



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

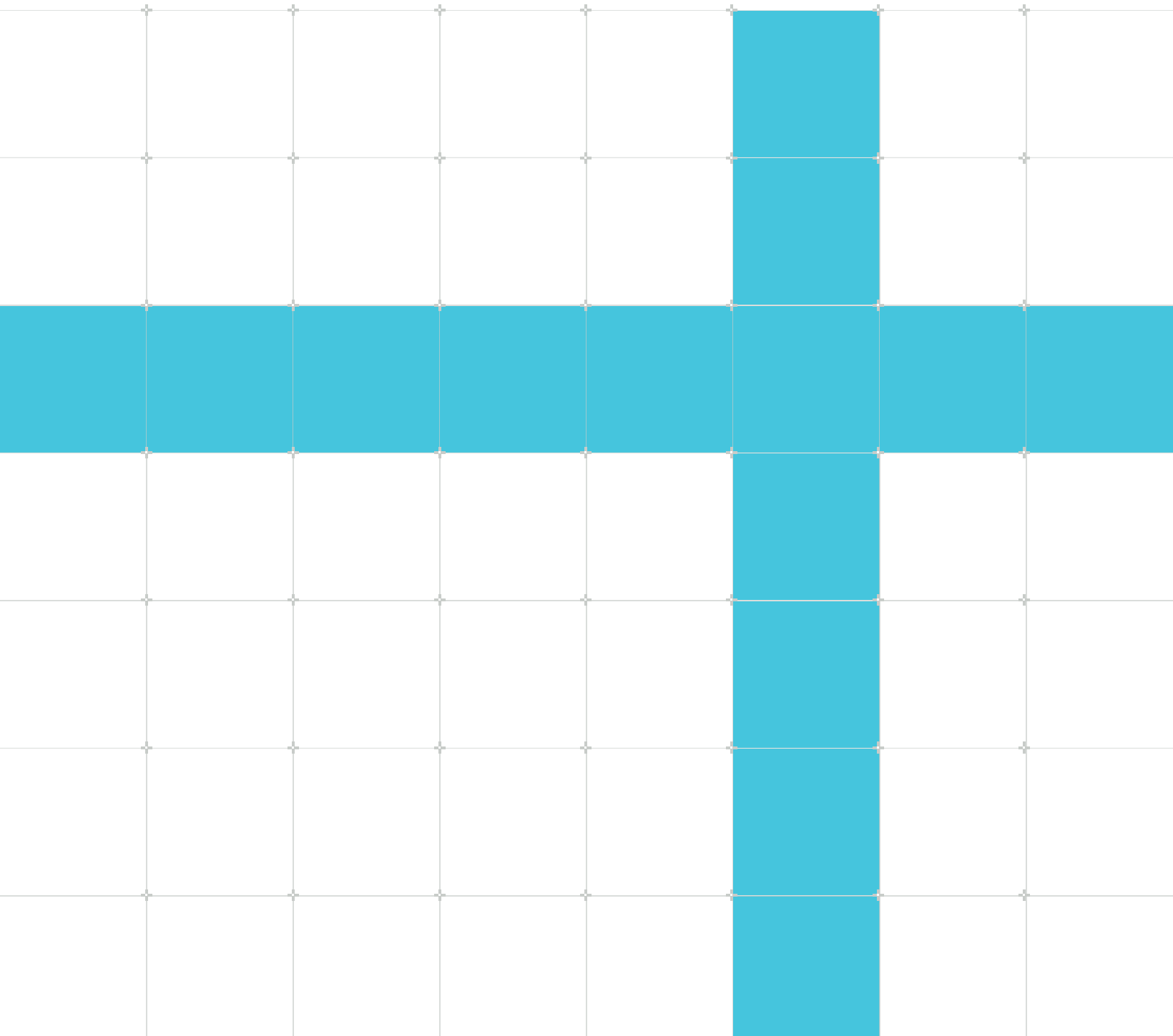
Known issues in Issue G.b

Non-Confidential

Issue 00

Copyright © 2020–2021 Arm Limited (or its affiliates). 102105_G.b_00_en

All rights reserved.



Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile Known issues in Issue G.b

Copyright © 2020–2021 Arm Limited (or its affiliates). All rights reserved.

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.a-05	30 June 2021	Non-Confidential	Known Issues in Arm® Architecture Reference Manual, Issue G.a, as of 18 June 2021
G.b-00	22 July 2021	Non-Confidential	Known Issues in Arm® Architecture Reference Manual, Issue G.b, as of 9 July 2021

Proprietary Notice

This document is protected by copyright and other related rights and the practice or implementation of the information contained in this document may be protected by one or more patents or pending patent applications. No part of this document may be reproduced in any form by any means without the express prior written permission of Arm. No license, express or implied, by estoppel or otherwise to any intellectual property rights is granted by this document unless specifically stated.

Your access to the information in this document is conditional upon your acceptance that you will not use or permit others to use the information for the purposes of determining whether implementations infringe any third party patents.

THIS DOCUMENT IS PROVIDED “AS IS”. ARM PROVIDES NO REPRESENTATIONS AND NO WARRANTIES, EXPRESS, IMPLIED OR STATUTORY, INCLUDING, WITHOUT LIMITATION, THE IMPLIED WARRANTIES OF MERCHANTABILITY, SATISFACTORY QUALITY, NON-INFRINGEMENT OR FITNESS FOR A PARTICULAR PURPOSE WITH RESPECT TO THE DOCUMENT. For the avoidance of doubt, Arm makes no representation with respect to, and has undertaken no analysis to identify or understand the scope and content of, third party patents, copyrights, trade secrets, or other rights.

This document may include technical inaccuracies or typographical errors.

TO THE EXTENT NOT PROHIBITED BY LAW, IN NO EVENT WILL ARM BE LIABLE FOR ANY DAMAGES, INCLUDING WITHOUT LIMITATION ANY DIRECT, INDIRECT, SPECIAL, INCIDENTAL, PUNITIVE, OR CONSEQUENTIAL DAMAGES, HOWEVER CAUSED AND

REGARDLESS OF THE THEORY OF LIABILITY, ARISING OUT OF ANY USE OF THIS DOCUMENT, EVEN IF ARM HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

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

If any of the provisions contained in these terms conflict with any of the provisions of any click through or signed written agreement covering this document with Arm, then the click through or signed written agreement prevails over and supersedes the conflicting provisions of these terms. This document may be translated into other languages for convenience, and you agree that if there is any conflict between the English version of this document and any translation, the terms of the English version of the Agreement shall prevail.

The Arm corporate logo and words marked with ® or ™ are registered trademarks or trademarks of Arm Limited (or its subsidiaries) in the US and/or elsewhere. All rights reserved. Other brands and names mentioned in this document may be the trademarks of their respective owners. Please follow Arm’s trademark usage guidelines at <https://www.arm.com/company/policies/trademarks>.

Copyright © 2020–2021 Arm Limited (or its affiliates). All rights reserved.

Arm Limited. Company 02557590 registered in England.

110 Fulbourn Road, Cambridge, England CB1 9NJ.

(LES-PRE-20349)

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.

Product Status

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

Web address

developer.arm.com

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.....	6
1.1 Conventions.....	6
1.2 Additional reading.....	7
1.3 Feedback.....	7
1.4 Other information.....	8
2 Known issues.....	9
2.1 D17015.....	9
2.2 D17119.....	9
2.3 C17811.....	9
2.4 D17956.....	10
2.5 D18000.....	11
2.6 D18001.....	12
2.7 D18002.....	13
2.8 D18035.....	14
2.9 D18055.....	15
2.10 D18093.....	15
2.11 D18106.....	16
2.12 D18133.....	17
2.13 D18138.....	17
2.14 D18140.....	18
2.15 D18152.....	18
2.16 D18160.....	19
2.17 D18162.....	19
2.18 D18165.....	20
2.19 D18183.....	20
2.20 R18187.....	20
2.21 D18240.....	22
2.22 R18243.....	22

