(Unpredictable_RESTARTALIGNPC) then new_pc<0> =
'0';
        // AArch32 branch
        BranchTo(new_pc<31:0>, BranchType_DBGEXIT, branch_conditional);
    else
        // If targeting AArch32 then PC[63:32,1:0] might be set to UNKNOWN.
        if illegal_psr_state && spsr<4> == '1' then
            new_pc<63:32> = bits(32) UNKNOWN;
            new_pc<1:0> = bits(2) UNKNOWN;
```

2.45 D18742

In section D6.7 (PE handling of Tag Check Faults), the following text is added:

> If an instruction that accesses memory generates both a synchronous Data Abort and a Tag Check Fault, where Tag Check Faults are configured to set a bit in a TFSR_ELx or TFSRE0_EL1 register, and the bit is 0, the bit becomes **UNKNOWN**.

2.46 R18746

In section B2.7.2 (Device memory), in the subsection ‘Multi-register loads and stores that access Device memory’, the following paragraph is added:

> The architecture permits that the non-speculative execution of an instruction that loads or stores more than one general-purpose or SIMD and floating-point register might result in repeated accesses to the same address.

The equivalent edit is made in section E2.8.2 (Device Memory), in the subsection ‘Multi-register loads and stores that access Device memory’.

2.47 D18764

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 the 64-bit external PMU programmers' model extension is implemented', the following row is removed:

Offset	Register	Domain	Off	DLK	OSLK	EPMAD	Default
0xA00+8xn	PMEVFILTR<n>	Core	Error	Error	Error	Error	**RES0**

Similarly, in section I5.2.1 (Performance Monitors external register views), in Table I5-1 'Performance Monitors external register views when the 64-bit external PMU programmers' model extension is implemented', the following row is removed:

Name	Type	Description	Offset
PMEVFILTR<n>	**RES0**	Performance Monitors Event Type Select Register <n>	0xA00+8xn

2.48 R18765

In section D6.2.1 (Cache activity and Allocation Tags), the text that reads:

> When Allocation Tags are evicted from a cache entry at a cache level, the evicted Allocation Tags can overwrite Allocation Tags in memory that have been written by another observer only if the following are true:
>
> - The entry contains a memory location where the Allocation Tags have been written to by an observer in the Shareability domain of that memory location.
> - The maximum size of the memory that can be overwritten is defined by the Cache Write-Back Granule in CTR_EL0.

is changed to read:

> When Allocation tags are evicted from a cache entry at a cache level, the evicted Allocation tags can overwrite Allocation tags in memory that has been written by another observer if either, or any of the following are true:
>
> - The Allocation tags associated with memory within an address range of the size of the Cache Write-Back Granule, aligned to that size, have been written by an observer in the Shareability domain of that memory location.
> - Data within an address range of the size of Cache Write-Back Granule in CTR_EL0, aligned to that size, of the associated data has been written to by an observer in the Shareability domain of that memory location.
>
> Note: If an implementation supports both clean write-back of Allocation Tags, and storage for Allocation Tags can be repurposed to store data when not being used to store Allocation Tags, it is an implementation responsibility to prevent the clean write-back of Allocation Tags from overwriting data.

In section C5.3.13 (DC CISW, Data or unified Cache line Clean and Invalidate by Set/Way), the following Purpose text is deleted:

> When FEAT_MTE is implemented, this instruction might clean and invalidate Allocation Tags from caches.

The equivalent changes are made in sections C5.3.14 (DC CIVAC, Data or unified Cache line Clean and Invalidate by VA to PoC) to C5.3.18 (DC CVAP, Data or unified Cache line Clean by VA to PoP).

In the following sections:

- C5.3.26 (DC ISW, Data or unified Cache line Invalidate by Set/Way).
- C5.3.27 (DC IVAC, Data or unified Cache line Invalidate by VA to PoC).

The Purpose text that reads:

> When FEAT_MTE is implemented, this instruction might invalidate Allocation Tags from caches. When it invalidates Allocation Tags from caches, it also cleans them.

is corrected to read:

> When FEAT_MTE2 is implemented, this instruction might invalidate Allocation Tags from caches. When it invalidates Allocation Tags from caches, it also cleans them.

2.49 D18773

In section H7.1.3 (Permitted behavior that might make the PC Sample-based profiling registers **UNKNOWN**), the text that reads:

> When FEAT_MOPS and FEAT_PCSRv8p2 are implemented, if no instruction has been retired since the last read of PMPCSR[31:0], then the value of PMPCSR[31:0] is unchanged.

is changed to read:

> When FEAT_MOPS and FEAT_PCSRv8p2 are not implemented, if no branch instruction has been retired since the last read of PMPCSR[31:0], then the value of PMPCSR[31:0] is **UNKNOWN**.

2.50 D18775

In section H2.4.2 (Executing A64 instructions in Debug state), in the subsection ‘A64 instructions that are unchanged in Debug state’, under the list of ‘Barriers’ instructions, the entry that currently reads:

> - When FEAT_TRF is implemented, TSB CSYNC.

is changed to read:

- When FEAT_TRBE is implemented, TSB CSYNC.

Additionally, in section H2.4.3 (Decode tables), in Table H2-4 ‘A64 instructions that are unchanged in Debug state’, the row for ‘CSDB, ESB, PSB’ is changed to read ‘CSDB, ESB, PSB, and, if FEAT_TRBE is implemented, TSB’, and the cell that reads ‘!=010’ is replaced with ‘op2’ for this row.

2.51 D18776

In section C6.2 (Alphabetical list of A64 base instructions), in the CPYFP* instructions, the code that reads as:

```
if stage == MOPSStage_Prologue then
    ...
    if supports_option_a then
        PSTATE.C = '0';
        // Copy in the forward direction offsets the arguments.
        toaddress = toaddress + cpysize;
        fromaddress = fromaddress + cpysize;
        cpysize = Zeros(64) - cpysize;
    else
        PSTATE.C = '1';
    PSTATE.N = '0';
    PSTATE.V = '0';
    PSTATE.Z = '0';
...
    // Check if this version is consistent with the state of the call.
    if zero_size_exceptions || SInt(cpysize) != 0 then
        if supports_option_a then
            if PSTATE.C == '1' then
                ...
        else
            if PSTATE.C == '0' then
                ...
...
if stage == MOPSStage_Prologue then
    X[n] = cpysize;
    X[d] = toaddress;
    X[s] = fromaddress;
```

Is updated to read:

```
bits(4) nzcv = PSTATE.<N,Z,C,V>;
...
if stage == MOPSStage_Prologue then
    ...
    if supports_option_a then
        nzcv = '0000';
        // Copy in the forward direction offsets the arguments.
        toaddress = toaddress + cpysize;
        fromaddress = fromaddress + cpysize;
        cpysize = Zeros(64) - cpysize;
    else
        nzcv = '0010';
...
    // Check if this version is consistent with the state of the call.
    if zero_size_exceptions || SInt(cpysize) != 0 then
        if supports_option_a then
            if nzcv<1> == '1' then // PSTATE.C
                ...
        else
```

