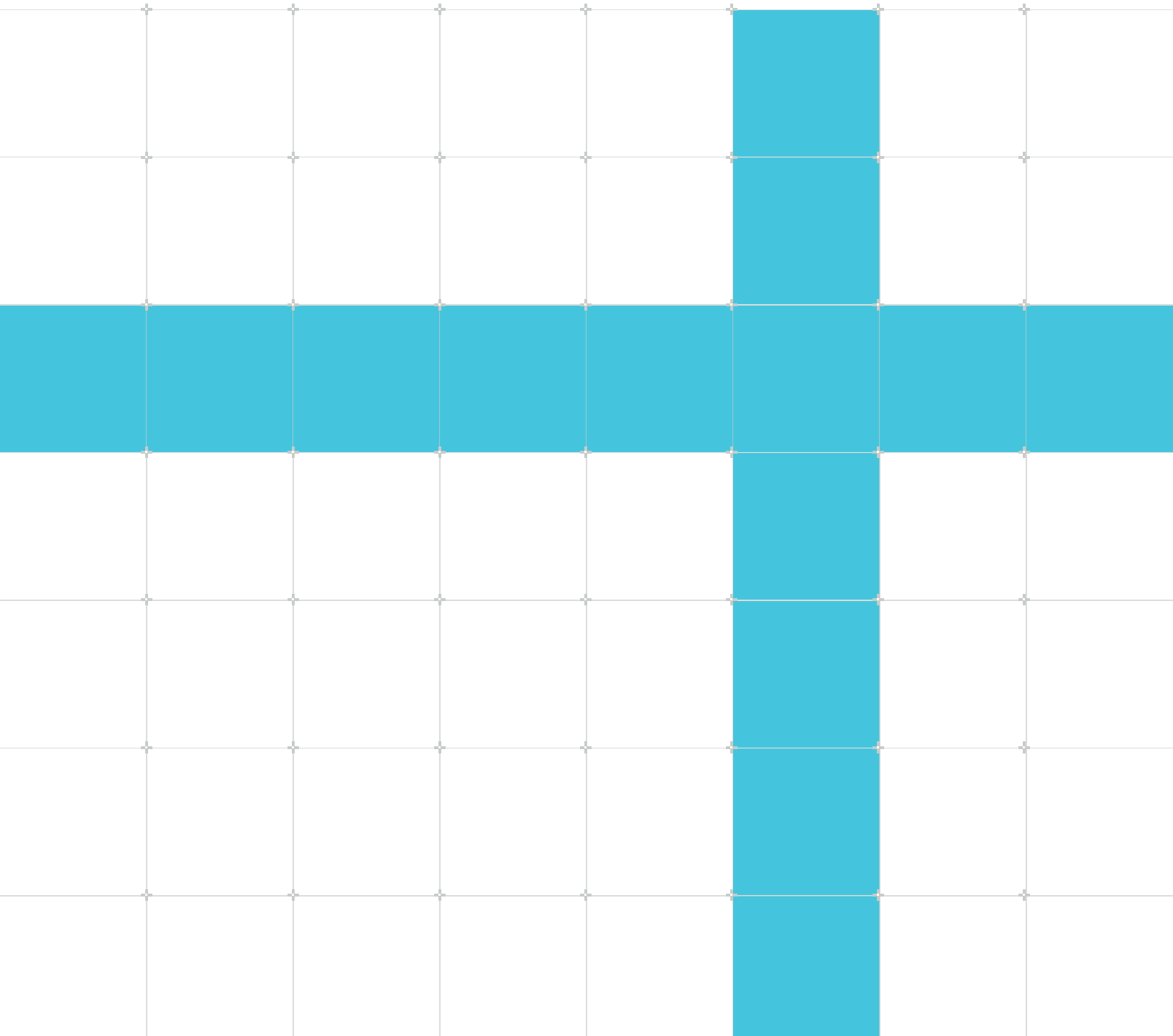
Arm® Architecture Reference Manual for A-profile architecture

Known issues in Issue J.a

Issue 03
102105_J.a_03_en



Arm® Architecture Reference Manual for A-profile architecture

Known issues in Issue J.a

Release information

Document history

Issue	Date	Confidentiality	Change
F.c-04	18 December 2020	Non-Confidential	Known Issues in Arm® Architecture Reference Manual, Issue F.c, as of 18 December 2020
G.b-05	31 January 2022	Non-Confidential	Known Issues in Arm® Architecture Reference Manual, Issue G.b, as of 7 January 2022
H.a-06	22 July 2022	Non-Confidential	Known Issues in Arm® Architecture Reference Manual, Issue H.a, as of 22 July 2022
I.a-06	21 April 2023	Non-Confidential	Known Issues in Arm® Architecture Reference Manual, Issue I.a, as of 31 March 2023
J.a-00	5 May 2023	Non-Confidential	Known Issues in Arm® Architecture Reference Manual, Issue J.a, as of 31 March 2023
J.a-01	16 June 2023	Non-Confidential	Known Issues in Arm® Architecture Reference Manual, Issue J.a, as of 19 May 2023
J.a-02	30 June 2023	Non-Confidential	Known Issues in Arm® Architecture Reference Manual, Issue J.a, as of 23 June 2023
J.a-03	31 August 2023	Non-Confidential	Known Issues in Arm® Architecture Reference Manual, Issue J.a, as of 18 August 2023

Proprietary Notice

Confidentiality Status

This document is Non-Confidential. The right to use, copy and disclose this document may be subject to license restrictions in accordance with the terms of the agreement entered into by Arm and the party that Arm delivered this document to.

Unrestricted Access is an Arm internal classification.

Product Status

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

Feedback

Arm welcomes feedback on this product and its documentation. To provide feedback on the product, create a ticket on https://support.developer.arm.com

To provide feedback on the document, fill the following survey: https://developer.arm.com/documentation-feedback-survey.

Inclusive language commitment

Arm values inclusive communities. Arm recognizes that we and our industry have used language that can be offensive. Arm strives to lead the industry and create change.

We believe that this document contains no offensive language. To report offensive language in this document, email terms@arm.com.

Contents

1. Introduction

1.1 Conventions

The following subsections describe conventions used in Arm documents.

Glossary

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

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

Typographic conventions

Arm documentation uses typographical conventions to convey specific meaning.

Convention	Use
italic	Citations.
bold	Interface elements, such as menu names. Terms in descriptive lists, where appropriate.
`monospace`	Text that you can enter at the keyboard, such as commands, file and program names, and source code.
`monospace underline`	A permitted abbreviation for a command or option. You can enter the underlined text instead of the full command or option name.
`<and>`	Encloses replaceable terms for assembler syntax where they appear in code or code fragments. For example: `MRC p15, 0, <Rd>, <CRn>, <CRm>, <Opcode_2>`
SMALL CAPITALS	Terms that have specific technical meanings as defined in the *Arm® Glossary*. For example, **IMPLEMENTATION DEFINED**, **IMPLEMENTATION SPECIFIC**, **UNKNOWN**, and **UNPREDICTABLE**.
Caution	Recommendations. Not following these recommendations might lead to system failure or damage.
Warning	Requirements for the system. Not following these requirements might result in system failure or damage.
Danger	Requirements for the system. Not following these requirements will result in system failure or damage.
Note	An important piece of information that needs your attention.

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

1.2 Useful resources

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

Access to Arm documents depends on their confidentiality:

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

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


Note

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

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

1.3 Other information

See the Arm website for other relevant information.

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

2. Known issues

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

Key

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

2.1 D16198

In section D19.2.124 (SCTLR_EL1, System Control Register (EL1)) field 'A, bit [1]', the value `0b0` description that reads:

> Alignment fault checking disabled when executing at EL1 or EL0.
>
> Instructions that load or store one or more registers, other than load/store exclusive and load-acquire/store-release, do not check that the address being accessed is aligned to the size of the data element(s) being accessed.

is corrected to read:

> Alignment fault checking disabled when executing at EL1 or EL0. Alignment checks on some instructions are not disabled by this control. For more information, see Alignment of data accesses.

The following text in the field description is deleted:

> Load/store exclusive and load-acquire/store-release instructions have an alignment check regardless of the value of the A bit.
>
> If FEAT_MOPS is implemented, SETG* instructions have an alignment check regardless of the value of the A bit.

Equivalent changes are made in the following sections:

- D19.2.125 (SCTLR_EL2, System Control Register (EL2)).
- D19.2.126 (SCTLR_EL3, System Control Register (EL3)).

2.2 D16648

In section D19.2.48 (HCR_EL2, Hypervisor Configuration Register), in the `0b1` value description of 'FWB, bit [46]', the text that reads:

> When this bit is 1, then:
>
> - Bit[5] of stage 2 page or block descriptor is **RES0**.
> - When bit[4] of stage 2 page or block descriptor is 1 and when:
>   - Bits[3:2] of stage 2 page or block descriptor are `0b11`, the resultant memory type and inner or outer cacheability attribute is the same as the stage 1 memory type and inner or outer cacheability attribute.
>   - Bits[3:2] of stage 2 page or block descriptor are `0b10`, the resultant memory type and attribute is Normal Write-Back.
>   - Bits[3:2] of stage 2 page or block descriptor are 0b0x, the resultant memory type will be Normal Non-cacheable except where the stage 1 memory type was Device-<attr> the resultant memory type will be Device-<attr>.

is corrected to read:

> When this bit is 1, then:
>
> - If the stage 1 translation specifies a cacheable memory type, then the stage 1 cache allocation hint is applied to the final cache allocation hint where the final memory type is cacheable.
> - If the stage 1 translation does not specify a cacheable memory type, then if the final memory type is cacheable, it is treated as Read-Allocate, Write-Allocate.
>
> The encoding of the stage 2 memory type and cacheability attributes in bits[5:2] of the stage 2 page or block descriptors are as described in 'Stage 2 memory type and Cacheability attributes when FEAT_S2FWB is enabled'.

2.3 D16729

In section D19.2.43 (FPEXC32_EL2, Floating-Point Exception Control register), in the field 'EN, bit [30]', the text that reads:

> - For Advanced SIMD instructions only:
>   - CPACR.ASEDIS.
>   - If executing in Non-secure state, HCPTR.TASE and NSACR.NSTRCDIS.

is replaced with:

> - For Advanced SIMD instructions only:
>   - CPACR.ASEDIS.

- If executing in Non-secure state, HCPTR.TASE and NSACR.NSASEDIS.

Similar changes are made in the following sections:

- G8.2.32 (CPACR, Architectural Feature Access Control register), in the field 'cp10, bits [21:20]'.
- G8.2.53 (FPEXC, Floating-Point Exception Control register), in the field 'EN, bit [30]'.

2.4 D17119

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

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

2.5 C17311

In section D8.11.3 (Additional behavior when HCR_EL2.NV is 1 and HCR_EL2.NV1 is 1), the text in I_{JKLJK} that reads:

> For Block descriptors and Page descriptors in the EL1&0 translation regime, all of the following apply:
> - Block descriptor and Page descriptor bit[54] holds PXN, not UXN.
> - Block descriptor and Page descriptor bit[53] is **RES0**.
> - Block descriptor and Page descriptor bit[6], AP[1], is treated as 0 regardless of the actual value.

is updated to read:

> For Block descriptors and Page descriptors in the EL1&0 translation regime, all of the following apply:
> - Block descriptor and Page descriptor bit[54] holds PXN, not UXN.
> - The Effective value of UXN is 0.
> - Block descriptor and Page descriptor bit[53] is **RES0**.
> - Block descriptor and Page descriptor bit[6], AP[1], is treated as 0 regardless of the actual value.

An equivalent change is made in section D19.2.48 (HCR_EL2, Hypervisor Configuration Register), in the definition of 'NV1, bit [43]'.

2.6 R17462

In section I2.2.2 (Halt-on-debug), the statement that reads:

> Arm recommends that a system counter implements a Halt-on-debug signal that can be controlled by a debugger using the Embedded Cross-Trigger (ECT) using a system-level cross-trigger interface that includes:
>
> - A debug request output trigger event that asserts the Halt-on-debug signal.
> - A restart request output trigger event that deasserts the Halt-on-debug signal.
>
> For more information, see About the Embedded Cross-Trigger on page H5-11166.

is updated to read:

> Where the system counter implements a Halt-on-debug signal and the system supports halting the system counter, Arm recommends that the Halt-on-debug signal can be controlled by a debugger using the Embedded Cross-Trigger (ECT) using a system-level cross-trigger interface that includes:
>
> - A debug request output trigger event that asserts the Halt-on-debug signal.
> - A restart request output trigger event that deasserts the Halt-on-debug signal.
>
> For more information, see About the Embedded Cross-Trigger on page H5-11166.

2.7 D17465

In section D19.1.2 (General behavior of accesses to the AArch64 System registers), in the subsection 'Synchronization requirements for AArch64 System registers', the statements that read:

> Conceptually, explicit synchronization occurs as the first step of each of these events, so that if the event uses state that has previously been changed but was not synchronized by the time of the event, the event is guaranteed to use the state as if it had been synchronized.
>
> Note: This explicit synchronization applies as the first step of the execution of the events, and does not apply to any effect of System registers that apply to the fetch and decode of the instructions that cause these events, such as breakpoints or changes to the translation table.

are updated to read:

> The pseudocode for each of these events defines the point at which the explicit synchronization takes effect, as a result of a call to the SynchronizeContext() function or InstructionSynchronizationBarrier() function.
>
> The principle behind the position of the call to SynchronizeContext() or InstructionSynchronizationBarrier() is that it occurs before indirect reads of System registers that

> are used in the construction of the target context, but is permitted to occur after indirect reads of System registers that apply to the context in which the instruction is executed or from which the event is taken.
>
> Note: For some instructions, this means that some System registers are indirectly read before the explicit synchronization, and therefore changes to those System registers might need an explicit Context synchronization event before the instruction. For example, changes to some fields of HCR_EL2 at EL2 need an explicit ISB in program order before an ERET instruction.

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

```
    ...
        FailTransaction(cause, FALSE);

    SynchronizeContext();

    // If coming from AArch32 state, the top parts of the X[] registers might be set
to zero
    ...
```

is corrected to read:

```
    ...
        FailTransaction(cause, FALSE);

    if !IsFeatureImplemented(FEAT_ExS) || SCTLR[target_el].EIS == '1' then
        SynchronizeContext();

    // If coming from AArch32 state, the top parts of the X[] registers might be set
to zero
    ...
```

In section J1.1.4 (aarch64/instrs), in the pseudocode function Arch64.ExceptionReturn(), the code that reads:

```
    ...
            TakeUnmaskedPhysicalSErrorInterrupts(iesb_req);
    SynchronizeContext();

    // Attempts to change to an illegal state will invoke the Illegal Execution
state mechanism
    ...
```

is corrected to read:

```
    ...
            TakeUnmaskedPhysicalSErrorInterrupts(iesb_req);

    if !IsFeatureImplemented(FEAT_ExS) || SCTLR[].EOS == '1' then
        SynchronizeContext();

    // Attempts to change to an illegal state will invoke the Illegal Execution
state mechanism
    ...
```

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

```
DRPSInstruction()

    SynchronizeContext();

    sync_errors = HaveIESB() && SCTLR[].IESB == '1';
    if HaveDoubleFaultExt() && !UsingAArch32() then
        sync_errors = sync_errors || (SCR_EL3.EA == '1' && SCR_EL3.NMEA == '1' &&
 PSTATE.EL == EL3);
    // SCTLR[].IESB might be ignored in Debug state.
    if !ConstrainUnpredictableBool(Unpredictable_IESBinDebug) then
        sync_errors = FALSE;
    if sync_errors then
        SynchronizeErrors();

    DebugRestorePSR();

    return;
```

is corrected to read:

```
DRPSInstruction()

    sync_errors = HaveIESB() && SCTLR[].IESB == '1';
    if HaveDoubleFaultExt() && !UsingAArch32() then
        sync_errors = sync_errors || (SCR_EL3.EA == '1' && SCR_EL3.NMEA == '1' &&
 PSTATE.EL == EL3);
    // SCTLR[].IESB might be ignored in Debug state.
    if !ConstrainUnpredictableBool(Unpredictable_IESBinDebug) then
        sync_errors = FALSE;
    if sync_errors then
        SynchronizeErrors();

    SynchronizeContext();

    DebugRestorePSR();

    return;
```

2.8 D17494

In section D19.12.2, (CNTHCTL_EL2, Counter-timer Hypervisor Control Register), the EL1TVCT, bit[14] field description, the text that reads:

> Traps EL0 and EL1 accesses to the EL1 virtual counter registers to EL2, when EL2 is enabled for the current Security state.

EL1TVCT	Meaning
0b0	Watchpoint exception. See Watchpoint exceptions on page D2-5446.

EL1TVCT	Meaning
0b1	If HCR_EL2.E2H is 0 or HCR_EL2.TGE is 0, then: • In AArch64 state, traps EL0 and EL1 accesses to CNTVCT_EL0 to EL2, unless they are trapped by CNTKCTL_EL1.EL0VCTEN. • In AArch32 state, traps EL0 and EL1 accesses to CNTVCT to EL2, unless they are trapped by CNTKCTL_EL1.EL0VCTEN or CNTKCTL.PL0VCTEN. If HCR_EL2.{E2H, TGE} is {1, 1}, this control does not cause any instructions to be trapped.

> If EL3 is implemented and EL2 is not implemented, behavior is as if this bit is 0 other than for the purpose of a direct read.
>
> This control applies regardless of the value of the CNTHCTL_EL2.ECV bit.

is clarified to read:

> Traps EL0 and EL1 accesses to the EL1 virtual counter registers to EL2 when EL2 is enabled in the current Security state, as follows:
>
> • In AArch64 state, accesses to CNTVCT_EL0 and CNTVCTSS_EL0 are trapped to EL2 and reported using EC syndrome value 0x18, unless they are trapped by CNTKCTL_EL1.EL0VCTEN.
>
> • In AArch32 state, accesses to CNTVCT are trapped to EL2 and reported with EC syndrome value 0x04, unless they are trapped by CNTKCTL_EL1.EL0VCTEN or CNTKCTL.PL0VCTEN.

EL1TVCT	Meaning
0b0	This control does not cause any instructions to be trapped.
0b1	EL0 and EL1 accesses to the specified registers are trapped to EL2.

> If EL3 is implemented and EL2 is not implemented, behavior is as if this bit is 0 other than for the purpose of a direct read.
>
> If HCR_EL2.{E2H, TGE} is {1, 1}, this control does not cause any instructions to be trapped.
>
> This control applies regardless of the value of the CNTHCTL_EL2.ECV bit.

Equivalent changes are made in the following locations, where the text for relevant trap fields has been changed to include CNTVCTSS_EL0 and CNTPCTSS_EL0: EL1PCTEN, EL0VCTEN, and EL0PCTEN fields in section D19.12.2, (CNTHCTL_EL2, Counter-timer Hypervisor Control Register) as well as EL0VCTEN and EL0PCTEN fields in section D19.12.15, (CNTKCTL_EL1, Counter-timer Kernel Control Register).

2.9 D18202

In section D10.2.7 (Operation Type packet), subsection 'Operation Type packet payload (load/store)', the PRED field definition is updated to include the following text:

> Predicated SVE operation. The operation is one of the following:

- If FEAT_SPEv1p2 is implemented, a predicated load operation that writes to one or more vector destination registers under a Governing predicate using zeroing predication.
- A predicated store of one or more vector registers. Note: If FEAT_SPEv1p2 is not implemented, it is **IMPLEMENTATION DEFINED** whether this field is `0b0` or `0b1` for a predicated load operation that writes to one or more vector destination registers under a Governing predicate using zeroing predication.

2.10 C18270

In section D1.3.5 (Synchronous Exception types), in the table of information statement IZFGJP, the text that reads:

Priority	Synchronous exception type
31	In the following priority order: 1. Data Abort exceptions on translation table walks and translation table entry updates. 2. GPC Exceptions on translation table walks and translation table entry updates, if FEAT_RME is implemented. 3. Data Abort exceptions due to synchronous External aborts on translation table walks and translation table entry updates. 4. Data Abort exceptions on the final physical address access of the address translation process. 5. If prioritized here, then in the following priority order: a. Data Abort exceptions due to a Granule Protection Fault (GPF) on the final physical address access of the address translation process, if FEAT_RME is implemented. b. GPC exceptions on the final physical address access of the address translation process, if FEAT_RME is implemented. c. Data Abort exceptions due to synchronous External aborts on the final physical address access of the address translation process. Whether these are prioritized here or as Priority 33 is **IMPLEMENTATION DEFINED**. See also: • MMU fault prioritization from a single address translation stage on page D8-5926. • External aborts on page D7-5799.
32	Watchpoint exception. See Watchpoint exceptions on page D2-5446.

Priority	Synchronous exception type
33	• If prioritized here, then in the following priority order: 1. Data Abort exceptions due to a Granule Protection Fault (GPF) on the final physical address access of the address translation process, if FEAT_RME is implemented. 2. GPC exceptions on the final physical address access of the address translation process, if FEAT_RME is implemented. 3. Data Abort exceptions due to synchronous External aborts on the final physical address access of the address translation process. Whether these are prioritized here or as Priority 31 is **IMPLEMENTATION DEFINED**. • Data Abort exceptions on Tag Check Faults, if FEAT_MTE2 is implemented. See also: • PE handling of Tag Check Fault on page D10-5979. • External aborts on page D7-5799.

is changed to read:

Priority	Synchronous exception type
31	In the following priority order: 1. Data Abort exceptions on translation table walks and translation table entry updates. 2. GPC Exceptions on translation table walks and translation table entry updates, if FEAT_RME is implemented. 3. Data Abort exceptions due to synchronous External aborts on translation table walks and translation table entry updates. 4. Data Abort exceptions arising from an MMU fault not on a translation table walk, that are not covered by priorities 32 or 34. See also: • MMU fault prioritization from a single address translation stage on page D8-5926.
32	If prioritized here, then in the following priority order: 1. Data Abort exceptions due to a Granule Protection Fault (GPF) on the final physical address access of the address translation process, if FEAT_RME is implemented. 2. GPC exceptions on the final physical address access of the address translation process, if FEAT_RME is implemented. 3. Any of a. Data Abort exceptions due to synchronous External aborts on the final physical address access of the address translation process. b. Data Abort exceptions on Tag Check Faults, if FEAT_MTE2 is implemented. See also: • PE handling of Tag Check Fault on page D10-5979. • External aborts on page D7-5799.
33	Watchpoint exception. See Watchpoint exceptions on page D2-5446.
34	The exceptions listed for priority 32 if they are prioritized as 34.

This table is also upgraded from an information statement to a rule.

Similarly, the following sections are updated to match the new priority definitions:

- G1.12.2 (Exception prioritization for exceptions taken to AArch32 state).
- H2.2.5 (Debug state entry and debug event prioritization).

In section D10.7 (PE handling of Tag check faults), the text that reads:

> A Data Abort due to a Tag Check Fault is prioritized as a Data Abort exception generated by a synchronous External abort that was not generated by a translation table walk.

> If an access generates both a Data Abort due to a Synchronous Tag Check Fault, and a Data Abort due to a synchronous External abort that was not generated by a translation table walk, it is **IMPLEMENTATION DEFINED** which abort is reported. For more information on prioritization of exceptions see Synchronous exception types on page D1-5363.

is replaced with:

> For the priority of a synchronous exception due to a Tag Check Fault, see section D1.3.5 Synchronous exception types.

2.11 D18330

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

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

2.12 D18465

In section D19.2.125 (SCTLR_EL2, System Control Register (EL2)), for all of the bits that are described as having a function when HCR_EL2.E2H==1 && HCR_EL2.TGE==1 and being **RES0** otherwise, it is clarified that these bits:

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

2.13 C18635

In Section D19.2.37, (ESR_EL1, Exception Syndrome Register (EL1)), subsection 'ISS encoding for an exception from a Pointer Authentication instruction authentication failure', the text that reads:

> It is **IMPLEMENTATION DEFINED** whether the following instructions generate an exception directly from the authorization failure, rather than changing the address in a way that will generate a Translation fault when the address is accessed:

is changed to read:

> If FEAT_FPACCOMBINE is implemented, the following instructions generate a PAC Fail exception when the Pointer Authentication Code (PAC) is incorrect:

The equivalent changes are made in the following sections:

- D19.2.38 (ESR_EL2, Exception Syndrome Register (EL2)), subsection 'ISS encoding for an exception from a Pointer Authentication instruction authentication failure'
- D19.2.39 (ESR_EL3, Exception Syndrome Register (EL3)), subsection 'ISS encoding for an exception from a Pointer Authentication instruction authentication failure'

2.14 D18710

In section G8.3.15 (DBGDSCRint, Debug Status and Control Register, Internal View), the accessibility pseudocode that reads:

```
if Halted() && ConstrainUnpredictableBool(Unpredictable_IGNORETRAPINDEBUG) then
    if t == 15 then
        PSTATE.<N,Z,C,V> = DBGDSCRint<31:28>;
    else
        R[t] = DBGDSCRint;
elsif PSTATE.EL == EL0 then
```

is corrected to read:

```
if Halted() && ConstrainUnpredictableBool(Unpredictable_IGNORETRAPINDEBUG) then
    if t == 15 then
        ConstrainUnpredictableProcedure(Unpredictable_MRC_APSR_TARGET);
    else
        R[t] = DBGDSCRint;
elsif PSTATE.EL == EL0 then
```

In four other places within the accessibility pseudocode for this register, the code that reads:

```
    if t == 15 then
        PSTATE.<N,Z,C,V> = DBGDSCRint<31:28>;
    else
        R[t] = DBGDSCRint;
```

is corrected to read:

```
    if t == 15 then
        if Halted() then
            ConstrainUnpredictableProcedure(Unpredictable_MRC_APSR_TARGET);
        else
            PSTATE.<N,Z,C,V> = DBGDSCRint<31:28>;
    else
        R[t] = DBGDSCRint;
```

2.15 C18842

In section I6.5.14 (AMDEVARCH, Activity Monitors Device Architecture Register), the text in the ARCHID, bits [15:0] description that reads:

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

is changed to read:

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

2.16 D18887

In section G8.2.120 (PAR, Physical Address Register), the MCR accessibility pseudocode that reads:

```
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 HaveEL(EL3) && ELUsingAArch32(EL3) then
        PAR_NS = ZeroExtend(R[t]);
    else
        PAR = ZeroExtend(R[t]);
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && ELUsingAArch32(EL3) then
        PAR_NS = ZeroExtend(R[t]);
    else
```

```
        PAR = ZeroExtend(R[t]);
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        PAR_S = ZeroExtend(R[t]);
    else
        PAR_NS = ZeroExtend(R[t]);
```

is updated to read:

```
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 HaveEL(EL3) && ELUsingAArch32(EL3) then
        PAR_NS<31:0> = R[t];
    else
        PAR<31:0> = R[t];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && ELUsingAArch32(EL3) then
        PAR_NS<31:0> = R[t];
    else
        PAR<31:0> = R[t];
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        PAR_S<31:0> = R[t];
    else
        PAR_NS<31:0> = R[t];
```

An equivalent change is made in the MCR accessibility pseudocode of sections G8.4.2 (PMCCNTR, Performance Monitors Cycle Count Register), G8.2.166 (TTBR0, Translation Table Base Register 0), and G8.2.167 (TTBR1, Translation Table Base Register 1).

