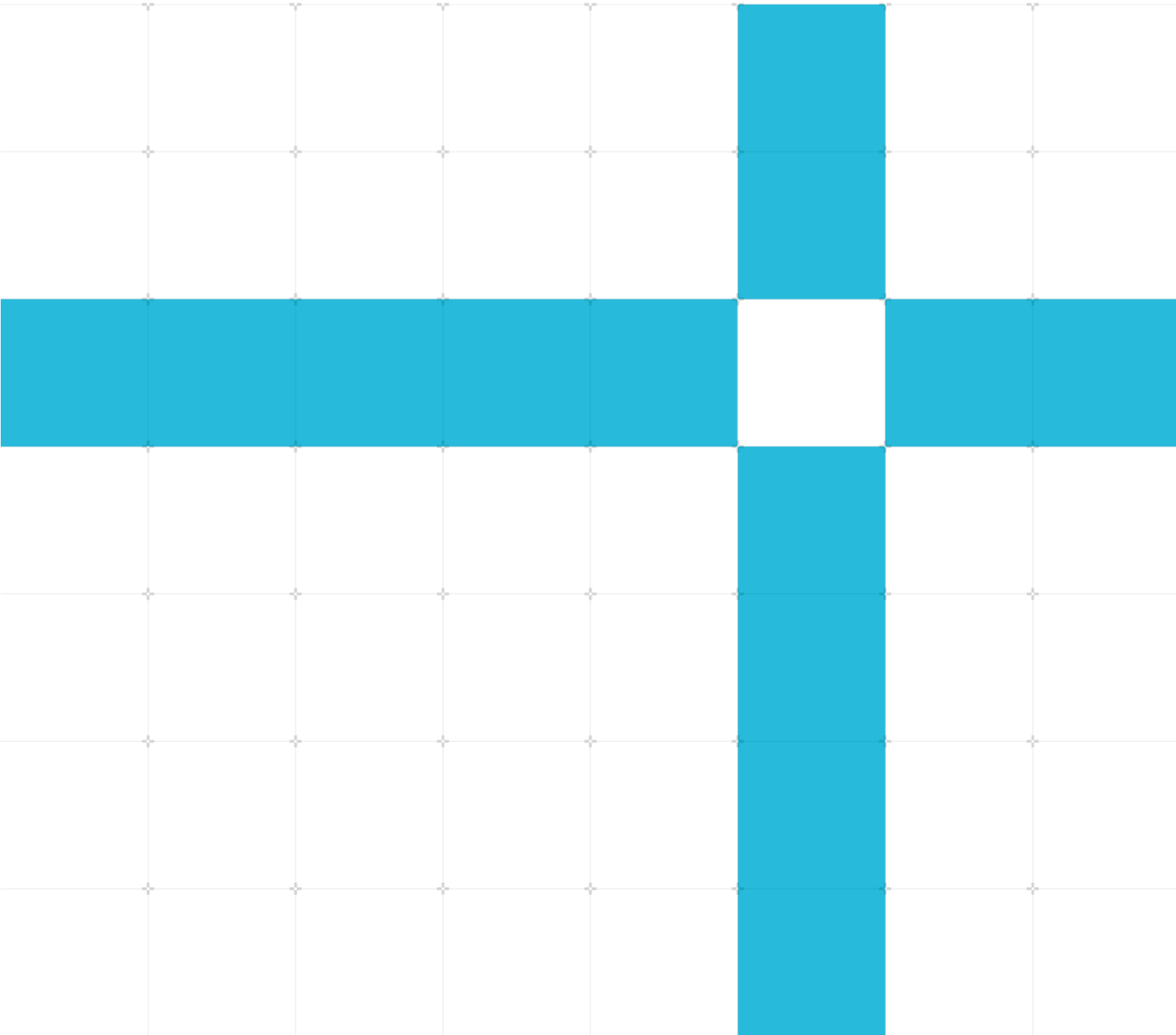
Arm® Neoverse™ N1

Revision: r4p1

PMU Guide

Non-Confidential

Issue 2.0
PJDOC-466751330-547673



Arm® Neoverse™ N1

PMU Guide

Release information

Document history

Issue	Date	Confidentiality	Change
1.0	May 7. 2021	Non-Confidential	Initial public release
2.0	May 25, 2022	Non-Confidential	Updated release for new referenced documentation

Non-Confidential Proprietary Notice

(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.

Unrestricted Access is an Arm internal classification.

Product Status

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

Web Address

developer.arm.com

Progressive terminology commitment

We believe that this document contains no offensive terms. If you find offensive terms in this document, please email **terms@arm.com**.

Contents

1 Introduction

1.1 Product revision status

The *rxpy* identifier indicates the revision status of the product described in this book, for example, r1p2, where:

rx
: Identifies the major revision of the product, for example, r1.

Py
: Identifies the minor revision or modification status of the product, for example, p2.

1.2 Intended audience

This document is intended for software developers running code on the Neoverse N1.

1.3 Conventions

The following subsections describe conventions used in Arm documents.

1.3.1 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: **https://developer.arm.com/glossary**.

1.3.2 Typographical conventions

Convention	Use
italic	Introduces citations.
bold	Highlights interface elements, such as menu names. Denotes signal names. Also used for terms in descriptive lists, where appropriate.
`monospace`	Denotes text that you can enter at the keyboard, such as commands, file and program names, and source code.
`monospace` **`bold`**	Denotes language keywords when used outside example code.
`monospace underline`	Denotes a permitted abbreviation for a command or option. You can enter the underlined text instead of the full command or option name.
<and>	Encloses replaceable terms for assembler syntax where they appear in code or code fragments. For example: `MRC p15, 0, <Rd>, <CRn>, <CRm>, <Opcode_2>`
SMALL CAPITALS	Used in body text for a few terms that have specific technical meanings, that are defined in the Arm® Glossary. For example, IMPLEMENTATION DEFINED, IMPLEMENTATION SPECIFIC, UNKNOWN, and UNPREDICTABLE.
	This represents a recommendation which, if not followed, might lead to system failure or damage.
	This represents a requirement for the system that, if not followed, might result in system failure or damage.
	This represents a requirement for the system that, if not followed, will result in system failure or damage.
	This represents an important piece of information that needs your attention.
	This represents a useful tip that might make it easier, better or faster to perform a task.
	This is a reminder of something important that relates to the information you are reading.

1.4 Additional reading

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

Table 1-1 Arm publications

Document name	Document ID	Licensee only
Arm® Architecture Reference Manual for A-profile architecture	DDI 0487H.a (ID020222)	No
Arm® Neoverse™ N1 Core Revision: r4p1 Technical Reference Manual	100616	No
Arm® DynamIQ™ Shared Unit Revision: r4p1 Technical Reference Manual	100453	No
Arm® Neoverse™ N1 Software Optimization Guide	PJDOC-466751330-9707	No
Arm Neoverse N1 (MP050) Software Developer Errata Notice	SDEN-885747	No
Arm Cortex-A Series Programmers Guide for Armv8-A	DEN0024A	No

1.5 Feedback

Arm welcomes feedback on this product and its documentation.

1.5.1 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.

1.5.2 Feedback on content

If you have comments on content, send an email to errata@arm.com and give:

- The title [Product Name] [Document Title].
- The number [Document ID].
- 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.

2 Overview

This document describes the behavior of the different *Performance Monitor Unit* (PMU) events implemented in the Neoverse N1.

The Neoverse N1 has six programmable 32-bit counters (counters 0-5), and each individual counter can be programmed to count when one of the PMU events described in this document occurs.

2.1 Scope

This document provides high level descriptions of Neoverse N1 PMU events. There are references to both architectural behavior and Neoverse N1 micro-architectural behavior that clarify those event descriptions. For more complete descriptions of the Arm Architecture, please refer to the *Arm® Architecture Reference Manual Armv8-A*. For more detailed descriptions of the Neoverse N1, please refer to the *Arm® Neoverse™ N1 Technical Reference Manual*.

This document does not discuss using software development tools or a performance analysis program (such as Linux perf), to program the Neoverse N1 PMUs.

Certain PMU events may be discussed in the *Neoverse N1 Software Developer Errata Notice* (SDEN). Users are encouraged to check that document for information about events that they are using.

3 Architecture and micro-architecture definitions

This section provides additional information regarding relevant areas of the Arm Architecture and details of the Neoverse N1 micro-architecture. This section covers architectural and micro-architectural functional areas that affect the behavior of the different PMU events implemented in the Neoverse N1.

Please note that this section is not intended to be a complete guide for either the architectural or micro-architectural behavior of the Neoverse N1. For a more complete overview, please reference:

- *Arm Cortex-A Series Programmers Guide for Armv8-A*,
- A-Profile architecture guides available on https://Developer.Arm.com.
- The *Definitions* section of the PMU Event chapter in the *Arm® Architecture Reference Manual*

3.1 Arm Architecture definitions

The *Glossary* section of the *Arm® Architecture Reference Manual Armv8-A* contains definitions for different architectural terms used for PMU event descriptions. This section provides additional explanations for some of those terms (particularly ones that apply to the Neoverse N1).

Please note that in all cases, the actual specifications in the *Arm® Architecture Reference Manual Armv8-A* should be used.

3.1.1 Attributability

Some event descriptions reference the term "attributability", which is defined in the PMU section of the *Arm® Architecture Reference Manual Armv8-A*. Usually, that term refers to whether or not an event can be attributed to a single *Processing element* (PE). Attributability can mean two things in this context:

- Attributable to a hardware thread in a simultaneous multithreading (SMT) CPU. Since the Neoverse N1 is not multi-threaded, attributable used in this sense is not applicable.
- Attributable to a particular CPU in a multi-CPU cluster or system. Where the term applies to other CPUs in the system, it is specifically addressed in the PMU event description.

3.1.2 PMU Version

The Neoverse N1 implements PMUv3 for Armv8.1. That information is specified in the PMUVer bits in the ID_AA64DFR0_EL1 register.

3.1.3 Speculatively executed versus architecturally executed

The Arm Architecture makes a distinction between an instruction which is speculatively executed versus an instruction which is architecturally executed. For example, instructions following a branch instruction (where the branch condition has been predicted by the CPU) are considered speculatively executed until the branch is resolved. Instructions could also be abandoned if an interrupt or exception from a previously executed instruction occur. Architecturally executed instructions update the architectural state of the CPU when they complete.

If the branch is mispredicted, and the instructions are speculatively executed, they will not be considered architecturally executed. The *Arm® Architecture Reference Manual Armv8-A* also refers to architecturally executed instructions as “retired” or “committed”. Speculatively executed instructions that are not architecturally executed will be abandoned; that is, their results will be discarded and not counted as part of the program flow.

An instruction is considered architecturally executed when that instruction is found to be on the correct execution path of the program flow. While the Neoverse N1 CPU can execute instructions out of order, architecturally executed instructions are always resolved in program order. Please see the Out of Order Execution section below.

Many PMU events measure speculatively executed operations. The *Arm® Architecture Reference Manual Armv8-A* says "The definition of speculatively executed does not mean only those operations that are executed speculatively and later abandoned, for example due to a branch misprediction or fault. That is, speculatively executed operations must count operations on both false and correct execution paths."

That definition means that events that count speculatively executed instructions will count instructions that were architecturally executed as well as instructions that were not architecturally executed.

For more information, please read the definitions of “Speculative” and “Architecturally executed” in the Glossary section of the *Arm® Architecture Reference Manual Armv8-A*.

3.1.4 Taken locally

The *Arm® Architecture Reference Manual Armv8-A* glossary defines "taken locally" as an exception taken without being virtualized; in effect, the exception is taken by the host kernel. For exceptions to be considered taken locally, one of the following conditions must apply:

- The kernel is running in EL1
- The kernel is running in EL2 and the host virtualization extensions are enabled (HCR_EL2.E2H and TGE are both set to 1)

Please note that some of the exception events do not use that definition. For example, an HVC exception would normally be taken locally in EL2 (since the hypervisor runs in EL2), and an SMC exception would be taken in EL3.

Exceptions caused by speculatively executed instructions or speculative memory accesses will not be taken until the instruction that caused that exception condition is architecturally executed.

3.1.5 Aligned/unaligned memory access

Memory accesses are aligned if the address for the access is a multiple of the data size. For example, a word (32-bit access) is aligned if the address ends in 0x00, 0x04, 0x08, or 0x0C. If the address is not on an aligned boundary, then it is unaligned. For example, a word access that has an address ending in 0x01, 0x02, 0x03, 0x05, 0x06, is unaligned.

While the CPU can process data side accesses to unaligned addresses, the CPU issue aligned accesses to the memory system. For example, with a 32-bit access located at an address ending in 0x01, the CPU would issue a larger access, such as a 64-bit access starting at the address ending in 0x00.

Please note that if an access is cacheable, and the data is not present in the cache, then the entire cache line containing that data may be brought into the cache. Cache lines are also aligned to the cache line size.

Cache line size is discussed in section 3.2.6, Cache Line Sizes.

3.2 Neoverse N1 micro-architecture information

The Neoverse N1 implements the Armv8-A architecture and the Armv8.2-A architecture extension. However, there are a number of processor features such as pipeline and caches that are implementation specific. This section defines those features and behaviors.

Please note that Arm documentation may refer to CPUs as “cores”. This document refers to the Neoverse N1 CPU, but that term can be used interchangeably with the term “core”.

3.2.1 CPU and DynamIQ shared unit configuration

The Neoverse N1 is required to have the *DynamIQ Shared Unit* (DSU) as an interface between the CPU and the external interconnect. The DSU is a separate piece of logic, and contains all external interfaces for the Neoverse N1, including the bus interface, the power management interface, the interrupt controller interface, as well as all power and clocking interfaces.

For certain configurations, the DSU also contains an optional L3 cache, as well as a (required) *Snoop Control Unit* (SCU) for tracking coherency requests inside the DSU cluster. For the L3 and SCU configuration, the DSU contains its own set of PMU counters and events.

Designers can implement the DSU in two different ways with the Neoverse N1:

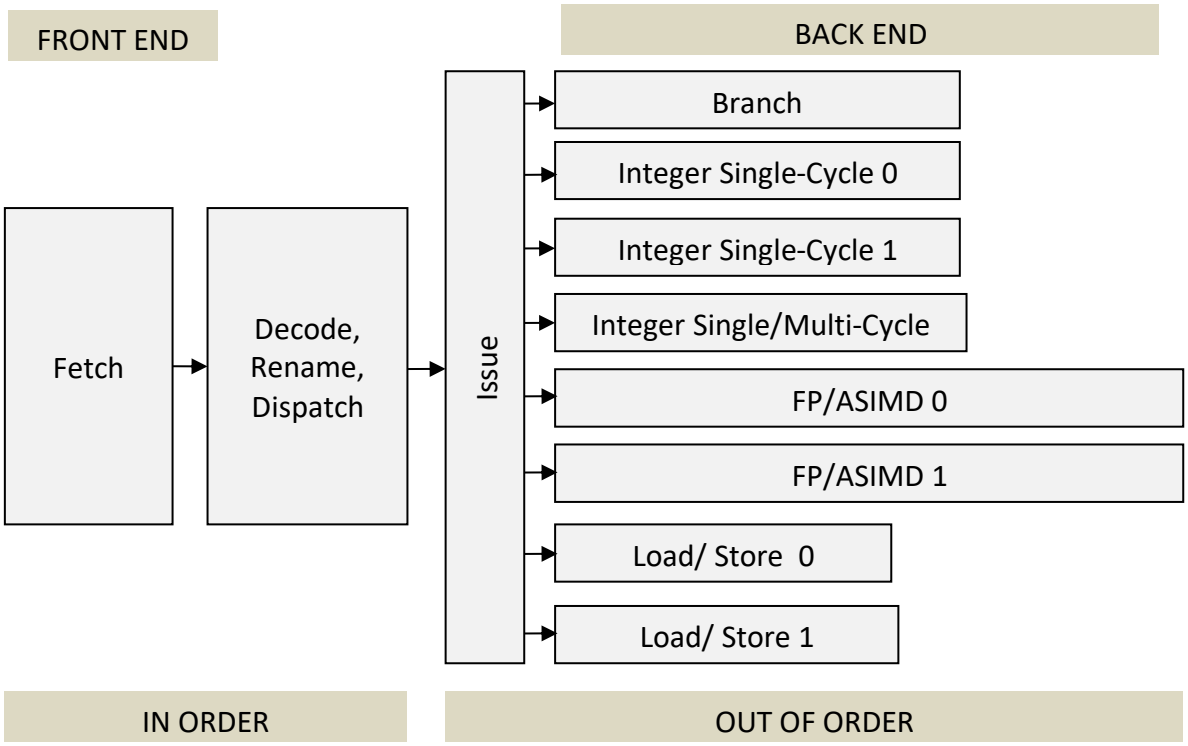
- Direct Connect: In this implementation, there is a single Neoverse N1 inside the DSU, and the DSU does not have an L3 cache or SCU. Memory transactions to and from the Neoverse N1 pass directly through the DSU wrapper. This configuration requires a special connection in the associated *Coherent Mesh Network* (CMN) CHI-based interconnect. Most production Neoverse N1 designs use this implementation.
- Multiple CPUs: In this implementation, up to four Neoverse N1 CPUs are contained in a single cluster with one DSU, with SCU logic and one optional shared L3 cache. This implementation is used on the Arm Neoverse N1SDP development platform, which has two clusters each with two Neoverse N1 CPUs.