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
<i>italic</i>	Introduces special terminology, denotes cross-references, and citations.
bold	Highlights interface elements, such as menu names. Denotes signal names. Also used for terms in descriptive lists, where appropriate.
monospace	Denotes text that you can enter at the keyboard, such as commands, file and program names, and source code.
<i>monospace italic</i>	Denotes arguments to monospace text where the argument is to be replaced by a specific value.
monospace bold	Denotes language keywords when used outside example code.
monospace <u>underline</u>	Denotes a permitted abbreviation for a command or option. You can enter the underlined text instead of the full command or option name.
<and>	Encloses replaceable terms for assembler syntax where they appear in code or code fragments. For example: <pre>MRC p15, 0, <Rd>, <CRn>, <CRm>, <Opcode_2></pre>
SMALL CAPITALS	Used in body text for a few terms that have specific technical meanings, that are defined in the <i>Arm Glossary</i> . For example, IMPLEMENTATION DEFINED , IMPLEMENTATION SPECIFIC , UNKNOWN , and UNPREDICTABLE .
 Caution	This represents a recommendation which, if not followed, might lead to system failure or damage.
 Warning	This represents a requirement for the system that, if not followed, might result in system failure or damage.
 Danger	This represents a requirement for the system that, if not followed, will result in system failure or damage.

Convention	Use
 Note	This represents an important piece of information that needs your attention.
 Tip	This represents a useful tip that might make it easier, better or faster to perform a task.
 Remember	This is a reminder of something important that relates to the information you are reading.

1.2 Additional reading

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

Table 1-2: Arm publications

Document Name	Document ID	Licensee only
Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile, Issue G.b	DDI 0487G.b	No

1.3 Feedback

Arm welcomes feedback on this product and its documentation.

Feedback on this product

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

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

Feedback on content

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

- The title Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile Known issues in Issue G.b.
- The number 102105_G.b_00_en.
- If applicable, the page number(s) to which your comments refer.
- A concise explanation of your comments.

Arm also welcomes general suggestions for additions and improvements.



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

1.4 Other information

See the Arm website for other relevant information.

- [Arm® Developer](#).
- [Arm® Documentation](#).
- [Technical Support](#)
- [Arm® Glossary](#).

2 Known issues

This document records known issues in the Arm Architecture Reference Manual, Armv8, for Armv8-A architecture profile (DD10487), Issue G.b.

Key

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

2.1 D17015

Details of traps will be added through the use of new LDC and STC accessibility pseudocode in sections G8.3.17 (DBGDTRRXint) and G8.3.18 (DBGDTRTXint). This accessibility pseudocode is the same as for the equivalent MRC and MCR instructions, except that:

- The reported exception syndrome value, if applicable, is $0x06$.
- For LDC instructions the accessibility pseudocode loads the value to be written to the System register from 'MemA[address, 4]', where 'address' is the virtual address calculated by the LDC instruction.

2.2 D17119

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

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

2.3 C17811

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

To ensure correct and portable operation, when software is clearing the valid fields in the register to allow new errors to be recorded, Arm recommends that software:

- Read ERR<n>STATUS and determine which fields need to be cleared to zero.

- Write ones to all the W1C fields that are nonzero in the read value.
- Write zero to all the W1C fields that are zero in the read value.
- Write zero to all the RW fields.

is clarified to read:

To ensure correct and portable operation, when software is clearing the valid fields in the register to allow new errors to be recorded, Arm recommends that software:

- Read ERR<n>STATUS and determine which fields need to be cleared to zero.
- In a single write to ERR<n>STATUS:
 - Write ones to all the W1C fields that are nonzero in the read value.
 - Write zero to all the W1C fields that are zero in the read value.
 - Write zero to all the RW fields.
- Read back ERR<n>STATUS after the write to confirm no new fault has been recorded.

2.4 D17956

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

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

is corrected to read:

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

Within the same section, the AArch64 code that reads:

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

is corrected to read:

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

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

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

is corrected to read:

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

The AArch32 code within the same section that reads:

```
BEQ block_start
```

is enhanced to read:

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

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

2.5 D18000

In section D7.10.3 (Common event numbers), subsection 'Common microarchitectural events', the following events are redefined as counting the accesses made by the hardware prefetcher, rather than the refills:

- 0x8145, L1I_CACHE_HWPRF.
- 0x814D, L2I_CACHE_HWPRF.
- 0x8154, L1D_CACHE_HWPRF.
- 0x8155, L2D_CACHE_HWPRF.
- 0x8156, L3D_CACHE_HWPRF.

