Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile

Known issues in Issue G.a

Non-Confidential

Issue 05
102105_G.a_05_en

Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile

Known issues in Issue G.a

Release information

Document history

Issue	Date	Confidentiality	Change
F.c-00	27 August 2020	Non-Confidential	Known Issues in Arm® Architecture Reference Manual, Issue F.c, as of 21 August 2020
F.c-01	30 September 2020	Non-Confidential	Known Issues in Arm® Architecture Reference Manual, Issue F.c, as of 25 September 2020
F.c-02	30 October 2020	Non-Confidential	Known Issues in Arm® Architecture Reference Manual, Issue F.c, as of 23 October 2020
F.c-03	30 November 2020	Non-Confidential	Known Issues in Arm® Architecture Reference Manual, Issue F.c, as of 20 November 2020
F.c-04	18 December 2020	Non-Confidential	Known Issues in Arm® Architecture Reference Manual, Issue F.c, as of 18 December 2020
G.a-00	22 January 2021	Non-Confidential	Known Issues in Arm® Architecture Reference Manual, Issue G.a, as of 18 December 2020
G.a-01	26 February 2021	Non-Confidential	Known Issues in Arm® Architecture Reference Manual, Issue G.a, as of 19 February 2021
G.a-02	31 March 2021	Non-Confidential	Known Issues in Arm® Architecture Reference Manual, Issue G.a, as of 19 March 2021
G.a-03	30 April 2021	Non-Confidential	Known Issues in Arm® Architecture Reference Manual, Issue G.a, as of 23 April 2021
G.a-04	31 May 2021	Non-Confidential	Known Issues in Arm® Architecture Reference Manual, Issue G.a, as of 21 May 2021
G.a-05	30 June 2021	Non-Confidential	Known Issues in Arm® Architecture Reference Manual, Issue G.a, as of 18 June 2021

Proprietary Notice

Confidentiality Status

Product Status

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

Web address

developer.arm.com

Inclusive language commitment

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	Introduces special terminology, denotes cross-references, and citations.
bold	Highlights interface elements, such as menu names. Denotes signal names. Also used for terms in descriptive lists, where appropriate.
`monospace`	Denotes text that you can enter at the keyboard, such as commands, file and program names, and source code.
`monospace italic`	Denotes arguments to monospace text where the argument is to be replaced by a specific value.
`monospace bold`	Denotes language keywords when used outside example code.
`monospace underline`	Denotes 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	Used in body text for a few terms that have specific technical meanings, that are defined in the *Arm Glossary*. For example, IMPLEMENTATION DEFINED, IMPLEMENTATION SPECIFIC, UNKNOWN, and UNPREDICTABLE.
⚠ Caution	This represents a recommendation which, if not followed, might lead to system failure or damage.
⚠ Warning	This represents a requirement for the system that, if not followed, might result in system failure or damage.
⚠ Danger	This represents a requirement for the system that, if not followed, will result in system failure or damage.

Convention	Use
Note	This represents an important piece of information that needs your attention.
Tip	This represents a useful tip that might make it easier, better or faster to perform a task.
Remember	This is 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 Armv8, for Armv8-A architecture profile, Issue G.a	DDI 0487G.a	No

1.3 Feedback

Arm welcomes feedback on this product and its documentation.

Feedback on this product

If you have any comments or suggestions about this product, contact your supplier and give:

- The product name.
- The product revision or version.
- An explanation with as much information as you can provide. Include symptoms and diagnostic procedures if appropriate.

Feedback on content

If you have comments on content then send an e-mail to errata@arm.com. Give:

- The title Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile Known issues in Issue G.a.
- The number 102105_G.a_05_en.
- If applicable, the page number(s) to which your comments refer.
- A concise explanation of your comments.

Arm also welcomes general suggestions for additions and improvements.

Note

Arm tests the PDF only in Adobe Acrobat and Acrobat Reader, and cannot guarantee the quality of the represented document when used with any other PDF reader.

1.4 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, Armv8, for Armv8-A architecture profile (DD10487), Issue G.a.

Issue G.a is the initial release of the Arm Architecture Reference Manual including the Armv8.7 architecture extensions.

The pseudocode descriptions for the Armv8.7 architecture extensions in Issue G.a are at Alpha quality, which means that most major features of the specification are described in the manual, some features and details might be missing. For the latest status and updates to the pseudocode descriptions, please see: https://developer.arm.com/architectures/cpu-architecture/a-profile/exploration-tools.

The remainder of the information in Issue G.a is at EAC quality, which means that all features of the specification are described in the manual.

Key

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

2.1 D8901

In section G1.21.2 (PL1 configurable controls), subsection 'Disabling or enabling PL0 and PL1 use of AArch32 deprecated functionality', the bullet points that read:

> - The SED control is always implemented.
> - Whether each of the ITD, CP15BEN controls is implemented is **IMPLEMENTATION DEFINED**. If a control is not implemented, then the associated functionality cannot be disabled.

are changed to read:

> - The SED control is implemented if the implementation supports mixed-endian operation at any Exception level.
> - Whether the ITD control is implemented is **IMPLEMENTATION DEFINED**.
> - Whether the CP15BEN control is implemented is **IMPLEMENTATION DEFINED**.
> - If a control is not implemented, then the associated functionality cannot be disabled.

The equivalent changes are made in sections D1.14.2 (EL1 configurable controls), subsection 'Disabling or enabling EL0 use of AArch32 deprecated functionality', and G1.21.3 (EL2 configurable controls), subsection 'Disabling or enabling EL2 use of deprecated AArch32 functionality'. Instances of 'deprecated' are changed to 'optional' in the subsection titles.

In section D13.2.116 (SCTLR_EL1), the following text in the ITD field description:

> ITD is optional, but if it is implemented in the SCTLR then it must also be implemented in the SCTLR_EL1.

is changed to read:

> ITD is optional, but if it is implemented in the SCTLR_EL1 then it must also be implemented in the SCTLR_EL2, HSCTLR, and SCTLR.

The equivalent change is made to the SCTLR_EL1.CP15BEN field description, and to the ITD and CP15BEN fields in sections D13.2.117 (SCTLR_EL2), G8.2.72 (HSCTLR), and G8.2.126 (SCTLR).

2.2 D10129

In section K1.1.33 (**CONSTRAINED UNPREDICTABLE** behavior of EL2 features), the sub-section 'Accessing registers that cannot be accessed using MSR/MRS instructions' is deleted, and the contents of sub-section 'MSR (banked register) and MRS (banked register)' are replaced with:

> If the target register specified by the {R, SYSm} fields of the instruction encoding is not accessible from the PE mode in which the instruction was executed (see 'Usage restrictions on the banked register transfer instructions'), then one of the following behaviors must occur:
>
> - The instruction is UNDEFINED.
> - The instruction executes as a NOP.
> - For MRS (banked register) instructions, the destination general-purpose register becomes **UNKNOWN**.
> - For MSR (banked register) instructions, if the register specified could be accessed from the current mode by other mechanisms, then this register is **UNKNOWN**. Otherwise, the instruction is a NOP.
>
> If the instruction was executed specifying an unallocated {R, SYSm} field value or an unimplemented register (see 'Encoding the register argument in the banked register transfer instructions'), then one of the following behaviors must occur:
>
> - The instruction is UNDEFINED.
> - The instruction executes as a NOP.
> - An allocated MRS (banked register) or MSR (banked register) instruction is executed.

2.3 R10550

In section F5.1.228 (STMIB, STMFA), in the description of the **CONSTRAINED UNPREDICTABLE** behavior when BitCount(registers) < 1, the bullet that reads:

> The instruction operates as an STM with the same addressing mode but targeting an unspecified set of registers. These registers might include R15.

is corrected to read:

> The instruction operates as an STM with the same addressing mode but targeting an unspecified set of registers. These registers might include R15. If the instruction specifies writeback, the modification to the base address on writeback might differ from the number of registers stored.

2.4 D11312

In sections D3.2.1 (Controls to prohibit trace at Exception levels) and G3.2.1 (Controls to prohibit trace at Exception levels), Tables D3-1 'Export of PMU events prohibited' and G3-1 'Export of PMU events prohibited' are deleted, and are replaced with the following text:

> If SelfHostedTraceEnabled() == TRUE, no events are exported to the PE Trace Unit when tracing is prohibited.
>
> If SelfHostedTraceEnabled() == FALSE, no events are exported to the PE Trace Unit when the PE is in Secure state and counting in Secure state is prohibited.
>
> When PMCR_EL0.X==0 or PMCR.X==0, no PMU events are exported to the PE Trace Unit.
>
> Otherwise, PMU events are exported to the PE Trace Unit.

2.5 D12962

In section G8.3.22 (DBGOSLAR), under the 'Configurations' heading, the text that reads:

> AArch32 System register DBGOSLAR bits [31:0] are architecturally mapped to AArch64 System register OSLAR_EL1[31:0]. AArch32 System register DBGOSLAR bits [31:0] are architecturally mapped to External register OSLAR_EL1[31:0].

is deleted, and is replaced with the following text:

> The OS lock can also be locked or unlocked using the AArch64 System register OSLAR_EL1 and External register OSLAR_EL1.

The equivalent changes are made in section D13.3.25 (OSLAR_EL1), and H9.2.46 (OSLAR_EL1).

2.6 E13850

In section J1 (Armv8 Pseudocode), the description of the pseudocode functions AArch64.TranslateAddress() and AArch32.TranslateAddress(), and the functions they call pertaining to VMSA implementation, is refactored as follows:

- For each enabled stage of translation, a common structure containing all the parameters affecting translation is populated from the appropriate registers for the regime.
- A structure representing the current walk state is populated with initial values based on the walk parameters.
- The walk process iteratively updates the walk state, and state that would be reported in a fault, until a result or a fault is detected.
- For a successful translation, the final walk state is used to perform the translation and the fault information is discarded.
- For a faulting access, the walk state is discarded and the fault state is used to populate the appropriate syndrome registers.

Additionally, the pseudocode is updated to describe the behavior of FEAT_LPA2, FEAT_PAN3, and FEAT_XS.

2.7 D14169

In section D5.2.9 (The effects of disabling a stage of address translation), the following bulleted text:

- No memory access permission checks are performed and therefore no MMU faults can be generated for this stage of address translation.

is corrected to read:

- No memory access permission checks are performed, and therefore no MMU Permission faults can be generated for this stage of address translation.

The equivalent change is made in section G5.2.1 (VMSAv8-32 behavior when stage 1 address translation is disabled).

2.8 E14362

In section B2.2.5 (Concurrent modification and execution of instructions), the two lists of instructions supported by concurrent modification and execution are updated to include the B.cond, CBZ, TBZ, CBNZ, and TBNZ instructions.

2.9 D14810

In section D13.2.148 (VTCR_EL2, Virtualization Translation Control Register), in the values table for the VS field, references to the System register VSTTBR_EL2 are removed.

2.10 D14928

In section D9.6.9 (Exceptions), the text that reads:

> If the sampled operation generates an exception, it is UNPREDICTABLE whether the sample record contains any other information.
>
> Where a sampled operation generates an exception and the type of exception means that a particular item is not computed by the sampled operation, that information is not collected by the profiling operation. For more information, see Synchronization and Statistical Profiling on page D9-2953.

is corrected to read:

> If the sampled operation generates an exception condition, it is UNPREDICTABLE whether the sample record contains any other information. This includes operations that generate faults or other exception conditions but do not generate exceptions. For example:
>
> - An instruction on a misspeculated path.
> - A load operation that is part of a Non-fault load instruction or is not the First active element of a First-fault load instruction that generates an MMU fault or watchpoint.
> - An address translation operation or prefetch instruction that generates an MMU fault.
>
> Where a sampled operation generates an exception condition and the type of exception condition means that a particular item is not computed by the sampled operation, that information is not collected by the profiling operation. For more information, see Synchronization and Statistical Profiling on page D9-2953.

2.11 D15346

In section D1.7.1 (Accessing PSTATE fields), the following text:

> PSTATE.{N, Z, C, V, TCO} can be accessed at EL0. Access to PSTATE.{D, A, I, F} at EL0 using AArch64 depends on SCTLR_EL1.UMA, see Traps to EL1 of EL0 accesses to the PSTATE.{D, A, I, F} interrupt masks on page D1-2496. All other PSTATE access instructions can be executed at EL1 or higher and are **UNDEFINED** at EL0.

is corrected to read:

PSTATE.{N, Z, C, V, SSBS, DIT, TCO} can be accessed at EL0. Access to PSTATE.{D, A, I, F} at EL0 using AArch64 depends on SCTLR_EL1.UMA, see Traps to EL1 of EL0 accesses to the PSTATE.{D, A, I, F} interrupt masks on page D1-2496. All other PSTATE access instructions can be executed at EL1 or higher and are **UNDEFINED** at EL0.

2.12 D15977

In section D13.2.115 (SCR_EL3, Secure Configuration Register), the bullet point in the API, bit [17] field when FEAT_SEL2 is not implemented and FEAT_PAuth is implemented, that reads:

- In Secure EL0, when the associated SCTLR_EL2.En<N><M> == 1.

is corrected to read:

- In Secure EL0, when the associated SCTLR_EL1.En<N><M> == 1.

2.13 D16029

In section D13.8.29 (CNTVCT_EL0, Counter-timer Virtual Count register), the accessibility pseudocode that currently reads:

```
if PSTATE.EL == EL0 then
    if !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') && CNTKCTL_EL1.EL0VCTEN == '0'
 then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && HCR_EL2.<E2H,TGE> == '11' && CNTHCTL_EL2.EL0VCTEN == '0'
 then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.<E2H,TGE> != '11' && CNTHCTL_EL2.EL1TVCT == '1'
 then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        return CNTVCT_EL0;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && CNTHCTL_EL2.EL1TVCT == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        return CNTVCT_EL0;
elsif PSTATE.EL == EL2 then
    return CNTVCT_EL0;
elsif PSTATE.EL == EL3 then
    return CNTVCT_EL0;
```

is changed to read:

```
if PSTATE.EL == EL0 then
    if !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') && CNTKCTL_EL1.EL0VCTEN == '0'
 then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && HCR_EL2.<E2H,TGE> == '11' && CNTHCTL_EL2.EL0VCTEN == '0'
 then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.<E2H,TGE> != '11' && CNTHCTL_EL2.EL1TVCT == '1'
 then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        if HaveEL(EL2) && (!EL2Enabled() || HCR_EL2.<E2H,TGE> != '11') then
            return PhysicalCountInt() - CNTVOFF_EL2;
        else
            return PhysicalCountInt();
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && CNTHCTL_EL2.EL1TVCT == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        if HaveEL(EL2) then
            return PhysicalCountInt() - CNTVOFF_EL2;
        else
            return PhysicalCountInt();
elsif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '0' then
        return PhysicalCountInt() - CNTVOFF_EL2;
    else
        return PhysicalCountInt();
elsif PSTATE.EL == EL3 then
    if HaveEL(EL2) && !ELUsingAArch32(EL2) then
        return PhysicalCountInt() - CNTVOFF_EL2;
    elsif HaveEL(EL2) && EL2UsingAArch32(EL2) then
        return PhysicalCountInt() - CNTVOFF;
    else
        return PhysicalCountInt();
```

Equivalent changes are made in the following sections:

- D13.8.28 (CNTVCTSS_EL0, Counter-timer Self-Synchronized Virtual Count register),
- G8.7.24 (CNTVCT, Counter-timer Virtual Count register),
- G8.7.25 (CNTVCTSS, Counter-timer Self-Synchronized Virtual Count register).

In section D13.8.20 (CNTPCT_EL0, Counter-timer Physical Count), the accessibility pseudocode that currently reads:

```
if PSTATE.EL == EL0 then
    if !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') && CNTKCTL_EL1.EL0PCTEN == '0'
 then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && HCR_EL2.E2H == '0' && CNTHCTL_EL2.EL1PCTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.<E2H,TGE> == '10' && CNTHCTL_EL2.EL1PCTEN == '0'
 then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.<E2H,TGE> == '11' && CNTHCTL_EL2.EL0PCTEN == '0'
 then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        return CNTPCT_EL0;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && CNTHCTL_EL2.EL1PCTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        return CNTPCT_EL0;
elsif PSTATE.EL == EL2 then
    return CNTPCT_EL0;
elsif PSTATE.EL == EL3 then
    return CNTPCT_EL0;
```

is changed to read:

```
if PSTATE.EL == EL0 then
    if !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') && CNTKCTL_EL1.EL0PCTEN == '0'
 then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && HCR_EL2.E2H == '0' && CNTHCTL_EL2.EL1PCTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.<E2H,TGE> == '10' && CNTHCTL_EL2.EL1PCTEN == '0'
 then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.<E2H,TGE> == '11' && CNTHCTL_EL2.EL0PCTEN == '0'
 then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        if IsFeatureImplemented(FEAT_ECV) && EL2Enabled() && SCR_EL3.ECVEn == '1' &&
 CNTHCTL_EL2.ECV == '1' && HCR_EL2.<E2H,TGE> != '11' then
            return PhysicalCountInt() - CNTPOFF_EL2;
        else
            return PhysicalCountInt();
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && CNTHCTL_EL2.EL1PCTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        if IsFeatureImplemented(FEAT_ECV) && EL2Enabled() && SCR_EL3.ECVEn == '1' &&
 CNTHCTL_EL2.ECV == '1' then
            return PhysicalCountInt() - CNTPOFF_EL2;
        else
            return PhysicalCountInt();
elsif PSTATE.EL == EL2 then
    return PhysicalCountInt();
elsif PSTATE.EL == EL3 then
    return PhysicalCountInt();
```

Equivalent changes are made in the following sections:

- D13.8.19 (CNTPCTSS_EL0, Counter-timer Self-Synchronized Physical Count register),
- G8.7.19 (CNTPCT, Counter-timer Physical Count register),
- G8.7.20 (CNTPCTSS, Counter-timer Self-Synchronized Physical Count register).

2.14 D16093

In section D7.10.3 (Common event numbers), in the subsection 'Common microarchitectural events', instances of 'demand refill' are corrected to 'refill' in the following event descriptions:

- `0x0001`, L1I_CACHE_REFILL, Level 1 instruction cache refill,
- `0x0003`, L1D_CACHE_REFILL, Level 1 data cache refill,

- `0x0017`, L2D_CACHE_REFILL, Level 2 data cache refill,
- `0x0028`, L2I_CACHE_REFILL, Attributable Level 2 instruction cache refill,
- `0x002A`, L3D_CACHE_REFILL, Attributable Level 3 data cache refill.

2.15 D16367

In section J1.2.4 (aarch32/translation), the AArch32.CheckWatchpoint() function is updated to ignore excluded access types from the watchpoint check.

The code that reads:

```
    assert ELUsingAArch32(S1TranslationRegime());

    match = FALSE;
```

is updated to read:

```
    assert ELUsingAArch32(S1TranslationRegime());
    if acctype IN {AccType_PTW, AccType_IC, AccType_AT} then
        return AArch32.NoFault();
    if acctype == AccType_DC then
        if !iswrite then
            return AArch32.NoFault();
        elsif !(boolean IMPLEMENTATION_DEFINED \"DCIMVAC generates watchpoint\")
 then
            return AArch32.NoFault();

    match = FALSE;
```

In section J1.1.5 (aarch64/translation), the AArch64.CheckWatchpoint() function is also updated.

The code that reads:

```
    assert !ELUsingAArch32(S1TranslationRegime());

    match = FALSE;
```

is updated to read:

```
    assert !ELUsingAArch32(S1TranslationRegime());
    if acctype IN {AccType_PTW, AccType_IC, AccType_AT} then
        return AArch32.NoFault();
    if acctype == AccType_DC then
        if !iswrite then
            return AArch32.NoFault();

    match = FALSE;
```

In section J1.2.4 (aarch32/translation), in the AArch32.TranslateAddress() function, the check for the excluded access types is moved to the AArch32.CheckWatchpoint() function.

The code that reads:

```
    result = AArch32.FullTranslate(vaddress, acctype, iswrite, wasaligned, size);
    if !(acctype IN {AccType_PTW, AccType_IC, AccType_AT}) && !IsFault(result) then
        result.fault = AArch32.CheckDebug(vaddress, acctype, iswrite, size);
```

is updated to read:

```
    result = AArch32.FullTranslate(vaddress, acctype, iswrite, wasaligned, size);
    if !IsFault(result) then
        result.fault = AArch32.CheckDebug(vaddress, acctype, iswrite, size);
```

In section J1.1.5 (aarch64/translation), in the AArch64.TranslateAddress() function, the check for the excluded access types is moved to the AArch64.CheckWatchpoint() function.

The code that reads:

```
    result = AArch64.FullTranslate(vaddress, acctype, iswrite, wasaligned, size);
    if !(acctype IN {AccType_PTW, AccType_IC, AccType_AT}) && !IsFault(result) then
        result.fault = AArch64.CheckDebug(vaddress, acctype, iswrite, size);
```

is updated to read:

```
    result = AArch64.FullTranslate(vaddress, acctype, iswrite, wasaligned, size);
    if !IsFault(result) then
        result.fault = AArch64.CheckDebug(vaddress, acctype, iswrite, size);
```

2.16 D16403

In section D5.9.1 (Use of ASIDs and VMIDs to reduce TLB maintenance requirements), in the subsection 'Common not private translations', the bullet that reads:

- If an ASID applies to the stage of translation corresponding to that TTBR_ELx then the current ASID value must be the same for all of the PEs that are sharing entries for any translation table entry that is not global or leaf level.

is changed to read:

- If an ASID applies to the stage of translation corresponding to that TTBR_ELx then the current ASID value must be the same for all of the PEs that are sharing entries for any translation table entry that is not global or not leaf level.

Similar changes are made in section G5.9.1 (General TLB maintenance requirements), subsection 'Common not private translations in VMSAv8-32'.

2.17 D16493

In section C6.2.114 (LDAXP), the text that currently reads:

> A 32-bit pair requires the address to be doubleword aligned and is single-copy atomic at doubleword granularity. A 64-bit pair requires the address to be quadword aligned and is single-copy atomic for each doubleword at doubleword granularity.

is replaced by the following text:

> For information on single-copy atomicity and alignment requirements, see Requirements for single-copy atomicity and Alignment of data accesses.

Equivalent changes are made in sections C6.2.174 (LDXP), C6.2.272 (STLXP), and C6.2.305 (STXP).

2.18 D16664

In section D1.14.3 (EL2 configurable controls), the following controls are added to Table D1-21 (Instruction disables and trap controls provided by EL2), along with the relevant overview subsections in the 'Description' column of each entry:

Control	Control type	Description
CPTR_EL2.FPEN	T	Traps to EL2 of EL2, EL1, and EL0 accesses to SIMD and floating-point functionality on page D1-2504
SCTLR_EL2.UCI	T	Traps to EL2 of EL0 execution of cache maintenance instructions on page D1-2511
SCTLR_EL2.{nTWE, nTWI}	T	Traps to EL2 of EL0 and EL1 execution of WFE, WFI, WFET, and WFIT instructions on page D1-2508
SCTLR_EL2.UCT	T	Traps to EL2 of EL0 accesses to the CTR_EL0 on page D1-2511
SCTLR_EL2.DZE	T	Traps to EL2 of EL0 execution of DC ZVA instructions on page D1-2512
SCTLR_EL2.{SED, ITD}	D	Disabling or enabling EL0 use of AArch32 optional functionality on page D1-2512
SCTLR_EL2.CP15BEN	E	Disabling or enabling EL0 use of AArch32 optional functionality on page D1-2512

2.19 C16668

In section D7.6 (Multithreaded implementations), the text that reads:

> However, even when the Effective value of PMEVTYPER<n>.MT is 1, the PE does not count an event that is Attributable to Secure state or EL2 on another thread if counting events Attributable to Secure state or EL2 is prohibited on the PE that is counting the events.

is replaced by the following:

> However, even when the Effective value of PMEVTYPER<n>.MT is 1, PE_A does not count an event that is Attributable to Secure state on PE_B if counting events Attributable to Secure state

> is prohibited on PE_A. Similarly, PE_Adoes not count an event that is Attributable to EL2 on PE_B if counting events Attributable to EL2 is prohibited on PE_A.