For more information on the DSU, please see the *Arm® DynamIQ™ Shared Unit Technical Reference Manual*.

3.2.2 Pipeline and operations

The following diagram shows the high-level Neoverse N1 instruction processing pipeline.

Figure 1:



The Neoverse N1 pipeline fetches instructions which (after some internal decoding) proceed through the pipeline register renaming and dispatch stages. Those decoded instructions could be split further into two micro-operations (uops) at dispatch stage. Once dispatched, uops wait for their operands and issue out-of-order to one of eight execution pipelines. There are multiple issue queues in the issue stage, but each execution pipeline can accept and complete one uop per cycle.

Please note that while some less complex instructions (for example, ADD or MOV) may be broken down into a single micro-operation, other instructions may be broken down into multiple micro-operations. Arm does not publish the list of micro-operations.

For PMU event definitions, some events specifically count instructions, while other events count micro-operations (which are referred to as operations). Please be aware of the use of the word "operations" or "instructions" in the event description.

3.2.3 Out of order execution

The Neoverse N1 pipeline can issue and speculatively execute instructions out of order. As long as there is not a data dependency between different instructions, they can speculatively execute out of order and save their results. That allows the pipeline to issue instructions to the different back end pipelines so that the pipeline is executing as many instructions as possible.

Those speculatively executed instructions can be committed or resolved to being architecturally executed. Architecturally executed instructions are always resolved in program order (what the actual software says).

Because of changes in program control flow, speculatively executed instructions may not be architecturally executed. For example, if instructions were speculatively executed after a mispredicted branch instruction, those speculatively executed instructions would be abandoned. Instructions could also be abandoned if an exception is taken. When that occurs, the CPU will determine in the program flow where the last architecturally instruction was, and then the remaining instructions will be abandoned (even if some of them have already speculatively executed). Those speculatively executed (but abandoned) instructions could be executed again following the return from exception.

Memory load instructions (for normal memory) can also be executed speculatively. By architectural definition, read accesses to normal memory can be repeated. If the CPU issues a memory load operation that is later abandoned, memory related PMU events may count (if the actual memory access completed). The specific PMU event descriptions will give those conditions.

3.2.4 Architecturally defined events

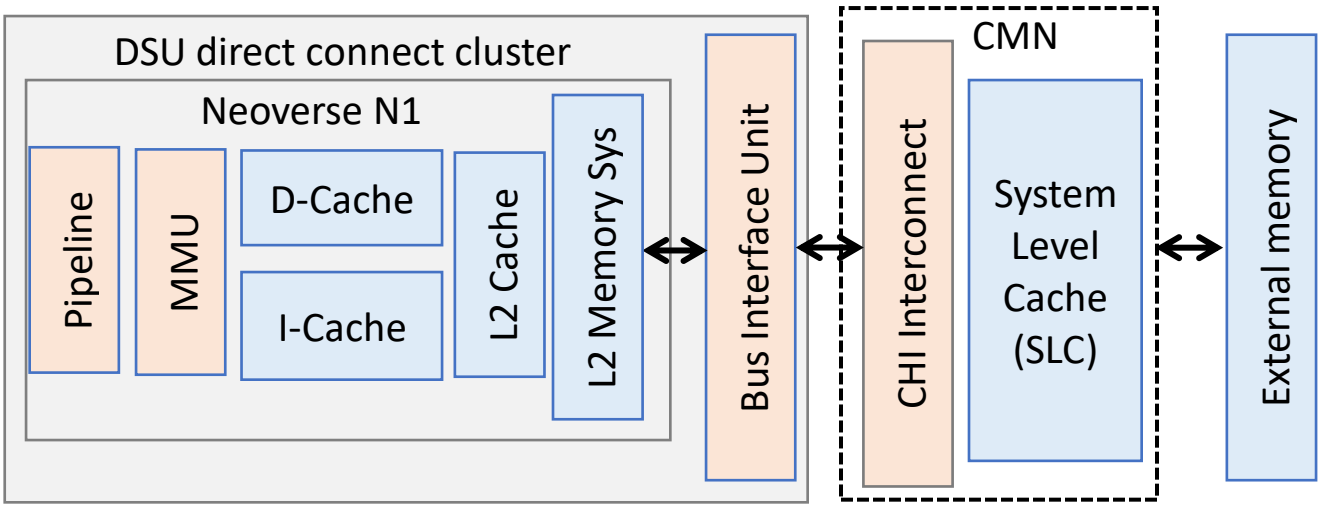
Most of the PMU events listed in this guide are architecturally defined, and are listed in the *Performance Monitors Extension* section of the *Arm® Architecture Reference Manual Armv8-A*. However, some architecturally defined events are not implemented on the Neoverse N1.

3.2.5 Cache architecture

The Neoverse N1 implements separate instruction (I-side) and data (D-side) Level 1 caches, and a unified L2 cache. The Neoverse N1 caches are:

- Allocate on read/write: any cacheable memory access will attempt to allocate a cache line inside the Neoverse N1 caches.
- Write-back caches: data written to the caches will not update external memory unless the cache line is evicted or there is an explicit request by a cache maintenance operation.

Figure 2:



3.2.6 Cache line sizes

Cache lines in the Neoverse N1 and other v8-A CPUs are 64 bytes (16 words) long. Whenever data is allocated into a cache, or written back or evicted from a cache, the full cache line will be read in or written out. While individual 32-bit words or 64-bit double words may be read into a cache first (those are known as critical words – the direct word that a load or store instruction specifies), the entire cache line around that word will be allocated into the cache. The CPU will be able to access the critical word first before the rest of the cache line has been read in.

The Neoverse N1 caches are set associative caches; that is, there are multiple ways of cache lines in the cache where a particular address could be stored. Both L1 caches are 4-way set associative caches, and the L2 cache is 8 way set associative. When the PMU event descriptions use the term “full” with respect to a cache, the particular way where a line could be stored is full.

For example, out of the full 64K cache, an L1 cache could have only 4 cache lines allocated to it. If all 4 cache lines are in the same set, and another cache line that would also be allocated to that set needs to be allocated in the cache, one of those 4 lines will need to be evicted.

3.2.7 Data side cache allocation

The L1 data cache is strongly inclusive with respect to the L2 unified cache. Strongly inclusive means that any cache line present in the L1 data cache will also be present in the L2 unified cache. However, the L2 cache may also contain cache lines that are not present in the L1 data cache.

3.2.8 Instruction side cache allocation

The L1 instruction cache is weakly inclusive with respect to the L2 unified cache. Weakly inclusive means that a cache line will initially be allocated into the L1 instruction cache and the L2 cache but can later be evicted from the L2 cache.

3.2.9 DSU L3 cache allocation

If the Neoverse N1 is in a system with a DSU with an L3 cache, the L3 cache allocates cache lines in the following manner:

- If a cache line is present in only one CPU inside the DSU cluster, then the L3 uses exclusive allocation; that is, the cache line is not allocated in the L3 until it is evicted from the CPU that contained that cache line.
- If the cache line is present in more than one CPU inside the DSU cluster, then the L3 uses inclusive allocation; that is, the line is allocated in the L3 cache if the line is shared by multiple CPUs.

3.2.10 Cache terminology and behavior

A data cache line is considered “clean” if it has not been modified after loading into the cache and is considered “dirty” if the data in the cache line has been changed. If a cache is full, and a new cache line needs to allocate, then an existing cache line will be evicted – the term used in PMU events is “refill”.

3.2.11 Cache Maintenance Operations

The *Arm® Architecture Reference Manual Armv8-A* defines a series of instructions for *Cache Maintenance Operations* (CMOs). Examples of those operations include forcing a data writeback to external memory, and invalidating (emptying) cache lines. In some cases, cache behavior based on those operations is not counted by PMU events. Where behavior affected by CMOs applies to cache related PMU events, it is specifically mentioned in the PMU event description. Please note that there are no cache maintenance operations that can force cache allocation; this applies to all PMU events relating to cache refill.

3.2.12 Cache coherency

The Neoverse N1 cache logic and the cache coherent interconnect automatically maintain coherency among any caches for any memory marked as normal, cacheable, and inner sharable.

For an overview about memory types, please consult the Learn The Architecture Guides on the https://Developer.Arm.com website.

Cache coherency operates by:

- "Snooping" across different caches in the system in case a memory access misses in the local CPU cache.
- Invalidating other copies of cached data in case one CPU writes to an existing cache line .

Please note that on the Neoverse N1, I-caches can be configured as coherent. I-cache coherency works by keeping the I-cache strongly inclusive with the unified L2 cache. If the CPU updates code/instructions (for example, through some sort of just-in-time compiler) that would cause a data write with the new instructions. That write would behave like any coherent data write and invalidate copies of that data in any other caches in the system. That invalidation would automatically invalidate copies of that cache line present in any instruction caches, so new instruction fetches would have to snoop for the updated (correct) copy.

Note that instruction fetches that miss in the I-cache will look in the D-cache and the L2 cache first.

The CTR_EL0.DIC bit will be set to 1 if the I-cache is coherent.

3.2.13 L2 cache and memory interface interaction

The external interface on an Neoverse N1 is referred to as the L2 memory system; it provides an interface between the CPU and the CHI bus interface in the DSU. The Neoverse N1 has no direct interface to the interconnect and external memory system, and every memory access passes from the Neoverse N1 to the DSU's CHI bus interface. The L2 interface is also referred to as the load/store unit and should not be confused with the load/store portions of the Neoverse N1 pipeline. The load/store portions of the pipeline process memory operations and then issues memory commands to the L2 memory system. The L2 memory system issues the actual transactions to the CHI interface. Figure 2 above illustrates the different memory interfaces for a Neoverse N1 system.

3.2.14 Cache lookup

Each cacheable memory access (after translation by the memory management unit) attempts to look up in the L1 cache (the D-cache for data loads or stores, or the I-cache for instruction fetches). If that cache line is not present in the L1 cache, then the Neoverse N1 will attempt to look up in the L2 unified cache. If the cache line is present in the L2 unified cache, it will allocate (PMU event descriptions use the term "refill") in the L1 cache. Allocation can also potentially evict an already existing cache line.

If the lookup misses in the L2 cache, then the Neoverse N1 L2 memory system attempts to look up in the next level of cache. That next level can either be the L3 cache in the DSU (when present), an L2 cache in a different Neoverse N1, or the system level cache in the external coherent interconnect (the *Coherent Mesh Network*, or CMN). PMU events that reference accesses to the next level cache describe how the CPU determines what that next level is.

3.2.15 Cache eviction

When a cache set is full and a new line needs to be allocated into the cache, there is a victim counter that selects which cache line is next to be evicted. As discussed in the Cache Line Size section, the term full in the relevant PMU event descriptions refers to the particular cache set where an address can be allocated.

3.2.16 Unaligned accesses

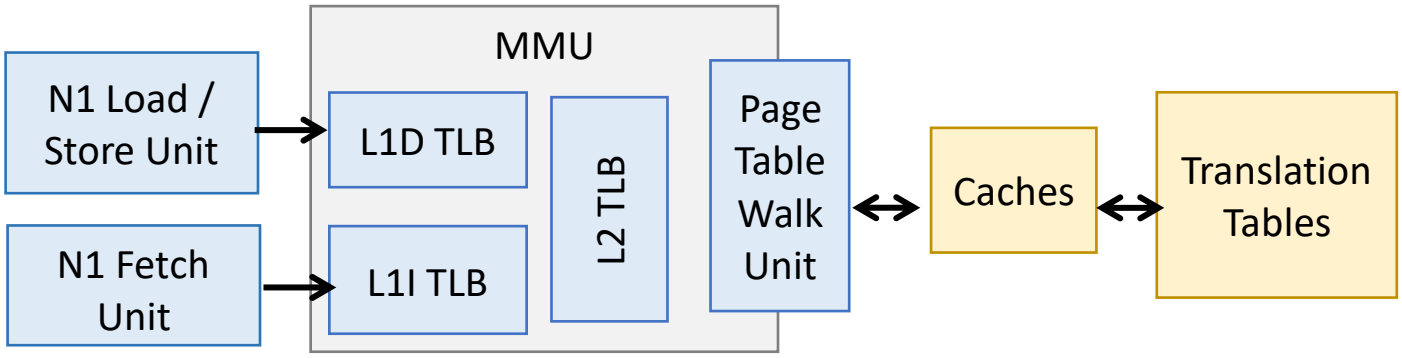
The Neoverse N1 may execute a data memory operation on an unaligned address. The actual external memory transaction issued by the CPU unit to the bus will be aligned. The CPU will execute a series of aligned accesses to bring the requested (unaligned address) data in and pass it back to the load/store pipeline.

For example, if the Neoverse N1 executed a load instruction for a 32-bit word at address 0x8001, it could issue a 64-bit read from address 0x8000, and then pull the requested 32-bit word from that full read.

3.2.17 Memory Management Unit behavior

All memory accesses will first go through the *Memory Management Unit* (MMU) for virtual to physical address translation, as well as to assign memory attributes to the memory transaction. Memory translation is defined with a series of memory page tables that live in actual memory and are programmed by the application. Note that the MMU can issue memory transactions itself when accessing page tables.

Figure 3:



3.2.18 TLB behavior

Existing memory translations are cached in *Translation Look-aside Buffers* (TLBs). TLBs function as small caches for memory translations, and work similarly to normal data caches (hits, misses, allocation/refill, etc.).

The Neoverse N1 has two levels of TLBs:

- L1 I-side and D-side TLBs (each fully associative with 48 entries).

- L2 unified TLB (5 way set associative with 1280 entries).

Operations that access memory cause the CPU MMU to look up the virtual to physical translation in one of the L1 TLBs (depending whether it is an instruction fetch or a data side read or write). If the attempted L1 access misses, the MMU will look up in the L2 TLB. If the attempted L2 access misses, then the MMU will do a page table walk.

Misses will allocate a new entry in the L2 TLB, which will then forward to the L1 TLB (which will also allocate a new entry). The L2 TLB also functions as a walk cache; it stores partial page table walk entries. For example, a 2nd stage translation that could be used by multiple 1st stage translations could allocate into the L2 TLB.

Please note that while events which cover the L2 TLB behavior are named L2D_<event>, the L2 TLB is a unified TLB, and contains both data side and instruction side translations.

3.2.19 TLB maintenance operations

The *Arm® Architecture Reference Manual Armv8* defines a series of instructions for TLB maintenance. These operations are used to invalidate TLB entries when the associated MMU mapping has changed. In some cases, TLB behavior based on those operations is not counted by PMU events. Where behavior that can be caused by TLB maintenance operations applies to TLB related PMU events, it is specifically mentioned in the PMU event description.

Please note that there are no TLB maintenance operations that can force TLB allocation.