For example, '0x8154, L1D_CACHE_HWPRF, Level 1 data cache hardware prefetch' is defined as:

The counter counts each access counted by L1D_CACHE that is not counted by L1D_CACHE_RW or L1D_CACHE_PRFM.

Correspondingly, the events L<n>I_CACHE_REFILL_HWPRF and L<n>D_CACHE_REFILL_HWPRF are defined. For example, L1D_CACHE_REFILL_HWPRF is defined as:

The counter counts each hardware prefetch access counted by L1D_CACHE_HWPRF that causes a refill of the Level 1 data or unified cache from outside the Level 1 data or unified cache of this PE.

Equivalent event definitions are added for the other L<n>D_CACHE and L<n>I_CACHE cache levels.

Within the same subsection, the definitions of the L<n>D_CACHE, L<n>I_CACHE, L<n>D_CACHE_REFILL, and L<n>I_CACHE_REFILL events are redefined to include the hardware prefetcher. For example, the text in '0x0004, L1D_CACHE, Level 1 data cache access' that reads:

When the L1D_CACHE_PRFM and L1D_CACHE_RW events are implemented, accesses to the Level 1 data or unified cache due to a preload or prefetch instruction are counted. Otherwise, it is **IMPLEMENTATION DEFINED** whether accesses to the Level 1 data or unified cache due to a preload or prefetch instruction are counted.

is changed to read:

When the L1D_CACHE_RW event is implemented:

- If the L1D_CACHE_PRFM event is implemented, accesses to the Level 1 data or unified cache due to a preload or prefetch instruction are counted. Otherwise, these are not counted.
- If the L1D_CACHE_HWPRF event is implemented, accesses to the Level 1 data or unified cache due to a hardware prefetcher are counted. Otherwise, these are not counted.

When the L1D_CACHE_RW event is not implemented, it is **IMPLEMENTATION DEFINED** whether accesses to the Level 1 data or unified cache due to a preload or prefetch instructions or due to a hardware prefetcher are counted.

Equivalent changes are made to the other L<n>D_CACHE and L<n>I_CACHE cache access events.

Also within the same subsection, the following text in '0x8140, L1D_CACHE_RW, Level 1 data or unified cache demand access':

The counter counts each access counted by L1D_CACHE that is not counted by L1D_CACHE_PRFM.

is changed to read:

The counter counts each access counted by L1D_CACHE that is due to a demand read or demand write access.

Equivalent changes are made to the other L<n>D_CACHE_RW and L<n>I_CACHE_RD demand cache access events.

2.6 D18001

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

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

is clarified to read:

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

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

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

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

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

■ Data Cache instructions other than DC ZVA.

is clarified to read:

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

2.7 D18002

In section D7.10.3 (Common event numbers), subsection 'Common microarchitectural events', in the definitions of the STALL_FRONTEND_L<n>I and STALL_FRONTEND_MEM events, the text that reads 'demand miss' is clarified to read 'demand instruction miss'. Similarly, in the STALL_BACKEND_L<n>D and STALL_BACKEND_MEM events, the text that reads 'demand miss' is clarified to read 'demand data miss'. For example, in STALL_BACKEND_L2D the text that reads:

■ The counter counts each cycle counted by STALL_BACKEND_MEMBOUND when there is a demand miss in the second level data or unified cache.

is clarified to read:

■ The counter counts each cycle counted by STALL_BACKEND_MEMBOUND when there is a demand data miss in the second level data or unified cache.

In section D8.3.1 (Architected event counters), the text in the definition of the '0x4005, STALL_BACKEND_MEM, Memory stall cycles' AMU event, that reads:

■ The counter counts cycles in which the PE is unable to dispatch instructions from the frontend to the backend of the PE due to a backend stall caused by a miss in the last level of cache within the PE clock domain or, if Armv8.7 is implemented, non-cacheable access in progress.