```
            if nzcv<1> == '0' then // PSTATE.C
                ...
...
if stage == MOPSStage_Prologue then
    X[n] = cpysize;
    X[d] = toaddress;
    X[s] = fromaddress;
    PSTATE.<N,Z,C,V> = nzcv;
```

The equivalent changes are made for the following instructions throughout the section:

- CPYP*.
- SETP*.
- SETGP*.

2.52 D18781

In section B2.2.5 (Concurrent modification and execution of instructions), the following bullet item is added to the Note in point 2:

> - In a multiprocessor system, the DC CVAU and IC IVAU are broadcast to all PEs within the Inner Shareable domain of the PE running this sequence.

In the same section, point 3 is changed to read:

> When the modified instructions are observable, each PE that is executing the modified instructions must execute an ISB or perform a context synchronizing event to ensure execution of the modified instructions:

2.53 R18784

In section D8.2 (Accuracy of the Performance Monitors), the following text is added:

> However, the Performance Monitors allow for:
>
> - **IMPLEMENTATION DEFINED** controls, such as those in ACTLR registers, that software must configure before using certain PMU events. For example, to configure how the PE generates PMU events for components such as external caches and external memory.

2.54 D18788

In section H2.4.1 (PSTATE in Debug state), the text that reads:

> Instructions executed in Debug state indirectly read PSTATE.{UAO, PAN, IL, E, M, nRW, EL, SP} as they would in Non-debug state.

is corrected to read:

> Instructions executed in Debug state indirectly read PSTATE.{TCO, UAO, PAN, IL, E, M, nRW, EL, SP} as they would in Non-debug state.

In section H2.4.8 (Accessing registers in Debug state), the text that reads:

> PSTATE.{IL, E, M, nRW, EL, SP} are indirectly read by instructions executed in Debug state, but all other PSTATE fields are ignored and cannot be observed.

is corrected to read:

> PSTATE.{TCO, UAO, PAN, IL, E, M, nRW, EL, SP} are indirectly read by instructions executed in Debug state, but all other PSTATE fields are ignored and cannot be observed.

In section D6.8 (PE generation of Tag Checked and Tag Unchecked accesses), the following line is added:

> Instructions in Debug state follow the same rules for generation of Tag Checked and Tag Unchecked accesses as in Non-Debug state. See Chapter H2 Debug state for more information.

In section D6.8.1 (Tag Unchecked accesses), the following line is removed:

> Data accesses by an external Debugger may generate Tag Checked accesses. See Chapter H2 Debug State for more information.

2.55 D18791

In section I3.1.1 (Endianness and supported access sizes), the text that reads:

> - Support word-aligned 32-bit accesses to access 32-bit registers or either half of a 64-bit register mapped to a doubleword-aligned pair of adjacent 32-bit locations, even if no PE in the system implements AArch32.

is updated to read:

> - Support word-aligned 32-bit accesses to access 32-bit registers or either half of a 64-bit register mapped to a doubleword-aligned pair of adjacent 32-bit locations, even if all components with direct memory access to the PMU support making 64-bit accesses.

Equivalent changes are made in the following sections:

- H8.2 (Endianness and supported access sizes).
- I1.1 (Supported access sizes).
- I2.1.1 (Registers in the system level implementation of the Generic Timer), subsection ‘Endianness and supported access sizes’.
- I4.1 (About the external interface to the Activity Monitors Extension registers).

2.56 D18794

In the following sections:

- C5.5.4 (TLBI ALLE2, TLBI ALLE2NXS, TLB Invalidate All, EL2).
- C5.5.5 (TLBI ALLE2IS, TLBI ALLE2ISNXS, TLB Invalidate All, EL2, Inner Shareable).
- C5.5.6 (TLBI ALLE2OS, TLBI ALLE2OSNXS, TLB Invalidate All, EL2, Outer Shareable).

The text that reads:

> If FEAT_RME is implemented, one of the following applies:
>
> - SCR_EL3.{NSE, NS} is {0, 0} and the entry would be required to translate an address using the Secure EL1&0 translation regime.
> - SCR_EL3.{NSE, NS} is {0, 1} and the entry would be required to translate an address using the Non-secure EL1&0 translation regime.
> - SCR_EL3.{NSE, NS} is {1, 1} and the entry would be required to translate an address using the Realm EL1&0 translation regime.
>
> If FEAT_RME is not implemented, one of the following applies:
>
> - SCR_EL3.NS is 0 and the entry would be required to translate an address using the Secure EL1&0 translation regime.
> - SCR_EL3.NS is 1 and the entry would be required to translate an address using the Non-secure EL1&0 translation regime.

is corrected to read:

> If FEAT_RME is implemented, one of the following applies:
>
> - SCR_EL3.{NSE, NS} is {0, 0} and the entry would be required to translate an address using the Secure EL2&0 translation regime.
> - SCR_EL3.{NSE, NS} is {0, 1} and the entry would be required to translate an address using the Non-secure EL2&0 translation regime.
> - SCR_EL3.{NSE, NS} is {1, 1} and the entry would be required to translate an address using the Realm EL2&0 translation regime.
>
> If FEAT_RME is not implemented, one of the following applies:
>
> - SCR_EL3.NS is 0 and the entry would be required to translate an address using the Secure EL2&0 translation regime.
> - SCR_EL3.NS is 1 and the entry would be required to translate an address using the Non-secure EL2&0 translation regime.

2.57 D18795

In section H2.4.1 (PSTATE in Debug state), the text that reads:

> PSTATE.{N, Z, C, V, Q, GE, IT, T, SS, D, A, I, F, SSBS, ALLINT} are all ignored in Debug state:
> - There are no conditional instructions in Debug state.
> - In AArch32 state, the PE executes only T32 instructions and PSTATE.IT is ignored.
> - Asynchronous exceptions and debug events are ignored.
> - Software step is inactive.

is corrected to read:

> PSTATE.{N, Z, C, V, Q, GE, IT, T, SS, BTYPE, D, A, I, F, SSBS, ALLINT} are all ignored in Debug state:
> - There are no conditional instructions in Debug state.
> - In AArch32 state, the PE executes only T32 instructions and PSTATE.IT is ignored.
> - In AArch64 state, PSTATE.BTYPE is treated as `0b00`.
> - Asynchronous exceptions and debug events are ignored.
> - Software step is inactive.

2.58 D18797

In section B2.3.10 (Restrictions on the effects of speculation), in the subsection 'Restrictions on exploitative control of speculative execution', the bullet that reads:

> - Code1 has control in determining the choice of the architecture state that causes the irreversible change to the microarchitectural state.

is clarified to read:

> - Code1 has control in determining the choice of the architecture state that the irreversible change to the microarchitectural state is indicative of.

2.59 C18798

In section D13.2.65 (ID_AA64MMFR1_EL1, AArch64 Memory Model Feature Register 1), the definition of nTLBPA, bits [51:48]:

> `0b0000` The intermediate caching of translation table walks might include non-coherent caches of previous valid translation table entries since the last completed relevant TLBI applicable to the PE where either:

- The caching is indexed by the physical address of the location holding the translation table entry.
- The caching is used for stage 1 translations and is indexed by the intermediate physical address of the location holding the translation table entry.

`0b0001` The intermediate caching of translation table walks does not include non-coherent caches of previous valid translation table entries since the last completed TLBI applicable to the PE where either:

- The caching is indexed by the physical address of the location holding the translation table entry.
- The caching is used for stage 1 translations and is indexed by the intermediate physical address of the location holding the translation table entry.

is clarified to read:

`0b0000` The intermediate caching of translation table walks might include non-coherent physical translation caches.

`0b0001` The intermediate caching of translation table walks does not include non-coherent physical translation caches.

Non-coherent physical translation caches are non-coherent caches of previous valid translation table entries since the last completed relevant TLBI applicable to the PE, where either:

- The caching is indexed by the physical address of the location holding the translation table entry.
- The caching is used for stage 1 translations and is indexed by the intermediate physical address of the location holding the translation table entry.

The equivalent edits are made in the following sections:

- D13.2.86 (ID_MMFR5_EL1, Memory Model Feature Register 5).
- G8.2.97 (ID_MMFR5 AArch32 Memory Model Feature Register 5).

2.60 D18800

In section D13.5.17 (PMUSERENR_EL0, Performance Monitors User Enable Register), the EN, bit [0] description is updated to read:

Enable EL0 accesses to the Performance Monitor registers. This applies to the following register accesses:

AArch64:

- MRS or MSR accesses to PMCR_EL0, PMOVSCLR_EL0, PMSELR_EL0, PMCCNTR_EL0, PMXEVTYPER_EL0, PMXEVCNTR_EL0, PMCNTENSET_EL0, PMCNTENCLR_EL0, PMOVSSET_EL0, PMEVCNTR<n>_EL0, PMEVTYPER<n>_EL0, PMCCFILTR_EL0.

- MSR accesses to PMSWINC_EL0.
- MRS accesses to PMCEID0_EL0, PMCEID1_EL0.

AArch32:

- MRC and MCR accesses to PMCR, PMOVSR, PMSELR, PMCCNTR, PMXEVTYPER, PMXEVCNTR, PMCNTENSET, PMCNTENCLR, PMOVSSET, PMEVCNTR<n>, PMEVTYPER<n>, PMCCFILTR.
- MCR accesses to PMSWINC.
- MRC accesses to PMCEID0, PMCEID1.
- If FEAT_PMUv3p1 is implemented, MRC accesses to PMCEID2, and PMCEID3.

`0b0` EL0 access to the specified registers is trapped, unless access is enabled by another field in this register.

`0b1` EL0 access to the specified registers is allowed, unless trapped by a higher priority exception.

When not enabled by any of the PMUSERENR_EL0.{ER, CR, SW, EN} controls, an accesses to the register at EL0 is trapped to EL1, or to EL2 when EL2 is implemented and enabled for the current Security state and HCR_EL2.TGE is 1.

Trapped MRS and MSR accesses are reported using EC syndrome value `0x18`.

Trapped MRC and MCR accesses are reported using EC syndrome value `0x03`.

Trapped MRRC and MCRR accesses are reported using EC syndrome value `0x04`.

The reset behavior of this field is:

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

Equivalent changes are made to the {ER, CR, SW} fields, and to the PMUSERENR.{ER, CR, SW, EN} fields in section G8.4.18 (PMUSERENR, Performance Monitors User Enable Register).

2.61 D18802

In section I3.1.2 (Differences in the external views of the Performance Monitors registers), in the subsection 'External views of the Performance Monitors registers when the 64-bit external PMU programmers' model extension is implemented', the following text is deleted:

- PMCEID01, provided as a concatenation of PMCEID0 and PMCEID1.
- PMCEID23, provided as a concatenation of PMCEID2 and PMCEID3.

Correspondingly:

- 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 the 64-bit external PMU

programmers' model extension is implemented', the entries for PMCEID01 and PMCEID23 are deleted.

- In section I5.2.1 (Performance Monitors external register views), in Table I5-1 'Performance Monitors external register views when the 64-bit external PMU programmers' model extension is implemented', the entries for PMCEID01 and PMCEID23 are deleted.

2.62 C18805

In section D5.7.2 (Enhanced support for nested virtualization), in the subsection 'Loads and stores generated by transforming register accesses', the text that reads:

> When a System register access is transformed into a memory access, that memory access has a defined format:
>
> - The addressees the memory access is translated by the EL2 translation regime.
> - The endianness of the memory access is defined by SCTLR_EL2.EE.

is clarified to read:

> When a System register access is transformed into a memory access, that memory access has a defined format:
>
> - The memory access is made as if it comes from EL2 as configured by the current EL2 state, meaning that:
>   - The address of the memory access is translated by the EL2 translation regime if HCR_EL2.E2H==0 or the EL2&0 translation regime if HCR_EL2.E2H==1.
>   - The endianness of the memory access is defined by SCTLR_EL2.EE.

2.63 D18812

In section D8.11.6 (Meaningful combinations of SVE events), in the subsection 'Vector loop efficiency', the table is updated to read:

Numerator	Denominator	Vector loop metric
SVE_PLOOP_ELTS_SPEC × VL ÷ 128	SVE_PLOOP_TERM_SPEC	Source level iterations per loop
SVE_PLOOP_TEST_SPEC	SVE_PLOOP_TERM_SPEC	Vectorized iterations per loop
SVE_PLOOP_ELTS_SPEC × VL ÷ 128	SVE_PLOOP_TEST_SPEC	Parallelism per vector loop

2.64 D18815

In section A2.7.1 (Architectural features added by Armv8.4), in the description of 'FEAT_S2FWB, Stage 2 forced Write-Back', the text that reads:

> FEAT_S2FWB reduces the requirement of additional cache maintenance instructions in systems where the data Cacheability attributes used by the Guest operating system are different from those expected by the Hypervisor.

is updated to read:

> FEAT_S2FWB reduces the requirement of additional cache maintenance instructions in systems where the data Cacheability attributes used by the Guest operating system are different from those expected by the Hypervisor. If this feature is implemented, there is no meaningful distinction between the Inner and Outer Shareability domains for accesses to Normal Cacheable memory.

2.65 D18816

In section H2.4.2 (Executing instructions in Debug state), in the subsection 'A64 instructions that are unchanged in Debug state', under the list of Memory Tagging Extension instructions, the entries that read:

> - STG.
> - STZG.
> - ST2G.
> - STZ2G.
> - STGP.

are changed to read:

> - STG (signed offset).
> - STZG (signed offset).
> - ST2G (signed offset).
> - STZ2G (signed offset).
> - STGP (signed offset).

In section H2.4.3 (Decode tables), in Table H2-4 'A64 instructions that are unchanged in Debug state', the following changes are made:

- The description for the row 'STG <Xt|SP>, [<Xn|SP>{, #<simm}]!, Signed offset' is corrected to: 'STG <Xt|SP>, [<Xn|SP>{, #<simm}], Signed offset'.
- The value for bit 10 in the row for 'ST2G <Xt|>, [Xt|SP>{, #<simm>}] Signed offset' is corrected to '0'.

- The value for bit 10 in the row for 'STZG <XT|SP>, [<Xn|SP{. #<simm>}]! Signed offset' is corrected to '0', and the description is corrected to 'STZG <XT|SP>, [<Xn|SP{. #<simm>}], Signed offset'.
- The description for the row 'STZ2G <XT|SP>, [<Xn|SP{. #<simm>}]! Signed offset' is corrected to 'STZ2G <XT|SP>, [<Xn|SP{. #<simm>}], Signed offset'.

2.66 D18825

In section C6.2.116 (DSB), the following changes are made:

- In the description of the <option> field for the memory nXS barrier variant, the references to 'CRm<3:2>' are changed to 'imm2'.
- The assembler syntax for the memory nXS barrier variant is changed to 'DSB <option>nXS'.

2.67 D18829

In section A2.7.1 (Architectural features added by Armv8.4), subsection 'FEAT_TRF, Self-hosted Trace Extensions', the text that reads:

> If an ETM Architecture PE Trace Unit is implemented and the ETM PE Trace Unit includes System register access to its control registers, this feature is mandatory. If a different PE Trace Unit is implemented or the ETM PE Trace Unit does not include System register access to its control registers, this feature is OPTIONAL.

is corrected to:

> If Armv8.4 and an ETM Architecture trace unit is implemented, this feature is mandatory. If this feature is implemented, the trace unit must include System register access to its control registers.

2.68 R18832

In section D10.6.2 (Additional information for each profiled branch or exception return), the following statement is added:

> It is **IMPLEMENTATION DEFINED** whether an ISB instruction is treated as a branch to the next instruction.

2.69 C18836

In section H6.4.1 (Powerup request mechanism if FEAT_DoPD is implemented), the text that reads:

> On reset, if FEAT_DoPD is implemented, DBGPRCR.CORENPDRQ is set to an **IMPLEMENTATION DEFINED** choice between 0 and 1 if the powerup request is implemented and asserted, and 0 otherwise.

is replaced with:

> On a Cold reset, if FEAT_DoPD is implemented, DBGPRCR.CORENPDRQ is set to an **IMPLEMENTATION DEFINED** choice between 0 and 1 if the powerup request is implemented and asserted, and 0 otherwise.

In section H6.4.2 (Powerup request mechanism if FEAT_DoPD is not implemented), the text that reads:

> On reset, if FEAT_DoPD is not implemented, DBGPRCR.CORENPDRQ is set to the value of EDPRCR.COREPURQ.

is replaced with:

> On Cold reset, if FEAT_DoPD is not implemented, DBGPRCR.CORENPDRQ is set to the value of EDPRCR.COREPURQ.

2.70 D18838

In section C6.2.378 (TLBI):

- The CRn field value ‘1000’ is corrected to ‘100x’ in the instruction encoding diagram.
- The FEAT_XS variants of the TLBI operations are added to the <tlbi_op> assembler symbol definition.

Furthermore, in section C6.2.372 (SYS), in the subsection ‘Alias conditions’, the table entry which reads:

Alias	Is preferred when
TLBI	CRn == ‘1000’ && SysOp(op1,‘1000’,CRm,op2) == Sys_TLBI

is corrected to read:

Alias	Is preferred when
TLBI	CRn == ‘100x’ && SysOp(op1,CRn,CRm,op2) == Sys_TLBI

2.71 C18842

In section I5.5.14 (AMDEVARCH, Activity Monitors Device Architecture Register), the text in the ARCHID, bits [15:0] description that reads:

> For AMU:
>
> - Bits [15:12] are the architecture version, `0x0`.
> - Bits [11:0] are the architecture part number, `0xA66`.
>
> This corresponds to AMU architecture version AMUv1.

is changed to read:

> For AMU:
>
> - Bits [19:16] are the minor architecture version, `0x0`.
> - Bits [15:12] are the major architecture version, `0x0`.
> - Bits [11:0] are the architecture part number, `0xA66`.
>
> This corresponds to a generic AMU, version 1.0.

2.72 C18843

The current description of FEAT_LPA2 in *Arm® Architecture Reference Manual, for A-profile architecture, Issue H.a* lacks clarity between the ability to describe the size of the output address as having 52 bits, and there being 52 bits of physical address. This will be rectified in a future release of *Arm® Architecture Reference Manual, for A-profile architecture*.

2.73 D18853

In section J1.1.3 (aarch64/functions), the following is added to the description of the pseudocode function AArch64.ChooseRandomNonExcludedTag():

```
// This function can read RGSR_EL1 and/or write RGSR_EL1 to an IMPLEMENTATION
 DEFINED value.
// If it is not capable of writing RGSR_EL1.SEED[15:0] to zero from a previous non-
zero RGSR_EL1.SEED
// value, it is IMPLEMENTATION DEFINED whether the randomness is significantly
 impacted if
// RGSR_EL1.SEED[15:0] is set to zero.
```

In section C6.2.130 (IRG), the following line is removed from the operational pseudocode:

```
RGSR_EL1 = bits(64) UNKNOWN;
```

In section D13.2.107 (RGSR_EL1, Random Allocation Tag Seed Register), the field descriptions are changed to read:

> When GCR_EL1.RRND == 0:
>
> Bits [63:24]
>
> Reserved, RES0.
>
> SEED, bits [23:8]
>
> Seed register used for generating values returned by RandomAllocationTag().
>
> The reset behavior of this field is:
> - On a Warm reset, this field resets to an architecturally **UNKNOWN** value.
>
> Bits [7:4]
>
> Reserved, RES0.
>
> TAG, bits [3:0]
>
> Tag generated by the most recent IRG instruction.
>
> The reset behavior of this field is:
> - On a Warm reset, this field resets to an architecturally **UNKNOWN** value.
>
> When GCR_EL1.RRND == 1:
>
> Bits [63:56]
>
> Reserved, RES0.
>
> SEED, bits [55:8]
>
> IMPLEMENTATION DEFINED
>
> Note: Software is recommended to avoid writing SEED[15:0] with a value of zero, unless this has been generated by the PE in response to an earlier value with SEED being non-zero.
>
> The reset behavior of this field is:
> - On a Warm reset, this field resets to an architecturally **UNKNOWN** value.
>
> Bits [7:4]
>
> Reserved, RES0.
>
> TAG, bits [3:0]

Tag generated by the most recent IRG instruction.

The reset behavior of this field is:

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

2.74 D18876

In section J1.1.3 (aarch64/functions), the code in function AArch64.FaultSyndrome() that reads as:

```
(bits(25), bits(5)) AArch64.FaultSyndrome(boolean d_side, FaultRecord fault)
....
    if !HaveFeatLS64() && HaveRASExt() && IsAsyncAbort(fault) then
        iss<12:11> = fault.errortype; // SET

    if d_side then
        if HaveFeatLS64() && fault.acctype == AccType_ATOMICLS64 then
            if (fault.statuscode IN {Fault_AccessFlag,
                                     Fault_Translation, Fault_Permission}) then
                (iss2, iss<24:14>, iss<12:11>) = LS64InstructionSyndrome();

....
    return (iss, iss2);
```

is changed to:

```
(bits(25), bits(5)) AArch64.FaultSyndrome(boolean d_side, FaultRecord fault)
....
    if HaveRASExt() && fault.statuscode == Fault_SyncExternal then
        iss<12:11> = fault.errortype; // SET
    if d_side then
        if HaveFeatLS64() && fault.acctype == AccType_ATOMICLS64 then
            if (fault.statuscode IN {Fault_AccessFlag, Fault_Translation,
                                     Fault_Permission}) then
                (iss2, iss<24:14>) = LS64InstructionSyndrome();

....
        if HaveFeatLS64() && fault.statuscode IN {Fault_AccessFlag,
 Fault_Translation,
                                                  Fault_Permission} then
            iss<12:11> = GetLoadStoreType();

....
    return (iss, iss2);
```

Also in section J1.1.3 (aarch64/functions), the function definition for LS64InstructionSyndrome() that reads as:

```
// LS64InstructionSyndrome()
// =========================
// Returns the syndrome information and LST for a Data Abort by a
// ST64B, ST64BV, ST64BV0, or LD64B instruction. The syndrome information
// includes the ISS2, extended syndrome field, and LST.

(bits(5), bits(11), bits(2)) LS64InstructionSyndrome();
```

is changed to:

```
// LS64InstructionSyndrome()
// =========================
// Returns the syndrome information and LST for a Data Abort by a
// ST64B, ST64BV, ST64BV0, or LD64B instruction. The syndrome information
// includes the ISS2, extended syndrome field.