3.2.20 Memory error behavior

The Neoverse N1 implements ECC and parity for some internal memory structures. Behavior, control, and reporting registers are part of the *Reliability, Availability, and Serviceability (RAS) Architectural Extension.*

Accesses to those memories will check the error status and respond in the following ways:

- The L1 instruction cache is protected by parity checking. If an error is detected, then the internal RAS counters are updated. The cache line is then invalidated so that the correct cache line can be fetched.
- The L1 data cache and L2 unified cache are protected with ECC. If a 1-bit error is detected, it is corrected, and internal counters are updated.
- If a 2-bit (uncorrectable) error is detected, and the memory access has been architecturally executed, that error is considered “consumed”. The CPU will take a synchronous error exception or an SError exception and update internal memory error registers.
- If a 2-bit (uncorrectable) error is detected, and the memory access has been speculatively executed but not architecturally committed, the CPU will mark the cache line as “poisoned”. Poisoning defers the error response until some other device in the system consumes that error. If a CPU that consumes the poisoned cache line, then it will take a synchronous error exception or an SError exception and update internal memory error registers.

Note: on the Neoverse N1, a speculatively executed instruction or memory access cannot directly cause an exception until that instruction or access is architecturally committed.

Please note:

- RAS register counts are separate from any related PMU event that counts memory errors.
- The MEMORY_ERROR PMU event does not count L2 cache memory errors.

3.2.21 Coherent Mesh Network configuration

In almost all production systems, the Neoverse N1 connects to a memory interconnect based on the AMBA *Coherent Hub Interconnect* (CHI) protocol. Arm's implementations of that protocol are the *Coherent Mesh Network* (CMN) products. Depending on the version or topology, large numbers of Neoverse N1 CPUs can be connected to CMN.

CMN has a built-in *System Level Cache* (SLC), which functions as a next level cache for all connected CPUs. The SLC allocates based on eviction from either:

- The L2 cache of a connected Neoverse N1 CPU in direct connect mode.
- The L3 cache in the DSU.

It is also possible to "stash" cache lines into the SLC from external I/O masters.

It is possible to connect multiple implementations of an Neoverse N1/CMN system (known as a "mesh") connected across a coherent network link. Each mesh is sometimes be referred to as another "chip" in Arm product documentation. "Socket" is another industry term that is sometimes used, but Arm generally uses the word chip for multiple coherent mesh implementations.

CMN products contain their own set of PMU counters and events. Those counters and events are described in the CMN technical documentation.

4 PMU event descriptions

PMU events are described in both the *Arm® Architecture Reference Manual Armv8-A* and the *Arm® Neoverse™ N1 Core Revision: r4p1 Technical Reference Manual*. The descriptions in this document provide a more specific and practical description of the event functionality with relation to the Neoverse N1 microarchitecture.

In addition, the *Arm® Architecture Reference Manual* has a section describing meaningful combinations of common PMU events. Those descriptions can be used to measure metrics such as cache hit rates or instruction performance.

There is also a JSON format list of the PMU events (with descriptions taken from the *Arm® Neoverse™ N1 Core Revision: r4p1 Technical Reference Manual*) here:

https://github.com/ARM-software/data/blob/master/pmu/neoverse_n1.json

4.1 TLB and MMU related events

Please note that while events which cover the L2 TLB behavior are named L2D_<event>, the L2 TLB is a unified TLB and contains both data side and instruction side translations.

This section describes the following events:

- 0x02, L1I_TLB_REFILL, L1 instruction TLB refill
- 0x05, L1D_TLB_REFILL, L1, data TLB refill
- 0x1C, TTBR_WRITE_RETIRED, TTBR write architecturally executed
- 0x25, L1D_TLB, Level 1 data TLB access
- 0x26, L1I_TLB, Level 1 instruction TLB access
- 0x2D, L2D_TLB_REFILL, Attributable L2 unified TLB refill
- 0x2F, L2D_TLB, Attributable L2 unified TLB access
- 0x34, DTLB_WALK, Access to data TLB that caused a page table walk
- 0x35, ITLB_WALK, Access to instruction TLB that caused a page table walk
- 0x4C, L1D_TLB_REFILL_RD, L1 data TLB refill, read
- 0x4D, L1D_TLB_REFILL_WR, L1 data TLB refill, write
- 0x4E, L1D_TLB_RD, L1 data TLB access, read
- 0x4F, L1D_TLB_WR, L1 data TLB access, write
- 0x5C, L2D_TLB_REFILL_RD, L2 unified TLB refill, read
- 0x5D, L2D_TLB_REFILL_WR, L2 unified TLB refill, write
- 0x5E, L2D_TLB_RD, L2 unified TLB access, read
- 0x5F, L2D_TLB_WR, L2 unified TLB access, write

The following architectural and micro-architectural descriptions are relevant to the events listed in this section:

- Attributability
- Speculatively executed versus architecturally executed
- MMU behavior
- TLB behavior
- TLB maintenance operations

4.1.1 0x02, L1I_TLB_REFILL, L1 instruction TLB refill

This event counts L1 I-side TLB refills from any I-side memory access. If there are multiple misses in the TLB that are resolved by the refill, then this event will only count once.

This event will not count if the page table walk results in a fault (such as a translation or access fault), since there is no new translation created for the TLB.

4.1.2 0x05, L1D_TLB_REFILL, L1 data TLB refill

This event counts L1 D-side TLB refills from any D-side memory access. If there are multiple misses in the TLB that are resolved by the refill, then this event will only count once. This event counts for refills caused by preload instructions or hardware prefetch accesses.

This event will count regardless of whether the miss hits in L2 or results in a page table walk.

This event will not count if the page table walk results in a fault (such as a translation or access fault), since there is no new translation created for the TLB.

This event will not count with an access from an AT (address translation) instruction.

This event is the sum of the L1D_TLB_REFILL_RD and L1D_TLB_REFILL_WR events.

4.1.3 0x1C, TTBR_WRITE_RETIRED, TTBR write architecturally executed

This event counts architectural writes to TTBR0/1_EL1. If virtualization host extensions are enabled (by setting the HCR_EL2.E2H bit to 1), then accesses to TTBR0/1_EL1 that are redirected to TTBR0/1_EL2, or accesses to TTBR0/1_EL12, are counted. TTBRn registers are typically updated when the kernel is swapping userspace threads or applications.

4.1.4 0x25, L1D_TLB, Level 1 data TLB access

This event counts any L1 D-side TLB access caused by any memory load or store operation. Note that load or store instructions can be broken up into multiple memory operations.

This event does not count TLB maintenance operations.

For Neoverse N1 CPU versions prior to r4p0, erratum 1356341 could affect this PMU event. For more information about this erratum, please see the *Arm Neoverse N1 (MP050) Software Developer Errata Notice*, available on https://Developer.Arm.com.

This event is the sum of the L1D_TLB_RD and L1D_TLB_WR events.

4.1.5 0x26, L1I_TLB, Level 1 instruction TLB access

This event counts any L1 I-side TLB access whether the access hits or misses in the TLB.

This event is a superset of the L1I_TLB_REFILL event.

4.1.6 0x2D, L2D_TLB_REFILL, Attributable L2 unified TLB refill

This event counts any allocation into the L2 TLB from either an I-side or D-side access.

This event is the sum of the L2D_TLB_REFILL_RD and L2D_TLB_REFILL_WR events.

4.1.7 0x2F, L2D_TLB, Attributable L2 unified TLB access

This event counts any access into the L2 TLB except those caused by TLB maintenance operations.

This event is the sum of the L2D_TLB_RD and L2D_TLB_WR events.

4.1.8 0x34, DTLB_WALK, Access to data TLB that caused a page table walk

This event counts any page table walk (caused by a miss in the L1 D-side and L2 TLB) driven by a D-side memory access. Note that partial translations that also cause a page walk are counted.

This event does not count walks caused by TLB maintenance operations.

4.1.9 0x35, ITLB_WALK, Access to instruction TLB that caused a page table walk

This event counts any page table walk (caused by a miss in the L1 I-side and L2 TLB) driven by a I-side memory access. Note that partial translations that also cause a page walk are counted.

This does not include walks for accessing translations used for accessing page table descriptors (since those are D-side, even if started by an I-side access).

This event does not count walks caused by TLB maintenance operations.

4.1.10 0x4C, L1D_TLB_REFILL_RD, L1 data TLB refill, read

This event counts L1 D-side TLB refills caused by a data side memory read operation. If there are multiple misses in the TLB that are resolved by the refill, then this event will only count once. This event counts for refills caused by preload instructions or hardware prefetch accesses.

This event will count regardless of whether the miss hits in L2 or results in a page table walk.

This event will not count if the page table walk results in a fault (such as a translation or access fault), since there is no new translation created for the TLB.

This event will not count with an access from an *Address Translation* (AT) instruction.

This event is a subset of the L1D_TLB_REFILL event.

4.1.11 0x4D, L1D_TLB_REFILL_WR, L1 data TLB refill, write

This event counts L1 D-side L1 TLB refills caused by a D-side memory write operation. If there are multiple misses in the TLB that are resolved by the refill, then this event will only count once. This event counts for refills caused by preload instructions or hardware prefetch accesses.

This event will count regardless of whether the miss hits in L2 or results in a page table walk.

This event will not count if the page table walk results in a fault (such as a translation or access fault), since there is no new translation created for the TLB.

This event will not count with an access from an *Address Translation (AT)* instruction.

This event is a subset of the L1D_TLB_REFILL event.

4.1.12 0x4E, L1D_TLB_RD, L1 data TLB access, read

This event counts any L1 D-side TLB access caused by a memory read operation. This event counts whether the access hits or misses in the TLB.

This event does not count TLB maintenance operations.

For Neoverse N1 CPU versions prior to r4p0, erratum 1356341 could affect this PMU event. For more information about this erratum, see the *Arm Neoverse N1 (MP050) Software Developer Errata Notice*, available on https://Developer.Arm.com.

This event is a subset of the L1D_TLB event.

4.1.13 0x4F, L1D_TLB_WR, L1 data TLB access, write

This event counts any L1 D-side TLB access caused by a memory write operation. This event counts whether the access hits or misses in the TLB.

This event does not count TLB maintenance operations.

For Neoverse N1 CPU versions prior to r4p0, erratum 1356341 could affect this PMU event. For more information about this erratum, see the *Arm Neoverse N1 (MP050) Software Developer Errata Notice*, available on https://Developer.Arm.com.

This event is a subset of the L1D_TLB event.

4.1.14 0x5C, L2D_TLB_REFILL_RD, L2 unified TLB refill, read

This event counts any allocation into the L2 TLB caused by an I-side or D-side memory read operation.

This event is a subset of the L2D_TLB_REFILL event.

4.1.15 0x5D, L2D_TLB_REFILL_WR, L2 unified TLB refill, write

This event counts any allocation into the L2 TLB caused by a D-side memory write operation.

This event is a subset of the L2D_TLB_REFILL event.

4.1.16 0x5E, L2D_TLB_RD, L2 unified TLB access, read

This event counts any access into the L2 TLB caused by a an I-side or D-side memory read operation except for those caused by TLB maintenance operations.

This event is a subset of the L2D_TLB event.

4.1.17 0x5F, L2D_TLB_WR, L2 unified TLB access, write

This event counts any access into the L2 TLB caused by a D-side memory write operation except for those caused by TLB maintenance operations.

This event is a subset of the L2D_TLB event.

4.2 L1 data cache related events

This section describes the following events:

- 0x03, L1D_CACHE_REFILL, L1 data cache refill
- 0x04, L1D_CACHE, L1 data cache access
- 0x15, L1D_CACHE_WB, L1 data cache write-back
- 0x40, L1D_CACHE_RD, L1 data cache access, read
- 0x41, L1D_CACHE_WR, L1 data cache access, write
- 0x42, L1D_CACHE_REFILL_RD, L1 data cache refill, read
- 0x43, L1D_CACHE_REFILL_WR, L1 data cache refill, write
- 0x44, L1D_CACHE_REFILL_INNER, L1 data cache refill, inner
- 0x45, L1D_CACHE_REFILL_OUTER, L1 data cache refill, outer
- 0x46, L1D_CACHE_WB_VICTIM, L1 data cache write-back, victim
- 0x47, L1D_CACHE_WB_CLEAN, L1 data cache write-back cleaning and coherency
- 0x48, L1D_CACHE_INVAL, L1 data cache invalidate

The following architectural and micro-architectural descriptions are relevant to the events listed in this section:

- Out of order execution
- Cache architecture
- Cache line sizes
- Data side cache allocation
- Cache terminology and behavior
- Cache maintenance operations
- Cache coherency
- Cache lookup
- Cache eviction

4.2.1 0x03, L1D_CACHE_REFILL, L1 data cache refill

This event counts L1 D-cache line allocations caused by speculatively executed load or store instructions where the memory operation misses in the L1 D-cache.

This event does not count cache line allocations from preload instructions or from hardware cache prefetching.

This event only counts one event per cache line. If two operations accessing the same cache line occur, and the second operation has to wait on the fetch of the rest of the cache line, only one event is counted. For example, if a program executes a read from address 0x800C, and then

executes a read from address 0x8000, the second read will not count. If a memory read operation accesses across two cache lines, this event will count twice (once for each cache line).

This event counts the sum of the L1D_CACHE_REFILL_RD and L1D_CACHE_REFILL_WR events.

Since Neoverse N1 caches are write-back only, there are no write-through cache accesses.

4.2.2 0x04, L1D_CACHE, L1 data cache access

This event counts D-cache accesses from any load/store operation that accesses the L1 D-cache.

Please be aware that atomic operations that resolve in the CPU's caches ("near" atomic operations) will count as both a write access and read access.

This event counts the sum of L1D_CACHE_RD and L1D_CACHE_WR.

For Neoverse N1 CPU versions prior to r4p0, erratum 1356341 could affect this PMU event. For more information about this erratum, see the *Arm Neoverse N1 (MP050) Software Developer Errata Notice*, available on https://Developer.Arm.com.

4.2.3 0x15, L1D_CACHE_WB, L1 data cache write-back

This event counts any write-back of dirty data from the L1 data cache to the L2 cache. This occurs when either:

- A dirty cache line is evicted from L1 D-cache and allocated in the L2 cache.
- Dirty data is written to the L2 and possibly to the next level of cache.

This event counts both victim cache line evictions and cache write backs from snoops or cache maintenance operations. The following cache operations are not counted:

- Invalidations which do not result in data being transferred out of the L1 (such as evictions of clean data).
- Full line writes which write to L2 without writing L1, such as write streaming mode.

This event is the sum of the L1D_CACHE_WB_CLEAN and L1D_CACHE_WB_VICTIM events.

4.2.4 0x40, L1D_CACHE_RD, L1 data cache access, read

This event counts any load operation which looks up in the L1 data cache, regardless of whether the access hits in the cache.

This event does not count reads caused by cache maintenance operations or prefetch operations.

This event is a subset of the L1D_CACHE event, except this event only counts memory read operations.

Please be aware that atomic operations that resolve in the CPU's caches ("near" atomic operations) will count as a write access and read access.

For Neoverse N1 CPU versions prior to r4p0, erratum 1356341 could affect this PMU event. For more information about this erratum, see the *Arm Neoverse N1 (MP050) Software Developer Errata Notice*, available on https://Developer.Arm.com.

4.2.5 0x41, L1D_CACHE_WR, L1 data cache access, write

This event counts any store operation which looks up in the L1 data cache. This event also counts accesses caused by a DC ZVA (data cache zero, specified by virtual address) instruction.

This event is a subset of the L1D_CACHE event, except this event only counts memory-write operations.

Please be aware that atomic operations that resolve in the CPU's caches ("near" atomic operations) will count as a write access and read access.

For Neoverse N1 CPU versions prior to r4p0, erratum 1356341 could affect this PMU event. For more information about this erratum, please see the *Arm Neoverse N1 (MP050) Software Developer Errata Notice*, available on https://Developer.Arm.com .

4.2.6 0x42, L1D_CACHE_REFILL_RD, L1 data cache refill, read

This event counts L1 D-cache line allocations caused by speculatively executed load instructions where the memory read operation misses in the L1 D-cache.

This event is a subset of the L1D_CACHE_REFILL event, but this event only counts memory read operations.

This event does not count reads caused by cache maintenance operations or preload instructions.

4.2.7 0x43, L1D_CACHE_REFILL_WR, L1 data cache refill, write

This event counts L1 D-cache line allocations caused by speculatively executed store instructions where the memory write operation misses in the L1 D-cache.

This event is a subset of the L1D_CACHE_REFILL event, but this event only counts memory write operations.

