arm

Arm® Neoverse™ V2 Core

Revision: r0p2

Software Optimization Guide
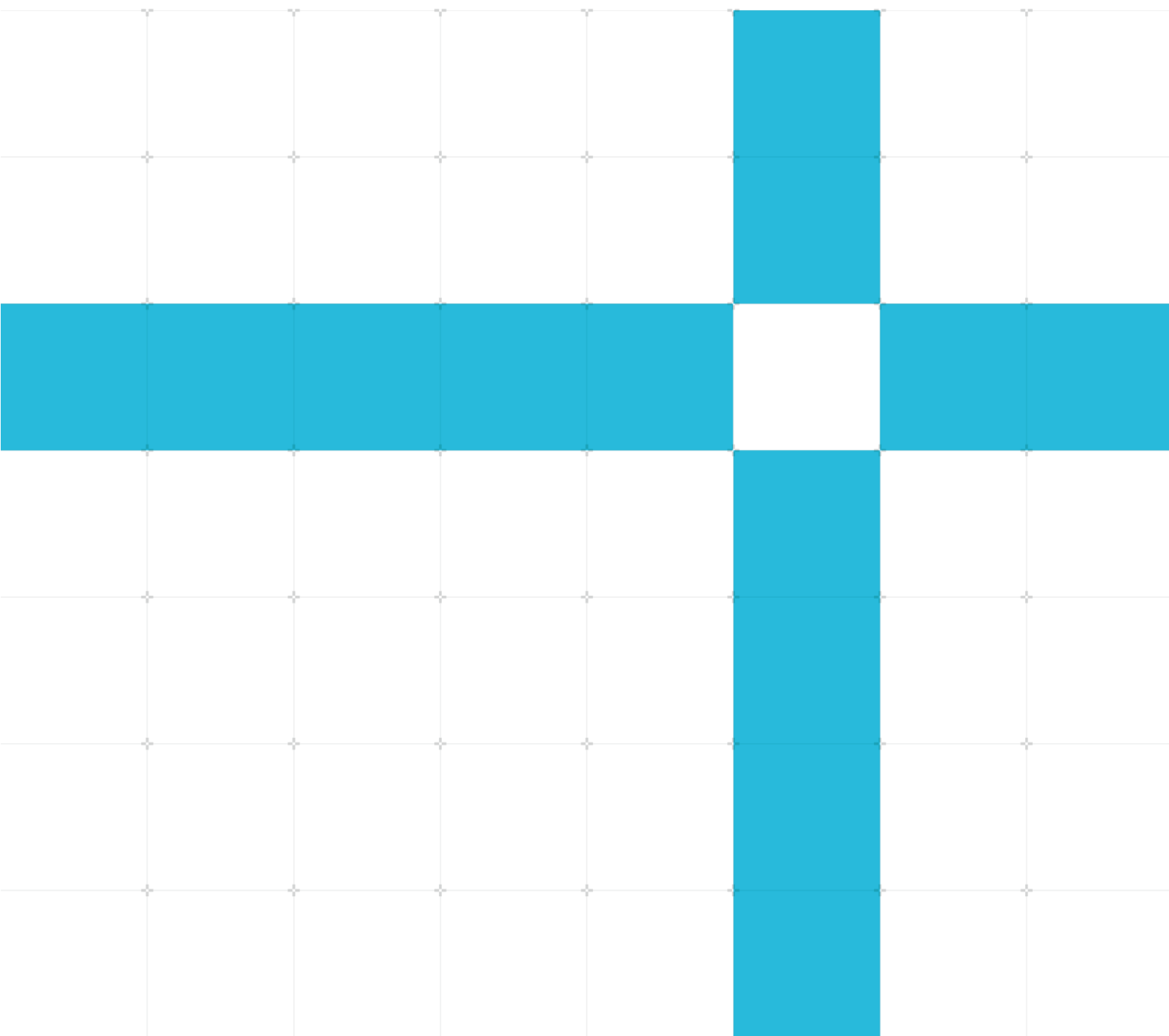
Non-Confidential

Issue 3.0

PJDOC-466751330-593177

Arm® Neoverse™ V2 Core
Software Optimization Guide

Copyright © 2021, 2022 Arm Limited (or its affiliates). All rights reserved.

Release information

Document history

Issue	Date	Confidentiality	Change
1.0	22 Oct 2021	Confidential	First release for r0p0
2.0	30 Sep 2022	Non-Confidential	First release for r0p1
3.0	16 Dec 2022	Non-Confidential	First release for r0p2

Non-Confidential Proprietary Notice

Confidentiality Status

Product Status

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

Web Address

developer.arm.com

Inclusive language commitment

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

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

Contents

1 Introduction

1.1 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 for system designers, system integrators, and programmers who are designing or programming a System-on-Chip (SoC) that uses an Arm core.

1.3 Scope

This document describes aspects of the Neoverse V2 core micro-architecture that influence software performance. Micro-architectural detail is limited to that which is useful for software optimization.

Documentation extends only to software visible behavior of the Neoverse V2 core and not to the hardware rationale behind the behavior.

1.4 Conventions

The following subsections describe conventions used in Arm documents.

1.4.1 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**.

1.4.2 Terms and abbreviations

This document uses the following terms and abbreviations.

Term	Meaning
ALU	Arithmetic and Logical Unit
ASIMD	Advanced SIMD
MOP	Macro-OPeration
µOP	Micro-OPeration
SQRT	Square Root
FP	Floating-point

1.4.3 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` <u>underline</u>	Denotes a permitted abbreviation for a command or option. You can enter the underlined text instead of the full command or option name.
<and>	Encloses replaceable terms for assembler syntax where they appear in code or code fragments. For example: `MRC p15, 0, <Rd>, <CRn>, <CRm>, <Opcode_2>`
SMALL CAPITALS	Used in body text for a few terms that have specific technical meanings, that are defined in the Arm® Glossary. For example, IMPLEMENTATION DEFINED, IMPLEMENTATION SPECIFIC, UNKNOWN, and UNPREDICTABLE.
Caution	This represents a recommendation which, if not followed, might lead to system failure or damage.
Warning	This represents a requirement for the system that, if not followed, might result in system failure or damage.
Danger	This represents a requirement for the system that, if not followed, will result in system failure or damage.
Note	This represents an important piece of information that needs your attention.
Tip	This represents a useful tip that might make it easier, better or faster to perform a task.
Remember	This is a reminder of something important that relates to the information you are reading.

1.5 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, Armv8, for Armv8-A architecture profile	DDI 0487	N
Arm® Neoverse™ V2 Core Technical Reference Manual	102375	N

1.6 Feedback

Arm welcomes feedback on this product and its documentation.

1.6.1 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.

1.6.2 Feedback on content

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

- The title Arm® Neoverse™ V2 Core Software Optimization Guide .

- The number PJDOC-466751330-593177.

- If viewing a PDF version of a document, the page number(s) to which your comments refer.

- If viewing online, the topic names to which your comments apply.

- A concise explanation of your comments.

Arm also welcomes general suggestions for additions and improvements.

Note

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

2 Overview

The Neoverse™ V2 core is a high-performance and low-power product that implements the Arm®v9.0-A architecture. The Arm®v9.0-A architecture extends the architecture defined in the Armv8-A architectures up to Arm®v8.6-A.

The key features of Neoverse V2 Core are:

- Implementation of the Armv9-A A64 instruction set

- AArch64 Execution state at all Exception levels, EL0 to EL3

- Memory Management Unit (MMU)

- 48-bit Physical Address (PA) and 48-bit Virtual Address (VA)

- Generic Interrupt Controller (GIC) CPU interface to connect to an external interrupt distributor

- Generic Timers interface that supports 64-bit count input from an external system counter

- Implementation of the Reliability, Availability, and Serviceability (RAS) Extension

- Implementation of the Scalable Vector Extension (SVE) with a 128-bit vector length and Scalable Vector Extension 2 (SVE2)

- Integrated execution unit with Advanced SIMD and floating-point support

- Support for the optional Cryptographic Extension, which is licensed separately

- Activity Monitoring Unit (AMU)

- Error protection on L1 instruction and data caches, L2 cache, and MMU Translation Cache (MMU TC) with parity or Error Correcting Code (ECC) allowing Single Error Correction and Double Error Detection (SECDED).

- Support for Memory System Resource Partitioning and Monitoring (MPAM)

- Armv9-A debug logic

- Embedded Trace Extension (ETE)

- Trace Buffer Extension (TRBE)

- Statistical Profiling Extension (SPE)

- Optional Embedded Logic Analyzer (ELA)

- Random Number Generator introduced in the Armv8.3-A extension.

- The Data Gathering Hint instruction introduced in the Armv8.6-A extension.

- Support for nested virtualization introduced in the Armv8.3-A and Armv8.4-A extensions

- Support for coherent instruction cache

This document describes elements of the Neoverse V2 core micro-architecture that influence software performance so that software and compilers can be optimized accordingly.

2.1 Pipeline overview

The following figure describes the high-level Neoverse V2 core instruction processing pipeline. Instructions are first fetched and then decoded into internal Macro-OPerations (MOPs). From there, the MOPs proceed through register renaming and dispatch stages. A MOP can be split into two Micro-OPerations (µOPs) further down the pipeline after the decode stage. Once dispatched, µOPs wait for their operands and issue out-of-order to one of seventeen issue pipelines. Each issue pipeline can accept one µOP per cycle.

Figure 2-1 Neoverse V2 core pipeline
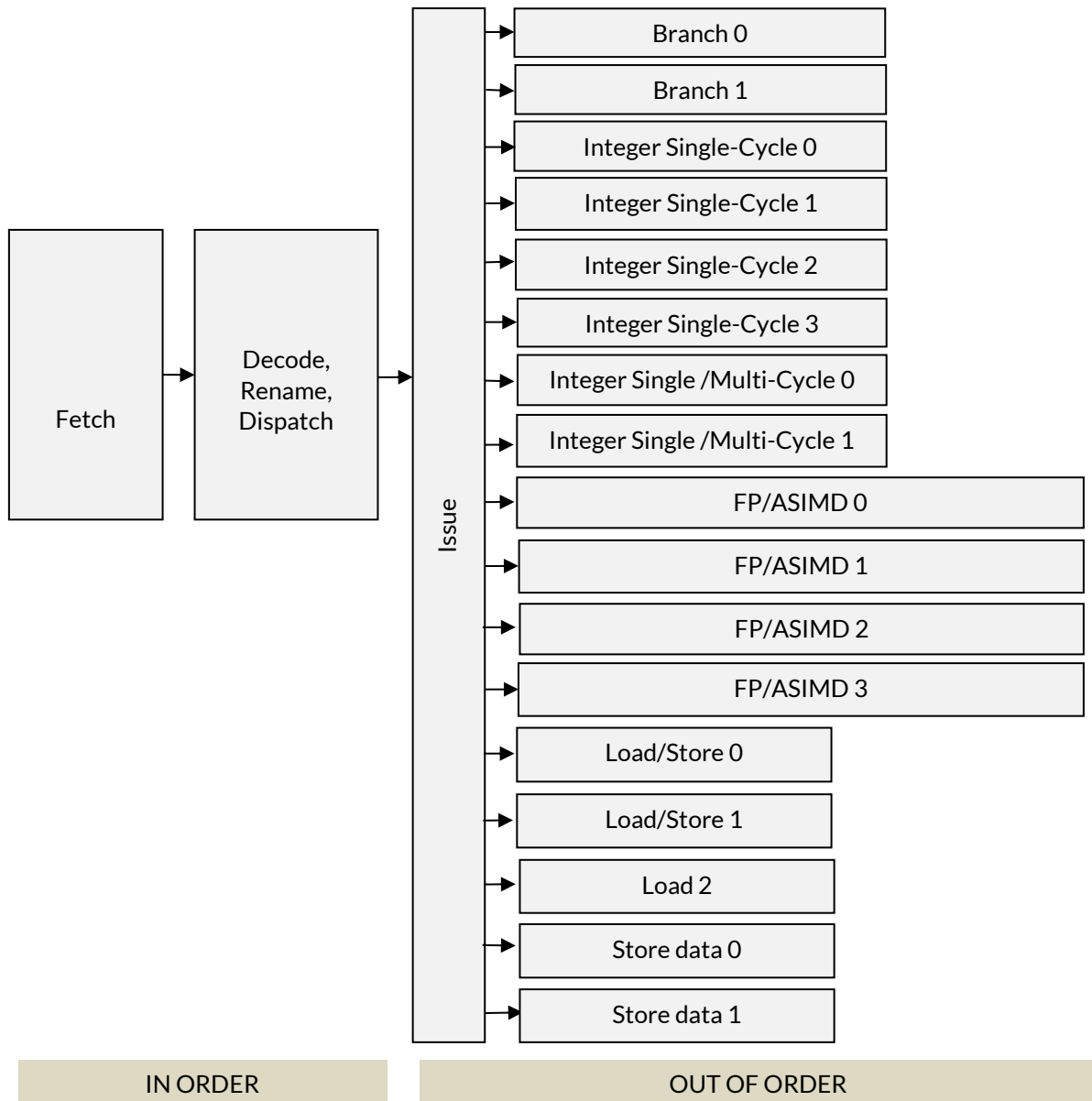
The execution pipelines support different types of operations, as shown in the following table.