(bits(5), bits(11)) LS64InstructionSyndrome();
```

Similarly, a new function GetLoadStoreType() is added to the same section:

```
// GetLoadStoreType()
// ==================
// Returns the Load/Store Type. Used when a Translation fault,
// Access flag fault, or Permission fault generates a Data Abort.

bits(2) GetLoadStoreType();
```

2.75 D18880

In section C5.1.4 (op0==`0b01`, cache maintenance, TLB maintenance, address translation, prediction restriction, and BRBE instructions), in the subsection 'Prediction restriction instructions', the following table entries:

Instruction	op0	op1	CRn	CRm	op2
CPP RCTX, Xt	1	3	7	3	5
DVP RCTX, Xt	1	3	7	3	7

are corrected to read:

Instruction	op0	op1	CRn	CRm	op2
DVP RCTX, Xt	1	3	7	3	5
CPP RCTX, Xt	1	3	7	3	7

2.76 R18907

In section D5.5.7 (Combining the stage 1 and stage 2 attributes, EL1&0 translation regime) in the subsection 'Combining the stage 1 and stage 2 Cacheability attributes for Normal memory', the following text is added:

> When the first stage of the translation regime specifies Normal memory with Cacheability other than Write-back cacheable, HCR_EL2.FWB is set to 1, and the stage 2 Page or Block descriptor [4:2] is set to `0b110`, it is **IMPLEMENTATION DEFINED** whether Atomic memory accesses or Exclusives are supported, in the same way as it is for accesses to memory locations whose resultant memory type is Normal memory other than Write-back cacheable.

2.77 D18914

In section C3.8.3 (Uniform DSP operations), in Table C3-152 'Uniform DSP instructions', the 'Mnemonic' and 'See' entries for the instruction 'Signed saturating rounding doubling multiply-add high to accumulator (unpredicated)' are changed from SQRDCMLAH to SQRDMLAH.

2.78 C18918

In section J1.1.4 (aarch64/instrs), a new function AltDecodeBitMasks() is added.

Similarly, the enumeration Unpredictable is added to section J1.3.3 (shared/functions):

```
enumeration Unpredictable {// VMSR on MVFR
                           Unpredictable_VMSR,
                           // Writeback/transfer register overlap (load)
                           Unpredictable_WBOVERLAPLD,
                           // Writeback/transfer register overlap (store)
                           Unpredictable_WBOVERLAPST,
                           // Load Pair transfer register overlap
                           Unpredictable_LDPOVERLAP,
                           // Store-exclusive base/status register overlap
                           Unpredictable_BASEOVERLAP,
                           // Store-exclusive data/status register overlap
                           Unpredictable_DATAOVERLAP,
                           // Load-store alignment checks
                           Unpredictable_DEVPAGE2,
                           // Instruction fetch from Device memory
                           Unpredictable_INSTRDEVICE,
                           // Reserved CPACR value
                           Unpredictable_RESCPACR,
                           // Reserved MAIR value
                           Unpredictable_RESMAIR,
                           // Effect of SCTLR_ELx.C on Tagged attribute
                           Unpredictable_S1CTAGGED,
                           // Reserved Stage 2 MemAttr value
                           Unpredictable_S2RESMEMATTR,
                           // Reserved TEX:C:B value
                           Unpredictable_RESTEXCB,
                           // Reserved PRRR value
                           Unpredictable_RESPRRR,
                           // Reserved DACR field
                           Unpredictable_RESDACR,
                           // Reserved VTCR.S value
                           Unpredictable_RESVTCRS,
                           // Reserved TCR.TnSZ value
                           Unpredictable_RESTnSZ,
                           // Reserved SCTLR_ELx.TCF value
                           Unpredictable_RESTCF,
                           // Tag stored to Device memory
                           Unpredictable_DEVICETAGSTORE,
                           // Reserved SCTLR_ELx.EnCSR value
                           Unpredictable_RESEnCSR,
                           // Reserved CSRCR_ELx.SIZE value
                           Unpredictable_RESCSRCRSIZE,
                           // Out-of-range TCR.TnSZ value
                           Unpredictable_OORTnSZ,
                           // IPA size exceeds PA size
                           Unpredictable_LARGEIPA,
                           // Syndrome for a known-passing conditional A32
 instruction
```

```
                        Unpredictable_ESRCONDPASS,
                        // Illegal State exception: zero PSTATE.IT
                        Unpredictable_ILZEROIT,
                        // Illegal State exception: zero PSTATE.T
                        Unpredictable_ILZEROT,
                        // Debug: prioritization of Vector Catch
                        Unpredictable_BPVECTORCATCHPRI,
                        // Debug Vector Catch: match on 2nd halfword
                        Unpredictable_VCMATCHHALF,
                        // Debug Vector Catch: match on Data Abort
                        // or Prefetch abort
                        Unpredictable_VCMATCHDAPA,
                        // Debug watchpoints: non-zero MASK and non-ones BAS
                        Unpredictable_WPMASKANDBAS,
                        // Debug watchpoints: non-contiguous BAS
                        Unpredictable_WPBASCONTIGUOUS,
                        // Debug watchpoints: reserved MASK
                        Unpredictable_RESWPMASK,
                        // Debug watchpoints: non-zero MASKed bits of address
                        Unpredictable_WPMASKEDBITS,
                        // Debug breakpoints and watchpoints: reserved control
bits
                        Unpredictable_RESBPWPCTRL,
                        // Debug breakpoints: not implemented
                        Unpredictable_BPNOTIMPL,
                        // Debug breakpoints: reserved type
                        Unpredictable_RESBPTYPE,
                        // Debug breakpoints: not-context-aware breakpoint
                        Unpredictable_BPNOTCTXCMP,
                        // Debug breakpoints: match on 2nd halfword of
instruction
                        Unpredictable_BPMATCHHALF,
                        // Debug breakpoints: mismatch on 2nd halfword of
instruction
                        Unpredictable_BPMISMATCHHALF,
                        // Debug: restart to a misaligned AArch32 PC value
                        Unpredictable_RESTARTALIGNPC,
                        // Debug: restart to a not-zero-extended AArch32 PC value
                        Unpredictable_RESTARTZEROUPPERPC,
                        // Zero top 32 bits of X registers in AArch32 state
                        Unpredictable_ZEROUPPER,
                        // Zero top 32 bits of PC on illegal return to
                        // AArch32 state
                        Unpredictable_ERETZEROUPPERPC,
                        // Force address to be aligned when interworking
                        // branch to A32 state
                        Unpredictable_A32FORCEALIGNPC,
                        // SMC disabled
                        Unpredictable_SMD,
                        // Stage-2 EL1&0 mismatch MPU configuration.
                        Unpredictable_S2EL1MPUCONFIG,
                        // FF speculation
                        Unpredictable_NONFAULT,
                        // Zero top bits of Z registers in EL change
                        Unpredictable_SVEZEROUPPER,
                        // Load mem data in NF loads
                        Unpredictable_SVELDNFDATA,
                        // Write zeros in NF loads
                        Unpredictable_SVELDNFZERO,
                        // SP alignment fault when predicate is all zero
                        Unpredictable_CHECKSPNONEACTIVE,
                        // Zero top bits of ZA registers in EL change
                        Unpredictable_SMEZEROUPPER,
                        // HCR_EL2.<NV,NV1> == '01'
                        Unpredictable_NVNV1,
                        // Reserved shareability encoding
                        Unpredictable_Shareability,
                        // Access Flag Update by HW
                        Unpredictable_AFUPDATE,
                        // Consider SCTLR[].IESB in Debug state
                        Unpredictable_IESBinDebug,
```

```
                            // Bad settings for PMSFCR_EL1/PMSEVFR_EL1/PMSLATFR_EL1
                            Unpredictable_BADPMSFCR,
                            // Zero saved BType value in SPSR_ELx/DPSR_EL0
                            Unpredictable_ZEROBTYPE,
                            // Timestamp constrained to virtual or physical
                            Unpredictable_EL2TIMESTAMP,
                            Unpredictable_EL1TIMESTAMP,
                             // Reserved MDCR_EL3.<NSTBE,NSTB> or
 MDCR_EL3.<NSPBE,NSPB> value
                             Unpredictable_RESERVEDNSxB,
                            // WFET or WFIT instruction in Debug state
                            Unpredictable_WFxTDEBUG,
                            // Address does not support LS64 instructions
                            Unpredictable_LS64UNSUPPORTED,
                            // Misaligned exclusives, atomics, acquire/release
                            // to region that is not Normal Cacheable WB are atomic
                            Unpredictable_MISALIGNEDATOMIC,
                            // Clearing DCC/ITR sticky flags when instruction is in
 flight
                            Unpredictable_CLEARERRITEZERO,
                            // ALUEXCEPTIONRETURN when in user/system mode in
                            // A32 instructions
                            Unpredictable_ALUEXCEPTIONRETURN,
                            // Trap to register in debug state are ignored
                            Unpredictable_IGNORETRAPINDEBUG,
                            // Compare DBGBVR.RESS for BP/WP
                            Unpredictable_DBGxVR_RESS,
                            // Inaccessible event counter
                            Unpredictable_PMUEVENTCOUNTER,
                            // Reserved PMSCR.PCT behaviour.
                            Unpredictable_PMSCR_PCT,
                            // Generate BRB_FILTRATE event on BRB injection
                            Unpredictable_BRBFILTRATE,
};
```

2.79 C18919

In sections C5.5.21 (TLBI RIPAS2E1, TLBI RIPAS2E1NXS, TLB Range Invalidate by Intermediate Physical Address, Stage 2, EL1) to C5.5.26 (TLBI RIPAS2LE1OS, TLBI RIPAS2LE1OSNXS, TLB Range Invalidate by Intermediate Physical Address, Stage 2, Last level, EL1, Outer Shareable), the Purpose text that reads:

> and the entry would be required to translate the specified IPA

is clarified to read:

> and the entry would be required to translate any IPA in the specified address range

In the following sections:

- C5.5.29 (TLBI RVAAE1, TLBI RVAAE1NXS, TLB Range Invalidate by VA, All ASID, EL1) to C5.5.37 (TLBI RVAE1OS, TLBI RVAE1OSNXS, TLB Range Invalidate by VA, EL1, Outer Shareable).
- C5.5.41 (TLBI RVAE3, TLBI RVAE3NXS, TLB Range Invalidate by VA, EL3) to C5.5.46 (TLBI RVALE1OS, TLBI RVALE1OSNXS, TLB Range Invalidate by VA, Last level, EL1, Outer Shareable).

- C5.5.50 (TLBI RVALE3, TLBI RVALE3NXS, TLB Range Invalidate by VA, Last level, EL3) to C5.5.52 (TLBI RVALE3OS, TLBI RVALE3OSNXS, TLB Range Invalidate by VA, Last level, EL3, Outer Shareable).

The Purpose text that reads:

> to translate the specified VA

is clarified to read:

> to translate any of the VAs in the specified address range

In the following sections:

- C5.5.38 (TLBI RVAE2, TLBI RVAE2NXS, TLB Range Invalidate by VA, EL2) to C5.5.40 (TLBI RVAE2OS, TLBI RVAE2OSNXS, TLB Range Invalidate by VA, EL2, Outer Shareable).
- C5.5.47 (TLBI RVALE2, TLBI RVALE2NXS, TLB Range Invalidate by VA, Last level, EL2) to C5.5.49 (TLBI RVALE2OS, TLBI RVALE2OSNXS, TLB Range Invalidate by VA, Last level, EL2, Outer Shareable).

The Purpose text that reads:

> would be used to translate the specified VA in the specified range

is clarified to read:

> would be used to translate any VA in the range

2.80 D18924

In section B2.3.10 (Restrictions on the effects of speculation), in the subsection 'Restrictions on the effects of speculation from Armv8.5', the text that reads:

> From Armv8.5, there are some further restrictions on the effects of speculation in addition to those in Armv8.0:
>
> - Data loaded under speculation with a permission or domain fault cannot be used to form an address, to generate condition codes, or to generate SVE predicate values to be used by other instructions in the speculative sequence.
> - Any System register read under speculation to a register that is not architecturally accessible from the current Exception level cannot be used to form an address, to generate condition codes, or to generate SVE predicate values to be used by other instructions in the speculative sequence.
>
> Note:
>
> As the effects of speculation are not architecturally visible, this restriction requires that the effect of any speculation cannot give rise to side channels that will leak the values held in

memory locations, System registers, or Special-purpose registers to a level of privilege that would otherwise not be able to determine those values.

- Code running in one hardware-defined context cannot exploitatively control speculative execution of code in a different hardware-defined context as a result of the behavior of any execution prediction resources that predict address or register values. In the case of 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.
  - When in AArch64 state, the current SCXTNUM_ELx value if SCXTNUM_ELx is implemented and the hardware identifies that SCXTNUM_ELx is part of the context. Where SCXTNUM_ELx is not included as part of the hardware-indicated context, an implementation can further identify that branch targets trained for branches situated at one address can control speculative execution of branches situated at different addresses only in a hard-to-determine way.

  Note:
  - The definition of “hard-to-determine manner” is left open to implementations. Examples could include the complete separation of prediction resources, or the isolation of the predictions using a cryptographic or pseudo-random mechanism to separate each context.
  - The architecture does not require that prediction resources that simply predict the direction of a branch are separated in this way.
- Changes to System registers must not occur speculatively in a way that can affect a speculative memory access that can cause a change to the micro-architectural state.
- Changes to Special-purpose registers can occur speculatively.

is changed to read:

Further restrictions on speculation are introduced by some additional architectural features as described here.

FEAT_CSV3 introduces these restrictions:

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

codes, or to generate SVE predicate values to be used by other instructions in the speculative sequence.

Note:

As the effects of speculation are not architecturally visible, this restriction requires that the effect of any speculation cannot give rise to side channels that will leak the values of memory locations, System registers, or Special-purpose registers to a level of privilege that would otherwise not be able to determine those values.

- Changes to System registers must not occur speculatively in a way that can affect a speculative memory access that can cause a change to the micro-architectural state.

  Note:

  Changes to Special-purpose registers can occur speculatively.

FEAT_CSV2, FEAT_CSV2_1p1, FEAT_CSV2_1p2, FEAT_CSV2_2 and FEAT_CSV2_3 introduce a range of additional restrictions.

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, 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.
  - Any other prediction mechanisms, other than Branch, Data Value, or Cache Prefetch predictions.

  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.

If either FEAT_CSV2_1p1 or FEAT_CVS2_3 is implemented, code running in one hardware-defined context (context1) cannot either exploitatively control, or predictively leak to, the speculative execution of code in a different hardware-defined context (context2) as a result of the behavior of branch target prediction based on the branch history used in context1.

If FEAT_CSV2_1p1 is implemented, branch or data values trained from one instruction address can exploitatively control, or predictively leak to, the speculative execution of code from a different address only in a hard-to-determine way.

If FEAT_CSV2_1p2 is implemented, the SCXTNUM_ELx register is implemented, but is not part of the hardware-defined context.

Within the same section, a new subsection 'Restrictions on predictive leakage' is added:

The execution of some code (code1) can predictively leak to some other code (code2) if and only if all of the following apply:

- The execution of code 1 influences, in a manner that is not hard-to-determine, the predictive micro-architectural structures of the implementation to behave in a way that is indicative of some architectural state accessible to the execution context of code1.
- The predictive micro-architectural structures of the implementation impact the timing of the speculative execution of code2 in a way that enables code2 to recover the architecture state in manner that is not hard-to-determine.

Note:

Mechanisms to prevent the influence and the state recovery being "not hard-to-determine" are left open to implementations. Examples could include the complete separation of prediction resources, or the isolation of the predictions using a cryptographic or pseudo-random mechanism to separate each context.

Also in the same section, the subsection 'Restrictions on exploitative control of speculative execution' is updated to read:

The execution of some code (code1) can exploitatively control speculative execution of some other code (code2) if and only if all of the following apply:

- The actions of code1 can influence, in a manner that is not hard-to-determine, the speculative execution of code2 to cause an irreversible change to the microarchitectural state of the PE that is indicative of some architectural state accessible to the execution context of code2.
- Code1 has control in determining the choice of the architecture state that the irreversible change to the microarchitectural state is indicative of.
- The irreversible changes to the microarchitectural state of the PE can be measured by code executing in an execution context other than that of code2 to allow the retrieval of the architectural state in a manner that is not hard-to-determine.

Note:

> Mechanisms to prevent the influence and the state recovery being “not hard to determine” are left open to implementations. Examples could include the complete separation of prediction resources, or the isolation of the predictions using a cryptographic or pseudo-random mechanism to separate each context.

In section D13.2.67 (ID_AA64PFR0_EL1, AArch64 Processor Feature Register 0) the value definitions in CSV2, bits [59:56] are changed to read:

> `0b0000` The implementation does not disclose whether FEAT_CSV2 is implemented.
>
> `0b0001` FEAT_CSV2 is implemented, but FEAT_CSV2_2 and FEAT_CSV2_3 are not implemented. CVS2_frac determines whether FEAT_CSV2_1p1 and/or FEAT_CSV2_1p2 are not implemented.
>
> `0b0010` FEAT_CSV2_2 is implemented but FEAT_CSV2_3 is not implemented.
>
> `0b0011` FEAT_CSV2_3 is implemented.

The equivalent changes are made in sections D13.2.87 (ID_PFR0_EL1, AArch32 Processor Feature Register 0) and G8.2.98 (ID_PFR0, Processor Feature Register 0).

In section D13.2.68 (ID_AA64PFR1_EL1, AArch64 Processor Feature Register 1), the value definitions in CSV2_frac, bits [35:32] are changed to read:

> `0b0000` Either ID_AA64PFR0_EL1.CSV2 is not 0001, or the implementation does not disclose whether FEAT_CSV2_1p1 is implemented. FEAT_CSV2_1p2 is not implemented.
>
> `0b0001` FEAT_CSV2_1p1 is implemented but FEAT_CSV2_1p2 is not implemented.
>
> `0b0010` FEAT_CSV2_1p2 is implemented.

And the following text is added after the value definitions:

> The values `0b0001` and `0b0010` are only permitted when ID_AA64PFR0_EL1.CSV2 is `0b0001`.

In section A2.2.1 (Additional functionality added to Armv8.0 in later releases), the ‘FEAT_CSV2 and FEAT_CSV2_2, Cache Speculation Variant 2’ definition is renamed ‘FEAT_CSV2, FEAT_CSV2_2, and FEAT_CSV2_3, Cache Speculation Variant 2’, and the following text is added:

> FEAT_CSV2_3 adds a mechanism to identify if hardware cannot disclose information about whether branch targets and branch history trained in one hardware described context can control speculative execution in a different hardware described context.

Also within the definition, the text that reads:

> FEAT_CSV2_2 is supported in AArch64 state only.
>
> FEAT_CSV2_2 is OPTIONAL in Armv8.0 implementations.
>
> The ID_AA64PFR0_EL1.CSV2 field identifies the presence of FEAT_CSV2_2.

is updated to read:

> FEAT_CSV2_2 and FEAT_CSV2_3 are supported in AArch64 state only.
>
> FEAT_CSV2_2 and FEAT_CSV2_3 are OPTIONAL in Armv8.0 implementations.
>
> The ID_AA64PFR0_EL1.CSV2 field identifies the presence of FEAT_CSV2_2 and FEAT_CSV2_3.

2.81 D18926

In section C7.2.54 (FCCMP), under 'Operational information', the statement that reads:

> An unordered comparison sets the

is corrected to read:

> An unordered comparison sets the PSTATE condition flags to N=0, Z=0, C=1, and V=1.

Equivalent changes are made in the following sections:

- C7.2.55 (FCCMPE).
- C7.2.66 (FCMP).
- C7.2.67 (FCMPE).
- F6.1.54 (VCMP).
- F6.1.55 (VCMPE).

2.82 D18928

In section A3.1 (Armv9-A architecture extensions), in the feature introduction of FEAT_ETEv1p2, the text that reads:

> This feature requires FEAT_RME and FEAT_ETEv1p1.

is corrected to read:

> This feature requires FEAT_ETEv1p1.
>
> This feature is required if FEAT_RME is implemented and FEAT_ETE is implemented.

2.83 D18933

In section J1.3.1 (shared/debug), in the function ExternalNoninvasiveDebugAllowed(), the code that reads as:

```
boolean ExternalNoninvasiveDebugAllowed()
     ...
    ss = SecurityStateAtEL(PSTATE.EL);
    if (ELUsingAArch32(EL1) && PSTATE.EL == EL0 &&
        ss == SS_Secure && SDER.SUNIDEN == '1') then
       return TRUE;
```

is updated to read as:

```
boolean ExternalNoninvasiveDebugAllowed()
    return ExternalNoninvasiveDebugAllowed(PSTATE.EL);

boolean ExternalNoninvasiveDebugAllowed(bits(2) el)
    ...
    ss = SecurityStateAtEL(el);
     if ((ELUsingAArch32(EL3) || ELUsingAArch32(EL1)) && el == EL0 &&
         ss == SS_Secure && SDER.SUNIDEN == '1') then
        return TRUE;
```

In section J1.3.4 (shared/trace), in the function TraceAllowed(), the code that reads as:

```
boolean TraceAllowed(bits(2) el)
    if !HaveTraceExt() then return FALSE;
    ss = SecurityStateAtEL(el);
    if SelfHostedTraceEnabled() then
        ...
    else
        case ss of
            when SS_NonSecure return ExternalNoninvasiveDebugEnabled();
            when SS_Secure return ExternalSecureNoninvasiveDebugEnabled();
            when SS_Realm return ExternalRealmNoninvasiveDebugEnabled();
            when SS_Root return ExternalRootNoninvasiveDebugEnabled();
```

is updated to read as:

```
boolean TraceAllowed(bits(2) el)
    if !HaveTraceExt() then return FALSE;
    if SelfHostedTraceEnabled() then
        ss = SecurityStateAtEL(el);
        ...
    else
         return ExternalNoninvasiveDebugAllowed(el);
```

2.84 D18939

In section C3.8.12 (Complex integer arithmetic), in the subsection 'Uniform complex integer arithmetic', the line that reads:

> Two CMLA instructions can be used as follows: CMLA Zda.S, Zn.S, Zm.S, #ACMLA Zda.S, Zn.S, Zm.S, #B

is corrected to read:

> Two CMLA instructions can be used as follows:
>
> CMLA Zda.S, Zn.S, Zm.S, #A CMLA Zda.S, Zn.S, Zm.S, #B

2.85 D18948

In section D5.2.6 (Overview of the VMSAv8-64 address translation stages), the following footnotes in the tables are changed:

Under 'Table D5-13 'TCR_ELx.TnSZ values and IA ranges, 4KB granule with no concatenation of tables', the footnotes that read:

> 1. If FEAT_TTST is not implemented, or while the PE is executing in AArch32 state, TnSZmax is 39.
> 2. Only available if FEAT_TTST is implemented, while the PE is executing in AArch64 state.

are corrected to read:

> 1. If FEAT_TTST is not implemented, TnSZmax is 39.
> 2. Only available if FEAT_TTST is implemented.

Under Table D5-14 'VTCR_EL2.T0SZ values and IA ranges, 4KB granule with possible concatenation of translation tables', the footnotes that read:

> 1. If FEAT_TTST is not implemented or while the PE is executing in AArch32 state, the maximum value of T0SZ is 39 with corresponding IA[29:12]-IA[24:12].
> 2. If FEAT_TTST is implemented, while the PE is executing in AArch64 state, and is using 4KB granules, an initial lookup level 3, (VTCR_EL2.SL0 == 3) is possible.

are corrected to read:

> 1. If FEAT_TTST is not implemented, the maximum value of T0SZ is 39 with corresponding IA[29:12]-IA[24:12].
> 2. If FEAT_TTST is implemented, and stage 2 is using 4KB granules, an initial lookup level 3, (VTCR_EL2.SL0 == 3) is possible.

Under Table D5-16 'VTCR_EL2.T0SZ values and IA ranges, 16KB granule with possible concatenation of translation tables', the footnote that reads:

1. If FEAT_TTST is not implemented or while the PE is executing in AArch32 state, the maximum value of T0SZ is 39 with corresponding IA[24:14].

is corrected to read:

1. If FEAT_TTST is not implemented, the maximum value of T0SZ is 39 with corresponding IA[24:14].

Under Table D5-17 'TCR_ELx.TnSZ values and IA ranges, 64KB granule with no concatenation of tables', the footnote that reads:

1. If FEAT_TTST is not implemented or while the PE is executing in AArch32 state, the maximum value of TnSZ is 39 with IA[24:16].

is corrected to read:

1. If FEAT_TTST is not implemented, the maximum value of TnSZ is 39 with IA[24:16].

Under Table D5-18 'VTCR_EL2.T0SZ values and IA ranges, 64KB granule with possible concatenation of translation tables', the footnote that reads:

1. If FEAT_TTST is not implemented or while the PE is executing in AArch32 state, the maximum T0SZ value is 39, with IA[24:16].

is corrected to read:

1. If FEAT_TTST is not implemented, the maximum T0SZ value is 39, with IA[24:16].

2.86 D18951

In sections C6.2.276 (SETP, SETM, SETE) to C6.2.279 (SETPTN, SETMTN, SETETN), the code that reads as:

```
if setsize<63> == '1' then setsize = 0x007FFFFFFFFFFF0<63:0>;
```

Is updated to read:

```
if setsize<63> == '1' then setsize = 0x7FFFFFFFFFFFFFF<63:0>;
```

2.87 D18981

In section D6.2.1 (Cache activity and Allocation Tags), the following text is added, following the list beginning 'When Allocation Tags are evicted from a cache entry at cache level':

> If an implementation can overwrite Allocation Tags in memory that have been written by another observer, where the Allocation Tags have not been written by an observer in the Shareability domain of that memory location, then:
>
> - A cache maintenance operation which cleans data from a cache level must also clean the associated Allocation Tags.
> - A cache maintenance operation which invalidates, or cleans and invalidates data from a cache level, must also clean and invalidate the associated Allocation Tags.

The same text is added to section D4.4.8 (Cache maintenance instructions), in the subsection 'General requirements for the scope of maintenance instructions'.

2.88 D18985

In section C6.2.396 (WFE), the operational pseudocode that reads:

```
Hint_WFE(1, WFxType_WFE);
```

is corrected to read as:

```
integer localtimeout = 1 << 64;  // No local timeout event is generated
Hint_WFE(localtimeout, WFxType_WFE);
```

An equivalent change is made in section C6.2.398 (WFI).

2.89 D18986

In section J1.1.5 (aarch64/translation), the pseudocode for AArch64.CheckWatchpoint() that reads:

```
if match && acctype == AccType_ATOMICRW then
    iswrite = !match_on_read;
```

is updated to read as :

```
if match && IsAtomicRW(acctype) then
    iswrite = !match_on_read;
```

In section J1.1.1 (aarch64/debug), the pseudocode for AArch64.WatchpointMatch() that reads:

```
if acctype == AccType_ATOMICRW then
    ls_match = (DBGWCR_EL1[n].LSC != '00');
else
    ls_match = (DBGWCR_EL1[n].LSC<(if iswrite then 1 else 0)> == '1');
```

is updated to read as:

```
if IsAtomicRW(acctype) then
    ls_match = (DBGWCR_(n,LSC) != '00');
elsif iswrite then
    ls_match = (DBGWCR_(n,LSC)<1> != '0');
else
    ls_match = (DBGWCR_(n,LSC)<0> != '0');
```

In section J1.1.2 (aarch64/exceptions), the pseudocode for AArch64.TagCheckFault() that reads:

```
readwrite = acctype IN {AccType_ATOMICRW,
AccType_ORDEREDATOMICRW,
AccType_ORDEREDRW};
if !iswrite || readwrite then
    AArch64.RaiseTagCheckFault(vaddress, iswrite);
else
    AArch64.ReportTagCheckFault(PSTATE.EL, vaddress<55>);
```

is updated to read as:

```
if !iswrite || IsAtomicRW(acctype) then
    AArch64.RaiseTagCheckFault(vaddress, iswrite);
else
    AArch64.ReportTagCheckFault(PSTATE.EL, vaddress<55>);
```

2.90 D18990

In section C6.2.65 (CMPP), the CMPP instruction's dependency on FEAT_MTE is added. In sections C6.2.264 (SB) and F5.1.175 (SB), the SB instructions' dependency on FEAT_SB is added.

2.91 D18992

In section C5.2.8 (FPCR, Floating-point Control Register), the descriptions of the floating-point exception trap enables IDE, IXE, UFE, OFE, DZE, and IOE which read:

> The value of this bit controls both scalar and Advanced SIMD floating-point arithmetic.

are corrected to read:

> The value of this bit controls both scalar and vector floating-point arithmetic.

2.92 C18996

In the following sections:

- F6.1.96 (VHADD).
- F6.1.198 (VRHADD).
- C7.2.253 (SHADD).
- C7.2.310 (SRHADD).
- C7.2.358 (UHADD).
- C7.2.383 (URHADD).
- C8.2.464 (SHADD).
- C8.2.566 (SRHADD).
- C8.2.668 (UHADD).
- C8.2.731 (URHADD).

Operational pseudocode of the form:

```
integer sum = op1 + op2;
Elem[result,e,esize] = sum<esize:1>;
```

is changed to:

```
integer sum = (op1 + op2) >> 1;
Elem[result,e,esize] = sum<esize-1:0>;
```

2.93 C19002

In section J1.3.3 (shared/functions), the code in the function InsertIESBBeforeException() that reads:

```
// InsertIESBBeforeException()
// ===========================
// If SCTLR_ELx.IESB is 1 when an exception is generated to ELx, any pending
 Unrecoverable
// SError interrupt must be taken before executing any instructions in the exception
 handler.
// However, this can be before the branch to the exception handler is made.
boolean InsertIESBBeforeException(bits(2) el);
```

is updated to read:

```
// InsertIESBBeforeException()
// ===========================
```

```
// Returns an implementation defined choice whether to insert an implicit error
 synchronization
// barrier before exception.
// If SCTLR_ELx.IESB is 1 when an exception is generated to ELx, any pending
 Unrecoverable
// SError interrupt must be taken before executing any instructions in the exception
 handler.
// However, this can be before the branch to the exception handler is made.

boolean InsertIESBBeforeException(bits(2) el)
    return (HaveIESB() && boolean IMPLEMENTATION_DEFINED "Has Implicit Error
 Synchronization Barrier before Exception");
```

2.94 D19012

In section D13.2.37 (ESR_EL1, Exception Syndrome Register (EL1)), in the section 'ISS encoding for an exception from a Watchpoint exception', the text in the field CM, bit[8]:

> Indicates whether the Watchpoint exception came from a cache maintenance or address translation instruction:

is corrected to read:

> Indicates whether the Watchpoint exception came from a cache maintenance instruction:

and the text that reads:

> The Watchpoint exception was generated by either the execution of a cache maintenance instruction or by a synchronous Watchpoint exception on the execution of an address translation instruction.

is corrected to read:

> The Watchpoint exception was generated by the execution of a cache maintenance instruction.

In the field WnR, bit[6], the text that reads:

> For Watchpoint exceptions on cache maintenance and address translation instructions, this bit always returns a value of 1.

is corrected to read:

> For Watchpoint exceptions on cache maintenance instructions, this bit always returns a value of 1.

The equivalent changes are made in section D13.2.38 (ESR_EL2, Exception Syndrome Register (EL2)).

2.95 C19017

In section D13.9.16 (VSESR_EL2, Virtual SError Exception Syndrome Register), in the 'AET' field description, the following text:

> When a virtual SError interrupt is taken to EL1 using AArch32, DFSR[15:4] is set to VSESR_EL2.AET.
>
> When a virtual SError interrupt is deferred by an ESB instruction, VDISR_EL2[15:4] is set to VSESR_EL2.AET.

is clarified to read:

> When a virtual SError interrupt is taken to EL1 using AArch32, DFSR[15:14] is set to VSESR_EL2.AET.
>
> When a virtual SError interrupt is deferred by an ESB instruction, VDISR_EL2[15:14] is set to VSESR_EL2.AET.

The equivalent change is made in section G8.6.19 (VDFSR, Virtual SError Exception Syndrome Register).

2.96 D19026

In section C5.2.18 (SPSR_EL1, Saved Program Status Register (EL1)), in the field description for 'When exception taken from AArch64 state:', M[3:0], bits [3:0], the bullet that reads:

> - 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 10.

is corrected to read:

> - 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`.

2.97 C19027

In section D8.11.3 (Common event numbers), subsection 'Common microarchitectural events', the following text is reintroduced into the descriptions of MEM_ACCESS_CHECKED (`0x4024`), MEM_ACCESS_CHECKED_RD (`0x4025`), and MEM_ACCESS_CHECKED_WR (`0x4026`):

> It is **IMPLEMENTATION DEFINED** whether the counter increments on a Tag Checked access made when Tag Check Faults are configured to be ignored by SCTLR_ELx.TCF or SCTLR_ELx.TCF0.

2.98 D19036

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

```
AArch64.DataMemZero(...)
    iswrite = TRUE;
    AddressDescriptor memaddrdesc = memaddrdesc_in;
    for i = 0 to size-1
        accdesc = CreateAccessDescriptor(AccType_DCZVA);
        if HaveMTEExt() then
            if AArch64.AccessIsTagChecked(vaddress, AccType_DCZVA) then
                ...
```

is updated to read:

```
AArch64.DataMemZero(...)
    iswrite = TRUE;
    AddressDescriptor memaddrdesc = memaddrdesc_in;
    accdesc = CreateAccessDescriptor(AccType_DCZVA);

    if HaveTME() then


        accdesc.transactional = TSTATE.depth > 0;


        if accdesc.transactional && !MemHasTransactionalAccess(memaddrdesc.memattrs)
 then

            FailTransaction(TMFailure_IMP, FALSE);

    for i = 0 to size-1
        if HaveMTE2Ext() then
            if AArch64.AccessIsTagChecked(vaddress, AccType_DCZVA) then
                ...
```

2.99 D19037

In section B2.3.2 (Dependency definitions), the definition of 'Pick Basic dependency' that reads:

> A Pick Basic dependency from a read Register effect or read Memory effect R1 to a Register effect or Memory effect E2 exists if and only if all of the following apply:
>
> - There is a Dependency through registers from R1 to a Register effect E3, and one of the following applies:
>   - There is an Intrinsic control dependency from the Register effect E3 to a Register effect E4, and there is a chain of Dependency through registers or Dependency through memory from E4 to E2.
>   - There is an Intrinsic control dependency from the Register effect E3 to E2.

is changed to read:

> A Pick Basic dependency from a read Register effect or read Memory effect R1 to a Register effect or Memory effect E2 exists if and only if one of the following apply:

- There is a Dependency through registers from R1 to E2.
- There is an Intrinsic Control dependency from R1 to E2.
- There is a Pick Basic dependency from R1 to an Effect E3 and there is a Pick Basic dependency from E3 to E2.

Equivalent changes are made in section E2.3.2 (Dependency definitions).

2.100 C19047

In section D13.2.27 (CLIDR_EL1, Cache Level ID Register), the following Note is added to the descriptions of LoUU, bits [29:27], and LoUIS, bits [23:21]:

> Note: This field does not describe the requirements for instruction cache invalidation. See CTR_EL0.DIC.

The equivalent changes are made in section G8.2.27 (CLIDR, Cache Level ID Register).

2.101 D19049

In section K1.2.11 (Instruction fetches from Device memory), the text that reads:

> If a branch causes the program counter to point to an area of memory with the Device attribute that is not marked as execute-never for the current Exception level for instruction fetches, then an implementation must perform one of the following behaviors:
>
> - It treats the instruction fetch as if it were to a memory location with the Normal, Non-cacheable attribute.
> - It generates a Permission fault.

is corrected to read:

> For an instruction fetch from a memory location with the Device attribute that is not marked as execute-never for the current Exception level, an implementation can either:
>
> - Treat the instruction fetch as if it were to a memory location with the Normal Non-cacheable attribute.
> - Take a Permission fault.

2.102 C19051

In section C3.2.12 (Atomic instructions), in the subsection 'Single-copy atomic 64-byte load/store', the text that reads:

> When the instructions access a memory type for an enabled translation stage that is not one of the following, a data abort for that translation stage occurs:
>
> - Normal Inner Non-cacheable, Outer Non-cacheable.
> - Device-GRE.
> - Device-nGRE.
> - Device-nGnRE.
> - Device-nGnRnE.

is corrected to read:

> When the instructions access a memory type that is not one of the following, a Data abort for Unsupported Exclusive or Atomic access is generated for the stage of translation that provided the memory type:
>
> - Normal Inner Non-cacheable, Outer Non-cacheable.
> - Device-GRE.
> - Device-nGRE.
> - Device-nGnRE.
> - Device-nGnRnE.

And the following statements are added after this text:

> Regardless of the memory type:
>
> - The memory access generated by an ST64BV or ST64BV0 instruction is not merged with any accesses.
> - The memory access generated by an ST64B instruction is not merged with any accesses generated by store instructions appearing in program order after the instruction.

2.103 D19068

In section D5.4.2 (About PSTATE.PAN), the following text is removed:

> When HCR_EL2.{E2H, TGE} == {1, 1} SCTLR_EL1.SPAN and SCTLR_EL2.SPAN are ignored.

Similarly, in section D13.2.118 (SCTLR_EL1, System Control Register (EL1)), in the field SPAN, bit [23], the following text is removed:

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

2.104 D19077

The following feature dependencies are added:

- In section C6.2.100 (CRC32B, CRC32H, CRC32W, CRC32X), the dependency of CRC32B, CRC32H, CRC32W, CRC32X on FEAT_CRC32 is added.
- In section C6.2.101 (CRC32CB, CRC32CH, CRC32CW, CRC32CX), the dependency of CRC32CB, CRC32CH, CRC32CW, CRC32CX on FEAT_CRC32 is added.
- In section C7.2.7 (AESD), the dependency of AESD on FEAT_AES is added.
- In section C7.2.8 (AESE), the dependency of AESE on FEAT_AES is added.
- In section C7.2.215 (PMULL, PMULL2), the dependency of PMULL, PMULL2 on FEAT_PMULL is added.
- In section C7.2.239 (SHA1C), the dependency of SHA1C on FEAT_SHA1 is added.
- In section C7.2.240 (SHA1H), the dependency of SHA1H on FEAT_SHA1 is added.
- In section C7.2.241 (SHA1M), the dependency of SHA1M on FEAT_SHA1 is added.
- In section C7.2.242 (SHA1P), the dependency of SHA1P on FEAT_SHA1 is added.
- In section C7.2.243 (SHA1SU0), the dependency of SHA1SU0 on FEAT_SHA1 is added.
- In section C7.2.244 (SHA1SU1), the dependency of SHA1SU1 on FEAT_SHA1 is added.
- In section C7.2.245 (SHA256H2), the dependency of SHA256H2 on FEAT_SHA256 is added.
- In section C7.2.246 (SHA256H), the dependency of SHA256H on FEAT_SHA256 is added.
- In section C7.2.247 (SHA256SU0), the dependency of SHA256SU0 on FEAT_SHA256 is added.
- In section C7.2.248 (SHA256SU1), the dependency of SHA256SU1 on FEAT_SHA256 is added.

Equivalent changes are made to the AArch32 variants of these instructions that exist.

2.105 D19093

In section D13.2.117 (SCR_EL3, Secure Configuration Register), for the field SIF, bit[9], the text that reads:

> Secure instruction fetch. When the PE is in Secure state, this bit disables instruction fetch from Non-secure memory.
>
> `0b0` Secure state instruction fetches from Non-secure memory are permitted.

> `0b1` Secure state instruction fetches from Non-secure memory are not permitted.

is corrected to read:

> Secure instruction fetch. When the PE is in Secure state, this bit disables instruction execution from memory marked in the first stage of translation as being Non-secure.
>
> `0b0` Secure state instruction execution from memory marked in the first stage of translation as being Non-secure is permitted.
>
> `0b1` Secure state instruction execution from memory marked in the first stage of translation as being Non-secure is not permitted.

The same correction is made in section G8.2.125 (SCR, Secure Configuration Register).

2.106 C19099

In section D13.7.7 (PMSEVFR_EL1, Sampling Event Filter Register), the text under 'Purpose' that reads:

> Controls sample filtering by events. The overall filter is the logical AND of these filters. For example, if E[3] and E[5] are both set to 1, only samples that have both event 3 (Level 1 unified or data cache refill) and event 5 set (TLB walk) are recorded.

is clarified to read:

> Controls sample filtering by events. The overall filter is the logical AND of these filters. For example, if PMSEVFR_EL1.E[3] and PMSEVFR_EL1.E[5] are both set to 1, only samples that have both event 3 (Level 1 unified or data cache refill) and event 5 (TLB walk) set to 1 are recorded.

Similarly, in section D13.7.13 (PMSNEVFR_EL1, Sampling Inverted Event Filter Register), the text that reads:

> Controls sample filtering by events. The overall filter is the logical AND of these filters. For example, if E[3] and E[5] are both set to `0b1`, only samples that have both event 3 (Level 1 unified or data cache refill) and event 5 (TLB walk) clear are recorded.

is clarified to read:

> Controls sample filtering by events. The overall inverted filter is the logical OR of these filters. For example, if PMSNEVFR_EL1.E[3] and PMSNEVFR_EL1.E[5] are both set to 1, samples that have either event 3 (Level 1 unified or data cache refill) or event 5 (TLB walk) set to 1 are not recorded.

2.107 D19121

In section D13.2.118 (SCTLR_EL1, System Control Register (EL1)), in field 'C, bit [2]', the text that reads:

> When the value of the HCR_EL2.DC bit is 1, the PE ignores SCTLR.C. This means that Non-secure EL0 and Non-secure EL1 data accesses to Normal memory are Cacheable.
>
> When FEAT_VHE is implemented, and the value of HCR_EL2.{E2H, TGE} is {1, 1}, this bit has no effect on the PE.

is changed to read:

> When the Effective value of the HCR_EL2.DC bit in the current Security state is 1, the PE ignores SCTLR_EL1.C. This means that EL0 and EL1 data accesses to Normal memory are Cacheable.
>
> When FEAT_VHE is implemented, and the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}, this bit has no effect on the PE.

Similarly in field 'M, bit [0]', the text that reads:

> If the value of HCR_EL2.{DC, TGE} is not {0, 0} then in Non-secure state the PE behaves as if the value of the SCTLR_EL1.M field is 0 for all purposes other than returning the value of a direct read of the field.
>
> When FEAT_VHE is implemented, and the value of HCR_EL2.{E2H, TGE} is {1, 1}, this bit has no effect on the PE.

is changed to read:

> If the Effective value of HCR_EL2.{DC, TGE} in the current Security state is not {0, 0} then the PE behaves as if the value of the SCTLR_EL1.M field is 0 for all purposes other than returning the value of a direct read of the field.
>
> When FEAT_VHE is implemented, and the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}, this bit has no effect on the PE.

The equivalent changes are made in section G8.2.126 (SCTLR, System Control Register).

2.108 C19125

In section D2.12.9 (Exception syndrome information and preferred return address), in the description of 'Exception syndrome information', the following Note is added:

> Note: If the PE cannot determine the correct value of ESR_ELx.EX for the stepped instruction, then it sets ESR_ELx.{ISV, EX} to {0, 0}. For example, the exception is taken before the PE decodes the stepped instruction, or the exception means the PE has no valid stepped instruction to decode.

Similarly, in section H3.2.8 (Syndrome information on Halting Step), the following Note is added:

> Note: If the PE cannot determine whether the stepped instruction was a Load-Exclusive instruction or not, then it sets EDSCR.STATUS to Halting Step, no syndrome. For example, the exception is taken before the PE decodes the stepped instruction, or the exception means the PE has no valid stepped instruction to decode.

2.109 D19129

In section D13.2.55 (HPFAR_EL2, Hypervisor IPA Fault Address Register), in the FIPA encoding for when FEAT_LPA is implemented, the text:

> FIPA, bits [38:0]

is corrected to:

> FIPA, bits [39:0]

and in the FIPA encoding for when FEAT_LPA is not implemented, the text:

> Bits [38:35]
>
> Reserved, **RES0**.
>
> FIPA, bits [34:0]

is corrected to:

> Bits [39:36]
>
> Reserved, **RES0**.
>
> FIPA, bits [35:0]

The encoding diagrams are updated to match.

2.110 C19147

In section D8.11.1 (Definitions), in the subsection, 'Levels of caches and TLBs', the following text is added:

> - If L3D_CACHE events are implemented, then L2D_CACHE and if applicable L2I_CACHE events should be implemented.
> - If L2D_CACHE events are implemented, then L1D_CACHE and if applicable L1I_CACHE events should be implemented.
> - If L2I_CACHE events are implemented, then L1I_CACHE events should be implemented.

- Where the Last Level of cache is also the Level 3 or Level 2 unified cache, the LL_CACHE events should be implemented in preference to the L3D_CACHE or L2D_CACHE events as applicable.
- For <n> = 1 and <n> = 2:
  - If the Level <n> cache is unified, but the cache can disambiguate between Data and Instruction accesses to the cache, then both the L<n>D_CACHE and L<n>I_CACHE events should be implemented.
  - If the cache is unified and the cache cannot disambiguate between Data and Instruction accesses, then only the L<n>D_CACHE should be implemented, counting all accesses.

This final property is **IMPLEMENTATION DEFINED**.

In the same section, in the subsection 'Definition of terms', within the following bullet list in the definition for 'Memory-read operations':

> For levels of cache hierarchy beyond the Level 1 caches, memory-read operations also include accesses made as part of a refill of another cache closer to the PE. Such refills might be due to:

The following bullet is deleted:

- The execution of an instruction preload instruction on a unified cache.

And the bullet point that reads:

- Instruction memory accesses.

is corrected to read:

> Instruction memory accesses to a unified cache, when the cache does not implement the L<n>I_CACHE event.

2.111 D19162

In section B2.3.10 (Restrictions on the effects of speculation), in the subsection 'Restrictions on the effects of speculation from Armv8.5', the text that reads:

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

is updated to read:

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

The equivalent change is made in section E2.3.9 (Restrictions on the effects of speculation), in the subsection 'Further restrictions on the effects of speculation from Armv8.5'.

Similar changes are also made in the following sections:

- A2.2.1 (Additional functionality added to Armv8.0 in later releases), in the definition of 'FEAT_CSV3, Cache Speculation Variant 3'.
- D13.2.67 (ID_AA64PFR0_EL1, AArch64 Processor Feature Register 0), in the field 'CSV3, bits [63:60]'.
- D13.2.89 (ID_PFR2_EL1, AArch32 Processor Feature Register 2), in the field 'CSV3, bits [3:0]'.
- G8.2.100 (ID_PFR2, Processor Feature Register 2), in the field 'CSV3, bits [3:0]'.

2.112 D19166

In section D6.4.1 (Virtual address translation), the text that reads:

> If stage 1 translation at the current Exception level is disabled:
>
> - When the value of HCR_EL2.DC is 1, stage 1 translations are Tagged or Untagged depending on the value of HCR_EL2.DCT.
> - When the value of HCR_EL2.DC is 0, stage 1 translations are treated as Untagged.

is corrected to read:

> If stage 1 translation is disabled for the EL1&0 translation regime:
>
> - If the value of HCR_EL2.DC is 1, stage 1 translations are Tagged or Untagged depending on the value of HCR_EL2.DCT.
> - If the value of HCR_EL2.DC is 0, stage 1 translations are treated as Untagged.
>
> For all other translation regimes, if stage 1 translation is disabled, stage 1 translations are treated as Untagged.

2.113 D19169

In section J1.1.5 (aarch64/translation) a new function, AArch64.SettingDirtyStatePermitted(), is added:

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

In the same section, the code in AArch64.S2Translate() that reads:

```
// If HW update of dirty bit is enabled, the walk state permissions
// will already reflect a configuration permitting writes.
// The update of the descriptor occurs only if the descriptor bits in
// memory do not reflect that and the access instigates a write.
if (fault.statuscode == Fault_None &&
        ...
        !(acctype IN {AccType_AT, AccType_ATPAN, AccType_IC, AccType_DC})) then
    // Set descriptor S2AP[1] bit permitting stage 2 writes
    new_desc<7> = '1';
```

Is updated to read as:

```
// If HW update of dirty bit is enabled, the walk state permissions
// will already reflect a configuration permitting writes.
// The update of the descriptor occurs only if the descriptor bits in
// memory do not reflect that and the access instigates a write.
if (AArch64.SettingDirtyStatePermitted(fault) &&
        ...
        !(acctype IN {AccType_AT, AccType_ATPAN, AccType_IC, AccType_DC})) then
    // Set descriptor S2AP[1] bit permitting stage 2 writes
    new_desc<7> = '1'
```

An equivalent change is made in AArch64.S1Translate().

2.114 D19178

In section J1.1.3 (aarch64/functions), the function AddressSupportsLS64(), that reads as:

```
boolean AddressSupportsLS64(bits(64) address)
```

Is updated to read as:

```
boolean AddressSupportsLS64(bits(52) paddress);
```

The following changes are also made in the same section:

In MemStore64B(), the code that reads:

```
MemStore64B(bits(64) address, bits(512) value, AccType acctype)
    boolean iswrite = TRUE;
    constant integer size = 64;
    aligned = AArch64.CheckAlignment(address, size, acctype, iswrite);
    if !AddressSupportsLS64(address) then
        c = ConstrainUnpredictable(Unpredictable_LS64UNSUPPORTED);
        assert c IN {Constraint_LIMITED_ATOMICITY, Constraint_FAULT};
        if c == Constraint_FAULT then
            ...
        else
            // Accesses are not single-copy atomic above the byte level.
            for i = 0 to 63
                AArch64.MemSingle[address+8*i, 1, acctype, aligned] = value<7+8*i :
 8*i>;
    else
        -= MemStore64BWithRet(address, value, acctype);  // Return status is ignored
 by ST64B
```

```
return;
```

Is updated to read:

```
MemStore64B(bits(64) address, bits(512) value, AccType acctype)
    boolean iswrite = TRUE;
    constant integer size = 64;
    aligned = AArch64.CheckAlignment(address, size, acctype, iswrite);
    AddressDescriptor memaddrdesc = AArch64.TranslateAddress(address, acctype,
 iswrite,
                                                             istagaccess, aligned,
 size);

    // Check for aborts or debug exceptions
    if IsFault(memaddrdesc) then
        AArch64.Abort(address, memaddrdesc.fault);

    // Effect on exclusives
    if memaddrdesc.mematttrs.shareability != Shareability_NSH then
        ClearExclusiveByAddress(memaddrdesc.paddress, ProcessorID(), 64);

    // Memory array access
    accdesc = CreateAccessDescriptor(acctype);
    if !AddressSupportsLS64(memaddrdesc.paddress.address) then
        c = ConstrainUnpredictable(Unpredictable_LS64UNSUPPORTED);
        assert c IN {Constraint_LIMITED_ATOMICITY, Constraint_FAULT};

        if c == Constraint_FAULT then
            ...
        else
            // Accesses are not single-copy atomic above the byte level.
            accdesc.acctype = AccType_ATOMIC;
            for i = 0 to size-1
                memstatus = PhysMemWrite(memaddrdesc, 1, accdesc, value<8*i+7:8*i>);
                if IsFault(memstatus) then
                    HandleExternalWriteAbort(memstatus, memaddrdesc, 1, accdesc);
                memaddrdesc.paddress.address = memaddrdesc.paddress.address+1;
    else
        memstatus = PhysMemWrite(memaddrdesc, size, accdesc, value);
        if IsFault(memstatus) then
            HandleExternalWriteAbort(memstatus, memaddrdesc, size, accdesc);
    return;
```

In MemLoad64B(), the code that reads:

```
bits(512) MemLoad64B(bits(64) address, AccType acctype)
    ...

    aligned = AArch64.CheckAlignment(address, size, acctype, iswrite);

    if !AddressSupportsLS64(address) then
        c = ConstrainUnpredictable(Unpredictable_LS64UNSUPPORTED);
        assert c IN {Constraint_LIMITED_ATOMICITY, Constraint_FAULT};

        if c == Constraint_FAULT then
            // Generate a stage 1 Data Abort reported using the DFSC code of 110101.
            boolean secondstage = FALSE;
            boolean s2fs1walk = FALSE;
            FaultRecord fault = AArch64.ExclusiveFault(acctype, iswrite,
 secondstage, s2fs1walk);
            AArch64.Abort(address, fault);
        else
            // Accesses are not single-copy atomic above the byte level
            for i = 0 to 63
                data<7+8*i : 8*i> = AArch64.MemSingle[address+8*i, 1, acctype,
 aligned];
```

```
            return data;

    AddressDescriptor memaddrdesc;
    memaddrdesc = AArch64.TranslateAddress(address, acctype, iswrite, istagaccess,
 aligned, size);

    // Check for aborts or debug exceptions
    if IsFault(memaddrdesc) then
        ...
    accdesc = CreateAccessDescriptor(acctype);
    PhysMemRetStatus memstatus;
    (memstatus, data) = PhysMemRead(memaddrdesc, size, accdesc);
    if IsFault(memstatus) then
        HandleExternalReadAbort(memstatus, memaddrdesc, size, accdesc);
    return data;
```

Is updated to read as:

```
bits(512) MemLoad64B(bits(64) address, AccType acctype)
    ...

    aligned = AArch64.CheckAlignment(address, size, acctype, iswrite);

    AddressDescriptor memaddrdesc;
    memaddrdesc = AArch64.TranslateAddress(address, acctype, iswrite, istagaccess,
 aligned, size);

    // Check for aborts or debug exceptions
    if IsFault(memaddrdesc) then
       ...
    accdesc = CreateAccessDescriptor(acctype);
    if !AddressSupportsLS64(memaddrdesc.paddress.address) then
        c = ConstrainUnpredictable(Unpredictable_LS64UNSUPPORTED);
        assert c IN {Constraint_LIMITED_ATOMICITY, Constraint_FAULT};

        if c == Constraint_FAULT then
            // Generate a stage 1 Data Abort reported using the DFSC code of 110101.
            boolean secondstage = FALSE;
            boolean s2fs1walk = FALSE;
            FaultRecord fault = AArch64.ExclusiveFault(acctype, iswrite,
 secondstage, s2fs1walk);
            AArch64.Abort(address, fault);
        else
            // Accesses are not single-copy atomic above the byte level.
            accdesc.acctype = AccType_ATOMIC;
            for i = 0 to size-1
                PhysMemRetStatus memstatus;
                (memstatus, data<8*i+7:8*i>) = PhysMemRead(memaddrdesc, 1, accdesc);
                if IsFault(memstatus) then
                    HandleExternalReadAbort(memstatus, memaddrdesc, 1, accdesc);
                memaddrdesc.paddress.address = memaddrdesc.paddress.address + 1;
    else
        PhysMemRetStatus memstatus;
        (memstatus, data) = PhysMemRead(memaddrdesc, size, accdesc);
        if IsFault(memstatus) then
            HandleExternalReadAbort(memstatus, memaddrdesc, size, accdesc);
    return data;
```

2.115 D19201

In section J1.1.5 (aarch64/translation), the function AArch64.MaxTxSZ() that reads:

```
integer AArch64.MaxTxSZ(TGx tgx)
   if HaveSmallTranslationTableExt() && !UsingAArch32() then
       case tgx of
           when TGx_4KB return 48;
           when TGx_16KB return 48;
           when TGx_64KB return 47;

   return 39;
```

is updated to read:

```
integer AArch64.MaxTxSZ(TGx tgx)
   if HaveSmallTranslationTableExt() then
       case tgx of
           when TGx_4KB return 48;
           when TGx_16KB return 48;
           when TGx_64KB return 47;

   return 39;
```

2.116 D19234

In section J1.1.5 (aarch64/translation), the code in the function AArch64.MemSwapTableDesc() that reads:

```
(FaultRecord, bits(64)) AArch64.MemSwapTableDesc(...)
    ...
        if IsFault(memstatus) then
            iswrite = TRUE;
            fault = HandleExternalTTWAbort(memstatus, iswrite, descupdateaddress,
 descupdateaccess, 8, fault);
            fault.acctype = memstatus.acctype;
            if IsFault(fault.statuscode) then
                fault.acctype = AccType_ATOMICRW;
                return (fault, bits(64) UNKNOWN);
```

is updated to read:

```
(FaultRecord, bits(64)) AArch64.MemSwapTableDesc(...)
    ...
        if IsFault(memstatus) then
            iswrite = TRUE;
            fault = HandleExternalTTWAbort(memstatus, iswrite, descupdateaddress,
 descupdateaccess, 8, fault);

            if IsFault(fault.statuscode) then
                fault.acctype = AccType_ATOMICRW;
                return (fault, bits(64) UNKNOWN);
```

2.117 C19236

In section D13.5.1 (PMCCFILTR_EL0, Performance Monitors Cycle Count Filter Register), the following text is removed from the SH, bit [24] description:

> If Secure EL2 is disabled, this field is **RES0**.

2.118 D19237

In section D13.2.34 (CTR_EL0, Cache Type Register), in field 'L1Ip, bits[15:14]', the text that reads:

> The value `0b01` is reserved in Armv8.

is replaced with:

> From Armv8, the value `0b01` is reserved.

In section G8.2.36 (CTR, Cache Type Register), in the description of the equivalent field 'L1Ip, bits[15:14]', the text that reads:

> The value `0b01` is not permitted in Armv8.

is replaced with:

> From Armv8, the value `0b01` is not permitted.

2.119 D19239

In section D13.2.49 (HCRX_EL2, Extended Hypervisor Configuration Register), the text in the fields MSCEN, MCE2, CMOW, and SMPME that reads:

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

is corrected to read:

> On a Warm reset:
>
> When EL3 is not implemented and EL2 is implemented, this field resets to 0.
>
> Otherwise, this field resets to an architecturally **UNKNOWN** value.

In the same register, the text in the fields VFNMI, VINMI, TALLINT, FGTnXS, FnXS, EnASR, EnALS, and EnAS0 that reads:

> On a Warm reset, when EL3 is not implemented and EL2 is implemented, this field resets to 0.

is corrected to read:

On a Warm reset:

When EL3 is not implemented and EL2 is implemented, this field resets to 0.

Otherwise, this field resets to an architecturally **UNKNOWN** value.

2.120 D19265

In section J1.3.3 (shared/functions), the function HaveEMPAMExt() is removed, and the functions HaveMPAMv0p1Ext() and HaveMPAMv1p1Ext() are added.

The code that reads:

```
// HaveEMPAMExt()
// ==============
// Returns TRUE if Enhanced MPAM is implemented, and FALSE otherwise.

boolean HaveEMPAMExt()
    return (HasArchVersion(ARMv8p6) &&
            HaveMPAMExt() &&
            boolean IMPLEMENTATION_DEFINED "Has enhanced MPAM extension");
```

Is modified to:

```
// HaveMPAMv0p1Ext()
// =================
// Returns TRUE if MPAMv0p1 is implemented, and FALSE otherwise.

boolean HaveMPAMv0p1Ext()
    return (HasArchVersion(ARMv8p6) &&
            HaveMPAMExt() &&
            boolean IMPLEMENTATION_DEFINED "Has enhanced MPAMv0p1 extension");

// HaveMPAMv1p1Ext()
// =================
// Returns TRUE if MPAMv1p1 is implemented, and FALSE otherwise.

boolean HaveMPAMv1p1Ext()
    return (HasArchVersion(ARMv8p6) &&
            HaveMPAMExt() &&
            boolean IMPLEMENTATION_DEFINED "Has enhanced MPAMv1p1 extension");
```

Also in section J1.3.3 (shared/functions), the code in the function GenMPAMcurEL() that reads:

```
MPAMinfo GenMPAMcurEL(AccType acctype)
    ...

    if HaveEMPAMExt() && security == SS_Secure then
        if MPAM3_EL3.FORCE_NS == '1' then
            pspace = PIdSpace_NonSecure;
        if MPAM3_EL3.SDEFLT == '1' then
            return DefaultMPAMinfo(pspace);
```

Is modified to read:

```
MPAMinfo GenMPAMcurEL(AccType acctype)
    ...

    if HaveMPAMv0p1Ext() && MPAMIDR_EL1.HAS_FORCE_NS == '1' then
        if MPAM3_EL3.FORCE_NS == '1' && security == SS_Secure then
            pspace = PIdSpace_NonSecure;
    if (HaveMPAMv0p1Ext() || HaveMPAMv1p1Ext()) && MPAMIDR_EL1.HAS_SDEFLT == '1'
 then
        if MPAM3_EL3.SDEFLT == '1' && security == SS_Secure then
            return DefaultMPAMinfo(pspace);
```

2.121 D19267

In section I5.8.32 (ERR<n>STATUS, Error Record <n> Primary Status Register, n = 0 - 65534), in field 'MV, bit [26]', for the value `0b1`, the text that reads:

> The **IMPLEMENTATION DEFINED** contents of the ERR<n>MISC<m> registers contains additional information for an error recorded by this record.

is corrected to read:

> The contents of the ERR<n>MISC<m> registers contain additional information for an error recorded by this record.

2.122 D19294

In section A2.7.1 (Architectural features added by Armv8.4), under 'FEAT_TRF, Self-hosted Trace Extensions', the text that reads:

> - A context synchronization instruction TSB CSYNC which can be used to prevent reordering of trace operation accesses with respect to other accesses of the same System registers.

is corrected to read:

> - A barrier instruction TSB CSYNC which can be used to prevent reordering of trace operation accesses with respect to other accesses of the same System registers.

In section B2.3.11 (Memory barriers), in the subsection 'Trace Synchronization Barrier (TSB CSYNC)', the text that reads:

> The TSB CSYNC instruction is a memory barrier instruction that preserves the relative order of memory accesses to System registers due to trace operations and other memory accesses to the same registers.

is corrected to read:

> The TSB CSYNC instruction is a barrier instruction that preserves the relative order of accesses to System registers due to trace operations and other accesses to the same registers.

The equivalent correction is made in section E2.3.10 (Memory barriers).

Also in section B2.3.11 (Memory barriers), in the subsection 'Profiling Synchronization Barrier (PSB CSYNC)', the text that reads:

> The PSB CSYNC instruction is a memory barrier that ensures that all existing profiling data for the current PE has been formatted, and profiling buffer addresses have been translated such that all writes to the profiling buffer have been initiated.

is corrected to read:

> The PSB CSYNC instruction is a barrier that ensures that all existing profiling data for the current PE has been formatted, and Profiling Buffer addresses have been translated such that all writes to the Profiling Buffer have been initiated.

2.123 R19359

In section C6.2 (Alphabetical list of A64 base instructions), in sections C6.2.68 to C6.2.99, the decode pseudocode in the CPY* instructions that reads:

```
if s == n || s == d || n == d then UNDEFINED;
if d == 31 || s == 31 || n == 31 then UNDEFINED;
```

is updated to read:

```
if s == n || s == d || n == d || d == 31 || s == 31 || n == 31 then
    c = ConstrainUnpredictable(Unpredictable_MOPSOVERLAP31);
    assert c IN {Constraint_UNDEF, Constraint_NOP};
    case c of
        when Constraint_UNDEF      UNDEFINED;
        when Constraint_NOP        EndOfInstruction();
```

Similarly, in sections C6.2.272 to C6.2.279, the decode pseudocode in the SET* instructions that reads:

```
if s == n || s == d || n == d then UNDEFINED;
if d == 31 || n == 31 then UNDEFINED;
```

is updated to read:

```
if s == n || s == d || n == d || d == 31 || n == 31 then
    c = ConstrainUnpredictable(Unpredictable_MOPSOVERLAP31);
    assert c IN {Constraint_UNDEF, Constraint_NOP};
    case c of
        when Constraint_UNDEF      UNDEFINED;
        when Constraint_NOP        EndOfInstruction();
```

2.124 R19370

In section E2.8.1 (Normal memory), after the text that reads:

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

The following text is added:

> For the purposes of this requirement, the endpoint for a location in Conventional memory is the PoC.

2.125 D19372

In section D13.2.107 (RGSR_EL1, Random Seed Allocation Tag Seed Register), the following text is added under 'Configurations':

> When GCR_EL1.RRND==0b0, direct and indirect reads and writes to the register appear to occur in program order relative to other instructions, without the need for any explicit synchronization.

2.126 D19378

In section J1.3.3 (shared/functions), the function HaveFeatWFxT2() is removed.

In section J1.1.2 (aarch64/exceptions), the function AArch64.WFxTrap() that reads:

```
AArch64.WFxTrap(WFxType wfxtype, bits(2) target_el)
    assert UInt(target_el) > UInt(PSTATE.EL);
    ...
        when WFxType_WFIT
            exception.syndrome<1:0> = '10';
            if HaveFeatWFxT2() then
                exception.syndrome<2>   = '1';   // Register field is valid
                exception.syndrome<9:5> = ThisInstr()<4:0>;
            else
                exception.syndrome<2> = '0';     // Register field is invalid
        when WFxType_WFET
            exception.syndrome<1:0> = '11';
            if HaveFeatWFxT2() then
                exception.syndrome<2>   = '1';   // Register field is valid
                exception.syndrome<9:5> = ThisInstr()<4:0>;
            else
                exception.syndrome<2> = '0';     // Register field is invalid

    ...
```

Is modified to read:

```
AArch64.WFxTrap(WFxType wfxtype, bits(2) target_el)
    assert UInt(target_el) > UInt(PSTATE.EL);
    ...
        when WFxType_WFIT
            exception.syndrome<1:0> = '10';
            exception.syndrome<2>   = '1';   // Register field is valid
            exception.syndrome<9:5> = ThisInstr()<4:0>;
        when WFxType_WFET
            exception.syndrome<1:0> = '11';
            exception.syndrome<2>   = '1';   // Register field is valid
            exception.syndrome<9:5> = ThisInstr()<4:0>;

     ...
```

2.127 D19410

In section J1.1.2 (aarch64/exceptions), in the function AArch64.TagCheckFault(), the code that reads as:

```
AArch64.TagCheckFault(bits(64) vaddress, AccType acctype, boolean iswrite)
    el = AArch64.AccessUsesEL(acctype);
    ...
        when '11'       // Tag Check Faults cause a synchronous exception on reads
 or on
                        // a read/write access, and are asynchronously accumulated
 on writes
            // Check for access performing both a read and a write.
            if !iswrite || IsAtomicRW(acctype) then
                AArch64.RaiseTagCheckFault(vaddress, iswrite);
            else
                AArch64.ReportTagCheckFault(PSTATE.EL, vaddress<55>);
```

is updated to read as:

```
AArch64.TagCheckFault(bits(64) vaddress, AccType acctype, boolean iswrite)
    el = AArch64.AccessUsesEL(acctype);
    ...
        when '11'       // Tag Check Faults cause a synchronous exception on reads
 or on
                        // a read/write access, and are asynchronously accumulated
 on writes
            // Check for access performing both a read and a write.
            if !iswrite || IsAtomicRW(acctype) then
                AArch64.RaiseTagCheckFault(vaddress, iswrite);
            else
                AArch64.ReportTagCheckFault(el, vaddress<55>);
```

2.128 D19420

In section J1.3.4 (shared/trace), the code in the function GetTimestamp() that reads:

```
        when TimeStamp_OffsetPhysical
```

```
            return PhysicalCountInt() - CNTPOFF_EL2;
```

is updated to read:

```
        when TimeStamp_OffsetPhysical
            bits(64) physoff = if PhysicalOffsetIsValid() then CNTPOFF_EL2 else
 Zeros();
            return PhysicalCountInt() - physoff;
```

Within the same section, the following function is added:

```
// PhysicalOffsetIsValid()
// =================
// Returns whether the Physical offset for the timestamp is valid
boolean PhysicalOffsetIsValid()
    if !HaveAArch64() then
        return FALSE;
    elsif !HaveEL(EL2) || !HaveECVExt() then
        return FALSE;
    elsif HaveEL(EL3) && SCR_EL3.NS == '1' && EffectiveSCR_EL3_RW() == '0' then
        return FALSE;
    elsif HaveEL(EL3) && SCR_EL3.ECVEn == '0' then
        return FALSE;
    elsif CNTHCTL_EL2.ECV == '0' then
        return FALSE;
    else
        return TRUE;
```

In section J1.3.3 (shared/functions), the function EffectiveSCR_EL3_RW() is added to return the Effective value of SCR_EL3.RW.

2.129 D19452

Following the update communicated as D18736, in section I5.8.7 (ERRCRICR2, Critical Error Interrupt Configuration Register 2), the text that reads:

> If accessed as a Non-secure access, access to this field is **RES1**.

is updated to read:

> If accessed as a Non-secure or Realm access, access to this field is WI.

The equivalent changes are made in the following sections:

- I5.8.13 (ERRERICR2, Error Recovery Interrupt Configuration Register 2).
- I5.8.16 (ERRFHICR2, Faulting Handling Interrupt Configuration Register 2).

2.130 D19502

In section A2.2.1 (Additional functionality added to Armv8.0 in later releases), the text in the definition of 'FEAT_CSV2_1p1 and FEAT_CSV2_1p2, Cache Speculation Variant 2' that reads:

> These features are supported in AArch64 state only.
>
> These features are OPTIONAL in Armv8.0 implementations.
>
> The ID_AA64PFR1_EL1.CSV2_frac field identifies the presence of FEAT_CSV2_1p1 and FEAT_CSV2_1p2.

is updated to read:

> FEAT_CSV2_1p1 is supported in both AArch64 and AArch32 states.
>
> FEAT_CSV2_1p2 is supported in AArch64 state only.
>
> These features are OPTIONAL in Armv8.0 implementations.
>
> The following fields identify the presence of FEAT_CSV2_1p1:
> - ID_AA64PFR1_EL1.CSV2_frac.
> - ID_PFR0_EL1.CSV2.
> - ID_PFR0.CSV2.
>
> The ID_AA64PFR1_EL1.CSV2_frac field identifies the presence of FEAT_CSV2_1p2.

In section D13.2.87 (ID_PFR0_EL1, AArch32 Processor Feature Register 0), in CSV2, bits [19:16], the text that reads:

> FEAT_CSV2 implements the functionality identified by the values `0b0001` and `0b0010`.

is updated to read:

> FEAT_CSV2 implements the functionality identified by the value `0b0001`.
>
> FEAT_CSV2_1p1 implements the functionality identified by the value `0b0010`.

The equivalent changes are made in section G8.2.98 (ID_PFR0, Processor Feature Register 0).

2.131 D19503

In section J1.1.5 (aarch64/translation), the functions AArch64.HaveS1TG() and AArch64.HaveS2TG() are added, reading:

```
// AArch64.HaveS1TG()
// ==================
```

```
// Determine whether the given translation granule is supported for stage 1

boolean AArch64.HaveS1TG(TGx tgx)
   case tgx of
        when TGx_4KB  return boolean IMPLEMENTATION_DEFINED "Has 4K Translation
 Granule";
        when TGx_16KB return boolean IMPLEMENTATION_DEFINED "Has 16K Translation
 Granule";
        when TGx_64KB return boolean IMPLEMENTATION_DEFINED "Has 64K Translation
 Granule";

// AArch64.HaveS2TG()
// ==================
// Determine whether the given translation granule is supported for stage 2

boolean AArch64.HaveS2TG(TGx tgx)
    assert HaveEL(EL2);

    if HaveGTGExt() then
        case tgx of
            when TGx_4KB  return boolean IMPLEMENTATION_DEFINED "Has Stage 2 4K
 Translation Granule";
            when TGx_16KB return boolean IMPLEMENTATION_DEFINED "Has Stage 2 16K
 Translation Granule";
            when TGx_64KB return boolean IMPLEMENTATION_DEFINED "Has Stage 2 64K
 Translation Granule";
    else
        return AArch64.HaveS1TG(tgx);
```

In section J1.1.5 (aarch64/translation), the functions AArch64.DecodeTG0() and AArch64.DecodeTG1() are replaced by the functions AArch64.S1DecodeTG0(), AArch64.S1DecodeTG1() and AArch64.S2DecodeTG0(), reading:

```
// AArch64.S1DecodeTG0()
// =====================
// Decode stage 1 granule size configuration bits TG0

TGx AArch64.S1DecodeTG0(bits(2) tg0_in)
    bits(2) tg0 = tg0_in;
    TGx tgx;

    if tg0 == '11' then
        tg0 = bits(2) IMPLEMENTATION_DEFINED "TG0 encoded granule size";

    case tg0 of
        when '00'   tgx = TGx_4KB;
        when '01'   tgx = TGx_64KB;
        when '10'   tgx = TGx_16KB;

    if !AArch64.HaveS1TG(tgx) then
        case bits(2) IMPLEMENTATION_DEFINED "TG0 encoded granule size" of
            when '00'   tgx = TGx_4KB;
            when '01'   tgx = TGx_64KB;
            when '10'   tgx = TGx_16KB;

    return tgx;

// AArch64.S1DecodeTG1()
// =====================
// Decode stage 1 granule size configuration bits TG1

TGx AArch64.S1DecodeTG1(bits(2) tg1_in)
    bits(2) tg1 = tg1_in;
    TGx tgx;

    if tg1 == '00' then
        tg1 = bits(2) IMPLEMENTATION_DEFINED "TG1 encoded granule size";
```

```
    case tg1 of
        when '10'   tgx = TGx_4KB;
        when '11'   tgx = TGx_64KB;
        when '01'   tgx = TGx_16KB;

    if !AArch64.HaveS1TG(tgx) then
        case bits(2) IMPLEMENTATION_DEFINED "TG1 encoded granule size" of
            when '10'   tgx = TGx_4KB;
            when '11'   tgx = TGx_64KB;
            when '01'   tgx = TGx_16KB;

    return tgx;

// AArch64.S2DecodeTG0()
// =====================
// Decode stage 2 granule size configuration bits TG0

TGx AArch64.S2DecodeTG0(bits(2) tg0_in)
    bits(2) tg0 = tg0_in;
    TGx tgx;

    if tg0 == '11' then
        tg0 = bits(2) IMPLEMENTATION_DEFINED "TG0 encoded granule size";

    case tg0 of
        when '00'   tgx = TGx_4KB;
        when '01'   tgx = TGx_64KB;
        when '10'   tgx = TGx_16KB;

    if !AArch64.HaveS2TG(tgx) then
        case bits(2) IMPLEMENTATION_DEFINED "TG0 encoded granule size" of
            when '00'   tgx = TGx_4KB;
            when '01'   tgx = TGx_64KB;
            when '10'   tgx = TGx_16KB;

    return tgx;
```

Call sites for the previous function calls are updated to call the appropriate decode TGx function.

2.132 R19519

In section B2.3.10 (Restrictions on the effects of speculation), in the subsection ‘Restrictions on the effects of speculation from Armv8.5’, the sub bullet point that reads as follows after the changes introduced by D18924:

- Data Value predictions based on data value from execution in context1.

is updated to include the following Note:

PSTATE.{N,Z,C,V} values from context1 are not considered a data value for this purpose.

The equivalent change is made in section E2.3.9 (Restrictions on the effects of speculation), in the subsection ‘Further restrictions on the effects of speculation from Armv8.5’.

In section C5.6.3 (DVP RCTX, Data Value Prediction Restriction by Context), the following note is added:

The prediction of the PSTATE.{N,Z,C,V} values is not considered a data value for this purpose.

The equivalent change is made in section G8.2.50 (DVPRCTX, Data Value Prediction Restriction by Context).

2.133 D19521

In section C5.2.25 (SVCR, Streaming Vector Control Register), for the field ZA, bit [1], the text that reads:

> When a write to SVCR.ZA changes the value of PSTATE.ZA, the following applies:
>
> When changed from 0 to 1, all implemented bits of the storage are set to zero.
>
> When changed from 1 to 0, there is no observable change to the storage.
>
> Changes to this field do not have an affect on the SVE vector and predicate registers and FPSR.

is corrected to read:

> When a write to SVCR.ZA changes the value of PSTATE.ZA from 0 to 1, all implemented bits of the storage are set to zero.
>
> Changes to this field do not have an effect on the SVE vector and predicate registers and FPSR.

2.134 D19549

In section D8.11.3 (Common event numbers), in the subsection '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 updated to read:

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

2.135 D19560

In section D13.2.26 (CCSIDR_EL1, Current Cache Size Register), the text in LineSize, bits [2:0] when FEAT_CCIDX is implemented that reads:

> When FEAT_MTE is implemented and enabled, where a cache only holds Allocation tags, this field is **RES0**.

is changed to read:

> When FEAT_MTE is implemented, where a cache only holds Allocation tags, this field is **RES0**.

The following text is added to LineSize, bits [2:0] when FEAT_CCIDX is not implemented:

> When FEAT_MTE is implemented, where a cache only holds Allocation tags, this field is **RES0**.

2.136 D19561

In section D13.2.107 (RGSR_EL1, Random Allocation Tag Seed Register), the text that reads:

> When GCR_EL1.RRND=0, direct and indirect reads and writes to the register appear to occur in program order relative to other instructions, without the need for any explicit synchronization.

is changed to read:

> Direct and indirect reads and writes to the register appear to occur in program order relative to other instructions, without the need for any explicit synchronization.

2.137 D19581

In the function AArch64.RestrictPrediction() in section J1.1.4 (aarch64/instrs), the code that reads:

```
// If the instruction is executed at an EL lower than the specified
// level, it is treated as a NOP.
if UInt(target_el) > UInt(PSTATE.EL) then return;
```

Is updated to read:

```
// If the target EL is not implemented or the instruction is executed at an
// EL lower than the specified level, the instruction is treated as a NOP.
if !HaveEL(target_el) || UInt(target_el) > UInt(PSTATE.EL) then EndOfInstruction();
```

This affects the A64 System instructions in the following sections:

- C5.6.1 (CFP RCTX, Control Flow Prediction Restriction by Context)
- C5.6.2 (CPP RCTX, Cache Prefetch Prediction Restriction by Context).
- C5.6.3 (DVP RCTX, Data Value Prediction Restriction by Context).

An equivalent change is made in AArch32.RestrictPrediction() affecting the AArch32 System Registers in the following sections:

- G8.2.26 (CFPRCTX, Control Flow Prediction Restriction by Context).
- G8.2.34 (CPPRCTX, Cache Prefetch Prediction Restriction by Context).
- G8.2.50 (DVPRCTX, Data Value Prediction Restriction by Context).

2.138 D19630

In section I5.3.25 (PMEVTYPER<n>_EL0, Performance Monitors Event Type Registers, n = 0 - 30), the text that reads:

> PMEVTYPER<n>_EL0 is a 32-bit register.

is corrected to:

> PMEVTYPER<n>_EL0 is a 64-bit register.

The following fields are added, with appropriately reworded contents taken from section I5.3.24 (PMEVFILTR<n>, Performance Monitors Event Type Select Register <n>, n = 0 - 30):

- TC, bits [63:61].
- TH, bits [43:32].
- Bits [60:44] are Reserved, **RES0**.

The following text is added to the 'Accessing the PMEVTYPER<n>_EL0' subsection:

> If FEAT_PMUv3_TH or FEAT_PMUv3p8 is implemented, bits [63:32] of this interface are accessible at offset `0xA00` + (4*n). Otherwise, accesses at this offset are IMPLEMENTATION DEFINED.

And a new accessor encoding is added:

Component	Offset	Instance	Range
PMU	`0xA00` + (4 * n)	PMEVTYPER<n>_EL0	63:32

> This interface is accessible as follows:
>
> - When IsCorePowered(), !DoubleLockStatus(), !OSLockStatus(), AllowExternalPMUAccess() and SoftwareLockStatus(), accesses to this register are RO.
> - When IsCorePowered(), !DoubleLockStatus(), !OSLockStatus(), AllowExternalPMUAccess() and !SoftwareLockStatus(), accesses to this register are RW.
>
> Otherwise, accesses to this register generate an error response.

The external register in section I5.3.24 (PMEVFILTR<n>, Performance Monitors Event Type Select Register <n>, n = 0 - 30) is removed.

In addition, all references to ext-PMMIR are replaced with references to ext-PMEVTYPER<n>_EL0 across *Arm® Architecture Reference Manual, for A-profile architecture*.

2.139 D19642

In section D8.11.3 (Common event numbers), subsection 'Common microarchitectural events', the PMU events that read:

> `0x4025`, MEM_ACCESS_RD_CHECKED, Checked data memory access, read
>
> `0x4026`, MEM_ACCESS_WR_CHECKED, Checked data memory access, write

are corrected to read:

> `0x4025`, MEM_ACCESS_CHECKED_RD, Checked data memory access, read
>
> `0x4026`, MEM_ACCESS_CHECKED_WR, Checked data memory access, write

2.140 C19644

In section D8.11.3 (Common event numbers), subsection 'Common microarchitectural events', the text in the descriptions of MEM_ACCESS_CHECKED_RD (`0x4025`) and MEM_ACCESS_CHECKED_WR (`0x4026`) that reads:

> Implementation of this optional event requires that FEAT_MTE is implemented.

is corrected to read:

> Implementation of this optional event requires that FEAT_MTE2 is implemented.

This text is also added to the MEM_ACCESS_CHECKED (`0x4024`) event description.

2.141 C19649

In section B2.7.2 (Device Memory), in subsection 'Reordering', the bullet point in the note that reads:

> The non-Reordering property is only required by the architecture to apply the order of arrival of accesses to a single memory-mapped peripheral of an **IMPLEMENTATION DEFINED** size, and is not required to have an impact on the order of observation of memory accesses to SDRAM. For this reason, there is no effect of the non-Reordering attribute on the ordering relations between accesses to different locations described in Ordering relations on page B2-161 as part of the formal definition of the memory model.

is updated to read:

> The non-Reordering property is only required by the architecture to apply the order of arrival of accesses to a single memory-mapped peripheral of an **IMPLEMENTATION DEFINED** size, and is not required to have an impact on the order of observation of memory accesses to SDRAM. For

this reason, there is no effect of the non-Reordering attribute on the ordering relations between accesses to different locations described in B2.3.3 Ordering relations on page B2-161 as part of the formal definition of the memory model. It does have an effect on the Peripheral Coherence Order described in section B2.3.7 (Completion and endpoint ordering).

2.142 C19749

In section D13.2.51 (HDFGWTR_EL2, Hypervisor Debug Fine-Grained Write Trap Register), in field PMCCNTR_EL0, bit [15], the following text is added:

> 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.

and in field PMEVCNTRn_EL0, bit [12], the following text is added:

> 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.

The same changes are made in the corresponding fields in section D13.2.50 (HDFGRTR_EL2, Hypervisor Debug Fine-Grained Read Trap Register).

2.143 D447: SVE2

In section C8.2.143 (FMAX (vectors)), the text that reads:

> If one element value is numeric and the other is a quiet NaN, then the result is the numeric value.

is corrected to read:

> If either element value is NaN then the result is NaN.

In section C8.2.145 (FMAXNM (vectors)), the text that reads:

> If one element value is NaN then the result is the numeric value.

is corrected to read:

> If one element value is numeric and the other is a quiet NaN, then the result is the numeric value.

In section C8.2.151 (FMIN (vectors)), the text that reads:

> If the element value is a quiet NaN, then the result is the immediate.

is corrected to read:

> If either element value is NaN then the result is NaN.

In section C8.2.152 (FMINNM (vectors)), the text that reads:

> If one element value is numeric and the other is a quiet NaN, then the result is the numeric value.

is corrected to read:

> If the element value is a quiet NaN, then the result is the immediate.

2.144 D449: SVE2

In chapter C8 (SVE Instruction Descriptions), in the following subsections:

- C8.2.520 (SQDMULH (indexed)).
- C8.2.521 (SQDMULH (vectors)).
- C8.2.538 (SQRDMLAH (indexed)).
- C8.2.539 (SQRDMLAH (vectors)).
- C8.2.542 (SQRDMULH (indexed)).
- C8.2.543 (SQRDMULH (vectors)).

The following text is added to each instruction description:

> Operational information
>
> If FEAT_SVE2 is implemented or FEAT_SME is implemented, then when PSTATE.DIT is 1:
> - The execution time of this instruction is independent of:
>   - The values of the data supplied in any of its registers.
>   - The values of the NZCV flags.
> - The response of this instruction to asynchronous exceptions does not vary based on:
>   - The values of the data supplied in any of its registers.
>   - The values of the NZCV flags.

2.145 D1298: Armv9 Debug

In section J1.3.3 (shared/functions), AArch64.BranchAddr() is updated to take an Exception level as a parameter.

The signature for AArch64.BranchAddr() that reads:

```
bits(64) AArch64.BranchAddr(bits(64) vaddress)
```

Is updated to read:

```
bits(64) AArch64.BranchAddr(bits(64) vaddress, bits(2) el)
```

In section J1.1.1 (aarch64/debug), the code in BRBEException() that reads as:

```
       bits(64) source_address = if source_valid then preferred_exception_return
else Zeros(64);
       if !target_valid then target_address = Zeros(64);

       UpdateBranchRecordBuffer(ccu, cc, lastfailed, transactional, branch_type,
el, mispredict,
                                sv:tv, source_address, target_address);
```

Is updated to read:

```
       bits(64) source_address = if source_valid then preferred_exception_return
else Zeros(64);

       if !target_valid then
           target_address = Zeros(64);
       else
           target_address = AArch64.BranchAddr(target_address, target_el);

       UpdateBranchRecordBuffer(ccu, cc, lastfailed, transactional, branch_type,
el, mispredict,
                                sv:tv, source_address, target_address);
```

2.146 C1378: Armv9 Debug

In section H9.3.46 (TRCLAR, Lock Access Register), the text under 'Configurations' that reads:

> This register is present only when FEAT_ETE is implemented.

is changed to read:

> This register is present only when FEAT_ETE is implemented and the Software Lock is implemented.

2.147 D1406: Armv9 Debug

In section J1.1.1 (aarch64/debug), the code in the function AArch64.BranchRecordAllowed() that reads:

```
boolean BranchRecordAllowed(bits(2) el)
    if BRBFCR_EL1.PAUSED == '1' then
        return FALSE;
    ...
```

is extended to read:

```
boolean BranchRecordAllowed(bits(2) el)
    if ELUsingAArch32(el) then
        return FALSE;

    if BRBFCR_EL1.PAUSED == '1' then
        return FALSE;
    ...
```

2.148 D1445: Armv9 Debug

In section J1.3.4 (shared/trace), the code in the function TraceAllowed() that reads:

```
if HaveTraceBufferExtension() && TraceBufferEnabled() then
    (owning_ss, owning_el) = TraceBufferOwner();
    if (ss != owning_ss || UInt(owning_el) < UInt(el) ||
        (EffectiveTGE() == '1' && el == EL1)) then
        trace_allowed = FALSE;
```

is changed to read:

```
if HaveTraceBufferExtension() && TraceBufferEnabled() then
    (owning_ss, owning_el) = TraceBufferOwner();
    if (ss != owning_ss || UInt(owning_el) < UInt(el) ||
        (EffectiveTGE() == '1' && owning_el == EL1)) then
        trace_allowed = FALSE;
```

2.149 D1466: Armv9 Debug

In section D8.11.3 (Common event numbers), in the subsection ‘Common microarchitectural events’, the description for each CTI_TRIGOUT<n> event, where <n> is in the range 4 to 7, that reads:

> This event must be implemented if FEAT_ETE is implemented.

is updated to read:

> This event must be implemented if FEAT_ETE is implemented and TRCIDR5.NUMEXTINSEL > (n - 4).