4.2.8 0x44, L1D_CACHE_REFILL_INNER, L1 data cache refill, inner

If the system has a direct connect configuration (with no L3 cache), this event counts any L1 D-cache allocation (as counted by the L1D_CACHE_REFILL event) where the cache line data came from a hit in the L2 cache.

If the system has a DSU with L3 cache, this event counts any L1 D-cache allocation (as counted by the L1D_CACHE_REFILL event) where the cache line data came from a hit in the L2 cache, L3 cache, or another CPU in the cluster.

4.2.9 0x45, L1D_CACHE_REFILL_OUTER, L1 data cache refill, outer

This event counts any cache line allocation into the L1 D-cache (as counted by the L1D_CACHE_REFILL event) which obtains data from outside the cluster. It does not count when the data comes from the L2 cache, the L3 cache, or from another CPU in the DSU cluster.

4.2.10 0x46, L1D_CACHE_WB_VICTIM, L1 data cache write-back, victim

This event counts dirty cache line evictions from the L1 data cache because of a new cache line being allocated.

This event is a subset of the L1D_CACHE_WB event, but the event only counts write-backs that are a result of the line being allocated for an access made by the CPU.

This event does not count evictions caused by cache maintenance operations.

4.2.11 0x47, L1D_CACHE_WB_CLEAN, L1 data cache write-back cleaning and coherency

This event counts write-backs from the L1 data cache that are a result of a coherency operation (including cache maintenance operations) made by another CPU.

This event is a subset of the L1D_CACHE_WB event.

4.2.12 0x48, L1D_CACHE_INVAL, L1 data cache invalidate

This event counts each explicit invalidation of a cache line in the Level 1 data cache caused by:

- *Cache Maintenance Operations* (CMO) that operate by a virtual address.
- Broadcast cache coherency operations from another CPU in the system.

This event does not count for the following conditions:

- A cache refill invalidates a cache line.
- A CMO which is executed on that CPU and invalidates a cache line specified by set/way. Note that CMOs that operate by set/way cannot be broadcast from one CPU to another.

4.3 L1 instruction cache related events

This section describes the following events:

- 0x01, L1I_CACHE_REFILL, L1 instruction cache refill
- 0x14, L1I_CACHE, Level 1 instruction cache access

The following architectural and micro-architectural descriptions are relevant to the events listed in this section:

- Cache architecture
- Cache line sizes
- Instruction side cache allocation
- Cache terminology and behavior
- Cache maintenance operations
- Cache lookup
- Cache eviction

4.3.1 0x01, L1I_CACHE_REFILL, L1 instruction cache refill

This event counts any cache line allocation in the L1 I-cache. Allocations are caused by an instruction fetch which misses in the L1 I-cache. Instruction fetches may include accessing multiple instructions, but the single cache line allocation is counted once.

4.3.2 0x14, L1I_CACHE, Level 1 instruction cache access

This event counts any instruction fetch (which may include accessing multiple instructions) which accesses the L1 instruction cache. Instruction cache accesses caused by cache maintenance operations are not counted.

4.4 L2 cache related events

This section describes the following events:

- 0x16, L2D_CACHE, L2 data cache access
- 0x17, L2D_CACHE_REFILL, L2 cache refill
- 0x18, L2D_CACHE_WB, L2 cache write-back
- 0x20, L2D_CACHE_ALLOCATE, L2 cache allocation without refill
- 0x50, L2D_CACHE_RD, L2 cache access, read
- 0x51, L2D_CACHE_WR, L2 cache access, write
- 0x52, L2D_CACHE_REFILL_RD, L2 cache refill, read
- 0x53, L2D_CACHE_REFILL_WR, L2 cache refill, write
- 0x56, L2D_CACHE_WB_VICTIM, L2 cache write-back, victim
- 0x57, L2D_CACHE_WB_CLEAN, L2 cache write-back, cleaning and coherency
- 0x58, L2D_CACHE_INVAL, L2 cache invalidate

Please note that while L2 cache PMU events are listed as "L2D_<event name>", the Neoverse N1 L2 cache is a unified cache. It contains both instruction and data cache lines.

The following architectural and micro-architectural descriptions are relevant to the events listed in this section:

- CPU and DSU configuration
- Cache architecture
- Cache line sizes
- Data side cache allocation
- Instruction side cache allocation
- Cache terminology and behavior
- Cache maintenance operations
- Cache coherency
- L2 cache and memory interface interaction
- Cache lookup
- Cache eviction

4.4.1 0x16, L2D_CACHE, L2 cache access

This event counts any memory access issued by the CPU to the L2 cache. Accesses are either:

- L1 I-cache miss that looks up in the L2 cache.
- L1 D-cache miss that looks up into the L2 cache.
- L1 D-cache writeback of dirty data into the L2 cache.

This event counts whether the access hits or misses in the L2 cache.

This event is the sum of the L2D_CACHE_RD and L2D_CACHE_WR events.

4.4.2 0x17, L2D_CACHE_REFILL, L2 cache refill

This event counts any cache line allocation into the L2 cache.

This event is a superset of the L2D_CACHE_REFILL_RD event.

4.4.3 0x18, L2D_CACHE_WB, L2 cache write-back

This event counts any write-back of data from the L2 cache to outside the CPU. This includes snoops to the L2 (from other CPUs) which return data even if the snoops cause an invalidation.

L2 cache line invalidations which do not write data outside the CPU and snoops which return data from an L1 cache are not counted. Data would not be written outside the cache when invalidating a clean cache line.

This event is the sum of the L2D_CACHE_WB_VICTIM and L2D_CACHE_WB_CLEAN events.

4.4.4 0x20, L2D_CACHE_ALLOCATE, L2 cache allocation without refill

This event does not count on the Neoverse N1.

4.4.5 0x50, L2D_CACHE_RD, L2 cache access, read

This event counts any read operation issued by the CPU which looks up in the (unified) L2 cache. This event counts whether the access hits or misses in the L2 cache. Snoops from outside the CPU are not counted.

This event is a subset of the L2D_CACHE event, but this event only counts access caused by memory read operations.

4.4.6 0x51, L2D_CACHE_WR, L2 cache access, write

This event counts any memory write operation issued by the CPU which looks up in the (unified) L2 cache. The event counts whether the access hits or misses in the L2 cache. The event also counts any write-back from the L1 data cache that allocates into the L2 cache. This event treats *Data Cache Zero by Virtual Address* (DC ZVA) operations as a store instruction and counts those accesses. Snoops from outside the CPU are not counted.

This event is a subset of the L2D_CACHE event, but this event only counts memory write operations.

4.4.7 0x52, L2D_CACHE_REFILL_RD, L2 cache refill, read

This event counts any cacheable read operation issued by the CPU which causes data to be read from outside the CPU. Store instructions that miss inside the L2 cache will cause this event to count since the cache line is read and allocated into the L2 cache.

This event is a subset of the L2D_CACHE_REFILL event.

This event does not count L2 refills caused by stashes into L2.

4.4.8 0x53, L2D_CACHE_REFILL_WR, L2 cache refill, write

This event does not count on the Neoverse N2.

4.4.9 0x56, L2D_CACHE_WB_VICTIM, L2 cache write-back, victim

This event counts evictions from the L2 cache because of a line being allocated into the L2 cache.

This event does not count evictions caused by cache maintenance operations.

This event is a subset of the L2D_CACHE_WB event.

4.4.10 0x57, L2D_CACHE_WB_CLEAN, L2 cache write-back, cleaning and coherency

This event counts write-backs from the L2 cache that are a result of either:

- Cache maintenance operations.
- Snoop responses.
- Direct cache transfers to another CPU due to a forwarding snoop request.

This event is a subset of the L2D_CACHE_WB event.

4.4.11 0x58, L2D_CACHE_INVAL, L2 cache invalidate

This event counts each explicit invalidation of a cache line in the Level 2 cache by cache maintenance operations that operate by a virtual address, or by external coherency operations.

This event does not count if either:

- A cache refill invalidates a cache line.
- A *Cache Maintenance Operation* (CMO), which invalidates a cache line specified by set/way, is executed on that CPU. CMOs that operate by set/way cannot be broadcast from one CPU to another.

4.5 L3 cache/external system cache related events (for direct connect configuration with no DSU L3)

This section describes the following events:

- 0x29, L3D_CACHE_ALLOCATE, Attributable Level 3 data cache allocation without refill
- 0x2A, L3D_CACHE_REFILL, Attributable Level 3 unified cache refill
- 0x2B, L3D_CACHE, Attributable Level 3 unified cache access
- 0x36, LL_CACHE_RD, Last level cache access, read
- 0x37, LL_CACHE_MISS_RD, Last level cache miss, read
- 0xA0, L3_CACHE_RD, L3 cache read

Some of these events can be affected by the system register CPUECTLR.EXTLLC bit. CPUECTLR.EXTLLC is described in the *Arm® Neoverse™ N1 Technical Reference Manual* and indicates that there is an external (to the CPU) last level cache in the system. The CPUECTLR.EXTLLC bit is set by system software in most system configurations.

Please note that section 4.6 provides a description of these events for systems with a DSU L3.

The following architectural and micro-architectural descriptions are relevant to the events listed in this section:

- Attributability
- CPU and DSU configuration
- Cache architecture
- Cache line sizes
- Cache terminology and behavior
- Cache maintenance operations
- Cache coherency
- L2 cache and memory interface interaction
- Cache lookup
- Cache eviction
- CMN configuration

4.5.1 0x29, L3D_CACHE_ALLOCATE, Attributable Level 3 data cache allocation without refill

This event does not count on the Neoverse N1 for this DSU configuration (direct connect with no L3 cache).

4.5.2 0x2A, L3D_CACHE_REFILL, Attributable Level 3 unified cache refill

This event does not count on the Neoverse N1 for this DSU configuration (direct connect with no L3 cache).

4.5.3 0x2B, L3D_CACHE, Attributable Level 3 unified cache access

This event does not count on the Neoverse N1 for this DSU configuration (direct connect with no L3 cache).

4.5.4 0x36, LL_CACHE_RD, Last level cache access, read

This event does not count when the system register CPUECTLR.EXTLLC bit is not set.

This event counts read transactions returned from outside the Neoverse N1 when the system register CPUECTLR.EXTLLC bit is set.

This event counts any cacheable read bus transaction that returns a data source of:

- System level cache in the coherent interconnect (for example, in the CMN).
- Caches in a CPU in another cluster.
- External system memory (DRAM).
- Remote device.

The data source of the transaction is indicated by a field in the CHI transaction returning to the CPU.

This event does not count reads caused by cache maintenance operations.

Please note that a bus read transaction (which would be counted by this event) could be caused by a store instruction which misses in the L1 D-cache. The Neoverse N1 caches allocate for load and store operations, which would require the cache line containing that memory to be read into the CPU.

This event is a superset of the LL_CACHE_MISS_RD event, since it counts hits in the SLC along with data returned from other external sources.

4.5.5 0x37, LL_CACHE_MISS_RD, Last level cache miss, read

This event does not count when the system register CPUECTLR.EXTLLC bit is not set.

When the system register CPUECTLR.EXTLLC bit is set, then the following applies:

This event counts read transactions returned from outside the Neoverse N1 if the transactions are not returned from the CMN *System Level Cache* (SLC).

This event counts any cacheable read bus transaction that returns a data source of:

- Caches in a CPU in another cluster.
- External system memory (DRAM).

- Remote device.

The data source of the transaction is indicated by a field in the CHI transaction returning to the CPU.

This event does not count reads caused by cache maintenance operations.

Please note that a bus read transaction could be caused by a store operation in the CPU. The Neoverse N1 caches allocate for load and store operations, which would require the cache line containing that memory to be read into the CPU.

This event is a subset of the LL_CACHE_RD event, since it does not count hits in the *System Level Cache* (SLC) along with data returned from other external sources.

4.5.6 0xA0, L3_CACHE_RD, L3 cache read

This event does not count on the Neoverse N1 for this DSU configuration (direct connect with no L3 cache).

4.6 L3 cache/external system cache related events (for DSU with L3)

This section describes the following events:

- 0x29, L3D_CACHE_ALLOCATE, Attributable Level 3 data cache allocation without refill
- 0x2A, L3D_CACHE_REFILL, Attributable Level 3 unified cache refill
- 0x2B, L3D_CACHE, Attributable Level 3 unified cache access
- 0x36, LL_CACHE_RD, Last level cache access, read
- 0x37, LL_CACHE_MISS_RD, Last level cache miss, read
- 0xA0, L3_CACHE_RD, L3 cache read

Some of these events can be affected by the system register CPUECTLR.EXTLLC bit. CPUECTLR.EXTLLC is described in the *Arm® Neoverse™ N1 Technical Reference Manual* and indicates that there is an external (to the CPU) last level cache in the system. The CPUECTLR.EXTLLC is set by system software in most system configurations.

Please note that while some L3 cache PMU events are listed as "L3D_<event name>", the DSU L3 cache is a unified cache. It contains both instruction and data cache lines.

Please note that section 4.5 provides a description of these events for direct connect systems with no DSU L3.

The following architectural and micro-architectural descriptions are relevant to the events listed in this section:

- Attributability
- CPU and DSU configuration
- Cache architecture
- Cache line sizes
- Cache terminology and behavior
- DSU L3 cache allocation
- Cache maintenance operations
- Cache coherency
- L2 cache and memory interface interaction
- Cache lookup
- Cache eviction
- CMN configuration

4.6.1 0x29, L3D_CACHE_ALLOCATE, Attributable Level 3 data cache allocation without refill

This event counts any write from the CPU to an already present cache line in the L3. These writes include:

- Data write-backs from evictions from the L2 cache.
- *Data Cache Zero by Virtual Address* (DC ZVA) operations.
- Streaming writes, such as cache stashing operations.

4.6.2 0x2A, L3D_CACHE_REFILL, Attributable Level 3 unified cache refill

This event counts any cache refill from data returned from outside the CPU cluster. Data returned from another Neoverse N1 in the cluster would not cause this event to count.

4.6.3 0x2B, L3D_CACHE, Attributable Level 3 unified cache access

This event counts all DSU L3 cache accesses.

This event is the sum of the L3_CACHE_RD and L3D_CACHE_REFILL events.

4.6.4 0x36, LL_CACHE_RD, Last level cache access, read

This event does not count when the system register CPUECTLR.EXTLLC bit is not set.

This event counts read transactions returned from outside the Neoverse N1. This event counts any cacheable read bus transaction that returns a data source of:

- System level cache in the coherent interconnect (for example, in the CMN).
- Caches in a CPU in another cluster.
- External system memory (DRAM).
- Remote device.

The data source of the transaction is indicated by a field in the CHI transaction returning to the CPU.

This event does not count reads caused by cache maintenance operations.

Please note that a bus read transaction could be caused by a store operation in the CPU. The Neoverse N1 caches allocate for load and store operations, which would require the cache line containing that memory to be read into the CPU.

This event is a superset of the LL_CACHE_MISS_RD event, since it counts hits in the SLC along with data returned from other external sources.

4.6.5 0x37, LL_CACHE_MISS_RD, Last level cache miss, read

This event does not count when the system register CPUECTLR.EXTLLC bit is not set.

This event counts read transactions returned from outside the Neoverse N1 when the system register CPUECTLR.EXTLLC bit is set.

Data returned from another Neoverse N1 in the cluster, or from the DSU L3 cache, would not cause this event to count. This event counts the same as the L3D_CACHE_REFILL event.

When the system register CPUECTLR.EXTLLC bit is set, this event counts read transactions returned from outside the Neoverse N1 if those transactions are not returned from the CMN System Level Cache. This event counts any cacheable read bus transaction that returns a data source of:

- Caches in a CPU in another cluster.
- External system memory (DRAM).
- Remote device.

The data source of the transaction is indicated by a field in the CHI transaction returning to the CPU.

This event does not count reads caused by cache maintenance operations.

Please note that a bus read transaction could be caused by a store operation in the CPU. The Neoverse N1 caches allocate for load and store operations, which would require the cache line containing that memory to be read into the CPU.

This event is a subset of the LL_CACHE_RD event, since it does not count hits in the SLC along with data returned from other external sources.

4.6.6 0xA0, L3_CACHE_RD, L3 cache read

This event counts cacheable L2 read misses, far atomics, and prefetches targeting the L3 that access the DSU L3 cache.