2.17 C18973

In section D19.1.3 (Principles of the ID scheme for fields in ID registers) the paragraph which reads:

> The ID fields, which are either signed or unsigned, use increasing numerical values to indicate increases in functionality. Therefore, if a value of `0x1` indicates the presence of some instructions, then the value `0x2` will indicate the presence of those instructions plus some additional instructions or functionality.

is clarified to read:

> The ID fields, which are either signed or unsigned, use increasing numerical values to indicate increases in functionality. Therefore, for an unsigned ID field, if the value `0x1` indicates the presence of some instructions, then the value `0x2` will indicate the presence of those instructions plus some additional instructions or functionality.

In the same section, the text which reads:

> In a few cases, the architecture has been changed to permit implementations to exclude a feature that has previously been required and for which no ID field has been defined. In these cases, a new ID field is defined and:
>
> - The field holds a signed value.
> - The field value `0xF` indicates that the feature is not implemented.
> - The field value `0x0` indicates that the feature is implemented.
> - Software that depends on the feature can use the test:

```
if value >= 0 {
// Software features that depend on the presence of the hardware feature
}
```

is clarified to read:

> In a few cases, the architecture has been changed to permit implementations to exclude features that have previously been required and for which no ID field has been defined. In these cases, a new ID field is defined and:
>
> - The field holds a signed value.
> - The field value `0x0` indicates that the feature is implemented.
> - The field value `0xF` indicates that the feature is not implemented.
> - Increasingly negative values indicate additional aspects of the architecture that are not implemented as a result of the feature not being implemented.
> - Software that depends on the feature can use the test:

```
if value >= 0 {
// Software features that depend on the presence of the hardware feature
}
```

2.18 D19270

In section A2.18 (The Memory Partitioning and Monitoring (MPAM) Extension), the text that reads:

> The fields that identify the presence of the MPAM Extension are:
>
> - ID_AA64PFR0_EL1.MPAM.
> - EDPFR.MPAM.

is corrected to read:

> The following fields identify the presence of the MPAM Extension:
>
> - ID_AA64PFR0_EL1.MPAM.
> - ID_AA64PFR1_EL1.MPAM_frac.

2.19 C19346

In section A2.5.3 (Features added to the Armv8.3 extension in later releases), the description of 'FEAT_CONSTPACFIELD, PAC algorithm enhancement' that reads:

> FEAT_CONSTPACFIELD introduces functionality that permits an implementation with pointer authentication to use the value of bit[55] in the virtual address to determine the size of the PAC field, even when the top byte is not being ignored.

is updated to read:

> FEAT_CONSTPACFIELD introduces functionality that permits an implementation with pointer authentication to use the value of bit[55] in the virtual address to determine the size of the PAC field when adding a PAC to the virtual address, even when the top byte is not being ignored.

In section D8.8 (Pointer authentication), rule R_{NQZWG} that reads:

> If FEAT_CONSTPACFIELD is implemented, then an implementation is permitted to use the value in Xn[55] to determine the size of the PAC field, even when address tagging is not used.

is updated to read:

> If FEAT_CONSTPACFIELD is implemented, then an implementation is permitted to use the value in Xn[55] to determine the size of the PAC field when adding a PAC to Xn, even when address tagging is not used.

2.20 D19583

In section D1.3.8 (Configurable instruction controls), rule R_{JTXTF} that reads:

> It is **UNPREDICTABLE** / **CONSTRAINED UNPREDICTABLE** whether configurable instruction controls generate an exception when the instruction is **UNPREDICTABLE** or **CONSTRAINED UNPREDICTABLE** in the PE state in which the instruction is executed.

is updated to read:

> It is **CONSTRAINED UNPREDICTABLE** whether configurable instruction controls generate an exception when the instruction is **UNPREDICTABLE** or **CONSTRAINED UNPREDICTABLE** in the PE state in which the instruction is executed, with all of the following constraints:
>
> - If the instruction description explicitly states that the configurable instruction control is applied with higher priority than the **CONSTRAINED UNPREDICTABLE** behavior, then the configurable instruction control generates an exception.
> - The **CONSTRAINED UNPREDICTABLE** behaviors cannot lead to any behavior that is prohibited by the general definition of **UNPREDICTABLE**.

2.21 D19696

In section B1.2.5 (Process state, PSTATE), in the subsection 'Accessing PSTATE fields at EL0', the table B1-1 'Accessing PSTATE fields at EL0 using MRS and MSR (register)' that reads:

Special-purpose Register	PSTATE fields
NZCV	N, Z, C, V
DAIF	D, A, I, F

is corrected to read:

Special-purpose Register	PSTATE fields
NZCV	N, Z, C, V
DAIF	D, A, I, F
SSBS	SSBS
DIT	DIT
TCO	TCO

Within the same section, the text that reads:

> Software can also use the MSR (immediate) instruction to directly write to PSTATE.{D, A, I, F}. Table B1-2 shows the MSR (immediate) operands that can directly write to PSTATE.{D, A, I, F} when the PE is at EL0 using AArch64 state.
>
> Table B1-2 'Accessing PSTATE.{D, A, I, F} at EL0 using MSR (immediate)'

Operand	PSTATE fields	Notes
DAIFSet	D, A, I, F	Directly sets any of the PSTATE.{D,A, I, F} bits to 1
DAIFClr	D, A, I, F	Directly clears any of the PSTATE.{D, A, I, F} bits to 0

is corrected to read:

> Software can also use the MSR (immediate) instruction to directly write to PSTATE.{D, A, I, F, SSBS, DIT, TCO}. Table B1-2 shows the MSR (immediate) operands that can directly write to PSTATE.{D, A, I, F, SSBS, DIT, TCO} when the PE is at EL0 using AArch64 state.
>
> Table B1-2 'Accessing PSTATE.{D, A, I, F, SSBS, DIT, TCO} at EL0 using MSR (immediate)'

Operand	PSTATE fields	Notes
DAIFSet	D, A, I, F	Directly sets any of the PSTATE.{D,A, I, F} bits to 1
DAIFClr	D, A, I, F	Directly clears any of the PSTATE.{D, A, I, F} bits to 0
SBSS	SBSS	Directly sets the PSTATE.SSBS bit to CRm<0>
DIT	DIT	Directly sets the PSTATE.DIT bit to CRm<0>
TCO	TCO	Directly sets the PSTATE.TCO bit to CRm<0>

2.22 D19804

In section D9.4.1 (Virtual address translation), the following text is added:

> If a tag write by an STG instruction that does not also write data is translated by a writeable-clean descriptor, but the tag write effect is IGNORED due to a stage 1 descriptor not having the Tagged memory attribute, or because Allocation tag access is disabled for the instruction by SCR_EL3.ATA, HCR_EL2.ATA, SCTLR_ELx.ATA or SCTLR_ELx.ATA0, it is **CONSTRAINED UNPREDICTABLE** whether hardware updates the dirty state of that descriptor.

2.23 C20016

In section C6.2.244 (PACGA), the instruction description that reads:

> Pointer Authentication Code, using Generic key. This instruction computes the pointer authentication code for an address in the first source register, using a modifier in the second source register, and the Generic key. The computed pointer authentication code is returned in the upper 32 bits of the destination register.

is clarified to read:

> Pointer Authentication Code, using Generic key. This instruction computes the pointer authentication code for a 64-bit value in the first source register, using a modifier in the second source register, and the Generic key. The computed pointer authentication code is written to the most significant 32 bits of the destination register, and the least significant 32 bits of the destination register are set to zero.

2.24 R20031

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

> All of these memory types have the following properties:
>
> - Speculative data accesses are not permitted to any memory location with any Device memory attribute. This means that each memory access to any Device memory type must be one that would be generated by a simple sequential execution of the program. The following exceptions to this apply:

the following sub-bullet is added:

> - An LDRAA or LDRAB instruction that fails the pointer authentication check and loads from a location in Device memory is permitted to cause one read access to that location if all of the other requirements for accessing that Device location are met.

2.25 C20037

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

```
PARTIDspaceType AltPARTIDspace(bits(2) el, SecurityState security,
                               PARTIDspaceType primaryPIdSpace)
    case security of
        when SS_NonSecure
            assert el != EL3;
            return primaryPIdSpace; // there is no ALTSP for Non_secure
    ...
```

is updated to read:

```
PARTIDspaceType AltPARTIDspace(bits(2) el, SecurityState security,
                               PARTIDspaceType primaryPIdSpace)
    case security of
        when SS_NonSecure
            assert el != EL3;
            return primaryPIdSpace;
    ...
```

2.26 C20048

In section D8.8.1 (PAC field), the rule that reads:

> R_{FWYCF}
>
> When a 64-bit register, Xn, holds an address with a PAC field, the location of the PAC field is determined by all of the following:
>
> - The bottom_PAC_bit is 64-TCR_ELx.TnSZ.
> - If FEAT_CONSTPACFIELD is implemented, or address tagging is used, then the value of Xn[55] is the value of n used in TnSZ.
> - If FEAT_CONSTPACFIELD is not implemented and address tagging is not used, then the value of Xn[55] is the value of n used in TnSZ.
> - If address tagging is used, the PAC field is Xn[54:bottom_PAC_bit].
> - If address tagging is not used, the PAC field is Xn[63:56, 54:bottom_PAC_bit].

is updated to read:

> R_{FWYCF}
>
> If FEAT_CONSTPACFIELD is not implemented, when a 64-bit register, Xd, holds an address with a PAC field, the location of the PAC field is determined by all of the following:
>
> - The bottom_PAC_bit is 64-TCR_ELx.TnSZ, and the value of n is determined as follows:
>   - If inserting a PAC into an address and address tagging is not used, then the value of Xd[63] is the value of n.

    - Otherwise, the value of Xd[55] is the value of n.
- If address tagging is used, the PAC field is Xd[54:bottom_PAC_bit].
- If address tagging is not used, the PAC field is Xd[63:56, 54:bottom_PAC_bit].

R_{X0001}

If FEAT_CONSTPACFIELD is implemented, when a 64-bit register, Xd, holds an address with a PAC field, the location of the PAC field is determined by all of the following:

- The bottom_PAC_bit is 64-TCR_ELx.TnSZ.
- The value of Xd[55] is the value of n used in TnSZ.
- If address tagging is used, the PAC field is Xd[54:bottom_PAC_bit].
- If address tagging is not used, the PAC field is Xd[63:56, 54:bottom_PAC_bit].

The following information statement is also added to the section:

In the case where address tagging is not used, then any use of a VA that has differing values of bit [63] and bit [55] is a non-canonical VA that will generate a fault. For this reason, the presence or absence of FEAT_CONSTPACFIELD does not affect the use of pointer authentication when inserting a PAC into a canonical VA.

2.27 E20075

In section A2.2.1 (Additional functionality added to Armv8.0 in later releases), under the description of 'FEAT_ETS2, Enhanced Translation Synchronization', the text that reads:

This feature is OPTIONAL in Armv8.0 implementations and mandatory in Armv8.7 implementations.

is corrected to read:

This feature is OPTIONAL in Armv8.0 implementations and mandatory in Armv8.8 implementations.

In section A2.9.2 (Additional requirements of Armv8.7), the reference to 'FEAT_ETS, Enhanced Translation Synchronization' is deleted. The following text is instead added to section A2.10.2 (Additional requirements of Armv8.8):

FEAT_ETS2, Enhanced Translation Synchronization

All implementations of the Armv8.8 architecture are required to implement FEAT_ETS2.

For more information, see FEAT_ETS2 on page A2-84.

In section D8.2.6 (Translation table walk properties), in the subsection 'Ordering of memory accesses from translation table walks', the rule R_{LTJGW} is deleted, and is replaced with the following rule R_{NNFPF}:

> If FEAT_ETS2 is implemented, E1 is an Explicit Memory Effect, E2 is an Implicit Read of a TTD and all of the following apply, then E1 is Ordered-before E2:
>
> - E1 is program-order-before a Fault Effect E3.
> - E2 is Translation-intrinsically-before E3.

In the following sections:

- D19.2.65 (ID_AA64MMFR1_EL1, AArch64 Memory Model Feature Register 1), field 'ETS, bits [39:36]'.
- D19.2.89 (ID_MMFR5_EL1, AArch32 Memory Model Feature Register 5), field 'ETS, bits [3:0]'.
- G8.2.97 (ID_MMFR5, Memory Model Feature Register 5), field 'ETS, bits [3:0]'.

The field description that reads:

> Indicates support for Enhanced Translation Synchronization. >Defined values are:

`0b0000`	Enhanced Translation Synchronization is not supported.
`0b0001`	Enhanced Translation Synchronization is supported.

> All other values are reserved. FEAT_ETS2 implements the functionality identified by the value `0b0001`. In Armv8.0, the permitted values are `0b0000` and `0b0001`. From Armv8.7, the only permitted value is `0b0001`.

is updated to read:

> Indicates support for Enhanced Translation Synchronization. Defined values are:

`0b0000`	Enhanced Translation Synchronization is not supported.
`0b0001`	Enhanced Translation Synchronization is not supported.
`0b0010`	Enhanced Translation Synchronization is supported.

> All other values are reserved.
>
> FEAT_ETS2 implements the functionality identified by the value `0b0010`.
>
> From Armv8.8, the values `0b0000` and `0b0001` are not permitted.

In section E2.4 (Ordering of translation table walks), the text that reads:

> If FEAT_ETS2 is implemented, and a memory access RW1 is Ordered-before a second memory access RW2, then RW1 is also Ordered-before any translation table walk generated by RW2 that generates any of the following:
>
> - A Translation fault.
> - An Address size fault.

- An Access flag fault.

is updated to read:

If FEAT_ETS2 is implemented, E1 is an Explicit Memory Effect, E2 is an Implicit Read of a TTD and all of the following apply, then E1 is Ordered-before E2:

- E1 is program-order-before a Fault Effect E3.
- E2 is Translation-intrinsically-before E3.

References to FEAT_ETS are replaced with FEAT_ETS2 throughout the document.

2.28 D20134

In section D12.11.1 (Definitions), the following definitions are added:

Architectural

An architectural event is an event which gives the same result for the same program on any implementation of the Arm architecture, subject to the program having the same inputs, including asynchronous events triggered by the system, any **IMPLEMENTATION DEFINED** or **UNPREDICTABLE** variation permitted by the event definition, and the *reasonable degree of inaccuracy* described by *The PMU event number space and common events* on page D12-6027. INST_RETIRED is an example of an architectural event.

An architectural event should not be confused with an event defined by the architecture, which is referred to as a Common event.

Microarchitectural

A microarchitectural event is any event which is not architectural. That is, it will give different results for the same program on two different implementations of the Arm architecture, due to microarchitectural differences in the implementations. L1D_CACHE_REFILL is an example of a microarchitectural event.

A microarchitectural event should not be confused with an event defined by the implementation, which is referred to as an **IMPLEMENTATION DEFINED** event.

Correspondingly, the following event definitions in section D12.11.3 (Common event numbers), subsection 'Common microarchitectural events', are moved to the subsection 'Common architectural events':

- `0x0081`, EXC_UNDEF.
- `0x0082`, EXC_SVC.
- `0x0083`, EXC_PABORT.
- `0x0084`, EXC_DABORT.
- `0x0086`, EXC_IRQ.

- `0x0087`, EXC_FIQ.
- `0x0088`, EXC_SMC.
- `0x008A`, EXC_HVC.
- `0x008B`, EXC_TRAP_PABORT.
- `0x008C`, EXC_TRAP_DABORT.
- `0x008D`, EXC_TRAP_OTHER.
- `0x008E`, EXC_TRAP_IRQ.
- `0x008F`, EXC_TRAP_FIQ.

2.29 D20135

In section I6.9.32 (ERR<n>STATUS, Error Record <n> Primary Status Register, n = 0 - 65534), in the 'SERR, bits [7:0]' field, the value descriptions that read:

> `0x10` Internal data register. For example, parity on a SIMD&FP register. For a PE, all general-purpose, stack pointer, SIMD&FP, and SVE registers are data registers.
>
> `0x11` Internal control register. For example, Parity on a System register. For a PE, all registers other than general-purpose, stack pointer, SIMD&FP, and SVE registers are control registers.

are updated to read:

> `0x10` Internal data register. For example, parity on a SIMD&FP register. For a PE, all general-purpose, stack pointer, SIMD&FP, SVE, and SME registers are data registers.
>
> `0x11` Internal control register. For example, parity on a System register. For a PE, all registers other than general-purpose, stack pointer, SIMD&FP, SVE, and SME registers are control registers.

2.30 D20159

In section D19.5.7 (PMCR_EL0, Performance Monitors Control Register), in the subsection 'Configurations', the text that reads:

> AArch64 System register PMCR_EL0 bits [7:0] are architecturally mapped to External register PMCR_EL0[7:0].

is updated to read:

> AArch64 System register PMCR_EL0 bits [63:32,10:0] are architecturally mapped to External register PMCR_EL0[63:32,10:0].

Equivalent changes are made in sections G8.4.9 (PMCR, Performance Monitors Control Register) and I6.3.19 (PMCR_EL0, Performance Monitors Control Register).

2.31 D20192

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

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

is changed to read:

> When the instructions access a memory type that is not one of the following, a Data Abort for Unsupported Exclusive or Atomic access is generated:
>
> - Normal Inner Non-cacheable, Outer Non-cacheable.
> - Device-GRE.
> - Device-nGRE.
> - Device-nGnRE.
> - Device-nGnRnE.
>
> It is **IMPLEMENTATION DEFINED** which of the following approaches is used to provide this check:
>
> - The check is performed at each enabled stage of translation, and the fault is reported for the first stage of translation that provides an inappropriate memory type. In this case, the value of the HCR_EL2.DC bit does not cause accesses generated by these instructions to generate a stage 1 Data abort.
> - The check is performed agains the resulting memory type after all enabled stages of translation. In this case the fault is reported at the final enabled stage of translation.

2.32 R20208

In section D12.11.3 (Common event numbers), in the subsection 'Common microarchitectural events', the text in the description of '0x0024, STALL_BACKEND, No operation sent for execution due to the backend' that reads:

> The counter counts each cycle counted by CPU_CYCLES where no Attributable instruction or operation was sent for execution and either:
>
> - The backend is unable to accept any of the instruction operations available for the PE.
> - The backend is unable to accept any operations for the PE.
>
> Note: In a single cycle, both the STALL_BACKEND and STALL_FRONTEND events might be counted, if both the backend is unable to accept any operations and there are no operations available to issue from the frontend.

is updated to read:

> The counter counts each cycle counted by CPU_CYCLES where Attributable instructions or operations are available to dispatch for the PE from the frontend, but no Attributable instruction or operation is sent for execution because the backend is unable to accept any of the instructions or operations available for the PE.
>
> It is **IMPLEMENTATION DEFINED** whether the counter also counts each cycle counted by CPU_CYCLES where no Attributable instructions or operations are available to dispatch for the PE from the frontend and the backend is unable to accept any instructions or operations for the PE.
>
> Note: This means that it is **IMPLEMENTATION DEFINED** whether both the STALL_BACKEND and STALL_FRONTEND events can be counted in the same cycle.

Equivalent changes are made to the following event descriptions within the same subsection:

- 0x003D, STALL_SLOT_BACKEND, No operation sent for execution on a Slot due to the backend.
- 0x003E, STALL_SLOT_FRONTEND, No operation sent for execution on a Slot due to the frontend.

Within the same subsection, the Note text in the event description of '0x0023, STALL_FRONTEND, No operation sent for execution due to the frontend' that reads:

> In a single cycle, both the STALL_BACKEND and STALL_FRONTEND events might be counted, if both the backend is unable to accept any operations and there are no operations available to issue from the frontend.

is updated to read:

> This means that it is **IMPLEMENTATION DEFINED** whether both the STALL_BACKEND and STALL_FRONTEND events can be counted in the same cycle. For more information, see STALL_BACKEND.

2.33 C20237

In section H9.2.11 (EDACR, External Debug Auxiliary Control Register), the 'Configuration' text that reads:

> If FEAT_DoPD is implemented, this register is implemented in the Core power domain.
>
> If FEAT_DoPD is not implemented, the power domain that this register is implemented in is **IMPLEMENTATION DEFINED**.

> If the EDACR contains any control bits that must be preserved over power down, then these bits must be accessible by the external debug interface when the OS Lock is locked, OSLSR_EL1.OSLK == 1, and when the Core is powered off.

is updated to read:

> If FEAT_DoPD is implemented:
>
> - This register is implemented in the Core power domain.
> - Any mechanism to preserve control bits in EDACR over power down is optional and **IMPLEMENTATION DEFINED**.
>
> If FEAT_DoPD is not implemented:
>
> - The power domain that this register is implemented in is **IMPLEMENTATION DEFINED**.
> - If the EDACR contains any control bits that must be preserved over power down, then these bits must be accessible by the external debug interface when the OS Lock is locked, OSLSR_EL1.OSLK == 1, and, when the Core is powered off.

2.34 D20270

In section D8.4.1 (Effect of PSTATE on access permission), the text in R_{PMTWB} that reads:

> The PSTATE.PAN bit has no effect on all of the following:
>
> - Instruction fetches.
> - Data cache instructions, except DC ZVA.

is changed to read:

> The PSTATE.PAN bit has no effect on all of the following:
>
> - Instruction fetches.
> - Data cache instructions, except DC ZVA, DC GZVA, and DC GVA.

2.35 C20275

In section D12.11.3 (Common event numbers), in the subsection 'Common architectural events', the event descriptions that read:

> `0x000B`, CID_WRITE_RETIRED, Instruction architecturally executed, Condition code check pass, write to CONTEXTIDR
>
> The counter counts each MSR write to CONTEXTIDR_EL1 and each MCR write to CONTEXTIDR.
>
> If the PE performs two architecturally-executed writes to CONTEXTIDR without an intervening Context synchronization event, it is **CONSTRAINED UNPREDICTABLE** whether the first write is counted.
>
> When FEAT_VHE is implemented, the counter:
>
> - Counts each architecturally-executed instruction accessing the named register CONTEXTIDR_EL1, including when executing at EL2 when HCR_EL2.E2H is `0b1`.
> - Does not count instructions accessing the named register CONTEXTIDR_EL12.
>
> Note: The event is defined by the name used to access the register. The counter does not count writes to the named register CONTEXTIDR_EL2.
>
> `0x001C`, TTBR_WRITE_RETIRED, Instruction architecturally executed, Condition code check pass, write to TTBR
>
> The counter counts MSR writes to TTBR0_EL1 and TTBR1_EL1 in AArch64 state and MCR and MCRR writes to TTBR0 and TTBR1 in AArch32 state. When EL3 is implemented and using AArch32, this includes counting writes to both banked copies of TTBR0 and TTBR1.
>
> If the PE executes two writes to the same TTBR, without an intervening Context synchronization event, it is **CONSTRAINED UNPREDICTABLE** whether the first write to the TTBR, is counted.
>
> If EL3 is implemented and using AArch64, the counter does not count writes to TTBR0_EL3.
>
> If EL2 is implemented and using AArch64, the counter does not count writes to TTBR0_EL2 and VTTBR_EL2.
>
> If EL2 is implemented and using AArch32, the counter does not count writes to HTTBR and VTTBR.
>
> When FEAT_VHE is implemented, the counter:
>
> - Counts each architecturally-executed instruction accessing the named registers TTBR0_EL1 and TTBR1_EL1, including when executing at EL2 when HCR_EL2.E2H is `0b1`.
> - Does not count instructions accessing the named registers TTBR0_EL12 and TTBR1_EL12.

are updated to read:

`0x000B`, CID_WRITE_RETIRED, Instruction architecturally executed, Condition code check pass, write to CONTEXTIDR

The counter counts each MSR write to CONTEXTIDR_EL1 and each MCR write to CONTEXTIDR.

If the PE performs two architecturally executed writes to CONTEXTIDR without an intervening Context synchronization event, it is **CONSTRAINED UNPREDICTABLE** whether the first write is counted.

Note: The counter counts only writes to these named registers. For example:

- When FEAT_VHE or FEAT_Debugv8p2 is implemented, the counter does not count writes to the named register CONTEXTIDR_EL2.
- When FEAT_VHE is implemented, the counter:
  - Counts each architecturally executed instruction accessing the named register CONTEXTIDR_EL1, including when executing at EL2 when HCR_EL2.E2H is `0b1`.
  - Does not count instructions accessing the named register CONTEXTIDR_EL12.
- When FEAT_NV2 is implemented, the counter counts each write to the named register CONTEXTIDR_EL1, including when executing at EL1 when HCR_EL2 {NV2,NV1,NV} is {`0b1`,`0b1`,`0b1`}.

`0x001C`, TTBR_WRITE_RETIRED, Instruction architecturally executed, Condition code check pass, write to TTBR

The counter counts MSR writes to TTBR0_EL1 and TTBR1_EL1 in AArch64 state and MCR and MCRR writes to TTBR0 and TTBR1 in AArch32 state. When EL3 is implemented and using AArch32, this includes counting writes to both banked copies of TTBR0 and TTBR1.