In the same section, in example D7-1 (The effect of having PMEVTYPER<n>.MT==1), the text that reads:

> If the value of MDCR_EL3.SPME is 0, and <n> is less than PMCR.N on one thread, then event counter <n> on this thread does not count events Attributable to Secure state on another thread, even if one or both of the following applies:
>
> - This thread is in Non-secure state.
> - MDCR_EL3.SPME==1 on the other thread.

is replaced by the following:

> If the value of MDCR_EL3.SPME is 0, and <n> is less than PMCR.N on PE_A, then event counter <n> on PE_A does not count events Attributable to Secure state on PE_B, even if one or both of the following applies:
>
> - PE_A is in Non-secure state.
> - MDCR_EL3.SPME==1 on PE_B.

In the same section, in example D7-2 (The effect of having PMEVTYPER<n>.MT==1), the text that reads:

> If the value of MDCR_EL2.HPMD is 1 and <n> is less than MDCR_EL2.HPMN on one thread, then event counter <n> on this thread does not count events Attributable to EL2 on another thread, even if one of the following applies:
>
> - MDCR_EL2.HPMD==0 on the other thread.
> - This thread is not executing at EL2.

is replaced by the following:

> If the value of MDCR_EL2.HPMD is 1 and <n> is less than MDCR_EL2.HPMN on PE_A, then event counter <n> on PE_A does not count events Attributable to EL2 on PE_B, even if one of the following applies:
>
> - MDCR_EL2.HPMD==0 on PE_B.
> - PE_A is not executing at EL2.

The text later within the same section that reads:

> When the current configuration prohibits counting of events Attributable to Secure state or EL2 in Secure state or at EL2, it is **IMPLEMENTATION DEFINED** whether:
>
> - Counting events Attributable to Secure state on this PE in Non-secure state is permitted.
> - Counting events Attributable to EL2 when this PE is using another Exception level is permitted

- Counting Unattributable events related to other secure operations in the system or at EL2 is permitted. If not specified above, counting events that are not prohibited on either PE is permitted.

is replaced by the following:

When the current configuration is not multithreaded, and PE_A prohibits counting of events Attributable to Secure state when PE_A is in Secure state, it is **IMPLEMENTATION DEFINED** whether:

- Counting events Attributable to Secure state when PE_A is in Non-secure state is permitted.
- Counting Unattributable events related to other secure operations in the system when PE_A is in Non-secure state is permitted. Otherwise, counting events in Non-secure state is permitted.

When the current configuration is not multithreaded, and PE_A prohibits counting of events Attributable to EL2 when $PE_A$is at EL2, it is **IMPLEMENTATION DEFINED** whether:

- Counting events Attributable to EL2 when PE_A is using another Exception level is permitted.
- Counting Unattributable events related EL2 when PE_A is using another Exception level is permitted. Otherwise, counting events at another Exception level is permitted.

2.20 D16815

In section D13.2.101 (MVFR1_EL1, AArch32 Media and VFP Feature Register 1), in the SIMDHP field, the text that reads:

In Armv8-A, the permitted values are:

- `0b0000` in an implementation without SIMD floating-point support.
- `0b0010` in an implementation with SIMD floating-point support that does not include the FEAT_FP16 extension.
- `0b0011` in an implementation with SIMD floating-point support that includes the FEAT_FP16 extension.

is corrected to:

In Armv8-A, the permitted values are:

- `0b0000` in an implementation without SIMD floating-point support.
- `0b0001` in an implementation with SIMD floating-point support that does not include the FEAT_FP16 extension.
- `0b0010` in an implementation with SIMD floating-point support that includes the FEAT_FP16 extension.

The equivalent change is made in section G8.2.116 (MVFR1, Media and VFP Feature Register 1).

2.21 D16907

In section J1.1.2 (aarch64/exceptions), in CheckLDST64BEnabled(), CheckST64BVEnabled(), CheckST64BV0Enabled(), and AArch64.RaiseTagCheckFault(), the check for HCR_EL2 is now qualified with EL2Enabled().

In section J1.2.2 (aarch32/exceptions), in AArch32.CheckForSMCUndefOrTrap(), the check for HCR_EL2 is now qualified with EL2Enabled().

In section J1.2.2 (aarch32/exceptions), in AArch32.TakePrefetchAbortException() and AArch32.TakeDataAbortException(), the check '(HaveEL(EL2) && !IsSecure()' has been simplified by the use of EL2Enabled().

In section J1.2.2 (aarch32/exceptions), in AArch32.CheckAdvSIMDOrFPEnabled() the code that reads:

```
    if PSTATE.EL == EL0 && (!HaveEL(EL2) || (!ELUsingAArch32(EL2) && HCR_EL2.TGE ==
  '0')) && !ELUsingAArch32(EL1) then
        // The PE behaves as if FPEXC.EN is 1
        AArch64.CheckFPAdvSIMDEnabled();
    elsif PSTATE.EL == EL0 && HaveEL(EL2) && !ELUsingAArch32(EL2) && HCR_EL2.TGE ==
  '1' && !ELUsingAArch32(EL1) then
```

is updated to read:

```
    if PSTATE.EL == EL0 && (!EL2Enabled() || (!ELUsingAArch32(EL2) && HCR_EL2.TGE ==
  '0')) && !ELUsingAArch32(EL1) then
        // The PE behaves as if FPEXC.EN is 1
        AArch64.CheckFPEnabled();
        AArch64.CheckFPAdvSIMDEnabled();
    elsif PSTATE.EL == EL0 && EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE ==
  '1' && !ELUsingAArch32(EL1) then
```

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

```
    if PSTATE.EL == EL3 || route_to_el3 then
        AArch64.TakeException(EL3, exception, preferred_exception_return, vect_off\
set);
    elsif PSTATE.EL == EL2 || route_to_el2 then
        AArch64.TakeException(EL2, exception, preferred_exception_return, vect_off\
set);
    else
        AArch64.TakeException(EL1, exception, preferred_exception_return, vect_off\
set);
```

is updated to read:

```
    bits(2) target_el = EL1;
    if PSTATE.EL == EL3 || route_to_el3 then
        target_el = EL3;
    elsif PSTATE.EL == EL2 || route_to_el2 then
        target_el = EL2;
    AArch64.TakeException(target_el, exception, preferred_exception_return, vec\
t_offset);
```

In section J1.1.2 (aarch64/exceptions), in AArch64.PCAlignmentFault(), AArch64.SPAlignmentFault() and AArch64.BranchTargetException(), the code that reads:

```
    if UInt(PSTATE.EL) > UInt(EL1) then
        AArch64.TakeException(PSTATE.EL, exception, preferred_exception_return, vec\
t_offset);
    elsif EL2Enabled() && HCR_EL2.TGE == '1' then
        AArch64.TakeException(EL2, exception, preferred_exception_return, vect_off\
set);
    else
        AArch64.TakeException(EL1, exception, preferred_exception_return, vect_off\
set);
```

is updated to read:

```
    bits(2) target_el = EL1;
    if UInt(PSTATE.EL) > UInt(EL1) then
        target_el = PSTATE.EL;
    elsif EL2Enabled() && HCR_EL2.TGE == '1' then
        target_el = EL2;
    AArch64.TakeException(target_el, exception, preferred_exception_return, vec\
t_offset);
```

2.22 D16933

In section D9.6.3 (Additional information for each profiled memory access operation), after the text that reads:

> The sampled data physical address is the address generated from translating the sampled data virtual address. The sampled data physical address packet is not output if any of the following are true:
>
> - The PE does not translate the address, for example because it does not perform the access or the address translation generates a Translation fault.
> - The sampled data virtual address packet is not output.
> - Prohibited by System register controls.

The following text is added:

> If a sampled virtual address packet is not output:
>
> - It is **IMPLEMENTATION DEFINED** whether the Translation latency Counter packet for the load or store is either not recorded, or recorded with a value of zero.
> - It is **IMPLEMENTATION DEFINED** whether the bits corresponding to the access in the Events packet are recorded or always zero. If access does not occur, these bits are zero.

2.23 D16936

In section D7.10.3 (Common event numbers), in subsection 'Common microarchitectural events', in the STALL_SLOT event definition, the text that reads:

> If FEAT_PMUv3p4 is implemented:
>
> - If STALL_SLOT is not implemented, it is **IMPLEMENTATION DEFINED** whether the PMMIR System registers are implemented.
> - If STALL_SLOT is implemented, then the PMMIR System registers are implemented.

is changed to read:

> If FEAT_PMUv3p4 is implemented then PMMIR.SLOTS defines the largest value by which this event can increment the counter in a single cycle.

2.24 D16960

In section D9.8.4 (External aborts), in the bullet list under 'If a write to the Profiling Buffer generates an External abort that is reported to the Statistical Profiling Extension:', the following text is added:

> - If PMBSR_EL1.S == 0, a buffer management event is generated:
>   - PMBSR_EL1.S is set to 1.
>   - PMBSR_EL1.EC is set to one of:
>   - `0b100100`, stage 1 data abort on write to buffer.
>   - `0b100101`, stage 2 data abort on write to buffer.
>   - PMBSR_EL1.MSS is set as follows:
>   - PMBSR_EL1.FSC is set to indicate a synchronous or asynchronous external abort.

Additionally, in section D9.8.2 (Buffer full event), the following bullet is added to the list under 'If, after writing a sample record, there is not sufficient space in the Profiling Buffer for a sample record of the size indicated by PMSIDR_EL1.MaxSize, and PMBSR_EL1.S is 0, a Profiling Buffer management event is generated':

> - PMBSR_EL1.S is set to 1.

2.25 R16961

In section D9.6.3 (Additional information for each profiled memory access operation), the below text is added after the first Note:

> If FEAT_MTE2 is implemented, an instruction which loads or stores an Allocation Tag or multiple Allocation Tags will be treated as a load or store if profiling is enabled. Each Allocation Tag covers multiple locations in a Tag Granule. It is **IMPLEMENTATION DEFINED** whether the implementation treats each Allocation Tag access as an access to the data location addressed in the operation, or the whole Tag Granule. That is, whether the data virtual address associated with the sampled access or chosen part of the access is the address of the location being accessed, or the lowest address covered by the same Allocation Tag or Allocation Tags. For more information, see Chapter D6 Memory Tagging Extension.

And the following text:

> The sampled data physical address is the address generated from translating the sampled data virtual address.

Is updated to read:

> If FEAT_MTE2 is implemented and the operation is an access to an Allocation Tag or multiple Allocation Tags, it is **IMPLEMENTATION DEFINED** whether the sampled data physical address is the address generated from translating the sampled data virtual address or the address generated from translating the lowest address covered by the same Allocation Tag or Allocation Tags, when these differ. Otherwise, the sampled data physical address is the address generated from translating the sampled data virtual address.

In addition, the following paragraph is deleted:

> If FEAT_MTE2 is implemented, an instruction which loads or stores an Allocation tag will be treated as a load or store if profiling is enabled. For more information, see Chapter D6 Memory Tagging Extension.

2.26 D16997

In section D13.2.37 (ESR_EL1, Exception Syndrome Register (EL1)), under the heading 'ISS encoding an exception from an MCR or MRC access', the Rt field description that currently reads:

> The Rt value from the issued instruction, the general-purpose register used for the transfer. The reported value gives the AArch64 view of the register.

is corrected to read:

> The Rt value from the issued instruction, the general-purpose register used for the transfer. If the Rt value is not `0b1111`, then the reported value gives the AArch64 view of the register. Otherwise, if the Rt value is `0b1111`:

- If the instruction that generated the exception is not UNPREDICTABLE, then the register specifier takes the value `0b11111`.
- If the instruction that generated the exception is UNPREDICTABLE, then the register specifier takes an **UNKNOWN** value, which is restricted to either:
    - The AArch64 view of one of the registers that could have been used in AArch32 state at the Exception level that the instruction was executed at.
    - The value `0b11111`.

Similar changes are made to the Rt2 and Rt field descriptions under the heading 'ISS encoding an exception from an MCRR or MRRC access', and to the Rn field description under the heading 'ISS encoding an exception from an LDC or STC instruction'.

The equivalent changes are made in sections D13.2.38 (ESR_EL2, Exception Syndrome Register (EL2)) and D13.2.39 (ESR_EL3, Exception Syndrome Register (EL3)).

2.27 D17015

In section F5.1.216 (STC), the STC execution pseudocode is changed to read:

```
if ConditionPassed() then
    EncodingSpecificOperations();
    offset_addr = if add then (R[n] + imm32) else (R[n] - imm32);
    address = if index then offset_addr else R[n];

    // System register read from DBGDTRRXint.
    MemA[address,4] = AArch32.SysRegRead(cp, ThisInstr());

    if wback then R[n] = offset_addr;
```

In section F5.1.64 (LDC (immediate)), the LDC execution pseudocode is changed to read:

```
if ConditionPassed() then
    EncodingSpecificOperations();
    offset_addr = if add then (R[n] + imm32) else (R[n] - imm32);
    address = if index then offset_addr else R[n];

    // System register write to DBGDTRTXint.
    AArch32.SysRegWriteM(cp, ThisInstr(), address);

    if wback then R[n] = offset_addr;
```

Details of traps will be added through the use of new LDC and STC accessibility pseudocode in sections G8.3.17 (DBGDTRRXint) and G8.3.18 (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.

The equivalent change is made in section F5.1.65 (LDC (literal)).

2.28 D17020

In section D13.3.18 (MDCR_EL3, Monitor Debug Configuration Register (EL3)), in the SDD field, the following additional text is added:

> If Secure EL2 is implemented and enabled, and Secure EL1 is using AArch32 then:
>
> - If debug exceptions from Secure EL1 are enabled, then debug exceptions from Secure EL0 are also enabled.
> - Otherwise, debug exceptions from Secure EL0 are enabled only if the value of SDER32_EL3.SUIDEN is `0b1`.

2.29 D17052

In section C6.2.82 (DSB), the encoding shows the CRm field as being restricted with the condition != `0x00`. This is intended to cover encodings `0b0000` and `0b0100`, which are used for the SSBB and PSSBB instructions.

This description is clarified by making SSBB and PSSBB as architectural aliases of DSB.

2.30 D17119

In the following sections:

- F3.1.10 (Advanced SIMD shifts and immediate generation), sub-section 'Advanced SIMD two registers and shift amount'
- F4.1.22 (Advanced SIMD shifts and immediate generation), sub-section 'Advanced SIMD two registers and shift amount'

The following constraints are added to VMOVL:

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

2.31 D17151

In section J1.3.1 (shared/debug), the pseudocode for OSLockStatus() and SoftwareLockStatus() is added:

```
// OSLockStatus()
// ==============
// Returns the state of the OS Lock.
boolean OSLockStatus()
    return (if ELUsingAArch32(EL1) then DBGOSLSR.OSLK else OSLSR_EL1.OSLK) == '1';

// Component
// =========
// Component Types.

enumeration Component {
        Component_PMU,
        Component_Debug,
        Component_CTI
};

// SoftwareLockStatus()
// ====================
// Returns the state of the Software Lock.

boolean SoftwareLockStatus()
    Component component = GetAccessComponent();
    if !HaveSoftwareLock(component) then
        return FALSE;
    case component of
        when Component_Debug
            return EDLSR.SLK == '1';
        when Component_PMU
            return PMLSR.SLK == '1';
        when Component_CTI
            return CTILSR.SLK == '1';
        otherwise
            Unreachable();

// GetAccessComponent()
// ====================
// Returns the accessed component.

Component GetAccessComponent();

// HaveSoftwareLock()
// ==================
// Returns TRUE if Software Lock is implemented.

boolean HaveSoftwareLock(Component component)
    if Havev8p4Debug() then
        return FALSE;
    if HaveDoPD() && component != Component_CTI then
        return FALSE;
    case component of
        when Component_Debug
            return boolean IMPLEMENTATION_DEFINED \"Debug has Software Lock\";
        when Component_PMU
            return boolean IMPLEMENTATION_DEFINED \"PMU has Software Lock\";
        when Component_CTI
            return boolean IMPLEMENTATION_DEFINED \"CTI has Software Lock\";
        otherwise
            Unreachable();
```

2.32 D17170

In section D3.3 (Self-hosted trace timestamps), the fourth item in the indented bulleted list that currently reads:

- EL2 is enabled in the current Security state and is using AArch32.

is changed to read:

- The Effective value of SCR_EL3.{NS,RW} is {1,0}.

An equivalent change is made in section D9.6 (The profiling data).

2.33 D17193

In section D13.2.48 (HCR_EL2, Hypervisor Configuration Register), in the TID2 field, the text that currently states:

> If the value of SCTLR_EL1.UCT is 0, then EL0 reads of CTR_EL0 are **UNDEFINED** and any resulting exception takes precedence over this trap.

is updated to:

> If the value of SCTLR_EL1.UCT is 0, then EL0 reads of CTR_EL0 are trapped to EL1 and the resulting exception takes precedence over this trap.

2.34 D17196

In section D13.3.27 (SDER32_EL2, AArch32 Secure Debug Enable Register), the following lines of the accessibility pseudocode:

```
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED
 \"EL3 trap priority when SDD == '1'\" && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.TDA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return SDER32_EL2;
```

are replaced with:

```
elsif PSTATE.EL == EL1 then
    if !IsSecure() then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if !IsSecure() then
```

```
        UNDEFINED;
    elsif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DE\
FINED \"EL3 trap priority when SDD == '1'\" && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.TDA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return SDER32_EL2;
elsif PSTATE.EL == EL3 then
    if SCR_EL3.EEL2 == '0' then
        UNDEFINED;
    else
        return SDER32_EL2;
```

This fix also results in the accessibility pseudocode line in the EL1 and EL2 accesses of the following registers:

- D13.2.146 (VSTCR_EL2, Virtualization Secure Translation Control Register)
- D13.2.147 (VSTTBR_EL2, Virtualization Secure Translation Table Base Register)
- D13.8.6 (CNTHPS_CTL_EL2, Counter-timer Secure Physical Timer Control register (EL2))
- D13.8.7 (CNTHPS_CVAL_EL2, Counter-timer Secure Physical Timer CompareValue register (EL2))
- D13.8.8 (CNTHPS_TVAL_EL2, Counter-timer Secure Physical Timer TimerValue register (EL2))
- D13.8.12 (CNTHVS_CTL_EL2, Counter-timer Secure Virtual Timer Control register (EL2))
- D13.8.13 (CNTHVS_CVAL_EL2, Counter-timer Secure Virtual Timer CompareValue register (EL2))
- D13.8.14 (CNTHVS_TVAL_EL2, Counter-timer Secure Virtual Timer TimerValue register (EL2))

changing from:

```
if HaveEL(EL3) && SCR_EL3.NS =='1' then
   UNDEFINED;
```

to:

```
if !IsSecure() then
   UNDEFINED;
```

2.35 D17199

In section D13.2.42 (FPEXC32_EL2, Floating-Point Exception Control register), the definition of EN, bit [30] that reads:

> When executing at EL0 using AArch32:
>
> - If EL1 is using AArch64 then behavior is as if the value of FPEXC.EN is 1.

> - If EL2 is using AArch64 and enabled in the current Security state, and the value of HCR_EL2.{RW, TGE} is {1, 1} then behavior is as if the value of FPEXC.EN is 1.
> - If EL2 is using AArch64 and enabled in the current Security state, and the value of HCR_EL2.{RW, TGE} is {0, 1} then it is IMPLEMENTATION DEFINED whether the behavior is:
>   - As if the value of FPEXC.EN is 1.
>   - Determined by the value of FPEXC32_EL2.EN, as described in this field description. However, Arm deprecates using the value of FPEXC32_EL2.EN to determine behavior.

is updated to read:

> When executing at EL0 using AArch32:
> - If EL1 is using AArch64, then the Effective value of FPEXC.EN is 1.
> - If EL2 is using AArch64 and is enabled in the current Security state, HCR_EL2.TGE is 1, and the Effective value of HCR_EL2.RW is 1, then the Effective value of FPEXC.EN is 1. However, Arm deprecates using the value of FPEXC32_EL2.EN to determine behavior.

Similarly, in section G8.2.53 (FPEXC, Floating-Point Exception Control register), the definition of EN, bit [30] that reads:

> When executing at EL0 using AArch32:
> - If EL1 is using AArch64 then behavior is as if the value of FPEXC.EN is 1.
> - If EL2 is using AArch64 and enabled in the current Security state, and the value of HCR_EL2.{RW, TGE} is {1, 1}, then the behavior is as if the value of FPEXC.EN is 1.
> - If EL2 is using AArch64 and enabled in the current Security state, and the value of HCR_EL2.{RW, TGE} is {0, 1}, then it is IMPLEMENTATION DEFINED whether the behavior is:
>   - As if the value of FPEXC.EN is 1.
>   - Determined by the value of FPEXC.EN, as described in this field description. However, Arm deprecates using the value of FPEXC.EN to determine behavior.

is updated to read:

> When executing at EL0 using AArch32:
> - If EL1 is using AArch64, then the Effective value of FPEXC.EN is 1. This includes when EL2 is using AArch64 and enabled in the current Security state, HCR_EL2.TGE is 1, and the Effective value of HCR_EL2.RW is 1.
> - If EL2 is using AArch64 and is enabled in the current Security state, HCR_EL2.TGE is 1, and the Effective value of HCR_EL2.RW is 0, then it is IMPLEMENTATION DEFINED whether the Effective value of FPEXC.EN is 1 or the value written to FPEXC.EN. However, Arm deprecates using the value of FPEXC.EN to determine behavior.

In section J1.2.3 (aarch32/functions), in the function AArch32.CheckAdvSIMDOrFPEnabled(), the code that reads:

```
    if PSTATE.EL == EL0 && (!EL2Enabled() || (!ELUsingAArch32(EL2) && HCR_EL2.TGE ==
 '0')) && !ELUsingAArch32(EL1) then
```

```
        // The PE behaves as if FPEXC.EN is 1

        AArch64.CheckFPEnabled();

        AArch64.CheckFPAdvSIMDEnabled();

    elsif PSTATE.EL == EL0 && EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE ==
 '1' && !ELUsingAArch32(EL1) then
        if fpexc_check && HCR_EL2.RW == '0' then

            fpexc_en = bits(1) IMPLEMENTATION_DEFINED "FPEXC.EN value when TGE==1
 and RW==0";
            if fpexc_en == '0' then UNDEFINED;

        AArch64.CheckFPEnabled();

    else

        …


        // If required, check FPEXC enabled bit.

        if fpexc_check && FPEXC.EN == '0' then UNDEFINED;



        AArch32.CheckFPAdvSIMDTrap(advsimd);    // Also check against HCPTR and CP\
TR_EL3
```

is corrected to:

```
    if PSTATE.EL == EL0 && !ELUsingAArch32(EL1) then
        // When executing at EL0 using AArch32, if EL1 is using AArch64 then the Ef\
fective value of FPEXC.EN is 1.
        // This includes when EL2 is using AArch64 and enabled in the current Secu\
rity state, HCR_EL2.TGE is 1,
        // and the Effective value of HCR_EL2.RW is 1.
        AArch64.CheckFPAdvSIMDEnabled();

    else

        …


        // If required, check the FPEXC enabled bit
        if fpexc_check && PSTATE.EL == EL0 && EL2Enabled() && !ELUsingAArch32(EL2)
 && HCR_EL2.TGE == '1' then
            // When executing at EL0 using AArch32, if EL2 is using AArch64 and en\
abled in the current Security state, HCR_EL2.TGE is 1,
            // and the Effective value of HCR_EL2.RW is 0, then it is IMPLEMENTATION
 DEFINED whether the Effective value of FPEXC.EN is 1
            // or the value of FPEXC32_EL2.EN.
            fpexc_check = boolean IMPLEMENTATION_DEFINED "Use FPEXC32_EL2.EN value
 when {TGE,RW} == {1,0}";

        if fpexc_check && FPEXC.EN == '0' then
            UNDEFINED;


        AArch32.CheckFPAdvSIMDTrap(advsimd);    // Also check against HCPTR and CP\
TR_EL3
```

That is, the following corrections are made to align with the definition of FPEXC.EN:

- The tests that include HCR_EL2 are updated to check that EL2 is enabled in the current security state.
- The '!ELUsingAArch32(EL1)' check in the 'HCR_EL2.TGE == '1'' case is not correct and is removed.
- For the case when HCR_EL2.{RW,TGE} == {0,1}, the IMPLEMENTATION DEFINED choice is between the value in FPEXC32_EL2.EN and '1', not '0' and '1' as previously.