4.7 Memory system related events

This section describes the following events:

- 0x13, MEM_ACCESS, Data memory access
- 0x19, BUS_ACCESS, Bus access
- 0x1A, MEMORY_ERROR, Local memory error
- 0x31, REMOTE_ACCESS, Access to another socket in a multi-socket system
- 0x60, BUS_ACCESS_RD, Bus access read
- 0x61, BUS_ACCESS_WR, Bus access write
- 0x66, MEM_ACCESS_RD, Data memory access, read
- 0x67, MEM_ACCESS_WR, Data memory access, write

The following architectural and micro-architectural descriptions are relevant to the events listed in this section:

- Attributability
- Speculatively executed versus architecturally executed
- Aligned/Unaligned memory accesses
- CPU and DSU configuration
- Out of order execution
- Cache architecture
- Cache line sizes
- Cache terminology and behavior
- Cache maintenance operations
- Cache coherency
- L2 cache and memory interface interaction
- Cache lookup
- Cache eviction
- CMN configuration

4.7.1 0x13, MEM_ACCESS, Data memory access

This event counts memory accesses issued by the CPU load store unit, where those accesses are issued due to load or store operations. This event also counts any memory access, no matter whether the data is located in any level of cache or external memory.

If memory accesses are broken up into smaller transactions than what were specified in the load or store instructions, then the event counts those smaller memory transactions. Memory accesses generated by the following instructions or activity are not counted:

- Instruction fetches.
- Cache maintenance instructions.
- Translation table walks or prefetches.
- Memory prefetch operations.

This event counts the sum of the MEM_ACCESS_RD and MEM_ACCESS_WR events.

4.7.2 0x19, BUS_ACCESS, Bus access

This event counts any memory accesses issued by the load/store memory system (also referred to as the L2 system) from the CPU to the DSU. If the DSU is implemented in the direct connect configuration, then the transaction will go to the system interconnect (bus). This event counts both D-side and I-side accesses. Each actual bus transaction issued is counted, including snoop requests and snoop responses. If memory accesses are broken up into smaller transactions than what were specified in the load or store instructions, then the event counts those smaller memory transactions.

This event is the sum of the BUS_ACCESS_RD and BUS_ACCESS_WR events.

4.7.3 0x1A, MEMORY_ERROR, Local memory error

This event counts any detected correctable or uncorrectable physical memory error (ECC or parity) in protected CPUs RAMs. On the Neoverse N1, this event counts errors in the caches (including data and tag rams). Any detected memory error (from either a speculative and abandoned access, or an architecturally executed access) is counted.

Please note that errors are only detected when the actual protected memory is accessed by an operation.

4.7.4 0x31, REMOTE_ACCESS, Access to another socket in a multi-socket system

This event counts accesses to another socket, which is implemented as a different CMN mesh in the system. If the CHI bus response back to the Neoverse N1 indicates that the data source is from another chip (mesh), then the counter is updated. If no data is returned, even if the system snoops another chip/mesh, then the counter is not updated.

4.7.5 0x60, BUS_ACCESS_RD, Bus access read

This event counts any memory read transactions issued by the load store unit in the CPU to the DSU. If the DSU is implemented in the direct connect configuration, then the transaction will go to the system interconnect (bus).

The event counts explicit read accesses, as well as accesses from cache prefetching. If memory accesses are broken up into smaller transactions that are issued by the bus interface, then the event counts those smaller transactions.

This event does not count accesses such as coherent snoops that were issued from outside the CPU.

4.7.6 0x61, BUS_ACCESS_WR, Bus access write.

This event counts any memory write transactions issued by the load store unit in the CPU to the DSU. If the DSU is implemented in the direct connect configuration, then the transaction will go to the system interconnect (bus).

The event counts explicit accesses and accesses issued by the caches due to cache evictions. If memory accesses are broken up into smaller transactions that are issued by the bus interface, then the event counts those smaller transactions.

This event does not count accesses such as coherent snoops that were issued from outside the CPU.

4.7.7 0x66, MEM_ACCESS_RD, Data memory access, read

This event counts memory accesses issued by the CPU due to load operations. The event counts any memory load access, no matter whether the data is located in any level of cache or external memory. The event also counts atomic load operations.

If memory accesses are broken up by the load/store unit into smaller transactions that are issued by the bus interface, then the event counts those smaller transactions.

The following instructions are not counted:

- Instruction fetches.
- Cache maintenance instructions.
- Translation table walks or prefetches.
- Memory prefetch operations.

This event is a subset of the MEM_ACCESS event but the event only counts memory-read operations.

4.7.8 0x67, MEM_ACCESS_WR, Data memory access, write

This event counts memory accesses issued by the CPU due to store operations. The event counts any memory store access, no matter whether the data is located in any level of cache or external memory. The event also counts atomic load and store operations.

If memory accesses are broken up by the load/store unit into smaller transactions that are issued by the bus interface, then the event counts those smaller transactions.

The following instructions and operations are not counted:

- Cache maintenance instructions.
- Normal cache operations (for example, evictions)

- Barrier operations (DSB, ESB, DMB, SSBB).
- CLREX (exclusive clear) instructions.
- AT (address translation) instructions.
- Atomic swap operations.

This event is a subset of the MEM_ACCESS event but the event only counts memory-write operations.

4.8 Pipeline related events

This section describes the following events:

- 0x23, STALL_FRONTEND, No operation issued due to the front end
- 0x24, STALL_BACKEND, No operation issued due to the back end

The following architectural and micro-architectural descriptions are relevant to the events listed in this section:

- Attributability
- Speculatively executed versus architecturally executed
- Pipeline and operations
- Out of order execution

4.8.1 0x23, STALL_FRONTEND, No operation issued due to the front end

This event counts cycles whenever the front end (fetch) stages of the pipeline have no operations to send to the rename stage (in the decode/rename/dispatch portion) of the pipeline. That condition would stop those stages from sending operations to be issued to the (backend) execute stages of the pipeline.

In some cases, this event will also count stalls because of certain pipeline resource problems on the back end.

4.8.2 0x24, STALL_BACKEND, No operation issued due to the back end

This event counts cycles whenever the decode/rename/dispatch stage is unable to send operations to be issued to the back end execute stages of the pipeline because of resource constraints. These constraints can include issue stage fullness, execute stage fullness, or other internal pipeline resource fullness.

Note: operations that use a different back end execute pipeline can still be issued if there are pipeline resources available to allow it.

4.9 Load or store instruction related events

This section describes the following events:

- 0x68, UNALIGNED_LD_SPEC, Unaligned access, read
- 0x69, UNALIGNED_ST_SPEC, Unaligned access, write
- 0x6A, UNALIGNED_LDST_SPEC, Unaligned access
- 0x6C, LDREX_SPEC, Exclusive load speculatively executed
- 0x6D, STREX_PASS_SPEC, Successful exclusive store speculatively executed
- 0x6E, STREX_FAIL_SPEC, Failed exclusive store speculatively executed
- 0x6F, STREX_SPEC, Exclusive store speculatively executed
- 0x70, LD_SPEC, Load operation speculatively executed
- 0x71, ST_SPEC, Store operation speculatively executed
- 0x72, LDST_SPEC, Load or store operation speculatively executed
- 0x7D, DSB_SPEC, DSB speculatively executed
- 0x7E, DMB_SPEC, DMB speculatively executed
- 0x90, RC_LD_SPEC, Load-acquire operation speculatively executed
- 0x91, RC_ST_SPEC, Store-release operation speculatively executed

The following architectural and micro-architectural descriptions are relevant to the events listed in this section:

- Attributability
- Speculatively executed versus architecturally executed
- Aligned/Unaligned memory accesses
- CPU and DSU configuration
- Out of order execution
- Cache architecture
- Cache line sizes
- Cache terminology and behavior
- Cache maintenance operations
- Cache coherency
- L2 cache and memory interface interaction
- Cache lookup
- Cache eviction
- CMN configuration

4.9.1 0x68, UNALIGNED_LD_SPEC, Unaligned access, read

This event counts unaligned memory read instructions issued by the CPU. This event counts unaligned accesses (as defined by the actual instruction), even if they are subsequently issued as multiple aligned accesses.

The event does not count preload instructions (PLD, PLI).

This event is a subset of the UNALIGNED_LDST_SPEC event.

4.9.2 0x69, UNALIGNED_ST_SPEC, Unaligned access, write

This event counts unaligned memory write instructions issued by the CPU. This event counts unaligned accesses (as defined by the actual instruction), even if they are subsequently issued as multiple aligned accesses.

This event is a subset of the UNALIGNED_LDST_SPEC event.

4.9.3 0x6A, UNALIGNED_LDST_SPEC, Unaligned access

This event counts unaligned memory instructions issued by the CPU. This event counts unaligned accesses (as defined by the actual instruction), even if they are subsequently issued as multiple aligned accesses.

This event is the sum of the UNALIGNED_ST_SPEC and UNALIGNED_LD_SPEC events.

4.9.4 0x6C, LDREX_SPEC, Exclusive load speculatively executed

This event counts Load-Exclusive instructions (such as LDREX or LDX) that have been speculatively executed.

4.9.5 0x6D, STREX_PASS_SPEC, Successful exclusive store speculatively executed

This event counts Store-Exclusive instructions that have been speculatively executed and have successfully completed the store operation.

4.9.6 0x6E, STREX_FAIL_SPEC, Failed exclusive store speculatively executed

This event counts Store-Exclusive instructions that have been speculatively executed and have not successfully completed the store operation.

4.9.7 0x6F, STREX_SPEC, Exclusive store speculatively executed

This event counts Store-Exclusive instructions that have been speculatively executed. This event is the sum of STREX_PASS_SPEC and STREX_FAIL_SPEC events.

4.9.8 0x70, LD_SPEC, Load instruction speculatively executed

This event counts any speculatively executed load instruction including *Single Instruction Multiple Data* (SIMD) load instructions.

4.9.9 0x71, ST_SPEC, Store instruction speculatively executed

This event counts any speculatively executed store instruction including *Single Instruction Multiple Data* (SIMD) store instructions.

4.9.10 0x72, LDST_SPEC, Load or store instruction speculatively executed

This event does not count on the Neoverse N1.

4.9.11 0x7D, DSB_SPEC, DSB speculatively executed

This event counts DSB instructions that are speculatively (not just architecturally) issued to Load/Store unit in the CPU.

4.9.12 0x7E, DMB_SPEC, DMB speculatively executed

This event counts DMB instructions that are speculatively (not just architecturally) issued to the Load/Store unit in the CPU.

This event does not count implied barriers from load acquire/store release instructions.

4.9.13 0x90, RC_LD_SPEC, Load-acquire operation speculatively executed

This event counts any load acquire (for example LDAR, LDARH, LDARB) instructions that are speculatively executed.

4.9.14 0x91, RC_ST_SPEC, Store-release operation speculatively executed

This event counts any store release (for example STLR, STLRH, STLRB) instructions that are speculatively executed.

4.10 General instruction related events

This section describes the following events:

- 0x08, INST_RETIRED, Instruction architecturally executed
- 0x1B, INST_SPEC, Operation speculatively executed
- 0x73, DP_SPEC, Integer data-processing operation speculatively executed
- 0x74, ASE_SPEC, Advanced SIMD operation speculatively executed
- 0x75, VFP_SPEC, Floating point operation speculatively executed
- 0x76, PC_WRITE_SPEC, PC write operation speculatively executed
- 0x77, CRYPTO_SPEC, Crypto operation speculatively executed
- 0x7C, ISB_SPEC, ISB speculatively executed

The following architectural and micro-architectural descriptions are relevant to the events listed in this section:

- Attributability
- Speculatively executed versus architecturally executed
- Pipeline and operations
- Out of order execution

4.10.1 0x08, INST_RETIRED, Instruction architecturally executed

This event counts any instruction that has been architecturally executed.

For example, speculatively executed instructions that have been abandoned for a branch mispredict will not be counted. This event count should be the same for programs running on any processor regardless of the micro-architectural implementation (since it counts instructions, not operations).

4.10.2 0x1B, INST_SPEC, Instruction speculatively executed

This event counts any instruction that has been speculatively executed.

4.10.3 0x73, DP_SPEC, Integer data-processing instruction speculatively executed

This event counts any speculatively executed logical or arithmetic instruction, including MOV/MVN instructions. For completeness, users should verify the list of instructions as defined in the event description in the *Arm® Architecture Reference Manual Armv8-A*.

4.10.4 0x74, ASE_SPEC, Advanced SIMD instruction speculatively executed

This event counts speculatively executed advanced SIMD instructions (as defined in the *Arm® Architecture Reference Manual Armv8-A*).

This event does not count instructions that move data to or from SIMD (vector) registers.

4.10.5 0x75, VFP_SPEC, Floating point instruction speculatively executed

This event counts speculatively executed floating point instructions (as defined in the *Arm® Architecture Reference Manual Armv8-A*).

This event does not count instructions that move data to or from floating point (vector) registers.

4.10.6 0x76, PC_WRITE_SPEC, PC write instruction speculatively executed

This event counts speculatively executed instructions which cause software changes of the PC. Those instructions include:

- Branch instructions.
- Load instructions with the program counter (PC) as a destination register.
- Exception instructions such as SMC or HVC.
- BKPT instructions.

4.10.7 0x77, CRYPTO_SPEC, Crypto instruction speculatively executed

This event counts speculatively executed Cryptographic instructions except for PMULL and VMULL instructions.

4.10.8 0x7C, ISB_SPEC, ISB speculatively executed

This event counts speculatively executed ISB instructions.

4.11 Branch related events

This section describes the following events:

- 0x10, BR_MIS_PRED, Mispredicted or not predicted branch speculatively executed
- 0x12, BR_PRED, Predictable branch speculatively executed
- 0x21, BR_RETIRED, Branch instruction architecturally executed
- 0x22, BR_MIS_PRED_RETIRED, Mispredicted branch instruction architecturally
- 0x78, BR_IMMED_SPEC, Branch immediate instructions speculatively executed
- 0x79, BR_RETURN_SPEC, Procedure return speculatively executed
- 0x7A, BR_INDIRECT_SPEC, Indirect branch speculatively executed

The following architectural and micro-architectural descriptions are relevant to the events listed in this section:

- Attributability
- Speculatively executed versus architecturally executed
- Pipeline and operations
- Out of order execution

4.11.1 0x10, BR_MIS_PRED, Mispredicted or not predicted branch speculatively executed

This event counts speculatively executed branches that were either not predicted or mispredicted. Please note that the branch needs to be resolved, whether or not it is actually architecturally executed.

4.11.2 0x12, BR_PRED, Predictable branch speculatively executed

This event counts any branch instruction speculatively executed by the CPU. This event counts any predictable branch (including B instructions), whether or not that branch is taken, and whether or not the branch instruction is architecturally executed. This event also counts branches that were possibly mispredicted. This event is a superset of the BR_MIS_PRED event.

4.11.3 0x21, BR_RETIRED, Branch instruction architecturally executed

This event counts all architecturally executed branches, whether the branch is taken or not. Instructions that explicitly write to the PC are also counted.

4.11.4 0x22, BR_MIS_PRED_RETIRED, Mispredicted branch instruction architecturally executed

This event counts any branch instruction (as counted by BR_RETIRED) which is not correctly predicted and causes a pipeline flush.

This event is a subset of BR_RETIRED.

4.11.5 0x78, BR_IMMED_SPEC, Branch immediate instructions speculatively executed

This event counts immediate branch instructions which are speculatively executed. Instructions are defined in the *Arm® Architecture Reference Manual Armv8-A*, and include:

- B <label>
- B.cond <label>
- BL <label>
- CBZ <Rn>, <label>
- CBNZ <Rn>, <label>
- TBZ <Rn>, <label>
- TBNZ <Rn>, <label>

4.11.6 0x79, BR_RETURN_SPEC, Procedure return instruction speculatively executed

This event counts procedure return instructions (RET) which are speculatively executed.

4.11.7 0x7A, BR_INDIRECT_SPEC, Indirect branch instruction speculatively executed

This event counts indirect branch instructions (such as BR Xn or a RET) which are speculatively executed. This includes instructions that force a software change of the PC, other than exception-generating instructions and immediate branch instructions.