If the PE executes two writes to the same TTBR, without an intervening Context synchronization event, it is **CONSTRAINED UNPREDICTABLE** whether the first write to the TTBR, is counted.

Note: The counter counts only writes to these named registers. For example:

- If EL3 is implemented and using AArch64, the counter does not count writes to TTBR0_EL3.
- If EL2 is implemented and using AArch64, the counter does not count writes to TTBR0_EL2 and VTTBR_EL2.
- If EL2 is implemented and using AArch32, the counter does not count writes to HTTBR and VTTBR.
- When FEAT_VHE is implemented, the counter:
  - Counts each write to the named registers TTBR0_EL1 and TTBR1_EL1, including when executing at EL2 when HCR_EL2.E2H is `0b1`.
  - Does not count instructions accessing the named registers TTBR0_EL12 and TTBR1_EL12.
- When FEAT_NV2 is implemented, the counter counts each write to the named registers TTBR0_EL1 and TTBR1_EL1, including when executing at EL1 when HCR_EL2.{NV2,NV1,NV} is {`0b1`,`0b1`,`0b1`}.

2.36 D20282

In section D12.5.3 (Prohibiting event and cycle counting), the bullet item that reads:

> The cycle counter, PMCCNTR, counts unless any of the following are true:
>
> - Event counting by event counters in the range [0..(HDCR.HPMN-1)] is prohibited or frozen, and PMCR.DP is1.

is updated to read:

> The cycle counter, PMCCNTR, counts unless any of the following are true:
>
> - Event counting by event counters in the range [0..(HDCR.HPMN-1)] is prohibited or frozen by PMCR.FZO, and PMCR.DP is set to 1.

In section D19.5.7 (PMCR_EL0, Performance Monitors Control Register), the description of 'FZO, bit [9]' that reads:

> `0b0` Do not freeze on overflow.
>
> `0b1` Event counter PMEVCNTR<n>_EL0 does not count when PMOVSCLR_EL0[(PMN-1):0] is nonzero and n is in the range of affected event counters.
>
> If PMN is not 0, this field affects the operation of event counters in the range [0 .. (PMN-1)].
>
> This field does not affect the operation of other event counters and PMCCNTR_EL0.

is updated to read:

> `0b0` Do not freeze on overflow.
>
> `0b1` Affected counters do not count when PMOVSCLR_EL0[(PMN-1):0] is nonzero. If PMCR_EL0.DP is `0b1`, then PMCCNTR_EL0 is also disabled. Otherwise, PMCCNTR_EL0 is not affected by this mechanism.
>
> The counters affected by this bit are:
>
> - If PMN is not 0, event counters PMEVCNTR<n> for values of n in the range [0 .. (PMN-1)].
> - If PMCR_EL0.DP is `0b1`, the cycle counter, PMCCNTR_EL0.
>
> Other event counters are not affected by this bit.

In section D19.2.60 (ID_AA64DFR1_EL1, AArch64 Debug Feature Register 1), the field 'DPFZS' is added at bits [55:52], as follows:

> Behavior of the cycle counter when event counting is frozen by a Statistical Profiling management event. Defined values are:
>
> `0b0000` The cycle counter PMCCNTR_EL0 is never affected by PMCR_EL0.FZS.

> `0b0001` The cycle counter PMCCNTR_EL0 does not count when PMCR_EL0.DP is `0b1` and counting by event counters accessible to EL1 is frozen by the PMCR_EL0.FZS mechanism.
>
> If FEAT_PMUv3p7 is not implemented or FEAT_SPEv1p2 is not implemented, the only permitted value is `0b0000`.

In section J1.1.1 (aarch64/debug), the function AArch64.CountPMUEvents() is updated to support PMCR.FZS.

2.37 D20284

In section D19.8.7 (BRBSRC<n>_EL1, Branch Record Buffer Source Address Register <n>, n = 0 - 31), the text that reads:

> When an indirect write occurs with a value with ADDRESS bits [63:P] being other than all zeroes or all ones, an **UNKNOWN** value which is not all zeroes or all ones is written to bits [63:P]. P is defined as the virtual address size supported by the PE, as returned by VAMax(). The value in bits [P-1:0] are the value written.

is updated to read:

> When an indirect write occurs with a value with ADDRESS bits [63:P] being other than all zeroes or all ones, an **UNKNOWN** value which is not all zeroes or all ones is written to bits [63:P]. P is defined as:
>
> - 52 when FEAT_LVA is implemented.
> - 48, otherwise.
>
> The value in bits [P-1:0] is the value written.

Equivalent changes are made in the following sections:

- D19.8.8 (BRBSRCINJ_EL1, Branch Record Buffer Source Address Injection Register).
- D19.8.9 (BRBTGT<n>_EL1, Branch Record Buffer Target Address Register <n>, n = 0 - 31).
- D19.8.10 (BRBTGTINJ_EL1, Branch Record Buffer Target Address Injection Register).

In section D19.4.9 (TRCACVR<n>, Address Comparator Value Register <n>, n = 0 - 15), the text in the 'ADDRESS, bits [63:0]' description that reads:

> The result of writing a value other than all zeros or all ones to ADDRESS at bits[63:P] is an **UNKNOWN** value, where P is defined as the maximum virtual address size supported by the PE.

is updated to read:

> The result of writing a value other than all zeros or all ones to ADDRESS at bits[63:P] is an **UNKNOWN** value, where P is defined as:
>
> - 52 when FEAT_LVA is implemented.

- 48, otherwise.

The same change is made in section H9.3.2 (TRCACVR<n>, Address Comparator Value Register <n>, n = 0 - 15).

In section D4.5.9 (Element Generation), subsection 'Exception element', I_{CMRCN} that reads:

> An invalid address is one where bits [63:P] are not all zeros or all ones, where P is defined as the maximum virtual address size supported by the PE.

is updated to read:

> An invalid address is one where bits [63:P] are not all zeroes or all ones, where P is defined as:
>
> - 52 when FEAT_LVA is implemented.
> - 48, otherwise.

The same change is made to the statement I_{KZXQW} within subsection 'Target Address element' of the same section.

2.38 D20303

In section D1.3.1 (Exception entry terminology), in the subsection 'Definition of a precise exception and imprecise exception', the bullet within rule R_{TNVSL} that reads:

> - For a synchronous exception that is taken from AArch64 state during an instruction that performs more than one single-copy atomic memory access, the values in registers or memory affected by the instructions can be **UNKNOWN**, if all of the following apply:

is updated to read:

> - For a precise exception that is taken from AArch64 state during an instruction that performs more than one single-copy atomic memory access, the values in registers or memory affected by the instructions can be **UNKNOWN**, if all of the following apply:

In section D1.3.6 (Asynchronous exception types), in the subsection 'Taking an interrupt during a multi-access load or store', rule R_{ZBFSL} that reads:

> If in AArch64 state, interrupts can be taken during a sequence of memory accesses caused by a single load or store instruction. This is true regardless of the memory type being accessed.

is updated to read:

> In AArch64 state, interrupts can be taken during a sequence of memory accesses caused by a single load or store instruction. This is true regardless of the memory type being accessed.
>
> In this situation, the behavior is consistent with the requirements described in R_{TNVSL} in Definition of a precise exception and imprecise exception on page D1-5355.

2.39 C20312

In section D8.12.1 (MMU fault types), in the subsection 'TLB conflict abort', the rules that read:

> R FVQCK
>
> If an address matches multiple entries in a TLB and does not generate a TLB conflict abort, then all of the following apply:
>
> - The resulting behavior is **CONSTRAINED UNPREDICTABLE**.
> - The **CONSTRAINED UNPREDICTABLE** behavior cannot permit access to memory regions with permissions or attributes that would not be possible in the current Security state at the current Exception level.
>
> I HLHBH
>
> For more information, see **CONSTRAINED UNPREDICTABLE** behaviors due to caching of control or data values on page K1-12573.

are moved to section D8.14.1 (Using break-before-make when updating translation table entries), and updated to read:

> R FVQCK
>
> If translation table entries are changed without appropriate TLB maintenance operations, including in the case where use of the break-before-make sequence is required but software does not follow the break-before-make sequence, it is possible that TLBs concurrently hold multiple different copies of those translation table entries.
>
> In this situation, the following behaviors are permitted for a speculative or architectural access to the address resolved by those TLB entries:
>
> - Use of the address matches multiple entries in a TLB, and a TLB conflict abort is detected. In this case, no access is made to memory based on those TLB entries. If the access is architectural, then the TLB conflict abort is reported as an exception.
> - The resulting behavior is **CONSTRAINED UNPREDICTABLE**, and gives a behavior consistent with translation using one of the matching entries, or an amalgamation of more than one of the matching entries, but cannot permit access to memory regions with permissions or attributes that would not be possible to be assigned by valid translation table entries in the translation regime and stage of translation being used for access. This includes, for example:
>   - Insufficient TLB maintenance for stage 1 translations by EL1 must not permit it to bypass the configuration of stage 2 translation.
>   - Insufficient TLB maintenance by Non-secure state must not permit it to access any memory in Secure PA space.
>
> I HLHBH

For more information, see **CONSTRAINED UNPREDICTABLE** behaviors due to caching of control or data values on page K1-12573.

2.40 D20317

In section D16.3 (Branch record buffer operation), the text in rule R_{QKQZL} that reads:

If any of the following are true, the physical offset is zero, otherwise the physical offset is the value of CNTPOFF_EL2:

- EL3 is implemented and SCR_EL3.ECVEn is 0.
- EL2 is implemented and CNTHCTL_EL2.ECV is 0.

is updated to read:

If any of the following are true, the physical offset is zero, otherwise the physical offset is the value of CNTPOFF_EL2:

- FEAT_ECV is not implemented.
- EL2 is not implemented.
- EL3 is implemented and SCR_EL3.ECVEn is 0.
- CNTHCTL_EL2.ECV is 0.

Additionally, the following text is added after Table D16-11 ‘Captured timestamp’:

If EL2 is not implemented, then the Effective value of BRBCR_EL2.TS is `0b00`.

2.41 D20320

In section J1.1.4 (aarch64/instrs), the pseudocode within the function TLBIMatch() which reads:

```
// TLBIMatch()
// ===========
// Determine whether the TLB entry lies within the scope of inavlidation

boolean TLBIMatch(TLBIRecord tlbi, TLBRecord entry)
    boolean match;
    case tlbi.op of
    ...
        when TLBIOp_RIPAS2
            match = (!entry.context.includes_s1 && entry.context.includes_s2 &&
                            ... &&
                            UInt(tlbi.address) <= UInt(entry.context.ia) &&
                            UInt(tlbi.end_address) > UInt(entry.context.ia));
        when TLBIOp_RVAA
            match = (entry.context.includes_s1 &&
                            ... &&
                            UInt(tlbi.address) <= UInt(entry.context.ia) &&
                            UInt(tlbi.end_address) > UInt(entry.context.ia));
        when TLBIOp_RVA
            match = (entry.context.includes_s1 &&
```

```
                            ... &&
                            UInt(tlbi.address) <= UInt(entry.context.ia) &&
                            UInt(tlbi.end_address) > UInt(entry.context.ia));
        when TLBIOp_RPA
            match = (entry.context.includes_gpt &&
                            UInt(tlbi.address) <=
 UInt(entry.walkstate.baseaddress.address) &&
                            UInt(tlbi.end_address) >
 UInt(entry.walkstate.baseaddress.address));
```

is corrected to read:

```
// TLBIMatch()
// ===========
// Determine whether the TLB entry lies within the scope of invalidation

boolean TLBIMatch(TLBIRecord tlbi, TLBRecord entry)
    boolean match;
    bits(64) entry_block_mask = ZeroExtend(Ones(entry.blocksize), 64);
    bits(64) entry_end_address = entry.context.ia OR entry_block_mask;
    bits(64) entry_start_address = entry.context.ia AND NOT entry_block_mask;
    case tlbi.op of
        when TLBIOp_RIPAS2
            match = (!entry.context.includes_s1 && entry.context.includes_s2 &&
                            ... &&
                            UInt(tlbi.address<51:0>) <=
 UInt(entry_end_address<51:0>) &&
                            UInt(tlbi.end_address<51:0>) >
 UInt(entry_start_address<51:0>));
        when TLBIOp_RVAA
            match = (entry.context.includes_s1 &&
                            ... &&
                            UInt(tlbi.address<55:0>) <=
 UInt(entry_end_address<55:0>) &&
                            UInt(tlbi.end_address<55:0>) >
 UInt(entry_start_address<55:0>));
        when TLBIOp_RVA
            match = (entry.context.includes_s1 &&
                            ... &&
                            UInt(tlbi.address<55:0>) <=
 UInt(entry_end_address<55:0>) &&
                            UInt(tlbi.end_address<55:0>) >
 UInt(entry_start_address<55:0>));
        when TLBIOp_RPA
            entry_end_address<51:0> = (entry.walkstate.baseaddress.address<51:0> OR
                                       entry_block_mask<52:0>);
            entry_start_address<51:0> = (entry.walkstate.baseaddress.address<51:0>
 AND
                                         NOT entry_block_mask<51:0>);
            match = (entry.context.includes_gpt &&
                            UInt(tlbi.address<51:0>) <=
 UInt(entry_end_address<51:0>) &&
                            UInt(tlbi.end_address<51:0>) >
 UInt(entry_start_address<51:0>));
```

In section J1.1.4 (aarch64/instrs), the pseudocode within the function GPTTLBIMatch() which reads:

```
// GPTTLBIMatch()
// ==============
// Determine whether the GPT TLB entry lies within the scope of inavlidation

boolean GPTTLBIMatch(TLBIRecord tlbi, GPTEntry entry)
    assert tlbi.op IN {TLBIOp_RPA, TLBIOp_PAALL};
```

```
    boolean match;
    case tlbi.op of
        when TLBIOp_RPA
            match = (UInt(tlbi.address<51:entry.size>) <=
 UInt(entry.pa<51:entry.size>) &&
                                UInt(tlbi.end_address<51:entry.size>) >
 UInt(entry.pa<51:entry.size>));
        when TLBIOp_PAALL
            match = TRUE;

    return match;
```

is corrected to read:

```
// GPTTLBIMatch()
// ==============
// Determine whether the GPT TLB entry lies within the scope of invalidation

boolean GPTTLBIMatch(TLBIRecord tlbi, GPTEntry entry)
    assert tlbi.op IN {TLBIOp_RPA, TLBIOp_PAALL};

    boolean match;
    bits(64) entry_size_mask = ZeroExtend(Ones(entry.size), 64);
    bits(64) entry_end_address = ZeroExtend(entry.pa<51:0> OR entry_size_mask<51:0>,
 64);
    bits(64) entry_start_address = ZeroExtend(entry.pa<51:0> AND NOT
 entry_size_mask<51:0>, 64);

    case tlbi.op of
        when TLBIOp_RPA
            match = (UInt(tlbi.address<51:0>) <= UInt(entry_end_address<51:0>) &&
                                UInt(tlbi.end_address<51:0>) >
 UInt(entry_start_address<51:0>) &&
                                (tlbi.level == TLBILevel_Any || entry.level == 1));
        when TLBIOp_PAALL
            match = TRUE;

return match;
```

2.42 D20340

In section D12.11.3 (Common event numbers), in the subsection ‘Common microarchitectural events’, the event definition ‘`0x8174`, CAS_SPEC, Atomic memory Operation speculatively executed, Compare and Swap’ that reads:

> The counter counts each load atomic operation counted by LSE_LD_SPEC that is a Compare and Swap operation.

is updated to read:

> The counter counts each Compare and Swap operation.

2.43 C20341

In section D12.11.3 (Common events numbers), in the subsection 'Common microarchitectural events', the definition of '`0x8194`, DSNP_HIT_RD, Snoop hit, demand data read' that reads:

> The counter counts each snoop generated in response to a demand Memory-read operation counted by DSNP_HIT_RW that hits in a cache outside of the cache hierarchy of this PE.

is updated to read:

> The counter counts each snoop generated by the PE in response to a demand Memory-read operation counted by DSNP_HIT_RW that hits in and returns data from a cache outside of the cache hierarchy of this PE.
>
> Note: The event is counted by the PE generating the snoop, not the PE being snooped.

Equivalent changes are made to the ISNP_* and DSNP_* event descriptions throughout this section, although the Note is only added in the description of '`0x8190`, ISNP_HIT_RD, Snoop hit, demand instruction fetch'.

2.44 D20346

In section C5.6.1 (CFP RCTX, Control Flow Prediction Restriction by Context), in the description of 'EL, bits [25:24]', the text that reads:

> If the instruction is executed at an Exception level lower than the specified level, this instruction is treated as a **NOP**.

is updated to read:

> If the instruction is executed at an Exception level lower than the specified level, or is specified to apply to a combination of Exception level and Security state that are not implemented, this instruction is treated as a **NOP**.

Equivalent changes are made in the following sections:

- C5.6.2 (CPP RCTX, Cache Prefetch Prediction Restriction by Context), in the description of 'EL, bits [25:24]'.
- C5.6.3 (DVP RCTX, Data Value Prediction Restriction by Context), in the description of 'EL, bits [25:24]'.
- G8.2.26 (CFPRCTX, Control Flow Prediction Restriction by Context), in the description of 'EL, bits [25:24]'.
- G8.2.34 (CPPRCTX, Cache Prefetch Prediction Restriction by Context), in the description of 'EL, bits [25:24]'.
- G8.2.50 (DVPRCTX, Data Value Prediction Restriction by Context), in the description of 'EL, bits [25:24]'.

2.45 D20351

In section H9.2.21 (EDDEVID1, External Debug Device ID register 1), subsection 'HSR, bits [7:4]', the following text is added:

> FEAT_EDHSR implements the functionality identified by the value `0b0001`.

In section H9.2.28 (EDHSR, External Debug Halt Status Register), subsection 'Configuration', the text that reads:

> The field EDDEVID1.HSR indicates support for this register.

is updated to read:

> This register is present only when FEAT_EDHSR is implemented.

In section A2.4.3 (Features added to the Armv8.2 extension in later releases), the following text is added:

> FEAT_EDHSR, Support for EDHSR
>
> FEAT_EDHSR indicates that the External Debug Halting Syndrome Register is implemented. FEAT_EDHSR is OPTIONAL in Armv8.2 implementations and is mandatory in Armv8.9. EDDEVID1.HSR identifies the presence of this FEAT_EDHSR.

2.46 D20375

In section J1.3.1 (shared/debug), the functions Halt() and UpdateEDSCRFields() do not correctly update the EDSCR.SDD bit.

The code in Halt() that reads:

```
Halt(bits(6) reason, boolean is_async)
    ...
    EDSCR.ITO = '0';
    if HaveRME() then
        EDSCR.SDD = if ExternalRootInvasiveDebugEnabled() then '0' else '1';
    elsif CurrentSecurityState() == SS_Secure then
...
```

is updated to read:

```
Halt(bits(6) reason, boolean is_async)
    ...
    EDSCR.ITO = '0';
    if HaveRME() then
        if PSTATE.EL == EL3 then
            EDSCR.SDD = '0';
        else
            EDSCR.SDD = if ExternalRootInvasiveDebugEnabled() then '0' else '1';
```

```
    elsif CurrentSecurityState() == SS_Secure then
...
```

The code in UpdateEDSCRFields()that reads:

```
UpdateEDSCRFields()
        EDSCR.EL = '00';
        if HaveRME() then
            EDSCR.<NSE,NS> = bits(2) UNKNOWN;
        else
            EDSCR.NS = bit UNKNOWN;
        EDSCR.RW  ='1111';
```

is updated to read:

```
UpdateEDSCRFields()
        EDSCR.EL = '00';
        if HaveRME() then
           // SDD bit.
            EDSCR.SDD = if ExternalRootInvasiveDebugEnabled() then '0' else '1';
            EDSCR.<NSE,NS> = bits(2) UNKNOWN;
        else
            // SDD bit.
            EDSCR.SDD = if ExternalSecureInvasiveDebugEnabled() then '0' else '1';
            EDSCR.NS = bit UNKNOWN;
        EDSCR.RW = '1111';
...
```

2.47 D20378

In section E1.3.5 (Flushing denormalized numbers to zero), in the subsection ‘Flushing denormalized outputs to zero’, the text that reads:

> - If FPSCR.FZnstructions that convert from single-precision floating-point values to BF16 format flush denormalized outputs to zero.

is corrected to read:

> - If FPSCR.FZ is 1, instructions that convert from single-precision floating-point values to BF16 format flush denormalized outputs to zero.

2.48 D20397

In section B2.9.5 (Load-Exclusive and Store-Exclusive instruction usage restrictions), the text that reads:

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

1. Between the Load-Exclusive and the Store-Exclusive, there are no explicit memory effects, preloads, direct or indirect System register writes, address translation instructions, cache or TLB maintenance instructions, exception generating instructions, exception returns, ISB barriers, or indirect branches.
2. Between the Store-Exclusive returning a failing result and the retry of the corresponding Load-Exclusive:

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

is updated to read:

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

1. Between the Load-Exclusive and the Store-Exclusive, there are no explicit memory effects, preloads, direct or indirect System register writes, address translation instructions, cache or TLB maintenance instructions, exception generating instructions, exception returns, ISB barriers, indirect branches, or Branch with Link instructions.
2. Between the Store-Exclusive returning a failing result and the retry of the corresponding Load-Exclusive:

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

Equivalent changes are made in section E2.10.5 (Load-Exclusive and Store-Exclusive instruction usage restrictions).

2.49 D20441

In section D19.2.34, (CTR_EL0, Cache Type Register), subsection 'DIC, bit [29]', the following text is added:

> All PEs in the same Inner Shareable shareability domain must have a common value of this field.

In the same section, subsection 'IDC, bit [28]', the following text is added:

> If CTR_EL0.DIC is 1 then the value reported in this field must also be 1. The Effective value of IDC is (CTR_EL0.IDC or CLIDR_EL1.LoC == `0b000` or (CLIDR_EL1.LoUIS == `0b000` && CLIDR_EL1.LoUU == `0b000`)). All PEs in the same Inner Shareable shareability domain must have a common Effective value of IDC.

The equivalent changes are made in the following sections:

- Section G8.2.36, (CTR, Cache Type Register), subsection 'DIC, bit [29]'
- Section G8.2.36, (CTR, Cache Type Register), subsection 'IDC, bit [28]'

In section B2.4.4, (Implication of caches for the application programmer), subsection 'Synchronization and coherency issues between data and instruction accesses', the following text is removed:

> CTR_EL0.{DIC, IDC} == {1, 0} The location written to has been cleaned to the Point of unification (PoU) from the data cache, and that clean is complete for the shareability domain.

The equivalent changes are made in the following section:

- Section E2.3.3, (Ordering of instruction fetches)

2.50 D20444

In section C5.3.25 (DC GZVA, Data Cache set Allocation Tags and Zero by VA), subsection 'Executing DC GZVA' the text that reads:

> If the memory region being zeroed is any type of Device memory, this instruction can give an alignment fault which is prioritized in the same way as other alignment faults that are determined by the memory type.

is changed to read:

> If the memory region being zeroed is any type of Device memory, this instruction generates an alignment fault which is prioritized in the same way as other alignment faults that are determined by the memory type.

Similar changes are made in the following sections:

- C5.3.32 (DC ZVA, Data Cache Zero by VA).
- C5.3.24 (DC GVA, Data Cache set Allocation Tag by VA).

2.51 C20451

In section D10.7 (PE handling of Tag check fault), the text that reads:

> When FEAT_SVE is implemented, if a load of an element in a SVE Non-faulting or First-faulting load instruction causes a Tag Check Fault, and is the First active element in a First-faulting instruction, the Tag Check Fault:
>
> - ...

is changed to read:

> When FEAT_SVE is implemented, if a load of an element in a First-faulting load instruction causes a Tag Check Fault, and is the First active element, the Tag Check Fault:
>
> - ...

2.52 C20503

In section D10.4 (Tagged and Untagged Addresses), the text that reads:

> D10.4 Tagged and Untagged Addresses
>
> Virtual addresses can either be Tagged or Untagged.
>
> An access to memory at:
>
> - An Untagged virtual address generates a Tag Unchecked access.
> - A Tagged virtual address permits the generation of a Tag Checked or Tag Unchecked access.
>
> A read of an Allocation Tag from an Untagged virtual address returns the value `0b0000`.
>
> A write of an Allocation Tag to an Untagged address is IGNORED.
>
> Accesses of Allocation Tags at Tagged virtual addresses are permitted.
>
> All virtual addresses in AArch32 state are Untagged.
>
> D10.4.1 Virtual address translation
>
> If stage 1 translation at the current Exception level is enabled, stage 1 translations are Tagged or Untagged depending on the Memory Attributes for the memory location being accessed.
>
> If stage 1 translation is disabled for the EL1&0 translation regime:
>
> - If the value of HCR_EL2.DC is 1, stage 1 translations are Tagged or Untagged depending on the value of HCR_EL2.DCT.

- If the value of HCR_EL2.DC is 0, stage 1 translations are treated as Untagged. For all other translation regimes, if stage 1 translation is disabled, stage 1 translations are treated as Untagged.

Memory locations are treated as Tagged where all of the following is true:

- The combined effects of stage 1 and stage 2 translations define the memory attributes as:
  - Normal memory.
  - Inner, and Outer Write-Back Non-Transient Read-Allocate Write-Allocate.
- The stage 1 translation is treated as Tagged.

Otherwise memory locations are Untagged.

If a memory location is marked as Untagged, a data cache invalidation operation that would invalidate Allocation Tags at that location cleans and invalidates the Allocation Tags.

Note: If a memory location is marked as both Tagged and Non-shared, it is **IMPLEMENTATION DEFINED** whether the memory location is treated as Tagged or Untagged.

When the EL1&0 stage 1 translation regime is disabled and HCR_EL2.DC is 1, in the current Security state, the execution of any of the AT S1E0, AT S1E1, AT S12E0, AT S12E1 address translation instructions will reflect the effect of HCR_EL2.DCT in PAR_EL1.ATTR.