2.36 C17238

In section F3.1.13 (Floating-point data-processing), in subsection 'Floating-point data-processing (two registers)', the following changes are made:

The row that currently appears as:

o1	opc2	size	o3	Instruction page	Feature
0	010	-	0	VCVTB - Half-precision to double-precision variant	-

is replaced by the following rows:

o1	opc2	size	o3	Instruction page	Feature
0	010	10	0	VCVTB - Half-precision to single-precision variant	-
0	010	11	0	VCVTB - Half-precision to double-precision variant	-

and the row that currently appears as:

o1	opc2	size	o3	Instruction page	Feature
0	010	-	1	VCVTT - Half-precision to double-precision variant	-

is replaced by the following rows:

o1	opc2	size	o3	Instruction page	Feature
0	010	10	1	VCVTT - Half-precision to single-precision variant	-
0	010	11	1	VCVTT - Half-precision to double-precision variant	-

In section F4.1.17 (Floating-point data-processing), in subsection 'Floating-point data-processing (two registers)', the following changes are made:

The row that currently appears as:

o1	opc2	size	o3	Instruction page	Feature
0	010	-	0	VCVTB - Half-precision to double-precision variant	-

is replaced by the following rows:

o1	opc2	size	o3	Instruction page	Feature
0	010	10	0	VCVTB - Half-precision to single-precision variant	-
0	010	11	0	VCVTB - Half-precision to double-precision variant	-

and the row that currently appears as:

o1	opc2	size	o3	Instruction page	Feature
0	010	-	1	VCVTT - Half-precision to double-precision variant	-

is replaced by the following rows:

o1	opc2	size	o3	Instruction page	Feature
0	010	10	1	VCVTT - Half-precision to single-precision variant	-
0	010	11	1	VCVTT - Half-precision to double-precision variant	-

2.37 D17292

In a future release, Arm will introduce specific reset domains for the following register specifications:

- Timer reset domain for external Timer registers. These are currently IMPLEMENTATION DEFINED, and referenced generically as reset in the registers.
- AMU reset domain, for AMU registers that are currently indicated as reset on Cold reset in the registers, or IMPLEMENTATION DEFINED in D8.2.3 (Power and reset domains).
- GIC reset domain for GICD, GICR, GITS registers. These are currently generically referenced as reset in the registers.
- MSC reset domain for MPAM registers prefixed with: MPAMCFG, MPAMF, and MSMON.

Additionally, RW fields that currently do not specify a reset domain and reset value will be updated with a specific reset domain and reset value (typically architecturally **UNKNOWN**).

2.38 D17295

In section G8.3.35 (SDER, Secure Debug Enable Register), the EL1 accessibility pseudocode:

```
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T1 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T1 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif (!HaveEL(EL3) || !ELUsingAArch32(EL3)) && SCR_EL3.NS == '0' then
        return SDER;
    else
        UNDEFINED;
```

is updated to:

```
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T1 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T1 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif !IsSecure() then
        UNDEFINED;
    elsif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DE\
FINED \"EL3 trap priority when SDD == '1'\" && !ELUsingAArch32(EL3) && MDCR_EL3.TDA
 == '1' then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.<TDE,TDA> != '00' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        return SDER;
```

2.39 C17310

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

```
// SCTLR[].IESB might be ignored in Debug state.
if !ConstrainUnpredictableBool() then
    sync_errors = FALSE;
```

is clarified to read:

```
// SCTLR[].IESB and/or SCR_EL3.NMEA (if applicable) might be ignored in Debug state.
if !ConstrainUnpredictableBool() then
    sync_errors = FALSE;
```

2.40 D17338

In sections G8.4.3 (PMCEID0, Performance Monitors Common Event Identification register 0), G8.4.4 (PMCEID1, Performance Monitors Common Event Identification register 1), G8.4.5 (PMCEID2, Performance Monitors Common Event Identification register 2) and G8.4.6 (PMCEID3, Performance Monitors Common Event Identification register 3), the following is added to the accessibility pseudocode for access at EL0:

```
elsif EL2Enabled() && !ELUsingAArch32(EL1) && HCR_EL2.<E2H,TGE> != '11' && (!
HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGRTR_EL2.PMCEIDn_EL0 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
```

In sections D13.4.3 (PMCEID0_EL0, Performance Monitors Common Event Identification register 0) and D13.4.4 (PMCEID1_EL0, Performance Monitors Common Event Identification register 1), the following is added to the accessibility pseudocode for access at EL0:

```
elsif EL2Enabled() && HCR_EL2.<E2H,TGE> != '11' && (!HaveEL(EL3) || SCR_EL3.FGTEn ==
 '1') && HDFGRTR_EL2.PMCEIDn_EL0 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
```

and at EL1:

```
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGRTR_EL2.PM\
CEIDn_EL0 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
```

2.41 D17359

In section B2.7.2 (Device memory) in the subsection 'Early Write Acknowledgement', the text that reads:

> For memory system endpoints where the system architecture in which the PE is operating requires that acknowledgement of a write comes from the endpoint, assigning the No Early Write Acknowledgement attribute to a Device memory location guarantees that:
>
> - Only the endpoint of the write access returns a write acknowledgement of the access.
> - No earlier point in the memory system returns a write acknowledgement.

is clarified to read:

> If the No Early Write Acknowledgement attribute is assigned for a Device memory location:
>
> - For memory system endpoints where the system architecture in which the PE is operating requires that acknowledgement of a write comes from the endpoint, it is guaranteed that:
>   - Only the endpoint of the write access returns a write acknowledgement of the access.
>   - No earlier point in the memory system returns a write acknowledgement.
> - For memory system endpoints where the system architecture in which the PE is operating does not require that acknowledgement of a write comes from the endpoint, the acknowledgement of a write is not required to come from the endpoint.
>
> Note: A write with the No Early Write Acknowledgement attribute assigned for a Device memory location is not expected to generate an abort in any situation where the equivalent write to the same location without the No Early Write Acknowledgement attribute assigned does not generate an abort.

The same change is made in section E2.8.2 (Device memory).

2.42 D17387

In section H9.2.42 (EDSCR, External Debug Status and Control Register), the following text is added in the definition of INTdis, bits [23:22] when FEAT_Debugv8p4 is implemented:

> When FEAT_Debugv8p4 is implemented, bit[23] of the register is RES0.

and references to 'ExternalDebugEnabled' and 'ExternalSecureDebugEnabled' are corrected to 'ExternalInvasiveDebugEnabled' and 'ExternalSecureInvasiveDebugEnabled', respectively.

The following text is added when FEAT_Debugv8p4 is not implemented:

> Support for the values `0b01` and `0b10` is IMPLEMENTATION DEFINED. If these values are not supported, they are reserved. If programmed with a reserved value, the PE behaves as if INTdis has been programmed with a defined value, other than for a direct read of EDSCR, and the value returned by a read of EDSCR.INTdis is **UNKNOWN**.

and the following condition is added to the value definitions:

> This field is ignored by the PE and treated as zero when ExternalInvasiveDebugEnabled() == FALSE.

2.43 D17392

In the D7.5.3 (Prohibiting event and cycle counting) section, the following software usage information is added:

> If a direct read of PMOVSCLR_EL0 returns a non-zero value for a subset of the overflow flags, which means an event counter <n> should not count, then a sequence of direct reads of PMEVCNTR<n>_EL0 ordered after the read of PMOVSCLR_EL0 and before the PMSOVSCLR_EL0 flags are cleared to zero, will return the same value for each read, because the event counter has stopped counting.
>
> Note: Direct reads of System registers require explicit synchronization for following direct reads of other System registers to be ordered after the first direct read.

2.44 C17393

In sections C5.3.23 (DC IGDVAC, Invalidate of Data and Allocation Tags by VA to PoC), C5.3.25 (DC IGVAC, Invalidate of Allocation Tags by VA to PoC), and C5.3.27 (DC IVAC, Data or unified Cache line Invalidate by VA to PoC), the following EL1 accessibility pseudocode is removed:

```
elsif EL2Enabled() && HCR_EL2.<DC,VM> != '00' then
    DC_CIGDVAC(X[t]);
```

Similarly, the following EL1 accessibility pseudocode is removed from sections C5.3.22 (DC IGDSW, Invalidate of Data and Allocation Tags by Set/Way), C5.3.24 (DC IGSW, Invalidate of Allocation Tags by Set/Way), and C5.3.26 (DC ISW, Data or unified Cache line Invalidate by Set/Way):

```
elsif EL2Enabled() && HCR_EL2.SWIO == '1' then
    DC_CIGDSW(X[t]);
elsif EL2Enabled() && HCR_EL2.<DC,VM> != '00' then
    DC_CIGDSW(X[t]);
```

In section G8.2.43 (DCIMVAC, Data Cache line Invalidate by VA to PoC) the following EL1 accessibility pseudocode is removed:

```
elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<DC,VM> != '00' then
    DCCIMVAC(R[t]);
elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.<DC,VM> != '00' then
    DCCIMVAC(R[t]);
```

Similarly, the following EL1 accessibility pseudocode in section G8.2.44 (DCISW, Data Cache line Invalidate by Set/Way) is removed:

```
elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.SWIO == '1' then
    DCCISW(R[t]);
elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<DC,VM> != '00' then
    DCCISW(R[t]);
elsif EL2Enabled() && EL2UsingAArch32(EL2) && HCR_EL2.SWIO == '1' then
    DCCISW(R[t]);
elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.<DC,VM> != '00' then
    DCCISW(R[t]);
```

2.45 D17396

In section D1.14.3 (EL2 configurable controls), the text that reads:

> These controls are ignored if EL2 is not enabled in the current Security state.

is corrected to read:

> If Secure EL2 is implemented and enabled, configurable instruction controls available at EL2 apply in Secure state. If Secure EL2 is not implemented or not enabled, the configurable instruction controls available at EL2 are ignored in Secure state.

2.46 D17401

In section H3.2.4 (Detailed Halting Step state machine behavior), in subsection 'Entering the active-not-pending state', the text that reads:

> The PE enters the active-not-pending state:

- By exiting Debug state with EDECR.SS == 1.

is corrected to read:

> The PE enters the active-not-pending state:
>
> - By exiting Debug state to a state where halting is allowed with EDECR.SS == 1.

Within the same section, in subsection 'PE behavior in the active-not-pending state', the text that reads:

> When the PE is in the active-not-pending state it does one of the following:
>
> - It executes one instruction and does one of the following:
>   - Completes it without generating a synchronous exception.
>   - Generates a synchronous exception.
>   - Generates a debug event that causes entry to Debug state.

is clarified to read:

> When the PE is in the active-not-pending state it does one of the following:
>
> - It executes one instruction and does one of the following:
>   - Completes it without taking a synchronous exception.
>   - Takes a synchronous exception generated by the instruction.
>   - Generates a debug event that causes entry to Debug state.

2.47 D17405

In section A2.7.1 (Architectural features added by Armv8.4), the text that reads:

> FEAT_TLBIOS provides TLBI maintenance instructions that extend to the Outer Shareable domain and TLBI invalidation instructions that apply to a range of input addresses.

is changed to read:

> FEAT_TLBIOS provides TLBI maintenance instructions that extend to the Outer Shareable domain.

And the text that reads:

> FEAT_TLBIRANGE provides TLBI maintenance instructions that extend to the Outer Shareable domain and TLBI invalidation instructions that apply to a range of input addresses.

is changed to read:

> FEAT_TLBIRANGE provides TLBI maintenance instructions that apply to a range of input addresses. FEAT_TLBIRANGE being implemented implies that FEAT_TLBIOS is implemented.

2.48 R17415

In section I5.8.23 (ERR<n>CTLR, Error Record Control Register, n = 0 - 65534), in the descriptions of the WUE and UE fields, the following text is added:

> It is **IMPLEMENTATION DEFINED** whether an uncorrected error that is deferred but not deferred to the Requester will signal an in-band error response.

The same addition is made to the description of the UE field in section I5.8.24 (ERR<n>FR, Error Record Feature Register, n = 0 - 65534).

Additionally, in section I5.8.32 (ERR<n>STATUS, Error Record Primary Status Register, n = 0 - 65534), in the description of the ER field, the text that currently reads:

> It is **IMPLEMENTATION DEFINED** whether this bit can be set to `0b1` by a Deferred error.

is replaced by the following text:

> It is **IMPLEMENTATION DEFINED** whether an uncorrected error that is deferred but not deferred to the Requester will signal an in-band error response causing this bit to be set to `0b1`.

2.49 D17416

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

```
    sync_errors = HaveIESB() && SCTLR[].IESB == '1';
    if HaveDoubleFaultExt() then
        sync_errors = sync_errors || (SCR_EL3.EA == '1' && SCR_EL3.NMEA == '1' &&
  PSTATE.EL == EL3);
        if sync_errors then
            SynchronizeErrors();
            iesb_req = TRUE;
            TakeUnmaskedPhysicalSErrorInterrupts(iesb_req);
```

is corrected to read:

```
    if HaveIESB() then
        sync_errors = SCTLR[].IESB == '1';
        if HaveDoubleFaultExt() then
            sync_errors = sync_errors || (SCR_EL3.<EA,NMEA> == '11' && PSTATE.EL ==
  EL3);
        if sync_errors then
            SynchronizeErrors();
            iesb_req = TRUE;
```

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

```
    sync_errors = HaveIESB() && SCTLR[target_el].IESB == '1';
    if HaveDoubleFaultExt() then
```

```
        sync_errors = sync_errors || (SCR_EL3.EA == '1' && SCR_EL3.NMEA == '1' &&
  target_el == EL3);
     if sync_errors && InsertIESBBeforeException(target_el) then
        SynchronizeErrors();
        iesb_req = FALSE;
        sync_errors = FALSE;
        TakeUnmaskedPhysicalSErrorInterrupts(iesb_req);
```

is corrected to read:

```
     if HaveIESB() then
        sync_errors = SCTLR[target_el].IESB == '1';
        if HaveDoubleFaultExt() then
            sync_errors = sync_errors || (SCR_EL3.<EA,NMEA> == '11' && target_el ==
  EL3);
        if sync_errors && InsertIESBBeforeException(target_el) then
            SynchronizeErrors();
            iesb_req = FALSE;
            sync_errors = FALSE;
            TakeUnmaskedPhysicalSErrorInterrupts(iesb_req);
     else
        sync_errors = FALSE;
```

2.50 D17417

In section A2.7.1 (Architectural features added by Armv8.4), in the subsection titled 'FEAT_RASv1p1, RAS Extension v1.1', the text that currently reads:

> FEAT_RASv1p1 implements RAS System Architecture v1.1 and adds support for:
>
> - FEAT_DoubleFault.
> - Simplifications to ERR<n>STATUS.
> - Additional ERR<n>MISC<m> registers.
> - The OPTIONAL RAS Common Fault Injection Model Extension.

is corrected to read:

> FEAT_RASv1p1 implements RAS System Architecture v1.1 and adds support for:
>
> - Simplifications to ERR<n>STATUS.
> - Additional ERR<n>MISC<m> registers.
> - The OPTIONAL RAS Common Fault Injection Model Extension.

In section D13.2.67 (ID_AA64PFR0_EL1, AArch64 Processor Feature Register 0), in the RAS, bits [31:28] field, the text that currently reads:

> `0b0010` FEAT_RASv1p1 present. As `0b0001`, and adds support for:
>
> - If EL3 is implemented, FEAT_DoubleFault.

is corrected to read:

> `0b0010` FEAT_RASv1p1 and, if EL3 is implemented, FEAT_DoubleFault present. As `0b0001`, and adds support for:
>
> - If EL3 is implemented, FEAT_DoubleFault.

2.51 R17420

In section D13.2.118 (SCTLR_EL3, System Control Register (EL3)), in the IESB, bit [21] field, the text that currently reads:

> When the PE is in Debug state, the effect of this field is **CONSTRAINED UNPREDICTABLE**, and its Effective value might be 0 or 1 regardless of the value of the field.

is corrected to read:

> When the PE is in Debug state, the effect of this field is **CONSTRAINED UNPREDICTABLE**, and its Effective value might be 0 or 1 regardless of the value of the field and, if implemented, SCR_EL3.NMEA.

Within the same section, the text that currently reads:

> When FEAT_DoubleFault is implemented, and the Effective value of SCR_EL3.NMEA is 1, this field is ignored and its Effective value is 1.

is corrected to read:

> When FEAT_DoubleFault is implemented, the PE is in Non-debug state, and the Effective value of SCR_EL3.NMEA is 1, this field is ignored and its Effective value is 1.

2.52 D17422

In section D8.3.1 (Architected event counters), the AMU CPU_CYCLES event description text:

> `0x0011`, CPU_CYCLES, Processor frequency cycles This event is defined identically to CPU_CYCLES in the PMUv3 architecture. When the PE is in WFI or WFE, this counter does not increment. When in a multithreaded implementation, regardless of which PE is currently active, this counter continues to count for all PEs not in WFI or WFE. This event is counted by AMEVCNTR0<n>, where n is 0.

is changed to read:

> `0x0011`, CPU_CYCLES, Processor frequency cycles The counter increments on every cycle when the PE is not in WFI or WFE state. When the PE is in WFI or WFE state, this counter does not increment. This event is counted by AMEVCNTR0<n>, where n is 0.

And the AMU CNT_CYCLES event description is changed from:

> `0x4004`, CNT_CYCLES, Constant frequency cycles The constant frequency cycles counter increments at a constant frequency equal to the rate of increment of the System counter, CNTPCT_EL0. When the PE is in WFI or WFE, this counter does not increment. When in a multithreaded implementation, regardless of which PE is currently active, this counter continues to count for all PEs not in WFI or WFE. This event is counted by AMEVCNTR0<n>, where n is 1.

to read:

> `0x4004`, CNT_CYCLES, Constant frequency cycles The counter increments at a constant frequency when the PE is not in WFI or WFE state, equal to the rate of increment of the System counter, CNTPCT_EL0. When the PE is in WFI or WFE state, this counter does not increment. This event is counted by AMEVCNTR0<n>, where n is 1.

In section D7.10.3 (Common event numbers), the following text in the `0x0011`, CPU_CYCLES, Cycle event description:

> All counters are subject to changes in clock frequency, including when a WFI or WFE instruction stops the clock. This means that it is **CONSTRAINED UNPREDICTABLE** whether or not CPU_CYCLES continues to increment when the clocks are stopped by WFI and WFE instructions.

is changed to read:

> All counters are subject to changes in clock frequency. It is **CONSTRAINED UNPREDICTABLE** whether or not CPU_CYCLES continues to increment when the PE is in WFI or WFE state.

And the following text:

> In a multithreaded implementation, CPU_CYCLES counts each cycle for the processor for which this PE thread was active and could issue an instruction. For more information, see Cycle event counting on multithreaded implementations.

is changed to read:

> In a multithreaded implementation, CPU_CYCLES counts each cycle for the processor for which this PE thread is active and can issue an instruction. For more information, see Cycle event counting.

Within the same section, the PMU CNT_CYCLES event description is changed from:

> `0x4004`, CNT_CYCLES, Constant frequency cycles The counter is defined identically to CNT_CYCLES in the AMUv1 architecture.

to read:

> `0x4004`, CNT_CYCLES, Constant frequency cycles The counter increments at a constant frequency equal to the rate of increment of the System counter, CNTPCT_EL0. It is **CONSTRAINED UNPREDICTABLE** whether or not CNT_CYCLES continues to increment when the PE is in WFI or WFE state. In a multithreaded implementation, CNT_CYCLES counts when this PE thread is active and can issue an instruction. For more information, see Cycle event counting.

Also, the PMU STALL_BACKEND_MEM event description is changed from:

> `0x4005`, STALL_BACKEND_MEM, Memory stall cycles The counter is defined identically to STALL_BACKEND_MEM in the AMUv1 architecture.

to read:

> `0x4005`, STALL_BACKEND_MEM, Memory stall cycles The counter counts each cycle counted by STALL_BACKEND where there is a cache miss in the last level of cache within the PE clock domain. It is IMPLEMENTATION DEFINED whether the counter counts backend stall cycles when a non-cacheable access is in progress.

Section D7.1.5 (Interaction with power saving operations) is deleted, and section D7.10.4 (Cycle event counting on multithreaded implementations) is changed to the following:

> D7.10.4 Cycle event counting
>
> The CPU_CYCLES event and the cycle counter, PMCCNTR, count cycles. The duration of a cycle is subject to any changes in clock frequency, including clock stopping caused by the WFI and WFE instructions. It is implementation specific whether CPU_CYCLES and PMCCNTR count when the PE is in WFI or WFE state, even if the clocks are not stopped.
>
> In addition, events such as STALL, STALL_FRONTEND and STALL_BACKEND that are defined to only count cycles that are counted by the CPU_CYCLES event have the same limitation.
>
> Multithreaded implementations
>
> Multithreaded implementations can have various forms, some examples of these are:
>
> - Simultaneous Multithreading (SMT), where every PE thread is active on every Processor cycle.
> - Fine-grained Multithreading (FGMT), also known as a Barrel processor, where one PE thread is active on each Processor cycle, and this changes regularly.
> - Switch on Event Multithreading (SoEMT), also known as Coarse-grained Multithreading (CGMT), where high latency events cause the processor to switch the active PE thread.
>
> In the above examples, active means that the PE might execute the instructions. A PE can be active but not executing instructions when no instruction is available or because of limited execution resources.
>
> It is implementation specific whether a thread is active when the thread is in WFE or WFI state. This applies for all forms of multithreaded implementation.
>
> When the PMU implementation supports multithreading, and the Effective value of PMEVTYPER<n>_EL0.MT bit is 0, the CPU_CYCLES event does not count Processor cycles on which the thread was not active. For the example multithreaded implementations, this means that, if the event counter is enabled, event counting is not prohibited, and the thread is not in WFE or WFI state:
>
> - For an SMT implementation, the CPU_CYCLES event counts every Processor cycle.
> - For a particular FGMT implementation, that alternates between two threads on each Processor cycle, the CPU_CYCLES event counts every other Processor cycle.

- For a particular SoEMT implementation, that is waiting for a long latency operation, the CPU_CYCLES event does not count Processor cycles, as the PE thread is not active.

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

Events that only count cycles that are counted by the CPU_CYCLES event have the same limitation. For example, in an SMT implementation, if a PE thread cannot issue an instruction because of contention with other PE threads, these are counted as STALL_BACKEND cycles. If the Effective value of PMEVTYPER<n>_EL0.MT bit is 1, the PE only counts cycles on which no operation is issued from any thread.

Note: The cycle counter, PMCCNTR, is not affected by whether the thread is active or inactive. When enabled, PMCCNTR counts every Processor cycle.

See Multithreaded implementations, MDCR_EL3.MTPME, SDCR.MTPME, MDCR_EL2.MTPME, and HDCR.MTPME for more information about when the Effective value of PMEVTYPER<n>_EL0.MT is 0.

2.53 D17423

In section D5.10.2 (TLB maintenance instructions), in the subsection 'Scope of the A64 TLB maintenance instructions', for each of the entries VA, VAL, VAA, VAAL, the bullet list that currently reads:

- The Security state specified by SCR_EL3.NS and SCR_EL3.EEL2.
- For the Secure or Non-secure EL1&0, when EL2 is enabled, translation regime, the current VMID.

is changed to read:

- The Security state specified by SCR_EL3.NS and SCR_EL3.EEL2.
- The current VMID, for the Secure or Non-secure EL1&0 translation regime, when EL2 is enabled.
- The current translation regime. For EL2&0 translation regimes, this is determined by HCR_EL2.E2H.

2.54 D17425

In section F5.1.229 (STR (immediate)), T4 encoding, the following text is removed:

If wback && n == 15, then one of the following behaviors must occur:

- The instruction is **UNDEFINED**.
- The instruction executes as NOP.

- The instruction executes without writeback of the base address.
- The instruction uses the addressing mode described in the equivalent immediate offset instruction.

The same text is also removed from section F5.1.231 (STRB (immediate)), T3 encoding.