4.12 Exception related events

This section describes the following events:

- 0x09, EXC_TAKEN, Exception taken
- 0x0A, EXC_RETURN, Exception return
- 0x81, EXC_UNDEF, Undefined exceptions taken locally
- 0x82, EXC_SVC, Supervisor Call exception taken locally
- 0x83, EXC_PABORT, Instruction abort exception taken locally
- 0x84, EXC_DABORT, Data abort or SError taken locally
- 0x86, EXC_IRQ, IRQ exception taken locally
- 0x87, EXC_FIQ, FIQ exception taken locally
- 0x88, EXC_SMC, Secure Monitor Call exception
- 0x8A, EXC_HVC, Hypervisor Call exception
- 0x8B, EXC_TRAP_PABORT, Instruction abort exception not taken locally
- 0x8C, EXC_TRAP_DABORT, Data abort or SError not taken locally
- 0x8D, EXC_TRAP_OTHER, Other exception not taken locally
- 0x8E, EXC_TRAP_IRQ, IRQ exception not taken locally
- 0x8F, EXC_TRAP_FIQ, FIQ exception not taken locally

The following architectural and micro-architectural descriptions are relevant to the events listed in this section:

- Taken locally
- Memory error behavior
- Out of order execution

Please note a speculatively executed instruction or memory access cannot directly cause an exception until that instruction or access is architecturally executed.

4.12.1 0x09, EXC_TAKEN, Exception taken

This event counts taken exceptions (IRQ, FIQ, SError, and Synchronous). Exceptions are counted whether or not they are taken locally.

4.12.2 0x0A, EXC_RETURN, Exception return

This event counts any architecturally executed exception return instruction for both AArch32 (for example, SUBS PC, LR, #4) and AArch64 (ERET). The *Arm® Architecture Reference Manual Armv8-A* defines a complete list of those instructions.

4.12.3 0x81, EXC_UNDEF, Undefined exceptions taken locally

This event counts the number of synchronous exceptions which are taken locally that are not SVC, CSMC, HVC, data aborts, instruction aborts, or interrupts.

4.12.4 0x82, EXC_SVC, Supervisor Call exception taken locally

This event counts the number of SVC exceptions that are taken locally.

4.12.5 0x83, EXC_PABORT, Instruction abort exception taken locally

This event counts each synchronous exception that is taken locally and is caused by an instruction abort.

4.12.6 0x84, EXC_DABORT, Data abort or SError taken locally

This event counts exceptions that are taken locally and are caused by data aborts or SErrors. Conditions that could cause those exceptions are:

- Attempting to read or write memory where the MMU generates a fault.
- Attempting to read or write memory with a misaligned address.
- Interrupts from the nREI or nSEI inputs.
- Internally generated SErrors.

4.12.7 0x86, EXC_IRQ, IRQ exception taken locally

This event counts IRQ exceptions that are taken locally. This event will count IRQs delivered by the hypervisor to a guest OS, but it will not count IRQs taken by the hypervisor (when IRQs are configured as virtual).

4.12.8 0x87, EXC_FIQ, FIQ exception taken locally

This event counts FIQ exceptions that are taken locally. In real world software, that would mean FIQs taken to EL3 from EL3. This event also counts FIQ exceptions taken to EL1 (which is not a normal use case).

4.12.9 0x88, EXC_SMC, Secure Monitor Call exception

This event counts SMC exceptions (taken to EL3).

4.12.10 0x8A, EXC_HVC, Hypervisor Call exception

This event counts HVC exceptions (taken to EL2).

4.12.11 0x8B, EXC_TRAP_PABORT, Instruction abort exception not taken locally

This event counts synchronous exceptions which are not taken locally and are caused by and is caused by an instruction abort.

4.12.12 0x8C, EXC_TRAP_DABORT, Data abort or SError not taken locally

This event counts exceptions which are not taken locally and are caused by data aborts or SError interrupts. Conditions that could cause those exceptions are:

- Attempting to read or write memory where the MMU generates a fault.
- Attempting to read or write memory with a misaligned address.
- Interrupts from the REI or SEI input.
- Internally generated SErrors.

4.12.13 0x8D, EXC_TRAP_OTHER, Other exception not taken locally

This event counts the number of synchronous exceptions which are not taken locally that are not SVC, CSMC, HVC, data aborts, instruction aborts, or interrupts.

4.12.14 0x8E, EXC_TRAP_IRQ, IRQ exception not taken locally

This event counts IRQs which are taken from EL2 or EL3.

4.12.15 0x8F, EXC_TRAP_FIQ, FIQ exception not taken locally

This event counts FIQs which are taken from EL0, EL1, or EL2 to EL3 (which would be the normal behavior for FIQs when not executing in EL3).

4.13 General CPU related events

This section describes the following events:

- 0x00, SW_INCR, Software increment
- 0x0B, CID_WRITE_RETIRED CONTEXTIDR, register write
- 0x11, CPU_CYCLES, Cycles
- 0x1D, BUS_CYCLES, Bus cycles
- 0x1E, CHAIN PMU, Counter

4.13.1 0x00, SW_INCR Software increment

This event counts software writes to the PMSWINC_EL0 (software PMU increment) register. The PMSWINC_EL0 register is a manually updated counter for use by application software.

This event could be used to measure any user program event, such accesses to a particular data structure (by writing to the PMSWINC_EL0 register each time the data structure is accessed).

To use the PMSWINC_EL0 register and event, developers must insert instructions that write to the PMSWINC_EL0 register into the source code. Since the SW_INCR event records writes to the PMSWINC_EL0 register, there is no need to do a read/increment/write sequence to the PMSWINC_EL0 register.

4.13.2 0x0B, CID_WRITE_RETIRED, CONTEXTIDR register write

This event counts any architecturally executed write to the CONTEXTIDR register. Normally, that register would contain the kernel PID and would be output with hardware trace. For more information, please consult the Linux documentation found here:

https://elixir.bootlin.com/linux/v4.20.17/source/arch/arm64/Kconfig.debug#L19

4.13.3 0x11, CPU_CYCLES, Cycles

This event counts CPU clock cycles (not timer cycles). The clock measured by this event is defined as the physical clock driving the CPU logic.

4.13.4 0x1D, BUS_CYCLES, Bus cycles

This event counts bus cycles in the CPU. Bus cycles represent a clock cycle in which a message could be sent or received on the interface from the CPU to the DSU (either the DSU wrapper used in the direct connect configuration or the full DSU). Since that interface is driven at the same clock speed as the CPU, this event is a duplicate of CPU_CYCLES. For more information, please see the *Arm® DynamIQ™ Shared Unit Technical Reference Manual*.

4.13.5 0x1E, CHAIN, PMU counter overflow increment

This event counts when the immediately preceding even-number counter overflows. When the two counters are effectively chained together, the PMU counter pair implements a 64-bit counter, with the event defined by the odd numbered counter. For example, if PMUEVCNTR2 is programmed to measure an event, and PMU3 is programmed with the CHAIN event, then PMU2 and PMU3 will function as a 64-bit counter for the event programmed for PMU2.

5 CPU memory system flows

These flows show how the internal MMU or cache accesses work following load or store instructions, or for any instruction fetch. Events that can be counted in each flow are numbered in the flowcharts, and then listed on that page.

For MMU and TLB accesses and flows, those flows will be followed for any memory access or instruction fetch. Cacheability is determined by the page tables in the MMU, and those translations and memory attributes for a given address are stored in the TLBs.

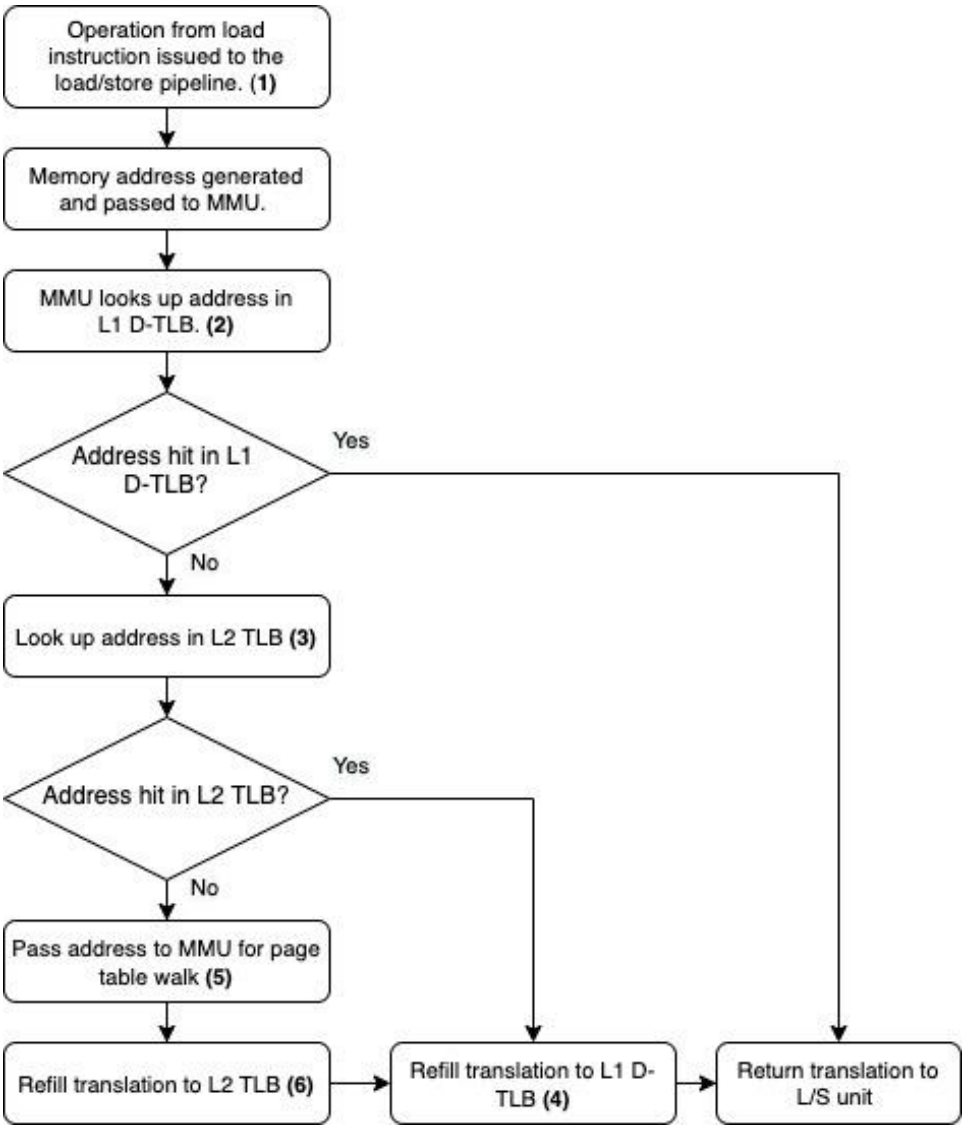
The flows showing cache behavior and events will only be followed if the CPU is accessing an address that is marked as cacheable in the MMU page tables.

These descriptions are for a direct connect configuration (the DSU has no L3 or snoop logic and connects directly to the CHI-based interconnect, such as CMN).

Please also note that other PMU events (such as events counting the number of instructions that have been executed) may also count. However, these diagrams are only intended to show PMU events resulting from the internal TLB/MMU and cache behaviors.

5.1 Data side TLB access for a load instruction

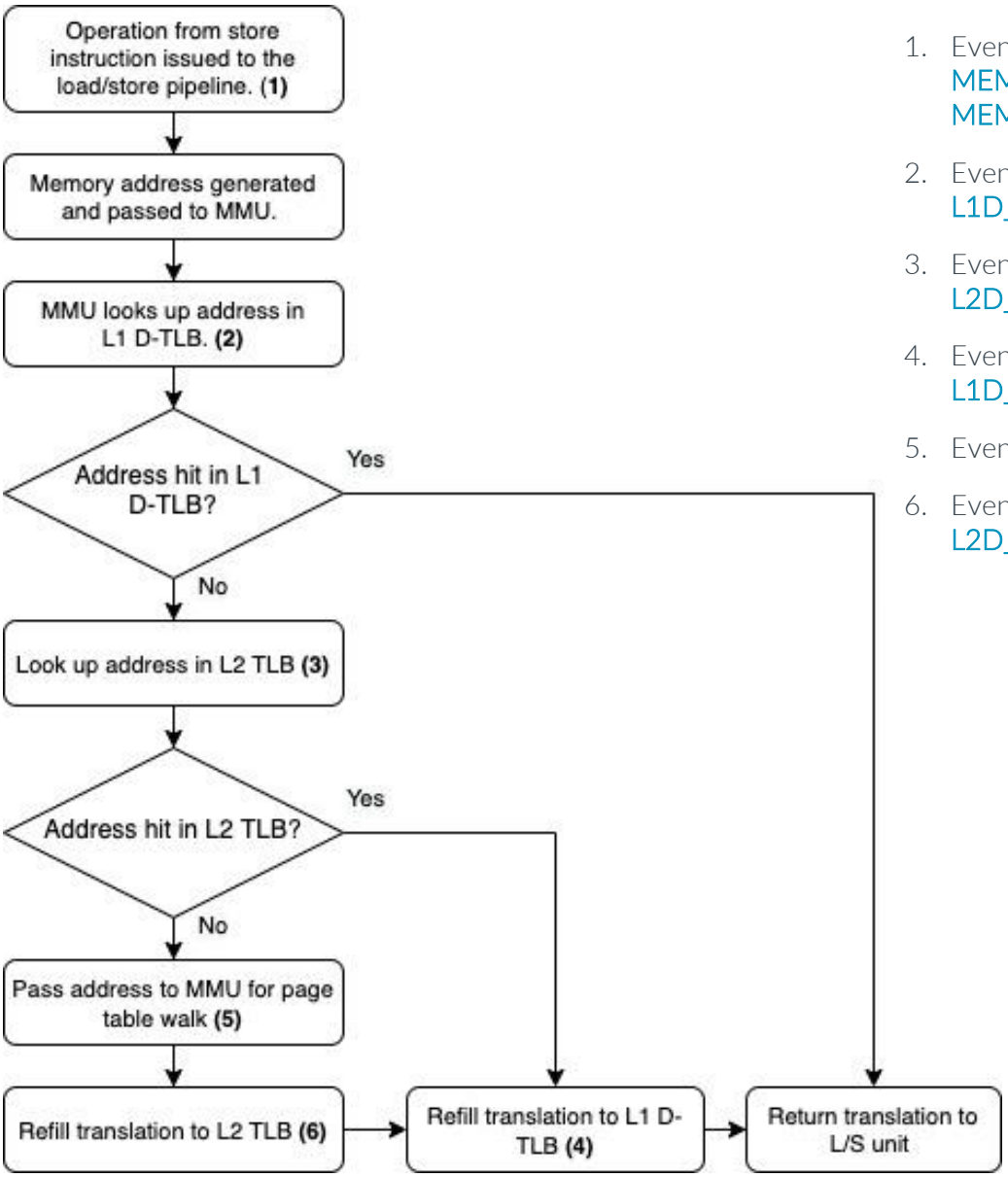
The following flowchart describes the data side access for a read from any cacheable or non-cacheable location



1. Events: LD_SPEC and MEM_ACCESS and MEM_ACCESS_RD
2. Events: L1D_TLB and L1D_TLB_RD
3. Events: L2D_TLB and L2D_TLB_RD
4. Events: L1D_TLB_REFILL and L1D_TLB_REFILL_RD
5. Event: DTLB_WALK
6. Events: L2D_TLB_REFILL and L2D_TLB_REFILL_RD