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 behavior, software can set SCTLR_ELx.ATA to 0 when SCTLR_ELx.C is 0.

For more information on Virtual address translation, see Address translation on page D8-5806.

is updated to read:

D10.4 Tagged and Untagged memory locations

A memory location is either Tagged or Untagged.

A read from an Allocation tag at an Untagged memory location returns the value `0b0000`.

A write to an Allocation tag at an Untagged memory location does not modify the Allocation tag.

There are no instructions to access Allocation Tags in AArch32.

A memory location is Tagged if all the following apply, otherwise it is Untagged:

- For an EL1&0 translation regime the combined effect of stage 1 and stage 2 translations, and for other translation regimes stage 1 translation, defines the memory attributes as:

  - Tagged.
  - Normal.
  - Write-back cacheable Non-Transient, Read-Allocate, Write-allocate.
- Allocation tag access is enabled.

For more information on Virtual address translation, see Address translation on page D8-5806.

For more information on when Allocation tag access is enabled see Enabling the Memory Tagging Extension.

If a memory location is both Tagged and Non-shareable it is **IMPLEMENTATION DEFINED** whether the memory location is treated as Tagged or Untagged.

For the EL1&0 translation regime, if stage 1 translation is disabled and HCR_EL2.DC is 1, in the current Security state, execution of any of the AT S1E0, AT S1E1, AT S12E0, AT S12E1 address translation instructions will reflect the effect of HCR_EL2.DCT in PAR_EL1.ATTR.

If SCTLR_ELx.C is 0 for a translation regime, it is **CONSTRAINED UNPREDICTABLE** whether a Tagged memory location is treated as Tagged or Untagged.

Note: To ensure consistent behavior, software can set SCTLR_ELx.ATA to 0 when SCTLR_ELx.C is 0.

In section D10.8.1 (Tag Unchecked accesses), the following text is added:

> An access to an Untagged memory location generates a Tag Unchecked access.

In section D10.5 (PE access to Allocation Tags), the text that reads:

> A read of an Allocation Tag that returns zero due to access to Allocation tags being disabled by HCR_EL2.ATA, SCR_EL3.ATA or SCTLR_ELx.{ATA, ATA0}, or due to the memory type not having the Tagged attribute, is permitted to generate an External abort if a read of data from the same address would generate an External abort.

is updated to read:

> A read of an Allocation Tag from an Untagged memory location is permitted to generate an External abort if a read of data from the same memory location would generate an External abort.

In section D19.2.48 (HCR_EL2, Hypervisor Configuration Register), the text in the DCT field description that reads:

> When HCR_EL2.DC is in effect, controls whether stage 1 translations are treated as Tagged or Untagged.
>
> `0b0` Stage 1 translations are treated as Untagged. `0b1` Stage 1 translations are treated as Tagged.

is updated to read:

When HCR_EL2.DC is in effect, controls whether Stage 1 translations have the Tagged attribute.

`0b0` Stage 1 translations do not have the Tagged attribute. `0b1` Stage 1 translations have the Tagged attribute.

2.53 D20506

In section D8.8.3 (PAC instructions), rule R $_{ZYPJV}$ that reads:

For the PACGA instruction, if the PAC is generated using an **IMPLEMENTATION DEFINED** algorithm, then all of the following are required:

- The **IMPLEMENTATION DEFINED** algorithm uses the same arguments as the ComputePAC() pseudocode function.
- For a set of arguments passed to the **IMPLEMENTATION DEFINED** algorithm, the same result is produced by all PEs that an execution thread could migrate between.

For more information, see aarch64/functions/pac/computepac/ComputePAC on page J1-12065.

is updated to read:

R $_{ZYPJV}$

If the PAC is generated using an **IMPLEMENTATION DEFINED** algorithm, then the **IMPLEMENTATION DEFINED** algorithm uses the same arguments as the ComputePAC() pseudocode function.

and the following rule is added below:

R $_{JTRCS}$

For a set of arguments passed to the ComputePAC() pseudocode function, the same result is produced by all PEs that an execution thread could migrate between.

For more information, see aarch64/functions/pac/computepac/ComputePAC on page J1-12065.

2.54 C20514

In section D8.4.1 (Effect of PSTATE on access permission), in the subsection ‘PSTATE.BTYPE’, the text that reads:

I $_{CKJFH}$

The BTI instruction is a **NOP** in a non-guarded page.

is updated to read:

I $_{CKJFH}$

> In a non-guarded page, the BTI instruction executes as a **NOP**.
>
> I $_{X0001}$
>
> The effect of a **NOP** on PSTATE.BTYPE is described in R $_{YWFHD}$.

2.55 C20530

In section D1.6.1 (Wait for Event), rule R $_{TJTFC}$ that reads:

> Except for all or the following, the architecture does not define the exact nature of the low-power state:
>
> - When a WFE or WFET instruction is executed, the architecture requires that memory coherency is not lost.
> - If the system is configured such that the WFE or WFET instruction can be completed, then the architecture requires that the architectural state is not lost.

is updated to read:

> The architecture does not define the exact nature of the low-power state entered by WFE or WFET, except that when a WFE or WFET instruction is executed, memory coherency and architectural state are not lost.

2.56 D20542

In section D8.8.3 (PAC instructions), the statement I$_{RHMHV}$ that reads:

> If PAC generation and validation is disabled, all of the following are examples of the behavior of instructions that combine pointer authentication with another operation:
>
> - A RETAA instruction operates as a RET instruction.
> - A LDRAA Xt, [Xn, #<simm10>]! instruction operates as a LDR Xt, [Xn, #<simm10>:000]! instruction.

is changed to read:

> If PAC generation and validation is disabled, then all of the following are examples of the resulting behavior of instructions that combine pointer authentication with another operation:
>
> - A RETAA instruction operates as a RET instruction.
> - A LDRAA Xt, [Xn, #<simm10>]! instruction operates as a LDR Xt, [Xn, #<simm10>]! instruction.

2.57 D20578

In section D19.2.37 (ESR_EL1, Exception Syndrome Register (EL1)), subsection 'ISS encoding for an exception from a Data Abort', in the field description for 'Bits [12:11]' the condition for the LST field which reads:

> When (DFSC == 0b00xxxx || DFSC == `0b101011`) && DFSC != 0b0000xx:

is updated to read:

> When (DFSC == 0b00xxxx || DFSC == 0b10101x) && DFSC != 0b0000xx:

The same change is made in sections D19.2.38 (ESR_EL2, Exception Syndrome Register (EL2)) and D19.2.39 (ESR_EL3, Exception Syndrome Register (EL3)).

2.58 C20580

In section D8.4.1 (Effect of PSTATE on access permission), in the subsection 'PSTATE.BTYPE', the statement that reads:

> I_{GMGRS}
>
> When PACIASP and PACIBSP instructions access a guarded memory region, the instructions have an implicit branch target identification instruction that is compatible with one of the following:
>
> - The PSTATE.BTYPE value is `0b10` or `0b01`.
> - When the associated SCTLR_ELx.{BT0, BT1, BT} bits are 0, the PSTATE.BTYPE value is `0b11`.

is clarified to read:

> I_{GMGRS}
>
> The PACIASP and PACIBSP instructions have an implicit BTI behavior that is compatible with any of the following:
>
> - The PSTATE.BTYPE value is `0b10` or `0b01`.
> - The PSTATE.BTYPE value is `0b11`, and one of the following:
>   - The PE is executing at EL0, the value of HCR_EL2.{E2H,TGE} is not {1,1}, and SCTLR_EL1.BT0 is 0.
>   - The PE is executing at EL0, the value of HCR_EL2.{E2H,TGE} is {1,1}, and SCTLR_EL2.BT0 is 0.
>   - The PE is executing at EL1, and SCTLR_EL1.BT1 is 0.
>   - The PE is executing at EL2, the value of HCR_EL2.{E2H,TGE} is not {1,1}, and SCTLR_EL2.BT is 0.
>   - The PE is executing at EL2, the value of HCR_EL2.{E2H,TGE} is {1,1}, and SCTLR_EL2.BT1 is 0.

- The PE is executing at EL3, and SCTLR_EL3.BT is 0.

2.59 C20583

In section D10.3 (Tag checking), the text that reads:

> A memory access that is a read or write can be either Tag Checked or Tag Unchecked. An access to the data PA space can be either Tag Checked or Tag Unchecked. An access to the tag PA space is always Tag Unchecked. A data access which is performed as part of a prefetch operation is Tag Unchecked. When the value of PSTATE.TCO is 1, all loads and stores are Tag Unchecked. A Tag Checked memory access includes a Physical Address Tag.

is changed to read:

> A memory access that is a read or write can be either Tag Checked or Tag Unchecked. Bits [59:56] of a 64-bit VA used for a memory access define a Logical Address Tag. A Tag Checked memory access includes a Physical Address Tag generated from the Logical Address Tag for the memory access.

Also in section D10.8 (PE generation of Tag Checked and Tag Unchecked accesses), and section D10.8.1 (Tag Unchecked accesses), the text that reads:

> D10.8 PE generation of Tag Checked and Tag Unchecked accesses
>
> A Logical Address Tag is formed by bits [59:56] of the 64-bit address that is used for a load or store instruction. The PE generates a Physical Address Tag from the Logical Address Tag for each Tag Checked access to memory. Unless an access is explicitly defined as a Tag Unchecked access, it is a Tag Checked access. Instructions in Debug state follow the same rules for generation of Tag Checked and Tag Unchecked accesses as in Non-Debug state. See Chapter H2 Debug State for more information.
>
> D10.8.1 Tag Unchecked accesses
>
> The following operations generate a Tag Unchecked access:
>
> - An instruction fetch.
> - A load instruction that loads an Allocation Tag.
> - A store instruction that stores an Allocation Tag.
>
> When PSTATE.TCO is 1, all loads and stores generate Tag Unchecked accesses.
>
> A cache maintenance by virtual address operation other than DC ZVA, Data Cache Zero by VA, generates a Tag Unchecked access.
>
> An access due to a translation table walk generates a Tag Unchecked access.

> If FEAT_NV2 is implemented, loads and stores relative to VNCR_EL2 generate a Tag Unchecked access.
>
> If the Statistical Profiling Extension is implemented, all accesses to the Profiling Buffer are Tag Unchecked accesses. See Chapter D14 The Statistical Profiling Extension for more information.
>
> An access which would be translated using TTBR0_ELx is Tag Unchecked, irrespective of whether the stage 1 address translation for the ELx translation regime is enabled or not, where either of the following conditions apply:
>
> - TCR_ELx.TBI is 0.
> - TCR_ELx.TBI0 is 0.
>
> If TCR_ELx.TBI1 has the value of zero, an access which would be translated using TTBR1_ELx is Tag Unchecked, irrespective of whether the stage 1 address translation for the ELx translation regime is enabled or not.
>
> An access will be Tag Unchecked, irrespective of whether the stage 1 address translation for the ELx translation regime is enabled or not, where all of the following conditions apply:
>
> - The access would be translated using TTBR0_ELx.
> - The Logical Address Tag is `0b0000`.
> - TCR_ELx.TCMA is 1, or TCR_ELx.TCMA0 is 1.
>
> An access will be Tag Unchecked, irrespective of whether the stage 1 address translation for the ELx translation regime is enabled or not, when all of the following conditions apply:
>
> - The access would be translated using TTBR1_ELx.
> - The Logical Address Tag is `0b1111`.
> - TCR_ELx.TCMA1 is 1.
>
> A Tag Unchecked access will be generated for a load or store that uses either of the following:
>
> - A base register only, with the SP as the base register.
> - A base register plus immediate offset addressing form, with the SP as the base register.
>
> Literal (PC-relative) loads generate a Tag Unchecked access.

is changed to read:

> D10.8 PE generation of Tag Checked and Tag Unchecked accesses
>
> A memory access is Tag Checked unless it is Tag Unchecked due to any of the following:
>
> - The access an instruction fetch.
> - The access is to an Untagged memory location.
> - The access is by an instruction that directly loads or stores an Allocation Tag.
> - The access is a read of an Allocation Tag due to a Tag check operation.

- PSTATE.TCO is 1.
- The access is due to a cache maintenance operation by virtual address operation other than DC ZVA, Data Cache Zero by VA.
- The access is due to a translation table walk.
- If FEAT_NV2 is implemented, the access is a load or store relative to VNCR_EL2.
- If the Statistical Profiling Extension is implemented, the access is to the Profiling Buffer. See Chapter D14 The Statistical Profiling Extension for more information.
- Address Tagging is disabled for the memory location.
- Irrespective of whether the stage 1 address translation for the ELx translation regime is enabled or not, where all of the following conditions apply:
  - The Logical Address Tag is `0b0000`.
  - If the stage 1 translation supports a single VA range, TCR_ELx.TCMA is 1.
  - If the stage 1 translation supports two VA ranges, TCR_ELx.TCMA0 is 1 and the access is to the lower address range.
- Irrespective of whether the stage 1 address translation for the ELx translation regime is enabled or not, where all of the following conditions apply:
  - The Logical Address Tag is `0b1111`.
  - The stage 1 translation supports two VA ranges.
  - TCR_ELx.TCMA1 is 1 and the access is to the upper address range.
- The access is by an instruction that uses any of the following addressing modes:
  - A base register only, with the SP as the base register.
  - A base register plus immediate offset addressing form, with the SP as the base register.
  - Literal (PC-relative).

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

2.60 D20589

In section D1.3.2 (Exception entry), subsection 'SVE MOVPRFX exception entry behavior', the rule that reads:

R_{RWVTR}

When a MOVPRFX instruction pairs legally with another instruction and the execution of the pair generates a synchronous exception, the return address that is stored in ELR_ELx is one of the following:

- When the MOVPRFX instruction did not cause a change to the architectural state, the address of the MOVPRFX instruction is stored.

- When the MOVPRFX instruction caused a change to the architectural state, the address of the prefixed instruction is stored.

is changed to read:

R_{RWVTR}

When a MOVPRFX instruction pairs legally with another instruction and the execution of the pair generates a synchronous exception:

- If the generated exception is a Breakpoint Instruction exception from a prefixed BRK instruction then MOVPRFX is required to update the architectural state and ELR_ELx is required to store the address of BRK instruction.
- Otherwise, the return address that is stored in ELR_ELx is one of the following:
  - When the MOVPRFX instruction did not cause a change to the architectural state, the address of the MOVPRFX instruction is stored.
  - When the MOVPRFX instruction caused a change to the architectural state, the address of the prefixed instruction is stored.

Similarly, the rule within the same section that reads:

R_{XRWVD}

When a MOVPRFX instruction pairs legally with another instruction and the execution of the pair causes entry to Debug state, the return address that is stored in DLR_EL0 is one of the following:

- When the MOVPRFX instruction did not cause a change to the architectural state, the address of the MOVPRFX instruction is stored.
- When the MOVPRFX instruction caused a change to the architectural state, the address of the prefixed instruction is stored.

is changed to read:

R_{XRWVD}

When a MOVPRFX instruction pairs legally with another instruction and the execution of the pair causes synchronous entry to Debug state:

- If the Debug state entry is due to a Halt Instruction debug event from a prefixed HLT instruction then MOVPRFX is required to update the architectural state and DLR_EL0 is required to store the address of the HLT instruction.
- Otherwise, the return address that is stored in DLR_EL0 is one of the following:
  - When the MOVPRFX instruction did not cause a change to the architectural state, the address of the MOVPRFX instruction is stored.
  - When the MOVPRFX instruction caused a change to the architectural state, the address of the prefixed instruction is stored.

2.61 R20604

In section D12.11.3 (Common event numbers), in the subsection 'Common microarchitectural events', the statement in event '0x8140, L1D_CACHE_RW, Level 1 data cache demand access' that reads:

> This event must be implemented if any of the following are true:
> - Event L1D_CACHE_PRFM is implemented.
> - Event L1D_CACHE_HWPRF is implemented.

is updated to read:

> When any of the following are true, Arm recommends this event is implemented:
> - Event L1D_CACHE_PRFM is implemented.
> - Event L1D_CACHE_HWPRF is implemented.

A similar change is also made to the following events within the same section:

- 0x8130, L1D_TLB_RW, Level 1 data TLB demand access.
- 0x8131, L1I_TLB_RD, Level 1 instruction TLB demand access.
- 0x8132, L1D_TLB_PRFM, Level 1 data TLB software preload.
- 0x8133, L1I_TLB_PRFM, Level 1 instruction TLB software preload.
- 0x8141, L1I_CACHE_RD, Level 1 instruction cache demand fetch.
- 0x8142, L1D_CACHE_PRFM, Level 1 data cache software preload.
- 0x8143, L1I_CACHE_PRFM, Level 1 instruction cache software preload.
- 0x8148, L2D_CACHE_RW, Level 2 data cache demand access.
- 0x8149, L2I_CACHE_RD, Level 2 instruction cache demand fetch.
- 0x814A, L2D_CACHE_PRFM, Level 2 data cache software preload.
- 0x814B, L2I_CACHE_PRFM, Level 2 instruction cache software preload.
- 0x8150, L3D_CACHE_RW, Level 3 data cache demand access.
- 0x8151, L3D_CACHE_PRFM, Level 3 data cache software preload.
- 0x8298, LL_CACHE_RW, Last level cache demand access.
- 0x8299, LL_CACHE_PRFM, Last level cache software preload.

2.62 R20607

In section D19.2.72 (ID_AA64SMFR0_EL1, SME Feature ID register 0), the accessibility pseudocode at EL1 that reads:

```
elsif PSTATE.EL == EL1 then
```

```
    if EL2Enabled() && HCR_EL2.TID3 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        X[t, 64] = ID_AA64SMFR0_EL1;
```

is updated to read:

```
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && (IsFeatureImplemented(FEAT_FGT) || !IsZero(ID_AA64SMFR0_EL1)
 || boolean IMPLEMENTATION_DEFINED "ID_AA64SMFR0_EL1 trapped by HCR_EL2.TID3") &&
 HCR_EL2.TID3 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        X[t, 64] = ID_AA64SMFR0_EL1;
```

Equivalent changes are made in section D19.2.68 (ID_AA64MMFR4_EL1, AArch64 Memory Model Feature Register 4).

2.63 D20616

In section C5.3.10 (DC CIGDPAE, Clean and invalidate of data and allocation tags by PA to PoE), the accessibility pseudocode that reads:

```
elsif PSTATE.EL == EL2 then
AArch64.DC(X[t, 64], CacheType_Data, CacheOp_CleanInvalidate, CacheOpScope_PoE);
```

is corrected to read:

```
elsif PSTATE.EL == EL2 then
if !IsCurrentSecurityState(SS_Realm) then
UNDEFINED;
else
AArch64.DC(X[t, 64], CacheType_Data, CacheOp_CleanInvalidate, CacheOpScope_PoE);
```

An equivalent change is made in the accessibility pseudocode of section C5.3.16 (DC CIPAE, Data or unified Cache line Clean and Invalidate by PA to PoE).

2.64 D20617

In section D19.11.2 (MECID_A0_EL2, Alternate MECID for EL2 and EL2&0 translation regimes), the accessibility pseudocode at EL2 that reads:

```
elsif PSTATE.EL == EL2 then
    if !IsCurrentSecurityState(SS_Realm) then
        UNDEFINED;
    else
        X[t, 64] = MECID_A0_EL2;
```

is corrected to read:

```
elsif PSTATE.EL == EL2 then
    if !IsCurrentSecurityState(SS_Realm) then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.MECEn == '0' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = MECID_A0_EL2;
```

Equivalent changes are made in the following sections:

- D19.11.3 (MECID_A1_EL2, Alternate MECID for EL2&0 translation regimes).
- D19.11.4 (MECID_P0_EL2, Primary MECID for EL2 and EL2&0 translation regimes).
- D19.11.5 (MECID_P1_EL2, Primary MECID for EL2&0 translation regimes).
- D19.11.7 (VMECID_A_EL2, Alternate MECID for EL1&0 stage 2 translation regime).
- D19.11.8 (VMECID_P_EL2, Primary MECID for EL1&0 stage 2 translation regime).

2.65 D20618

In section C5.3.10 (DC CIGDPAE, Clean and invalidate of data and allocation tags by PA to PoE), the accessibility pseudocode at EL1 that reads:

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

is corrected to read:

```
elsif PSTATE.EL == EL1 then
    UNDEFINED;
```

An equivalent change is made in section C5.3.16 (DC CIPAE, Data or unified Cache line Clean and Invalidate by PA to PoE).

2.66 C20625

In section D19.7.3 (PMBPTR_EL1, Profiling Buffer Write Pointer Register), in the description of 'PTR, bits [63:0]', the text that reads:

> The architecture places restrictions on the values software can write to the pointer. For more information see Restrictions on the current write pointer on page D14-6212.

> Note: As a result, an implementation might treat some of bits[M:0], where M is defined by PMBIDR_EL1.Align, as **RES0**.

is updated to read:

> If PMBIDR_EL1.Align is not zero, then it is **IMPLEMENTATION DEFINED** whether bits [M-1:0] are **RES0** or read/write, where M is an integer between 1 and PMBIDR_EL1.Align inclusive.
>
> The architecture places restrictions on the values software can write to the pointer when the SPU is not in Discard mode. For more information see Restrictions on the current write pointer on page D14-6212.

A similar correction is made in section D19.4.5 (TRBPTR_EL1, Trace Buffer Write Pointer Register), where the text in the description of 'PTR, bits [63:0]' that reads:

> The architecture places restrictions on the values that software can write to the pointer.
>
> Note: As a result of the restrictions an implementation might treat some of PTR[M:0] as **RES0**, where M is defined by TRBIDR_EL1.Align.

is updated to read:

> If TRBIDR_EL1.Align is not zero, then it is **IMPLEMENTATION DEFINED** whether bits [M-1:0] are **RES0** or read/write, where M is an integer between 1 and TRBIDR_EL1.Align inclusive.
>
> The architecture places restrictions on the values that software can write to the pointer. For more information see Restrictions on programming the Trace Buffer Unit on page D6-5736.

In section D14.7 (The Profiling Buffer) the text that reads:

> The profile data is collected in a memory Profiling Buffer.

is updated to read:

> When the SPU is not in Discard mode, profile data is collected in a memory Profiling Buffer.

2.67 D20635

In section A2.5.3 (Features added to the Armv8.3 extension in later releases), subsection 'FEAT_SPEv1p1, Armv8.3 Statistical Profiling Extensions', the text that reads:

> This feature is OPTIONAL in Armv8.3 implementations. An Armv8.5 implementation that includes the Statistical Profiling Extension must include FEAT_SPEv1p1.

is updated to read:

> This feature is OPTIONAL in Armv8.3 implementations. An Armv8.5 implementation that includes the Statistical Profiling Extension must include FEAT_SPEv1p1. An implementation that

> includes FEAT_SVE and the Statistical Profiling Extension is strongly recommended to implement FEAT_SPEv1p1 whenever possible.

2.68 C20643

In section B2.7.2 (Device memory), in the subsection 'Gathering', the following statement:

> Gathering between two memory accesses generated by a Load-Acquire/Store-Release is not permitted.

is clarified to read:

> Gathering between the memory accesses generated by one Load-Acquire or Store-Release instruction, and the memory accesses generated by another Load-Acquire or Store-Release instruction is not permitted.

2.69 D20644

In section C6.2.316 (STLXP), the operational pseudocode that reads:

```
bits(64) address;
bits(datasize) data;
...
AccessDescriptor accdesc = CreateAccDescExLDST(MemOp_STORE, TRUE, tagchecked);
...
if AArch64.ExclusiveMonitorsPass(address, dbytes) then
    ...
    Mem[address, dbytes, accdesc] = data;
...
```

is updated to read as :

```
bits(64) address;
bits(datasize) data;
...
AccessDescriptor accdesc = CreateAccDescExLDST(MemOp_STORE, TRUE, tagchecked);
...
if AArch64.ExclusiveMonitorsPass(address, dbytes, accdesc) then
    ...
    Mem[address, dbytes, accdesc] = data;
...
```

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

```
 boolean AArch64.ExclusiveMonitorsPass(bits(64) address, integer size)
     ...
     boolean acqrel = FALSE;
     boolean tagchecked = FALSE;
     AccessDescriptor accdesc = CreateAccDescExLDST(MemOp_STORE, acqrel,
 tagchecked);
     boolean aligned = IsAligned(address, size);
```

```
      if !aligned && AArch64.UnalignedAccessFaults(accdesc, address, size) then
         AArch64.Abort(address, AlignmentFault(accdesc));
```

is updated to read as:

```
boolean AArch64.ExclusiveMonitorsPass(bits(64) address, integer size,
 AccessDescriptor accdesc)
      ...
      boolean aligned = IsAligned(address, size);

      if !aligned && AArch64.UnalignedAccessFaults(accdesc, address, size) then
         AArch64.Abort(address, AlignmentFault(accdesc));
```

Similar changes are made in the operational pseudocode of the following A64 instructions:

- C6.2.317 (STLXR).
- C6.2.318 (STLXRB).
- C6.2.319 (STLXRH).
- C6.2.349 (STXP).
- C6.2.350 (STXR).
- C6.2.351 (STXRB).
- C6.2.352 (STXRH).

2.70 D20664

In section D19.2.38 (ESR_EL2, Exception Syndrome Register (EL2)), in the EC field value `0b011010`, value name 'ISS encoding for an exception from an ERET, ERETAA, or ERETAB instruction', the condition in the EC table that reads:

> When FEAT_PAuth is implemented and FEAT_NV is implemented:

is updated to read:

> When FEAT_FGT is implemented or FEAT_NV is implemented:

2.71 D20668

In section D19.2.138 (TCR2_EL2, Extended Translation Control Register (EL2)), subsection 'When HCR_EL2.E2H == 0:' the field 'AMEC1, bit[13]' is removed and added to subsection 'When HCR_EL2.E2H == 1:'. In addition, the text in this field that reads:

> This field controls the enabling of the Alternate MECID translations for the EL2&0 TTBR1 translation regime.

> TCR2_EL2.AMEC1 is provided to enable the safe update of TTBR_EL2 and MECID_A_EL2, by disabling access and speculation to AMEC == 1 Block or Page descriptors during the update.

is updated to read:

> This field controls the enabling of the Alternate MECID translations for accesses in the TTBR1_EL2 half of the VA range in the EL2&0 translation regime.
>
> TCR2_EL2.AMEC1 is provided to enable the safe update of TTBR1_EL2 and MECID_A1_EL2, by disabling access and speculation to AMEC == 1 Block or Page descriptors during the update.

Also, in field 'AMEC0, bit [12]', the text that reads:

> This field controls the enabling of the Alternate MECID translations for the EL2 and EL2&0 TTBR0 translation regimes.
>
> TCR2_EL2.AMEC0 is provided to enable the safe update of TTBR_EL2 and MECID_A_EL2, by disabling access and speculation to AMEC=1 Block or Page descriptors during the update.

is updated to read:

> This field controls the enabling of the Alternate MECID translations for the EL2 translation regime.
>
> TCR2_EL2.AMEC0 is provided to enable the safe update of TTBR0_EL2 and MECID_A0_EL2, by disabling access and speculation to AMEC=1 Block or Page descriptors during the update.

2.72 D20675

In section D19.2.53 (HFGRTR_EL2, Hypervisor Fine-Grained Read Trap Register), the text in the description of 'ERXPFGF_EL1, bit [46]' that reads:

> When FEAT_RAS is implemented:

is corrected to read:

> When FEAT_RASv1p1 is implemented:

2.73 D20678

In section I6.9.24 (ERR<n>FR, Error Record <n> Feature Register, n = 0 - 65534), the text that reads:

> CFI, bits [11:10]
>
> When ERR<n>FR.FI != `0b00`

is corrected to read:

> CFI, bits [11:10]
>
> When ERR<n>FR.FI != 0b0x

2.74 D20682

In section H2.4.2 (Executing instructions in Debug state), in the subsection 'Instructions that explicitly write to the PC (branches)', the following bullet point is added:

> - When FEAT_PAuth is implemented, RETAA, RETAB, BRAA, BRAB, BLRAA, BLRAB, BLRAAZ, BLRABZ.

Also, in the subsection 'Exception return and related instructions', the text that reads:

> This instruction is:
> - ERET.

is updated to read:

> These instructions are:
> - ERET.
> - When FEAT_PAuth is implemented, ERETAA, ERETAB.

2.75 D20684

In section D19.2.48 (HCR_EL2, Hypervisor Configuration Register) field 'EnSCXT, bit [53]', the text that reads:

> `0b0` When HCR_EL2.E2H is 0 or HCR_EL2.TGE is 0, and EL2 is enabled in the current Security state, EL1 and EL0 access to SCXTNUM_EL0 and EL1 access to SCXTNUM_EL1 is disabled by this mechanism, causing an exception to EL2, and the values of these registers to be treated as 0.
>
> When HCR_EL2.{E2H, TGE} is {1, 1} and EL2 is enabled in the current Security state, EL0 access to SCXTNUM_EL0 is disabled by this mechanism, causing an exception to EL2, and the value of this register to be treated as 0.
>
> `0b1` This control does not cause accesses to SCXTNUM_EL0 or SCXTNUM_EL1 to be trapped.
>
> When FEAT_VHE is implemented, and the value of HCR_EL2.{E2H, TGE} is {1,1}, this bit has no effect on execution at EL0.

is replaced by the following text:

`0b0` When EL2 is enabled in the current Security state, EL1 accesses to SCXTNUM_EL0 and SCXTNUM_EL1 are disabled, causing an exception to EL2, and the value of the registers to be treated as 0.

When HCR_EL2.E2H is 0 or HCR_EL2.TGE is 0, and EL2 is enabled in the current Security state, EL0 access to SCXTNUM_EL0 is disabled, causing an exception to EL2, and the value of the register to be treated as 0.

`0b1` This control does not cause accesses to SCXTNUM_EL0 or SCXTNUM_EL1 to be trapped.

Note: When FEAT_VHE is implemented, the value of HCR_EL2.{E2H, TGE} is {1,1}, and the value of this field is `0b0`, accesses at EL0 are not trapped by this control.

2.76 D20689

In section H9.2.42 (EDPRSR, External Debug Processor Status Register), in the field 'EPMAD, bit [9]', the text that reads:

Otherwise:

Reserved, **RES0**.

is changed to read:

When FEAT_PMUv3 is implemented and FEAT_PMUv3_EXT is not implemented:

Reserved, **UNKNOWN**.

Otherwise:

Reserved, **RES0**.

2.77 D20692

In section D19.2.48 (HCR_EL2, Hypervisor Configuration Register), in the description of 'TID3, bit [18]', the text that reads:

Trap ID group 3. Traps EL1 reads of group 3 ID registers to EL2, when EL2 is enabled in the current Security state, as follows:

In AArch64 state:

- Reads of the following registers are trapped to EL2, reported using EC syndrome value `0x18`:
- ID_PFR0_EL1, ID_PFR1_EL1, ID_PFR2_EL1, ID_DFR0_EL1, ID_AFR0_EL1, ID_MMFR0_EL1, ID_MMFR1_EL1, ID_MMFR2_EL1, ID_MMFR3_EL1, ID_ISAR0_EL1, ID_ISAR1_EL1,

ID_ISAR2_EL1, ID_ISAR3_EL1, ID_ISAR4_EL1, ID_ISAR5_EL1, MVFR0_EL1, MVFR1_EL1, MVFR2_EL1.

- ID_AA64PFR0_EL1, ID_AA64PFR1_EL1, ID_AA64DFR0_EL1, ID_AA64DFR1_EL1, ID_AA64ISAR0_EL1, ID_AA64ISAR1_EL1, ID_AA64MMFR0_EL1, ID_AA64MMFR1_EL1, ID_AA64AFR0_EL1, ID_AA64AFR1_EL1.
- If FEAT_FGT is implemented:
  - ID_MMFR4_EL1 and ID_MMFR5_EL1 are trapped to EL2.
  - ID_AA64MMFR2_EL1 and ID_ISAR6_EL1 are trapped to EL2.
  - ID_DFR1_EL1 is trapped to EL2.
  - ID_AA64ZFR0_EL1 is trapped to EL2.
  - ID_AA64SMFR0_EL1 is trapped to EL2.
  - ID_AA64ISAR2_EL1 is trapped to EL2.
  - This field traps all MRS accesses to registers in the following range that are not already mentioned in this field description: Op0 == 3, op1 == 0, CRn == c0, CRm == {c1-c7}, op2 == {0-7}.
- If FEAT_FGT is not implemented:
  - ID_MMFR4_EL1 and ID_MMFR5_EL1 are trapped to EL2, unless implemented as **RAZ**, when it is **IMPLEMENTATION DEFINED** whether accesses to ID_MMFR4_EL1 or ID_MMFR5_EL1 are trapped to EL2.
  - ID_AA64MMFR2_EL1 and ID_ISAR6_EL1 are trapped to EL2, unless implemented as **RAZ**, when it is **IMPLEMENTATION DEFINED** whether accesses to ID_AA64MMFR2_EL1 or ID_ISAR6_EL1 are trapped to EL2.
  - ID_DFR1_EL1 is trapped to EL2, unless implemented as **RAZ**, when it is **IMPLEMENTATION DEFINED** whether accesses to ID_DFR1_EL1 are trapped to EL2.
  - ID_AA64ZFR0_EL1 is trapped to EL2, unless implemented as **RAZ** then it is **IMPLEMENTATION DEFINED** whether accesses to ID_AA64ZFR0_EL1 are trapped to EL2.
  - ID_AA64SMFR0_EL1 is trapped to EL2, unless implemented as **RAZ** then it is **IMPLEMENTATION DEFINED** whether accesses to ID_AA64SMFR0_EL1 are trapped to EL2.
  - ID_AA64ISAR2_EL1 is trapped to EL2, unless implemented as **RAZ** then it is **IMPLEMENTATION DEFINED** whether accesses to ID_AA64ISAR2_EL1 are trapped to EL2.
  - Otherwise, it is **IMPLEMENTATION DEFINED** whether this bit traps MRS accesses to registers in the following range that are not already mentioned in this field description: Op0 == 3, op1 == 0, CRn == c0, CRm == {c1-c7}, op2 == {0-7}.

is corrected to read:

Trap ID group 3. Traps EL1 reads of group 3 ID registers to EL2, when EL2 is enabled in the current Security state, as follows:

In AArch64 state:

Reads of the following registers are trapped to EL2:

- ID_PFR0_EL1, ID_PFR1_EL1, ID_DFR0_EL1, ID_AFR0_EL1, ID_MMFR0_EL1, ID_MMFR1_EL1, ID_MMFR2_EL1, ID_MMFR3_EL1, ID_ISAR0_EL1, ID_ISAR1_EL1, ID_ISAR2_EL1, ID_ISAR3_EL1, ID_ISAR4_EL1, ID_ISAR5_EL1, MVFR0_EL1, MVFR1_EL1, MVFR2_EL1.
- ID_AA64PFR0_EL1, ID_AA64PFR1_EL1, ID_AA64DFR0_EL1, ID_AA64DFR1_EL1, ID_AA64ISAR0_EL1, ID_AA64ISAR1_EL1, ID_AA64MMFR0_EL1, ID_AA64MMFR1_EL1, ID_AA64AFR0_EL1, ID_AA64AFR1_EL1.

If FEAT_FGT is implemented, reads of the following registers are trapped to EL2. If FEAT_FGT is not implemented, reads of the following registers are trapped to EL2, unless the registers are implemented as **RAZ**, when it it **IMPLEMENTATION DEFINED** whether reads are trapped to EL2.

- ID_PFR2_EL1, ID_MMFR4_EL1 and ID_MMFR5_EL1.
- ID_AA64MMFR3_EL1.
- ID_AA64MMFR4_EL1.
- ID_AA64PFR2_EL1.
- ID_AA64MMFR2_EL1 and ID_ISAR6_EL1.
- ID_DFR1_EL1.
- ID_AA64ZFR0_EL1.
- ID_AA64SMFR0_EL1.
- ID_AA64ISAR2_EL1.

If FEAT_FGT is implemented, reads of registers in the following range that are not already mentioned in this field description: op0 == 3, op1 == 0, CRn == 0, CRm == {2-7}, op2 == {0-7} are trapped to EL2. If FEAT_FGT is not implemented, it is **IMPLEMENTATION DEFINED** whether reads of these registers in the range are trapped to EL2. Trapped registers are reported using EC syndrome value `0x18`.

In section D19.2.92 (ID_PFR2_EL1, AArch32 Processor Feature Register 2), the accessibility pseudocode that reads:

```
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TID3 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        X[t, 64] = ID_PFR2_EL1;
```

is replaced with:

```
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && (IsFeatureImplemented(FEAT_FGT) || !IsZero(ID_PFR2_EL1)
 || boolean IMPLEMENTATION_DEFINED "ID_PFR2_EL1 trapped by HCR_EL2.TID3") &&
 HCR_EL2.TID3 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        X[t, 64] = ID_PFR2_EL1;
```

2.78 R20697

In section D19.2.40 (FAR_EL1, Fault Address Register (EL1)), in the description of 'Bits [63:0]', the following text is added:

> If a memory fault that sets FAR_EL1 is generated from a STZGM instruction, the address held in FAR_EL1 is **IMPLEMENTATION DEFINED** as one of the following:
>
> - The lowest address that gave rise to the fault.
> - The address specified in the register argument.

The same change is made in the following sections:

- D19.2.41 (FAR_EL2, Fault Address Register (EL2)).
- D19.2.42 (FAR_EL3, Fault Address Register (EL3)).

In section D2.10.5 (Determining the memory location that caused a Watchpoint exception), subsection 'Address recorded for Watchpoint exceptions generated by other instructions', the following text:

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

is changed to read:

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

2.79 C20702

In section H9.2.24 (EDDFR, External Debug Feature Register), subsection 'Accessing the EDDFR', the text and tables that read:

> EDDFR[31:0] can be accessed through the external debug interface:

Component	Offset	Instance	Range
Debug	0xD28	EDDFR	31:0

> This interface is accessible as follows:
>
> - When IsCorePowered() and !DoubleLockStatus() accesses to EDDFR[31:0] are RO.
> - Otherwise accesses to EDDFR[31:0] are IMPDEF.
>
> EDDFR[63:32] can be accessed through the external debug interface:

Component	Offset	Instance	Range
Debug	0xD2C	EDDFR	63:32

> This interface is accessible as follows:
>
> - When IsCorePowered() and !DoubleLockStatus() accesses to EDDFR[63:32] are RO.
> - Otherwise accesses to EDDFR[63:32] are IMPDEF.

are corrected to read:

> EDDFR can be accessed through the external debug interface:

Component	Offset	Instance	Range
Debug	0xD28	EDDFR	

> This interface is accessible as follows:
>
> - When IsCorePowered() and !DoubleLockStatus() accesses to EDDFR are RO.
> - Otherwise accesses to EDDFR are IMPDEF.

Equivalent changes are made in the following sections:

- H9.2.35 (EDPFR, External Debug Processor Feature Register).
- H9.2.46 (EDWAR, External Debug Watchpoint Address Register).

D21046 further updates the accessors of EDDFR and EDPFR.

2.80 D20711

In section D10.5 (PE access to Allocation Tags), the text which reads:

> Instructions that store Allocation Tags to memory locations marked as Device memory result in a **CONSTRAINED UNPREDICTABLE** choice between:
>
> - Storing the data, if any, to the specified locations.
> - Generating an Alignment Fault, which is prioritized in the same way as other alignment faults that are determined by the memory type.

is changed to read:

> An STZGM instruction to any type of Device memory is **CONSTRAINED UNPREDICTABLE** between:
>
> - Zeroing the data at the specified locations and leaving any Allocation Tags unchanged.
> - Generating an Alignment Fault determined by the memory type.

2.81 D20728

In section D19.2.37 (ESR_EL1, Exception Syndrome Register (EL1)), in the subsections 'ISS encoding for an exception from a Data Abort' and 'ISS encoding for an exception from a Watchpoint exception', the VNCR field is deleted.

In section D19.2.38 (ESR_EL2, Exception Syndrome Register (EL2)), subsection 'ISS encoding for an exception from a Data Abort', the VNCR values that read:

VNCR	Meaning
0b0	The watchpoint was not generated by the use of VNCR_EL2 by EL1 code.
0b1	The watchpoint was generated by the use of VNCR_EL2 by EL1 code.

are updated to read:

VNCR	Meaning
0b0	The fault was not generated by the use of VNCR_EL2 by EL1 code.
0b1	The fault was generated by the use of VNCR_EL2 by EL1 code.

The same correction is made in section D19.2.39 (ESR_EL3, Exception Syndrome Register (EL3)), in the subsections 'ISS encoding for a Granule Protection Check exception' and 'ISS encoding for an exception from a Data Abort'.

2.82 D20731

In section D15.2.7 (Operation Type packet), subsection 'Operation Type packet payload (Other)', the 'SUBCLASS, byte<0>' field description that reads:

> Second-level instruction class. Defines the type of instruction. The defined values of this field are:
>
> 0b0000000x Other operation.
>
> 0b0xxx1xx0 SVE operation. If FEAT_SVE is implemented, and if FEAT_SPE is implemented, bits [6:4:2:1] are further defined as the EVL, PRED, and FP fields. Otherwise this value is reserved.

is corrected to read:

> Second-level instruction class. Defines the type of instruction. The defined values of this field are:
>
> 0b0000000x Other operation.
>
> 0b0xxx1xx0 SVE operation. If FEAT_SVE is implemented, and if FEAT_SPE is implemented, bits [6:4,2,1] are further defined as the EVL, PRED, and FP fields. Otherwise this value is reserved.

Within the same section, the field description 'FP, byte 0 bits [6:4], when SVE operation' that reads:

> Floating-point operation. The defined values of this bit are:

> 0 Integer
>
> 1 Floating-point

is renamed to 'FP, byte 0 bit [1], when SVE operation', and updated to read:

> Floating-point operation. The defined values of this bit are:
>
> 0 Not floating-point
>
> 1 Floating-point
>
> Where a floating-point instruction is any instruction which is counted by the FP_SVE_SPEC event.

Additionally, in section D14.6.5 (Additional information for each profiled Scalable Vector Extension operation), the text that reads:

> For a Sampled SVE operation, the Operation Type packet.EVL field records an upper bound on the Effective vector length. The value recorded in the Operation Type packet.EVL field is the Effective vector length rounded up to a power-of-two value.

is updated to read:

> For a Sampled SVE operation:
>
> - Operation Type packet.EVL field records an upper bound on the Effective vector length. The value recorded in the Operation Type packet.EVL field is the Effective vector length rounded up to a power of-two value.
> - Operation Type packet.FP is set to 1 if the instruction would be counted by the SVE_FP_SPEC event.

2.83 D20738

In section J1.1.4 (aarch64/instrs), in the function AArch64.AT(), access to HCR_EL2 is qualified with checks for EL2Enabled().

The code in AArch64.AT() that reads:

```
AArch64.AT(bits(64) address, TranslationStage stage_in, bits(2) el_in, ATAccess
 ataccess)
    ...
     // For stage 1 translation, when HCR_EL2.{E2H, TGE} is {1,1} and requested EL
 is EL1,
     // the EL2&0 translation regime is used.
     if HCR_EL2.<E2H, TGE> == '11' && el == EL1 && stage == TranslationStage_1 then
        el = EL2;
     if HaveEL(EL3) && stage == TranslationStage_12 && !EL2Enabled() then
      ...
```

is updated to read:

```
AArch64.AT(bits(64) address, TranslationStage stage_in, bits(2) el_in, ATAccess
ataccess)
    ...
    // For stage 1 translation, when HCR_EL2.{E2H, TGE} is {1,1} and requested EL
is EL1,
    // the EL2&0 translation regime is used.
    if EL2Enabled() && HCR_EL2.<E2H, TGE> == '11' && el == EL1 && stage ==
TranslationStage_1 then
        el = EL2;
    if HaveEL(EL3) && stage == TranslationStage_12 && !EL2Enabled() then
    ...
```

Equivalent changes are made in the following locations:

- Section J1.1.3 (aarch64/functions), in the function MaybeZeroSVEUppers().
- Section J1.2.4 (aarch32/translation), in the following functions:
  - AArch32.S1Enabled().
  - AArch32.S1DisabledOutput().
- Section J1.3.3 (shared/functions), in the function IllegalExceptionReturn().

2.84 C20749

In section D10.2 (Allocation Tags), the following text is added:

> All memory accesses to any Allocation tag are single-copy atomic.

In section D10.3 (Tag checking), the text that reads:

> A memory access that is a read or write can be either Tag Checked or Tag Unchecked.

is changed to read:

> A single-copy atomic memory access that is a read or write can be either Tag Checked or Tag Unchecked.

Also within this section, the text that reads:

> The read of an Allocation Tag due to a Tag Check operation, and the dependent data access, are not required to form an atomic operation.

is changed to read:

> Tag Check operation performs a single-copy atomic read of an Allocation tag.
>
> Note: Any MMU Faults due to the read of an Allocation tag for a Tag checked memory access are not ordered with respect to any MMU faults due to that memory access.

In section D10.5 (PE access to Allocation tags), the following text is added:

> An instruction which directly reads or writes Allocation tags performs a separate single-copy atomic access for each Allocation tag.
>
> Note: For an instruction which directly accesses both Allocation tags and data, any MMU Faults due to the separate accesses are not ordered.

2.85 C20759

In section D18.3.1 (Instructions for accessing non-debug System registers), the following clarification is added in the bullet list of the Note:

> - All unused encodings in the range Op0 == 3, op1 == 0, CRn == 0, CRm == {2-7}, op2 == {0-7} are defined to be accessible as Reserved, **RAZ** to ensure correct behavior if the encodings are used for ID registers in future.

2.86 D20760

In the following sections:

- D19.2.68 (ID_AA64MMFR4_EL1, AArch64 Memory Model Feature Register 4).
- D19.2.88 (ID_MMFR4_EL1, AArch32 Memory Model Feature Register 4).
- D19.2.89 (ID_MMFR5_EL1, AArch32 Memory Model Feature Register 5).
- D19.2.92 (ID_PFR2_EL1, AArch32 Processor Feature Register 2).

The following Note is added under 'Configurations':

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

The same change is made in section D19.2.71 (ID_PFR2_EL1, AArch64 Processor Feature Register 2), where in addition, the following text in the 'Purpose' subsection is removed:

> Reserved for future expansion of information about implemented PE features in AArch64 state.

2.87 D20791

In section C5.2.4 (DIT, Data Independent Timing), the list of instructions that reads:

> The data processing instructions affected by this bit are:
> - All cryptographic instructions. These instructions are:
>   - AESD, AESE, ...

- A subset of those instructions which use the general-purpose register file. These instructions are:
  - ADC, ADCS, ...
- A subset of those instructions which use the SIMD&FP register file. These instructions are:
  - ABS, ADD, ...

is updated to read:

> The Operational Information section of a data processing instruction description indicates whether or not that instruction is affected by this bit.

Similar changes are made in section G8.2.33 (CPSR, Current Program Status Register), in the description of DIT, bit [21], where the lists of individual instructions mnemonics are removed.

The text in section B1.3.6 (About PSTATE.DIT) that reads:

- The instructions listed in DIT are required to have;

is updated to read:

- The instructions affected by DIT are required to have:

The bullet point in the Note that follows, that reads:

- The Operational information section of an SVE or an SVE2 instruction description indicates whether or not that instruction honors the PSTATE.DIT control. If the Operational information section of an SVE instruction description does not mention PSTATE.DIT or if the section does not exist, then the instruction timing is not affected by PSTATE.DIT.

is moved to the top of the bullet list, and updated to read:

- The Operational information section of an instruction description indicates whether or not that instruction honors the PSTATE.DIT control. If the Operational information section of an instruction description does not mention PSTATE.DIT or if the section does not exist, then the instruction timing is not affected by PSTATE.DIT.

Similar changes are also made in section E1.2.5 (About the DIT bit).

2.88 D20794

In section J1.1.5 (aarch64/translation) in the function AArch64.S1DisabledOutput(), the code that reads:

```
(FaultRecord, AddressDescriptor) AArch64.S1DisabledOutput(...)
    ...
    if regime == Regime_EL10 && EL2Enabled() && walkparams.dc == '1' then
        ...
        if walkparams.dct == '1' then
            mematttrs.tags = MemTag_AllocationTagged;
        elsif walkparams.mtx == '1' then;
```

```
            memattrs.tags = MemTag_CanonicallyTagged;
            if walkparams.tbi == '0' then
                // For the purpose of the checks in this function, the MTE tag bits
 are ignored.
                va<59:56>     = Replicate(va<55>, 4);
        else
            memattrs.tags     = MemTag_Untagged;
     elsif accdesc.acctype == AccessType_IFETCH then
        ...
        if walkparams.mtx == '1' then
            memattrs.tags     = MemTag_CanonicallyTagged;
        else
            memattrs.tags     = MemTag_Untagged;
    else
        ...
        if walkparams.mtx == '1' then
            memattrs.tags = MemTag_CanonicallyTagged;
            if walkparams.tbi == '0' then
                // For the purpose of the checks in this function, the MTE tag bits
 are ignored.
                if HasUnprivileged(regime) then
                    va<59:56> = Replicate(va<55>, 4);
                else
                    va<59:56> = '0000';
```

Is updated to read:

```
(FaultRecord, AddressDescriptor) AArch64.S1DisabledOutput(...)
    ...
    if regime == Regime_EL10 && EL2Enabled() && walkparams.dc == '1' then
        ...
        if walkparams.dct == '1' then
            memattrs.tags     = MemTag_AllocationTagged;
         elsif walkparams.mtx == '1' then
            memattrs.tags     = MemTag_CanonicallyTagged;
         else
            memattrs.tags     = MemTag_Untagged;
    elsif accdesc.acctype == AccessType_IFETCH then
        ...
    else
        ...
        if walkparams.mtx == '1' then
            memattrs.tags = MemTag_CanonicallyTagged;

    if walkparams.mtx == '1' && walkparams.tbi == '0' && accdesc.acctype !=
 AccessType_IFETCH then
        // For the purpose of the checks in this function, the MTE tag bits are
 ignored.
        va<59:56> = if HasUnprivileged(regime) then Replicate(va<55>, 4) else
 '0000';
```

2.89 R20805

In sections D19.8.5 (BRBINF<n>_EL1, Branch Record Buffer Information Register <n>, n = 0 - 31) and D19.8.6 (BRBINFINJ_EL1, Branch Record Buffer Information Injection Register), the following value is added to 'TYPE, bits [13:8]':

`0b110000` **IMPLEMENTATION DEFINED** exception to EL3.

2.90 R20809

In section D19.2.104 (MPIDR_EL1, Multiprocessor Affinity Register) in the subsection 'Purpose', the text that reads:

> In a multiprocessor system, provides an additional PE identification mechanism for scheduling purposes.

is changed to read:

> In a multiprocessor system, provides an additional PE identification mechanism.

Also in the description of the 'Aff0, bits [7:0]' field, the text that reads:

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

is changed to read:

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

2.91 D20816

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

> Bits[63:0]
>
> Reserved, **RES0**.

is updated to read:

> NS, bit [63]
>
> Together with the NSE field, this field specifies the target physical address space.

NSE	NS	Meaning
0b0	0b0	Reserved.
0b0	0b1	Reserved.
0b1	0b0	Reserved.
0b1	0b1	Realm.

> If {NSE, NS} != {0b1, 0b1}, then no cache entries are required to be cleaned or invalidated

NSE, bit [62]

Together with the NS field, this field specifies the target physical address space.

For a description of the values derived by evaluating NS and NSE together, see DC CIGDPAE.NS.

Bits [61:52]

Reserved, **RES0**.

PA, bits [51:0]

Physical address to use. No alignment restrictions apply to this PA.

Similar changes are made in section C5.3.16 (DC CIPAE, Data or unified Cache line Clean and Invalidate by PA to PoE).