2.55 E17427

In section J1 (Armv8 pseudocode), in order to better represent the handling of External aborts, _Mem[] getters and setters are replaced with physical memory read and write functions PhysMemRead() and PhysMemWrite(). These functions return a new structure PhysMemRetStatus, which may contain External abort related information. For MTE support, getters and setters _MemTag[] are similarly replaced with PhysMemTagRead() and PhysMemTagWrite().

2.56 D17428

In section J1.1.1 (aarch64/debug) the pseudocode function AArch64.BreakpointValueMatch() that reads:

```
elsif match_cid2 then
    BXVR_match = ((HaveVirtHostExt() || HaveV82Debug()) && EL2Enabled() && DBGB\
VR_EL1[n]<63:32> == CONTEXTIDR_EL2);
```

is corrected to read:

```
elsif match_cid2 then
    BXVR_match = (PSTATE.EL != EL3 && (HaveVirtHostExt() || HaveV82Debug()) &&
 EL2Enabled() && DBGBVR_EL1[n]<63:32> == CONTEXTIDR_EL2);
```

2.57 D17428

In section J1.1.1 (aarch64/debug) the pseudocode function AArch64.BreakpointValueMatch() that reads:

```
elsif match_cid2 then
    BXVR_match = ((HaveVirtHostExt() || HaveV82Debug()) && EL2Enabled() && DBGB\
VR_EL1[n]<63:32> == CONTEXTIDR_EL2);
```

is corrected to read:

```
elsif match_cid2 then
    BXVR_match = (PSTATE.EL != EL3 && (HaveVirtHostExt() || HaveV82Debug()) &&
 EL2Enabled() && DBGBVR_EL1[n]<63:32> == CONTEXTIDR_EL2);
```

2.58 D17433

In section D10.1.3 (Byte Order), the text that currently reads:

> Header bytes and payload bytes are written in ascending address order. Within a payload value, values are written in little-endian byte order.

is corrected to read:

> This chapter describes header bytes and payload bytes in ascending memory address order. Within a payload value, values are in little-endian byte order.

Additionally, the following Note is removed:

> Note: This means that if the memory type accessed is non-Gathering Device, the architecture does not require a specific access granule size at the end device.

2.59 R17435

In section H9.2.25 (EDECCR, External Debug Exception Catch Control Register), the text in the description for each field that currently reads:

> A value of the (NSR, SR, NSE, SE) field that enables an Exception Catch debug event for an Exception level that is not implemented is reserved. If the (NSR, SR, NSE, SE) field is programmed with a reserved value then:
>
> - The PE behaves as if it is programmed with a defined value, other than for a read of EDECCR.
> - The value returned for (NSR, SR, NSE, SE) by a read of EDECCR is **UNKNOWN**.

is changed to the following:

in the NSR field:

> If EL<n> is not implemented then NSR<n> is RES0.

in the SR field:

> If FEAT_SEL2 is not implemented then SR<2> is RES0. If EL<n> is not implemented then SR<n> is RES0.

in the NSE fields:

> NSE<0> is RES0. If EL<n> is not implemented then NSE<n> is RES0.

in the SE fields:

> SE<0> is RES0. If FEAT_SEL2 is not implemented then SE<2> is RES0. If EL<n> is not implemented then SE<n> is RES0.

2.60 C17438

In section D13.2.52 (HFGITR_EL2, Hypervisor Fine-Grained Instruction Trap Register), in the DCZVA field, bit [11], the following Note is added to the specification:

> Note: Unlike HCR_EL2.TDZ, this field does not have an impact on DCZID_EL0.DZP.

2.61 D17441

In section D13.2.25 (CCSIDR2_EL1, Current Cache Size ID Register 2), the text that currently reads:

> In an AArch64 only implementation, it is IMPLEMENTATION DEFINED whether reading this register gives an **UNKNOWN** value or is UNDEFINED.

is relaxed to read:

> In an implementation which does not support AArch32 at EL1, it is IMPLEMENTATION DEFINED whether reading this register gives an **UNKNOWN** value or is UNDEFINED.

Similarly, in section G8.2.25 (CCSIDR2, Current Cache Size ID Register 2), the text that reads:

> This register is present only when AArch32 is supported at any Exception level and FEAT_CCIDX is implemented. Otherwise, direct accesses to CCSIDR2 are UNDEFINED.

is relaxed to read:

> This register is present only when AArch32 is supported at EL1 and FEAT_CCIDX is implemented. Otherwise, direct accesses to CCSIDR2 are UNDEFINED.

2.62 D17445

In section D7.10.3 (Common Event Numbers), in the subsection 'Common microarchitectural events', some event descriptions are replaced.

The description of event `0x80DB`, 'LDST_FIXED_BYTES_SPEC' is replaced with:

> The counter counts bytes speculatively read or written due to all non-SVE load, store and atomic operations, all SVE non-vector load and store operations, and SVE replicating LD1R and LD1RQ instructions.

> For each instruction, the counter is incremented by the number of bytes transferred per register multiplied by the number of registers transferred multiplied by the number of transfers made per register. For example, the counter counts bytes as follows:
>
> - Non-SVE load and store of a single register instructions increment the counter by (MSIZE ÷ 8).
> - Non-SVE load and store of a pair of registers instructions increment the counter by 2 × (MSIZE ÷ 8).
> - AArch32 load and store multiple registers instructions increment the counter by the number of registers transferred multiplied by (MSIZE ÷ 8).
> - Atomic store instructions increment the counter by (MSIZE ÷ 8). These are instructions that atomically update a value in memory without returning a value to a register.
> - Atomic load, compare and swap of a single register, and swap instructions increment the counter by 2 × (MSIZE ÷ 8). Atomic load instructions are instructions that atomically update a value in memory, returning a value to a register.
> - Compare and swap of a pair of registers increment the counter by 4 × (MSIZE ÷ 8).
> - SVE and Advanced SIMD LD1R instructions increment the counter by (MSIZE ÷ 8).
> - SVE LD1RQ instructions increment the counter by 16.
> - Advanced SIMD LD[1-4] and ST[1-4] instructions increment the counter by the number of registers transferred multiplied by the number of bytes being transferred per register.
> - DC ZVA and DC GZVA instructions increment by the counter by 2^(DCZID_EL0.BS).

The description of event `0x80DD`, 'LD_FIXED_BYTES_SPEC' is replaced with:

> The counter counts bytes speculatively read due to all non-SVE load and atomic operations, all SVE non-vector load operations, and SVE replicating LD1R and LD1RQ instructions. For each instruction, the counter is incremented by the number of bytes transferred per register multiplied by the number of registers transferred. That is, the counter is incremented by:
>
> - Half the value that the LDST_FIXED_BYTES_SPEC event counts if the operation is a load atomic, compare and swap, or compare operation.
> - The same as for LDST_FIXED_BYTES_SPEC if the operation is any other load operation.

The description of event `0x80DF`, 'ST_FIXED_BYTES_SPEC' is replaced with:

> The counter counts bytes written due to all non-SVE store and atomic operations, and all SVE non-vector store operations. For each instruction, the counter is incremented by the number of bytes transferred per register multiplied by the number of registers transferred. That is, the counter is incremented by:
>
> - Half the value that the LDST_FIXED_BYTES_SPEC event counts if the operation is a compare and swap or compare operation.
> - The same as for LDST_FIXED_BYTES_SPEC if the operation is any other store operation, including an atomic store operation.

2.63 D17464

In section C5.6.1 (CFP RCTX, Control Flow Prediction Restriction by Context), the paragraph that reads:

> When this instruction is complete and synchronized, control flow prediction does not permit later speculative execution within the target execution context to be observable through side channels.

is replaced by the following paragraph:

> Control flow predictions determined by the actions of code in the target execution context(s) appearing in program order before the instruction cannot exploitatively control speculative execution occurring after the instruction is complete and synchronized.

The equivalent edits are made to the following sections:

- C5.6.2 (CPP RCTX, Cache Prefetch Prediction Restriction by Context).
- C5.6.3 (DVP RCTX, Data Value Prediction Restriction by Context).
- C6.2.51 (CFP).
- C6.2.65 (CPP).
- C6.2.83 (DVP).
- 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).

2.64 C17466

In section D9.6.3 (Additional information for each profiled memory access operation), the text that currently reads:

> For each of the Last level cache and another socket indicators, it is **IMPLEMENTATION DEFINED** whether this information is present only for load accesses, only for store accesses, for neither, or for both.
>
> Note: A store might be marked as not accessing a cache or another socket because it microarchitecturally finished before doing so. For example, the write was held in a write buffer. This behavior is **IMPLEMENTATION DEFINED**, and such events must be interpreted with care.

is changed to read:

> For each of the cache and another socket indicators, it is **IMPLEMENTATION DEFINED** and might be UNPREDICTABLE whether this information is present for store accesses. The Last level cache and another socket indicators are optional and might not be present.

> Note: A store might be marked as not accessing a cache or another socket because it microarchitecturally-finished execution before doing so. For example, the write was placed into a write buffer. This behavior is **IMPLEMENTATION DEFINED** and might change from time to time, and such events must be interpreted with care.

In section D10.2.6 (Events packet), the following clarification is added to each of the E[2] (Level 1 Data cache access) and E[3] (Level 1 Data cache refill) events:

> It is **IMPLEMENTATION DEFINED** and might be UNPREDICTABLE whether a store can finish execution before this event is generated, meaning this event is never recorded for stores.

Within the same section, the text in E[2] that reads:

> If PMUv3 is implemented this Event is required to be implemented consistently with L1D_CACHE.

is changed to the following, with a similar change made in E[3]:

> If PMUv3 is implemented this event is required to be implemented consistently with L1D_CACHE or L1D_CACHE_RD.

Additionally, the following clarification is added to each of the E[8] (Last Level cache access), E[9] (Last Level cache miss), and E[10] (Remote access) events:

> This event is optional. When this event is implemented, it is further **IMPLEMENTATION DEFINED** and might be UNPREDICTABLE whether a store can finish execution before this event is generated, meaning this event is never recorded for stores.

And the text in E[8] that reads:

> If PMUv3 is implemented this Event is required to be implemented consistently with LL_CACHE.

is changed to the following, with similar changes made in E[9] and E[10]:

> If this event and PMUv3 are both implemented this event is required to be implemented consistently with LL_CACHE or LL_CACHE_RD.

2.65 D17478

In section D4.4.13 (Execution and data prediction restriction System instructions), the text that reads:

> When FEAT_SPECRES is implemented, the System instructions listed in A64 System instructions for prediction restriction on page C5-845 prevent predictions based on information gathered from earlier execution within a particular execution context from affecting the later speculative execution within that context, to the extent that the speculative execution is observable through side-channels.

> The prediction restriction System instructions being used by a particular execution context apply to:

is clarified to read:

> When FEAT_SPECRES is implemented, the System instructions listed in A64 System instructions for prediction restriction on page C5-845 prevent predictions based on information gathered from earlier execution within a particular execution context, termed for these instructions as a CTX, from affecting the later Speculative execution within that CTX, to the extent that the speculative execution is observable through side-channels.
>
> The prediction restriction System instructions being used by a particular CTX apply to:

Within the same section, the text that reads:

> For these System instructions, the execution context is defined by:

is clarified to read:

> For these System instructions, the CTX is defined by:

The equivalent changes are made to section G4.4.8 (Execution and data prediction restriction System instructions).

2.66 D17492

In section I5.5.7 (AMCNTENCLR0, Activity Monitors Count Enable Clear Register 0), the field 'P<n>, bit [n], for n = 31 to 0' that currently reads:

> P<n>, bit [n], for n = 31 to 0 Activity monitor event counter disable bit for AMEVCNTR0<n>. Bits [31:N] are RAZ/WI. N is the value in AMCGCR.CG0NC.

is corrected to read:

> P<n>, bit [n], for n = 15 to 0 Activity monitor event counter disable bit for AMEVCNTR0<n>. Bits [15:N] are RAZ/WI, where N is the value in AMCGCR.CG0NC.

A new **RES0** field is added for bits [31:16].

The equivalent changes are made to the following sections:

- I5.5.8 (AMCNTENCLR1, Activity Monitors Count Enable Clear Register 1),
- I5.5.9 (AMCNTENSET0, Activity Monitors Count Enable Set Register 0),
- I5.5.10 (AMCNTENSET1, Activity Monitors Count Enable Set Register 1).

2.67 D17527

In section J1.2.4 (aarch32/translation), the pseudocode for AArch32.FirstStageTranslate() that reads:

```
    if PSTATE.EL == EL2 then
        s1_enabled = HSCTLR.M == '1';
    elsif EL2Enabled() then
        tge = (if ELUsingAArch32(EL2) then HCR.TGE else HCR_EL2.TGE);
        dc = (if ELUsingAArch32(EL2) then HCR.DC else HCR_EL2.DC);
        s1_enabled = tge == '0' && dc == '0' && SCTLR.M == '1';
    else
        s1_enabled = SCTLR.M == '1';
```

is replaced with:

```
    el = S1TranslationRegime();
    if el == EL2 then
        s1_enabled = HSCTLR.M == '1';
    elsif el == EL1 && EL2Enabled() then
        tge = (if ELUsingAArch32(EL2) then HCR.TGE else HCR_EL2.TGE);
        dc = (if ELUsingAArch32(EL2) then HCR.DC else HCR_EL2.DC);
        s1_enabled = tge == '0' && dc == '0' && SCTLR.M == '1';
    else
        s1_enabled = SCTLR.M == '1';
```

2.68 D17534

In section G8.2.80 (ICIALLUIS, Instruction Cache Invalidate All to PoU, Inner Shareable), the following lines of the MCR register write pseudocode:

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T7 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T7 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TPU == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TICAB == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TOCU == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TPU == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR2.TICAB == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR2.TOCU == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        ICIALLUIS();
```

are replaced with:

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T7 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T7 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TPU == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TICAB == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TPU == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR2.TICAB == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        ICIALLUIS();
```

2.69 D17539

In D13.4.12 (PMMIR_EL1, Performance Monitors Machine Identification Register) and G8.3.33 (PMMIR, Performance Monitors Machine Identification Register), the following text in the BUS_WIDTH and BUS_SLOTS fields is removed:

> From Armv8.7:

2.70 D17541

In D7.10.3 (Common event numbers), subsection 'Common microarchitectural events', each instance of 'the access' is changed to 'the access that caused the walk', in the final list of the following event number descriptions:

- `0x8136`, 'DTLB_STEP, Data TLB translation table walk, step',
- `0x8137`, 'ITLB_STEP, Instruction TLB translation table walk, step'.

2.71 D17557

In section D7.10.3 (Common event numbers), in the subsection 'Common microarchitectural events', the event definition '0x80CB, LDST_FIXED_OPS_SPEC, Non-scalable load and store element Operations speculatively executed' that reads:

> The counter counts speculatively executed memory read and write operations as follows:
> - Loading or storing a single scalar register increments the counter by 1.
> - Loading or storing a pair of scalar registers increments the counter by 2.

- An atomic store instruction increments the counter by 1.
- An atomic load instruction increments the counter by 2.
- SVE and Advanced SIMD LD1R instructions increment the counter by 1.
- SVE LD1RQ instructions increment the counter by (128 ÷ CSIZE).
- Advanced SIMD LD[1-4] and ST[1-4] instructions increment the counter by the number of elements transferred per vector multiplied by the number of transferred registers.

is clarified to read:

The counter counts speculatively executed Memory-read and Memory-write operations due to all non-SVE load, store and atomic operations, all SVE non-vector load and store operations, and SVE replicating LD1R and LD1RQ instructions.

For each instruction, the counter is incremented by the number of operations specified by the instruction. For example, the counter counts operations as follows:

- Non-SVE load and store of a single register instructions increment the counter by 1. This includes loads and stores of Sx, Dx, and Qx SIMD&FP registers.
- Non-SVE load and store of a pair of registers instructions increment the counter by 2. This includes loads and stores of pairs of Sx, Dx, and Qx SIMD&FP registers.
- AArch32 load and store multiple registers instructions increment the counter by the number of registers transferred.
- Atomic store instructions increment the counter by 1. These are instructions that atomically update a value in memory without returning a value to a register.
- Atomic load, compare and swap of a single register, and swap instructions increment the counter by 2. Atomic load instructions are instructions that atomically update a value in memory, returning a value to a register.
- Compare and swap of a pair of registers increment the counter by 4.
- SVE and Advanced SIMD LD1R instructions increment the counter by 1.
- SVE LD1RQ instructions increment the counter by (128 ÷ CSIZE).
- Advanced SIMD LD[1-4] and ST[1-4] instructions increment the counter by the number of elements transferred per vector multiplied by the number of transferred registers.
- DC ZVA and DC GZVA instructions increment by the counter by an **IMPLEMENTATION DEFINED** amount.

In the same section, the event definition '0x80CD, LD_FIXED_OPS_SPEC, Non-scalable load element Operations speculatively executed' is changed to read:

The counter counts speculatively executed Memory-read operations due to all non-SVE load and atomic operations, all SVE non-vector load operations, and SVE replicating LD1R and LD1RQ instructions.

For each instruction, the counter is incremented by the number of operations specified by the instruction. That is, the counter is incremented by:

- Half the value that the LDST_FIXED_OPS_SPEC event counts if the operation is a load atomic, compare and swap, or compare operation.
- The same as for LDST_FIXED_OPS_SPEC if the operation is any other load operation.

and the event definition '0x80CF, ST_FIXED_OPS_SPEC, Non-scalable store element Operations speculatively executed' is changed to read:

> The counter counts speculatively executed Memory-write operations due to all non-SVE store and atomic operations, and all SVE non-vector store operations.
>
> For each instruction, the counter is incremented by the number of operations specified by the instruction. That is, the counter is incremented by:
>
> - Half the value that the LDST_FIXED_OPS_SPEC event counts if the operation is a compare and swap or compare operation.
> - The same as for LDST_FIXED_OPS_SPEC if the operation is any other store operation, including an atomic store operation.

2.72 D17558

In section B2.3.10 (Memory barriers), in the subsection 'Data Synchronization Barrier (DSB)', the text that reads:

> If FEAT_MTE2 is implemented, on completion of a DSB instruction, all updates to TFSR_ELx.TFx or TFSRE0_EL1.TFx due to Tag Check fails caused by accesses for which the DSB operates will be complete.

Is changed to read:

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

Also, the text in section D6.7 (PE handling of Tag Check Fault) that reads:

> Indirect writes to TFSRE0_EL1, and any TFSR_ELx accessible at ELy, which are caused by a Tag Check Fault are synchronized by any of:
>
> - An exception entry to ELy, if SCTLR_ELy.ITFSB has the value of `0b1`.
> - A DSB at ELy in program order, after the instruction causing the Tag Check Fault.

is changed to read:

> Indirect writes to TFSRE0_EL1 and any TFSR_ELx accessible at ELy that are caused by a Tag Check Fault are synchronized by any of:
>
> - An exception entry to ELy, if SCTLR_ELy.ITFSB has the value of `0b1`.

> - A DSB over the Non-shareable domain at ELy in program order, after the instruction causing the Tag Check Fault.

In section D13.1.2 (General behavior of accesses to the AArch64 System registers), in the subsection 'Synchronization requirements for AArch64 System registers', the text that reads:

> If FEAT_MTE2 is implemented, a data synchronization barrier (DSB) or an exception entry to ELy with SCTLR_ELy.ITFSB = `0b1` is required between an indirect write to TFSRE0_EL1, or any TFSR_ELx accessible at ELy, and a direct read or direct write of that register

Is changed to read:

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

2.73 C17562

In D13.2.47 (HAFGRTR_EL2, Hypervisor Activity Monitors Fine-Grained Read Trap Register), each of the bits AMEVCNTR1x_EL0 and AMEVTYPER1x_EL0 are made conditional on AMEVCNTR1<x> and AMEVTYPER1<x> registers being implemented. Otherwise, these fields are **RES0**.

2.74 D17565

In section J1.1.4 (aarch64/functions), the following is changed:

The code that reads:

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

is updated to read:

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

2.75 D17571

In sections C6.2.94 (HVC) and F5.1.54 (HVC), the text that reads:

> Non-secure software executing at EL1 can use this instruction to call the hypervisor to request a service.

is changed to read:

> Software executing at EL1 can use this instruction to call the hypervisor to request a service.

In section C6.2.94 (HVC), the text under 'The HVC instruction is **UNDEFINED**:' that currently reads:

> - At EL0.
> - At EL1 if EL2 is not enabled in the current Security state.
> - When SCR_EL3.HCE is set to 0.

is changed to read:

> - When EL3 is implemented and SCR_EL2.HCE is set to 0.
> - When EL3 is not implemented and HCR_EL3.HCD is set to 1.
> - When EL2 is not implemented.
> - At EL1 if EL2 is not enabled in the current Security state.
> - At EL0.

In section F5.1.54 (HVC), the text that currently reads:

> The HVC instruction is:
>
> - **UNDEFINED** in Secure state, and in User mode in Non-secure state.
> - When SCR_EL3.HCE is set to 0, **UNDEFINED** in Non-secure EL1 modes and CONSTRAINED UNPREDICTABLE in Hyp mode.

is changed to read:

> The HVC instruction is **UNDEFINED**:
>
> - When EL3 is implemented and using AArch64, and SCR_EL3.HCE is set to 0.
> - In Non-secure EL1 modes when EL3 is implemented and using AArch32, and SCR.HCE is set to 0.
> - When EL3 is not implemented and either HCR_EL2.HCD is set to 1, or HCR.HCD is set to 1.
> - When EL2 is not implemented.
> - In Secure state, if EL2 is not enabled in the current Security state.
> - In User mode.

> The HVC instruction is CONSTRAINED UNPREDICTABLE in Hyp mode when EL3 is implemented and using AArch32, and SCR.HCE is set to 0.

In section D1.14.3 (EL2 configurable controls), the subsection title 'Disabling Non-secure state execution of HVC instructions' is changed to 'Disabling execution of HVC instructions'. Similarly, in section D1.14.4 (EL3 configurable controls), the subsection title 'Enabling EL3, EL2, and Non-secure EL1 execution of HVC instructions' is changed to 'Enabling EL3, EL2, and EL1 execution of HVC instructions', and the text within the subsection that reads:

> For EL1, this enable control applies to HVC instructions in Non-secure state only.

is changed to read:

> For EL1, this enable control applies to Secure state only if EL2 is enabled in Secure state in the current Execution state.

In section G1.9.1 (AArch32 state PE mode descriptions), in the subsection 'Notes on the AArch32 PE modes', the text under the 'Hyp mode' entry that reads:

> The Hypervisor Call exception and Hyp Trap exception are implemented as part of EL2 and are always taken to Hyp mode.
>
> Note: This means that Hypervisor Call and Hyp Trap exceptions cannot be taken from Secure state.
>
> When the value of the Hypervisor Call enable bit, SCR.HCE, is 1, executing an HVC (Hypervisor Call) instruction in a Non-secure EL1 mode generates a Hypervisor Call exception.

is changed to read:

> The Hypervisor Call exception and Hyp Trap exception are implemented as part of EL2 and are always taken to Hyp mode when EL2 is using AArch32.
>
> Executing an HVC (Hypervisor Call) instruction generates a Hypervisor Call exception. See Hypervisor Call (HVC) exception on page G1-6012.

Within the same section, in the subsection 'Hyp mode', the following text is removed:

> In an implementation that includes EL3, from reset, the HVC instruction is **UNDEFINED** in Non-secure EL1 modes, meaning entry to Hyp mode is disabled by default.

and the following text:

> If EL3 is not implemented and HCR_EL2 or HCR.HCD is set to 1, the HVC instruction is undefined in Hyp mode.

is changed to read:

> If EL3 is not implemented and HCR.HCD is set to 1, the HVC instruction is undefined in Hyp mode

2.76 D17572

In section C3.2.12 (Atomic instructions), the following text is added at the start of the section:

> The atomic instructions perform atomic read and write operations on a memory location such that the architecture guarantees that no modification of that memory location by another observer can occur between the read and the write defined by that instruction.