5.2 Data side TLB access for a store instruction

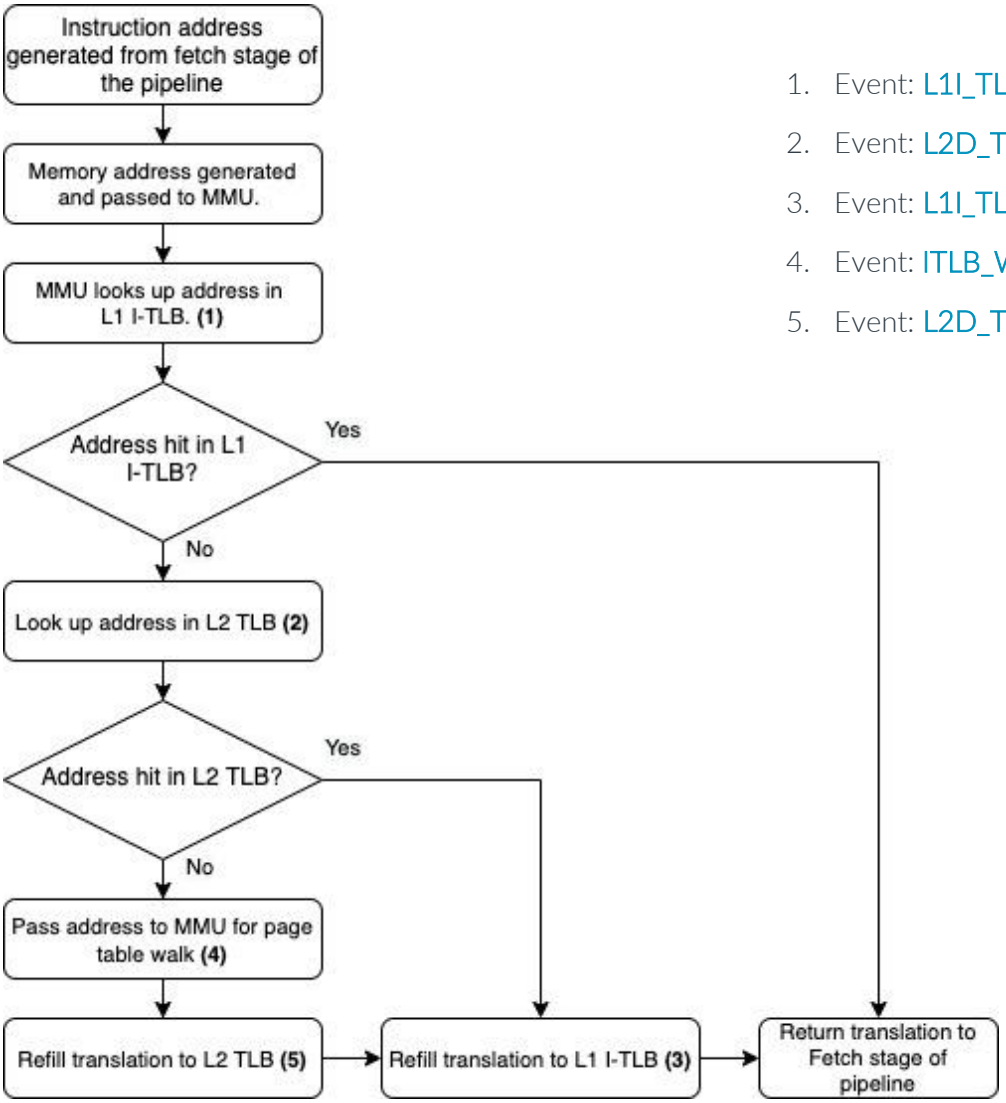
The following flowchart describes the data side access for a store to any cacheable or non-cacheable location.



1. Events: ST_SPEC and MEM_ACCESS and MEM_ACCESS_WR
2. Events: L1D_TLB and L1D_TLB_WR
3. Events: L2D_TLB and L2D_TLB_WR
4. Events: L1D_TLB_REFILL and L1D_TLB_REFILL_WR
5. Event: DTLB_WALK
6. Events: L2D_TLB_REFILL and L2D_TLB_REFILL_WR

5.3 Instruction side TLB access

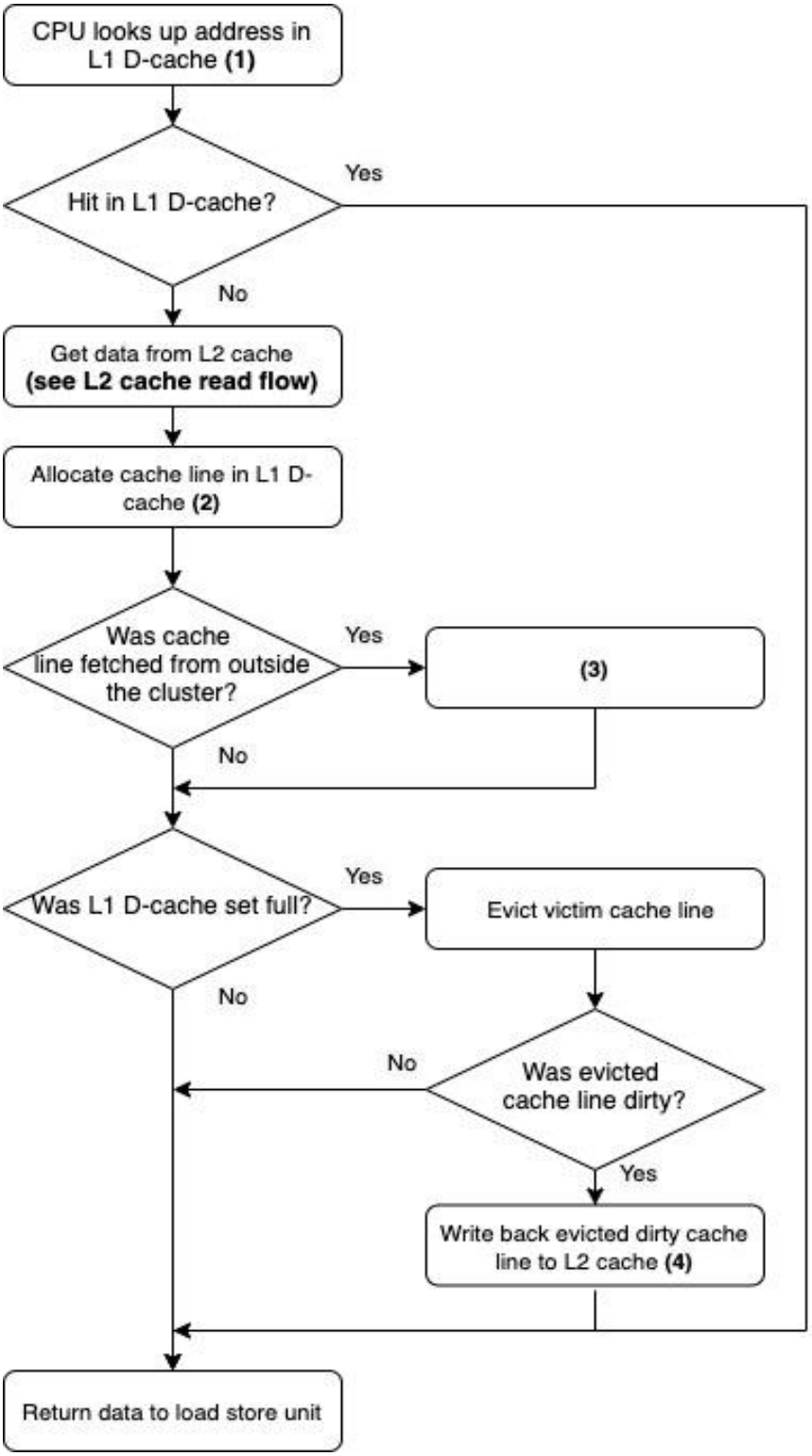
The following flowchart describes the address translation for any instruction fetch.



1. Event: L1I_TLB
2. Event: L2D_TLB
3. Event: L1I_TLB_REFILL
4. Event: ITLB_WALK
5. Event: L2D_TLB_REFILL

5.4 L1 Data cache read access

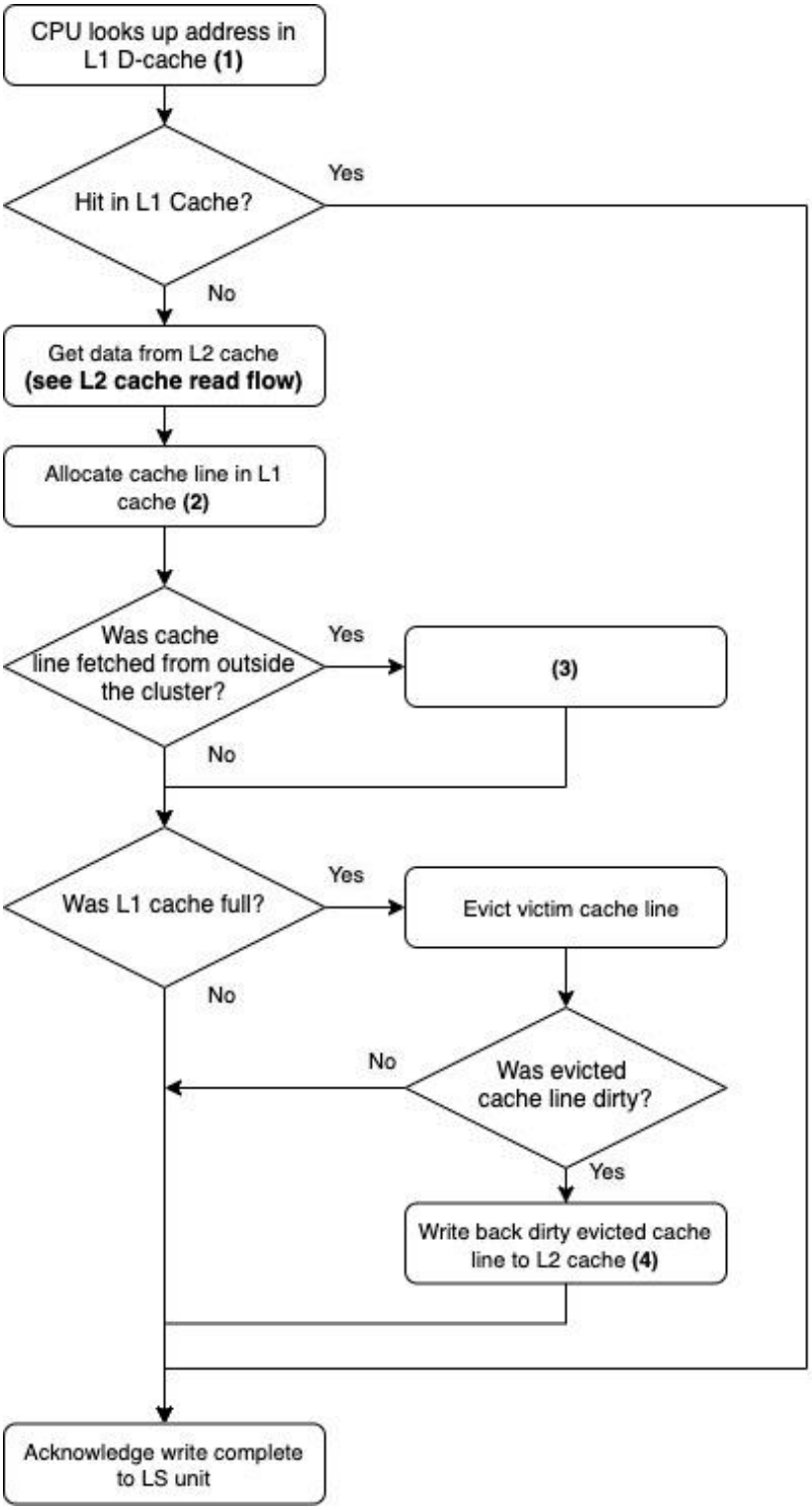
The following flowchart describes a read from a location that is marked as cacheable.



1. Events: L1D_CACHE and L1D_CACHE_RD and MEM_ACCESS and MEM_ACCESS_RD
2. Events: L1D_CACHE_REFILL and L1D_CACHE_REFILL_RD
3. Event: L1D_CACHE_REFILL_OUTER
4. Events: L1D_CACHE_WB and L1D_CACHE_WB_VICTIM

5.5 L1 Data cache write access

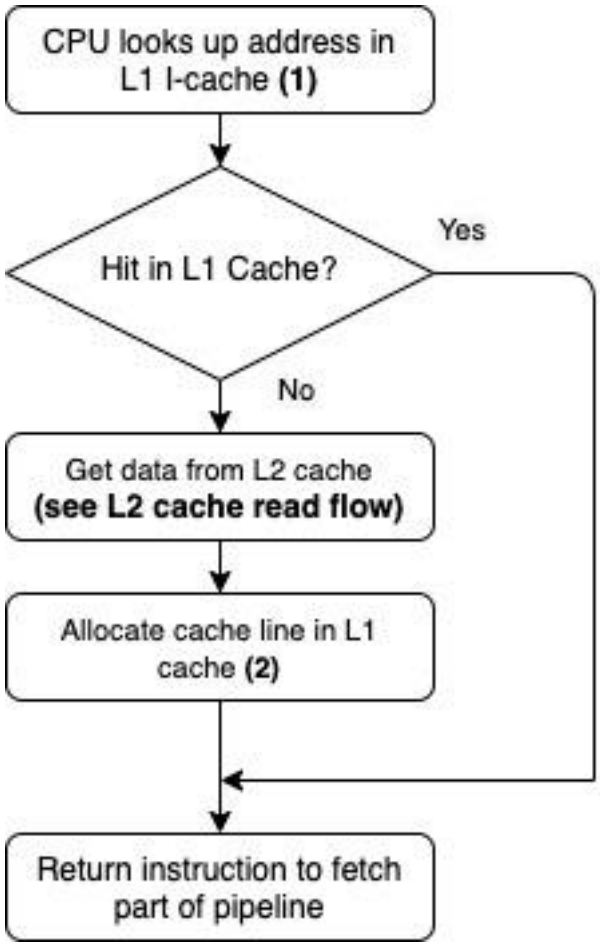
The following flowchart describes a store to a location that is marked as cacheable.



1. Events: L1D_CACHE and L1D_CACHE_WR and MEM_ACCESS and MEM_ACCESS_WR
2. Events: L1D_CACHE_REFILL and L1D_CACHE_REFILL_WR
3. Event: L1D_CACHE_REFILL_OUTER
4. Events: L1D_CACHE_WB and L1D_CACHE_WB_VICTIM

5.6 Instruction side cache access

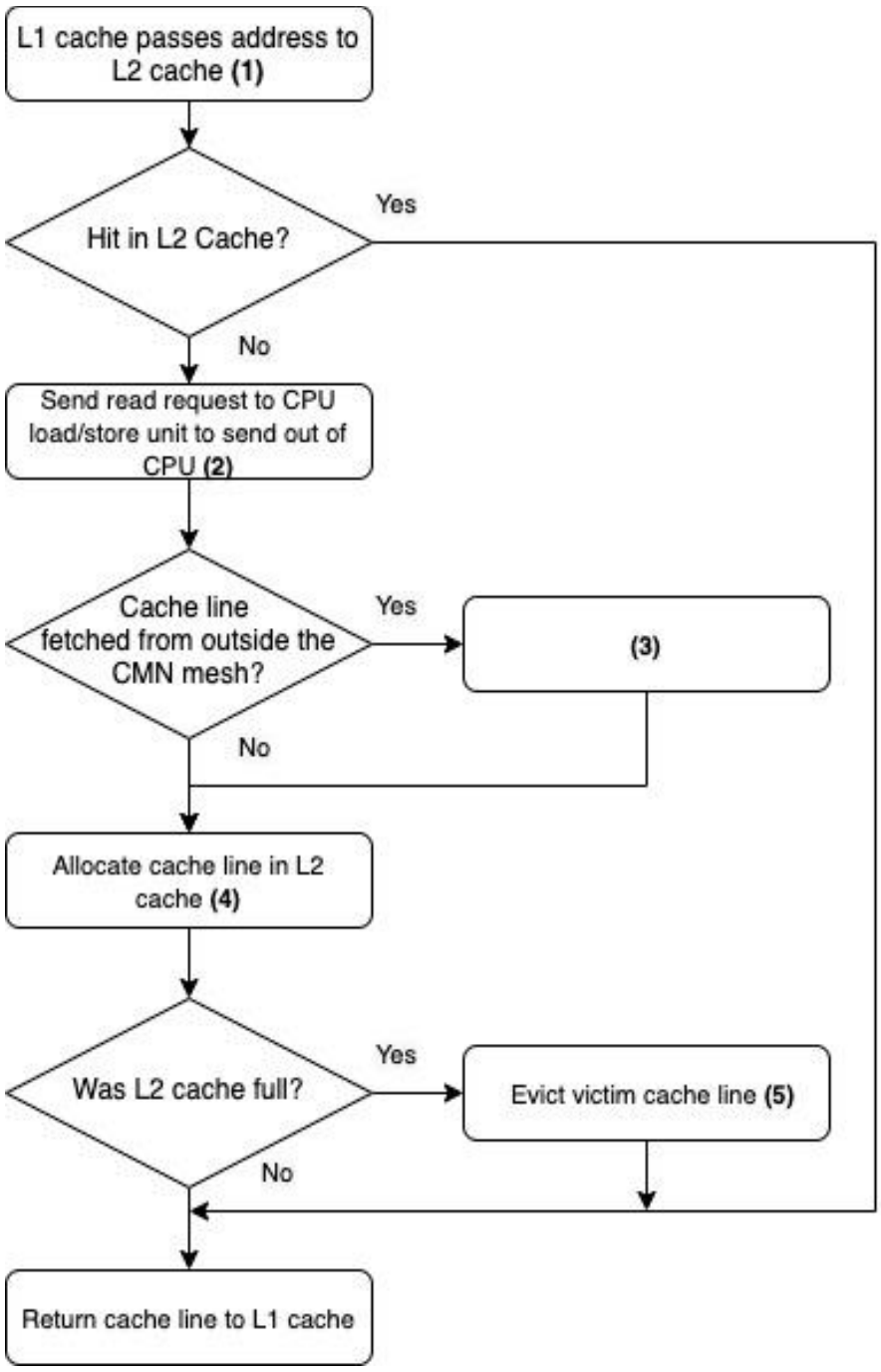
The following flowchart describes an instruction fetch from the pipeline.



1. Event: L1I_CACHE
2. Event: L1I_CACHE_REFILL

5.7 L2 cache read access

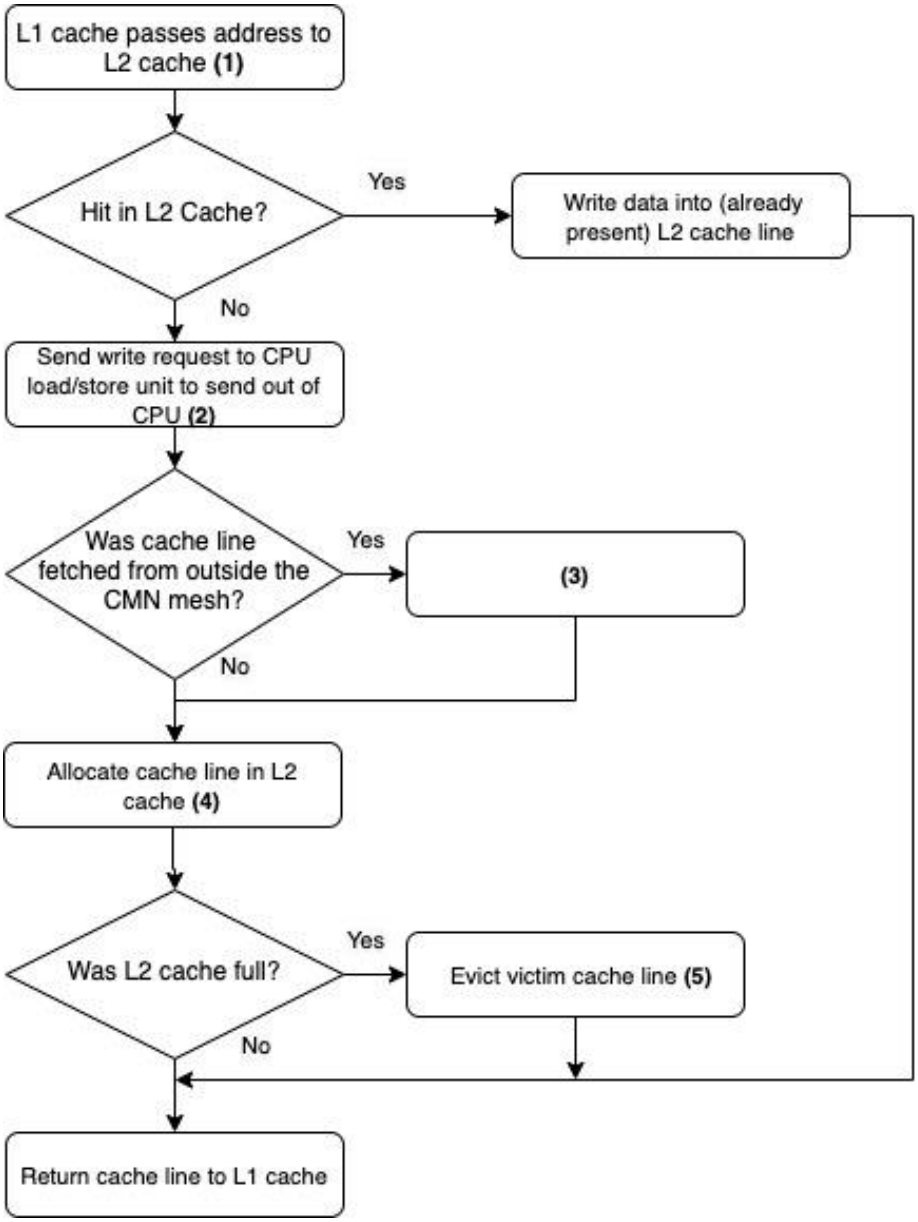
The following flowchart describes an L2 read access from either the L1 I-cache or L1 D-cache.



1. Events: L2D_CACHE and L2D_CACHE_RD
2. Events: BUS_ACCESS and BUS_ACCESS_RD
3. Event: REMOTE_ACCESS
4. Events: L2D_CACHE_REFILL_RD and L2D_CACHE_REFILL
5. Events: L2D_CACHE_WB and L2D_CACHE_WB_VICTIM

5.8 L2 cache write access

The following flowchart describes an L2 write access from the L1 D-cache.



1. Events: L2D_CACHE and L2D_CACHE_WR
2. Events: BUS_ACCESS and BUS_ACCESS_WR
3. Event: REMOTE_ACCESS
4. Events: L2D_CACHE_REFILL_RD and L2D_CACHE_REFILL
5. Events: L2D_CACHE_WB and L2D_CACHE_WB_VICTIM

6 Metrics

This section lists performance metrics that can be measured using different combinations of PMU events. To calculate these metrics, two or more PMU counters will be programmed with the events listed for the metric. Both counters will be read at the same time to determine the metric value.

In addition, the *Arm® Architecture Reference Manual* has a section describing meaningful combinations of common PMU events. Instruction/pipeline/execution metrics

Metric	Formula
Architecturally executed *Instructions Per Cycle* (IPC)	INST_RETIRED / CPU_CYCLES
Speculatively executed *Instructions Per Cycle* (IPC)	INST_SPEC / CPU_CYCLES
Front end stall rate	STALL_FRONTEND / CPU_CYCLES
Back end stall rate	STALL_BACKEND / CPU_CYCLES
Exception rate over time	EXC_TAKEN / CPU_CYCLES
Exception rate per instructions	EXC_TAKEN / INST_RETIRED
Branch misprediction rate	BR_MIS_PRED_RETIRED / BR_RETIRED
Successful exclusive store access rate	STREX_PASS_SPEC / STREX_SPEC
Failed exclusive store accesses rate	STREX_FAIL_SPEC / STREX_SPEC
DSB rate per instructions	DSB_SPEC / INST_SPEC
DSB rate over time	DSB_SPEC / CPU_CYCLES
DMB rate per instructions	DMB_SPEC / INST_SPEC
DMB rate over time	DMB_SPEC / CPU_CYCLES
ISB rate per instructions	ISB_SPEC / INST_SPEC
ISB rate over time	ISB_SPEC / CPU_CYCLES
SIMD instruction rate per instructions	ASE_SPEC / INST_SPEC
SIMD instruction rate over time	ASE_SPEC / CPU_CYCLES

6.1 TLB/MMU metrics

Metric	Formula
L1 data TLB miss rate	L1D_TLB_REFILL / L1D_TLB
L1 data TLB read miss rate	L1D_TLB_REFILL_RD / L1D_TLB_RD
L1 data TLB write miss rate	L1D_TLB_REFILL_WR / L1D_TLB_WR
L1 data TLB misses per context swap (see note below)	L1D_TLB_REFILL / TTBR_WRITE_RETIRED
L1 instruction TLB miss rate	L1I_TLB_REFILL / L1I_TLB
L1 instruction TLB misses per context swap (see note below)	L1I_TLB_REFILL / TTBR_WRITE_RETIRED
L2 TLB miss rate	L2D_TLB_REFILL / L2D_TLB
L2 TLB read miss rate	L2D_TLB_REFILL_RD / L2D_TLB_RD
L2 TLB write miss rate	L2D_TLB_REFILL_WR / L2D_TLB_WR
L2 TLB misses per context swap (see note below)	L2D_TLB_REFILL / TTBR_WRITE_RETIRED
D-side page table walk rate	DTLB_WALK / L1D_TLB
I-side page table walk rate	ITLB_WALK / L1I_TLB

Note: If the operating system is using *Kernel Page Table Isolation* (KPTI) or a similar technique, there may be additional writes to the TTBR registers.

6.2 Cache metrics

Metric	Formula
L1 I-cache miss rate	L1I_CACHE_REFILL / L1I_CACHE
L1 I-cache miss per instructions	L1I_CACHE_REFILL / INST_SPEC
L1 D-cache miss rate	L1D_CACHE_REFILL / L1D_CACHE
L1 D-cache rate of cache misses in L1 and L2	L1D_CACHE_REFILL_OUTER / L1D_CACHE_REFILL
L1 D-cache read miss rate	L1D_CACHE_REFILL_RD / L1D_CACHE_RD
L1 D-cache write miss rate	L1D_CACHE_REFILL_WR / L1D_CACHE_WR
L1 D-cache read rate	L1D_CACHE_RD/ L1D_CACHE
L1 D-cache write rate	L1D_CACHE_WR / L1D_CACHE
L1 D-cache eviction rate	L1D_CACHE_WB_VICTIM / L1D_CACHE
L2 cache miss rate	L2D_CACHE_REFILL / L2D_CACHE
L2 cache read rate	L2D_CACHE_RD / L2D_CACHE
L2 cache write rate	L2D_CACHE_WR / L2D_CACHE
L2 cache eviction rate	L2D_CACHE_WB_VICTIM / L2D_CACHE

Appendix A Revisions

This appendix describes the technical changes between released issues of this document.

Table A-1 Issue 01

Change	Location	Affects
n/a	Initial release	n/a

Table A-2 Differences between issue 01 and issue 02

Change	Location	Affects
n/a	n/a	n/a

Appendix B List of PMU events by number

0x20 L2D_CACHE_ALLOCATE L2 cache allocation without refill

0x21 BR_RETIRED Branch instruction architecturally executed

0x22 BR_MIS_PRED_RETIRED Mispredicted branch instruction architecturally executed

0x23 STALL_FRONTEND No operation issued due to the front end

0x24 STALL_BACKEND No operation issued due to the back end

0x25 L1D_TLB Level 1 data TLB access

0x26 L1I_TLB Level 1 instruction TLB access

0x29 L3D_CACHE_ALLOCATE Attributable Level 3 data cache allocation without refill

0x2A L3D_CACHE_REFILL Attributable Level 3 unified cache refill

0x2B L3D_CACHE Attributable Level 3 unified cache access

0x2D L2D_TLB_REFILL Attributable L2 unified TLB refill

0x2F L2D_TLB Attributable L2 or unified TLB access

0x31 REMOTE_ACCESS Access to another socket in a multi-socket system

0x34 DTLB_WALK Access to data TLB that caused a page table walk

0x35 ITLB_WALK Access to instruction TLB that caused a page table walk

0x36 LL_CACHE_RD Last level cache access, read

0x37 LL_CACHE_MISS_RD Last level cache miss, read

0x40 L1D_CACHE_RD L1 data cache access, read

0x41 L1D_CACHE_WR L1 data cache access, write

0x42 L1D_CACHE_REFILL_RD L1 data cache refill, read

0x43 L1D_CACHE_REFILL_WR L1 data cache refill, write

0x44 L1D_CACHE_REFILL_INNER L1 data cache refill, inner

0x45 L1D_CACHE_REFILL_OUTER L1 data cache refill, outer

0x46 L1D_CACHE_WB_VICTIM L1 data cache write-back, victim

0x47 L1D_CACHE_WB_CLEAN L1 data cache write-back cleaning and coherency

0x48 L1D_CACHE_INVAL L1 data cache invalidate

0x4C L1D_TLB_REFILL_RD L1 data TLB refill, read

0x4D L1D_TLB_REFILL_WR L1 data TLB refill, write

0x4E L1D_TLB_RD L1 data TLB access, read

0x4F, L1D_TLB_WR, L1 data TLB access, write

0x50 L2D_CACHE_RD L2 cache access, read

0x51 L2D_CACHE_WR L2 cache access, write

0x52 L2D_CACHE_REFILL_RD L2 cache refill, read

0x53 L2D_CACHE_REFILL_WR L2 cache refill, write

0x56 L2D_CACHE_WB_VICTIM L2 cache write-back, victim

0x57 L2D_CACHE_WB_CLEAN L2 cache write-back, cleaning and coherency

0x58 L2D_CACHE_INVAL L2 cache invalidate

0x5C L2D_TLB_REFILL_RD L2 or unified TLB refill, read

0x5D L2D_TLB_REFILL_WR L2 or unified TLB refill, write

0x5E L2D_TLB_RD L2 or unified TLB access, read

0x5F L2D_TLB_WR L2 or unified TLB access, write

0x60 BUS_ACCESS_RD Bus access read

0x61 BUS_ACCESS_WR Bus access write.

0x66 MEM_ACCESS_RD Data memory access, read

0x67 MEM_ACCESS_WR Data memory access, write

0x68 UNALIGNED_LD_SPEC Unaligned access, read

0x69 UNALIGNED_ST_SPEC Unaligned access, write

0x6A UNALIGNED_LDST_SPEC Unaligned access

0x6C LDREX_SPEC Exclusive load speculatively executed

0x6D STREX_PASS_SPEC Successful exclusive store speculatively executed

0x6E STREX_FAIL_SPEC Failed exclusive store speculatively executed

0x6F STREX_SPEC Exclusive store speculatively executed

0x70 LD_SPEC Load operation speculatively executed

0x71 ST_SPEC Store operation speculatively executed

0x72 LDST_SPEC. Load or store operation speculatively executed

0x73 DP_SPEC Integer data-processing operation speculatively executed

0x74 ASE_SPEC Advanced SIMD operation speculatively executed

0x75 VFP_SPEC Floating point operation speculatively executed

0x76 PC_WRITE_SPEC PC write operation speculatively executed

0x77 CRYPTO_SPEC Crypto operation speculatively executed

0x78 BR_IMMED_SPEC Branch immediate instructions speculatively executed

0x79 BR_RETURN_SPEC Procedure return speculatively executed

0x7A BR_INDIRECT_SPEC Indirect branch speculatively executed

0x7C ISB_SPEC ISB speculatively executed

0x7D DSB_SPEC DSB speculatively executed

0x7E DMB_SPEC DMB speculatively executed

0x81 EXC_UNDEF Undefined exceptions taken locally

0x82 EXC_SVC Supervisor Call exception taken locally

0x83 EXC_PABORT Instruction abort exception taken locally

0x84 EXC_DABORT Data abort or SError taken locally

0x86 EXC_IRQ IRQ exception taken locally

0x87 EXC_FIQ FIQ exception taken locally

0x88 EXC_SMC Secure Monitor Call exception

0x8A EXC_HVC Hypervisor Call exception

0x8B EXC_TRAP_PABORT. Instruction abort exception not taken locally

0x8C EXC_TRAP_DABORT Data abort or SError not taken locally

0x8D EXC_TRAP_OTHER Other exception not taken locally

0x8E EXC_TRAP_IRQ IRQ exception not taken locally

0x8F EXC_TRAP_FIQ FIQ exception not taken locally

0x90 RC_LD_SPEC Load-acquire operation speculatively executed

0x91 RC_ST_SPEC Store-release operation speculatively executed

0xA0 L3_CACHE_RD L3 cache read