Table 2-1 Neoverse V2 core operations

Instruction groups	Instructions
Branch 0/1	Branch µOPs
Integer Single-Cycle 0/1	Integer ALU µOps
Integer Single-Cycle 2/3	Integer ALU µOPs that do not read PC or write flags
Integer Single/Multi-cycle 0/1	Integer ALU, integer shift-ALU, multiply, divide, CRC µOPs
Load/Store 0/1	Load, Store address generation and special memory µOPs
Load 2	Load µOPs
Store data 0/1	Store data µOPs
FP/ASIMD-0	ASIMD ALU, ASIMD misc, ASIMD integer multiply, FP convert, FP misc, FP add, FP multiply, FP divide, FP sqrt, crypto µOPs, store data µOPs
FP/ASIMD-1	ASIMD ALU, ASIMD misc, FP misc, FP add, FP multiply, ASIMD shift µOPs, store data µOPs, crypto µOPs.
FP/ASIMD-2	ASIMD ALU, ASIMD misc, ASIMD integer multiply, FP convert, FP misc, FP add, FP multiply, FP divide, FP sqrt, crypto µOPs.
FP/ASIMD-3	ASIMD ALU, ASIMD misc, FP misc, FP add, FP multiply, ASIMD shift µOPs, crypto µOPs

3 Instruction characteristics

3.1 Instruction tables

This chapter describes high-level performance characteristics for most Armv9-A instructions. A series of tables summarize the effective execution latency and throughput (instruction bandwidth per cycle), pipelines utilized, and special behaviors associated with each group of instructions. Utilized pipelines correspond to the execution pipelines described in chapter 2.

In the tables below, Exec Latency is defined as the minimum latency seen by an operation dependent on an instruction in the described group.

In the tables below, Execution Throughput is defined as the maximum throughput (in instructions per cycle) of the specified instruction group that can be achieved in the entirety of the Neoverse V2 core microarchitecture.

3.2 Legend for reading the utilized pipelines

Table 3-1 Neoverse V2 core pipeline names and symbols

Pipeline name	Symbol used in tables
Branch 0/1	B
Integer single cycle 0/1	R
Integer single cycle 0/1/2/3	S
Integer single cycle 0/1 and single/multicycle 0/1	F
Integer single cycle 0/1/2/3 and single/multicycle 0/1	I
Integer single/multicycle 0/1	M
Integer multicycle 0	M0
Load/Store 01	L01
Load/Store 0/1 and Load 2	L
Store data 0/1	D
FP/ASIMD 0/1/2/3	V
FP/ASIMD 0/1	V01
FP/ASIMD 0/2	V02
FP/ASIMD 1/3	V13
FP/ASIMD 0	V0
FP/ASIMD 1	V1

3.3 Branch instructions

Table 3-2 AArch64 Branch instructions

Instruction Group	AArch64 Instructions	Exec Latency	Execution Throughput	Utilized Pipelines	Notes
Branch, immed	B	1	2	B	-
Branch, register	BR, RET	1	2	B	-
Branch and link, immed	BL	1	2	B, R	-
Branch and link, register	BLR	1	2	B, R	-
Compare and branch	CBZ, CBNZ, TBZ, TBNZ	1	2	B	-

3.4 Arithmetic and logical instructions

Table 3-3 AArch64 Arithmetic and logical instructions

Instruction Group	AArch64 Instructions	Exec Latency	Execution Throughput	Utilized Pipelines	Notes
ALU, basic	ADD, ADC, AND, BIC, EON, EOR, ORN, ORR, SUB, SBC	1	6	I	-
ALU, basic, flagset	ADDS, ADCS, ANDS, BICS, SUBS, SBCS	1	3	F	-
ALU, extend and shift	ADD{S}, SUB{S}	2	2	M	-
Arithmetic, LSL shift, shift <= 4	ADD, SUB	1	6	I	-
Arithmetic, flagset, LSL shift, shift <= 4	ADDS, SUBS	1	3	F	-
Arithmetic, LSR/ASR/ROR shift or LSL shift > 4	ADD{S}, SUB{S}	2	2	M	-
Arithmetic, immediate to logical address tag	ADDG, SUBG	2	2	M	-
Conditional compare	CCMN, CCMP	1	3	F	-
Conditional select	CSEL, CSINC, CSINV, CSNEG	1	6	I	-
Convert floating-point condition flags	AXFLAG, XAFLAG	1	1	F	-
Flag manipulation instructions	SETF8, SETF16, RMIF, CFINV	1	1	F	-
Insert Random Tag	IRG	2, 3	2, 1	M, M0	1
Insert Tag Mask	GMI	1	6	I	-

Instruction Group	AArch64 Instructions	Exec Latency	Execution Throughput	Utilized Pipelines	Notes
Logical, shift, no flagset	AND, BIC, EON, EOR, ORN, ORR	1	6	I	-
Logical, shift, flagset	ANDS, BICS	2	2	M	-
Subtract Pointer	SUBP	1	6	I	-
Subtract Pointer, flagset	SUBPS	1	3	F	-

Notes:

1. The latency is 2, throughput is 2 and utilized pipeline is M when GCR_EL1.RRND = 1. When GCR_EL1.RRND = 0, latency is 3, throughput is 1 and pipeline utilized is M0.

3.5 Divide and multiply instructions

Table 3-4 AArch64 Divide and multiply instructions

Instruction Group	AArch64 Instructions	Exec Latency	Execution Throughput	Utilized Pipelines	Notes
Divide, W-form	SDIV, UDIV	5 to 12	1/12 to 1/5	M0	1
Divide, X-form	SDIV, UDIV	5 to 20	1/20 to 1/5	M0	1
Multiply	MUL, MNEG	2	2	M	-
Multiply accumulate, W-form	MADD, MSUB	2(1)	1	M0	2
Multiply accumulate, X-form	MADD, MSUB	2(1)	1	M0	2
Multiply accumulate long	SMADDL, SMSUBL, UMADDL, UMSUBL	2(1)	1	M0	2
Multiply high	SMULH, UMULH	3	2	M	2
Multiply long	SMNEGL, SMULL, UMNEGL, UMULL	2	2	M	-

Notes:

1. Integer divides are performed using an iterative algorithm and block any subsequent divide operations until complete. Early termination is possible, depending upon the data values.

2. Multiply-accumulate pipelines support late-forwarding of accumulate operands from similar µOPs, allowing a typical sequence of multiply-accumulate µOPs to issue one every N cycles (accumulate latency N shown in parentheses). Accumulator forwarding is not supported for consumers of 64 bit multiply high operations.

3.6 Pointer Authentication Instructions

Table 3-5 AArch64 pointer authentication instructions

Instruction Group	AArch64 Instructions	Exec Latency	Execution Throughput	Utilized Pipelines	Notes
Authenticate data address	AUTDA, AUTDB, AUTDZA, AUTDZB	5	1	M0	-
Authenticate instruction address	AUTIA, AUTIB, AUTIA1716, AUTIB1716, AUTIASP, AUTIBSP, AUTIAZ, AUTIBZ, AUTIZA, AUTIZB	5	1	M0	-
Branch and link, register, with pointer authentication	BLRAA, BLRAAZ, BLRAB, BLRABZ	6	1	M0, B	-
Branch, register, with pointer authentication	BRAA, BRAAZ, BRAB, BRABZ	6	1	M0, B	-
Branch, return, with pointer authentication	RETA, RETB	6	1	M0, B	-
Compute pointer authentication code for data address	PACDA, PACDB, PACDZA, PACDZB	5	1	M0	-
Compute pointer authentication code, using generic key	PACGA	5	1	M0	-
Compute pointer authentication code for instruction address	PACIA, PACIB, PACIA1716, PACIB1716, PACIASP, PACIBSP, PACIAZ, PACIBZ, PACIZA, PACIZB	5	1	M0	-
Load register, with pointer authentication	LDRAA, LDRAB	9	1	M0, L	-
Strip pointer authentication code	XPACD, XPACI, XPACLRI	2	1	M0	-

3.7 Miscellaneous data-processing instructions

Table 3-6 AArch64 Miscellaneous data-processing instructions

Instruction Group	AArch64 Instructions	Exec Latency	Execution Throughput	Utilized Pipelines	Notes
Address generation	ADR, ADRP	1	4	F	-
Bitfield extract, one reg	EXTR	1	6	I	-
Bitfield extract, two regs	EXTR	3	2	I, M	-
Bitfield move, basic	SBFM, UBFM	1	6	I	-
Bitfield move, insert	BFM	2	2	M	-
Count leading	CLS, CLZ	1	6	I	-
Move immed	MOVN, MOVK, MOVZ	1	6	I	-
Reverse bits/bytes	RBIT, REV, REV16, REV32	1	6	I	-
Variable shift	ASRV, LSLV, LSRV, RORV	1	6	I	-

3.8 Load instructions

The latencies shown assume the memory access hits in the Level 1 Data Cache and represent the maximum latency to load all the registers written by the instruction.

Table 3-7 AArch64 Load instructions

Instruction Group	AArch64 Instructions	Exec Latency	Execution Throughput	Utilized Pipelines	Notes
Load register, literal	LDR, LDRSW, PRFM	5	3	L, F	-
Load register, unscaled immed	LDUR, LDURB, LDURH, LDURSB, LDURSH, LDURSW, PRFUM	4	3	L	-
Load register, immed post-index	LDR, LDRB, LDRH, LDRSB, LDRSH, LDRSW	4	3	L, I	-
Load register, immed pre-index	LDR, LDRB, LDRH, LDRSB, LDRSH, LDRSW	4	3	L, I	-
Load register, immed unprivileged	LDTR, LDTRB, LDTRH, LDTRSB, LDTRSH, LDTRSW	4	3	L	-

Instruction Group	AArch64 Instructions	Exec Latency	Execution Throughput	Utilized Pipelines	Notes
Load register, unsigned immed	LDR, LDRB, LDRH, LDRSB, LDRSH, LDRSW, PRFM	4	3	L	-
Load register, register offset, basic	LDR, LDRB, LDRH, LDRSB, LDRSH, LDRSW, PRFM	4	3	L	-
Load register, register offset, scale by 4/8	LDR, LDRSW, PRFM	4	3	L	-
Load register, register offset, scale by 2	LDRH, LDRSH	4	3	L	-
Load register, register offset, extend	LDR, LDRB, LDRH, LDRSB, LDRSH, LDRSW, PRFM	4	3	L	-
Load register, register offset, extend, scale by 4/8	LDR, LDRSW, PRFM	4	3	L	-
Load register, register offset, extend, scale by 2	LDRH, LDRSH	4	3	L	-
Load pair, signed immed offset, normal, W-form	LDP, LDNP	4	3	L	-
Load pair, signed immed offset, normal, X-form	LDP, LDNP	4	2	L	-
Load pair, signed immed offset, signed words	LDPSW	5	1	I, L	-
Load pair, immed post-index or immed pre-index, normal, W-form	LDP	4	3	L, I	-
Load pair, immed post-index or immed pre-index, normal, X-form	LDP	4	2	L, I	-
Load pair, immed post-index or immed pre-index, signed words	LDPSW	5	1	I, L	-

3.9 Store instructions

The following table describes performance characteristics for standard store instructions. Stores µOPs are split into address and data µOPs. Once executed, stores are buffered and committed in the background.

Table 3-8 AArch64 Store instructions

Instruction Group	AArch64 Instructions	Exec Latency	Execution Throughput	Utilized Pipelines	Notes
Store register, unscaled immed	STUR, STURB, STURH	1	2	L01, D	-
Store register, immed post-index	STR, STRB, STRH	1	2	L01, D, I	-
Store register, immed pre-index	STR, STRB, STRH	1	2	L01, D, I	-
Store register, immed unprivileged	STTR, STTRB, STTRH	1	2	L01, D	-
Store register, unsigned immed	STR, STRB, STRH	1	2	L01, D	-
Store register, register offset, basic	STR, STRB, STRH	1	2	L01, D	-
Store register, register offset, scaled by 4/8	STR	1	2	L01, D	-
Store register, register offset, scaled by 2	STRH	1	2	I, L01, D	-
Store register, register offset, extend	STR, STRB, STRH	1	2	L01, D	-
Store register, register offset, extend, scale by 4/8	STR	1	2	L01, D	-
Store register, register offset, extend, scale by 2	STRH	1	2	I, L01, D	-
Store pair, immed offset	STP, STNP	1	2	L01, D	-
Store pair, immed post-index	STP	1	2	L01, D, I	-
Store pair, immed pre-index	STP	1	2	L01, D, I	-

3.10 Tag Load Instructions

Table 3-9 AArch64 Tag load instructions

Instruction Group	AArch64 Instructions	Exec Latency	Execution Throughput	Utilized Pipelines	Notes
Load allocation tag	LDG	4	3	L	-
Load multiple allocation tags	LDGM	4	3	L	-

3.11 Tag Store instructions

Table 3-10 AArch64 Tag store instructions

Instruction Group	AArch64 Instructions	Exec Latency	Execution Throughput	Utilized Pipelines	Notes
Store allocation tags to one or two granules, post-index	STG, ST2G	1	2	L01, D, I	-
Store allocation tags to one or two granules, pre-index	STG, ST2G	1	2	L01, D, I	-
Store allocation tags to one or two granules, signed offset	STG, ST2G	1	2	L01, D	-
Store allocation tag to one or two granules, zeroing, post-index	STZG, STZ2G	1	2	L01, D, I	-
Store Allocation Tag to one or two granules, zeroing, pre-index	STZG, STZ2G	1	2	L01, D, I	-
Store allocation tag to two granules, zeroing, signed offset	STZG, STZ2G	1	2	L01, D	-
Store allocation tag and reg pair to memory, post-Index	STGP	1	2	L01, D, I	-
Store allocation tag and reg pair to memory, pre-Index	STGP	1	2	L01, D, I	-
Store allocation tag and reg pair to memory, signed offset	STGP	1	2	L01, D	-
Store multiple allocation tags	STGM	1	2	L01, D	-
Store multiple allocation tags, zeroing	STZGM	1	2	L01, D	-

3.12 FP data processing instructions

Table 3-11 AArch64 FP data processing instructions

Instruction Group	AArch64 Instructions	Exec Latency	Execution Throughput	Utilized Pipelines	Notes
FP absolute value	FABS	2	4	V	-
FP arithmetic	FADD, FSUB	2	4	V	-
FP compare	FCCMP{E}, FCMP{E}	2	1	V0	-
FP divide, H-form	FDIV	7	8/7	V02	1
FP divide, S-form	FDIV	7 to 10	8/9 to 8/7	V02	1
FP divide, D-form	FDIV	7 to 15	2/7 to 4/7	V02	1
FP min/max	FMIN, FMINNM, FMAX, FMAXNM	2	4	V	-

Instruction Group	AArch64 Instructions	Exec Latency	Execution Throughput	Utilized Pipelines	Notes
FP multiply	FMUL, FNMUL	3	4	V	2
FP multiply accumulate	FMADD, FMSUB, FNMADD, FNMSUB	4 (2)	4	V	3
FP negate	FNEG	2	4	V	-
FP round to integral	FRINTA, FRINTI, FRINTM, FRINTN, FRINTP, FRINTX, FRINTZ, FRINT32X, FRINT64X, FRINT32Z, FRINT64Z	3	2	V02	-
FP select	FCSEL	2	4	V	-
FP square root, H-form	FSQRT	7	8/7	V02	1
FP square root, S-form	FSQRT	7 to 9	1 to 8/7	V02	1
FP square root, D-form	FSQRT	7 to 16	4/15 to 4/7	V02	1

Notes:

1. FP divide and square root operations are performed using an iterative algorithm and block subsequent similar operations to the same pipeline until complete.

2. FP multiply-accumulate pipelines support late forwarding of the result from FP multiply µOPs to the accumulate operands of an FP multiply-accumulate µOP. The latter can potentially be issued 1 cycle after the FP multiply µOP has been issued.

3. FP multiply-accumulate pipelines support late-forwarding of accumulate operands from similar µOPs, allowing a typical sequence of multiply-accumulate µOPs to issue one every N cycles(accumulate latency N shown in parentheses).

3.13 FP miscellaneous instructions

Table 3-12 AArch64 FP miscellaneous instructions

Instruction Group	AArch64 Instructions	Exec Latency	Execution Throughput	Utilized Pipelines	Notes
FP convert, from gen to vec reg	SCVTF, UCVTF	3	1	M0	-
FP convert, from vec to gen reg	FCVTAS, FCVTAU, FCVTMS, FCVTMU, FCVTNS, FCVTNU, FCVTPS, FCVTPU, FCVTZS, FCVTZU	3	1	V01	-
FP convert, Javascript from vec to gen reg	FJCVTZS	3	1	V0	-
FP convert, from vec to vec reg	FCVT, FCVTXN	3	2	V02	-
FP move, immed	FMOV	2	4	V	-
FP move, register	FMOV	2	4	V	-
FP transfer, from gen to low half of vec reg	FMOV	3	1	M0	-
FP transfer, from gen to high half of vec reg	FMOV	5	1	M0, V	
FP transfer, from vec to gen reg	FMOV	2	1	V01	-

3.14 FP load instructions

The latencies shown assume the memory access hits in the Level 1 Data Cache. Compared to standard loads, an extra cycle is required to forward results to FP/ASIMD pipelines.

Table 3-13 AArch64 FP load instructions

Instruction Group	AArch64 Instructions	Exec Latency	Execution Throughput	Utilized Pipelines	Notes
Load vector reg, literal, S/D/Q forms	LDR	7	3	F, L	-
Load vector reg, unscaled immed	LDUR	6	3	L	-
Load vector reg, immed post-index	LDR	6	3	L, I	-
Load vector reg, immed pre-index	LDR	6	3	L, I	-
Load vector reg, unsigned immed	LDR	6	3	L	-
Load vector reg, register offset, basic	LDR	6	3	L	-
Load vector reg, register offset, scale, S/D-form	LDR	6	3	L	-
Load vector reg, register offset, scale, H/Q-form	LDR	7	3	I, L	-
Load vector reg, register offset, extend	LDR	6	3	L	-
Load vector reg, register offset, extend, scale, S/D-form	LDR	6	3	L	-
Load vector reg, register offset, extend, scale, H/Q-form	LDR	7	3	I, L	-
Load vector pair, immed offset, S/D-form	LDP, LDNP	6	3	L	-
Load vector pair, immed offset, Q-form	LDP, LDNP	6	3/2	L	-
Load vector pair, immed post-index, S/D-form	LDP	6	3	I, L	-
Load vector pair, immed post-index, Q-form	LDP	6	3/2	L, I	-
Load vector pair, immed pre-index, S/D-form	LDP	6	3	I, L	-
Load vector pair, immed pre-index, Q-form	LDP	6	3/2	L, I	-

3.15 FP store instructions

Stores MOPs are split into store address and store data μOPs. Once executed, stores are buffered and committed in the background.

Table 3-14 AArch64 FP store instructions

Instruction Group	AArch64 Instructions	Exec Latency	Execution Throughput	Utilized Pipelines	Notes
Store vector reg, unscaled immed, B/H/S/D-form	STUR	2	2	L01, V01	-
Store vector reg, unscaled immed, Q-form	STUR	2	2	L01, V01	-
Store vector reg, immed post-index, B/H/S/D-form	STR	2	2	L01, V01, I	-
Store vector reg, immed post-index, Q-form	STR	2	2	L01, V01, I	-
Store vector reg, immed pre-index, B/H/S/D-form	STR	2	2	L01, V01, I	-
Store vector reg, immed pre-index, Q-form	STR	2	2	L01, V01, I	-
Store vector reg, unsigned immed, B/H/S/D-form	STR	2	2	L01, V01	-
Store vector reg, unsigned immed, Q-form	STR	2	2	L01, V01	-
Store vector reg, register offset, basic, B/H/S/D-form	STR	2	2	L01, V01	-
Store vector reg, register offset, basic, Q-form	STR	2	2	L01, V01	-
Store vector reg, register offset, scale, H-form	STR	2	2	I, L01, V01	-
Store vector reg, register offset, scale, S/D-form	STR	2	2	L01, V01	-
Store vector reg, register offset, scale, Q-form	STR	2	2	I, L01, V01	-
Store vector reg, register offset, extend, B/H/S/D-form	STR	2	2	L01, V01	-
Store vector reg, register offset, extend, Q-form	STR	2	2	L01, V01	-
Store vector reg, register offset, extend, scale, H-form	STR	2	2	I, L01, V01	-
Store vector reg, register offset, extend, scale, S/D-form	STR	2	2	L01, V01	-
Store vector reg, register offset, extend, scale, Q-form	STR	2	2	I, L01, V01	-
Store vector pair, immed offset, S-form	STP, STNP	2	2	L01, V01	-

Instruction Group	AArch64 Instructions	Exec Latency	Execution Throughput	Utilized Pipelines	Notes
Store vector pair, immed offset, D-form	STP, STNP	2	2	L01, V01	-
Store vector pair, immed offset, Q-form	STP, STNP	2	1	L01, V01	-
Store vector pair, immed post-index, S-form	STP	2	2	I, L01, V01	-
Store vector pair, immed post-index, D-form	STP	2	2	I, L01, V01	-
Store vector pair, immed post-index, Q-form	STP	2	1	I, L01, V01	-
Store vector pair, immed pre-index, S-form	STP	2	2	I, L01, V01	-
Store vector pair, immed pre-index, D-form	STP	2	2	I, L01, V01	-
Store vector pair, immed pre-index, Q-form	STP	2	1	I, L01, V01	-

3.16 ASIMD integer instructions

Table 3-15 AArch64 ASIMD integer instructions

Instruction Group	AArch64 Instructions	Exec Latency	Execution Throughput	Utilized Pipelines	Notes
ASIMD absolute diff	SABD, UABD	2	4	V	-
ASIMD absolute diff accum	SABA, UABA	4(1)	2	V13	2
ASIMD absolute diff accum long	SABAL(2), UABAL(2)	4(1)	2	V13	2
ASIMD absolute diff long	SABDL(2), UABDL(2)	2	4	V	-
ASIMD arith, basic	ABS, ADD, NEG, SADDL(2), SADDW(2), SHADD, SHSUB, SSUBL(2), SSUBW(2), SUB, UADDL(2), UADDW(2), UHADD, UHSUB, USUBL(2), USUBW(2)	2	4	V	-

Instruction Group	AArch64 Instructions	Exec Latency	Execution Throughput	Utilized Pipelines	Notes
ASIMD arith, complex	ADDHN(2), RADDHN(2), RSUBHN(2), SQABS, SQADD, SQNEG, SQSUB, SRHADD, SUBHN(2), SUQADD, UQADD, UQSUB, URHADD, USQADD	2	4	V	-
ASIMD arith, pair-wise	ADDP, SADDLP, UADDLP	2	4	V	-
ASIMD arith, reduce, 4H/4S	ADDV, SADDLV, UADDLV	2	2	V13	-
ASIMD arith, reduce, 8B/8H	ADDV, SADDLV, UADDLV	4	2	V13, V	-
ASIMD arith, reduce, 16B	ADDV, SADDLV, UADDLV	4	1	V13	-
ASIMD compare	CMEQ, CMGE, CMGT, CMHI, CMHS, CMLE, CMLT, CMTST	2	4	V	-
ASIMD dot product	SDOT, UDOT	3 (1)	4	V	2
ASIMD dot product using signed and unsigned integers	SUDOT, USDOT	3(1)	4	V	2
ASIMD logical	AND, BIC, EOR, MOV, MVN, NOT, ORN, ORR	2	4	V	-
ASIMD matrix multiply-accumulate	SMMLA, UMMLA, USMMLA	3(1)	4	V	2
ASIMD max/min, basic and pair-wise	SMAX, SMAXP, SMIN, SMINP, UMAX, UMAXP, UMIN, UMINP	2	4	V	-
ASIMD max/min, reduce, 4H/4S	SMAXV, SMINV, UMAXV, UMINV	2	2	V13	-
ASIMD max/min, reduce, 8B/8H	SMAXV, SMINV, UMAXV, UMINV	4	2	V13, V	-
ASIMD max/min, reduce, 16B	SMAXV, SMINV, UMAXV, UMINV	4	1	V13	-
ASIMD multiply	MUL, SQDMULH, SQRDMULH	4	2	V02	-
ASIMD multiply accumulate	MLA, MLS	4(1)	2	V02	1
ASIMD multiply accumulate high	SQRDMLAH, SQRDMLSH	4(2)	1	V02	-

Instruction Group	AArch64 Instructions	Exec Latency	Execution Throughput	Utilized Pipelines	Notes
ASIMD multiply accumulate long	SMLAL(2), SMLSL(2), UMLAL(2), UMLSL(2)	4(1)	2	V02	1
ASIMD multiply accumulate saturating long	SQDMLAL(2), SQDMLSL(2)	4	2	V02	-
ASIMD multiply/multiply long (8x8) polynomial, D-form	PMUL, PMULL(2)	3	2	V23	3
ASIMD multiply/multiply long (8x8) polynomial, Q-form	PMUL, PMULL(2)	3	2	V23	3
ASIMD multiply long	SMULL(2), UMULL(2), SQDMULL(2)	3	2	V02	-
ASIMD pairwise add and accumulate long	SADALP, UADALP	4(1)	2	V13	2
ASIMD shift accumulate	SSRA, SRSRA, USRA, URSRA	4(1)	2	V13	2
ASIMD shift by immed, basic	SHL, SHLL(2), SHRN(2), SSHLL(2), SSHR, SXTL(2), USHLL(2), USHR, UXTL(2)	2	2	V13	-
ASIMD shift by immed and insert, basic	SLI, SRI	2	2	V13	-
ASIMD shift by immed, complex	RSHRN(2), SQRSHRN(2), SQRSHRUN(2), SQSHL{U}, SQSHRN(2), SQSHRUN(2), SRSHR, UQRSHRN(2), UQSHL, UQSHRN(2), URSHR	4	2	V13	-
ASIMD shift by register, basic	SSHL, USHL	2	2	V13	-
ASIMD shift by register, complex	SRSHL, SQRSHL, SQSHL, URSHL, UQRSHL, UQSHL	4	2	V13	-

Notes:

1. Multiply-accumulate pipelines support late-forwarding of accumulate operands from similar µOPs, allowing a typical sequence of integer multiply-accumulate µOPs to issue one every cycle or one every other cycle (accumulate latency shown in parentheses).

2. Other accumulate pipelines also support late-forwarding of accumulate operands from similar µOPs, allowing a typical sequence of such µOPs to issue one every cycle (accumulate latency shown in parentheses).

3. This category includes instructions of the form "PMULL Vd.8H, Vn.8B, Vm.8B" and "PMULL2 Vd.8H, Vn.16B, Vm.16B".

3.17 ASIMD floating-point instructions

Table 3-16 AArch64 ASIMD floating-point instructions

Instruction Group	AArch64 Instructions	Exec Latency	Execution Throughput	Utilized Pipelines	Notes
ASIMD FP absolute value/difference	FABS, FABD	2	4	V	-
ASIMD FP arith, normal	FADD, FSUB, FADDP	2	4	V	-
ASIMD FP compare	FACGE, FACGT, FCMEQ, FCMGE, FCMGT, FCMLE, FCMLT	2	4	V	-
ASIMD FP complex add	FCADD	2	4	V	-
ASIMD FP complex multiply add	FCMLA	4(2)	4	V	1
ASIMD FP convert, long (F16 to F32)	FCVTL(2)	4	1	V02	-
ASIMD FP convert, long (F32 to F64)	FCVTL(2)	3	2	V02	-
ASIMD FP convert, narrow (F32 to F16)	FCVTN(2)	4	1	V02	-
ASIMD FP convert, narrow (F64 to F32)	FCVTN(2), FCVTXN(2)	3	2	V02	-
ASIMD FP convert, other, D-form F32 and Q-form F64	FCVTAS, FCVTAU, FCVTMS, FCVTMU, FCVTNS, FCVTNU, FCVTPS, FCVTPU, FCVTZS, FCVTZU, SCVTF, UCVTF	3	2	V02	-
ASIMD FP convert, other, D-form F16 and Q-form F32	FCVTAS, FCVTAU, FCVTMS, FCVTMU, FCVTNS, FCVTNU, FCVTPS, FCVTPU, FCVTZS, FCVTZU, SCVTF, UCVTF	4	1	V02	-

Instruction Group	AArch64 Instructions	Exec Latency	Execution Throughput	Utilized Pipelines	Notes
ASIMD FP convert, other, Q-form F16	FCVTAS, FCVTAU, FCVTMS, FCVTMU, FCVTNS, FCVTNU, FCVTPS, FCVTPU, FCVTZS, FCVTZU, SCVTF, UCVTF	6	1/2	V02	-
ASIMD FP divide, D-form, F16	FDIV	7	2/7	V02	3
ASIMD FP divide, D-form, F32	FDIV	7 to 10	4/9 to 4/7	V02	3
ASIMD FP divide, Q-form, F16	FDIV	10 to 13	2/13 to 1/5	V02	3
ASIMD FP divide, Q-form, F32	FDIV	7 to 10	2/9 to 2/7	V02	3
ASIMD FP divide, Q-form, F64	FDIV	7 to 15	1/7 to 2/7	V02	3
ASIMD FP max/min, normal	FMAX, FMAXNM, FMIN, FMINNM	2	4	V	-
ASIMD FP max/min, pairwise	FMAXP, FMAXNMP, FMINP, FMINNMP	2	4	V	-
ASIMD FP max/min, reduce, F32 and D-form F16	FMAXV, FMAXNMV, FMINV, FMINNMV	4	2	V	-
ASIMD FP max/min, reduce, Q-form F16	FMAXV, FMAXNMV, FMINV, FMINNMV	6	4/3	V	-
ASIMD FP multiply	FMUL, FMULX	3	4	V	2
ASIMD FP multiply accumulate	FMLA, FMLS	4(2)	4	V	1
ASIMD FP multiply accumulate long	FMLAL(2), FMLSL(2)	4(2)	4	V	1
ASIMD FP negate	FNEG	2	4	V	-
ASIMD FP round, D-form F32 and Q-form F64	FRINTA, FRINTI, FRINTM, FRINTN, FRINTP, FRINTX, FRINTZ, FRINT32X, FRINT64X, FRINT32Z, FRINT64Z	3	2	V02	-

Instruction Group	AArch64 Instructions	Exec Latency	Execution Throughput	Utilized Pipelines	Notes
ASIMD FP round, D-form F16 and Q-form F32	FRINTA, FRINTI, FRINTM, FRINTN, FRINTP, FRINTX, FRINTZ, FRINT32X, FRINT64X, FRINT32Z, FRINT64Z	4	1	V02	-
ASIMD FP round, Q-form F16	FRINTA, FRINTI, FRINTM, FRINTN, FRINTP, FRINTX, FRINTZ	6	1/2	V02	-
ASIMD FP square root, D-form, F16	FSQRT	7	2/7	V02	3
ASIMD FP square root, D-form, F32	FSQRT	7 to 10	4/9 to 4/7	V02	3
ASIMD FP square root, Q-form, F16	FSQRT	11 to 13	2/13 to 2/11	V02	3
ASIMD FP square root, Q-form, F32	FSQRT	7 to 10	2/9 to 2/7	V02	3
ASIMD FP square root, Q-form, F64	FSQRT	7 to 16	2/15 to 2/7	V02	3

Notes:

1. ASIMD multiply-accumulate pipelines support late-forwarding of accumulate operands from similar µOPs, allowing a typical sequence of floating-point multiply-accumulate µOPs to issue one every N cycles (accumulate latency N shown in parentheses).

2. ASIMD multiply-accumulate pipelines support late forwarding of the result from ASIMD FP multiply µOPs to the accumulate operands of an ASIMD FP multiply-accumulate µOP. The latter can potentially be issued 1 cycle after the ASIMD FP multiply µOP has been issued.

3. ASIMD divide and square root operations are performed using an iterative algorithm and block subsequent similar operations to the same pipeline until complete.

3.18 ASIMD BFloat16 (BF16) instructions

Table 3-17 AArch64 ASIMD BFloat (BF16) instructions

Instruction Group	AArch64 Instructions	Exec Latency	Execution Throughput	Utilized Pipelines	Notes
ASIMD convert, F32 to BF16	BFCVTN, BFCVTN2	4	1	V02	-
ASIMD dot product	BFDOT	5(3)	4	V	1
ASIMD matrix multiply accumulate	BFMMLA	6(4)	4	V	1
ASIMD multiply accumulate long	BFMLALB, BFMLALT	5(2)	4	V	1
Scalar convert, F32 to BF16	BFCVT	3	2	V02	-

3.19 ASIMD miscellaneous instructions

Table 3-18 AArch64 ASIMD miscellaneous instructions

Instruction Group	AArch64 Instructions	Exec Latency	Execution Throughput	Utilized Pipelines	Notes
ASIMD bit reverse	RBIT	2	4	V	-
ASIMD bitwise insert	BIF, BIT, BSL	2	4	V	-
ASIMD count	CLS, CLZ, CNT	2	4	V	-
ASIMD duplicate, gen reg	DUP	3	1	M0	-
ASIMD duplicate, element	DUP	2	4	V	-
ASIMD extract	EXT	2	4	V	-
ASIMD extract narrow	XTN(2)	2	4	V	-
ASIMD extract narrow, saturating	SQXTN(2), SQXTUN(2), UQXTN(2)	4	2	V13	-
ASIMD insert, element to element	INS	2	4	V	-
ASIMD move, FP immed	FMOV	2	4	V	-
ASIMD move, integer immed	MOVI, MVNI	2	4	V	-
ASIMD reciprocal and square root estimate, D-form U32	URECPE, URSQRTE	3	2	V02	-
ASIMD reciprocal and square root estimate, Q-form U32	URECPE, URSQRTE	4	1	V02	-
ASIMD reciprocal and square root estimate, D-form F32 and scalar forms	FRECPE, FRSQRTE	3	2	V02	-

Instruction Group	AArch64 Instructions	Exec Latency	Execution Throughput	Utilized Pipelines	Notes
ASIMD reciprocal and square root estimate, D-form F16 and Q-form F32	FRECPE, FRSQRTE	4	1	V02	-
ASIMD reciprocal and square root estimate, Q-form F16	FRECPE, FRSQRTE	6	1/2	V02	-
ASIMD reciprocal exponent	FRECPX	3	2	V02	
ASIMD reciprocal step	FRECPS, FRSQRTS	4	4	V	-
ASIMD reverse	REV16, REV32, REV64	2	4	V	-
ASIMD table lookup, 1 or 2 table regs	TBL	2	2	V01	-
ASIMD table lookup, 3 table regs	TBL	4	1	V01	-
ASIMD table lookup, 4 table regs	TBL	4	2/3	V01	-
ASIMD table lookup extension, 1 table reg	TBX	2	4	V	-
ASIMD table lookup extension, 2 table reg	TBX	4	2	V	-
ASIMD table lookup extension, 3 table reg	TBX	6	4/3	V	-
ASIMD table lookup extension, 4 table reg	TBX	6	4/5	V	-
ASIMD transfer, element to gen reg	UMOV, SMOV	2	1	V01	-
ASIMD transfer, gen reg to element	INS	5	1	M0, V	-
ASIMD transpose	TRN1, TRN2	2	4	V	-
ASIMD unzip/zip	UZP1, UZP2, ZIP1, ZIP2	2	4	V	-

3.20 ASIMD load instructions

The latencies shown assume the memory access hits in the Level 1 Data Cache and represent the maximum latency to load all the vector registers written by the instruction. Compared to standard loads, an extra cycle is required to forward results to vector pipelines.

Table 3-19 AArch64 ASIMD load instructions

Instruction Group	AArch64 Instructions	Exec Latency	Execution Throughput	Utilized Pipelines	Notes
ASIMD load, 1 element, multiple, 1 reg, D-form	LD1	6	3	L	-
ASIMD load, 1 element, multiple, 1 reg, Q-form	LD1	6	3	L	-
ASIMD load, 1 element, multiple, 2 reg, D-form	LD1	6	3/2	L	-
ASIMD load, 1 element, multiple, 2 reg, Q-form	LD1	6	3/2	L	-
ASIMD load, 1 element, multiple, 3 reg, D-form	LD1	6	1	L	-
ASIMD load, 1 element, multiple, 3 reg, Q-form	LD1	6	1	L	-
ASIMD load, 1 element, multiple, 4 reg, D-form	LD1	7	3/4	L	-
ASIMD load, 1 element, multiple, 4 reg, Q-form	LD1	7	3/4	L	-
ASIMD load, 1 element, one lane, B/H/S	LD1	8	3	L, V	-
ASIMD load, 1 element, one lane, D	LD1	8	3	L, V	-
ASIMD load, 1 element, all lanes, D-form, B/H/S	LD1R	8	3	L, V	-
ASIMD load, 1 element, all lanes, D-form, D	LD1R	8	3	L, V	-
ASIMD load, 1 element, all lanes, Q-form	LD1R	8	3	L, V	-
ASIMD load, 2 element, multiple, D-form, B/H/S	LD2	8	2	L, V	-
ASIMD load, 2 element, multiple, Q-form, B/H/S	LD2	8	3/2	L, V	-
ASIMD load, 2 element, multiple, Q-form, D	LD2	8	3/2	L, V	-
ASIMD load, 2 element, one lane, B/H	LD2	8	2	L, V	-
ASIMD load, 2 element, one lane, S	LD2	8	2	L, V	-

Instruction Group	AArch64 Instructions	Exec Latency	Execution Throughput	Utilized Pipelines	Notes
ASIMD load, 2 element, one lane, D	LD2	8	2	L, V	-
ASIMD load, 2 element, all lanes, D-form, B/H/S	LD2R	8	2	L, V	-
ASIMD load, 2 element, all lanes, D-form, D	LD2R	8	2	L, V	-
ASIMD load, 2 element, all lanes, Q-form	LD2R	8	2	L, V	-
ASIMD load, 3 element, multiple, D-form, B/H/S	LD3	8	4/3	L, V	-
ASIMD load, 3 element, multiple, Q-form, B/H/S	LD3	8	1	L, V	-
ASIMD load, 3 element, multiple, Q-form, D	LD3	8	1	L, V	-
ASIMD load, 3 element, one lane, B/H	LD3	8	4/3	L, V	-
ASIMD load, 3 element, one lane, S	LD3	8	4/3	L, V	-
ASIMD load, 3 element, one lane, D	LD3	8	4/3	L, V	-
ASIMD load, 3 element, all lanes, D-form, B/H/S	LD3R	8	4/3	L, V	-
ASIMD load, 3 element, all lanes, D-form, D	LD3R	8	4/3	L, V	-
ASIMD load, 3 element, all lanes, Q-form, B/H/S	LD3R	8	4/3	L, V	-
ASIMD load, 3 element, all lanes, Q-form, D	LD3R	8	4/3	L, V	-
ASIMD load, 4 element, multiple, D-form, B/H/S	LD4	8	1	L, V	-
ASIMD load, 4 element, multiple, Q-form, B/H/S	LD4	9	1/2	L, V	-
ASIMD load, 4 element, multiple, Q-form, D	LD4	9	1/2	L, V	-
ASIMD load, 4 element, one lane, B/H	LD4	8	1	L, V	-
ASIMD load, 4 element, one lane, S	LD4	8	1	L, V	-
ASIMD load, 4 element, one lane, D	LD4	8	1	L, V	-
ASIMD load, 4 element, all lanes, D-form, B/H/S	LD4R	8	1	L, V	-
ASIMD load, 4 element, all lanes, D-form, D	LD4R	8	1	L, V	-

Instruction Group	AArch64 Instructions	Exec Latency	Execution Throughput	Utilized Pipelines	Notes
ASIMD load, 4 element, all lanes, Q-form, B/H/S	LD4R	8	1	L, V	-
ASIMD load, 4 element, all lanes, Q-form, D	LD4R	8	1	L, V	-
(ASIMD load, writeback form)	-	-	-	I	1

Notes:

1. Writeback forms of load instructions require an extra µOP to update the base address. This update is typically performed in parallel with the load µOP (update latency shown in parentheses).

3.21 ASIMD store instructions

Stores MOPs are split into store address and store data µOPs. Once executed, stores are buffered and committed in the background. The latency represents the maximum latency to store all the vector registers written to memory by the instruction.

Table 3-20 AArch64 ASIMD store instructions

Instruction Group	AArch64 Instructions	Exec Latency	Execution Throughput	Utilized Pipelines	Notes
ASIMD store, 1 element, multiple, 1 reg, D-form	ST1	2	2	L01, V01	-
ASIMD store, 1 element, multiple, 1 reg, Q-form	ST1	2	2	L01, V01	-
ASIMD store, 1 element, multiple, 2 reg, D-form	ST1	2	2	L01, V01	-
ASIMD store, 1 element, multiple, 2 reg, Q-form	ST1	2	1	L01, V01	-
ASIMD store, 1 element, multiple, 3 reg, D-form	ST1	2	1	L01, V01	-
ASIMD store, 1 element, multiple, 3 reg, Q-form	ST1	2	2/3	L01, V01	-
ASIMD store, 1 element, multiple, 4 reg, D-form	ST1	2	1	L01, V01	-
ASIMD store, 1 element, multiple, 4 reg, Q-form	ST1	2	1/2	L01, V01	-
ASIMD store, 1 element, one lane, B/H/S	ST1	4	1	L01, V01	-
ASIMD store, 1 element, one lane, D	ST1	4	1	L01, V01	-
ASIMD store, 2 element, multiple, D-form, B/H/S	ST2	4	1	V01, L01	-
ASIMD store, 2 element, multiple, Q-form, B/H/S	ST2	4	1/2	V01, L01	-

Instruction Group	AArch64 Instructions	Exec Latency	Execution Throughput	Utilized Pipelines	Notes
ASIMD store, 2 element, multiple, Q-form, D	ST2	4	1/2	V01, L01	-
ASIMD store, 2 element, one lane, B/H/S	ST2	4	1	V01, L01	-
ASIMD store, 2 element, one lane, D	ST2	4	1	V01, L01	-
ASIMD store, 3 element, multiple, D-form, B/H/S	ST3	5	1/2	V01, L01	-
ASIMD store, 3 element, multiple, Q-form, B/H/S	ST3	6	1/3	V01, L01	-
ASIMD store, 3 element, multiple, Q-form, D	ST3	6	1/3	V01, L01	-
ASIMD store, 3 element, one lane, B/H	ST3	5	1/2	V01, L01	-
ASIMD store, 3 element, one lane, S	ST3	5	1/2	V01, L01	-
ASIMD store, 3 element, one lane, D	ST3	5	1/2	V01, L01	-
ASIMD store, 4 element, multiple, D-form, B/H/S	ST4	6	1/3	V01, L01	-
ASIMD store, 4 element, multiple, Q-form, B/H/S	ST4	7	1/6	V01, L01	-
ASIMD store, 4 element, multiple, Q-form, D	ST4	5	1/4	V01, L01	-
ASIMD store, 4 element, one lane, B/H/S	ST4	6	2/3	V01, L01	-
ASIMD store, 4 element, one lane, D	ST4	4	1/2	V01, L01	-
(ASIMD store, writeback form)	-	-	-	I	1

Notes:

1. Writeback forms of store instructions require an extra µOP to update the base address. This update is typically performed in parallel with the store µOP (update latency shown in parentheses).

3.22 Cryptography extensions

Table 3-21 AArch64 Cryptography extensions

Instruction Group	AArch64 Instructions	Exec Latency	Execution Throughput	Utilized Pipelines	Notes
Crypto AES ops	AESD, AESE, AESIMC, AESMC	2	4	V	-
Crypto polynomial (64x64) multiply long	PMULL (2)	2	4	V	-
Crypto SHA1 hash acceleration op	SHA1H	2	1	V0	-
Crypto SHA1 hash acceleration ops	SHA1C, SHA1M, SHA1P	4	1	V0	-
Crypto SHA1 schedule acceleration ops	SHA1SU0, SHA1SU1	2	1	V0	-
Crypto SHA256 hash acceleration ops	SHA256H, SHA256H2	4	1	V0	-
Crypto SHA256 schedule acceleration ops	SHA256SU0, SHA256SU1	2	1	V0	-
Crypto SHA512 hash acceleration ops	SHA512H, SHA512H2, SHA512SU0, SHA512SU1	2	1	V0	-
Crypto SHA3 ops	BCAX, EOR3, RAX1, XAR	2	1	V0	-
Crypto SM3 ops	SM3PARTW1, SM3PARTW2, SM3SS1, SM3TT1A, SM3TT1B, SM3TT2A, SM3TT2B	2	1	V0	-
Crypto SM4 ops	SM4E, SM4EKEY	4	1	V0	-

3.23 CRC

Table 3-22 AArch64 CRC

Instruction Group	AArch64 Instructions	Exec Latency	Execution Throughput	Utilized Pipelines	Notes
CRC checksum ops	CRC32, CRC32C	2	1	M0	1

Notes:

1. CRC execution supports late forwarding of the result from a producer µOP to a consumer µOP. This results in a 1 cycle reduction in latency as seen by the consumer.

3.24 SVE Predicate instructions

Table 3-23 SVE Predicate Instructions

Instruction Group	SVE Instruction	Exec Latency	Execution Throughput	Utilized Pipelines	Notes
Loop control, based on predicate	BRKA, BRKB	2	2	M	1
Loop control, based on predicate and flag setting	BRKAS, BRKBS	3	1	M	1
Loop control, propagating	BRKN, BRKPA, BRKPB	2	1	M0	1
Loop control propagating and flag setting	BRKNS, BRKPAS, BRKPBS	3	1	M0, M	1
Loop control, based on GPR	WHILEGE, WHILEGT, WHILEHI, WHILEHS, WHILELE, WHILELO, WHILELS, WHILELT, WHILERW, WHILEWR	3	1	M	-
Loop terminate	CTERMEQ, CTERMNE	1	1	M	-
Predicate counting scalar	ADDPL, ADDVL, CNTB, CNTH, CNTW, CNTD, DECB, DECH, DECW, DECD, INCB, INCH, INCW, INCD, RDVL, SQDECB, SQDECH, SQDECW, SQDECD, SQINCB, SQINCH, SQINCW, SQINCD, UQDECB, UQDECH, UQDECW, UQDECD, UQINCB, UQINCH, UQINCW, UQINCD	2	2	M	-
Predicate counting scalar, ALL, {1,2,4}	INC, DEC	1	4	I	

Instruction Group	SVE Instruction	Exec Latency	Execution Throughput	Utilized Pipelines	Notes
Predicate counting scalar, active predicate	CNTP, DECP, INCP, SQDECP, SQINCP, UQDECP, UQINCP	2	2	M	-
Predicate counting vector, active predicate	DECP, INCP, SQDECP, SQINCP, UQDECP, UQINCP	7	1	M, M0, V	-
Predicate logical	AND, BIC, EOR, MOV, NAND, NOR, NOT, ORN, ORR	1	1	M0	1
Predicate logical, flag setting	ANDS, BICS, EORS, MOV, NANDS, NORS, NOTS, ORNS, ORRS	1	1	M0, M	1
Predicate reverse	REV	2	2	M	-
Predicate select	SEL	1	1	M0	-
Predicate set	PFALSE, PTRUE	2	2	M	-
Predicate set/initialize, set flags	PTRUES	3	1	M	-
Predicate find first, next	PFIRST, PNEXT	2	2	M	-
Predicate test	PTEST	1	2	M	-
Predicate transpose	TRN1, TRN2	2	2	M	-
Predicate unpack and widen	PUNPKHI, PUNPKLO	2	2	M	-
Predicate zip/unzip	ZIP1, ZIP2, UZP1, UZP2	2	2	M	-

Notes:

1. When the governing predicate is the same as destination, the latency is increased by one cycle.

3.25 SVE integer instructions

Table 3-24 SVE integer instructions

Instruction Group	SVE Instruction	Exec Latency	Execution Throughput	Utilized Pipelines	Notes
Arithmetic, absolute diff	SABD, UABD	2	4	V	-
Arithmetic, absolute diff accum	SABA, UABA	4(1)	2	V13	2
Arithmetic, absolute diff accum long	SABALB, SABALT, UABALB, UABALT	4(1)	2	V13	2
Arithmetic, absolute diff long	SABDLB, SABDLT, UABDLB, UABDLT	2	4	V	-
Arithmetic, basic	ABS, ADD, ADR, CNOT, NEG, SADDLB, SADDLBT, SADDLT, SADDWB, SADDWT, SHADD, SHSUB, SHSUBR, SSUBLB, SSUBLBT, SSUBLT, SSUBLTB, SSUBWB, SSUBWT, SUB, SUBR, UADDLB, UADDLT, UADDWB, UADDWT, UHADD, UHSUB, UHSUBR, USUBLB, USUBLT, USUBWB, USUBWT	2	4	V	-

Instruction Group	SVE Instruction	Exec Latency	Execution Throughput	Utilized Pipelines	Notes
Arithmetic, complex	ADDHNB, ADDHNT, RADDHNB, RADDHNT, RSUBHNB, RSUBHNT, SQABS, SQADD, SQNEG, SQSUB, SQSUBR, SRHADD, SUBHNB, SUBHNT, SUQADD, UQADD, UQSUB, UQSUBR, URHADD, USQADD	2	4	V	-
Arithmetic, large integer	ADCLB, ADCLT, SBCLB, SBCLT	2	4	V	-
Arithmetic, pairwise add	ADDP	2	4	V	
Arithmetic, pairwise add and accum long	SADALP, UADALP	4(1)	2	V13	2
Arithmetic, shift	ASR, ASRR, LSL, LSLR, LSR, LSRR	2	2	V13	-
Arithmetic, shift and accumulate	SRSRA, SSRA, URSRA, USRA	4(1)	2	V13	2
Arithmetic, shift by immediate	SHRNB, SHRNT, SSHLLB, SSHLLT, USHLLB, USHLLT	2	2	V13	-
Arithmetic, shift by immediate and insert	SLI, SRI	2	2	V13	
Arithmetic, shift complex	RSHRNB, RSHRNT, SQRSHL, SQRSHLR, SQRSHRNB, SQRSHRNT, SQRSHRUNB, SQRSHRUNT, SQSHL, SQSHLR, SQSHLU, SQSHRNB, SQSHRNT, SQSHRUNB, SQSHRUNT, UQRSHL, UQRSHLR, UQRSHRNB, UQRSHRNT, UQSHL, UQSHLR, UQSHRNB, UQSHRNT	4	2	V13	-

Instruction Group	SVE Instruction	Exec Latency	Execution Throughput	Utilized Pipelines	Notes
Arithmetic, shift right for divide	ASRD	4	2	V13	-
Arithmetic, shift rounding	SRSHL, SRSHLR, SRSHR, URSHL, URSHLR, URSHR	4	2	V13	-
Bit manipulation	BDEP, BEXT, BGRP	6	1/2	V1	-
Bitwise select	BSL, BSL1N, BSL2N, NBSL	2	4	V	-
Count/reverse bits	CLS, CLZ, CNT, RBIT	2	4	V	-
Broadcast logical bitmask immediate to vector	DUPM, MOV	2	4	V	-
Compare and set flags	CMPEQ, CMPGE, CMPGT, CMPHI, CMPHS, CMPLE, CMPLO, CMPLS, CMPLT, CMPNE	4	1	V0, M0	1
Complex add	CADD, SQCADD	2	4	V	-
Complex dot product 8-bit element	CDOT	3(1)	4	V	2
Complex dot product 16-bit element	CDOT	4(1)	2	V02	2
Complex multiply-add B, H, S element size	CMLA	4(1)	2(1)	V02	2
Complex multiply-add D element size	CMLA	5(3)	1	V02	2
Conditional extract operations, scalar form	CLASTA, CLASTB	8	1	M0, V01	-
Conditional extract operations, SIMD&FP scalar and vector forms	CLASTA, CLASTB, COMPACT, SPLICE	3	1	V1	-
Convert to floating point, 64b to float or convert to double	SCVTF, UCVTF	3	2	V02	-
Convert to floating point, 32b to single or half	SCVTF, UCVTF	4	1	V02	-
Convert to floating point, 16b to half	SCVTF, UCVTF	6	1/2	V02	-
Copy, scalar	CPY	5	1	M0, V	
Copy, scalar SIMD&FP or imm	CPY	2	4	V	
Divides, 32 bit	SDIV, SDIVR, UDIV, UDIVR	7 to 12	1/11 to 1/7	V0	3
Divides, 64 bit	SDIV, SDIVR, UDIV, UDIVR	7 to 20	1/20 to 1/7	V0	3
Dot product, 8 bit	SDOT, UDOT	3(1)	4	V	2

Instruction Group	SVE Instruction	Exec Latency	Execution Throughput	Utilized Pipelines	Notes
Dot product, 8 bit, using signed and unsigned integers	SUDOT, USDOT	3(1)	4	V	2
Dot product, 16 bit	SDOT, UDOT	4(1)	2	V02	2
Duplicate, immediate and indexed form	DUP, MOV	2	4	V	-
Duplicate, scalar form	DUP, MOV	3	1	M0	-
Extend, sign or zero	SXTB, SXTH, SXTW, UXTB, UXTH, UXTW	2	2	V13	-
Extract	EXT	2	4	V	-
Extract narrow saturating	SQXTNB, SQXTNT, SQXTUNB, SQXTUNT, UQXTNB, UQXTNT	4	2	V13	-
Extract/insert operation, SIMD and FP scalar form	LASTA, LASTB, INSR	3	1	V1	-
Extract/insert operation, scalar	LASTA, LASTB, INSR	6	1	V1, M0	-
Histogram operations	HISTCNT, HISTSEG	2	4	V	-
Horizontal operations, B, H, S form, immediate operands only	INDEX	4	2	V02	-
Horizontal operations, B, H, S form, (scalar, immediate operands)/ scalar operands only / immediate, scalar operands	INDEX	7	1	M0, V02	-
Horizontal operations, D form, immediate operands only	INDEX	5	1	V02	-
Horizontal operations, D form, scalar, immediate operands)/ scalar operands only / immediate, scalar operands	INDEX	8	1/2	M0, V02	-
Logical	AND, BIC, EON, EOR, EORBT, EORTB, MOV, NOT, ORN, ORR	2	4	V	-
Max/min, basic and pairwise	SMAX, SMAXP, SMIN, SMINP, UMAX, UMAXP UMIN, UMINP	2	4	V	-
Matching operations	MATCH, NMATCH	2	1	V0, M	1,5
Matrix multiply-accumulate	SMMLA, UMMLA, USMMLA	3(1)	4	V	2

Instruction Group	SVE Instruction	Exec Latency	Execution Throughput	Utilized Pipelines	Notes
Move prefix	MOVPRFX	2	4	V	-
Multiply, B, H, S element size	MUL, SMULH, UMULH	4	2	V02	-
Multiply, D element size	MUL, SMULH, UMULH	5	1	V02	-
Multiply long	SMULLB, SMULLT, UMULLB, UMULLT	4	2	V02	-
Multiply accumulate, B, H, S element size	MLA, MLS	4(1)	1	V02	2
Multiply accumulate, D element size	MLA, MLS, MAD, MSB,	5(3)	1	V02	2
Multiply accumulate long	SMLALB, SMLALT, SMLSLB, SMLSLT, UMLALB, UMLALT, UMLSLB, UMLSLT	4(1)	2	V02	2
Multiply accumulate saturating doubling long regular	SQDMLALB, SQDMLALT, SQDMLALBT, SQDMLSLB, SQDMLSLT, SQDMLSLBT	4(2)	2	V02	4
Multiply saturating doubling high, B, H, S element size	SQDMULH	4	2	V02	-
Multiply saturating doubling high, D element size	SQDMULH	5	1	V02	-
Multiply saturating doubling long	SQDMULLB, SQDMULLT	4	2	V02	-
Multiply saturating rounding doubling regular/complex accumulate, B, H, S element size	SQRDMLAH, SQRDMLSH, SQRDCMLAH	4(2)	2	V02	4
Multiply saturating rounding doubling regular/complex accumulate, D element size	SQRDMLAH, SQRDMLSH, SQRDCMLAH	5(3)	1	V02	4
Multiply saturating rounding doubling regular/complex, B, H, S element size	SQRDMULH	4	2	V02	-
Multiply saturating rounding doubling regular/complex, D element size	SQRDMULH	5	1	V02	-
Multiply/multiply long, (8x8) polynomial	PMUL, PMULLB, PMULLT	2	2	V23	-

Instruction Group	SVE Instruction	Exec Latency	Execution Throughput	Utilized Pipelines	Notes
Predicate counting, vector form	DECH, DECW, DECD, INCH, INCW, INCD, SQDECH, SQDECW, SQDECD, SQINCH, SQINCW, SQINCD, UQDECH, UQDECW, UQDECD, UQINCH, UQINCW, UQINCD	2	4	V	-
Reciprocal estimate	URECPE, URSQRTE	4	1	V02	-
Reduction, arithmetic, B form	SADDV, UADDV, SMAXV, SMINV, UMAXV, UMINV	9	½	V, V13	-
Reduction, arithmetic, H form	SADDV, UADDV, SMAXV, SMINV, UMAXV, UMINV	8	1	V, V13	-
Reduction, arithmetic, S form	SADDV, UADDV, SMAXV, SMINV, UMAXV, UMINV	6	1	V, V13	-
Reduction, arithmetic, D form	SMAXV, SMINV, UMAXV, UMINV	4	2	V	-
Reduction, logical	ANDV, EORV, ORV	6	2	V, V13	-
Reverse, vector	REV, REVB, REVH, REVW	2	4	V	-
Select, vector form	MOV, SEL	2	4	V	-
Table lookup	TBL	2	4	V	-
Table lookup extension	TBX	2	4	V	
Transpose, vector form	TRN1, TRN2	2	4	V	-
Unpack and extend	SUNPKHI, SUNPKLO, UUNPKHI, UUNPKLO	2	4	V	-
Zip/unzip	UZP1, UZP2, ZIP1, ZIP2	2	4	V	-

Notes:

1. When the governing predicate is the same as destination, the latency is increased by one cycle.
2. SVE accumulate pipelines support late-forwarding of accumulate operands from similar µOPs, allowing a typical sequence of such µOPs to issue one every N cycles (accumulate latency N shown in parentheses).

3. SVE integer divide operations are performed using an iterative algorithm and block subsequent similar operations to the same pipeline until complete.

4. Same as 2 except that for saturating instructions require an extra cycle of latency for late-forwarding accumulate operands.

5. If the consuming instruction has a flag source, the latency for this instruction is 4 cycles.

3.26 SVE floating-point instructions

Table 3-25 SVE floating-point instructions

Instruction Group	SVE Instruction	Exec Latency	Execution Throughput	Utilized Pipelines	Notes
Floating point absolute value/difference	FABD, FABS	2	4	V	-
Floating point arithmetic	FADD, FADDP, FNEG, FSUB, FSUBR	2	4	V	-
Floating point associative add, F16	FADDA	10	1/9	V1	-
Floating point associative add, F32	FADDA	6	1/5	V1	-
Floating point associative add, F64	FADDA	4	4	V	-
Floating point compare	FACGE, FACGT, FACLE, FACLT, FCMEQ, FCMGE, FCMGT, FCMLE, FCMLT, FCMNE, FCMUO	2	1	V0	-
Floating point complex add	FCADD	3	4	V	-
Floating point complex multiply add	FCMLA	5(2)	4	V	1
Floating point convert, long or narrow (F16 to F32 or F32 to F16)	FCVT, FCVTLT, FCVTNT	4	1	V02	-
Floating point convert, long or narrow (F16 to F64, F32 to F64, F64 to F32 or F64 to F16)	FCVT, FCVTLT, FCVTNT	3	2	V02	-
Floating point convert, round to odd	FCVTX, FCVTXNT	3	2	V02	-
Floating point base2 log, F16	FLOGB	6	1/2	V02	-
Floating point base2 log, F32	FLOGB	4	1	V02	-
Floating point base2 log, F64	FLOGB	3	2	V02	-
Floating point convert to integer, F16	FCVTZS, FCVTZU	6	1/2	V02	-
Floating point convert to integer, F32	FCVTZS, FCVTZU	4	1	V02	-

Instruction Group	SVE Instruction	Exec Latency	Execution Throughput	Utilized Pipelines	Notes
Floating point convert to integer, F64	FCVTZS, FCVTZU	3	2	V02	-
Floating point copy	FCPY, FDUP, FMOV	2	4	V	-
Floating point divide, F16	FDIV, FDIVR	10 to 13	1/6 to 1/5	V02	2
Floating point divide, F32	FDIV, FDIVR	7 to 10	2/9 to 2/7	V02	2
Floating point divide, F64	FDIV, FDIVR	7 to 15	2/14 to 2/7	V02	2
Floating point min/max pairwise	FMAXP, FMAXNMP, FMINP, FMINNMP	2	4	V	-
Floating point min/max	FMAX, FMIN, FMAXNM, FMINNM	2	4	V	-
Floating point multiply	FSCALE, FMUL, FMULX	3	4	V	-
Floating point multiply accumulate	FMLA, FMLS, FMAD, FMSB, FNMAD, FNMLA, FNMLS, FNMSB	4(2)	4	V	1
Floating point multiply add/sub accumulate long	FMLALB, FMLALT, FMLSLB, FMLSLT	4(2)	4	V	
Floating point reciprocal estimate, F16	FRECPE, FRECPX, FRSQRTE	6	1/2	V02	-
Floating point reciprocal estimate, F32	FRECPE, FRECPX, FRSQRTE	4	1	V02	-
Floating point reciprocal estimate, F64	FRECPE, FRECPX, FRSQRTE	3	2	V02	-
Floating point reciprocal step	FRECPS, FRSQRTS	4	4	V	-
Floating point reduction, F16	FADDV, FMAXNMV, FMAXV, FMINNMV, FMINV	8	1	V	-
Floating point reduction, F32	FADDV, FMAXNMV, FMAXV, FMINNMV, FMINV	6	4/3	V	-
Floating point reduction, F64	FADDV, FMAXNMV, FMAXV, FMINNMV, FMINV	4	2	V	-

Instruction Group	SVE Instruction	Exec Latency	Execution Throughput	Utilized Pipelines	Notes
Floating point round to integral, F16	FRINTA, FRINTI, FRINTM, FRINTN, FRINTP, FRINTX, FRINTZ	6	1/2	V02	-
Floating point round to integral, F32	FRINTA, FRINTI FRINTM, FRINTN, FRINTP, FRINTX, FRINTZ	4	1	V02	-
Floating point round to integral, F64	FRINTA, FRINTI, FRINTM, FRINTN, FRINTP, FRINTX, FRINTZ	3	2	V02	-
Floating point square root, F16	FSQRT	10 to 13	1/12 to 1/10	V0	2
Floating point square root, F32	FSQRT	7 to 10	1/9 to 1/7	V0	2
Floating point square root F64	FSQRT	7 to 16	1/14 to 1/7	V0	2
Floating point trigonometric exponentiation	FEXPA	3	1	V1	
Floating point trigonometric multiply add	FTMAD	4	4	V	
Floating point trigonometric, miscellaneous	FTSMUL, FTSSEL	3	4	V	-

Notes:

1. SVE multiply-accumulate pipelines support late-forwarding of accumulate operands from similar µOPs, allowing a typical sequence of floating-point multiply-accumulate µOPs to issue one every N cycles (accumulate latency N shown in parentheses).

2. SVE divide and square root operations are performed using an iterative algorithm and block subsequent similar operations to the same pipeline until complete.

3.27 SVE BFloat16 (BF16) instructions

Table 3-26 SVE Bfloat16 (BF16) instructions

Instruction Group	SVE Instruction	Exec Latency	Execution Throughput	Utilized Pipelines	Notes
Convert, F32 to BF16	BFCVT, BFCVTNT	4	2	V02	-
Dot product	BFDOT	5(3)	4	V	1
Matrix multiply accumulate	BFMMLA	6(4)	4	V	1
Multiply accumulate long	BFMLALB, BFMLALT	5(2)	4	V	1

Notes:

1. SVE pipelines that execute these instructions support late-forwarding of accumulate operands from similar µOPs, allowing a typical sequence of µOPs to issue one every N cycles (accumulate latency N shown in parentheses).

3.28 SVE Load instructions

The latencies shown assume the memory access hits in the Level 1 Data Cache and represent the maximum latency to load all the vector registers written by the instruction. Compared to standard loads, an extra cycle is required to forward results to vector pipelines.

Table 3-27 SVE Load instructions

Instruction Group	SVE Instruction	Exec Latency	Execution Throughput	Utilized Pipelines	Notes
Load vector	LDR	6	3	L	-
Load predicate	LDR	6	2	L, M	-
Contiguous load, scalar + imm	LD1B, LD1D, LD1H, LD1W, LD1SB, LD1SH, LD1SW,	6	3	L	-
Contiguous load, scalar + scalar	LD1B, LD1D, LD1H, LD1W, LD1SB, LD1SH LD1SW	6	3	L	-
Contiguous load broadcast, scalar + imm	LD1RB, LD1RH, LD1RD, LD1RW, LD1RSB, LD1RSH, LD1RSW, LD1RQB, LD1RQD, LD1RQH, LD1RQW	6	3	L	-
Contiguous load broadcast, scalar + scalar	LD1RQB, LD1RQD, LD1RQH, LD1RQW	6	3	L	-
Non temporal load, scalar + imm	LDNT1B, LDNT1D, LDNT1H, LDNT1W	6	3	L	-
Non temporal load, scalar + scalar	LDNT1B, LDNT1D, LDNT1H, LDNT1W	6	3	L	-
Non temporal gather load, vector + scalar 32-bit element size	LDNT1B, LDNT1H, LDNT1W, LDNT1SB, LDNT1SH	9	1	L, V	-

Instruction Group	SVE Instruction	Exec Latency	Execution Throughput	Utilized Pipelines	Notes
Non temporal gather load, vector + scalar 64-bit element size	LDNT1B, LDNT1D, LDNT1H, LDNT1W, LDNT1SB, LDNT1SH, LDNT1SW	9	1/2	L, V1	-
Contiguous first faulting load, scalar + scalar	LDFF1B, LDFF1D, LDFF1H, LDFF1W, LDFF1SB, LDFF1SH, LDFF1SW	6	3	L, S	-
Contiguous non faulting load, scalar + imm	LDNF1B, LDNF1D, LDNF1H, LDNF1W, LDNF1SB, LDNF1SH, LDNF1SW	6	3	L	-
Contiguous Load two structures to two vectors, scalar + imm	LD2B, LD2D, LD2H, LD2W	8	3/2	V, L	-
Contiguous Load two structures to two vectors, scalar + scalar	LD2B, LD2D, LD2H, LD2W	9	3/2	V, L, S	-
Contiguous Load three structures to three vectors, scalar + imm	LD3B, LD3D, LD3H, LD3W	9	1	V, L	-
Contiguous Load three structures to three vectors, scalar + scalar	LD3B, LD3D, LD3H, LD3W	10	1	V, L, S	-
Contiguous Load four structures to four vectors, scalar + imm	LD4B, LD4D, LD4H LD4W	9	1/2	V, L	-
Contiguous Load four structures to four vectors, scalar + scalar	LD4B, LD4D, LD4H, LD4W	10	1/2	L, V, S	-
Gather load, vector + imm, 32-bit element size	LD1B, LD1H, LD1W, LD1SB, LD1SH, LDFF1B, LDFF1H, LDFF1W, LDFF1SB, LDFF1SH	9	1	L, V	-

Instruction Group	SVE Instruction	Exec Latency	Execution Throughput	Utilized Pipelines	Notes
Gather load, vector + imm, 64-bit element size	LD1B, LD1D, LD1H, LD1W, LD1SB, LD1SH, LD1SW, LDFF1B, LDFF1D LDFF1H, LDFF1W, LDFF1SB, LDFF1SH, LDFF1SW	9	1	L, V	-
Gather load, 32-bit scaled offset	LD1H, LD1SH, LDFF1H, LDFF1SH, LD1W, LDFF1W, LDFF1SW	10	1/2	L, V	-
Gather load, 32-bit unpacked unscaled offset	LD1B, LD1SB, LDFF1B, LDFF1SB, LD1D, LDFF1D, LD1H, LD1SH, LDFF1H, LDFF1SH, LD1W, LD1SW, LDFF1W, LDFF1SW	9	1	L, V	-

3.29 SVE Store instructions

Stores MOPs are split into store address and store data μOPs. Once executed, stores are buffered and committed in the background. The latency represents the maximum latency to store all the vector registers written to memory by the instruction.

Table 3-28 SVE Store instructions

Instruction Group	SVE Instruction	Exec Latency	Execution Throughput	Utilized Pipelines	Notes
Store from predicate reg	STR	1	2	L01	-
Store from vector reg	STR	2	2	L01, V01	-
Contiguous store, scalar + imm	ST1B, ST1H, ST1D, ST1W	2	2	L01, V01	-
Contiguous store, scalar + scalar	ST1H	2	2	L01, S, V01	-
Contiguous store, scalar + scalar	ST1B, ST1D, ST1W	2	2	L01, V01	-
Contiguous store two structures from two vectors, scalar + imm	ST2B, ST2H, ST2D, ST2W	4	1	L01, V01	-
Contiguous store two structures from two vectors, scalar + scalar	ST2H	4	1	L01, S, V01	-

Instruction Group	SVE Instruction	Exec Latency	Execution Throughput	Utilized Pipelines	Notes
Contiguous store two structures from two vectors, scalar + scalar	ST2B, ST2D, ST2W	4	1	L01, V01	-
Contiguous store three structures from three vectors, scalar + imm	ST3B, ST3D, ST3H, ST3W	7	2/9	L01, V01	-
Contiguous store three structures from three vectors, scalar + scalar	ST3H	7	2/9	L01, S, V01	-
Contiguous store three structures from three vectors, scalar + scalar	ST3B, ST3D, ST3W	7	2/9	L01, S, V01	-
Contiguous store four structures from four vectors, scalar + imm	ST4B, ST4D, ST4H, ST4W	11	1/9	L01, V01	-
Contiguous store four structures from four vectors, scalar + scalar	ST4H	11	1/9	L01, S, V01	-
Contiguous store four structures from four vectors, scalar + scalar	ST4B, ST4D, ST4W	11	1/9	L01, S, V01	-
Non temporal store, scalar + imm	STNT1B, STNT1D, STNT1H, STNT1W	2	2	L01, V01	-
Non temporal store, scalar + scalar	STNT1H	2	2	L01, S, V01	-
Non temporal store, scalar + scalar	STNT1B, STNT1D, STNT1W	2	2	L01, V01	-
Scatter non temporal store, vector + scalar 32-bit element size	STNT1B, STNT1H, STNT1W	4	1/2	L01, V01	-
Scatter non temporal store, vector + scalar 64-bit element size	STNT1B, STNT1D, STNT1H, STNT1W	2	1	L01, V01	-
Scatter store vector + imm 32-bit element size	ST1B, ST1H, ST1W	4	1/2	L01, V01	-
Scatter store vector + imm 64-bit element size	ST1B, ST1D, ST1H, ST1W	2	1	L01, V01	-
Scatter store, 32-bit scaled offset	ST1H, ST1W	4	1/2	L01, V01	-
Scatter store, 32-bit unpacked unscaled offset	ST1B, ST1D, ST1H, ST1W	2	1	L01, V01	-
Scatter store, 32-bit unpacked scaled offset	ST1D, ST1H, ST1W	2	1	L01, V01	-

Instruction Group	SVE Instruction	Exec Latency	Execution Throughput	Utilized Pipelines	Notes
Scatter store, 32-bit unscaled offset	ST1B, ST1H, ST1W	4	1/2	L01, V01	-
Scatter store, 64-bit scaled offset	ST1D, ST1H, ST1W	2	1	L01, V01	-
Scatter store, 64-bit unscaled offset	ST1B, ST1D, ST1H, ST1W	2	1	L01, V01	-

3.30 SVE Miscellaneous instructions

Table 3-29 SVE miscellaneous instructions

Instruction Group	SVE Instruction	Exec Latency	Execution Throughput	Utilized Pipelines	Notes
Read first fault register, unpredicated	RDFFR	2	1	M0	-
Read first fault register, predicated	RDFFR	3	1	M0, M	1
Read first fault register and set flags	RDFFRS	4	1/2	M0, M	1
Set first fault register	SETFFR	2	1	M0	-
Write to first fault register	WRFFR	2	1	M0	-

Notes:

1. When destination is same as the governing predicate, the latency of the instruction increases by one cycle.

3.31 SVE Cryptographic instructions

Table 3-30 SVE cryptographic instructions

Instruction Group	AArch64 Instructions	Exec Latency	Execution Throughput	Utilized Pipelines	Notes
Crypto AES ops	AESD, AESE, AESIMC, AESMC	2	4	V	-
Crypto SHA3 ops	BCAX, EOR3, RAX1, XAR	2	1	V0	-
Crypto SM4 ops	SM4E, SM4EKEY	4	1	V0	-

4 Special considerations

4.1 Dispatch constraints

Dispatch of µOPs from the in-order portion to the out-of-order portion of the microarchitecture includes several constraints. It is important to consider these constraints during code generation to maximize the effective dispatch bandwidth and subsequent execution bandwidth of the Neoverse V2 core.

The dispatch stage can process up to 8 MOPs per cycle and dispatch up to 16 µOPs per cycle, with the following limitations on the number of µOPs of each type that may be simultaneously dispatched.

> Up to 4 µOPs utilizing the S or B pipelines
>
> Up to 4 µOPs utilizing the M pipelines
>
> Up to 2 µOPs utilizing the M0 pipelines
>
> Up to 2 µOPs utilizing the V0 pipeline
>
> Up to 2 µOPs utilizing the V1 pipeline
>
> Up to 6 µOPs utilizing the L pipelines

In the event there are more µOPs available to be dispatched in a given cycle than can be supported by the constraints above, µOPs will be dispatched in oldest to youngest age-order to the extent allowed by the above.

4.2 Optimizing general-purpose register spills and fills

Register transfers between general-purpose registers (GPR) and ASIMD registers (VPR) are lower latency than reads and writes to the cache hierarchy, thus it is recommended that GPR registers be filled/spilled to the VPR rather to memory, when possible.

4.3 Optimizing memory routines

To achieve maximum throughput for memory copy (or similar loops), one should do the following.

Unroll the loop to include multiple load and store operations per iteration, minimizing the overheads of looping.

Align loads on 16B boundary wherever possible.

Use non-writeback forms of LDP and STP/STR instructions interleaving them like shown in the examples below:

For forward copies:

```
Loop_start:
    SUBS    x2,x2,#96
    LDP     q3,q4,[x1,#0]
    STP     q3,q4,[x0,#0]
    LDP     q3,q4,[x1,#32]
    STP     q3,q4,[x0,#32]
    LDP     q3,q4,[x1,#64]
    STP     q3,q4,[x0,#64]
    ADD     x1,x1,#96
    ADD     x0,x0,#96
    BGT     Loop_start
```

For backward copies:

```
Loop_start:
    SUBS    x2,x2,#96
    LDP     q4,q3,[x1,#-32]
    STR     q3,[x0,#-16]
    STR     q4,[x0,#-32]
    LDP     q4,q3,[x1,#-64]
    STR     q3,[x0,#-48]
    STR     q4,[x0,#-64]
    LDP     q4,q3,[x1,#-96]
    STR     q3,[x0,#-80]
    STR     q4,[x0,#-96]
    SUB     x1,x1,#96
    SUB     x0,x0,#96
    BGT     Loop_start
```

If the memory locations being copied are non-cacheable, the non-temporal version of LDPQ (LDNPQ) should be used. STPQ/STRQ should still be used for the stores.

Similarly, it Is recommended to use LDPQ to achieve maximum throughput for memcmp (memory compare) loops that compare cacheable memory. LDNPQ should be used for non-cacheable memory.

To achieve maximum throughput on memset, it is recommended that one do the following.

Unroll the loop to include multiple store operations per iteration, minimizing the overheads of looping.

```
Loop_start:
    STP     q1,q3,[x0,#0]
    STP     q1,q3,[x0,#0x20]
    STP     q1,q3,[x0,#0x40]
    STP     q1,q3,[x0,#0x60]
    ADD     x0,x0,#0x80
    SUBS    x2,x2,#0x80
```

```
    B.GT      Loop_start
```

To achieve maximum performance on memset to zero, it is recommended that one use DC ZVA instead of STP. An optimal routine might look something like the following.

```
Loop_start:
    SUBS      x2,x2,#0x80
    DC        ZVA,x0
    ADD       x0,x0,#0x40
    DC        ZVA,x0
    ADD       x0,x0,#0x40
    B.GT      Loop_start
```

4.4 Load/Store alignment

The Armv8-A architecture allows many types of load and store accesses to be arbitrarily aligned. The Neoverse V2 core handles most unaligned accesses without performance penalties. However, there are cases which could reduce bandwidth or incur additional latency, as described below.

- Load operations that cross a cache-line (64-byte) boundary.

- Quad-word load operations that are not 4B aligned.

- Store operations that cross a 32B boundary.

4.5 Store to Load Forwarding

The Neoverse V2 core allows data to be forwarded from store instructions to a load instruction with the restrictions mentioned below:

- Load start address should align with the start or middle address of the older store

- Loads of size greater than or equal to 8 bytes can get the data forwarded from a maximum of 2 stores. If there are 2 stores, then each store should forward to either first or second half of the load

- Loads of size less than or equal to 4 bytes can get their data forwarded from only 1 store

4.6 AES encryption/decryption

Neoverse V2 core can issue eight AESE/AESMC/AESD/AESIMC instructions every cycle (fully pipelined) with an execution latency of two cycles. This means encryption or decryption for at least eight data chunks should be interleaved for maximum performance:

```
AESE  data0, key_reg
AESMC data0, data0
AESE  data1, key_reg
AESMC data1, data1
AESE  data2, key_reg
AESMC data2, data2
AESE  data3, key_reg
AESMC data3, data3
AESE  data4, key_reg
AESMC data4, data4
AESE  data5, key_reg
AESMC data5, data5
AESE  data6, key_reg
AESMC data6, data6
AESE  data7, key_reg
AESMC data7, data7
AESE  data0, key_reg
AESMC data0, data0
...
```

Pairs of dependent AESE/AESMC and AESD/AESIMC instructions are higher performance when they are adjacent in the program code and both instructions use the same destination register.

4.7 Region based fast forwarding

The forwarding logic in the V pipelines is optimized to provide optimal latency for instructions which are expected to commonly forward to one another. The effective latency of FP and ASIMD instructions as described in section 3 is increased by one cycle if the producer and consumer instructions are not part of the same forwarding region. These optimized forwarding regions are defined in the following table.

Table 4-1 Optimized forwarding regions

Region	Instruction Types	Notes
1	ASIMD/SVE ALU, ASIMD/SVE shift, ASIMD/scalar insert and move, ASIMD/SVE abs/cmp/max/min and the ASIMD miscellaneous instructions in table 3-18.	1
2	FP/ASIMD/SVE multiply, FP/ASIMD/SVE multiply-accumulate, FP compare, FP add/sub and the ASIMD miscellaneous instructions in table 3-18.	1,2,3
3	ASIMD/SVE Crypto and SHA1/SHA256	-
4	ASIMD/SVE AES, ASIMD/SVE polynomial multiply and all the instruction types in region 1.	1
5	ASIMD/SVE BFDOT and BFMMLA instructions	-

Notes:

1. Reciprocal step and estimate instructions are excluded from this region.
2. ASIMD extract narrow, saturating instructions are excluded from this region.
3. ASIMD miscellaneous instructions can only be consumers of this region.

The following instructions are not a part of any region:

- FP div/sqrt

- FP convert and rounding instructions that do not write to general purpose registers

- ASIMD integer mul/mac

- ASIMD reduction

In addition to the regions mentioned in the table above, all instructions in regions 1 and 2 can fast forward to FP/ASIMD/SVE stores, FP/ASIMD vector to integer register transfers and ASIMD converts that write to general purpose registers.

More special notes about the forwarding region in table 4-1:

- Element sources used by FP multiply and multiply-accumulate operations cannot be consumers.

- Complex ASIMD shift by immediate/register and shift accumulate instructions cannot be producers (see section 3.16) in region 1.

- ASIMD extract narrow, saturating instructions cannot be producers (see section 3.19) in region 1.

- ASIMD absolute difference accumulate and pairwise add and accumulate instructions cannot be producers (see section 3.16) in region 1.

- For FP producer-consumer pairs, the precision of the instructions should match (single, double or half) in region 2.

- Pair-wise FP instructions cannot be producers or consumers in region 2.

It is not advisable to interleave instructions belonging to different regions. Also, certain instructions can only be producers or consumers in a particular region but not both (see footnote 3 for table 4-1). For example, the code below interleaves producers and consumers from regions 1 and 2. This will result in and additional latency of 1 cycle as seen by FMUL.

    FSUB   v27.2s, v28.2s, v20.2s – Region 2
    FADD   v20.2s, v28.2s, v20.2s – Region 2
    MOV   v27.s[1], v20.s[1] - Region 2 producer but not a region 2 consumer
    FMUL   v26.2s, v27.2s, v6.2s – Region 2

4.8 Branch instruction alignment

Branch instructions and their density can affect performance.

> **Note**
>
> For best case performance, avoid placing more than four branch instructions within an aligned 32-byte instruction memory region.

4.9 FPCR self-synchronization

Programmers and compiler writers should note that writes to the FPCR register are self-synchronizing, i.e. its effect on subsequent instructions can be relied upon without an intervening context synchronizing operation.

4.10 Special register access

The Neoverse V2 core performs register renaming for general purpose registers to enable speculative and out-of-order instruction execution. But most special-purpose registers are not renamed. Instructions that read or write non-renamed registers are subjected to one or more of the following additional execution constraints.

Non-Speculative Execution – Instructions may only execute non-speculatively.

In-Order Execution – Instructions must execute in-order with respect to other similar instructions or in some cases all instructions.

Flush Side-Effects – Instructions trigger a flush side-effect after executing for synchronization.

The table below summarizes various special-purpose register read accesses and the associated execution constraints or side-effects.

Table 4-2 Special-purpose register read accesses

Register Read	Non-Speculative	In-Order	Flush Side-Effect	Notes
APSR	Yes	Yes	No	3
CurrentEL	No	Yes	No	-
DAIF	No	Yes	No	-
DLR_EL0	No	Yes	No	-
DSPSR_EL0	No	Yes	No	-
ELR_*	No	Yes	No	-
FPCR	No	Yes	No	-
FPSCR	Yes	Yes	No	2
FPSR	Yes	Yes	No	2
NZCV	No	No	No	1
SP_*	No	No	No	1
SPSel	No	Yes	No	-
SPSR_*	No	Yes	No	-
FFR	No	Yes	No	-

Notes:

1. The NZCV and SP registers are fully renamed.
2. FPSR/FPSCR reads must wait for all prior instructions that may update the status flags to execute and retire.
3. APSR reads must wait for all prior instructions that may set the Q bit to execute and retire.

The table below summarizes various special-purpose register write accesses and the associated execution constraints or side-effects.

Table 4-3 Special-purpose register write accesses

Register Write	Non-Speculative	In-Order	Flush Side-Effect	Notes
APSR	Yes	Yes	No	4
DAIF	Yes	Yes	No	-
DLR_EL0	Yes	Yes	No	-
DSPSR_EL0	Yes	Yes	No	-
ELR_*	Yes	Yes	No	-
FPCR	Yes	Yes	Maybe	2
FPSCR	Yes	Yes	Maybe	2, 3
FPSR	Yes	Yes	No	3
NZCV	No	No	No	1
SP_*	No	No	No	1
SPSel	Yes	Yes	Yes	-
SPSR_*	Yes	Yes	No	-
FFR	Yes	Yes	No	-

Notes:

1. The NZCV and SP registers are fully renamed.

2. If the FPCR/FPSCR write is predicted to change the control field values, it will introduce a barrier which prevents subsequent instructions from executing. If the FPCR/FPSCR write is predicted to not change the control field values, it will execute without a barrier but trigger a flush if the values change.

3. FPSR/FPSCR writes must stall at dispatch if another FPSR/FPSCR write is still pending.

4. APSR writes that set the Q bit will introduce a barrier which prevents subsequent instructions from executing until the write completes.

4.11 Instruction fusion

Neoverse V2 core can accelerate certain instruction pairs in an operation called fusion. Specific Aarch64 instruction pairs that can be fused are as follows:

>AESE + AESMC (see Section 4.5 on AES Encryption/Decryption)

>AESD + AESIMC (see Section 4.5 on AES Encryption/Decryption)

>CMP/CMN (immediate) + B.cond

>CMP/CMN (register) + B.cond

>CMP + CSEL

>CMP + CSET

>TST (immediate) + B.cond

>TST (register) + B.cond

>BICS (register) + B.cond

>NOP + Any instruction

>MOVPRFX+SVE instruction fusion (see Section 4.16 on MOVPRFX fusion)

These instruction pairs must be adjacent to each other in program code. For CMP, CMN, TST and BICS, fusion is not allowed for shifted and/or extended register forms. For BICS, the destination register should be XZR or WZR if fusion is to take place.

4.12 Zero Latency MOVs

A subset of register-to-register move operations and move immediate operations are executed with zero latency. These instructions do not utilize the scheduling and execution resources of the machine. These are as follows:

MOV Xd, #0

MOV Xd, XZR

MOV Wd, #0

MOV Wd, WZR

MOV Hd, WZR

MOV Hd, XZR

MOV Sd, WZR

MOV Dd, XZR

MOVI Dd, #0

MOVI Vd.2D, #0

MOV Wd, Wn

MOV Xd, Xn

The last 2 instructions may not be executed with zero latency under certain conditions.

4.13 Cache maintenance operations

While using set way invalidation operations on L1 cache, it is recommended that software be written to traverse the sets in the inner loop and ways in the out loop.

4.14 Memory Tagging - Tagging Performance

To achieve maximum throughput for tag-only, it is recommended that one do the following.

Unroll the loop to include multiple store operations per iteration, minimizing the overheads of looping. Use STGM (or DCGVA) instruction as shown in the example below:

```
Loop_start:
SUBS   x2,x2,#0x80
STGM   x1,[x0]
ADD    x0,x0,#0x40
STGM   x1,[x0]
ADD    x0,x0,#0x40
B.GT   Loop_start
```

To achieve maximum throughput for tag and zeroing out data, it is recommended that one do the following.

Unroll the loop to include multiple store operations per iteration, minimizing the overheads of looping. Use STZGM (or DCZGVA) instruction as shown in the example below:

```
Loop_start:
SUBS   x2,x2,#0x80
STZGM  x1,[x0]
ADD    x0,x0,#0x40
STZGM  x1,[x0]
ADD    x0,x0,#0x40
B.GT   Loop_start
```

To achieve maximum throughput for tag-loading, it is recommended that one do the following.

Unroll the loop to include multiple load operations per iteration, minimizing the overheads of looping. Use LDGM instruction as shown in the example below:

```
Loop_start:
SUBS   x2,x2,#0x80
LDGM   x1,[x0]
ADD    x0,x0,#0x40
LDGM   x1,[x0]
ADD    x0,x0,#0x40
B.GT   Loop_start
```

Also, it is recommended to use STZGM (or DCZGVA) to set tag if data is not a concern.

4.15 Memory Tagging - Synchronous Mode

In synchronous tag checking mode, stores cannot be performed speculatively. Each store must complete a tag check before the next store can be executed non-speculatively. Thus, performance of stores in synchronous tag checking mode will be diminished.

It is recommended to use asynchronous or asymmetric mode for better performance.

4.16 Complex ASIMD and SVE instructions

The bandwidth of the following ASIMD and SVE instructions is limited by decode constraints and it is advisable to avoid them when high performing code is desired.

ASIMD

LD4R, post-indexed addressing, element size = 64b.

LD4, single 4-element structure, post indexed addressing mode, element size = 64b.

LD4, multiple 4-element structures, quad form.

LD4, multiple 4-element structures, double word form.

ST4, multiple 4-element structures, quad form, element size less than 64b.

ST4, multiple 4-element structures, quad form, element size = 64b, post indexed addressing mode.

SVE

LD1B gather (scalar + vector addressing) where vector index register is the same as the destination register and element size = 32. Addressing mode is 32b unscaled offset.

LD1H gather (scalar + vector addressing) where vector index register is the same as the destination register and element size = 32. Addressing mode is 32b scaled or unscaled offset.

LD1W gather (scalar + vector addressing) where vector index register is the same as the destination register and element size = 32. Addressing mode is 32b scaled or unscaled offset.

LD3[B/H/W/D] contiguous (scalar + scalar addressing).

LD4[B/H/D/W] contiguous (scalar + immediate addressing).

LD4[B/H/D/W] contiguous (scalar + scalar addressing).

LDFF1B gather (scalar + vector addressing) where vector index register is the same as the destination register and element size = 32. Addressing mode is 32b unscaled offset.

LDFF1H gather (scalar + vector addressing) where vector index register is the same as the destination register and element size = 32. Addressing mode is 32b scaled or unscaled offset.

LDFF1W gather (scalar + vector addressing) where vector index register is the same as the destination register and element size = 32. Addressing mode is 32b scaled or unscaled offset.

ST3[B/H/W/D] contiguous (scalar + scalar addressing).

ST4[B/H/D/W] contiguous (scalar + immediate addressing).

ST4[B/H/D/W] contiguous (scalar + scalar addressing).

4.17 MOVPRFX fusion

Under certain conditions, a mechanism called MOVPRFX fusion can be used to accelerate the execution of an instruction pair that consists of an SVE MOVPRFX instruction immediately followed in program order by an SVE integer, floating point or BF16 instruction. The list of SVE instructions and the conditions under which tis fusion can be applied is mentioned in the tables below.

Instruction Group	SVE Instruction	Notes
Integer Instructions		
Arithmetic, absolute difference accumulate	SABA, SABALB, SABALT, UABA, UABALB, UABALT	-
Arithmetic, basic	ABS, ADD, CNOT, NEG, SHADD, SHSUB, SHSUBR, SUB, SUBR, UHADD, UHSUB, UHSUBR	For ADD and SUB, only the immediate and vector, predicated forms are fusible.
Arithmetic, complex	SQABS, SQADD, SQNEG, SQSUB, SQSUBR, SRHADD, SUQADD, UQADD, UQSUB, UQSUBR, URHADD, USQADD	For SQABS, SQSUB, UQADD and UQSUB, only the immediate and vector, predicated forms are fusible.
Arithmetic, large integer	ADCLB, ADCLT, SBCLB, SBCLT	-
Arithmetic, pairwise add	ADDP	-
Arithmetic, pairwise add and accum long	SADALP, UADALP	-
Arithmetic, shift	ASR, ASRR, LSL, LSLR, LSR, LSRR	For ASR, LSL and LSR, only the immediate, predicated and vector forms are fusible.
Arithmetic, shift and accumulate	SRSRA, SSRA, URSRA, USRA	-
Arithmetic, shift complex	SQRSHL, SQRSHLR, SQSHL, SQSHLR, SQSHLU, UQRSHL, UQRSHLR, UQSHL, UQSHLR	-
Arithmetic, shift right for divide	ASRD	-
Arithmetic, shift rounding	SRSHL, SRSHLR, SRSHR, URSHL, URSHLR, URSHR	-
Bitwise select	BSL, BSL1N, BSL2N, NBSL	-
Count/reverse bits	CLS, CLZ, CNT, RBIT	-
Complex add	CADD, SQCADD	-
Complex dot product	CDOT	-
Complex multiply-add	CMLA	-
Conditional extract operations	CLASTA, CLASTB, SPLICE	For CLASTA and CLASTB, only the vector forms are fusible.
Convert to floating point	SCVTF, UCVTF	-

Instruction Group	SVE Instruction	Notes
Copy	CPY	All forms except the immediate, zeroing form are fusible.
Divides	SDIV, SDIVR, UDIV, UDIVR	-
Dot product	SDOT, UDOT, SUDOT, USDOT	-
Extend, sign or zero	SXTB, SXTH, SXTW, UXTB, UXTH, UXTW	-
Extract/insert operation	EXT, INSR	-
Logical	AND, BIC, EON, EOR, EORBT, EORTB, NOT, ORN, ORR	For AND, BIC, EOR and ORR, only the immediate and vector, predicated forms are fusible
Max/min, basic and pairwise	SMAX, SMAXP, SMIN, SMINP, UMAX, UMAXP UMIN, UMINP	-
Matrix multiply-accumulate	SMMLA, UMMLA, USMMLA	-
Multiply	MUL, SMULH, UMULH	For MUL, only the immediate and vector, predicated forms are fusible. For the others, only the predicated form is fusible.
Multiply accumulate	MLA, MLS	For the vector forms, only unpredicated and zeroing predicate forms of MOVPRFX are fusible.
Multiply accumulate long	SMLALB, SMLALT, SMLSLB, SMLSLT, UMLALB, UMLALT, UMLSLB, UMLSLT	-
Multiply accumulate saturating doubling long regular	SQDMLALB, SQDMLALT, SQDMLALBT, SQDMLSLB, SQDMLSLT, SQDMLSLBT	-
Multiply saturating rounding doubling regular/complex accumulate	SQRDMLAH, SQRDMLSH, SQRDCMLAH	-
Predicate counting, vector form	DECH, DECW, DECD, INCH, INCW, INCD, SQDECH, SQDECW, SQDECD, SQINCH, SQINCW, SQINCD, UQDECH, UQDECW, UQDECD, UQINCH, UQINCW, UQINCD	-
Reciprocal estimate	URECPE, URSQRTE	-
Reverse, vector	REV, REVB, REVH, REVW	-
Select, vector form	SEL	-
Floating point Instructions		
Floating point absolute value/difference	FABD, FABS	-
Floating point arithmetic	FADD, FADDP, FNEG, FSUB, FSUBR	For FADD and FSUB, only the immediate and vector, predicated forms are fusible.
Floating point complex add	FCADD	-
Floating point complex multiply add	FCMLA	For the vector form, only unpredicated and zeroing predicate forms of MOVPRFX are fusible.

Instruction Group	SVE Instruction	Notes
Floating point convert	FCVT, FCVTX	-
Floating point base2 log	FLOGB	-
Floating point convert to integer	FCVTZS, FCVTZU	-
Floating point copy	FCPY, FMOV	Only the predicated forms of FCPY are fusible
Floating point divide	FDIV, FDIVR	-
Floating point min/max pairwise	FMAXP, FMAXNMP, FMINP, FMINNMP	-
Floating point min/max	FMAX, FMIN, FMAXNM, FMINNM	-
Floating point multiply	FSCALE, FMUL, FMULX	For FMUL, only the immediate and vector, predicated forms are fusible
Floating point multiply accumulate	FMLA, FMLS, FMAD, FMSB, FNMAD, FNMLA, FNMLS, FNMSB	For FMLA and FMLS, only unpredicated and zeroing predicate forms of MOVPRFX are fusible.
Floating point multiply add/sub accumulate long	FMLALB, FMLALT, FMLSLB, FMLSLT	-
Floating point reciprocal estimate	FRECPX	-
Floating point round to integral	FRINTA, FRINTI, FRINTM, FRINTN, FRINTP, FRINTX, FRINTZ	-
Floating point square root	FSQRT	-
Floating point trigonometric multiply add	FTMAD	-
BF16 Instructions		
Dot product	BFDOT	-
Matrix multiply accumulate	BFMMLA	-
Multiply accumulate long	BFMLALB, BFMLALT	-
Cryptographic Instructions		
Crypto SHA3 ops	BCAX, EOR3, XAR	-