2.77 D17573

In section J1.1.4 (aarch64/functions), the function AArch64.AccessIsTagChecked() is changed:

The code that reads:

```
    if acctype IN {AccType_IFETCH, AccType_TTW} then
        return FALSE;
```

is updated to read:

```
    if acctype IN {AccType_IFETCH, AccType_TTW, AccType_DC, AccType_IC} then
        return FALSE;
```

2.78 R17578

In section I5.8.31 (ERR<n>PFGF, Pseudo-fault Generation Feature Register, n = 0 - 65534), a new ID field is added at bit [28]:

> NA, bits [28]
>
> No access. Defines whether this component fakes detection of the error on an access to the component or spontaneously in the fault injection state.
>
> The defined values of this field are:
>
> `0b0` The component fakes detection of the error on an access to the component.
>
> `0b1` The component fakes detection of the error spontaneously in the fault injection state.

2.79 R17579

In section B2.3.7 (Completion and endpoint ordering), subsection 'Peripherals', under the heading 'Peripheral coherence order', the following bullet point:

> - RW1 and RW2 are accesses using Device-nGnRE or Device-nGnRnE attributes and RW1 appears in program order before RW2.

is changed to read:

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

and the following text is added as a Note immediately after the bulleted list:

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

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

> If the same memory location is mapped with different aliases, with a different Reordering attribute value or any different Device memory attribute values, these are a type of mismatched attribute. For information about the effects of accessing memory with mismatched attributes, see Mismatched memory attributes on page B2-175.

is replaced with:

> If the same memory location is mapped with different aliases, and different attribute values, these are a type of mismatched attribute. The different attributes could be:
>
> - A different Reordering attribute value.
> - A different Device memory attribute value.
> - When FEAT_XS is implemented, a different XS attribute value. For information about the effects of accessing memory with mismatched attributes, see Mismatched memory attributes on page B2-175.

and in section B2.8 (Mismatched memory attributes), under the following bulleted list:

> Physical memory locations are accessed with mismatched attributes if all accesses to the location do not use a common definition of all of the following attributes of that location:
>
> - Memory type: Device-nGnRnE, Device-nGnRE, Device-nGRE, Device-GRE or Normal.
> - Shareability.
> - Cacheability, for the same level of the inner or outer cache, but excluding any cache allocation hints.

a new bullet point is added:

- When FEAT_XS is implemented, XS attribute.

2.80 D17583

In section D13.8.22 (CNTPOFF_EL2, Counter-timer Physical Offset register), the text that reads:

> The offsetting of the timers and counters based on EL2 using AArch64 apply at:
>
> - EL1 when EL1 is using AArch64 or AArch32.
> - EL0 when EL0 is using AArch64 or AArch32.
>
> When EL2 is implemented and enabled in the current Security state, the physical counter uses a fixed physical offset of zero if either of the following are true:
>
> - CNTHCTL_EL2.ECV is 0.
> - SCR_EL3.ECVEn is 0.
> - HCR_EL2.{E2H, TGE} is {1, 1}.

is changed to read:

> The CNTPOFF_EL2 offset applies to:
>
> - Direct reads of the physical counter from EL0 or EL1.
> - Indirect reads of the physical counter by the EL1 physical timer.
>
> When EL2 is implemented and enabled in the current Security state, the physical counter uses a fixed physical offset of zero if any of the following are true:
>
> - CNTHCTL_EL2.ECV is 0.
> - SCR_EL3.ECVEn is 0.
> - HCR_EL2.{E2H, TGE} is {1, 1}.

2.81 R17584

In section D6.4.1 (Virtual address translation), the following statement is added:

> If SCTLR_ELx.C is 0 for a stage 1 translation regime, it is **CONSTRAINED UNPREDICTABLE** between:
>
> - The stage 1 translation is treated as Untagged.
> - SCTLR_ELx.C has no effect on whether the stage 1 translation is treated as Tagged or Untagged.
>
> Note: To ensure consistent behaviour, software can set SCTLR_ELx.ATA to 0 when SCTLR_ELx.C is 0.

In section D5.5.7 (Combining the stage 1 and stage 2 attributes, EL1&0 translation regime), the statement which currently reads:

> If the stage 1 page or block descriptor specifies the Tagged attribute, the final memory type is Tagged only if the final cacheable memory type is Inner and Outer Write-back Cacheable and the final allocation hints are Read-Allocate, Write-Allocate.

is changed to read:

> If the stage 1 translation is treated as Tagged, the final memory type is Tagged only if the final cacheable memory type is Inner and Outer Write-back Cacheable and the final allocation hints are Read-Allocate, Write-Allocate.

In section J1.1.5 (aarch64/translation), the pseudocode for the S1AttrDecode() function which currently reads:

```
elsif HaveMTE2Ext() && attrfield == '11110000' then // Normal, Tagged WB-RWA
    memattrs.memtype = MemType_Normal;
    memattrs.outer = LongConvertAttrsHints('1111', acctype); // WB_RWA
    memattrs.inner = LongConvertAttrsHints('1111', acctype); // WB_RWA
    memattrs.shareable = SH<1> == '1';
    memattrs.outershareable = SH == '10';
    memattrs.tagged = TRUE;
```

is changed to read:

```
elsif HaveMTE2Ext() && attrfield == '11110000' then // Normal, Tagged WB-RWA
    memattrs.memtype = MemType_Normal;
    memattrs.outer = LongConvertAttrsHints('1111', acctype); // WB_RWA
    memattrs.inner = LongConvertAttrsHints('1111', acctype); // WB_RWA
    memattrs.shareable = SH<1> == '1';
    memattrs.outershareable = SH == '10';
    // if SCTLR_ELx.C is 0 it is Constrained UNPREDICTABLE if the S1 Cache disable
 has an effect on whether
    // the stage 1 translation is treated as Tagged
    if ConstrainedUnpredictableBool() then
        memattrs.tagged = memattrs.inner.attrs == MemAttr_WB && memattrs.outer.attrs
 == MemAttr_WB;
    else
        memattrs.tagged = TRUE;
```

2.82 D17590

In section D9.7.2 (The owning Exception level), in the subsection 'When the owning Exception level is Non-secure EL2', the text that reads:

> The Profiling Buffer addresses are in the Non-secure EL2 translation regime. If both HCR_EL2.E2H is set to 1 and HCR_EL2.TGE is set to 1, this is an EL2&0 translation regime using the current EL2&0 translation regime ASID from TTBRx_EL2.

is corrected to read:

> The Profiling Buffer addresses are in the Non-secure EL2 translation regime. If HCR_EL2.E2H is 1, this is an EL2&0 translation regime using the current EL2&0 translation regime ASID from TTBRx_EL2.

Within the same section, in the subsection 'When the owning Exception level is Secure EL2', the text that reads:

> The Profiling Buffer addresses are in the Secure EL2 translation regime. If both HCR_EL2.E2H is set to 1 and HCR_EL2.TGE is set to 1, this is an EL2&0 translation regime using the current EL2&0 translation regime ASID from TTBRx_EL2.

is corrected to read:

> The Profiling Buffer addresses are in the Secure EL2 translation regime. If HCR_EL2.E2H is 1, this is an EL2&0 translation regime using the current EL2&0 translation regime ASID from TTBRx_EL2.

2.83 D17592

In section J1.1.2 (aarch64/exceptions), the following is changed:

The code in AArch64.EffectiveTCF() that reads:

```
bits(2) AArch64.EffectiveTCF(bits(2) el)
    bits(2) tcf;

    if el == EL3 then
        tcf = SCTLR_EL3.TCF;
    elsif el == EL2 then
        tcf = SCTLR_EL2.TCF;
    elsif el == EL1 then
        tcf = SCTLR_EL1.TCF;
    elsif el == EL0 && HCR_EL2.<E2H,TGE> == '11' then
        tcf = SCTLR_EL2.TCF0;
    elsif el == EL0 && HCR_EL2.<E2H,TGE> != '11' then
        tcf = SCTLR_EL1.TCF0;

    return tcf;
```

is updated to read:

```
bits(2) AArch64.EffectiveTCF(AccType acctype)
    bits(2) tcf, el;
    el = S1TranslationRegime();

    if el == EL3 then
        tcf = SCTLR_EL3.TCF;
    elsif el == EL2 then
        if AArch64.AccessUsesEL(acctype) == EL0 then
            tcf = SCTLR_EL2.TCF0;
        else
            tcf = SCTLR_EL2.TCF;
    elsif el == EL1 then
        if AArch64.AccessUsesEL(acctype) == EL0 then
            tcf = SCTLR_EL1.TCF0;
        else
```

```
            tcf = SCTLR_EL1.TCF;
    if tcf == '11' then        //reserved value
        if !HaveMTE3Ext() then
            (-,tcf) = ConstrainUnpredictableBits(Unpredictable_RESTCF);

    return tcf;
```

And the code in AArch64.TagCheckFault() that reads:

```
AArch64.TagCheckFault(bits(64) vaddress, AccType acctype, boolean iswrite)
    bits(2) tcf = AArch64.EffectiveTCF(PSTATE.EL);
    if !HaveMTE3Ext() && tcf == '11' then
        (-,tcf) = ConstrainUnpredictableBits(Unpredictable_RESTCF);

    case tcf of
        when '00'       // Tag Check Faults have no effect on the PE
            return;
        when '01'       // Tag Check Faults cause a synchronous exception
            AArch64.RaiseTagCheckFault(vaddress, iswrite);
        when '10'       // Tag Check Faults are asynchronously accumulated
            AArch64.ReportTagCheckFault(PSTATE.EL, vaddress<55>);
        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.
            readwrite = acctype IN {AccType_ATOMICRW,
                                    AccType_ORDEREDATOMICRW,
                                    AccType_ORDEREDRW};
```

is updated to read:

```
AArch64.TagCheckFault(bits(64) vaddress, AccType acctype, boolean iswrite)
    bits(2) tcf;
    tcf = AArch64.EffectiveTCF(acctype);

    case tcf of
        when '00'       // Tag Check Faults have no effect on the PE
            return;
        when '01'       // Tag Check Faults cause a synchronous exception
            AArch64.RaiseTagCheckFault(vaddress, iswrite);
        when '10'       // Tag Check Faults are asynchronously accumulated
            AArch64.ReportTagCheckFault(PSTATE.EL, vaddress<55>);
        when '11'
            // Check for access performing both a read and a write.
            readwrite = acctype IN {AccType_ATOMICRW,
                                    AccType_ORDEREDATOMICRW,
                                    AccType_ORDEREDRW};
```

2.84 D17593

In section J1.1.2 (aarch64/exceptions), AArch64.TakeException and in section J1.2.2 (aarch32/exceptions) in each of the AArch32.EnterXXXMode functions, the following is added after existing calls to 'BranchTo':

```
    CheckExceptionCatch(TRUE);                  // Check for debug event on excep\
tion entry
```

In sections J1.1.4 (aarch64/instrs) and J1.2.3 (aarch32/functions), in the functions AArch64.ExceptionReturn and AArch32.ExceptionReturn respectively, the following is added after the calls to 'BranchTo':

```
   CheckExceptionCatch(FALSE);              // Check for debug event on exception
 return
```

In section J1.3.1 (shared/debug), the following comment in CheckExceptionCatch:

```
   // Called after an exception entry or exit, that is, such that IsSecure() and
 PSTATE.EL are correct
   // for the exception target.
```

is updated to read:

```
   // Called after an exception entry or exit, that is, such that IsSecure() and
 PSTATE.EL are correct
   // for the exception target. When FEAT_Debugv8p2 is not implemented, this func\
tion might also be called
   // at any time.
```

The similar comment for returns is also updated. Such calls are outside the scope of the Armv8 Pseudocode model.

2.85 D17604

In section D13.2.50 (HDFGRTR_EL2, Hypervisor Debug Fine-Grained Read Trap Register), the following field is added at bit [63]:

> PMBIDR_EL1, bit [63]
>
> When FEAT_SPE is implemented: PMBIDR_EL1. Trap MRS reads of PMBIDR_EL1 at EL1 using AArch64 to EL2.
>
> `0b0` MRS reads of PMBIDR_EL1 are not trapped by this mechanism.
>
> `0b1` If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == `0b1`, then MRS reads of PMBIDR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value `0x18`, unless the read generates a higher priority exception.
>
> On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.
>
> Otherwise: Reserved, **RES0**.

2.86 D17606

In section D13.2.48 (HCR_EL2, Hypervisor Configuration Register), in the EnSCXT field, the text that reads:

> When FEAT_CSV2 is implemented:

is replaced with:

> When FEAT_CSV2_2 or FEAT_CSV2_1p2 is implemented:

The same change is made in the following sections:

- D13.2.53 (HFGRTR_EL2, Hypervisor Fine-Grained Read Trap Register), for the SCXTNUM_EL0 and SCXTNUM_EL1 fields,
- D13.2.54 (HFGWTR_EL2, Hypervisor Fine-Grained Write Trap Register), for the SCXTNUM_EL0 and SCXTNUM_EL1 fields,
- D13.2.115 (SCR_EL3, Secure Configuration Register), for the EnSCXT field,
- D13.2.116 (SCTLR_EL1, System Control Register (EL1)), for the TSCXT field,
- D13.2.117 (SCTLR_EL2, System Control Register (EL2)), for the TSCXT field.

2.87 D17615

In section D13.6.10 (PMSIDR_EL1, Sample Profiling ID Register), the following field is added at bit [24]:

> PBT, bit [24]
>
> Previous branch target Address packet. Defined values are:
>
> `0b0` Previous branch target Address packet not supported.
>
> `0b1` Previous branch target Address packet support implemented. FEAT_SPEv1p2 adds the OPTIONAL functionality identified by the value 1.
>
> Otherwise: Reserved, **RAZ**.

In section D13.2.59 (ID_AA64DFR0_EL1, AArch64 Debug Feature Register 0), in the PMSVer field description, the following bulleted text in the `0b0011` value definition is deleted:

> - The last branch target Address packet.

and is replaced by the following bullet:

> - Support for the OPTIONAL previous branch target Address packet.

2.88 D17617

In section D5.5.1 (The stage 1 memory region attributes), in the subsection 'Stage 1 definition of the XS attribute', the text that reads:

> When FEAT_XS is implemented, all stage 1 memory types defined in the MAIR_ELx registers have the XS attribute set to 1, unless they are any of the following, which have the XS attribute set to 0:
>
> - For Device memory:
>   - Device memory types that use the MAIR_ELx.Attrn encoding `0b0000dd01`.
> - For Normal memory:
>   - Inner Write-Back Cacheable, Outer Write-back Cacheable memory types, including any memory types that are treated as Write-Back Cacheable as a result of **IMPLEMENTATION DEFINED** choices in the architecture.
>   - Inner Write-through Cacheable and Outer Write-through Cacheable memory types that use the MAIR_ELx.Attrn encoding `0b1010000`.
>   - Inner Non-cacheable, Outer Non-cacheable memory types that use the MAIR_ELx.Attrn encoding `0b01000000`.

is corrected to read:

> When FEAT_XS is implemented, all stage 1 memory types defined in the MAIR_ELx or TCR_ELx registers have the XS attribute set to 1, unless they are any of the following, which have the XS attribute set to 0:
>
> - For Device memory:
>   - Device memory types that use the MAIR_ELx.Attrn encoding `0b0000dd01`.
> - For Normal memory:
>   - Inner Write-Back Cacheable, Outer Write-back Cacheable memory types defined in the MAIR_ELx or TCR_ELx registers, including any memory types that are treated as Write-Back Cacheable as a result of **IMPLEMENTATION DEFINED** choices in the architecture.
>   - Inner Write-through Cacheable and Outer Write-through Cacheable memory types that use the MAIR_ELx.Attrn encoding `0b1010000`.
>   - Inner Non-cacheable, Outer Non-cacheable memory types that use the MAIR_ELx.Attrn encoding `0b01000000`.

Similarly, in section G5.7.1 (Overview of memory region attributes for stage 1 translations), in the subsection 'Stage 1 definition of the XS attribute', the text that reads:

> When FEAT_XS is implemented, all stage 1 memory types defined in the MAIR0, MAIR1, HMAIR0, HMAIR1, PRRR, and NMRR registers have the XS attribute set to 1, unless they are Inner Write-Back Cacheable, Outer Write-back Cacheable, which have the XS attribute set to 0. This includes any memory types which, as a result of **IMPLEMENTATION DEFINED** choices in the architecture, are treated as Write-Back Cacheable.

is clarified to read:

> When FEAT_XS is implemented, all stage 1 memory types defined in the MAIR0, MAIR1, HMAIR0, HMAIR1, PRRR, and NMRR or TTBCR or HTCR registers or in the page tables have the XS attribute set to 1, unless they are Inner Write-Back Cacheable, Outer Write-back Cacheable, which have the XS attribute set to 0. This includes any memory types which, as a result of **IMPLEMENTATION DEFINED** choices in the architecture, are treated as Write-Back Cacheable.

2.89 D17632

In section D13.4.1 (PMCCFILTR_EL0, Performance Monitors Cycle Count Filter), in the descriptions of the M and SH fields, the following Note is deleted:

> This field is not visible in the AArch32 PMCCFILTR System register.

The equivalent Notes are deleted in section D13.4.9 (PMEVTYPER<n>_EL0, Performance Monitors Event Type Registers, n = 0 - 30), in the descriptions of the M and SH fields.

2.90 D17633

In section J1.3.3 (shared/functions), the pseudocode function GenMPAMcurEL() generates the default PARTID in some conditions when it should not.

As such, the code that reads as:

```
MPAMinfo GenMPAMcurEL(AccType acctype)
    :
    if HaveEMPAMExt() then
        if MPAMisEnabled() && (!secure || MPAM3_EL3.SDEFLT == '0') then
            if UsingAArch32() then
                :
            else
                mpamel = PSTATE.EL;
    :
```

is changed to read:

```
MPAMinfo GenMPAMcurEL(AccType acctype)
    :
    if HaveEMPAMExt() then
        if MPAMisEnabled() && (!secure || MPAM3_EL3.SDEFLT == '0') then
            if UsingAArch32() then
                :
            else
                validEL = TRUE;
                mpamel = PSTATE.EL;
    :
```

2.91 D17640

In section D4.4.8 (A64 Cache maintenance instructions), in the subsection 'Effects of virtualization and Security state on the cache maintenance instructions', the footnote to table D4-7 that reads:

> Dependencies on the VMID apply even when HCR_EL2.VM is set to 0. VTTBR_EL2.VMID resets to zero, meaning there is a valid VMID.

is changed to read:

> Dependencies on the VMID apply even when HCR_EL2.VM is set to 0. The architecture does not define a reset value for VTTBR_EL2.VMID, and therefore, in any implementation that includes EL2, the boot software executed when reset is deasserted must initialize VTTBR_EL2.VMID.

2.92 C17641

In section D5.9.2 (About Armv8 Translation Lookaside Buffers (TLBs)), in the subsection 'Global and process-specific translation table entries', the text that reads:

> When a PE is using the VMSAv8-64 translation table format, and is in Secure state, a translation must be treated as non-global, regardless of the value of the nG bit, if NSTable is set to 1 at any level of the translation table walk.

is clarified to read:

> When a PE is using the VMSAv8-64 translation table format which supports both global and non-global entries, and is in Secure state, a stage 1 translation must be treated as non-global, regardless of the value of the nG bit, if NSTable is set to 1 at any level of the translation table walk.

In section D5.3.4 (Control of Secure or Non-secure memory access), in the subsection 'Hierarchical control of Secure or Non-secure memory accesses', the text that reads:

> In addition, an entry fetched in Secure state is treated as non-global if it is read from the Non-secure IPA space memory. That is, these entries must be treated as if nG==1, regardless of the value of the nG bit.

is clarified to read:

> In addition, an entry fetched in Secure state is treated as non-global if it is part of a stage 1 translation which supports both global and non-global entries, and the stage 1 translation was read from a non-secure stage 1 output address . These entries must be treated as if nG==1, regardless of the value of the nG bit.

2.93 C17651

In sections D13.3.9 (MDRAR_EL1, Monitor Debug ROM Address Register) and G8.3.12 (DBGDRAR, Debug ROM Address Register), the following text is added to the description of the 'Valid, bits [1:0]' field:

> Arm recommends implementations set this field to zero.

Additionally, in section K2.1 (About the recommended external debug interface), in Table K2-1 'Recommended debug interface signals', the following text is added to the Notes column entries for DBGROMADDR[n:12] and DBGROMADDRV:

> Arm recommends these signals are tied LOW.

2.94 D17653

In section G6.2.4 (Timers), the text that reads:

> A Secure PL1 physical timer. This timer:
> - Is accessible from Secure EL1 using AArch32 when EL3 is using AArch64.
> - Is accessible from Secure EL3 when EL3 is using AArch32.

is corrected to read:

> A Secure PL1 physical timer. This timer is accessible from EL3 when EL3 is using AArch32. Note: when EL3 is using AArch64, the AArch32 EL1 timers are not banked between Secure and Non-secure state.

2.95 D17654

In section A2.5.1 (Architectural features added by Armv8.2), for FEAT_PAN2 , the text that reads:

> This feature is mandatory in Armv8.2 implementations.

is updated to read:

> This feature is OPTIONAL in Armv8.1 implementations and mandatory in Armv8.2 implementations.

In section D13.2.65 (ID_AA64MMFR1_EL1, AArch64 Memory Model Feature Register 1), PAN bits [23:20], the values that read:

> In Armv8.1, the permitted values are `0b0001` and `0b0011`.

are updated to read:

> In Armv8.1, the permitted values are `0b0001`, `0b0010`, and `0b0011`.

2.96 C17669

In section B2.3.9 (Restrictions on the effects of speculation), and in section E2.3.9 (Restrictions on the effects of speculation), the following Note is added:

> Note: The prohibition of using data loaded under speculation with faults to form addresses, condition codes or SVE predicate values does not prohibit the use of value predicted data from such locations for such purposes, so long as the training of the data value prediction was from the hardware defined context that is using the prediction. A consequence of this is that training of value prediction cannot be based on data loaded under speculation with a translation or permission fault.

2.97 D17673

In section H9.2.40 (EDPRSR, External Debug Processor Status Register), in the EPMAD field, the condition that reads:

> When FEAT_Debugv8p4 is implemented:

is corrected to read:

> When FEAT_Debugv8p4 is implemented and FEAT_PMUv3 is implemented:

2.98 C17680

In section H9.2.42 (EDSCR, External Debug Status and Control Register), in the RW field, the value text that reads:

RW	Meaning	Applies when
`0b1111`	All Exception levels are using AArch64 or the PE is in Non-debug state.	
`0b1110`	The PE is in Debug state. EL0 is using AArch32. All other Exception levels are using AArch64. Only permitted if the PE is executing at EL0.	When AArch32 is supported at any Exception level

is clarified to read:

RW	Meaning	Applies when
`0b1111`	Any of the following:	
	- The PE is in Non-debug state.	
	- The PE is at EL0 using AArch64.	
	- The PE is not at EL0, and EL1, EL2, and EL3 are using AArch64.	

RW	Meaning	Applies when
`0b1110`	The PE is in Debug state at EL0. EL0 is using AArch32. EL1, EL2, and EL3 are using AArch64.	When AArch32 is supported at any Exception level

2.99 D17682

In section G8.2.54 (FPSCR, Floating-Point Status and Control Register), the accessibility pseudocode that reads:

```
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H != '1' && CPTR_EL2.TFP
 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x07);
```

is replaced with:

```
 if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED
 \"EL3 trap priority when SDD == '1'\" && !ELUsingAArch32(EL3) && CPTR_EL3.TFP ==
 '1' then
        UNDEFINED;
.
.
.

 elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TFP == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
```

Also, the accessibility pseudocode that reads:

```
    if HCR_EL2.E2H == '0' && CPTR_EL2.TFP == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x07);
```

is replaced with:

```
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED
 \"EL3 trap priority when SDD == '1'\" && !ELUsingAArch32(EL3) && CPTR_EL3.TFP ==
 '1' then
        UNDEFINED;

.
.
.

    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TFP == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
```

The following accessibility pseudocode is added:

```
    elsif CPTR_EL3.TFP == '1' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x07);
```

The equivalent changes are made in the following sections:

- G8.2.53 (FPEXC, Floating-Point Exception Control register),
- G8.2.55 (FPSID, Floating-Point System ID register),
- G8.2.115 (MVFR0, Media and VFP Feature Register 0),
- G8.2.116 (MVFR1, Media and VFP Feature Register 1),
- G8.2.117 (MVFR2, Media and VFP Feature Register 2).