2.92 D20829

In J1.1.1 (aarch64/debug) the functions BRBCycleCountingEnabled() and BRBEMispredictAllowed() are updated to check if EL2 is present.

The code that reads:

```
boolean BRBCycleCountingEnabled()
    if EL2Enabled() && BRBCR_EL2.CC == '0' then return FALSE;
    ...
```

Is updated to read:

```
boolean BRBCycleCountingEnabled()
    if HaveEL(EL2) && BRBCR_EL2.CC == '0' then return FALSE;
    ...
```

The code that reads as:

```
boolean BRBEMispredictAllowed()
    if EL2Enabled() && BRBCR_EL2.MPRED == '0' then return FALSE;
    ...
```

Is updated to read:

```
boolean BRBEMispredictAllowed()
    if HaveEL(EL2) && BRBCR_EL2.MPRED == '0' then return FALSE;
    ...
```

2.93 R20840

In section D15.2.1 (Address packet), in the subsection 'Address packet header', the value description in the field 'INDEX, byte 0 bits [1:0], byte 1 bits [2:0], when Extended format, INDEX, byte 0 bits [2:0], when Short format' that reads:

> `0b00011` Data access physical address:
>
> - Not included if disabled by CollectPhysicalAddress, or if the access generates an abort other than a Permission or Access Flag fault.
> - It is **IMPLEMENTATION DEFINED** and might be **UNPREDICTABLE** whether this is included for accesses that generate Permission or Access Flag faults.
> - Included for all other load, store and atomic operations.

is changed to read:

> `0b00011` Data access physical address:
>
> - Not included if disabled by CollectPhysicalAddress or the data virtual address for the same operation is not output.
> - Included for all other load, store and atomic operations.

In section D14.6.3 (Additional information for each profiled memory access operation), the text that reads:

> The sampled data physical address packet is not output if any of the following are true:
>
> - The sampled operation operates on a virtual address and 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.
> - Sampling of physical addresses is prohibited by System register controls.

is changed to read:

> The sampled data physical address packet is not output if any of the following are true:
>
> - The sampled operation operates on a virtual address and any of the following are true:
>   - The PE does not translate the address, for example because it does not perform the access.
>   - The sampled data virtual address packet is not output.
> - Sampling of physical addresses is prohibited by System register controls.

2.94 D20853

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

```
UpdateEDSCRFields()

    if !Halted() then

    ...

    else
       EDSCR.EL = PSTATE.EL;
       ss = CurrentSecurityState();
       ...
```

is updated to read

```
UpdateEDSCRFields()

    if !Halted() then

    ...

    else
       EDSCR.EL = PSTATE.EL;
        // SError Pending.
        if EL2Enabled() && HCR_EL2.<AMO,TGE> == '10'  && PSTATE.EL IN {EL0,EL1} then
            EDSCR.A = if IsVirtualSErrorPending() then '1' else '0';
        else
            EDSCR.A = if IsPhysicalSErrorPending() then '1' else '0';

       ss = CurrentSecurityState();
       ...
```

2.95 R20865

In section D16.1.1 (Branch records), statement I_{ZCHRF} that reads:

> When an exception, exception return instruction, or Instruction Synchronization Barrier instruction causes a Context synchronization event which synchronizes an update to one or more System registers which are indirectly read when generating a Branch record, the synchronization of those register updates occurs before the registers are indirectly read. Such order is generally consistent with indirect reads of System registers performed by events which cause a Context synchronization event.

is updated to read:

> When an Instruction Synchronization Barrier instruction causes a Context synchronization event which synchronizes an update to one or more System registers which are indirectly read when generating a Branch record, the synchronization of those register updates occurs before the registers are indirectly read. Such order is generally consistent with indirect reads of System registers performed by events which cause a Context synchronization event.

Within the same section, the following rule is added:

> When an exception or exception return instruction causes a Context synchronization event which synchronizes an update to one or more System registers which are used to determine whether the source of the branch record is from a BRBE Prohibited region, indirect reads of those System registers are permitted to occur before the Context synchronization event. Specifically, the registers indirectly read by BranchRecordAllowed().
>
> All other indirect reads of System registers used for creation of a Branch record occur after the Context synchronization event, including those for determining whether the Branch records is filtered or not, and those used for determining whether Branch recording is prohibited at the target.

2.96 C20871

In section D12.11.3 (Common event numbers), subsection 'Common microarchitectural events', the text in the '`0x0001`, L1I_CACHE_REFILL, Level 1 instruction cache refill' event description that reads:

> If the cache is shared, the counter counts only events Attributable to the PE counting the event, and, if the Effective value of PMEVTYPER<n>_EL0.MT for the counter is `0b1`, other PEs in the multithreaded implementation. If the cache is not shared, all events are counted.

is updated to read:

> If the cache is shared and the Effective value of PMEVTYPER<n>_EL0.MT for the counter is `0b0`, then the counter counts only events Attributable to the PE counting the event. For a multithreaded processor implementation, if the cache is shared by PEs other than the PEs in the multithreaded processor and the Effective value of PMEVTYPER<n>_EL0.MT for the counter is `0b1`, the counter counts only events Attributable to PEs in the multithreaded processor. In all other cases, it is **IMPLEMENTATION DEFINED** whether only events Attributable to the PE counting the event or all events are counted and might depend on the Effective value of PMEVTYPER<n>_EL1.MT.

Equivalent statements concerning shared resources, in other PMU event descriptions throughout the section, are similarly updated.

2.97 D20907

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

```
    fault = NoFault(accdesc);
    fault.statuscode = Fault_TagCheck;

    case tcf of
        ...
```

Is updated to read as:

```
    FaultRecord fault = NoFault();
    fault.access      = accdesc;
    fault.write       = accdesc.write;
    fault.statuscode  = Fault_TagCheck;

    case tcf of
        ...
```

In the same section in MemAtomic(), the code that reads as:

```
    // For Store-only Tag checking, the tag check is performed on the store.
    if (HaveMTE2Ext() && accdesc.tagchecked &&
        (!HaveMTEStoreOnlyExt() || !StoreOnlyTagCheckingEnabled(accdesc.el))) then
        ...
        if !AArch64.CheckTag(memaddrdesc, accdesc, ptag) then
            AArch64.TagCheckFault(address, accdesc);

    ...

    if HaveMTEStoreOnlyExt() && StoreOnlyTagCheckingEnabled(accdesc.el) then
        ...
            if !AArch64.CheckTag(memaddrdesc, accdesc, ptag) then
                accdesc.read = FALSE; // Tag Check Fault on a write.
                AArch64.TagCheckFault(address, accdesc);
```

Is updated to read as:

```
    // For Store-only Tag checking, the tag check is performed on the store.
    if (HaveMTE2Ext() && accdesc.tagchecked &&
        (!HaveMTEStoreOnlyExt() || !StoreOnlyTagCheckingEnabled(accdesc.el))) then
        ...
        if !AArch64.CheckTag(memaddrdesc, accdesc, ptag) then
            accdesc.write = FALSE;                // Tag Check Fault on a read
            AArch64.TagCheckFault(address, accdesc);

    ...

    if HaveMTEStoreOnlyExt() && StoreOnlyTagCheckingEnabled(accdesc.el) then
        ...
            if !AArch64.CheckTag(memaddrdesc, accdesc, ptag) then
                AArch64.TagCheckFault(address, accdesc);
```

An equivalent change is made in MemAtomicRCW() in the same section.

2.98 C20913

In section B2.3.11 (Restrictions on the effects of speculation), in the subsection ‘Restrictions on the effects of speculation from Armv8.5’, the text that reads:

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

is clarified to read:

> If FEAT_CSV2_1p1 is implemented, branch or data values trained from one instruction address cannot exploitatively control, or predictively leak to, the speculative execution of code from a different address.

2.99 D20918

In section D19.2.48 (HCR_EL2, Hypervisor Configuration Register), field 'ATA, bit [56]', the text that reads:

> `0b0` Access to Allocation Tags is prevented at EL1 and EL0. Accesses at EL1 to GCR_EL1, RGSR_EL1, TFSR_EL1, or TFSRE0_EL1 that are not **UNDEFINED** are trapped to EL2. Accesses at EL1 using MRS or MSR with the register name TFSR_EL2 that are not **UNDEFINED** are trapped to EL3. Memory accesses at EL1 and EL0 are not subject to a Tag Check operation.

is corrected to read:

> `0b0` Access to Allocation Tags is prevented at EL1 and EL0. Accesses at EL1 to GCR_EL1, RGSR_EL1, TFSR_EL1, or TFSRE0_EL1 that are not **UNDEFINED** are trapped to EL2. Accesses at EL1 using MRS or MSR with the register name TFSR_EL2 that are not **UNDEFINED** are trapped to EL2. Memory accesses at EL1 and EL0 are not subject to a Tag Check operation.

2.100 D20921

In section D19.2.125 SCTLR_EL2, System Control Register (EL2), field EPAN, bit [57], the heading that reads:

> When FEAT_PAN3 is implemented, HCR_EL2.E2H == 1 and HCR_EL2.TGE==1:

is changed to read

> When FEAT_PAN3 is implemented and HCR_EL2.E2H ==1:

the following text is added after the value table:

> Note: The value of HCR_EL2.TGE does not change the behavior of this field.

and the following heading and text is removed:

> When FEAT_PAN3 is implemented, HCR_EL2.E2H == 1 and HCR_EL2.TGE == 0: IGNORED.

2.101 C20933

In section D1.5.2 (Reset types), the following rule is added:

> Writing 1 to RMR_ELx.RR is only a request for a Warm reset. The reset is not guaranteed to occur unless the following code sequence is executed:

```
; In addition, interrupts and debug requests for this PE should be disabled
; in the system before running this sequence to ensure the WFI suspends execution

    MOV Wy, #3          ; for AArch64, #2 for AArch32; y is any register
    DSB                 ; ensure all stores etc are complete
    MSR RMR_ELx, Wy     ; request the reset
    ISB                 ; synchronise change to the RMR
Loop
    WFI                 ; enter a quiescent state
    B Loop
```

2.102 D20934

In section D10.7 (PE handling of Tag Check Fault), within the bullet list that reads:

> It is **CONSTRAINED UNPREDICTABLE** whether the FFR element associated with the read of an Active element in an SVE Non-fault load, or an Active element which is not the First active element in an SVE First-fault load, R2, to location X, is set to FALSE if all of the following are true:
>
> - Tag Check Faults are configured as asynchronous for both reads and writes.
> - A read or write RW1 to location Y causes a Tag Check Fault.

The following bullet is added:

> - A read to location X causes a Tag Check Fault.

2.103 D20940

In section D8.12.3 (MMU faults generated by cache maintenance operations), the rules that read:

> R_{FTXTG}
>
> If the Effective value of SCTLR_EL1.CMOW is 1, then when executing an IC IVAU, DC CVAC, DC CIGVAC, or DC CIGDVAC instruction at EL0 that has stage 1 read permission, but does not have stage 1 write permission, a stage 1 Permission fault is generated.
>
> R_{BBLTJ}
>
> If the Effective value of HCRX_EL2.CMOW is 1, then when executing an IC IVAU, DC CVAC, DC CIGVAC, or DC CIGDVAC instruction at EL1 or EL0 that has stage 2 read permission, but does have stage 2 write permission, a stage 2 Permission fault is generated.

> R_{HGLYG}
>
> If an IC IVAU, DC CVAC, DC CIGVAC, or DC CIGDVAC instruction is implemented as a **NOP**, then it is **IMPLEMENTATION DEFINED** whether the instruction generates a stage 1 or stage 2 Permission fault when it does not have read and write permission.

are updated to read:

> R_{FTXTG}
>
> If the Effective value of SCTLR_EL1.CMOW is 1, then when executing an IC IVAU, DC CIVAC, DC CIGVAC, or DC CIGDVAC instruction at EL0 that has stage 1 read permission, but does not have stage 1 write permission, a stage 1 Permission fault is generated.
>
> R_{BBLTJ}
>
> If the Effective value of HCRX_EL2.CMOW is 1, then when executing an IC IVAU, DC CIVAC, DC CIGVAC, or DC CIGDVAC instruction at EL1 or EL0 that has stage 2 read permission, but does have stage 2 write permission, a stage 2 Permission fault is generated.
>
> R_{HGLYG}
>
> If an IC IVAU, DC CIVAC, DC CIGVAC, or DC CIGDVAC instruction is implemented as a **NOP**, then it is **IMPLEMENTATION DEFINED** whether the instruction generates a stage 1 or stage 2 Permission fault when it does not have read and write permission.

2.104 C20943

In section D19.2.114, (RNDR, Random Number), subsection 'Purpose', the text that reads:

> Random Number. Returns a 64-bit random number which is reseeded from the True Random Number source at an **IMPLEMENTATION DEFINED** rate.

is corrected to read:

> Random Number. Returns a 64-bit random number from an approved Random Bit Generator, where the Deterministic Random Bit Generator (DRBG) within the Random Bit Generator is reseeded from an approved entropy source at an **IMPLEMENTATION DEFINED** rate.

The equivalent changes are made in the following section:

- Section D19.2.114, (RNDR, Random Number), subsection 'RNDR, bits [63:0]'.

In section D19.2.115, (RNDRRS, Reseeded Random Number), the Title text that reads:

> D19.2.115 RNDRRS, Reseeded Random Number

is corrected to read:

D19.2.115 RNDRRS, Random Number Full Entropy

In the same section, subsection 'Purpose', the text that reads:

> Reseeded Random Number. Returns a 64-bit random number which is reseeded from the True Random Number source immediately before the read of the random number.

is corrected to read:

> Random Number with fresh full entropy. Returns a 64-bit random number from an approved Random Bit Generator, using either a Non-deterministic Random Bit Generator (NRBG) or one where the DBRG is reseeded (where possible) from an approved entropy source before the return of the random number.

The equivalent changes are made in the following section:

- Section D19.2.115, (RNDRRS, Reseeded Random Number), subsection 'RNDRRS, bits [63:0]'.

In section K14.1, (Properties of the generated random number), the text that reads:

> When FEAT_RNG is implemented, reads of the RNDR and RNDRRS registers return a 64-bit random number. The random numbers must meet the properties and conform to the standards that are detailed in this section.
>
> The output random number is from a Deterministic Random Bit Generator (DRBG), which is seeded from a True Random Number Generator (TRNG).
>
> The TRNG provides entropy in the form of random numbers, from the sampled output of an unpredictable physical process.
>
> The TRNG should conform to:
>
> - The NIST SP800-90B standard.
> - The NIST SP800-22 standard.
> - The FIPS 140-2 standard.
> - The BSI AIS-31 standard.
>
> The DRBG produces random numbers from a cryptographically secure algorithm.
>
> The DRBG is seeded from the TRNG.
>
> The DRBG algorithm should conform to the NIST SP800-90A Rev 1 standard.
>
> The DRBG is reseeded after an **IMPLEMENTATION DEFINED** number of random numbers has been generated and read using the RNDR register.
>
> The DRBG is reseeded immediately before the random number is generated and read using the RNDRRS register.
>
> The entire random number generation should conform to the NIST SP800-90C standard.

Note:

- Since a TRNG can only generate random bits at a limited rate, the random number bits are commonly collected in an “entropy pool” until needed. An implementation should ensure that lower privileged software cannot impact the performance of higher privileged software by entirely draining this “entropy pool”. The refill time cost of the “entropy pool” should be paid for by the persistent caller.
- When FEAT_RNG_TRAP is implemented, reads of the RNDR and RNDRRS registers may be trapped to EL3. For more information about this trapping behavior, see control fields ID_AA64PFR1_EL1.RNDR_trap and SCR_EL3.TRNDR

is changed to read:

When FEAT_RNG is implemented, reads of the RNDR and RNDRRS registers return 64-bit random numbers. The RNDR and RNDRRS implementation should conform to approved standards that are appropriate for the market requirements.

For example, the NIST SP 800-90 series of documents:

- SP 800-90A Recommendation for Random Number Generation Using Deterministic Random Bit Generators
- SP 800-90B Recommendation for the Entropy Sources Used for Random Bit Generation
- SP 800-90C Recommendation for Random Bit Generator (RBG) Constructions

Note:

- Since an entropy source can only generate random bits at a limited rate, the random number bits are commonly collected in an “entropy pool” until needed. An implementation should ensure that lower privileged software cannot impact the performance of higher privileged software by entirely draining this “entropy pool”. The refill time cost of the “entropy pool” should be paid for by the persistent caller.
- When FEAT_RNG_TRAP is implemented, reads of the RNDR and RNDRRS registers may be trapped to EL3. For more information about this trapping behavior, see control fields ID_AA64PFR1_EL1.RNDR_trap and SCR_EL3.TRNDR

2.105 C20950

In section D19.2.37 (ESR_EL1, Exception Syndrome Register (EL1)), in the subsection ‘ISS encoding for an exception from a Watchpoint exception’, the text in the ‘WPTV, bit [17]’ field that reads:

When a Watchpoint exception is triggered by a watchpoint match:

- If the PE sets any of FnV, FnP, or WPF to 1, then the PE sets WPTV to 1.
- If the PE sets all of FnV, FnP, and WPF to 0, then the PE sets WPTV to an implementation defined value, 0 or 1.

is changed to read:

> When a Watchpoint exception is triggered by a watchpoint match:
> - If the PE sets any of FnV, FnP, or WPF to 1, then the PE sets WPTV to 1.
> - Otherwise, the PE sets WPTV to an implementation defined value, 0 or 1.

The equivalent change is made in section D19.2.38 (ESR_EL2, Exception Syndrome Register (EL2)).

2.106 D20957

In section D19.2.125 (SCTLR_EL2, System Control Register (EL2)), in the field 'nTWE, bit [18]', the text that reads:

> Traps execution of WFI instructions at EL0 to EL2, from both Execution states.

is changed to read:

> Traps execution of WFE instructions at EL0 to EL2, from both Execution states.
>
> When FEAT_WFxT is implemented, this trap also applies to the WFET instruction.

A similar change is made to field 'nT**WI**, bit [16]', in the same register.

In section D19.2.39 (ESR_EL3, Exception Syndrome Register (EL3)), subsection 'ISS encoding for an exception from a WF* instruction', the text that reads:

> The following fields describe configuration settings for generating this exception:
> - SCTLR_EL1.{nTWE, nTWI}.
> - HCR_EL2.{TWE, TWI}.
> - SCR_EL3.{TWE, TWI}.

is changed to read:

> The following fields describe configuration settings for generating this exception:
> - SCTLR_EL1.{nTWE, nTWI}.
> - SCTLR_EL2.{nTWE, nTWI}.
> - HCR_EL2.{TWE, TWI}.
> - SCR_EL3.{TWE, TWI}.

The equivalent change is made in the equivalent subsection of sections D19.2.38 (ESR_EL2, Exception Syndrome Register (EL2)) and D19.2.37 (ESR_EL1, Exception Syndrome Register (EL1)).

2.107 C20977

In section D19.2.26 (CCSIDR_EL1, Current Cache Size ID Register), the text that reads:

> If CSSELR_EL1.{TnD, Level, InD} is programmed to a cache level that is not implemented, then on a read of the CCSIDR_EL1 the behavior is **CONSTRAINED UNPREDICTABLE**, and can be one of the following:
>
> - The CCSIDR_EL1 read is treated as **NOP**.
> - The CCSIDR_EL1 read is **UNDEFINED**.
> - The CCSIDR_EL1 read returns an **UNKNOWN** value.

is clarified to read:

> If CSSELR_EL1.{TnD, Level, InD} is programmed to a cache level that is not implemented, then on a read of the CCSIDR_EL1 the behavior is **CONSTRAINED UNPREDICTABLE**, and can be one of the following:
>
> - The CCSIDR_EL1 read is treated as **NOP**.
> - The CCSIDR_EL1 read is **UNDEFINED**. If FEAT_IDST is implemented, this is permitted to be reported with EC code `0x18`.
> - The CCSIDR_EL1 read returns an **UNKNOWN** value.

The equivalent change is also made in section D19.2.25 (CCSIDR2_EL1, Current Cache Size ID Register 2).

2.108 R20990

In section A2.8.1 (Architectural features added by Armv8.6), the text in the description of 'FEAT_MTPMU, Multi-threaded PMU Extensions' that reads:

> Multithreaded Armv8.6 implementations with FEAT_PMUv3 implemented must implement FEAT_MTPMU to enable any multithreaded event counting.

is updated to read:

> From Armv8.6, when FEAT_PMUv3 is implemented, multithreaded event counting is only supported in multithreaded implementations that also include FEAT_MTPMU.

2.109 D21000

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

```
...
    StopInstructionPrefetchAndEnableITR();
    EDSCR.STATUS = reason;
```

```
    UpdateEDSCRFields();
...
```

Is updated to read :

```
...
    StopInstructionPrefetchAndEnableITR();
    (EDSCR.STATUS,EDPRSR.HALTED) = (reason,'1');
    UpdateEDSCRFields();
...
```

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

```
...
    (EDSCR.STATUS,EDPRSR.SDR) = ('000010','1');        // Atomically signal
 restarted
    UpdateEDSCRFields();
..
```

Is updated to read :

```
...
    (EDSCR.STATUS,EDPRSR.SDR) = ('000010','1');        // Atomically signal
 restarted
    EDPRSR.HALTED = '0';
    UpdateEDSCRFields();
...
```

2.110 C21010

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

M[3:0]	Meaning
0b0000	EL0t.
0b0100	EL1t.
0b0101	EL1h.

is clarified to read:

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

In section C5.2.19 (SPSR_EL2, Saved Program Status Register (EL2)), the value table that reads:

M[3:0]	Meaning
0b0000	EL0t.
0b0100	EL1t.
0b0101	EL1h.
0b1000	EL2t.
0b1001	EL2h.

is clarified to read:

M[3:0]	Meaning
0b0000	EL0.
0b0100	EL1 with SP_EL0 (EL1t).
0b0101	EL1 with SP_EL1 (EL1h).
0b1000	EL2 with SP_EL0 (EL2t).
0b1001	EL2 with SP_EL2 (EL2h).

In section C5.2.20 (SPSR_EL3, Saved Program Status Register (EL3)), the value table that reads:

M[3:0]	Meaning
0b0000	EL0t.
0b0100	EL1t.
0b0101	EL1h.
0b1000	EL2t.
0b1001	EL2h.
0b1100	EL3t.
0b1101	EL3h.

is clarified to read:

M[3:0]	Meaning
0b0000	EL0.
0b0100	EL1 with SP_EL0 (EL1t).
0b0101	EL1 with SP_EL1 (EL1h).
0b1000	EL2 with SP_EL0 (EL2t).
0b1001	EL2 with SP_EL2 (EL2h).
0b1100	EL3 with SP_EL0 (EL3t).
0b1101	EL3 with SP_EL3 (EL3h).

A similar change is made to section D19.3.14 (DSPSR_EL0, Debug Saved Program Status Register), in the subsection 'When AArch64 is supported and entering or exiting Debug state from or to AArch64 state:', where the value table that reads:

M[3:0]	Meaning
0b0000	EL0t.

M[3:0]	Meaning
0b0100	EL1t.
0b0101	EL1h.
0b1000	EL2t.
0b1001	EL2h.
0b1100	EL3t.
0b1101	EL3h.

is clarified to read:

M[3:0]	Meaning
0b0000	EL0.
0b0100	EL1 with SP_EL0 (EL1t).
0b0101	EL1 with SP_EL1 (EL1h).
0b1000	EL2 with SP_EL0 (EL2t).
0b1001	EL2 with SP_EL2 (EL2h).
0b1100	EL3 with SP_EL0 (EL3t).
0b1101	EL3 with SP_EL3 (EL3h).

In section D1.2.2 (The stack pointer registers), subsection 'Stack pointer register selection', the information statement I_{QZDYD}, that reads:

> The following are the AArch64 stack pointer options:

Exception level (EL)	Stack pointer (SP) options
EL0	SP_EL0t
EL1	SP_EL1t, SP_EL1h
EL2	SP_EL2t, SP_EL2h
EL3	SP_EL3t, SP_EL3h

is corrected to read:

> The following are the AArch64 stack pointer options:

Exception level (EL)	Stack pointer (SP) options
EL0	SP_EL0
EL1	SP_EL1, SP_EL0
EL2	SP_EL2, SP_EL0
EL3	SP_EL3, SP_EL0

Within the same section, the information statement I_{VYNZY} that reads:

> The selected stack pointer can be indicated by a suffix to the Exception level:
>
> The t suffix, referring to thread, indicates use of the SP_EL0 stack pointer. The h suffix, referring to host, indicates use of the SP_ELx stack pointer.

is corrected to read:

> The selected stack pointer can be indicated by a suffix to the Exception level:
>
> The t suffix, referring to thread, indicates use of the SP_EL0 stack pointer. The h suffix, referring to handler, indicates use of the SP_ELx stack pointer.

2.111 D21044

In section D15.2.9 (Timestamp packet), subsection “Timestamp packet payload”, the text that reads:

> TS, bytes <7:0>
>
> Timestamp value when the operation was sampled. The value depends on the result of CollectTimeStamp():
>
> - If TimeStamp_Virtual, this is the virtual timestamp, CNTVCT_EL0.
> - If TimeStamp_Physical, this is the physical timestamp, CNTPCT_EL0.
> - If TimeStamp_OffsetPhysical, this is the offset physical timestamp, CNTPCT_EL0 - CNTPOFF_EL2.
> - If TimeStamp_None, the Timestamp packet is not included and an End packet must come at the end of the record.

is changed to read:

> TS, bytes <7:0>
>
> Timestamp value when the operation was sampled. The timestamp value is described in [section D14.6.8].

2.112 D21046

In section H9.2.25 (EDDFR1, External Debug Feature Register 1), subsection “Accessing the EDDFR1”, the text:

> This interface is accessible as follows:
>
> - When IsCorePowered() and !DoubleLockStatus() accesses to EDDFR1[31:0] are RO.
> - Otherwise accesses to EDDFR1[31:0] are IMPDEF.

is corrected to:

> This interface is accessible as follows:
>
> - When IsCorePowered() and !DoubleLockStatus() and (!IsZero(EDDFR1) or !OSLockStatus()), accesses to this register are RO.

- When FEAT_DoPD is implemented and !IsCorePowered(), accesses to this register generate an error response.
- Otherwise accesses to this register are IMPDEF.

Equivalent changes are made in the following:

- H9.2.24 (EDDFR, External Debug Feature Register)
- H9.2.35 (EDPFR, External Debug Processor Feature Register)
- H9.2.10 (EDAA32PFR, External Debug Auxiliary Processor Feature Register)

2.113 C21061

In section A1.5.6 (Rounding), in the subsection 'Round to Nearest with Ties to Away', the sub-bullet that reads:

- If the value before rounding has an absolute value that is not too large to represent in the output format, the result is calculated as follows:
  - If the two nearest floating-point numbers bracketing the value before rounding are equally near, the result is the larger number.

is clarified to read:

- If the value before rounding has an absolute value that is not too large to represent in the output format, the result is calculated as follows:
  - If the two nearest floating-point numbers bracketing the value before rounding are equally near, the result is the number with the largest absolute value.

The equivalent change is made in section E1.3.7 (Rounding), in the subsection 'Round to Nearest with Ties to Away'.

2.114 D21077

In section B2.3.11 (Restrictions on the effects of speculation), the following text:

- 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.

and the following text in section E2.3.4 (Restrictions on the effects of speculation):

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

are both updated to read:

- When data is loaded under speculation from a location that does not have a valid 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.

2.115 R21100

In section D19.5.15 (PMSELR_EL0, Performance Monitors Event Counter Selection Register), in the field 'SEL, bits [4:0]', the following text is added:

If FEAT_FGT is not implemented and this field is set to a value greater than or equal to the number of implemented counters, but not equal to 31, then direct reads of this field return an **UNKNOWN** value.

2.116 C21111

In section C6.2.116 (DSB), the following statements are removed:

A DSB instruction with the nXS qualifier is complete when the subset of these memory accesses with the XS attribute set to 0 are complete. It does not require that memory accesses with the XS attribute set to 1 are complete.

2.117 D21122

In section A3.1 (Armv9-A architecture extensions), the following statement in the definition of 'FEAT_RME, Realm Management Extension':

If the PC Sample-based Profiling Extension is implemented, then a PE that implements FEAT_RME must implement all of the following:

- FEAT_PCSRv8.
- FEAT_PCSRv8p2.

is corrected to read:

If the PC Sample-based Profiling Extension is implemented, then a PE that implements FEAT_RME must implement FEAT_PCSRv8p2.

2.118 D21140

The Glossary definition of the term “Speculative” that reads:

> - Reads or writes generated by load or store instructions for which the data being written or the address being accessed comes from a register that has not been architecturally resolved

is updated to read:

> - Operations generated by instructions for which any arguments come from a register that has not been architecturally resolved.

In section B2.3.11 (Memory barriers), subsection “Speculation Barrier”, the paragraph that reads:

> An SB instruction can complete when:
> - It is known that it is not speculative.
> - All the predicted data values generated by instructions appearing in program order before the SB instruction have their predicted values confirmed.

is clarified to read:

> An SB instruction can complete when:
> - It is known that is not speculative, or it is speculative only as a result of either:
>   - Speculating that an instruction that could generate an exception does not generate an exception.
>   - Speculating past the point in the Execution stream where a precise asynchronous exception is taken.
> - All the data values generated by instructions appearing in program order before the SB instruction are architecturally resolved, and so are not speculative.

In section C6.2.264 (SB), the description of the behavior of the SB instruction is replaced with a cross-reference to the subsection “Speculation Barrier” described above.

Equivalent changes are made for AArch32, in both:

- Section E2.3.5 (Memory barriers), subsection “Speculation Barrier (SB)”
- Section F5.1.175 (SB)

2.119 D20284

In section D19.8.7 (BRBSRC<n>_EL1, Branch Record Buffer Source Address Register <n>, n = 0 - 31), the text that reads:

> When an indirect write occurs with a value with ADDRESS bits [63:P] being other than all zeroes or all ones, an **UNKNOWN** value which is not all zeroes or all ones is written to bits [63:P]. P is

> defined as the virtual address size supported by the PE, as returned by VAMax(). The value in bits [P-1:0] are the value written.

is updated to read:

> When an indirect write occurs with a value with ADDRESS bits [63:P] being other than all zeroes or all ones, an **UNKNOWN** value which is not all zeroes or all ones is written to bits [63:P]. P is defined as:
>
> - 52 when FEAT_LVA is implemented.
> - 48, otherwise.
>
> The value in bits [P-1:0] is the value written.

Equivalent changes are made in the following sections:

- D19.8.8 (BRBSRCINJ_EL1, Branch Record Buffer Source Address Injection Register).
- D19.8.9 (BRBTGT<n>_EL1, Branch Record Buffer Target Address Register <n>, n = 0 - 31).
- D19.8.10 (BRBTGTINJ_EL1, Branch Record Buffer Target Address Injection Register).

In section D19.4.9 (TRCACVR<n>, Address Comparator Value Register <n>, n = 0 - 15), the text in the 'ADDRESS, bits [63:0]' description that reads:

> The result of writing a value other than all zeros or all ones to ADDRESS at bits[63:P] is an **UNKNOWN** value, where P is defined as the maximum virtual address size supported by the PE.

is updated to read:

> The result of writing a value other than all zeros or all ones to ADDRESS at bits[63:P] is an **UNKNOWN** value, where P is defined as:
>
> - 52 when FEAT_LVA is implemented.
> - 48, otherwise.

The same change is made in section H9.3.2 (TRCACVR<n>, Address Comparator Value Register <n>, n = 0 - 15).

In section D4.5.9 (Element Generation), subsection 'Exception element', I_{CMRCN} that reads:

> An invalid address is one where bits [63:P] are not all zeros or all ones, where P is defined as the maximum virtual address size supported by the PE.

is updated to read:

> An invalid address is one where bits [63:P] are not all zeroes or all ones, where P is defined as:
>
> - 52 when FEAT_LVA is implemented.
> - 48, otherwise.

The same change is made to the statement I_{KZXQW} within subsection 'Target Address element' of the same section.

2.120 D21194

In section D8.6.1 (The Contiguous bit), subsection 'Misprogramming the Contiguous bit', the rules which read:

> R_{QTRPM}
>
> For all descriptors within the range indicated by one or more descriptors that have the Contiguous bit set to 1, if one or more of the following translation table programming errors is made, then a TLB might contain overlapping entries:
>
> - One or more of the contiguous translation table entries does not have the Contiguous bit set to 1.
> - One or more of the contiguous translation table entries holds an OA that is not consistent with all of the entries pointing to the same aligned contiguous address range.
> - The attributes and permissions of the contiguous entries are not the same.
>
> R_{NGLXZ}
>
> For a TLB lookup in a contiguous region mapped by translation table entries that are misprogrammed, that TLB lookup is permitted to produce one of the following:
>
> - An OA, access permissions, and memory attributes that are consistent with any of the programmed translation table values.
> - If BBM support levels 1 and 2 are not implemented, then an OA, access permissions, or memory attributes that are inconsistent with any of the programmed translation table values. For more information, see Support levels for changing block size on page D8-5935.
> - A TLB conflict abort.

are updated to read:

> R_{QTRPM}
>
> For all descriptors within the range indicated by one or more descriptors that have the Contiguous bit set to 1, then a TLB might contain overlapping entries if one or more of the contiguous translation table entries does not have the Contiguous bit set to 1.
>
> R_{NGLXZ}
>
> For a TLB lookup in a contiguous region mapped by translation table entries that have the Contiguous bit misprogrammed, that TLB lookup is permitted to produce one of the following:
>
> - An OA, access permissions, and memory attributes that are consistent with any of the programmed translation table values.

- If BBM support levels 1 and 2 are not implemented, then an OA, access permissions, or memory attributes that are inconsistent with any of the programmed translation table values. For more information, see Support levels for changing block size on page D8-5935.
- A TLB conflict abort.

R_{X0000}

For a TLB lookup in a contiguous region mapped by translation table entries that have consistent values for the Contiguous bit, but have the OA, attributes, or permissions misprogrammed, that TLB lookup is permitted to produce an OA, access permissions, and memory attributes that are consistent with any one of the programmed translation table values.

I_{X0001}

The Contiguous bit is present only in valid Block and Page translation table descriptors, and therefore neither of the following configurations are considered as misprogramming of the Contiguous bit:

- A contiguous range of descriptors that are each either invalid, or valid with Contiguous set to 1.
- A contiguous range of descriptors that are each either invalid, or valid with Contiguous set to 0.

2.121 R21198

In section D19.12.2 (CNTHCTL_EL2, Counter-timer Hypervisor Control register), the following pseudocode function is introduced:

```
// CNTHCTL_EL2_VHE()
// =================
// In the case where EL2 accesses the CNTKCTL_EL1 name, and the access
// is redirected to CNTHCTL_EL2 as a result of HCR_EL2.E2H being 1,
// then the bits of CNTHCTL_EL2 that are RES0 in CNTKCTL_EL1 are
// treated as being UNKNOWN. This function applies the UNKNOWN behavior.

bits(64) CNTHCTL_EL2_VHE(bits(64) original_value)
    assert PSTATE.EL == EL2;
    assert HCR_EL2.E2H == '1';

    bits(64) return_value = original_value;

    return_value<19:18> = bits(2) UNKNOWN;
    return_value<16:10> = bits(7) UNKNOWN;
    return return_value;
```

In section D19.12.15 (CNTKCTL_EL1, Counter-timer Kernel Control register), the accessor pseudocode for MRS <Xt>, CNTKCTL_EL1 becomes:

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    X[t, 64] = CNTKCTL_EL1;
```

```
elsif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' then
        X[t, 64] = CNTHCTL_EL2_VHE(CNTHCTL_EL2);
    else
        X[t, 64] = CNTKCTL_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = CNTKCTL_EL1;
```

The accessor pseudocode for MSR CNTKCTL_EL1, <Xt> becomes:

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    CNTKCTL_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' then
        CNTHCTL_EL2 = CNTHCTL_EL2_VHE(X[t, 64]);
    else
        CNTKCTL_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    CNTKCTL_EL1 = X[t, 64];
```

2.122 D21204

In section D19.2.94 (ISR_EL1, Interrupt Status Register), the 'Purpose' text that reads:

> Shows the pending status of the IRQ, FIQ, or SError interrupt.
>
> When executing at EL2, EL3 or Secure EL1 when SCR_EL3.EEL2 == `0b0`, this shows the pending status of the physical IRQ, FIQ, or SError interrupts.
>
> When executing at either Non-secure EL1 or at Secure EL1 when SCR_EL3.EEL2 == `0b1`:
>
> - If the HCR_EL2.{IMO,FMO,AMO} bit has a value of 1, the corresponding ISR_EL1.{I,F,A} bit shows the pending status of the virtual IRQ, FIQ, or SError.
> - If the HCR_EL2.{IMO,FMO,AMO} bit has a value of 0, the corresponding ISR_EL1.{I,F,A} bit shows the pending status of the physical IRQ, FIQ, or SError.

is changed to read:

> Shows the pending status of the IRQ, FIQ, and SError interrupts. When FEAT_NMI is implemented, also shows whether a pending IRQ or FIQ interrupt has Superpriority.

A similar change is made in section G8.2.103 (ISR, Interrupt Status Register), where the 'Purpose' is updated to read:

> Shows the pending status of the IRQ, FIQ, and SError interrupts.

Also in section D19.2.94, the following text is added to ISR_EL1.F, along with equivalent changes in the ISR_EL1.{A, I} descriptions:

> If all of the following apply then this field shows the pending status of virtual FIQ interrupts:

- EL2 is implemented and enabled in the current Security state.
- HCR_EL2.FMO is 1.
- The PE is executing at EL1.

Otherwise, this field shows the pending status of physical FIQ interrupts.

And the following text is added to ISR_EL1.FS, along with an equivalent change in ISR_EL1.IS:

If all of the following apply then this field shows the pending status of virtual FIQ interrupts with Superpriority:

- EL2 is implemented and enabled in the current Security state.
- HCR_EL2.FMO is 1.
- The PE is executing at EL1.

Similarly, in section G8.2.103, the following text is added to ISR.F, along with equivalent changes in the ISR.{A, I} descriptions:

If all of the following apply then this field shows the pending status of virtual FIQ interrupts:

- EL2 is implemented and enabled in the current Security state.
- Any of the following apply:
  - EL2 is using AArch64 and HCR_EL2.FMO is 1.
  - EL2 is using AArch32 and HCR.FMO is 1.
- The PE is executing at EL1.

Otherwise, this field shows the pending status of physical FIQ interrupts.

2.123 R21211

In section H9.2.35 (EDPFR, External Debug Processor Feature Register), the field 'Bits [55:52]' , which reads:

Reserved, **RES0**.

is updated to read

Reserved, **UNKNOWN**.

Similar changes made to other fields.

2.124 C21257

In section D9.4.4 (GPT Contiguous descriptor), the following information statement:

> I_{MVQFG}
>
> There is no encoding for a 64KB contiguous region for the case where PGS is set to 4KB. If PGS is 4KB, then an implementation is permitted to treat a GPT Granules descriptor containing 16 identical GPI values as a 64KB block region.

is removed.

2.125 D21271

In section D19.2.138 (TCR2_EL2, Extended Translation Control Register (EL2)), section 'When HCR_EL2.E2H == 0', field 'AMEC0, bit[12]', the reset information that reads:

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

is corrected to read:

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

Equivalent changes are made in section 'When HCR_EL2.E2H == 1', in the fields 'AMEC1, bit [13]' and 'AMEC0, bit [12]'.

2.126 D21272

In section D19.2.138 (TCR2_EL2, Extended Translation Control Register (EL2)), subsection 'When HCR_EL2.E2H == 0', subsection 'AMEC0, bit[12]' the following text is added:

> When SCTLR2_EL2.EMEC is 0, this field is ignored by the PE, and the bit position of AMEC is **RES0** in Block and Page descriptors.

and the text that reads:

> Accessing this field has the following behavior:
>
> - When IsCurrentSecurityState(SS_Secure), access to this field is **RES0**.
> - When IsCurrentSecurityState(SS_NonSecure), access to this field is **RES0**.
> - When SCTLR2_EL2.EMEC != 0, access to this field is **RES1**.

is changed to read:

Accessing this field has the following behavior:

- When IsCurrentSecurityState(SS_Secure), access to this field is **RES0**.
- When IsCurrentSecurityState(SS_NonSecure), access to this field is **RES0**.

The equivalent changes are made in the subsection 'When HCR_EL2.E2H==1', subsections 'AMEC1, bit[13]' and 'AMEC0, bit[12]'.

2.127 D21292

In section C1.2.5 (SVE Condition Code Aliases), in table C1-2 'SVE Condition codes', the row that reads:

cond	Mnemonic	SVE alias	Meaning	Condition flags
1001	LS	PLAST	The Last active element was TRUE, or all Active elements were FALSE, or there were no Active elements.	C==1 || Z==0

is corrected to read:

cond	Mnemonic	SVE alias	Meaning	Condition flags
1001	LS	PLAST	The Last active element was TRUE, or all Active elements were FALSE, or there were no Active elements.	C==0 || Z==1

2.128 D21307

In section D1.3.6 (Asynchronous exception types) the rule:

R_{GGWMZ}

If FEAT_NMI is implemented and SCTLR_ELx.NMI is 1, IRQ, FIQ, vIRQ, and vFIQ interrupts can have Superpriority as an additional attribute. When SCTLR_ELx.NMI is 0, the denotion of Superpriority has no effect on these interrupts. See also R_{MHWBP}, R_{GFXKY}, R_{SNLJH}, and R_{XSPSG}.

is changed to read:

R_{GGWMZ}

If FEAT_NMI is implemented, then IRQ, FIQ, vIRQ, and vFIQ interrupts can have Superpriority as an additional attribute.

I_{X0001}

When SCTLR_ELx.NMI is 0, the attribute of Superpriority has no effect on interrupts taken to ELx.

> See also Physical interrupt masking and Virtual interrupt masking.

2.129 D21308

In section D8.14.5 (TLB Maintenance instructions), subsection 'TLB maintenance instruction scope', the bullet under rule R_{VMSWQ} which reads:

> - For entries from a translation regime in which an ASID is valid, the instruction applies only to all of the following cached copies of translation table entries:
>   - A non-global entry from the final lookup level that matches the specified ASID.
>   - A global entry from the final lookup level.

is corrected to read:

> - For entries from a translation regime in which an ASID is valid, the instruction applies only to all of the following cached copies of translation table entries:
>   - A non-global entry from the final lookup level that matches the specified ASID.
>   - A global entry from any lookup level.

2.130 D21373

In section H9.2.42 (EDPRSR, External Debug Processor Status Register), the SPD, bit [1] field description, the text that reads:

> When FEAT_DoPD is not implemented and the Core power domain is in either retention or powerdown state, the value of EDPRSR.SPD is implementation defined. For more information, see 'EDPRSR.SPD when the Core domain is in either retention or powerdown state'.
>
> EDPRSR.{DLK, SPD, PU} describe whether registers in the Core power domain can be accessed, and whether their state has been lost since the last time the register was read. For more information, see 'EDPRSR.{DLK, SPD, PU} and the Core power domain'.
>
> This field is in the Core power domain.

is corrected to read:

> When FEAT_DoPD is not implemented and the Core power domain is in either retention or powerdown state, the value of EDPRSR.SPD is implementation defined. For more information, see 'EDPRSR.SPD when the Core domain is in either retention or powerdown state'.
>
> EDPRSR.{DLK, SPD, PU} describe whether registers in the Core power domain can be accessed, and whether their state has been lost since the last time the register was read. For more information, see 'EDPRSR.{DLK, SPD, PU} and the Core power domain'.

2.131 D21376

In section B2.9.5 (Load-Exclusive and Store-Exclusive instruction usage restrictions), the Note that reads:

> - The TLB invalidation will clear either the local or global monitor.
> - The PA will be checked between the LoadExcl and StoreExcl.

is changed to read:

> Arm expects that, in many implementations, either:
> - The TLB invalidation will clear either the local or global monitor.
> - The PA will be checked between the LoadExcl and StoreExcl.

2.132 D21392

In section D19.2.140, (TCR_EL2, Translation Control Register (EL2)), subsection 'When HCR_EL2.E2H == 0', the text that reads:

> Any of the bits in TCR_EL2, other than the A1 bit and the EPDx bits when they have the value 1, are permitted to be cached in a TLB.

is changed to read:

> Any of the bits in TCR_EL2 are permitted to be cached in a TLB.

The equivalent changes are made in D19.2.139 (TCR_EL1, Translation Control Register (EL1)), subsection 'When HCR_EL2.E2H == 0'.

In section D19.2.140, (TCR_EL2, Translation Control Register (EL2)), subsection 'When HCR_EL2.E2H == 1', the text that reads:

> Any of the bits in TCR_EL2 are permitted to be cached in a TLB.

is changed to read:

> Any of the bits in TCR_EL2, other than the A1 bit and the EPDx bits when they have the value 1, are permitted to be cached in a TLB.

The equivalent changes are made in D19.2.139 (TCR_EL1, Translation Control Register (EL1)), subsection 'When HCR_EL2.E2H == 1'.

In section D19.2.137 (TCR2_EL1, Extended Translation Control Register (EL1)), the text that reads:

> Unless stated otherwise, all the bits in TCR2_EL2 are permitted to be cached in a TLB.

is changed to read:

> Unless stated otherwise, all the bits in TCR2_EL1 are permitted to be cached in a TLB.

In section D.19.138 (TCR2_EL2, Extended Translation Control Register (EL2)), subsection 'When HCR_EL2.E2H == 0', field 'AMEC0, bit[12]', the following text is added:

> This bit is permitted to be cached in a TLB only if it is 1.

The equivalent changes are made in the following sections:

- D19.2.138 (TCR2_EL2, Extended Translation Control Register (EL2)), subsection 'When HCR_EL2.E2H==1', field 'AMEC1, bit[13]'
- D19.2.138 (TCR2_EL2, Extended Translation Control Register (EL2)), subsection 'When HCR_EL2.E2H==1', field 'AMEC0, bit[12]'

2.133 D21408

In section D8.9.1 (Effect of MEC on PA spaces), subsection 'Effect on the EL3 translation regime', in rule R_{FMLTL} the text that reads:

> For EL3 stage 1 translated addresses, if SCTLR2_EL3.MECRL is 0, then accesses to the Realm PA space use the default MECID of zero.

is corrected to:

> For EL3 stage 1 translated addresses, if SCTLR2_EL3.EMEC is 0, then accesses to the Realm PA space use the default MECID of zero.

The equivalent corrections are made throughout D8.9 (Memory Encryption Contexts extension)

In section D19.11.1 (MECIDR_EL2, MEC Identification Register), in field 'MECIDWidthm1, bits [3:0]', the following text is removed:

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

2.134 D21435

In section A2.10.1 (Architectural features added by Armv8.8), under "FEAT_SPEv1p3, Armv8.8 Statistical Profiling Extensions", the statement that reads:

> This feature is supported in both AArch64 and AArch32 states.

is corrected to read:

> This feature is supported only in AArch64 state.

2.135 C21437

In section D8.9.1 (Effect of MEC on PA spaces), subsection 'Effect on the Realm EL2 and Realm EL2&0 translation regimes', the following clarification is added after R_{XBDTH}:

> I_{TSRTP}
>
> TCR_EL2.A1 is not permitted to be cached in a TLB, and MECID values are not permitted to be cached in a TLB. However, the choice of which MECID register is used for translation table walks is permitted to be cached in a TLB, as long as it is only cached in TLB entries that are associated with the ASID that is also selected by the A1 field.

2.136 D21440

In section D19.2.48 (HCR_EL2, Hypervisor Configuration Register), subsection "Configurations", the sentence that reads:

> The bits in this register behave as if they are 0 for all purposes other than direct reads of the register if EL2 is not enabled in the current Security state.

is corrected to read:

> Unless otherwise stated, the bits in this register behave as if they are 0 for all purposes other than direct reads of the register if EL2 is not enabled in the current Security state.

In the same section, the following text is added to the description of field "ATA, bit [56]":

> If EL2 is not enabled in the current security state, this field behaves as if it is 1 for all purposes other than direct reads of the field.

2.137 D21444

The following rules apply to the following register descriptions:

> R_{CBZWL}
>
> In a system that supports the Realm Management Extension, CNTControlBase is accessible only by Root accesses.

R_{CBZWL} applies to all of:

- Section I6.7.2 (CNTCR, Counter Control Register)
- Section I6.7.3 (CNTCV, Counter Count Value register), in the CNTControlBase frame
- Section I6.7.5 (CNTFID0, Counter Frequency ID)
- Section I6.7.6 (CNTFID<n>, Counter Frequency IDs, n > 0, n = 1 - 1003)

- Section I6.7.8 (CNTID, Counter Identification Register)
- Section I6.7.14 (CNTSCR, Counter Scale Register)
- Section I6.7.15 (CNTSR, Counter Status Register)
- Section I6.7.23 (CounterID<n>, Counter ID registers, n = 0 - 11), in the CNTControlBase frame

> R_{LJZFN}
>
> For any register in CNTBaseN, CNTEL0BaseN, or CNTCTLBase described as only permitting Secure accesses:
>
> - For Root accesses, it is **IMPLEMENTATION DEFINED** whether accesses are permitted or behave as **RES0**.
> - For Realm accesses, the register behaves as **RES0**.

R_{LJZFN} applies to all of:

- Section I6.7.7 (CNTFRQ, Counter-timer Frequency), in the CNTCTLBase frame
- Section I6.7.9 (CNTNSAR, Counter-timer Non-secure Access Register)

> R_{LTXDR}
>
> For any register in CNTBaseN, CNTEL0BaseN, or CNTCTLBase described as permitting Non-secure access, it is **IMPLEMENTATION DEFINED** whether Root and Realm accesses are permitted. If not permitted, the register behaves as **RES0** for Root and Realm accesses.

R_{LTXDR} applies to all of:

- Section I6.7.16 (CNTTIDR, Counter-timer Timer ID Register)
- Section I6.7.23 (CounterID<n>, Counter ID registers, n = 0 - 11), in the CNTBaseN frame

For the following registers, R_{LJZFN} applies when CNTNSAR.NS<n>==0b0 and R_{LTXDR} applies when CNTNSAR.NS<n>==0b1:

- Section I6.7.1 (CNTACR<n>, Counter-timer Access Control Registers, n = 0 - 7)
- Section I6.7.4 (CNTEL0ACR, Counter-timer EL0 Access Control Register)
- Section I6.7.7 (CNTFRQ, Counter-timer Frequency), in the CNTBaseN frame
- Section I6.7.10 (CNTP_CTL, Counter-timer Physical Timer Control)
- Section I6.7.11 (CNTP_CVAL, Counter-timer Physical Timer CompareValue)
- Section I6.7.12 (CNTP_TVAL, Counter-timer Physical Timer TimerValue)
- Section I6.7.13 (CNTPCT, Counter-timer Physical Count)
- Section I6.7.17 (CNTV_CTL, Counter-timer Virtual Timer Control)
- Section I6.7.18 (CNTV_CVAL, Counter-timer Virtual Timer CompareValue)
- Section I6.7.19 (CNTV_TVAL, Counter-timer Virtual Timer TimerValue)
- Section I6.7.20 (CNTVCT, Counter-timer Virtual Count)

- Section I6.7.21 (CNTVOFF, Counter-timer Virtual Offset), in the CNTBaseN frame
- Section I6.7.22 (CNTVOFF<n>, Counter-timer Virtual Offsets, n = 0 - 7), in the CNTCTLBase frame
- Section I6.7.23 (CounterID<n>, Counter ID registers, n = 0 - 11), in the CNTBaseN frame

For each register in the CNTEL0BaseN frame, the same rule or rules apply as for the equivalent register in the CNTBaseN frame.

2.138 C21461

In section D1.3.5 (Synchronous exception types), All references to a pointer authentication instruction authentication failure are changed to use the term “PAC Fail exception”.

In the same section, Priority 25 in “Prioritization of Synchronous exceptions taken to AArch64 state” that reads:

> When FEAT_FPAC is implemented, an exception generated by the failure to authorize a Pointer Authentication instruction.

is changed to read:

> When FEAT_FPAC is implemented, a PAC Fail exception.

In section D8.8.4 (Faulting on pointer authentication), the references to “Pointer Authentication instruction authentication failure exception” are changed to “PAC Fail exception”.

The sub-section named “ISS encoding for an exception from a Pointer Authentication instruction authentication failure” is changed to “ISS encoding for a PAC Fail exception” in the following sections:

- D19.2.37 (ESR_EL1, Exception Syndrome Register (EL1))
- D19.2.38 (ESR_EL2, Exception Syndrome Register (EL2))
- D19.2.39 (ESR_EL3, Exception Syndrome Register (EL3))

2.139 D21467

In section B2.5.2 (Alignment of data accesses), the text that reads:

> An unaligned access to any type of Device memory causes an Alignment fault.

is updated to read:

> For an unaligned access to any type of Device memory:
>
> - If the memory location cannot support unaligned accesses then an Alignment fault is generated.

> - If the memory location supports unaligned accesses, then it is **IMPLEMENTATION DEFINED** whether the access generates an Alignment fault if the location would not generate an Alignment fault if the same access were made to Normal memory.

to reflect R17932.

2.140 C21481

In section D1.3.1 (Exception entry terminology), the rules that read:

> R_{JRMVK}
>
> Instructions that intentionally cause an exception to be taken are called exception producing instructions.
>
> I_{MGYJT}
>
> The following are exception producing instructions:
> - HVC.
> - SMC.
> - SVC.

are clarified to read:

> R_{JRMVK}
>
> The exception producing instructions are the following:
> - HVC.
> - SMC.
> - SVC.

In section D1.3.9 (Exception Producing Instructions), the information statement that reads:

> I_{YRYNQ}
>
> The exception producing instructions are commonly called system calls and refer to any of the following synchronous exception types:
> - Supervisor Call exception.
> - Hypervisor Call exception.
> - Secure Monitor Call exception.

is clarified to read:

> I_{YRYNQ}

> The exception producing instructions are commonly called system calls and refer to the instructions that generate any of the following synchronous exception types:
>
> - Supervisor Call exception.
> - Hypervisor Call exception.
> - Secure Monitor Call exception.

2.141 C21487

In section D9.2 (GPC faults) the following rule:

> R_{RLQVP}
>
> If a RAS error is detected on a fetch of GPT information to check an access, then the access generates a synchronous External abort on GPT fetch at level x where x is the level of the fetch that consumed the RAS error.

is removed.

2.142 D21512

In section D1.3.2, (Exception entry), rule R_{WTXBY}, the text that reads:

> When an exception is taken to an Exception level, ELx, that is using AArch64 state, after recording the previous values in SPSR_ELx, the following PSTATE bits are set:
>
> ...
>
> - For any of the following situations, PSTATE.PAN is set to 1:
> - The target Exception level is EL1 and SCTLR_EL1.SPAN is 0.
> - The exception is taken from EL0, the target Exception level is EL2 using AArch64, Secure EL2 is enabled, HCR_EL2.{TGE, E2H} is {1, 1}, and SCTLR_EL2.SPAN is 0.

is corrected to read:

> When an exception is taken to an Exception level, ELx, that is using AArch64 state, after recording the previous values in SPSR_ELx, the following PSTATE bits are set:
>
> ...
>
> - PSTATE.SS is set according to the rules in Chapter D2 AArch64 Self-hosted Debug.
> - For any of the following situations, PSTATE.PAN is set to 1:
> - The target Exception level is EL1 and SCTLR_EL1.SPAN is 0.
> - The exception is taken from EL0, the target Exception level is EL2 using AArch64, HCR_EL2.{TGE, E2H} is {1, 1}, and SCTLR_EL2.SPAN is 0.

2.143 C1408: SME

In section J1.3.3 (shared/functions), the FPDot() pseudocode function that reads:

```
    ...

    if (((inf1_a && zero2_a) || (zero1_a && inf2_a)) &&
        ((inf1_b && zero2_b) || (zero1_b && inf2_b))) then
        result = FPDefaultNaN(fpcr, N);
        if fpexc then FPProcessException(FPExc_InvalidOp, fpcr);

    if !done then
        ...

        // Non SNaN-generated Invalid Operation cases are multiplies of zero
        // by infinity and additions of opposite-signed infinities.
        invalidop = ((inf1_a && zero2_a) || (zero1_a && inf2_a) ||
            (inf1_b && zero2_b) || (zero1_b && inf2_b) || (infPa && infPb &&
signPa != signPb));

        if invalidop then
            result = FPDefaultNaN(fpcr, N);
            if fpexc then FPProcessException(FPExc_InvalidOp, fpcr);

    ...
```

is updated to read:

```
    ...

    if !done then
        ...

        // Non SNaN-generated Invalid Operation cases are multiplies of zero
        // by infinity and additions of opposite-signed infinities.
        invalidop = ((inf1_a && zero2_a) || (zero1_a && inf2_a) ||
            (inf1_b && zero2_b) || (zero1_b && inf2_b) || (infPa && infPb &&
signPa != signPb));

        if invalidop then
            result = FPDefaultNaN(fpcr, N);
            if fpexc then FPProcessException(FPExc_InvalidOp, fpcr);

    ...
```

This update does not cause any change to the behavior of instructions which call the FPDot() function.

2.144 R1555: SME

In sections C8.2 (Alphabetical list of SVE instructions) and C9.2 (Alphabetical list of SME instructions), the “Operational Information” subsection in the WHILE<cc> instruction pages that reads:

> When PSTATE.DIT is 1:

- The execution time of this instruction is independent of:
  - The values of the data supplied in any of its registers.
  - The values of the NZCV flags.
- The response of this instruction to asynchronous exceptions does not vary based on:
  - The values of the data supplied in any of its registers.
  - The values of the NZCV flags{panel}

is removed.

2.145 D494: SVE2

In section C8.2 (Alphabetical list of SVE instructions), the following text is added to the 'Operation Information' subsection of all predicated SVE load/store (vector) instructions, except for the first-fault (FF) and non-fault (NF) loads:

> If FEAT_SVE2 is implemented or FEAT_SME is implemented, then when PSTATE.DIT is 1, the timing of this instruction is insensitive to the value of the data being loaded or stored when its governing predicate register contains the same value for each execution.

The following text is added to the 'Operational Information' subsection of all unpredicated SVE load/store (vector and predicate) instructions:

> If FEAT_SVE2 is implemented or FEAT_SME is implemented, then when PSTATE.DIT is 1, the timing of this instruction is insensitive to the value of the data being loaded or stored.

In section C8.2.82 (CNT), the following text is added to the 'Operational Information' subsection:

> If FEAT_SVE2 is implemented or FEAT_SME is implemented, then when PSTATE.DIT is 1:
> - The execution time of this instruction is independent of:
>   - The values of the data supplied in any of its operand registers when its governing predicate register contains the same value for each execution.
>   - The values of the NZCV flags.
> - The response of this instruction to asynchronous exceptions does not vary based on:
>   - The values of the data supplied in any of its operand registers when its governing predicate register contains the same value for each execution.
>   - The values of the NZCV flags.

2.146 D210: SVE

In section D12.11.3 (Common event numbers), in the subsection 'Common microarchitectural events', the event descriptions that read:

> `0x803D`, ASE_FP_PREDUCE_SPEC, Floating-point operation speculatively executed, Advanced SIMD pairwise add step
>
> The counter counts each Speculatively executed floating-point pairwise add operation counted by ASE_FP_SPEC due to any of the following A64 instructions:
>
> - Advanced SIMD: FADDP.
>
> It is **IMPLEMENTATION DEFINED** which floating-point pairwise add operations are counted in AArch32 state.
>
> `0x803E`, SVE_FP_VREDUCE_SPEC, Floating-point operation speculatively executed, SVE vector reduction
>
> The counter counts each Speculatively executed floating-point treewise reduction operation counted by SVE_FP_SPEC due to any of the following A64 instructions:
>
> - SVE: FADDV, FMAXNMV, FMAXV, FMINNMV, or FMINV.
>
> Implementation of this optional event requires that FEAT_SVE is implemented.
>
> `0x803F`, ASE_SVE_FP_VREDUCE_SPEC, Floating-point operation speculatively executed, Advanced SIMD or SVE vector reduction
>
> The counter counts each Speculatively executed floating-point reduction operation counted by ASE_SVE_FP_SPEC due to any of the following A64 instructions:
>
> - Advanced SIMD: FADDP, FMAXNMV, FMAXV, FMINNMV, or FMINV.
> - SVE: FADDV, FMAXNMV, FMAXV, FMINNMV, or FMINV.
>
> It is **IMPLEMENTATION DEFINED** which floating-point reduction operations are counted in AArch32 state.

are updated to read:

> `0x803D`, ASE_FP_PREDUCE_SPEC, Floating-point operation speculatively executed, Advanced SIMD pairwise step
>
> The counter counts each Speculatively executed floating-point pairwise add operation counted by ASE_FP_SPEC due to any of the following A64 instructions:
>
> - Advanced SIMD: FADDP, FMAXNMP, FMAXP, FMINNMP or FMINP.
>
> It is **IMPLEMENTATION DEFINED** which floating-point pairwise add operations are counted in AArch32 state.

`0x803E`, SVE_FP_VREDUCE_SPEC, Floating-point operation speculatively executed, SVE vector reduction

The counter counts each Speculatively executed floating-point treewise or pairwise reduction operation counted by SVE_FP_SPEC due to any of the following A64 instructions:

- SVE: FADDV, FMAXNMV, FMAXV, FMINNMV, or FMINV.
- SVE2: FADDP, FMAXNMP, FMAXP, FMINNMP, or FMINP.

Implementation of this optional event requires that FEAT_SVE is implemented.

`0x803F`, ASE_SVE_FP_VREDUCE_SPEC, Floating-point operation speculatively executed, Advanced SIMD or SVE vector reduction

The counter counts each Speculatively executed floating-point treewise or pairwise reduction operation counted by ASE_SVE_FP_SPEC due to any of the following A64 instructions:

- Advanced SIMD: FADDP, FMAXNMP, FMAXNMV, FMAXP, FMAXV, FMINNMP, FMINNMV, FMINP, or FMINV.
- SVE: FADDV, FMAXNMV, FMAXV, FMINNMV, or FMINV.
- SVE2: FADDP, FMAXNMP, FMAXP, FMINNMP, or FMINP.

It is **IMPLEMENTATION DEFINED** which floating-point reduction operations are counted in AArch32 state.

Within the same section and subsection, the event descriptions that read:

`0x805D`, ASE_INT_VREDUCE_SPEC, Integer operation speculatively executed, Advanced SIMD reduction

The counter counts each Speculatively executed across-vector and pairwise integer reduction operation counted by ASE_INT_SPEC due to any of the following A64 instructions:

- Advanced SIMD: SADDLP, SADDLV, SMAXP, SMAXV, SMINP, SMINV, UADDVL, UMAXV, or UMINV.

It is **IMPLEMENTATION DEFINED** which across-vector and pairwise integer reduction operations arecounted in AArch32 state.

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

The counter counts each Speculatively executed across-vector integer reduction operation counted by SVE_INT_SPEC due to any of the following A64 instructions:

- SVE: ANDV, EORV, ORV, SADDV, SMAXV, SMINV, UADDV, UMAXV, or UMINV. Implementation of this optional event requires that FEAT_SVE is implemented.

`0x805F`, ASE_SVE_INT_VREDUCE_SPEC, Integer operation speculatively executed, Advanced SIMD or SVE reduction

> The counter counts each Speculatively executed across-vector and pairwise integer reduction operation counted by either ASE_INT_VREDUCE_SPEC or SVE_INT_VREDUCE_SPEC. That is, due to any of the following A64 instructions:
>
> - Advanced SIMD: SADDLP, SADDLV, SMAXP, SMAXV, SMINP, SMINV, UADDVL, UMAXV, or UMINV.
> - SVE: ANDV, EORV, ORV, SADDV, SMAXV, SMINV, UADDV, UMAXV, or UMINV.
>
> It is **IMPLEMENTATION DEFINED** which across-vector and pairwise integer reduction operations are counted in AArch32 state.

are updated to read:

> `0x805D`, ASE_INT_VREDUCE_SPEC, Integer operation speculatively executed, Advanced SIMD reduction
>
> The counter counts each Speculatively executed across-vector and pairwise integer reduction operation counted by ASE_INT_SPEC due to any of the following A64 instructions:
>
> - Advanced SIMD: ADDP, ADDV, SADALP, SADDLP, SADDLV, SMAXP, SMAXV, SMINP, SMINV, UADALP, UADDLP, UADDLV, UMAXP, UMAXV, UMINP, or UMINV.
>
> It is **IMPLEMENTATION DEFINED** which across-vector and pairwise integer reduction operations arecounted in AArch32 state.
>
> `0x805E`, SVE_INT_VREDUCE_SPEC, Integer operation speculatively executed, SVE reduction
>
> The counter counts each Speculatively executed across-vector and pairwise integer reduction operation counted by SVE_INT_SPEC due to any of the following A64 instructions:
>
> - SVE: ANDV, EORV, ORV, SADDV, SMAXV, SMINV, UADDV, UMAXV, or UMINV.
> - SVE2: ADDP, SADALP, SMAXP, SMINP, UADALP, UMAXP, or UMINP.
>
> Implementation of this optional event requires that FEAT_SVE is implemented.
>
> `0x805F`, ASE_SVE_INT_VREDUCE_SPEC, Integer operation speculatively executed, Advanced SIMD or SVE reduction
>
> The counter counts each Speculatively executed across-vector and pairwise integer reduction operation counted by either ASE_INT_VREDUCE_SPEC or SVE_INT_VREDUCE_SPEC. That is, due to any of the following A64 instructions:
>
> - Advanced SIMD: ADDP, ADDV, SADALP, SADDLP, SADDLV, SMAXP, SMAXV, SMINP, SMINV, UADALP, UADDLP, UADDLV, UMAXP, UMAXV, UMINP, or UMINV.
> - SVE: ANDV, EORV, ORV, SADDV, SMAXV, SMINV, UADDV, UMAXV, or UMINV.
> - SVE2: ADDP, SADALP, SMAXP, SMINP, UADALP, UMAXP, or UMINP.
>
> It is **IMPLEMENTATION DEFINED** which across-vector and pairwise integer reduction operations are counted in AArch32 state.

2.147 C217: SVE

In section D1.3.5 (Synchronous exception types), subsection “SVE First-fault and Non-fault loads”, the following information statement is added:

> I_{X0001}
>
> A memory access performed for the First active element of an SVE First-fault vector load instruction is not precisely equivalent to the memory access for an Active element of a regular SVE predicated vector load instruction. When the initial value of the FFR element corresponding to the First active element is FALSE, then even if an access is performed for that element and does not detect a Memory fault, the content of the corresponding destination vector element is still **CONSTRAINED UNPREDICTABLE**, as described in rule R_{NGFTJ}.

2.148 C313: SVE

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

> If FPCR.FZ16 == 1, for floating-point instructions other than FABS, FNEG, FMAX*, and FMIN*, if the instruction processes half-precision numbers, flushing denormalized output numbers to zero can be controlled as follows:

is clarified to read:

> If FPCR.FZ16 == 1, for floating-point instructions other than FABS, FNEG, FMAX, FMAXP, FMAXV, FMIN, FMINP, and FMINV, if the instruction processes half-precision numbers, flushing denormalized output numbers to zero can be controlled as follows:

In the same section, the bullet that reads:

> - For FABS, FNEG, FMAX*, and FMIN*, denormalized output operands are not flushed to zero.

is clarified to read:

> For FABS, FNEG, FMAX, FMAXP, FMAXV, FMIN, FMINP, and FMINV, denormalized output operands are not flushed to zero.

2.149 C314: SVE

In sections C8.2.150 (FMAX (vectors)) and C8.2.158 (FMIN (vectors)), the following text is removed:

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

In the following sections:

- C7.2.101 (FMAX (vector)).
- C7.2.102 (FMAX (scalar)).
- C7.2.108 (FMAXP (scalar)).
- C7.2.109 (FMAXP (vector)).
- C7.2.110 (FMAXV).
- C7.2.111 (FMIN (vector)).
- C7.2.112 (FMIN (scalar)).
- C7.2.118 (FMINP (scalar)).
- C7.2.119 (FMINP (vector)).
- C7.2.120 (FMINV).

The following text is added:

> When FPCR.AH == 0, the behavior is as follows:
>
> - Negative zero compares less than positive zero.
> - When FPCR.DN is 0, if either input is a NaN, the result is a Quiet NaN.
> - When FPCR.DN is 1, if either input is a NaN, the result is Default NaN.
>
> When FPCR.AH == 1, the behavior is as follows:
>
> - If both inputs are zeros, regardless of the sign of either zero, the result is the second input.
> - If either input is a NaN, regardless of the value of FPCR.DN, the result is the second input.

In sections C8.2.149 (FMAX (immediate)) and C8.2.157 (FMIN (immediate)), the text that reads:

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

is updated to read:

> When FPCR.AH == 0, the behavior is as follows:
>
> - Negative zero compares less than positive zero.
> - When FPCR.DN is 0, if the input is a NaN, the result is a Quiet NaN.
> - When FPCR.DN is 1, if the input is a NaN, the result is Default NaN.
>
> When FPCR.AH == 1, the behavior is as follows:
>
> - If both the input and the immediate are zeros, regardless of the sign of input zero, the result is the immediate.
> - If the input is a NaN, regardless of the value of FPCR.DN, the result is the immediate.

In the following sections:

- C7.2.103 (FMAXNM (vector)).

- C7.2.104 (FMAXNM (scalar)).
- C7.2.106 (FMAXNMP (vector)).
- C7.2.107 (FMAXNMV).
- C7.2.113 (FMINNM (vector)).
- C7.2.114 (FMINNM (scalar)).
- C7.2.116 (FMINNMP (vector)).
- C7.2.117 (FMINNMV).

The text that reads:

> NaNs are handled according to the IEEE 754-2008 standard. If one vector element is numeric and the other is a quiet NaN, the result that is placed in the vector is the numerical value, otherwise the result is identical to ...

is updated to read:

> Regardless of the value of FPCR.AH, the behavior is as follows:
>
> - Negative zero compares less than positive zero.
> - If one input is numeric and the other is a NaN, the result is the numeric value.
> - When FPCR.DN == 0, if either input is a signaling NaN or if both inputs are NaNs, the result is a Quiet NaN.
> - When FPCR.DN == 1, if either input is a signaling NaN or if both inputs are NaNs, the result is Default NaN.

This updated text also replaces the following text in sections C8.2.152 (FMAXNM (vectors)), C8.2.153 (FMAXNMP), C8.2.160 (FMINNM (vectors)), and C8.2.161 (FMINNMP):

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

In sections C8.2.151 (FMAXNM (immediate)) and C8.2.159 (FMINNM (immediate)), the text that reads:

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

is updated to read:

> Regardless of the value of FPCR.AH, the behavior is as follows:
>
> - Negative zero compares less than positive zero.
> - If the input is a Quiet NaN, the result is the immediate value.
> - When FPCR.DN == 0, if the input is a signaling NaN, the result is a Quiet NaN.
> - When FPCR.DN == 1, if the input is a signaling NaN, the result is Default NaN.

2.150 C318: SVE

In section D19.2.139 (TCR_EL1, Translation Control Register (EL1)), in the 'NFD0, bit [53]' field, the `0b1` value description that reads:

> `0b1` A TLB miss on a virtual address that is translated using TTBR0_EL1 due to the specified access types causes the access to fail without taking an exception. The failure should take the same amount of time to be handled as a Permission fault on a TLB entry that is present in the TLB, to mitigate attacks that use fault timing.

is updated to read:

> `0b1` A TLB miss on a virtual address that is translated using TTBR0_EL1 due to the specified access types causes the access to fail without taking an exception. The amount of time that the failure takes to be handled should not predictively leak whether it was caused by a TLB miss or a Permission fault, to mitigate attacks that use fault timing.

Equivalent changes are made to the 'NFD1, bit [54]' field description, and to the 'NFD0, bit [53]' and 'NFD1, bit [54]' field descriptions in section D19.2.140 (TCR_EL2, Translation Control Register (EL2)).

2.151 D1383: Armv9 Debug

In section D4.5.9 (Element Generation), in the subsections 'Exception element' and 'Target Address element', statements are added to recommend that when a branch occurs to an invalid address and the resultant exception is taken from that address, the addresses reported by the Target Address element and the Exception element are the same value.

2.152 D1273: RME

In section D12.5.3 (Prohibiting event and cycle counting), the text that reads:

> The cycle counter, PMCCNTR, counts unless any of the following is true:
>
> ...
>
> - FEAT_PMUv3p5 is implemented, EL3 is implemented, the PE is in Secure state, and SDCR.SCCD is 1.
> - FEAT_PMUv3p7 is implemented, the PE is at EL3, EL3 is using AArch64, and MDCR_EL3.MCCD is 1.

is corrected to read:

> The cycle counter, PMCCNTR, counts unless any of the following is true:
>
> ...

- FEAT_PMUv3p5 is implemented, EL3 is implemented, the PE is in Secure state, and SDCR.SCCD is 1.
- FEAT_PMUv3p7 is implemented, the PE is at EL3, EL3 is using AArch64, and MDCR_EL3.{SCCD, MCCD} is not {0, 0}.

In section J1.2.1 (shared/debug), subsection “shared/debug/pmu/CountPMUEvents”, the code that reads:

```
// Event counting in Secure state is prohibited if all of:
...
if HaveEL(EL3) && ss == SS_Secure then
    if !ELUsingAArch32(EL3) then
        prohibited = MDCR_EL3.SPME == '0' && HavePMUv3p7() && MDCR_EL3.MPMX == '0';
    else
        prohibited = SDCR.SPME == '0';
...
// If FEAT_PMUv3p5 is implemented, cycle counting can be prohibited.
// This is not overridden by PMCR_EL0.DP.
if HavePMUv3p5() && idx == CYCLE_COUNTER_ID then
     if HaveEL(EL3) && ss == SS_Secure then
        sccd = if HaveAArch64() then MDCR_EL3.SCCD else SDCR.SCCD;
        if sccd == '1' then
            prohibited = TRUE;
```

is corrected to read:

```
// Event counting in Secure state and EL3 is prohibited if all of:
...
if HaveEL(EL3) && (ss == SS_Secure || PSTATE.EL == EL3) then
    if !ELUsingAArch32(EL3) then
        prohibited = MDCR_EL3.SPME == '0' && HavePMUv3p7() && MDCR_EL3.MPMX == '0';
    else
        prohibited = SDCR.SPME == '0';
...
// If FEAT_PMUv3p5 is implemented, cycle counting can be prohibited.
// This is not overridden by PMCR_EL0.DP.
if HavePMUv3p5() && idx == CYCLE_COUNTER_ID then
     if HaveEL(EL3) && (ss == SS_Secure || PSTATE.EL == EL3) then
        sccd = if HaveAArch64() then MDCR_EL3.SCCD else SDCR.SCCD;
        if sccd == '1' then
            prohibited = TRUE;
```

In section D19.3.18 (MDCR_EL3, Monitor Debug Configuration Register (EL3)), the text that reads:

> SCCD, bit [23]
>
> When FEAT_PMUv3p5 is implemented:
>
> Secure Cycle Counter Disable. Prohibits PMCCNTR_EL0 from counting in Secure state.

0b0	Cycle counting by PMCCNTR_EL0 is not affected by this mechanism.
0b1	Cycle counting by PMCCNTR_EL0 is prohibited in Secure state.

is corrected to read:

> SCCD, bit [23]

When FEAT_PMUv3p5 is implemented:

Secure Cycle Counter Disable. Prohibits PMCCNTR_EL0 from counting in Secure state and EL3.

`0b0`	Cycle counting by PMCCNTR_EL0 is not affected by this mechanism.
`0b1`	Cycle counting by PMCCNTR_EL0 is prohibited in Secure state and EL3.

2.153 R1345: RME

In section D4.2.1 (Accessing ETE registers), in the subsection 'External debugger interface', the rule R_{KQMKX} is updated to consider the effect of FEAT_RME on external accesses to unimplemented or Reserved registers.

As a result, the text within this rule that reads:

Accesses from the external debugger interface to unimplemented or Reserved registers behave as follows:

- For accesses in the range of offsets `0xF00` to `0xFFC`, the access behaves as **RES0H**.
- For accesses in the range of offsets `0x000` to `0xEFC` when the OS Lock is locked, the access behaves as **RES0H** or returns an error.
- For accesses in the range of offsets `0x000` to `0xEFC` when the OS Lock is unlocked and MDCR_EL3.ETAD is 0, the access behaves as **RES0H**.
- For Secure accesses in the range of offsets `0x000` to `0xEFC` when the OS Lock is unlocked and MDCR_EL3.ETAD is 1, the access behaves as **RES0H**.
- For Non-secure accesses in the range of offsets `0x000` to `0xEFC` when the OS Lock is unlocked and MDCR_EL3.ETAD is 1, the access behaves as **RES0H** or returns an error.

is updated to read:

Accesses from the external debugger interface to unimplemented or Reserved trace unit registers behave as follows:

- When the trace unit Core power domain is off, the access returns an error.
- Otherwise:
  - For accesses in the range of offsets `0xF00` to `0xFFC`, the access behaves as **RES0H**.
  - For accesses in the range of offsets `0x000` to `0xEFC`:
    - When the OS Lock is locked, the response is a **CONSTRAINED UNPREDICTABLE** choice of an error response or behaving as **RES0H**.
    - When the OS Lock is unlocked and AllowExternalTraceAccess() returns FALSE, the response is a **CONSTRAINED UNPREDICTABLE** choice of an error response or behaving as **RES0H**.
    - Otherwise, the access behaves as **RES0H**.