■ If Armv8.7 is not implemented, it is **IMPLEMENTATION DEFINED** whether the counter counts backend stall cycles when a non-cacheable access is in progress.

is changed to read:

The counter counts cycles in which the PE is unable to dispatch instructions from the frontend to the backend of the PE due to a backend stall caused by a demand data miss in the last level of data or unified cache within the PE clock domain or, if Armv8.7 is implemented, a non-cacheable data access in progress.

If Armv8.7 is not implemented, it is **IMPLEMENTATION DEFINED** whether the counter counts backend stall cycles when a non-cacheable data access is in progress.

In section D7.10.3 (Common event numbers), subsection 'Common microarchitectural events', the STALL_FRONTEND_L<n>I and STALL_BACKEND_L<n>D events are modified such that they are only counted if the corresponding STALL_FRONTEND_L<n+1>I or STALL_BACKEND_L<n+1>D event is not counted. Similarly, these event definitions and the MEM event definitions are also updated, such that: if an LnI or LnD event is an alias for MEM, then the LnI or LnD event is not implemented, and the counter does not count. For example, '0x8165, STALL_BACKEND_L1D, Backend stall cycles, level 1 data cache' is updated to read:

The counter counts each cycle counted by STALL_BACKEND_MEMBOUND when there is a demand data miss in the first level of data or unified cache.

This counter does not count the cycle if any of the following are true:

- The STALL_BACKEND_L2D event is implemented and there is a demand data miss in the second level of data or unified cache, meaning the STALL_BACKEND_L2D event counts the cycle.
- There is a demand data miss in the last level of data or unified cache within the PE clock domain, meaning the STALL_BACKEND_MEM event counts the cycle.

This event is only implemented if the first level of data or unified cache is implemented within the PE clock domain and is not the last level of data or unified cache within the PE clock domain.

2.8 D18035

In section D13.4.9 (PMEVTYPER<n>_ELO, Performance Monitors Event Type Registers, n = 0 - 30), the text in the evtCount field that reads:

If evtCount is programmed to an event that is reserved or not supported by the PE, the behavior depends on the value written:

- For the range 0x0000 to 0x003F, no events are counted, and the value returned by a direct or external read of the evtCount field is the value written to the field.
- If 16-bit evtCount is implemented, for the range 0x4000 to 0x403F, no events are counted, and the value returned by a direct or external read of the evtCount field is the value written to the field.
- For **IMPLEMENTATION DEFINED** events, it is **UNPREDICTABLE** what event, if any, is counted, and the value returned by a direct or external read of the evtCount field is **UNKNOWN**.

is corrected to read:

If `PMEVTYPER<n>_ELO.evtCount` is programmed to an event that is reserved or not supported by the PE, the behavior depends on the value written:

- For the range `0x0000` to `0x003F`, no events are counted and the value returned by a direct or external read of the `PMEVTYPER<n>_ELO.evtCount` field is the value written to the field.
- If `FEAT_PMUv3p1` is implemented, for the range `0x4000` to `0x403F`, no events are counted and the value returned by a direct or external read of the `PMEVTYPER<n>_ELO.evtCount` field is the value written to the field.
- For other values, it is **UNPREDICTABLE** what event, if any, is counted, and the value returned by a direct or external read of the `PMEVTYPER<n>_ELO.evtCount` field is **UNKNOWN**.

Additionally, the text in the same field that reads:

Arm recommends that the behavior across a family of implementations is defined such that if a given implementation does not include an event from a set of common **IMPLEMENTATION DEFINED** events, then no event is counted and the value read back on `evtCount` is the value written.

is replaced by the following text:

Arm recommends that for all values, no events are counted and the value returned by a direct or external read of the `PMEVTYPER<n>_ELO.evtCount` field is the value written to the field.

The equivalent changes are made in sections G8.4.11 (`PMEVTYPER<n>`, Performance Monitors Event Type Registers, $n = 0 - 30$), and I5.3.24 (`PMEVTYPER<n>_ELO`, Performance Monitors Event Type Registers, $n = 0 - 30$).