Additionally, in section D13.2.32 (CPTR_EL3, Architectural Feature Trap Register (EL3)), the text in the TFP, bit [10] field description is clarified to read:

> This includes the following registers, all reported using ESR_ELx.EC value `0x07`:
>
> - FPCR, FPSR, FPEXC32_EL2, and any of the SIMD and floating-point registers V0-V31, including their views as D0-D31 registers or S0-S31 registers.
> - MVFR0, MVFR1, MVFR2, FPSCR, FPEXC, and any of the SIMD and floating-point registers Q0-Q15, including their views as D0-D31 registers or S0-S31 registers.
> - VMSR accesses to FPSID. Permitted VMSR accesses to FPSID are ignored, but for the purposes of this trap the architecture defines a VMSR access to the FPSID from EL1 or higher as an access to a SIMD and floating-point register.

2.100 D17684

In section D13.2.68 (ID_AA64PFR1_EL1, AArch64 Processor Feature Register 1), in the CSV2_frac field, the text that reads:

> In Armv8.0, the permitted values are `0b0000`, `0b0001`, and `0b0010`. From Armv8.5, the permitted values are `0b0001` and `0b0010`.

is changed to read:

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

2.101 C17685

In section B1.3.6 (About PSTATE.DIT), after the bullet point that reads:

> - All loads and stores must have their timing insensitive to the value of the data being loaded or stored.

the following Note is added:

> Note: The use of value prediction for load data values when PSTATE.DIT is set, is not compatible with the requirement that the timing is insensitive to the data value being loaded.

2.102 D17686

In section A2.5.1 (Architectural features added by Armv8.2), under the item 'FEAT_XNX, Translation table stage 2 Unprivileged Execute-never', the text that reads:

> This feature is mandatory in Armv8.2 implementations.

is clarified to read:

> This feature is mandatory in Armv8.2 implementations that implement EL2.

Similarly, in section A2.7.1 (Architectural features added by Armv8.4), under the item 'FEAT_S2FWB, Stage 2 forced Write-Back', the line that reads:

> This feature is mandatory in Armv8.4 implementations.

is clarified to read:

> This feature is mandatory in Armv8.4 implementations that implement EL2.

2.103 D17687

In section D13.2.135 (TTBR0_EL1, Translation Table Base Register 0 (EL1)), the text in the BADDR field description that reads:

> When the value of ID_AA64MMFR0_EL1.PARange indicates that the implementation does not support a 52 bit PA size, if a translation table lookup uses this register when TCR_EL1.IPS is `0b110` and the value of register bits[5:2] is nonzero, an Address size fault is generated.

is changed to read:

> When the value of ID_AA64MMFR0_EL1.PARange indicates that the implementation does not support a 52 bit PA size, if a translation table lookup uses this register when the Effective value of TCR_EL1.IPS is `0b110` and the value of register bits[5:2] is nonzero, an Address size fault is generated.

The equivalent changes are made in the following sections:

- D13.2.136 (TTBR0_EL2, Translation Table Base Register 0 (EL2)),
- D13.2.137 (TTBR0_EL3, Translation Table Base Register 0 (EL3)),
- D13.2.138 (TTBR1_EL1, Translation Table Base Register 1 (EL1)),
- D13.2.139 (TTBR1_EL2, Translation Table Base Register 1 (EL2)).

2.104 D17694

In section A2.10.1 (Architectural features added by Armv8.7), under the item 'FEAT_AFP, Alternate floating-point behavior', the text that reads:

> This feature is OPTIONAL in Armv8.7 implementations.

is corrected to read:

> This feature is mandatory in Armv8.7 implementations that implement floating-point.

2.105 D17700

In section G8.2.126 (SCTLR, System Control Register), the text that reads:

> When FEAT_CSV2 is implemented:

is corrected to read:

> When FEAT_SPECRES is implemented:

2.106 D17705

In section D13.2.48 (HCR_EL2, Hypervisor Configuration Register), in the FMO, bit [3] field, the definition of value `0b0` that currently reads:

> `0b0` When executing at Exception levels below EL2, and EL2 is enabled in the current Security state:
>
> - Physical FIQ interrupts are not taken to EL2.
> - When the value of HCR_EL2.TGE is 0, if the PE is executing at EL2 using AArch64, physical FIQ interrupts are not taken unless they are routed to EL3 by the SCR_EL3.FIQ bit.
> - Virtual FIQ interrupts are disabled.

is corrected to read:

> `0b0` When executing at Exception levels below EL2, and EL2 is enabled in the current Security state:
>
> - When the value of HCR_EL2.TGE is 0, Physical FIQ interrupts are not taken to EL2.
> - When the value of HCR_EL2.TGE is 1, Physical FIQ interrupts are taken to EL2 unless they are routed to EL3.
> - Virtual FIQ interrupts are disabled.

Within the same section, equivalent edits are made for the IMO, bit [4], and AMO, bit [5] fields.

2.107 D17712

In section D13.2.38 (ESR_EL2, Exception Syndrome Register (EL2)), in the subsection 'ISS encoding an exception from a Data Abort', the text in the SRT, bits [20:16] field that reads:

> Syndrome Register Transfer. When FEAT_LS64 is implemented, if a memory access generated by an ST64BV, ST64BV0, ST64B, or LD64B instruction generates a Data Abort for a Translation fault, Access flag fault, or Permission fault, then this field holds register specifier, Xt.

is corrected to read:

> Syndrome Register Transfer. The register number of the Wt/Xt/Rt operand of the faulting instruction.

2.108 D17715

In section D13.5.4 (AMCNTENCLR0_EL0, Activity Monitors Count Enable Clear Register 0), the text that reads:

> Bits [15:N] are **RAZ/WI**. N is the value in AMCGCR_EL0.CG0NC.

is clarified to read:

> When N is less than 16, bits [15:N] are **RAZ/WI**, where N is the value in AMCGCR_EL0.CG0NC.

Equivalent changes are made in the following sections:

- D13.5.6 (AMCNTENSET0_EL0, Activity Monitors Count Enable Set Register 0),
- G8.5.3 (AMCNTENCLR0, Activity Monitors Count Enable Clear Register 0),
- G8.5.5 (AMCNTENSET0, Activity Monitors Count Enable Set Register 0),
- I5.5.7 (AMCNTENCLR0, Activity Monitors Count Enable Clear Register 0),
- I5.5.9 (AMCNTENSET0, Activity Monitors Count Enable Set Register 0).

In section D13.5.5 (AMCNTENCLR1_EL0, Activity Monitors Count Enable Clear Register 1), the text that reads:

> Bits [15:N] are **RAZ/WI**. N is the value in AMCGCR_EL0.CG1NC.

is clarified to read:

> When N is less than 16, bits [15:N] are **RAZ/WI**, where N is the value in AMCGCR_EL0.CG1NC.

Equivalent changes are made in the following sections:

- D13.5.7 (AMCNTENSET1_EL0, Activity Monitors Count Enable Set Register 1),
- G8.5.4 (AMCNTENCLR1, Activity Monitors Count Enable Clear Register 1),

- G8.5.6 (AMCNTENSET1, Activity Monitors Count Enable Set Register 1),
- I5.5.8 (AMCNTENCLR1, Activity Monitors Count Enable Clear Register 1),
- I5.5.10 (AMCNTENSET1, Activity Monitors Count Enable Set Register 1).

2.109 D17722

In section D1.9.1 (PE state on reset to AArch64 state), the following text is deleted:

> - The enables for the counter event stream are set to 0. This means that the following bits are set to 0:
>   - CNTKCTL_EL1.EVNTEN.
>   - If the implementation includes EL2, CNTHCTL_EL2.EVNTEN.

Additionally, in section D13.8.2 (CNTHCTL_EL2, Counter-timer Hypervisor Control register) in the 'Otherwise' fieldset, and section D13.8.15 (CNTKCTL_EL1, Counter-timer Kernel Control register), the text in the EVNTEN, bit [2] description that reads:

> On a Warm reset, this field resets to 0.

is changed to read:

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

2.110 D17723

In H9.2.42 (EDSCR, External Debug Status and Control Register), in the field 'RW, bits [13:10]', the value text that reads:

RW	Meaning	Applies when
`0b110x`	The PE is in Debug state. EL0 and EL1 are using AArch32. EL2 and EL3 are using AArch64. Only permitted if EL2 is implemented and enabled in the current Security state.	When AArch32 is supported at any Exception level
`0b10xx`	The PE is in Debug state. EL0, EL1, and, if implemented in the current Security state, EL2 are using AArch32. EL3 is using AArch64.	When AArch32 is supported at any Exception level, EL3 is implemented, EL3 is using AArch64 and EL2 is implemented

is corrected to read:

RW	Meaning	Applies when
`0b110x`	The PE is in Debug state. EL0 and EL1 are using AArch32. EL2 is enabled in the current Security state and using AArch64. If implemented, EL3 is using AArch64.	When AArch32 is supported at any Exception level and EL2 is implemented
`0b10xx`	The PE is in Debug state. EL0 and EL1 are using AArch32. EL2 is not implemented, disabled in the current Security state, or using AArch32. EL3 is using AArch64.	When AArch32 is supported at any Exception level and EL3 is implemented

2.111 C17732

In section D13.1.3, (Principles of the ID scheme for fields in ID registers), a new subsection 'Alternative ID scheme used for ID_AA64MMFR0_EL1 granule fields' is created, which explains the scheme and the code required by software that depends on a particular granule size.

In section D13.2.64 (ID_AA64MMFR0_EL1, AArch64 Memory Model Feature Regsiter 0) for each of the TGranx_2 fields, the following note is added:

> Note: This field does not follow the usual ID scheme. See [xref to new section].

2.112 C17734

In section H6.4.1 (Powerup request mechanism if FEAT_DoPD is implemented), the following note is added:

> Note: If the Core power domain can be powered down independently of the Debug power domain, Arm recommends the system implements an external debug component with a powerup request mechanism which can request the Core power domain to be powered up.

2.113 C17735

In section D13.6.10 (PMSIDR_EL1, Sample Profiling ID Register), the following text is added to the description of the MaxSize:

> The values `0b0100` and `0b0101` are not permitted for an implementation.

2.114 R17738

In section H2.4.2 (Executing instructions in Debug state), sub-section 'Executing T32 instruction in Debug state', under the heading 'Instructions that move System or Special-purpose registers to or from a general-purpose register', the entry that reads:

> MRS SPSR, MSR SPSR.

is updated to read:

> MRS SPSR, MSR SPSR_fsxc (register).

2.115 D17754

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

```
    enabled = !HaveEL(EL3) || !secure || MDCR_EL3.SDD == '0';
```

is updated to read:

```
    if HaveEL(EL3) && secure then
        enabled = MDCR_EL3.SDD == '0';
        if from == EL0 && ELUsingAArch32(EL1) then
            enabled = enabled || SDER32_EL3.SUIDEN == '1';
    else
        enabled = TRUE;
```

2.116 D17762

In section G3.1.2 (Register controls to enable self-hosted trace), the lines that read:

> If FEAT_TRF is implemented, and external self-hosted trace is not implemented, self-hosted trace is always enabled. If FEAT_TRF is implemented, and external self-hosted trace is implemented, self-hosted trace is also enabled if one of the following is true:

are changed to read:

> If FEAT_TRF is implemented, self-hosted trace is enabled if one of the following is true:

And the line that reads:

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

is changed to read:

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

2.117 D17763

In section J1.1.4 (aarch64/instrs), the following is updated:

The lines in function TLBI_RIPAS2() that read as:

```
r.address.address = start_address<51:0>;

TLBI(r);
```

are corrected to read as:

```
r.address.address = start_address<51:0>;
r.address.NS = if security == SS_NonSecure then '1' else Xt<63>;

TLBI(r);
```

2.118 D17765

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

```
AArch64.TagCheckFault(bits(64) vaddress, AccType acctype, boolean iswrite)
    bits(2) tcf = AArch64.EffectiveTCF(acctype);

    case tcf of
        when '00'       // Tag Check Faults have no effect on the PE
            return;
        when '01'       // Tag Check Faults cause a synchronous exception
            AArch64.RaiseTagCheckFault(vaddress, iswrite);
        when '10'       // Tag Check Faults are asynchronously accumulated
            AArch64.ReportTagCheckFault(PSTATE.EL, vaddress<55>);
```

is updated to read:

```
AArch64.TagCheckFault(bits(64) vaddress, AccType acctype, boolean iswrite)
    bits(2) tcf, el;
    el = AArch64.AccessUsesEL(acctype);
    tcf = AArch64.EffectiveTCF(acctype);
    case tcf of
        when '00'       // Tag Check Faults have no effect on the PE
            return;
        when '01'       // Tag Check Faults cause a synchronous exception
            AArch64.RaiseTagCheckFault(vaddress, iswrite);
        when '10'       // Tag Check Faults are asynchronously accumulated
            AArch64.ReportTagCheckFault(el, vaddress<55>);
```

2.119 D17767

In section D5.11.2 (Instruction caches), in the subsection 'VPIPT (VMID-aware PIPT) instruction caches', the text that currently reads:

> For a VPIPT instruction cache:
>
> - Instruction fetches from Non-secure EL1 and Non-secure EL0 are only permitted to hit in the cache if the instruction fetch is made using the VMID that was used when the entry in the instruction cache was fetched.
> - An instruction cache maintenance instruction executed at Non-secure EL0 or at Non-secure EL1 is required to have an effect on entries in the instruction cache only if those entries were fetched using the VMID that is current when the cache maintenance instruction is executed.

is modified to read:

> For a VPIPT instruction cache:
>
> - If VMIDs are being used for the current security state, instruction fetches from EL1 and EL0 are only permitted to hit in the cache if the instruction fetch is made using the VMID that was used when the entry in the instruction cache was fetched.
> - If VMIDs are being used for the current security state, an instruction cache maintenance instruction executed at EL0 or at EL1 is required to have an effect on entries in the instruction cache only if those entries were fetched using the VMID that is current when the cache maintenance instruction is executed.

The same edit is made in section G5.10.2 (Instruction Caches).

2.120 D17773

In section D7.10.3 (Common event numbers), subsection 'Common microarchitectural events' event `0x813D`, ITLB_WALK_RD, the text:

> The counter counts each access counted by L1I_TLB_RD that causes a refill or update of an instruction or unified LB involving at least one translation table walk access.

is updated to:

> The counter counts each access counted by L1I_TLB_RD that causes a refill or update of an instruction TLB involving at least one translation table walk access.

And in event `0x813F` ITLB_WALK_PRFM, the text:

> The counter counts each access counted by L1I_TLB_PRFM that causes a refill or update of an instruction or unified TLB involving at least one translation table walk access.

is updated to:

> The counter counts each access counted by L1I_TLB_PRFM that causes a refill or update of an instruction TLB involving at least one translation table walk access.

2.121 C17774

In section K1.1.12 (**CONSTRAINED UNPREDICTABLE** behaviors due to caching of control or data values), the title that reads:

> **CONSTRAINED UNPREDICTABLE** behaviors due to caching of control or data values

is changed to read:

> **CONSTRAINED UNPREDICTABLE** behaviors due to caching of System register control or data values

Also, the text that reads:

> The Arm architecture allows copies of control values or data values to be cached in a cache or TLB.

is changed to read:

> The Arm architecture allows copies of System register control or data values to be cached in a cache or TLB.

2.122 C17775

In section D13.2.83 (ID_MMFR4_EL1, AArch32 Memory Model Feature Register 4), in the description of the EVT bit, the text that reads:

> From Armv8.5, the permitted values are:
> - `0b0000` when EL2 is not implemented.
> - `0b0010` when EL2 is implemented.

is clarified to read:

> From Armv8.5, the permitted values are:
> - `0b0000` when EL2 is not implemented or does not support AArch32.
> - `0b0010` when EL2 is implemented and supports AArch32.

An equivalent clarification is made in section G8.2.96 (ID_MMFR4, Memory Model Feature Register 4).

2.123 D17777

In section C5.1.3 (op0==`0b00`, architectural hints, barriers and CLREX, and PSTATE access), in the subsection 'Instructions for accessing the PSTATE fields', the line that reads:

> Writes to PSTATE.{PAN, D, A, I, F} occur in program order without the need for additional synchronization.

is updated to read:

> Writes to PSTATE occur in program order without the need for additional synchronization.

2.124 D17782

In section F4.1.7 (Data-processing immediate), in the subsection 'Move Special Register and Hints (immediate)', a new line is added to the table:

R:imm4	imm12	Instruction page	Feature
00000	xxxx0001011x	Reserved hint, behaves as **NOP**.	-

And the following line is removed:

R:imm4	imm12	Instruction page	Feature
00000	xxxx0001111x	Reserved hint, behaves as **NOP**.	-

2.125 D17783

In section D13.2.63 (ID_AA64ISAR2_EL1, AArch64 Instruction Set Attribute Register 2), the text that reads:

> Configurations
>
> This register is present only from Armv8.7. Otherwise, direct accesses to ID_AA64ISAR2_EL1 are UNDEFINED. This register is introduced in 8.7.

is replaced by the following:

> Configurations
>
> Note: Prior to the introduction of the features described by this register, this register was unnamed and reserved, **RES0** from EL1, EL2, and EL3.

2.126 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.

2.127 D17819

In section D7.10.1 (Definitions), the term 'Operation counts for dot-product and multiply-accumulate operations' is changed to 'ALU operation counts', and the text that reads:

> Table D7-5 gives the operation counts for any combination of an applicable instruction when a PMU event in the range `0x80C0` to `0x80CF` is implemented.

is changed to read:

> The PMU events `0x80C0` to `0x80CF` count the number of ALU operations performed by each instruction. Table D7-5 gives the ALU operation counts for applicable instructions for these PMU events. For other instructions:
>
> - Multiply-add, multiply-subtract, fused multiply-add, and fused multiply-subtract instructions generate two ALU operations of the specified type per input element. For floating-point operations, these are the instructions counted by FP_FMA_SPEC.
> - All other data processing operations generate 1 ALU operation of the specified type per input element.

Within the same section, the following text is added to the definition of the term 'Operation speculatively executed':

> A Microarchitectural operation that is Speculatively executed.
>
> Note: In some events, operation has a more specific meaning described in the event. See also ALU operation counts.

In section D7.10.3 (Common event numbers), in subsection 'Common microarchitectural events', the part of the `0x80C1`, FP_FIXED_OPS_SPEC event definition that reads:

> Non-scalable floating-point element Operations speculatively executed
>
> The counter counts speculatively executed operations that would be counted by FP_SPEC but not by SVE_FP_SPEC.

is changed to read:

> Non-scalable floating-point element ALU operation speculatively executed
>
> The counter counts the number of ALU operations generated for speculatively executed operations that would be counted by FP_SPEC but not by SVE_FP_SPEC.

Similar changes are made to the other FIXED_OPS and SCALE_OPS events (0x80C0 to `0x80CF`), and the descriptions of counts for scalar and multiply-add operations is removed from these events.

2.128 D17820

The description of the condition HaveAnyAArch32() will be replaced, to mean 'There is at least one Exception level that supports AArch32', and an equivalent clarification will be made for the condition HaveAnyAArch64().

As a consequence of existing architecture rules, HaveAnyAArch32() can be replaced by the newer condition HaveAArch32EL(EL0), and HaveAnyAArch64() can be replaced by the newer condition ! HighestELUsingAArch32(), but in some cases, stricter conditions can be applied.

2.129 D17826

In section D13.2.48 (HCR_EL2, Hypervisor Configuration Register), the following text in the ATA, bit [56] description is removed:

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

The equivalent changes are made in the following sections:

- D13.2.115 (SCR_EL3, Secure Configuration Register), in the ATA, bit [26] description,
- D13.2.116 (SCTLR_EL1, System Control Register (EL1)), in the ATA, bit [43] description, and ATA0, bit [42] description,

- D13.2.117 (SCTLR_EL2, System Control Register (EL2)), in the ATA, bit [43] description, and ATA0, bit [42] description,
- D13.2.118 (SCTLR_EL3, System Control Register (EL3)), in the ATA, bit [43] description.

2.130 D17831

In section J1.3.3 (shared/functions), InterruptPending() which reads:

```
boolean InterruptPending()
    bit vIRQstatus = (if VirtualIRQPending() then '1' else '0') OR HCR_EL2.VI;
    bit vFIQstatus = (if VirtualFIQPending() then '1' else '0') OR HCR_EL2.VF;
    bits(3) v_interrupts = HCR_EL2.VSE : vIRQstatus : vFIQstatus;

    pending_physical_interrupt = (IRQPending() || FIQPending() ||
                                  IsPhysicalSErrorPending());
    pending_virtual_interrupt  = !IsInHost() && ((v_interrupts AND
                                 HCR_EL2.<AMO,IMO,FMO>) != '000');
    return pending_physical_interrupt || pending_virtual_interrupt;
```

is updated to read:

```
    boolean pending_virtual_interrupt = FALSE;
    boolean pending_physical_interrupt = (IRQPending() || FIQPending() ||
                                          IsPhysicalSErrorPending());
    if EL2Enabled() && PSTATE.EL IN {EL0, EL1} && HCR_EL2.TGE == '0' then
        boolean virq_pending = HCR_EL2.IMO == '1' && (VirtualIRQPending() ||
 HCR_EL2.VI == '1') ;
        boolean vfiq_pending = HCR_EL2.FMO == '1' && (VirtualFIQPending() ||
 HCR_EL2.VF == '1');
        boolean vsei_pending = HCR_EL2.AMO == '1' && (IsVirtualSErrorPending() ||
 HCR_EL2.VSE == '1');
        pending_virtual_interrupt = vsei_pending || virq_pending || vfiq_pending;
\t\t
\t\treturn pending_physical_interrupt || pending_virtual_interrupt;
```

2.131 D17834

In section H9.3.14 (CTICONTROL, CTI Control register), in the GLBEN field, the text:

> If a previously asserted output trigger has not been acknowledged, it remains asserted after the mapping functions are disabled.

is replaced with:

> If a previously asserted output trigger has not been acknowledged, it is **CONSTRAINED UNPREDICTABLE** which of the following occurs:
>
> - The output trigger remains asserted after the mapping functions are disabled.
> - The output trigger is deasserted after the mapping functions are disabled.

2.132 D17837

In section J1.2.2 (aarch32/exceptions), the pseudocode function AArch32.ReportDataAbort() now uses the correct DFSR format to report synchronous External aborts to Monitor mode.

The code that reads as:

```
    long_format = FALSE;
    if route_to_monitor && !IsSecure() then
        long_format = TTBCR_S.EAE == '1';
        if !IsSErrorInterrupt(fault) && !long_format then
            long_format = PSTATE.EL == EL2 || TTBCR.EAE == '1';
```

is updated to read:

```
    long_format = FALSE;
    if route_to_monitor && !IsSecure() then
        long_format = ((TTBCR_S.EAE == '1') ||
                       (IsExternalSyncAbort(fault) && ((PSTATE.EL == EL2 ||
  TTBCR.EAE == '1') ||
                        (fault.secondstage && boolean IMPLEMENTATION_DEFINED \"Stage
  2 synchronous external abort reports using Long-descriptor format when TTBCR_S.EAE
  is 0b0\"))));
```

2.133 D17859

In section G8.2.89 (ID_ISAR4, Instruction Set Attribute Register 4), in the SMC field, the text that reads:

> In Armv8-A, the only permitted value is `0b0001`.

is corrected to read:

> In Armv8-A, the permitted values are:
>
> - If EL3 is implemented, the only permitted value is `0b0001`.
> - If neither EL3 nor EL2 is implemented, the only permitted value is `0b0000`.

The equivalent change is made in section D13.2.76 (ID_ISAR4_EL1, AArch32 Instruction Set Attribute Register 4).

2.134 D17867

In section D5.10.2 (TLB maintenance instructions), in the subsection 'Ordering and completion of TLB maintenance instructions', the text that currently reads:

> A TLB maintenance operation without the nXS qualifier generated by a TLB maintenance instruction is finished for a PE when all memory accesses generated by that PE using in-scope old translation information are complete.
>
> A TLB maintenance operation with the nXS qualifier generated by a TLB maintenance instruction is finished for a PE when all memory accesses with the XS attribute set to 0 generated by that PE using in-scope old translation information are complete.

is enhanced to read:

> A TLB maintenance operation without the nXS qualifier generated by a TLB maintenance instruction is finished for a PE when:
>
> - All memory accesses generated by that PE using in-scope old translation information are complete.
> - All memory accesses RWx generated by that PE are complete. RWx is the set of all memory accesses generated by instructions for that PE that appear in program order before an instruction (I1) executed by that PE where:
> - I1 uses the in-scope old translation information.
> - The use of the in-scope old translation information generates a synchronous data abort.
> - If I1 did not generate an abort from use of the in-scope old translation information, I1 would generate a memory access that RWx would be locally-ordered-before. A TLB maintenance operation with the nXS qualifier generated by a TLB maintenance instruction is finished for a PE when:
> - All memory accesses with the XS attribute set to 0 generated by that PE using in-scope old translation information are complete.
> - All memory accesses RWx generated by that PE are complete. RWx is the set of all memory accesses generated by instructions for that PE that appear in program order before an instruction (I1) executed by that PE where:
> - I1 uses the in-scope old translation information.
> - The use of the in-scope old translation information generates a synchronous data abort.
> - If I1 did not generate an abort from use of the in-scope old translation information, I1 would generate a memory access with the XS attribute set to 0 that RWx would be locally-ordered-before.

The equivalent changes are made in section G5.9.1 (General TLB maintenance requirements).

2.135 D17870

In sections B2.3.3 (Ordering relations) and E2.3.3 (Ordering relations), under the heading 'Barrier-ordered-before', the clause that reads:

- RW_1 appears in program order before an atomic instruction with both Acquire and Release semantics that appears in program order before RW_2.

is changed to read:

- RW_1 is a memory write effect W_1 and is generated by an atomic instruction with both Acquire and Release semantics.

2.136 D17883

In section A1.5.4 (Flushing denormalized numbers to zero), in the subsection 'Flushing denormalized outputs to zero', the line that reads:

If FPCR.AH == 1, and if FPCR.FZ == 1, Advanced SIMD, floating-point and BF16 instructions, for single-precision, double-precision and BF16 outputs, the FPCR.FZ setting does not cause denormalized outputs to be flushed to zero, although other factors might cause denormalized outputs to be flushed to zero.

is corrected to read:

If FPCR.AH == 1, and if FPCR.FZ == 0, Advanced SIMD, floating-point and BF16 instructions, for single-precision, double-precision and BF16 outputs, the FPCR.FZ setting does not cause denormalized outputs to be flushed to zero, although other factors might cause denormalized outputs to be flushed to zero.

In section C5.2.7 (FPCR, Floating-point Control Register), the following changes are made:

The definition of the FIZ, bit [0] that currently reads:

`0b0` If FPCR.AH is 0, this bit is **RES0**. If FPCR.AH is 1, disables flushing to zero of inputs that are single-precision, double-precision, and BF16 denormalized numbers, other than for some instructions. For more information, see 'Flushing denormalized numbers to zero'.

`0b1` If FPCR.AH is 0, this bit is **RES0**. If FPCR.AH is 1, enables flushing to zero of inputs that are single-precision, double-precision, and BF16 denormalized numbers, other than for some instructions. For more information, see 'Flushing denormalized numbers to zero'.

is modified to read:

`0b0` The flushing to zero of single-precision and double-precision denormalized inputs to floating-point instructions not enabled by this control, but other factors might cause the input denormalized numbers to be flushed to zero.

> `0b1` Denormalized single-precision and double-precision inputs to most floating-point instructions flushed to zero.
>
> For more information, see 'Flushing denormalized numbers to zero' and the pseudocode of the floating-point instructions.

The definition of the FZ, bit [24] that currently reads:

> `0b0` If FPCR.AH is 0, disables flushing to zero of inputs and outputs that are single-precision, double-precision, and BF16 denormalized numbers, other than for some instructions. For more information, see 'Flushing denormalized numbers to zero'. If FPCR.AH is 1, disables flushing to zero of outputs that are single-precision, double-precision, and BF16 denormalized numbers, other than for some instructions. For more information, see 'Flushing denormalized numbers to zero'.
>
> `0b1` Flushing denormalized numbers to zero enabled. If FPCR.AH is 0, enables flushing to zero of inputs and outputs that are single-precision, double-precision, and BF16 denormalized numbers, other than for some instructions. For more information, see 'Flushing denormalized numbers to zero'. If FPCR.AH is 1, enables flushing to zero of outputs that are single-precision, double-precision, and BF16 denormalized numbers, other than for some instructions. For more information, see 'Flushing denormalized numbers to zero'.
>
> The value of this bit controls both scalar and Advanced SIMD floating-point arithmetic.

is modified to read:

> `0b0` If FPCR.AH is 0, the flushing to zero of single-precision and double-precision denormalized inputs to, and outputs of, floating-point instructions not enabled by this control, but other factors might cause the input denormalized numbers to be flushed to zero. If FPCR.AH is 1, the flushing to zero of single-precision and double-precision denormalized outputs of floating-point instructions not enabled by this control, but other factors might cause the input denormalized numbers to be flushed to zero.
>
> `0b1` If FPCR.AH is 0, denormalized single-precision and double-precision inputs to, and outputs from, floating-point instructions are flushed to zero. If FPCR.AH is 1, denormalized single-precision and double-precision outputs from floating-point instructions are flushed to zero.
>
> When FEAT_AFP is implemented, for more information see 'Flushing denormalized numbers to zero' and the pseudocode of the floating-point instructions.

The definition of the AH, bit [1] that currently reads:

> Alternate Handling. Controls alternate handling of denormalized floating-point numbers.
>
> `0b0` FPCR.FZ controls flushing to zero of inputs and outputs that are single-precision, double-precision, and BF16 denormalized numbers. FPCR.FIZ is **RES0**. For half-precision, single-precision, and double-precision numbers, the test for a denormalized number for the purpose of flushing the output to zero occurs before rounding. For more information, see 'Flushing denormalized numbers to zero'.

> `0b1` FPCR.FZ controls flushing to zero of outputs that are single-precision, double-precision, and BF16 denormalized numbers. For all precisions, the test for a denormalized number for the purpose of flushing the output to zero occurs after rounding with an unbounded exponent. FPCR.FIZ controls flushing to zero of inputs that are single-precision, double-precision, and BF16 denormalized numbers. Some instructions unconditionally flush to zero. For more information, see 'Flushing denormalized numbers to zero'. The AH bit affects the generation and operation of floating-point exceptions. For more information, see 'Floating-point exceptions and exception traps'.

is modified to read:

> Alternate Handling. Controls alternate handling of floating-point numbers.
>
> The Arm architecture supports two models for handing some of the corner cases of the floating-point behaviors, such as the nature of flushing of denormalized numbers, the detection of tininess and other exceptions and a range of other behaviors. The value of FPCR.AH bit selects between these models.
>
> For more information on the FPCR.AH bit, see 'Flushing denormalized numbers to zero', 'Floating-point exceptions and exception traps' and the pseudocode of the floating-point instructions.

2.137 R17892

In sections D11.2.3 (Event streams) and G6.2.3 (Event Streams), the following Note is added:

> Note: If the event stream is configured to produce events from the low order bits of the counter when the counter frequency is very high (for example 1GHz), then the practical update rate of the counter might mean that the event stream is not generated as the low order bit might not change. Software can rely on an event stream rate of at least 1MHz in normal operation.

2.138 D17896

In section D13.2.123 (TCR_EL1, Translation Control Register (EL1)), the text under the 'Otherwise' heading in the HWU* fields that reads:

> Reserved, **RES0**.

is corrected to read:

> Reserved, RAZ/WI.

Equivalent changes are made to HWU* fields in the following sections:

- D13.2.124 (TCR_EL2, Translation Control Register (EL2)),
- D13.2.125 (TCR_EL3, Translation Control Register (EL3)),
- D13.2.148 (VTCR_EL2, Virtualization Translation Control Register),

- G8.2.165 (TTBCR2, Translation Table Base Control Register 2),
- G8.2.171 (VTCR, Virtualization Translation Control Register),

and to the T2E field in section G8.2.164 (TTBCR, Translation Table Base Control Register).

2.139 E17909

In section A2.10.1 (Architectural features added by Armv8.7), the title 'FEAT_WFxT, WFE and WFI instructions with timeout' is updated to include FEAT_WFxT2, and the following text is added to the feature description:

> FEAT_WFxT2 adds a mechanism to report the register number that holds the timeout value in ESR_ELx for trapped WFET and WFIT instructions.
>
> FEAT_WFxT is mandatory in Armv8.7 implementations. FEAT_WFxT2 is OPTIONAL in Armv8.7 implementations.
>
> Note: Arm deprecates not implementing FEAT_WFxT2.
>
> The ID_AA64ISAR2_EL1.WFxT field identifies the presence of FEAT_WFxT and FEAT_WFxT2.

In section D13.2.64 (ID_AA64ISAR2_EL1), the 'WFxT, bits [3:0]' field is updated to read:

> Indicates support for the WFET and WFIT instructions in AArch64 state. Defined values are:
>
> `0b0000` WFET and WFIT are not supported.
>
> `0b0001` WFET and WFIT are supported, but the register number is not reported in the ESR_ELx on exceptions.
>
> `0b0010` WFET and WFIT are supported, and the register number is reported in the ESR_ELx on exceptions.
>
> All other values are reserved.
>
> FEAT_WFxT implements the functionality identified by the value `0b0001`.
>
> FEAT_WFxT2 implements the functionality identified by the value `0b0010`.
>
> From Armv8.7, the permitted values are `0b0001` and `0b0010`.
>
> Note: Arm deprecates not implementing FEAT_WFxT2.

Correspondingly, in sections D13.2.37 (ESR_EL1), D13.2.38 (ESR_EL2), and D13.2.39 (ESR_EL3), in the ISS encoding an exception from a WF* instruction, the following fields are added:

> RN, bits [9:5]

When FEAT_WFxT2 is implemented:

Indicates the Register Number supplied for a WFET or WFIT instruction.

The reset behavior of this field is:

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

Otherwise:

Reserved, RES0.

RV, bit [2]

When FEAT_WFxT2 is implemented:

Register field Valid.

If TI[1] == 1, then this field indicates whether RN holds a valid register number for the register argument to the trapped WFET or WFIT instruction.

`0b0` Register field invalid.

`0b1` Register field valid.

If TI[1] == 0, then this field is RES0.

When FEAT_WFxT2 is implemented, RV is set to 1 on a trap on WFET or WFIT.

When FEAT_WFxT2 is not implemented, RV is set to 0 on a trap on WFET or WFIT.

The reset behavior of this field is:

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

Otherwise:

Reserved, RES0.

Instances of 'FEAT_WFxT' in the Arm ARM are updated to read 'FEAT_WFxT or FEAT_WFxT2'.

2.140 C17911

In section D2.10.5 (Determining the memory location that caused a Watchpoint exception), in 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:

> - From the inclusive range between:
>   - The lowest address accessed by the memory access that triggered the watchpoint.
>   - The highest watchpointed address accessed by the memory access. A watchpointed address is an address that the watchpoint is watching.
> - Within a naturally-aligned block of memory that is all of the following:
>   - A power-of-two size.
>   - No larger than the DC ZVA block size.
>   - Contains a watchpointed address accessed by the memory access.

is clarified to read:

> The address recorded must be both:
> - From the inclusive range between:
>   - The lowest address accessed by the memory access or set of contiguous memory accesses made by the instruction that triggered the watchpoint.
>   - The highest watchpointed address accessed by the memory access or set of contiguous memory accesses made by the instruction that triggered the watchpoint. A watchpointed address is an address that the watchpoint is watching.
> - Within a naturally-aligned block of memory that is all of the following:
>   - A power-of-two size.
>   - No larger than the DC ZVA block size.
>   - Contains a watchpointed address accessed by the memory access or set of contiguous memory accesses made by the instruction that triggered the watchpoint.

Similar changes are updated to section G2.10.5 (Determining the memory location that caused a Watchpoint exception), in subsection 'Address recorded for Watchpoint exceptions generated by instructions other than data cache maintenance instructions'.

2.141 C17921

At the end of section B2.8 (Mismatched memory attributes), the following Note is added:

> Note: As described in <xref:D4.4.6 Non-cacheable accesses and instruction caches>, a non-cacheable access is permitted to be cached in an instruction cache, despite the fact that a non-cacheable access is not permitted to be cached in a unified cache. Despite this, when cacheable and non-cacheable aliases exist for memory which is executable, these must be treated as mismatched aliases to avoid coherency issues from the data or unified caches that might hold entries that will be brought into the instruction caches.

2.142 R17932

In section B2.7.2 (Device memory), the bullet point that reads:

> - All accesses to memory with any Device memory attribute must be aligned. Any unaligned access generates an Alignment fault at the first stage of translation that defined the location as being Device.

is relaxed to read:

> - If a memory location is not capable of supporting unaligned memory accesses, then an unaligned access to that memory location generates an Alignment fault at the first stage of translation that defined the location as being Device.
> - If a memory location is capable of supporting unaligned memory accesses, then it is **IMPLEMENTATION DEFINED** whether, if such a memory location is marked as Device, then an unaligned access to that memory location generates an Alignment fault at the first stage of translation that defined the location as being Device.

Additionally, the following text is added to sections B2.2.6 (Possible implementation restrictions on using atomic instructions), B2.9.2 (Exclusive access instructions and Shareable memory locations) and D5.4.14 (Restriction on memory types for hardware updates on translation tables), after the bullet points that specify 'Inner/Outer Shareable, Inner Write-Back, Outer Write-Back Normal memory with Read allocation hints and Write allocation hints and not transient' memory types:

> The architecture only requires that conventional memory that is mapped in this way supports this functionality.

2.143 D17934

In section A2.2.1 (Additional functionality added to Armv8.0 in later releases), the title 'FEAT_SSBS, Speculative Store Bypass Safe' is updated to 'FEAT_SSBS, FEAT_SBSS2, Speculative Store Bypass Safe'. In the description of FEAT_SBSS and FEAT_SBSS2, the text that reads:

> FEAT_SSBS allows software to indicate whether hardware is permitted to load or store speculatively in a manner that could give rise to a cache timing side channel, which in turn could be used to derive an address from values loaded to a register from memory. To do this, the software sets the PSTATE.SSBS.
>
> This feature is supported in both AArch64 and AArch32 states.

is updated to read:

> FEAT_SSBS allows software to indicate whether hardware is permitted to load or store speculatively in a manner that could give rise to a cache timing side channel, which in turn could be used to derive an address from values loaded to a register from memory.

> FEAT_SSBS2 provides controls for the MSR and MRS instructions to read and write the PSTATE.SSBS field.
>
> FEAT_SSBS is supported in both AArch64 and AArch32 states. FEAT_SSBS2 is supported in AArch64 state only.

and the text that reads:

> The following fields identify the presence of FEAT_SSBS:

is updated to read:

> The following fields identify the presence of FEAT_SSBS and FEAT_SSBS2:

2.144 D17935

In section G8.2.167 (TTBR1, Translation Table Base Register 1), the accessibility pseudocode for write to Secure TTBR1 at EL3 is corrected.

The MCR pseudocode at EL3:

```
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' && CP15SDISABLE == HIGH then
        UNDEFINED;
    elsif SCR.NS == '0' && CP15SDISABLE2 == HIGH then
        UNDEFINED;
    else
        if SCR.NS == '0' then
            TTBR1_S = ZeroExtend(R[t]);
        else
            TTBR1_NS = ZeroExtend(R[t]);
```

is updated to:

```
elsif PSTATE.EL == EL3 then
    elsif SCR.NS == '0' && CP15SDISABLE2 == HIGH then
        UNDEFINED;
    else
        if SCR.NS == '0' then
            TTBR1_S = ZeroExtend(R[t]);
        else
            TTBR1_NS = ZeroExtend(R[t]);
```

The MCRR pseudocode at EL3:

```
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' && CP15SDISABLE == HIGH then
        UNDEFINED;
    else
        if SCR.NS == '0' then
            TTBR1_S = R[t2]:R[t];
        else
            TTBR1_NS = R[t2]:R[t];
```

is updated to:

```
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' && CP15SDISABLE2 == HIGH then
        UNDEFINED;
    else
        if SCR.NS == '0' then
            TTBR1_S = R[t2]:R[t];
        else
            TTBR1_NS = R[t2]:R[t];
```

2.145 C17938

In section C5.6.1 (CFP RCTX, Control Flow Prediction Restriction by Context), the following line is added to the VMID, bits [47:32] field:

> If the implementation supports 16 bits of VMID, then the upper 8 bits of the VMID must be written to 0 by software when the context being affected only uses 8 bits.

And the following line is added to the ASID, bits [15:0] field:

> If the implementation supports 16 bits of ASID, then the upper 8 bits of the ASID must be written to 0 by software when the context being affected only uses 8 bits.

The same changes are applied to sections C5.6.2 (CPP RCTX, Cache Prefetch Prediction Restriction by Context) and C5.6.3 (DVP RCTX, Data Value Prediction Restriction by Context).

2.146 D17943

In section C7.2.227 (SABDL, SABDL2), the sentence that reads:

> The SABDL instruction writes the vector to the lower half of the destination register and clears the upper half, while the SABDL2 instruction writes the vector to the upper half of the destination register without affecting the other bits of the register.

is corrected to read:

> The SABDL instruction extracts each source vector from the lower half of each source register, while the SABDL2 instruction extracts each source vector from the upper half of each source register.

2.147 D17944

In section C7.2.397 (USUBW, USUBW2), the sentence that reads:

> All the values in this instruction are signed integer values.

is corrected to read:

> All the values in this instruction are unsigned integer values.

2.148 D17945

In section B2.3.10 (Memory barriers), in subsection 'Data Synchronization Barrier (DSB)', the following bullet is deleted:

> - If the required access types of the DSB is reads and writes, then all cache maintenance instructions, all TLB maintenance instructions, and all PSB CYNC instructions issued by PEe before the DSB are complete for the required shareability domain.

2.149 D17948

The following changes are made:

> - In section I5.2.1 (Performance Monitors external register views), an entry for PMMIR is added to table I5-1, 'Performance Monitors external register views'.
> - In section I5.3.30 (PMMIR, Performance Monitors Machine Identification Register), BUS_WIDTH and BUS_SLOTS fields are added at bits [19:16] and [15:8] respectively.

2.150 D17956

In section K11.3.3 (Ticket Locks), the text that currently reads:

> Releasing the ticket lock simply involves incrementing the current ticket number, that is still assumed to be in R3, and doing a Store-Release:

is corrected to read:

> Releasing the ticket lock simply involves incrementing the current ticket number, which is assumed in this example to be in R6, and doing a Store Release:

Within the same section, the AArch64 code that reads:

```
ADD W5, W5, #0x10000 ; increment the next number
STXR W6, W5, [X1] ; and update the value
```

is corrected to read:

```
ADD W3, W5, #0x10000 ; increment the next number
STXR W6, W3, [X1] ; and update the value
```

Similarly, the AArch32 code within the same section that reads:

```
ADD R5, R5, #0x10000 ; increment the next number
STREX R6, R5, [R1] ; and update the value
```

is corrected to read:

```
ADD R3, R5, #0x10000 ; increment the next number
STREX R6, R3, [R1] ; and update the value
```

The AArch32 code within the same section that reads:

```
BEQ block_start
```

is enhanced to read:

```
MOV R6, R5
BEQ block_start
```

The equivalent changes for this enhancement are made in section K11.3.4 (Use of Wait For Event (WFE) and Send Event (SEV) with locks), in the AArch32 code within the subsection 'Ticket lock'.

2.151 D17958

In section J1.1.1 (aarch64/debug), the sense of the SPME bit when used in conjunction with MPMX is flipped.

In AArch64.CountEvents, the code that reads:

```
prohibited = (MDCR_EL3.MPMX == '1' && (MDCR_EL3.SPME == '1' || !resvd_for_el2));
```

is updated to read:

```
prohibited = (MDCR_EL3.MPMX == '1' && (MDCR_EL3.SPME == '0' || !resvd_for_el2));
```

2.152 D17975

In section J1.1.4 (aarch64/instrs), in the function AArch64.TLBI_RVA() the following line:

```
r.address.NS  = if security == SS_NonSecure then '1' else Xt<63>;
```

is removed.

2.153 R17983

The following bits are relaxed to **RES0**, to align with the equivalent bits in the AArch64 and AArch32 registers:

- Section I5.5.11 (AMCR, Activity Monitors Control Register), Bits [9:0],
- Section I5.5.18 (AMEVTYPER0<n>, Activity Monitors Event Type Registers 0, n = 0 - 15), Bits [31:25],
- Section I5.5.19 (AMEVTYPER1<n>, Activity Monitors Event Type Registers 1, n = 0 - 15), Bits [31:25].

2.154 C17986

In section I5.5.7 (AMCNTENCLR0, Activity Monitors Count Enable Clear Register 0), the text that reads:

> `0b0` When read, means that AMEVCNTR0<n> is disabled. When written, has no effect. `0b1` When read, means that AMEVCNTR0<n> is enabled. When written, disables AMEVCNTR0<n>.

is clarified to read:

> `0b0` When read, means that AMEVCNTR0<n> is disabled.
>
> `0b1` When read, means that AMEVCNTR0<n> is enabled.

The equivalent changes are made in the following sections:

- I5.5.8 (AMCNTENCLR1, Activity Monitors Count Enable Clear Register 1),
- I5.5.9 (AMCNTENSET0, Activity Monitors Count Enable Set Register 0),
- I5.5.10 (AMCNTENSET1, Activity Monitors Count Enable Set Register 1).

In the following sections, instances of 'RAZ/WI' in 'Accessing the [register]' are changed to 'RAZ':

- I5.5.8 (AMCNTENCLR1, Activity Monitors Count Enable Clear Register 1),
- I5.5.10 (AMCNTENSET1, Activity Monitors Count Enable Set Register 1),
- I5.5.16 (AMEVCNTR0<n>, Activity Monitors Event Counter Registers 0, n = 0 - 15),
- I5.5.17 (AMEVCNTR1<n>, Activity Monitors Event Counter Registers 1, n = 0 - 15),
- I5.5.18 (AMEVTYPER0<n>, Activity Monitors Event Type Registers 0, n = 0 - 15),
- I5.5.19 (AMEVTYPER1<n>, Activity Monitors Event Type Registers 1, n = 0 - 15).

The change from 'RAZ/WI' to 'RAZ' is also applied to the P<n> field descriptions in AMCNTENCLR1 and AMCNTENSET1.

In sections I5.5.16 (AMEVCNTR0<n>, Activity Monitors Event Counter Registers 0, n = 0 - 15) and I5.5.17 (AMEVCNTR1<n>, Activity Monitors Event Counter Registers 1, n = 0 - 15), the following text in the ACNT, bits [63:0] description is removed:

> If the counter is enabled, writes to this register have UNPREDICTABLE results.

In section I5.5.19 (AMEVTYPER1<n>, Activity Monitors Event Type Registers 1, n = 0 - 15), the following text in the evtCount, bits [15:0] field is removed:

> If software writes a value to this field which is not supported by the corresponding counter AMEVCNTR1<n>, then:
>
> - It is UNPREDICTABLE which event will be counted.
> - The value read back is **UNKNOWN**.
>
> Note: The event counted by AMEVCNTR1<n> might be fixed at implementation. In this case, the field is read-only and writes are UNDEFINED.
>
> If the corresponding counter AMEVCNTR1<n> is enabled, writes to this register have UNPREDICTABLE results.

2.155 D17991

In section C5.3.29 (IC IALLU, Instruction Cache Invalidate All to PoU), the line that reads:

> Purpose
>
> Invalidate all instruction caches to Point of Unification.

is corrected to read:

> Purpose
>
> Invalidate all instruction caches of the PE executing the instruction to the Point of Unification.

In section C5.3.30 (IC IALLUIS, Instruction Cache Invalidate All to PoU, Inner Shareable), the text that reads:

> Purpose
>
> Invalidate all instruction caches in Inner Shareable domain to Point of Unification.

is corrected to read:

> Purpose
>
> Invalidate all instruction caches in the Inner Shareable domain of the PE executing the instruction to the Point of Unification.