2.9 D18055

In section D1.12.4 (Synchronous exception prioritization for exceptions taken to AArch64 state), and section G1.12.2 (Exception prioritization for exceptions taken to AArch32 state), subsection 'Synchronous exception prioritization for exceptions taken to AArch32 state', the following bullet point is added to the list of priority 13 cases:

- When `FEAT_FGT` and `FEAT_PMUv3` are implemented, executing an MRS or MSR instruction in AArch64 state, or an MRC or MCR instruction in AArch32 state, that accesses a register associated with an unimplemented event counter.

2.10 D18093

In section J1.1.5 (aarch64/translation), in the pseudocode function `AArch64.S1ApplyOutputPerms()`, the lines that read:

```
if regime == Regime_EL10 && EL2Enabled() && walkparams.nv1 == '1' then
    permissions.ap<2:1> = descriptor<7>:'0';
    permissions.pxn    = descriptor<54>;
```

```
    return permissions;  
if HasUnprivileged(regime) then
```

are corrected to read:

```
if regime == Regime_EL10 && EL2Enabled() && walkparams.nvl == '1' then  
    permissions.ap<2:1> = descriptor<7>:'0';  
    permissions.pxn     = descriptor<54>;  
  
elseif HasUnprivileged(regime) then
```

2.11 D18106

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

If no instruction has been retired since the PE left Debug state, Reset state, or a state where PC Sample-based profiling is prohibited, the sampled value is **UNKNOWN**. If an instruction has been retired but this is the first time the PMPCSR or EDPCSR is read since the PE left Reset state, the sampled value is permitted but not required to return the value `0xFFFFFFFF`.

is relaxed to read:

If no branch instruction has been retired since the PE left Debug state, reset state, or a state where PC Sample-based profiling is prohibited, the sampled value is **UNKNOWN**. If a branch instruction has been retired but this is the first time the PMPCSR or EDPCSR is read since the PE left reset state, the sampled value is permitted but not required to return the value `0xFFFFFFFF`.

Similarly, in section H9.2.32 (EDPCSR, External Debug Program Counter Sample Register), the text in the Bits [31:0] description that reads:

If an instruction has retired since the PE left Reset state, then the first read of EDPCSR[31:0] is permitted but not required to return `0xFFFFFFFF`. EDPCSRlo reads as an **UNKNOWN** value when any of the following are true:

- The PE is in Reset state.
- No instruction has retired since the PE left Reset state, Debug state, or a state where PC Sample-based Profiling is prohibited.
- No instruction has retired since the last read of EDPCSR[31:0].

is relaxed to read:

If a branch instruction has retired since the PE left reset state, then the first read of EDPCSR[31:0] is permitted but not required to return `0xFFFFFFFF`. EDPCSRlo reads as an **UNKNOWN** value when any of the following are true:

- The PE is in reset state.

- No branch instruction has retired since the PE left reset state, Debug state, or a state where PC Sample-based Profiling is prohibited.
- No branch instruction has retired since the last read of EDPCSR[31:0].

The equivalent changes are made in section I5.3.33 (PMPCSR, Program Counter Sample Register).

2.12 D18133

In section D13.8.15 (CNTKCTL_EL1, Counter-timer Kernel Control register) in the definition of EVNTEN, bit [2], the text that reads:

When FEAT_VHE is not implemented, or when HCR_EL2.{E2H, TGE} is not {1, 1}, enables the generation of an event stream from the counter register CNTVCT_ELO.

is clarified to read:

When FEAT_VHE is not implemented, or when HCR_EL2.{E2H, TGE} is not {1, 1}, enables the generation of an event stream from the counter register CNTVCT_ELO as seen from EL1.

All other references to CNTVCT_ELO as part of the event stream in this section are clarified in a similar way.

Similarly, in section D13.8.2 (CNTHCTL_EL2, Counter-timer Hypervisor Control register) for the EVNTEN, bit [2] the text that reads:

Enables the generation of an event stream from the counter register CNTPCT_ELO.

is clarified to read:

Enables the generation of an event stream from the counter register CNTPCT_ELO as seen from EL2.

All other references to CNTPCT_ELO as part of the event stream in this section are clarified in a similar way.

2.13 D18138

In section C7.2.146 (FRECPX), the text that reads:

This instruction finds an approximate reciprocal exponent for each vector element in the source SIMD&FP register, places the result in a vector, and writes the vector to the destination SIMD&FP register.

is replaced by the following text:

This instruction finds an approximate reciprocal exponent for the source SIMD&FP register and writes the result to the destination SIMD&FP register.

2.14 D18140

In sections B2.3.8 (Ordering of instruction fetches) and E2.3.8 (Ordering of instruction fetches), the text that reads:

For two memory locations A and B, if A has been written to and been made coherent with the instruction fetches of the shareability domain, before an update to B by an observer in the same shareability domain, then the instruction stream of each observer in the shareability domain will not see the updated value of B without also seeing the updated value of A.

is corrected to read:

For two memory locations A and B:

If A has been written to with an updated value and been made coherent with the instruction fetches of the shareability domain, before B has been written to with an updated value by an observer in the same shareability domain, then where, for an observer in the shareability domain, an instruction read from B appears in program order before an instruction fetched from A, if the instruction read from B contains the updated value of B then the instruction read from A appearing later in program order will contain the updated value of A.

2.15 D18152

In section D7.5.2 (Freezing event counters), in the bullet list that is introduced by the paragraph 'When FEAT_PMUv3p7 is implemented, the PMU can be configured to freeze event counters...', the bullet point that reads:

- If EL2 is implemented, MDCR_EL2.HPMN is less than PMCR_ELO.N and n is in the range [MDCR_EL2.HPMN .. (PMCR_ELO.N-1)], when PMOVSLR_ELO[(PMCR_ELO.N-1):MDCR_EL2.HPMN] is non-zero, indicating an unsigned overflow in one of the event counters in the range, event counter n does not count when PMCR_ELO.FZO is 1.

is corrected to read:

- If EL2 is implemented, MDCR_EL2.HPMN is less than PMCR_ELO.N and n is in the range [MDCR_EL2.HPMN .. (PMCR_ELO.N-1)], when PMOVSLR_ELO[(PMCR_ELO.N-1):MDCR_EL2.HPMN] is non-zero, indicating an unsigned overflow in one of the event counters in the range, event counter n does not count when MDCR_EL2.HPMFZO is 1.

2.16 D18160

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

```
// PMCR_EL0.DP disables the cycle counter when event counting is prohibited
if enabled && prohibited && n == 31 then
    enabled = PMCR_EL0.DP == '0';
```

is corrected to read:

```
// PMCR_EL0.DP disables the cycle counter when event counting is prohibited
if prohibited && n == 31 then
    enabled = enabled && PMCR_EL0.DP == '0';
    prohibited = FALSE; // Otherwise whether event counting is prohibited does
    not affect the cycle counter
```

A similar correction is made in section J1.2.1 (aarch32/debug) to `AArch32.CountEvents()`.

2.17 D18162

In section D10.2.6 (Events packet), subsection 'Events packet payload', the text in 'E[17], byte 2 bit [17], when `SZ == 0b10`, or `SZ == 0b11`' that reads:

If PMUv3 and The Scalable Vector Extension (SVE) are implemented this Event is required to be implemented consistently with `SVE_PRED_EMPTY_SPEC` and `SVE_PRED_PARTIAL_SPEC` in the Arm Architecture Reference Manual Supplement, the Scalable Vector Extension, for v8-A.

is corrected to read:

If PMUv3 and The Scalable Vector Extension (SVE) are implemented, this Event is required to be implemented consistently with `<xref: SVE_PRED_NOT_FULL_SPEC>`.

Similarly, the text in 'E[18], byte 2 bit [18], when `SZ == 0b10`, or `SZ == 0b11`' that reads:

If PMUv3 and The Scalable Vector Extension (SVE) are implemented this Event is required to be implemented consistently with `SVE_PRED_EMPTY_SPEC` in the Arm Architecture Reference Manual Supplement, the Scalable Vector Extension, for v8-A.

is clarified to read:

If PMUv3 and The Scalable Vector Extension (SVE) are implemented, this Event is required to be implemented consistently with `<xref: SVE_PRED_EMPTY_SPEC>`.

2.18 D18165

In section D5.9.1 (Use of ASIDs and VMIDs to reduce TLB maintenance requirements), subsection 'VMID size', the line that reads:

When the value of VTCR_EL2.VS is 0, VMID[63:56]:

is corrected to read:

When the value of VTCR_EL2.VS is 0, VTTBR_EL2[63:56]:

2.19 D18183

In section C6.2.79 (DGH), the description that reads:

DGH is a hint instruction. A DGH instruction is not expected to be performance optimal to merge memory accesses with Normal Non-cacheable or Device-GRE attributes appearing in program order before the hint instruction with any memory accesses appearing after the hint instruction into a single memory transaction on an interconnect.

is updated to read:

Data Gathering Hint is a hint instruction that indicates that it is not expected to be performance optimal to merge memory accesses with Normal Non-cacheable or Device-GRE attributes appearing in program order before the hint instruction with any memory accesses appearing after the hint instruction into a single memory transaction on an interconnect.

2.20 R18187

In section I5.3.14 (PMCID2SR, CONTEXTIDR_EL2 Sample Register), in the description of CONTEXTIDR_EL2, bits [31:0], the text that reads:

When the most recent PMPCSR sample was generated:

- If EL2 is using AArch64, then this field is set to the Context ID sampled from CONTEXTIDR_EL2.
- If EL2 is using AArch32, then this field is set to an **UNKNOWN** value.

is changed to read:

When the most recent PMPCSR sample was generated:

- If EL2 is enabled in the current Security state, using AArch64, and the PE was not executing at EL3, then this field is set to the Context ID sampled from CONTEXTIDR_EL2.
- Otherwise, this field is set to an **UNKNOWN** value.

Similarly, the text in section H9.2.43 (EDVIDSR, External Debug Virtual Context Sample Register), in the description of CONTEXTIDR_EL2, bits [31:0], that reads:

When the most recent EDPCSR sample was generated:

- If EL2 was using AArch64 and the PE was executing in Non-secure state, then this field is set to the Context ID sampled from CONTEXTIDR_EL2.
- If EL2 was using AArch32 or the PE was executing in Secure state, then this field is set to an **UNKNOWN** value.

is changed to:

When the most recent EDPCSR sample was generated:

- If EL2 is enabled in the current Security state, using AArch64, and the PE was not executing at EL3, then this field is set to the Context ID sampled from CONTEXTIDR_EL2.
- Otherwise, this field is set to an **UNKNOWN** value.

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

```
pc_sample.has_el2 = EL2Enabled();
if EL2Enabled() then
    ...
```

is changed to read:

```
pc_sample.has_el2 = PSTATE.EL != EL3 && EL2Enabled();
if pc_sample.has_el2 then
    ...
```

Also in section J1.3.1 (shared/debug), within the pseudocode function EDPCSRlo[], the code that reads:

```
if (HaveVirtHostExt() || HaveV82Debug()) && EDSCR.SC2 == '1' then
    EDVIDSR = (if HaveEL(EL2) && pc_sample.ns == '1' then pc_sample.contextidr_el2
               else bits(32) UNKNOWN);
else
    if HaveEL(EL2) && pc_sample.ns == '1' && pc_sample.el IN {EL1,EL0} then
        EDVIDSR.VMID = pc_sample.vmid;
    else
        EDVIDSR.VMID = Zeros();
```

is simplified to read:

```
if (HaveVirtHostExt() || HaveV82Debug()) && EDSCR.SC2 == '1' then
    EDVIDSR = (if pc_sample.has_el2 then pc_sample.contextidr_el2 else bits(32) UN\
KNOWN);
else
    EDVIDSR.VMID = (if pc_sample.has_el2 && pc_sample.el IN {EL1,EL0} then pc_sam\
ple.vmid else Zeros());
```

2.21 D18240

In section G8.2.75 (HTCR, Hyp Translation Control Register), the description in HWU62, bit [28] that currently reads:

Otherwise:

Reserved, **RES0**.

is corrected to read:

Otherwise:

Reserved, **RAZ/WI**.

Equivalent changes are made in HWU61, bit [27], HWU60, bit [26], and HWU59, bit [25].

2.22 R18243

In section D13.2.48 (HCR_EL2, Hypervisor Configuration Register), the following text is added to the description of DCT, bit [57]:

This field is permitted to be cached in a TLB.