Equivalents edits are made in sections G8.2.79 (ICIALLU, Instruction Cache Invalidate All to PoU) and G8.2.80 (ICIALLUIS, Instruction Cache Invalidate All to PoU, Inner Shareable).

2.156 D17998

In section B2.3.9 (Restrictions on the effects of speculation), in the subsection 'Restrictions on the effects of speculation', the text that reads:

> - When data is loaded under speculation with a translation fault, it cannot be used to form an address, generate condition codes, or generate SVE predicate values to be used by instructions newer than the load in the speculative sequence.
> - When data is loaded under speculation from a location without a translation for the translation regime being speculated in, the data cannot be used to form an address, generate condition codes, or generate SVE predicate values to be used by instructions newer than the load in the speculative sequence.

is corrected to read:

> - When data is loaded under speculation with a translation fault, it cannot be used to form an address, generate condition codes, or generate SVE predicate values to be used by other instructions in the speculative sequence.
> - When data is loaded under speculation from a location without a translation for the translation regime being speculated in, the data cannot be used to form an address, generate condition codes, or generate SVE predicate values to be used by other instructions in the speculative sequence.

In the subsection 'Restrictions on the effects of speculation from Armv8.5', the text that reads:

> - 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 instructions newer than the load 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 instructions newer than the load in the speculative sequence.

is corrected to read:

> - 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.

The equivalent changes are made in section E2.3.9 (Restrictions on the effects of speculation), and in the definitions of the CSV3 field in sections D13.2.67 (ID_AA64PFR0_EL1, AArch64 Processor Feature Register 0), D13.2.87 (ID_PFR2_EL1, AArch32 Processor Feature Register 2), and G8.2.100 (ID_PFR2, Processor Feature Register 2).

2.157 D18001

In section D2.10.6 (Watchpoint behavior on other instructions), the Note that reads:

> Note: Despite its mnemonic, the DC ZVA, Data Cache Zero by VA instruction is not a data cache maintenance instruction.

is clarified to read:

> Note: Despite their mnemonics, the DC GVA, DC GZVA, and DC ZVA instructions are not data cache maintenance instructions.

The equivalent Notes in sections D2.10.5 (Determining the memory location that caused a Watchpoint exception) and D4.4.8 (A64 Cache maintenance instructions), subsection 'The data cache maintenance instruction (DC)', are similarly clarified.

The following changes are made in section D4.4.8 (A64 Cache maintenance instructions), subsection 'Ordering and completion of data and instruction cache instructions':

- The references to 'data cache instructions, other than DC ZVA' are clarified to 'data cache instructions, other than DC ZVA, GC GVA, and DC GZVA'.
- In the list for all data cache maintenance instructions that do not specify an address, the reference 'other than Data Cache Zero' is clarified to 'other than DC ZVA, GC GVA, and DC GZVA'.

In section D5.4.2 (About PSTATE.PAN), the following item in the list of instructions that the PAN bit does not affect:

> Data Cache instructions other than DC ZVA.

is clarified to read:

> Data cache instructions other than DC GVA, DC GZVA, and DC ZVA.

2.158 D18003

In section J1.1.5 (aarch64/translation), AArch64.S2Translate() which reads:

```
AArch64.S2Translate()
    ...
    if walkparams.vm != '1' then
        // Stage 2 translation is disabled
        fault.level = 0;
        if AArch64.S2HasAlignmentFault(acctype, aligned, ipa.memattrs) then
            fault.statuscode = Fault_Alignment;
            return (fault, AddressDescriptor UNKNOWN);
        else
            return (fault, ipa);
```

is updated to read:

```
AArch64.S2Translate()
    ...
    if walkparams.vm != '1' then
        // Stage 2 translation is disabled
        return (fault, ipa);
```

2.159 D18009

In section J1.1.4 (aarch64/instrs), the type TLBIRecord which reads:

```
type TLBIRecord is (
    ...
    FullAddress     address,      // VA/IPA/BaseAddress
    ...
)
```

is updated to read:

```
type TLBIRecord is (
    ...
    PASpace         ipaspace,     // For operations that take IPA as input address
    bits(64)        address,      // input address, for range operations, start ad\
dress
    ...
)
```

In section J1.1.4 (aarch64/instrs), for the functions AArch64.TLBI_VA() and AArch64.TLBI_VAA(), the line that reads:

```
r.address.address = Xt<39:0> : Zeros(12);
```

is updated to read:

```
r.address = ZeroExtend(Xt<43:0> : Zeros(12));
```

2.160 D18022

In section H2.4.2 (Executing instructions in Debug state), in the subsection 'A64 instructions that are **CONSTRAINED UNPREDICTABLE** in Debug state', the entry 'Instructions that request entry to a low-power state' is augmented to include WFET and WFIT, and the entry 'Instructions with register argument' is deleted.

2.161 D18023

In section J1.1.4 (aarch64/instrs), in the functions AArch64.TLBI_RVA(), AArch64.TLBI_RVAA(), and AArch64.TLBI_RIPAS2() the line that reads:

```
r.ttl = Xt<47:44>;
```

is corrected to read:

```
r.ttl = Xt<38:37>;
```

2.162 D18031

In section D13.6.1 (PMBIDR_EL1, Profiling Buffer ID Register), in the long description of the PMBIDR_EL1.P 'Programming not allowed' bit, the text that reads:

> Programming not allowed. The Profiling Buffer is owned by a higher Exception level or the other Security state.
>
> `0b0` Profiling Buffer is owned by the current or a lower Exception level in the current Security state.
>
> `0b1` Profiling Buffer is owned by a higher Exception level or the other Security state.

is corrected to read:

> Programming not allowed. When read at EL3, this field reads as zero. Otherwise, indicates that the Profiling Buffer is owned by a higher Exception level or another Security state. Defined values are:
>
> `0b0` Programming is allowed.
>
> `0b1` Programming not allowed.

2.163 D18034

In section D7.10.3 (Common event numbers), the Note that reads:

> The requirement that an event that is implemented retrospectively does not require additional features in the PMU means that it must be possible to represent the event n the

> PMEVTYPER<n>_EL0.evtCount field. This means, for example, that an implementation with a 12-bit PMEVTYPER<n>_EL0.evtCount field can only implement events with event numbers `0x000-0xFFF`.

is corrected to read:

> The requirement that an event that is implemented retrospectively does not require additional features in the PMU means that it must be possible to represent the event n the PMEVTYPER<n>_EL0.evtCount field. This means, for example, that an implementation with a 10-bit PMEVTYPER<n>_EL0.evtCount field can only implement events with event numbers `0x0000-0x03FF`.

2.164 D18037

In section D9.6.5 (Additional information for each profiled Scalable Vector Extension operation), the bullet point in the Note that reads:

> - FADDV, and SMAX write scalar values to SIMD&FP registers.

is corrected to read:

> - FADDV, and SMAXV write scalar values to SIMD&FP registers.

2.165 D18060

In section D7.10.3 (Common event numbers), in the subsection 'Common microarchitectural events', the following events are deleted, and the event numbers are reserved:

- `0x8125`, BUS_ACCESS_RD_PERCYC, Event in progress, BUS_ACCESS_RD.
- `0x8126`, BUS_ACCESS_WR_PERCYC, Event in progress, BUS_ACCESS_WR.
- `0x8127`, BUS_ACCESS_PERCYC, Event in progress, BUS_ACCESS.

To align with these changes, in section D7.10.2 (The PMU event number space and common events), in Table D7-6 'Allocation of the PMU event number space', the entry that reads:

Event numbers	Allocation
`0x8100-0x81FF`	Previously reserved. From Armv8.6, Common architectural and microarchitectural events.

is split into the following three entries:

Event numbers	Allocation
`0x8100-0x8124`	Previously reserved. From Armv8.6, common architectural and microarchitectural events.
`0x8125-0x8127`	Reserved.
`0x8128-0x81FF`	Previously reserved. From Armv8.6, common architectural and microarchitectural events.

2.166 D18100

In section D4.4.8 (A64 Cache maintenance instructions) in Table D4-3 System instructions for cache maintenance), the following forms of cache maintenance instructions are added:

System Instruction	Instruction	Notes
DC CVADP	Clean by Virtual Address to point of Deep Persistence	When SCTLR_EL1.UCIa == 1, EL0 access.
DC CIGVAC	Clean and invalidate of allocation tags by virtual address to Point of Coherency	When SCTLR_EL1.UCIa == 1, EL0 access.
DC CIGDVAC	Clean and invalidate of data and Allocation Tags by virtual address to Point of Coherency	When SCTLR_EL1.UCIa == 1, EL0 access.
DC IGVAC	Invalidate of Allocation Tags by virtual address to Point of Coherency	EL1 or higher access.
DC IGDVAC	Invalidate of data and Allocation Tags by virtual address to Point of Coherency	EL1 or higher access.
DC CGVAC	Clean of Allocation Tags by virtual address to Point of Coherency	When SCTLR_EL1.UCIa == 1, EL0 access.
DC CGDVAC	Clean of data and Allocation Tags by virtual address to Point of Coherency	When SCTLR_EL1.UCIa == 1, EL0 access.
DC CGVAP	Clean of Allocation Tags by virtual address to Point of Persistence	When SCTLR_EL1.UCIa == 1, EL0 access.
DC CGDVAP	Clean of data and Allocation Tags by virtual address to Point of Persistence	When SCTLR_EL1.UCIa == 1, EL0 access.
DC CGVADP	Clean of Allocation Tags by virtual address to Point of Deep Persistence	When SCTLR_EL1.UCIa == 1, EL0 access.
DC CGDVADP	Clean of data and Allocation Tags by virtual address to Point of Deep Persistence	When SCTLR_EL1.UCIa == 1, EL0 access.

2.167 D18106

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

> If no instruction has been retired since the PE left Debug state, Reset state, or a state where PC Sample-based profiling is prohibited, the sampled value is **UNKNOWN**. If an instruction has been retired but this is the first time the PMPCSR or EDPCSR is read since the PE left Reset state, the sampled value is permitted but not required to return the value `0xFFFFFFF`.

is relaxed to read:

> If no branch instruction has been retired since the PE left Debug state, Reset state, or a state where PC Sample-based profiling is prohibited, the sampled value is **UNKNOWN**. If a branch instruction has been retired but this is the first time the PMPCSR or EDPCSR is read since the PE left Reset state, the sampled value is permitted but not required to return the value `0xFFFFFFF`.

Similarly, in section H9.2.32 (EDPCSR, External Debug Program Counter Sample Register), the text in the Bits [31:0] description that reads:

> If an instruction has retired since the PE left Reset state, then the first read of EDPSCR[31:0] is permitted but not required to return `0xFFFFFFFF`. EDPCSRlo reads as an **UNKNOWN** value when any of the following are true:
>
> - The PE is in Reset state.
> - No instruction has retired since the PE left Reset state, Debug state, or a state where PC Sample-based Profiling is prohibited.
> - No instruction has retired since the last read of EDPCSR[31:0].

is relaxed to read:

> If a branch instruction has retired since the PE left Reset state, then the first read of EDPSCR[31:0] is permitted but not required to return `0xFFFFFFFF`. EDPCSRlo reads as an **UNKNOWN** value when any of the following are true:
>
> - The PE is in Reset state.
> - No branch instruction has retired since the PE left Reset state, Debug state, or a state where PC Sample-based Profiling is prohibited.
> - No branch instruction has retired since the last read of EDPCSR[31:0].

The equivalent changes are made in section I5.3.33 (PMPCSR, Program Counter Sample Register).

2.168 D18112

In section B2.3.9 (Restrictions on the effects of speculation), in the subsection 'Speculative Store Bypass Safe (SSBS)', the text that reads:

> When FEAT_SSBS is implemented, PSTATE.SSBS is a control that can be set by software to indicate whether hardware is permitted to load or store speculatively, in a manner that could be exploited to produce a cache timing side channel using an address derived from a register value that has been loaded from memory using a load instruction that speculatively read an entry for the location being loaded from, where the entry that is speculatively read is from earlier in the coherence order than the entry generated by the latest store to that location using the same virtual address as the load instruction.
>
> When the value of PSTATE.SSBS is 0, hardware is not permitted to load or store speculatively in this way.
>
> When the value of PSTATE.SSBS is 1, hardware is permitted to load or store speculatively in this way.

is clarified to read:

> When FEAT_SSBS is implemented, PSTATE.SSBS is a control that can be set by software to indicate whether hardware is permitted to use, in a manner that is potentially speculatively exploitable, a speculative value in a register that has been loaded from memory using a load instruction that speculatively read the location being loaded from, where the entry that is speculatively read is from earlier in the coherence order than the entry generated by the latest store to that location using the same virtual address as the load instruction.
>
> A speculative value in a register is used in a potentially speculatively exploitable manner if it is used to form an address, generate condition codes, or generate SVE predicate values to be used by other instructions in the speculative sequence or if the execution timing of any other instructions in the speculative sequence is a function of the data loaded under speculation.
>
> When the value of PSTATE.SSBS is 0, hardware is not permitted to use speculative register values in a potentially speculatively exploitable manner if the speculative read that loads the register is from earlier in the coherence order than the entry generated by the latest store to that location using the same virtual address as the load instruction.
>
> When the value of PSTATE.SSBS is 1, hardware is permitted to use speculative register values in a potentially speculatively exploitable manner if the speculative read that loads the register is from earlier in the coherence order than the entry generated by the latest store to that location using the same virtual address as the load instruction.

The clarification is also made to the equivalent text in section E2.3.9 (Restrictions on the effects of speculation), in the subsection 'Speculative Store Bypass Safe (SSBS)'.

2.169 D18119

The text in section B2.7.3 (Memory access restrictions) that reads:

> For accesses to any two bytes, p and q, that are generated by the same instruction:
>
> - The bytes p and q must have the same memory type and Shareability attributes, otherwise the results are **CONSTRAINED UNPREDICTABLE**. For example, an LD1, ST1, or an unaligned load or store that spans the boundary between Normal memory and Device memory is **CONSTRAINED UNPREDICTABLE**.
> - Except for possible differences in the cache allocation hints, Arm deprecates having different cacheability attributes for bytes p and q.

is clarified to read:

> For two explicit memory reads to any two adjacent bytes in memory, p and p+1, generated by the same instruction, and for two explicit writes to any two adjacent bytes in memory, p and p+1, that are generated by the same instruction:
>
> - The bytes p and p+1 must have the same memory type and Shareability attributes, otherwise the results are **CONSTRAINED UNPREDICTABLE**. For example, an LD1, ST1, or an unaligned load or store that spans the boundary between Normal memory and Device memory is **CONSTRAINED UNPREDICTABLE**.

> - Except for possible differences in the cache allocation hints, Arm deprecates having different cacheability attributes for bytes p and p+1.

The clarification is also made to the equivalent text in section E2.8.3 (Memory access restrictions).

2.170 D18138

In section C7.2.146 (FRECPX), the text that reads:

> This instruction finds an approximate reciprocal exponent for each vector element in the source SIMD&FP register, places the result in a vector, and writes the vector to the destination SIMD&FP register.

is replaced by the following text:

> This instruction finds an approximate reciprocal exponent for the source SIMD&FP register and writes the result to the destination SIMD&FP register.

2.171 D18140

In sections B2.3.8 (Ordering of instruction fetches) and E2.3.8 (Ordering of instruction fetches), the text that reads:

> For two memory locations A and B, if A has been written to and been made coherent with the instruction fetches of the shareability domain, before an update to B by an observer in the same shareability domain, then the instruction stream of each observer in the shareability domain will not see the updated value of B without also seeing the updated value of A.

is corrected to read:

> For two memory locations A and B:
>
> If A has been written to with an updated value and been made coherent with the instruction fetches of the shareability domain, before B has been written to with an updated value by an observer in the same shareability domain, then where, for an observer in the shareability domain, an instruction read from B appears in program order before an instruction fetched from A, if the instruction read from B contains the updated value of B then the instruction read from A appearing later in program order will contain the updated value of A.

2.172 D18141

In sections B2.3.10 (Memory barriers), in the subsection 'Data Memory Barrier (DMB)', the following line is deleted:

> A DMB instruction intended to ensure the completion of cache maintenance instructions must have an access type of both loads and stores.

In section G4.4.7 (AArch32 cache and branch predictor maintenance instructions), the following line has references to DMB removed:

> A DSB or DMB instruction intended to ensure the completion of cache maintenance instructions or branch predictor instructions must have an access type of both loads and stores.

2.173 C18142

In section D4.4.8 (A64 Cache maintenance instructions) in the subsection 'Ordering and completion of data and instruction cache instructions', a new sub-subsection is added, titled 'Ordering and completion of Data Cache Clean to Point of Persistence', as follows:

> The update of the persistent memory as a result of Data Cache Clean to the Point of Persistence is guaranteed to have occurred either after:
>
> - The execution of a DSB applying to both reads and writes after the execution of the Data Cache Clean to the Point of Persistence.
> - The update to persistent memory caused by a different Data Cache Clean to the Point of Persistence that is ordered after a DMB applying to both reads and writes that appears after the original Data Cache Clean to the Point of Persistence.
>
> Note: This second point is an aspect of the fact that the Data Cache Clean to the Point of Persistence instructions are ordered by DMB, and this controls the order of arrival in persistent memory.
>
> Note: The ordering effect for the Data Cache Clean to the Point of Persistence by DMB applying to both read and writes is not sufficient to ensure that in the following sequence, observation of the value '1' in the memory location X3 implies that the Data Cache Clean to the Point of Persistence has caused an update of persistent memory:
>
> ```
> ; initial conditions has [X3]=0.
> DC CVAP, X1
> DMB
> MOV X2,#1
> STR X2, [X3]
> ```

> However, in the following example, the ordering effects of the DMB instruction will ensure that the location pointed by P0:X1 will reach the Point of Persistence before, or at the same time as, the location pointed by P1:X8.
>
> ```
> ; initial conditions has P0:X3 and P1:X3 point to the same location, which is 0 at
>  the start of this example
> P0:
>    DC CVAP, X1
>    DMB
>    MOV X2, #1
>    STR X2, [X3]
>
> P1
> loop
>     LDR X2, [X3]
>     CBZ X2, loop
>     DMB
>     DC CVAP, X8
> ```

The following sub-subsection, titled 'Ordering and completion of Data Cache Clean to the Point of Deep Persistence', is also added:

> The update of the deep persistent memory as a result of Data Cache Clean to the Point of Deep Persistence is guaranteed to have occurred either after:
>
> - The execution of a DSB applying to both reads and writes after the execution of the Data Cache Clean to the Point of Deep Persistence.
> - The update to persistent memory caused by a different Data Cache Clean to the Point of Deep Persistence that is ordered after a DMB applying to both reads and writes that appears after the original Data Cache Clean to the Point of Deep Persistence.
>
> Note: This second point is an aspect of the fact that the Data Cache Clean to the Point of Deep Persistence instructions are ordered by DMB, and this controls the order of arrival in deep persistent memory.
>
> Note: The ordering effect for the Data Cache Clean to the Point of Deep Persistence by DMB applying to both read and writes is not sufficient to ensure that in the following sequence, observation of the value '1' in the memory location X3 implies that the Data Cache Clean to the Point of Deep Persistence has caused an update of deep persistent memory:
>
> ```
> ; initial conditions has [X3]=0.
> DC CVADP, X1
> DMB
> MOV X2,#1
> STR X2, [X3]
> ```
>
> However, in the following example, the ordering effects of the DMB instruction will ensure that the location pointed by P0:X1 will reach the Point of Deep Persistence before, or at the same time as, the location pointed by P1:X8.
>
> ```
> ; initial conditions has P0:X3 and P1:X3 point to the same location, which is 0 at
>  the start of this example
> P0:
>    DC CVADP, X1
>    DMB
> ```

```
    MOV X2, #1
    STR X2, [X3]

P1
loop
     LDR X2, [X3]
     CBZ X2, loop
     DMB
     DC CVADP, X8
```

2.174 D18149

In section J1.3.3 (shared/functions), the function FPRoundBase() that reads as:

```
    // Deal with flush-to-zero before rounding if FPCR.AH != '1'.
    if (!altfp && ((fpcr.FZ == '1' && N != 16) || (fpcr.FZ16 == '1' && N == 16)) &&
        exponent < minimum_exp) then
        // Flush-to-zero never generates a trapped exception.
        if UsingAArch32() then
            FPSCR.UFC = '1';
        else
            FPSR.UFC = '1';
        return FPZero(sign);
```

is updated to read as:

```
     // Deal with flush-to-zero before rounding if FPCR.AH != '1'.
    if (!altfp && ((fpcr.FZ == '1' && N != 16) || (fpcr.FZ16 == '1' && N == 16)) &&
        exponent < minimum_exp) then
        // Flush-to-zero never generates a trapped exception.
        if UsingAArch32() then
            FPSCR.UFC = '1';
        else
            if fpexc then FPSR.UFC = '1';
        return FPZero(sign);
```

2.175 D18159

In section J1.3.3 (shared/functions), the function GenMPAMcurEL() which reads:

```
MPAMinfo GenMPAMcurEL(AccType acctype)
    ...
    PARTIDspaceType pspace;
    ...
        otherwise
            // other access types are DATA accesses
            InD = FALSE;
    pspace = PARTIDspaceFromSS(security);
    if !validEL then
    ...
    return genMPAM(mpamEL, InD, pspace);
```

is updated to read:

```
MPAMinfo GenMPAMcurEL(AccType acctype)
    ...
    PARTIDspaceType pspace = PARTIDspaceFromSS(security);
    if pspace == PIdSpace_NonSecure && !MPAMisEnabled() then
        return DefaultMPAMinfo(pspace);
    ...
        otherwise
            // other access types are DATA accesses
            InD = FALSE;
    if !validEL then
    ...
    if !MPAMisEnabled() then
        return DefaultMPAMinfo(pspace);
    else
        return genMPAM(mpamEL, InD, pspace);
```

2.176 D18160

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

```
// PMCR_EL0.DP disables the cycle counter when event counting is prohibited
if enabled && prohibited && n == 31 then
    enabled = PMCR_EL0.DP == '0';
```

is corrected to read:

```
// PMCR_EL0.DP disables the cycle counter when event counting is prohibited
if prohibited && n == 31 then
    enabled = enabled && PMCR_EL0.DP == '0';
    prohibited = FALSE;     // Otherwise whether event counting is prohibited does
 not affect the cycle counter
```

A similar correction is made in section J1.2.1 (aarch32/debug) to AArch32.CountEvents().

2.177 D18162

In section D10.2.6 (Events packet), the text in 'E[17], byte 2 bit [17], when SZ == `0b10`, or SZ == `0b11`' that reads:

> If PMUv3 and The Scalable Vector Extension (SVE) are implemented this Event is required to be implemented consistently with SVE_PRED_EMPTY_SPEC and SVE_PRED_PARTIAL_SPEC in the Arm Architecture Reference Manual Supplement, the Scalable Vector Extension, for v8-A.

is corrected to read:

> If PMUv3 and The Scalable Vector Extension (SVE) are implemented, this Event is required to be implemented consistently with SVE_PRED_NOT_FULL_SPEC [link to PMU event `0x8079`].

Similarly, the text in 'E[18], byte 2 bit [18], when SZ == `0b10`, or SZ == `0b11`' that reads:

> If PMUv3 and The Scalable Vector Extension (SVE) are implemented this Event is required to be implemented consistently with SVE_PRED_EMPTY_SPEC in the Arm Architecture Reference Manual Supplement, the Scalable Vector Extension, for v8-A.

is clarified to read:

> If PMUv3 and The Scalable Vector Extension (SVE) are implemented, this Event is required to be implemented consistently with SVE_PRED_EMPTY_SPEC [link to PMU event `0x8075`].

2.178 D18165

In section D5.9.1 (Use of ASIDs and VMIDs to reduce TLB maintenance requirements), in the subsection 'VMID size', the line that reads:

> When the value of VTCR_EL2.VS is 0, VMID[63:56]:

is corrected to read:

> When the value of VTCR_EL2.VS is 0, VTTBR_EL2[63:56]: