Arm® C1-Pro Core

Revision r1p2

Software Optimization Guide

Non-Confidential

Issue 05
109879_0102_05_en

Arm® C1-Pro Core Software Optimization Guide

This document (109879_0102_05_en) was issued on 2025-09-10. There might be a later issue at http://developer.arm.com/documentation/109879

The product revision is r1p2.

See also: Proprietary notice | Product and document information | Useful resources

Start reading

If you prefer, you can skip to the start of the content.

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. This document provides additional performance information about the C1-Pro core unit. For a more complete description of the C1-Pro core unit, please refer to the Arm® C1-Pro Core Technical Reference Manual.

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.

Feedback

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

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

Contents

1. Product Overview

The C1-Pro core is a balanced-performance, low-power, and constrained area product that implements the Arm®v9.3-A architecture. The Arm®v9.3-A architecture extends the architecture defined in the Arm®v8-A architectures up to Arm®v8.8-A.

Core features

- Implementation of the Arm®v9.3-A A64 instruction set
- AArch64 Execution state at all Exception levels, EL0 to EL3
- *Memory Management Unit* (MMU)
- 40-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 *Single Instruction Multiple Data* (SIMD) and floating-point support
- Optional implementation of the *Scalable Matrix Extension* (SME) and *Scalable Matrix Extension 2* (SME2) and support for the C1-SME2 unit.

Note

The C1-SME2 unit is optional, unless the cluster includes a high-performance core. If the C1-SME2 unit is not implemented, SME ans SME2 are not supported. For more information about configuring the C1-SME2, see the Arm® C1-Scalable Matrix Extension 2 Configuration and Integration Manual and the RTL configuration process section in the Arm® C1-DynamIQ Shared Unit Configuration and Integration Manual.

- *Activity Monitoring Unit* (AMU)
- Support for the optional Cryptographic Extension

Note

The Cryptographic Extension is licensed separately.

Cache features

- Separate L1 data and instruction caches
- Private, unified data and instruction L2 cache

- Optional error protection with parity or *Error Correcting Code* (ECC) allowing:
    - *Single Error Correction and Double Error Detection* (SECDED) on L1 data cache and L2 cache
    - *Single Error Detection* (SED) on L1 instruction cache and L2 *Translation Lookaside Buffer* (TLB)
- Support for *Memory System Resource Partitioning and Monitoring* (MPAM)

Debug features

- Arm®v8.8 debug architecture
- *Performance Monitoring Unit* (PMU)
- *Embedded Trace Extension* (ETE)
- *TRace Buffer Extension* (TRBE)
- *Statistical Profiling Extension* (SPE)
- Optional *Embedded Logic Analyzer* (ELA), ELA-600

Note

The ELA-600 is licensed separately.

1.1 Pipeline overview

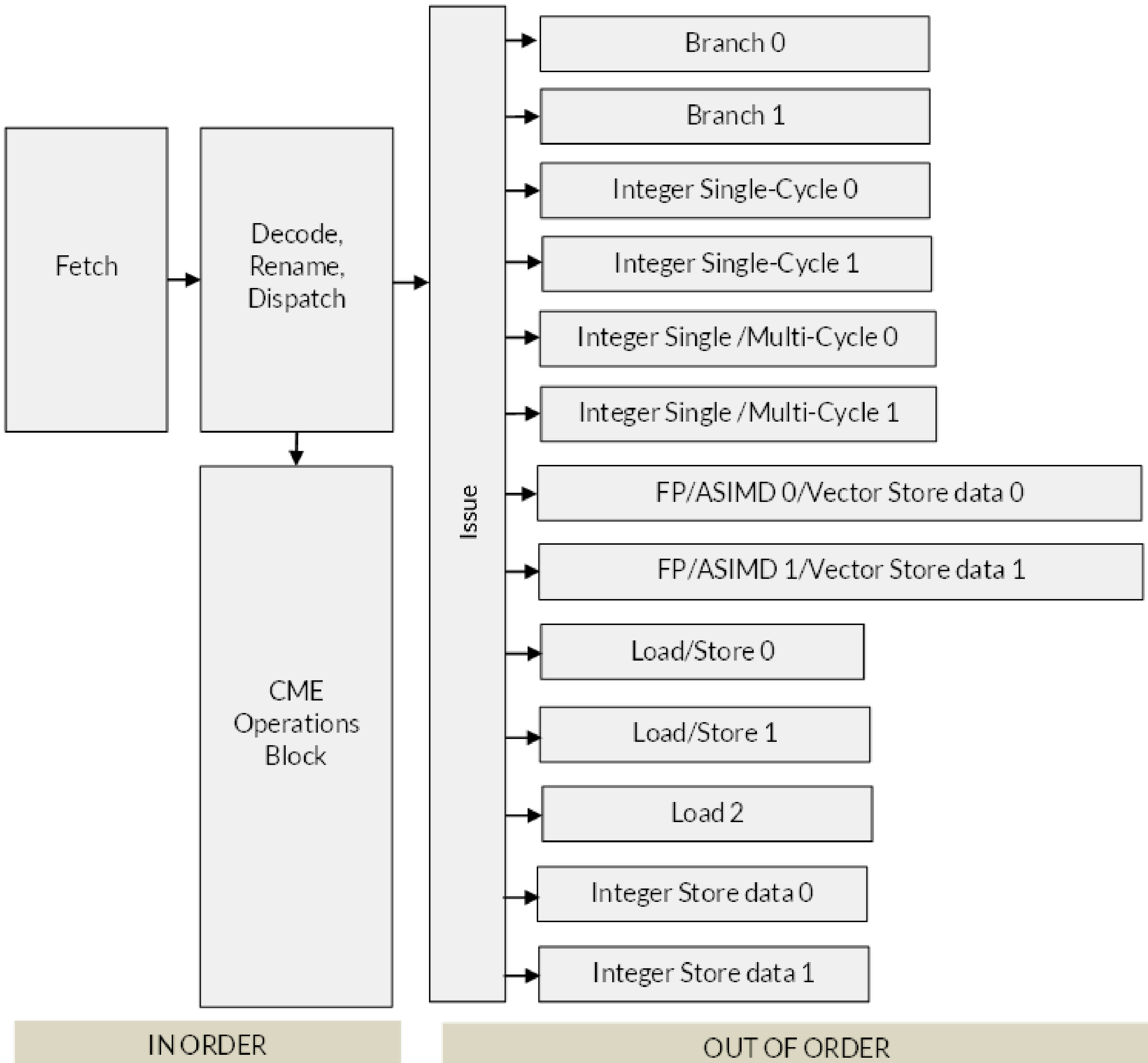
The following figure describes the high-level C1-Pro core instruction processing pipeline.

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 thirteen issue pipelines.

Each issue pipeline can accept one µOP per cycle.

Figure 1-1: C1-Pro core pipeline



The execution pipelines support different types of operations, as shown in the following table.

Table 1-1: C1-Pro core operations

Instruction groups	Instructions
Branch 0/1	Branch µOps
Integer Single-Cycle 0/1	Integer ALU µOPs
Integer Single/ Multi-cycle 0/1	Integer shift-ALU, multiply, divide and CRC µOPs
Load/Store 0/1	Load, Store address generation and special memory µOPs
Load 2	Load µOps

Instruction groups	Instructions
Integer Store data 0/1	Integer Store data µOPs
FP/ASIMD-0/ Vector Store data 0	ASIMD ALU, ASIMD misc, ASIMD integer multiply, FP convert, FP misc, FP add, FP multiply, FP divide, FP sqrt, ASIMD shift µOps without rounding, saturating or accumulating operations, AES µOps, crypto µOPs, store data µOPs
FP/ASIMD-1/ Vector Store data 1	ASIMD ALU, ASIMD misc, FP convert, FP misc, FP add, FP multiply, ASIMD shift µOPs, ASIMD reduction µOPs, AES µOPs., store data µOPs
CME Operations Block	Up to 4 µOPs sent to CME

2. Instruction characteristics

2.1 Instruction tables

This chapter describes high-level performance characteristics for most Armv9.2-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. Accumulate latency is provided in the same column in parenthesis for pipelines which support late-forwarding of accumulate operands from similar µOPs, allowing a typical sequence of operation (in general multiply)-accumulate µOPs to issue one every N cycles.

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 C1-Pro core microarchitecture.

2.2 Legend for reading the utilized pipelines

Table 2-1: C1-Pro core pipeline names and symbols

Pipeline name	Symbol used in tables
Branch 0/1	B
Integer single Cycle 0/1	S
Integer single Cycle 0/1 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
Integer Store data 0/1	ID
FP/ASIMD/Vector Store data 0/1	V
FP/ASIMD/Vector Store data 0	V0
FP/ASIMD/Vector Store data 1	V1
CME Operations Block	C

2.3 Branch instructions

Table 2-2: AArch64 Branch instructions

Instruction Group	AArch64 Instructions	Exec Latency	Execution Throughput	Utilized Pipelines	Notes
Branch, immed	B	0	2	-	-
Branch conditionally	B.cond, BC.cond	1	2	B	1
Branch, register	BR, RET	1	2	B	-
Branch and link, immed	BL	1	2	S	-
Branch and link, register	BLR	1	2	B, S	-
Compare and branch	CBZ, CBNZ, TBZ, TBNZ	1	2	B	-

Notes:

1. Additional optimizations are detailed in part Branch and Integer fusion.

2.4 Arithmetic and logical instructions

Table 2-3: 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	4	I	-
ALU, basic, flagset	ADDS, ADCS, ANDS, BICS, SUBS, SBCS	1	4	I	-
ALU, extend and shift	ADD{S}, SUB{S}	2	2	M	-
Arithmetic, LSL shift, shift <= 4	ADD, SUB	1	4	I	-
Arithmetic, flagset, LSL shift, shift <= 4	ADDS, SUBS	1	4	I	-
Arithmetic, LSR/ASR/ROR shift or LSL shift > 4	ADD{S}, SUB{S}	2	2	M	-
Arithmetic, immediate to logical address tag	ADDG, SUBG	1	4	I	-
Conditional compare	CCMN, CCMP	1	4	I	-
Conditional select	CSEL, CSINC, CSINV, CSNEG	1	4	I	-
Convert floating-point condition flags	AXFLAG, XAFLAG	1	4	I	-
Flag manipulation instructions	SETF8, SETF16, RMIF, CFINV	1	4	I	-
Insert Random Tags	IRG	2	1	M0	1
Insert Tag Mask	GMI	1	4	I	-
Logical, shift, no flagset	AND, BIC, EON, EOR, ORN, ORR	1	4	I	-
Logical, shift, flagset	ANDS, BICS	2	2	M	-
Subtract Pointer, no/with flagset	SUBP, SUBPS	1	4	I	-

Notes:

1.The latency is 2, throughput is 1 and utilized pipeline is M0 when GCR_EL1.RRND = 1. When GCR_EL1.RRND = 0, the description is not valid, execution throughput and latency are degraded.

2.5 Divide and multiply instructions

Table 2-4: 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 accumulate, W and X-forms	MADD, MSUB	2(1)	1	M0	2
Multiply accumulate long	SMADDL, SMSUBL, UMADDL, UMSUBL	2(1)	1	M0	2
Multiply without accumulate, W and X-forms	MADD, MSUB	2	2	M	3
Multiply without accumulate long	SMADDL, SMSUBL, UMADDL, UMSUBL	2	2	M	3
Multiply high	SMULH, UMULH	3	2	M	2

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. Multiply without accumulate when Ra is ZR `0b11111`, MUL, MNEG, SMULL, SMNEGL, UMULL and UMNEGL instructions can be executed on utilized pipeline M with an execution throughput of 2.

2.6 Pointer Authentication Instructions

Table 2-5: AArch64 pointer authentication instructions

Instruction Group	AArch64 Instructions	Exec Latency	Execution Throughput	Utilized Pipelines	Notes
Authenticate data address	AUTDA, AUTDB, AUTDZA, AUTDZB	1	2	M	

Instruction Group	AArch64 Instructions	Exec Latency	Execution Throughput	Utilized Pipelines	Notes
Authenticate instruction address	AUTIA, AUTIB, AUTIA1716, AUTIB1716, AUTIASP, AUTIBSP, AUTIAZ, AUTIBZ, AUTIZA, AUTIZB	1	2	M	
Branch and link, register, with pointer authentication	BLRAA, BLRAAZ, BLRAB, BLRABZ	2	2	M, B	1
Branch, register, with pointer authentication	BRAA, BRAAZ, BRAB, BRABZ	2	2	M, B	1
Branch, return, with pointer authentication	RETA, RETB	2	2	M, B	1
Compute pointer authentication code for data address	PACDA, PACDB, PACDZA, PACDZB	4	2	M	
Compute pointer authentication code, using generic key	PACGA	4	2	M	
Compute pointer authentication code for instruction address	PACIA, PACIB, PACIA1716, PACIB1716, PACIASP, PACIBSP, PACIAZ, PACIBZ, PACIZA, PACIZB	4	2	M	
Load register, with pointer authentication	LDRAA, LDRAB	5	2	M, L, I	1, 2
Strip pointer authentication code	XPACD, XPACI, XPACLRI	1	2	M	

Notes:

1. In case of AUTH FAIL the description is not valid, execution throughput and latency are degraded.

2. Only Immed pre-index with write back use I pipes

2.7 Miscellaneous data-processing instructions

Table 2-6: AArch64 Miscellaneous data-processing instructions

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

2.8 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 2-7: AArch64 Load instructions

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

Notes:

1. Only Immed pre-index with write back use I pipes

2. Execution Latency is 5 and Utilized Pipelines are L, I when scale with aligned offset of 128 bits

2.9 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 2-8: AArch64 Store instructions

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

2.10 Tag Load Instructions

Table 2-9: AArch64 Tag load instructions

Instruction Group	AArch64 Instructions	Exec Latency	Execution Throughput	Utilized Pipelines	Notes
Load allocation tag	LDG	5	3	L, I	-
Load multiple allocation tags	LDGM	4	3	L	-

2.11 Tag Store instructions

Table 2-10: 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, ID, I	-

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

2.12 FP data processing instructions when not in Streaming SVE mode

Table 2-11: AArch64 FP data processing instructions

Instruction Group	AArch64 Instructions	Exec Latency	Execution Throughput	Utilized Pipelines	Notes
FP absolute value	FABS, FABD	2	2	V	-
FP arithmetic	FADD, FSUB	2	2	V	-
FP compare	FCCMP{E}, FCMP{E}	2	2	V	-
FP divide, H-form	FDIV	5	1	V0	1
FP divide, S-form	FDIV	7	1	V0	1
FP divide, D-form	FDIV	12	1	V0	1
FP min/max	FMIN, FMINNM, FMAX, FMAXNM	2	2	V	-
FP multiply	FMUL, FNMUL	3	2	V	2
FP multiply accumulate	FMADD, FMSUB, FNMADD, FNMSUB	4 (2)	2	V	3
FP negate	FNEG	2	2	V	-
FP round to integral	FRINTA, FRINTI, FRINTM, FRINTN, FRINTP, FRINTX, FRINTZ, FRINT32X, FRINT64X, FRINT32Z, FRINT64Z	3	2	V	-
FP select	FCSEL	2	2	V	-

Instruction Group	AArch64 Instructions	Exec Latency	Execution Throughput	Utilized Pipelines	Notes
FP square root, H-form	FSQRT	5	1	V0	1
FP square root, S-form	FSQRT	7	1	V0	1
FP square root, D-form	FSQRT	12	1	V0	1

Notes:

1. FP divide and square root operations are now performed using a fully pipelined data path.

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 2 cycles 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).

2.13 FP miscellaneous instructions when not in Streaming SVE mode

Table 2-12: 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	2	V, S	-
FP convert, Javascript from vec to gen reg	FJCVTZS	3	2	V, S	-
FP convert, from vec to vec reg	FCVT, FCVTXN	3	2	V	-
FP move, immed	FMOV	2	2	V	1
FP move, register	FMOV	2	2	V	1
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	3	2	V, S	-

Notes:

1. Particular FMOV #0 or Register to Register can be optimized in rename stage pipeline, execution latency and throughput are then not representative.

2.14 FP load instructions when not in Streaming SVE mode

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, two extra cycles are required to forward results to FP/ASIMD pipelines.

Table 2-13: AArch64 FP load instructions

Instruction Group	AArch64 Instructions	Exec Latency	Execution Throughput	Utilized Pipelines	Notes
Load vector reg, literal, S/D/Q forms	LDR	6	3	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	6	3	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	6	3	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	3/2	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	3/2	I, L	-
Load vector pair, immed pre-index, Q-form	LDP	6	3/2	L, I	-

2.15 FP store instructions when not in Streaming SVE mode

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

Table 2-14: 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, V	-
Store vector reg, unscaled immed, Q-form	STUR	2	2	L01, V	-
Store vector reg, immed post-index, B/H/S/D-form	STR	2	2	L01, V, I	-
Store vector reg, immed post-index, Q-form	STR	2	2	L01, V, I	-
Store vector reg, immed pre-index, B/H/S/D-form	STR	3	2	L01, V, I	-
Store vector reg, immed pre-index, Q-form	STR	2	2	L01, V, I	-
Store vector reg, unsigned immed, B/H/S/D-form	STR	2	2	L01, V	-
Store vector reg, unsigned immed, Q-form	STR	2	2	L01, V	-
Store vector reg, register offset, basic, B/H/S/D-form	STR	2	2	L01, V	-
Store vector reg, register offset, basic, Q-form	STR	2	2	L01, V	-
Store vector reg, register offset, scale, H-form	STR	2	2	L01, V	-
Store vector reg, register offset, scale, S/D-form	STR	2	2	L01, V	-
Store vector reg, register offset, scale, Q-form	STR	2	2	I, L01, V	-
Store vector reg, register offset, extend, B/H/S/D-form	STR	2	2	L01, V	-
Store vector reg, register offset, extend, Q-form	STR	2	2	L01, V	-
Store vector reg, register offset, extend, scale, H-form	STR	2	2	L01, V	-
Store vector reg, register offset, extend, scale, S/D-form	STR	2	2	L01, V	-
Store vector reg, register offset, extend, scale, Q-form	STR	2	2	I, L01, V	-
Store vector pair, immed offset, S-form	STP, STNP	2	2	L01, V	-
Store vector pair, immed offset, D-form	STP, STNP	2	2	L01, V	-
Store vector pair, immed offset, Q-form	STP, STNP	2	2	L01, V	-
Store vector pair, immed post-index, S-form	STP	2	2	I, L01, V	-
Store vector pair, immed post-index, D-form	STP	2	2	I, L01, V	-
Store vector pair, immed post-index, Q-form	STP	2	2	I, L01, V	-
Store vector pair, immed pre-index, S-form	STP	2	2	I, L01, V	-
Store vector pair, immed pre-index, D-form	STP	2	2	I, L01, V	-
Store vector pair, immed pre-index, Q-form	STP	2	2	I, L01, V	-

2.16 ASIMD integer instructions

Table 2-15: AArch64 ASIMD integer instructions

Instruction Group	AArch64 Instructions	Exec Latency	Execution Throughput	Utilized Pipelines	Notes
ASIMD absolute diff	SABD, UABD	2	2	V	-
ASIMD absolute diff accum	SABA, UABA	4(1)	1	V1	2
ASIMD absolute diff accum long	SABAL(2), UABAL(2)	4(1)	1	V1	2
ASIMD absolute diff long	SABDL(2), UABDL(2)	2	2	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	2	V	-
ASIMD arith, complex	ADDHN(2), RADDHN(2), RSUBHN(2), SQABS, SQADD, SQNEG, SQSUB, SRHADD, SUBHN(2), SUQADD, UQADD, UQSUB, URHADD, USQADD	2	2	V	-
ASIMD arith, pair-wise	ADDP, SADDLP, UADDLP	2	2	V	-
ASIMD arith, reduce, 4H/4S	ADDV, SADDLV, UADDLV	3	1	V1	-
ASIMD arith, reduce, 8B/8H	ADDV, SADDLV, UADDLV	5	1	V1, V	-
ASIMD arith, reduce, 16B	ADDV, SADDLV, UADDLV	6	1/2	V1	-
ASIMD compare	CMEQ, CMGE, CMGT, CMHI, CMHS, CMLE, CMLT, CMTST	2	2	V	-
ASIMD dot product	SDOT, UDOT	3 (1)	2	V	2
ASIMD dot product using signed and unsigned integers	SUDOT, USDOT	3(1)	2	V	2
ASIMD logical	AND, BIC, EOR, MOV, MVN, NOT, ORN, ORR	2	2	V	-
ASIMD matrix multiply-accumulate	SMMLA, UMMLA, USMMLA	3(1)	2	V	2
ASIMD max/min, basic and pair-wise	SMAX, SMAXP, SMIN, SMINP, UMAX, UMAXP, UMIN, UMINP	2	2	V	-
ASIMD max/min, reduce, 4H/4S	SMAXV, SMINV, UMAXV, UMINV	3	1	V1	-
ASIMD max/min, reduce, 8B/8H	SMAXV, SMINV, UMAXV, UMINV	5	1	V1, V	-
ASIMD max/min, reduce, 16B	SMAXV, SMINV, UMAXV, UMINV	6	1/2	V1	-
ASIMD multiply	MUL, SQDMULH, SQRDMULH	4	1	V0	-
ASIMD multiply accumulate	MLA, MLS	4(1)	1	V0	1
ASIMD multiply accumulate high	SQRDMLAH, SQRDMLSH	4(2)	1	V0	1

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)	1	V0	1
ASIMD multiply accumulate saturating long	SQDMLAL(2), SQDMLSL(2)	4(2)	1	V0	1
ASIMD multiply/multiply long (8x8) polynomial, D-form	PMUL, PMULL(2)	2	1	V0	3
ASIMD multiply/multiply long (8x8) polynomial, Q-form	PMUL, PMULL(2)	2	1	V0	3
ASIMD multiply long	SMULL(2), UMULL(2), SQDMULL(2)	4	1	V0	-
ASIMD pairwise add and accumulate long	SADALP, UADALP	4(1)	1	V1	2, 4
ASIMD shift accumulate	SSRA, SRSRA, USRA, URSRA	4(1)	1	V1	2
ASIMD shift by immed, basic	SHL, SHLL(2), SSHR,USHR	2	2	V	-
ASIMD shift by immed, basic with rounding	SHRN(2), SSHLL(2), SXTL(2), USHLL(2), UXTL(2)	2	1	V1	4
ASIMD shift by immed and insert, basic	SLI, SRI	2	2	V	-
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	1	V1	4
ASIMD shift by register, basic	SSHL, USHL	2	1	V1	4
ASIMD shift by register, complex	SRSHL, SQRSHL, SQSHL, URSHL, UQRSHL, UQSHL	4	1	V1	4

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”.

4. Rounding, saturating or accumulating shift operations play only on V1 pipe.

2.17 ASIMD floating-point instructions

Table 2-16: AArch64 ASIMD floating-point instructions

Instruction Group	AArch64 Instructions	Exec Latency	Execution Throughput	Utilized Pipelines	Notes
ASIMD FP absolute value/ difference	FABS, FABD	2	2	V	-
ASIMD FP arith, normal	FADD, FSUB	2	2	V	-
ASIMD FP compare	FACGE, FACGT, FCMEQ, FCMGE, FCMGT, FCMLE, FCMLT	2	2	V	-
ASIMD FP complex add	FCADD	3	2	V	-
ASIMD FP complex multiply add	FCMLA	4(2)	2	V	1
ASIMD FP convert, long (F16 to F32)	FCVTL(2)	4	1	V	4
ASIMD FP convert, long (F32 to F64)	FCVTL(2)	3	2	V	-
ASIMD FP convert, narrow (F32 to F16)	FCVTN(2)	4	1	V	4
ASIMD FP convert, narrow (F64 to F32)	FCVTN(2), FCVTXN(2)	3	2	V	-
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	V	-
ASIMD FP convert, other, D-form F16 and Q-form F32	FCVTAS, VCVTAU, FCVTMS, FCVTMU, FCVTNS, FCVTNU, FCVTPS, FCVTPU, FCVTZS, FCVTZU, SCVTF, UCVTF	4	1	V	4
ASIMD FP convert, other, Q-form F16	FCVTAS, VCVTAU, FCVTMS, FCVTMU, FCVTNS, FCVTNU, FCVTPS, FCVTPU, FCVTZS, FCVTZU, SCVTF, UCVTF	6	1/2	V	-4
ASIMD FP divide, D-form, F16	FDIV	8	1/4	V0	3
ASIMD FP divide, D-form, F32	FDIV	8	1/2	V0	3
ASIMD FP divide, Q-form, F16	FDIV	12	1/8	V0	3
ASIMD FP divide, Q-form, F32	FDIV	10	1/4	V0	3
ASIMD FP divide, Q-form, F64	FDIV	13	1/2	V0	3
ASIMD FP max/min, normal	FMAX, FMAXNM, FMIN, FMINNM	2	2	V	-
ASIMD FP arith, max/min, pairwise	FADDP, FMAXP, FMAXNMP, FMINP, FMINNMP	3	2	V	-
ASIMD FP max/min, reduce, F32 and D-form F16	FMAXV, FMAXNMV, FMINV, FMINNMV	4	1	V	-
ASIMD FP max/min, reduce, Q-form F16	FMAXV, FMAXNMV, FMINV, FMINNMV	6	2/3	V	-
ASIMD FP multiply	FMUL, FMULX	3	2	V	2, 5
ASIMD FP multiply accumulate	FMLA, FMLS	4(2)	2	V	1

Instruction Group	AArch64 Instructions	Exec Latency	Execution Throughput	Utilized Pipelines	Notes
ASIMD FP multiply accumulate long	FMLAL(2), FMLSL(2)	4(2)	2	V	1
ASIMD FP negate	FNEG	2	2	V	-
ASIMD FP round, D-form F32 and Q-form F64	FRINTA, FRINTI, FRINTM, FRINTN, FRINTP, FRINTX, FRINTZ, FRINT32X, FRINT64X, FRINT32Z, FRINT64Z	3	2	V	-
ASIMD FP round, D-form F16 and Q-form F32	FRINTA, FRINTI, FRINTM, FRINTN, FRINTP, FRINTX, FRINTZ, FRINT32X, FRINT64X, FRINT32Z, FRINT64Z	4	1	V	4
ASIMD FP round, Q-form F16	FRINTA, FRINTI, FRINTM, FRINTN, FRINTP, FRINTX, FRINTZ, FRINT32X, FRINT64X, FRINT32Z, FRINT64Z	6	1/2	V	4
ASIMD FP square root, D-form, F16	FSQRT	8	1/4	V0	3
ASIMD FP square root, D-form, F32	FSQRT	8	1/2	V0	3
ASIMD FP square root, Q-form, F16	FSQRT	12	1/8	V0	3
ASIMD FP square root, Q-form, F32	FSQRT	10	1/4	V0	3
ASIMD FP square root, Q-form, F64	FSQRT	13	1/2	V0	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 2 cycles after the ASIMD FP multiply µOP has been issued.

3. ASIMD FP divide and square root operations are now performed using a fully pipelined data path.

4. ASIMD FP convert operations are performed using several paths on the pipeline.

5 FMULX scalar forms when not in Streaming SVE mode

2.18 ASIMD BFloat16 (BF16) instructions

Table 2-17: 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	V	2

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

Notes:

1. ASIMD 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).

2. ASIMD FP convert operations are performed using several paths on the pipeline.

2.19 ASIMD miscellaneous instructions

Table 2-18: AArch64 ASIMD miscellaneous instructions

Instruction Group	AArch64 Instructions	Exec Latency	Execution Throughput	Utilized Pipelines	Notes
ASIMD bit reverse	RBIT	2	2	V	2
ASIMD bitwise insert	BIF, BIT, BSL	2	2	V	
ASIMD count	CLS, CLZ, CNT	2	2	V	-
ASIMD duplicate, gen reg	DUP	3	1	M0	-
ASIMD duplicate, element	DUP	2	2	V	2
ASIMD extract	EXT	2	2	V	2
ASIMD extract narrow	XTN(2)	2	2	V	
ASIMD extract narrow, saturating	SQXTN(2), SQXTUN(2), UQXTN(2)	4	1	V1	4
ASIMD insert, element to element	INS	2	2	V	2
ASIMD move, FP immed	FMOV	2	2	V	1
ASIMD move, integer immed	MOVI, MVNI	2	2	V	-
ASIMD reciprocal and square root estimate, D-form U32	URECPE, URSQRTE	3	2	V	-
ASIMD reciprocal and square root estimate, Q-form U32	URECPE, URSQRTE	4	1	V	3
ASIMD reciprocal and square root estimate, D-form F32 and scalar forms	FRECPE, FRSQRTE	3	2	V	5
ASIMD reciprocal and square root estimate, D-form F16 and Q-form F32	FRECPE, FRSQRTE	4	1	V	3, 5
ASIMD reciprocal and square root estimate, Q-form F16	FRECPE, FRSQRTE	6	1/2	V	3, 5
ASIMD reciprocal exponent	FRECPX	3	2	V	5
ASIMD reciprocal step	FRECPS, FRSQRTS	4	2	V	5
ASIMD reverse	REV16, REV32, REV64	2	2	V	2
ASIMD table lookup, 1 or 2 table regs	TBL	2	2	V	2

Instruction Group	AArch64 Instructions	Exec Latency	Execution Throughput	Utilized Pipelines	Notes
ASIMD table lookup, 3 table regs	TBL	4	1	V	2
ASIMD table lookup, 4 table regs	TBL	4	2/3	V	2
ASIMD table lookup extension, 1 table reg	TBX	2	2	V	2
ASIMD table lookup extension, 2 table reg	TBX	4	1	V	2
ASIMD table lookup extension, 3 table reg	TBX	6	2/3	V	2
ASIMD table lookup extension, 4 table reg	TBX	6	1/2	V	2
ASIMD transfer, element to gen reg	UMOV, SMOV	3	2	V, S	5
ASIMD transfer, gen reg to element	INS	5	1	M0, V	
ASIMD transpose	TRN1, TRN2	2	2	V	2
ASIMD unzip/zip	UZP1, UZP2, ZIP1, ZIP2	2	2	V	2

Notes:

1. Particular FMOV #0 or Register to Register can be optimized in rename stage pipeline, execution latency and throughput are then not representative.

2 PERM instructions part of a particular region forwarding

3. ASIMD FP convert operations are performed using several paths on the pipeline.

4. Rounding, saturating or accumulating shift operations play only on V1 pipe.

5 When not in Streaming SVE mode

2.20 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, two extra cycles are required to forward results to FP/ASIMD pipelines.

Table 2-19: 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	2	L, V	-

Instruction Group	AArch64 Instructions	Exec Latency	Execution Throughput	Utilized Pipelines	Notes
ASIMD load, 1 element, one lane, D	LD1	8	2	L, V	-
ASIMD load, 1 element, all lanes, D-form, B/H/S	LD1R	6	3	L	-
ASIMD load, 1 element, all lanes, D-form, D	LD1R	6	3	L	-
ASIMD load, 1 element, all lanes, Q-form	LD1R	6	3	L	-
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	-
ASIMD load, 2 element, one lane, D	LD2	8	2	L, V	-
ASIMD load, 2 element, all lanes, D-form, B/H/S	LD2R	6	3/2	L	-
ASIMD load, 2 element, all lanes, D-form, D	LD2R	6	3/2	L	-
ASIMD load, 2 element, all lanes, Q-form	LD2R	6	3/2	L	-
ASIMD load, 3 element, multiple, D-form, B/H/S	LD3	8	2/3	L, V	-
ASIMD load, 3 element, multiple, Q-form, B/H/S	LD3	10	2/3	L, V	-
ASIMD load, 3 element, multiple, Q-form, D	LD3	10	2/3	L, V	-
ASIMD load, 3 element, one lane, B/H	LD3	8	2/3	L, V	-
ASIMD load, 3 element, one lane, S	LD3	8	2/3	L, V	-
ASIMD load, 3 element, one lane, D	LD3	8	2/3	L, V	-
ASIMD load, 3 element, all lanes, D-form, B/H/S	LD3R	6	1	L	-
ASIMD load, 3 element, all lanes, D-form, D	LD3R	6	1	L	-
ASIMD load, 3 element, all lanes, Q-form, B/H/S	LD3R	6	1	L	-
ASIMD load, 3 element, all lanes, Q-form, D	LD3R	6	1	L	-
ASIMD load, 4 element, multiple, D-form, B/H/S	LD4	8	1/2	L, V	-
ASIMD load, 4 element, multiple, Q-form, B/H/S	LD4	8	1/2	L, V	-
ASIMD load, 4 element, multiple, Q-form, D	LD4	8	1/2	L, V	-
ASIMD load, 4 element, one lane, B/H	LD4	8	1/2	L, V	-
ASIMD load, 4 element, one lane, S	LD4	8	1/2	L, V	-
ASIMD load, 4 element, one lane, D	LD4	8	1/2	L, V	-
ASIMD load, 4 element, all lanes, D-form, B/H/S	LD4R	8	2/3	L, V	-
ASIMD load, 4 element, all lanes, D-form, D	LD4R	8	1/2	L, V	-
ASIMD load, 4 element, all lanes, Q-form, B/H/S	LD4R	8	2/3	L, V	-
ASIMD load, 4 element, all lanes, Q-form, D	LD4R	8	1/2	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.

2.21 ASIMD store instructions

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

Table 2-20: 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, V	-
ASIMD store, 1 element, multiple, 1 reg, Q-form	ST1	2	2	L01, V	-
ASIMD store, 1 element, multiple, 2 reg, D-form	ST1	2	2	L01, V	-
ASIMD store, 1 element, multiple, 2 reg, Q-form	ST1	2	2	L01, V	-
ASIMD store, 1 element, multiple, 3 reg, D-form	ST1	2	1	L01, V	-
ASIMD store, 1 element, multiple, 3 reg, Q-form	ST1	2	1	L01, V	-
ASIMD store, 1 element, multiple, 4 reg, D-form	ST1	2	1	L01, V	-
ASIMD store, 1 element, multiple, 4 reg, Q-form	ST1	2	1	L01, V	-
ASIMD store, 1 element, one lane, B/H/S	ST1	2	2	L01, V	-
ASIMD store, 1 element, one lane, D	ST1	2	2	L01, V	-
ASIMD store, 2 element, multiple, D-form, B/H/S	ST2	2	2	V, L01	-
ASIMD store, 2 element, multiple, Q-form, B/H/S	ST2	2	2	V, L01	-
ASIMD store, 2 element, multiple, Q-form, D	ST2	2	2	V, L01	-
ASIMD store, 2 element, one lane, B/H/S	ST2	2	2	V, L01	-
ASIMD store, 2 element, one lane, D	ST2	2	2	V, L01	-
ASIMD store, 3 element, multiple, D-form, B/H/S	ST3	4	1	V, L01	-
ASIMD store, 3 element, multiple, Q-form, B/H/S	ST3	4	2/3	V, L01	-
ASIMD store, 3 element, multiple, Q-form, D	ST3	2	2/3	V, L01	-
ASIMD store, 3 element, one lane, B/H	ST3	2	1	V, L01	-
ASIMD store, 3 element, one lane, S	ST3	2	1	V, L01	-
ASIMD store, 3 element, one lane, D	ST3	2	1	V, L01	-
ASIMD store, 4 element, multiple, D-form, B/H/S	ST4	4	1	V, L01	-
ASIMD store, 4 element, multiple, Q-form, B/H/S	ST4	4	1/2	V, L01	-
ASIMD store, 4 element, multiple, Q-form, D	ST4	2	1	V, L01	-

Instruction Group	AArch64 Instructions	Exec Latency	Execution Throughput	Utilized Pipelines	Notes
ASIMD store, 4 element, one lane, B/H/S	ST4	2	1	V, L01	-
ASIMD store, 4 element, one lane, D	ST4	2	1	V, 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).

2.22 Cryptography extensions

Table 2-21: AArch64 Cryptography extensions

Instruction Group	AArch64 Instructions	Exec Latency	Execution Throughput	Utilized Pipelines	Notes
Crypto AES ops	AESD, AESE, AESIMC, AESMC	2	2	V	-
Crypto polynomial (64x64) multiply long	PMULL (2)	2	1	V0	-
Crypto SHA1 hash acceleration ops	SHA1H, SHA1C, SHA1M, SHA1P	2	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	2	V	2
Crypto SM3 ops	SM3PARTW1, SM3PARTW2SM3SS1, SM3TT1A, SM3TT1B, SM3TT2A, SM3TT2B	2	1	V0	-
Crypto SM4 ops	SM4E, SM4EKEY	4	1	V0	-

Notes:

1. Adjacent AESE/AESMC instruction pairs and adjacent AESD/AESIMC instruction pairs will exhibit the performance characteristics described in Section 3.6.

2. SHA3 ops are executed from the ALU pipeline

2.23 CRC

Table 2-22: AArch64 CRC

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

2.24 SVE Predicate instructions

Table 2-23: SVE Predicate Instructions

Instruction Group	SVE Instruction	Exec Latency	Execution Throughput	Utilized Pipelines	Notes
Loop control, based on predicate	BRKA, BRKB	2	2	M	-
Loop control, based on predicate and flag setting	BRKAS, BRKBS	2	2	M	-
Loop control, propagating	BRKN, BRKPA, BRKPB	2	2	M	-
Loop control, propagating and flag setting	BRKNS, BRKPAS, BRKPBS	2	2	M	-
Loop control, based on GPR	WHILEGE, WHILEGT, WHILEHI, WHILEHS, WHILELE, WHILELO, WHILELS, WHILELT, WHILERW, WHILEWR	1	2	M	-
Loop terminate	CTERMEQ, CTERMNE	1	2	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	1	4	I	-
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	2	M	--
Predicate logical, flag setting	ANDS, BICS, EORS, MOV, NANDS, NORS, NOTS, ORNS, ORRS	1	2	M	
Predicate reverse	REV	2	2	M	-
Predicate select	SEL	1	2	M	-
Predicate set	PFALSE, PTRUE	2	2	M	1
Predicate set/initialize, set flags	PTRUES	2	2	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. Operation leading to all or none element active are optimized in rename stage pipeline, execution latency and throughput are then not representative.

2.25 SVE floating-point instructions when not in Streaming SVE mode

Table 2-24: SVE floating-point instructions when not in Streaming SVE mode

Instruction Group	SVE Instruction	Exec Latency	Execution Throughput	Utilized Pipelines	Notes
Floating point absolute value/difference	FABD, FABS	2	2	V	-
Floating point arithmetic	FADD, FNEG, FSUB, FSUBR	2	2	V	-
Floating point associative add, F16	FADDA	16	1/4	V	-
Floating point associative add, F32	FADDA	8	1/2	V	-
Floating point associative add, F64	FADDA	4	1	V	-
Floating point compare	FACGE, FACGT, FACLE, FACLT, FCMEQ, FCMGE, FCMGT, FCMLE, FCMLT, FCMNE, FCMUO	2	2	V	-
Floating point complex add	FCADD	3	2	V	-
Floating point complex multiply add	FCMLA	4(2)	2	V	1
Floating point convert, long or narrow (F16 to F32 or F32 to F16)	FCVT, FCVTLT, FCVTNT	4	1	V	3
Floating point convert, long or narrow (F16 to F64, F32 to F64, F64 to F32 or F64 to F16)	FCVT, FCVTLT, FCVTNT	3	2	V	-
Floating point convert, round to odd	FCVTX, FCVTXNT	3	2	V	-
Floating point base2 log, F16	FLOGB	6	1/2	V	3
Floating point base2 log, F32	FLOGB	4	1	V	3
Floating point base2 log, F64	FLOGB	3	2	V	
Floating point convert to integer, F16	FCVTZS, FCVTZU	6	1/2	V	3
Floating point convert to integer, F32	FCVTZS, FCVTZU	4	1	V	3
Floating point convert to integer, F64	FCVTZS, FCVTZU	3	2	V	
Floating point copy	FCPY, FDUP, FMOV	2	2	V	-
Floating point divide, F16	FDIV, FDIVR	12	1/8	V0	2
Floating point divide, F32	FDIV, FDIVR	10	1/4	V0	2
Floating point divide, F64	FDIV, FDIVR	13	1/2	V0	2
Floating point arith, min/max pairwise	FADDP, FMAXP, FMAXNMP, FMINP, FMINNMP	3	2	V	
Floating point min/max	FMAX, DMIN, FMAXNM, FMINNM	2	2	V	-
Floating point multiply	FSCALE, FMUL, FMULX	3	2	V	-
Floating point multiply accumulate	FMLA, FMLS, FMAD, FMSB, FNMAD, FNMLA, FNMLS, FNMSB	4(2)	2	V	1
Floating point multiply add/sub accumulate long	FMLALB, FMLALT, FMLSLB, FMLSLT	4(2)	2	V	1
Floating point reciprocal estimate, F16	FRECPE, FRECPX, FRSQRTE	6	1/2	V	3
Floating point reciprocal estimate, F32	FRECPE, FRECPX, FRSQRTE	4	1	V	3

Instruction Group	SVE Instruction	Exec Latency	Execution Throughput	Utilized Pipelines	Notes
Floating point reciprocal estimate, F64	FRECPE, FRECPX, FRSQRTE	3	2	V	-
Floating point reciprocal step	FRECPS, FRSQRTS	4	2	V	-
Floating point reduction, F16	FADDV, FMAXNMV, FMAXV, FMINNMV, FMINV	12	2/3	V	-
Floating point reduction, F32	FADDV, FMAXNMV, FMAXV, FMINNMV, FMINV	9	1	V	-
Floating point reduction, F64	FADDV, FMAXNMV, FMAXV, FMINNMV, FMINV	6	2	V	-
Floating point round to integral, F16	FRINTA, FRINTM, FRINTN, FRINTP, FRINTX, FRINTZ	6	1/2	V	3
Floating point round to integral, F32	FRINTA, FRINTM, FRINTN, FRINTP, FRINTX, FRINTZ	4	1	V	3
Floating point round to integral, F64	FRINTA, FRINTM, FRINTN, FRINTP, FRINTX, FRINTZ	3	2	V	-
Floating point square root, F16	FSQRT	12	1/8	V0	2
Floating point square root, F32	FSQRT	10	1/4	V0	2
Floating point square root F64	FSQRT	13	1/2	V0	2
Floating point trigonometric exponentiation	FEXPA	2	2	V	
Floating point trigonometric multiply add	FTMAD	4	2	V	
Floating point trigonometric, miscellaneous	FTSMUL, FTSSEL	3	2	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 FP divide and square root operations are now performed using a fully pipelined data path.

3. ASIMD FP convert operations are performed using several paths on the pipeline.

2.26 SVE BFloat16 (BF16) instructions when not in Streaming SVE mode

Table 2-25: SVE Bfloat16 (BF16) instructions when not in Streaming SVE mode

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

2.27 SVE integer instructions when not in Streaming SVE mode

Table 2-26: SVE integer instructions when not in Streaming SVE mode

Instruction Group	SVE Instruction	Exec Latency	Execution Throughput	Utilized Pipelines	Notes
Arithmetic, absolute diff	SABD, UABD	2	2	V	-
Arithmetic, absolute diff accum	SABA, UABA	4(1)	1	V1	1
Arithmetic, absolute diff accum long	SABALB, SABALT, UABALB, UABALT	4(1)	1	V1	1
Arithmetic, absolute diff long	SABDLB, SABDLT, UABDLB, UABDLT	2	2	V	-
Arithmetic, basic	ABS, ADD, ADR, CNOT, NEG, SADDLB, SADDLBT, SADDLT, SADDWB, SADDWT, SHADD, SHSUB, SHSUBR, SSUBLB, SSUBLBT, SSUBLT, SSUBLTB, SSUBWB, SSUBWT, SUB, SUBHNB, SUBHNT, SUBR, UADDLB, UADDLT, UADDWB, UADDWT, UHADD, UHSUB, UHSUBR, USUBLB, USUBLT, USUBWB, USUBWT	2	2	V	-
Arithmetic, complex	ADDHNB, ADDHNT, RADDHNB, RADDHNT, RSUBHNB, RSUBHNT, SQABS, SQADD, SQNEG, SQSUB, SQSUBR, SRHADD, SUQADD, UQADD, UQSUB, UQSUBR, USQADD, URHADD	2	2	V	-
Arithmetic, large integer	ADCLB, ADCLT, SBCLB, SBCLT	2	2	V	-
Arithmetic, pairwise add	ADDP	2	2	V	-
Arithmetic, pairwise add and accum long	SADALP, UADALP	4(1)	1	V1	1
Arithmetic, shift by vector or by wide elements	ASR, ASRR, LSL, LSLR, LSR, LSRR	2	2	V	-
Arithmetic, shift by immediate predicated	ASR, LSL, LSR	2	1	V1	6
Arithmetic, shift by immediate unpredicated	ASR, LSL, LSR	2	2	V	-
Arithmetic, shift and accumulate	SRSRA, SSRA, URSRA, USRA	4(1)	1	V1	6
Arithmetic, shift by immediate	SSHLLB, SSHLLT, USHLLB, USHLLT	2	2	V	-
Arithmetic, shift right narrow by immediate	SHRNB, SHRNT	2	1	V1	6

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

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

Instruction Group	SVE Instruction	Exec Latency	Execution Throughput	Utilized Pipelines	Notes
Multiply accumulate, B, H, S element size	MLA, MLS	4(1)	1	V0	1
Multiply accumulate, D element size	MLA, MLS, MAD, MSB,	4(3)	1	V0	1
Multiply accumulate long	SMLALB, SMLALT, SMLSLB, SMLSLT, UMLALB, UMLALT, UMLSLB, UMLSLT	4(1)	1	V0	1
Multiply accumulate saturating doubling long regular	SQDMLALB, SQDMLALT, SQDMLALBT, SQDMLSLB, SQDMLSLT, SQDMLSLBT	4(3)	1	V0	2
Multiply saturating doubling high, B, H, S, D element size	SQDMULH	4	1	V0	-
Multiply saturating doubling long	SQDMULLB, SQDMULLT	4	1	V0	-
Multiply saturating rounding doubling regular/complex accumulate, B, H, S, D element size	SQRDMLAH, SQRDMLSH, SQRDCMLAH	4(3)	1	V0	2
Multiply saturating rounding doubling regular/complex, B, H, S, D element size	SQRDMULH	4	1	V0	-
Multiply/multiply long, (8x8) polynomial	PMUL, PMULLB, PMULLT	2	1	V0	-
Predicate counting vector	CNT, DECB, DECH, DECW, DECD, INCB, INCH, INCW, INCD, SQDECB, SQDECH, SQDECW, SQDECD, SQINCB, SQINCH, SQINCW, SQINCD, UQDECB, UQDECH, UQDECW, UQDECD, UQINCB, UQINCH, UQINCW, UQINCD	2	2	V	-
Reciprocal estimate for B	URECPE, URSQRTE	4	2	V	
Reciprocal estimate for H	URECPE, URSQRTE	6	1	V	
Reduction, arithmetic, B form	SADDV, UADDV, SMAXV, SMINV, UMAXV, UMINV	8	1/2	V, V1	3
Reduction, arithmetic, H form	SADDV, UADDV, SMAXV, SMINV, UMAXV, UMINV	7	1	V, V1	3
Reduction, arithmetic, S form	SADDV, UADDV, SMAXV, SMINV, UMAXV, UMINV	5	2	V	
Reduction, arithmetic, D form	UADDV, SMAXV, SMINV, UMAXV, UMINV	4	2	V	
Reduction, logical	ANDV, EORV, ORV	5	1	V, V1	-
Reverse, vector	REV, REVB, REVH, REVW	2	2	V	-
Select, vector form	MOV, SEL	2	2	V	-
Table lookup	TBL	2	2	V	-
Table lookup extension	TBX	2	2	V	-
Transpose, vector form	TRN1, TRN2	2	2	V	-

Instruction Group	SVE Instruction	Exec Latency	Execution Throughput	Utilized Pipelines	Notes
Unpack and extend	SUNPKHI, SUNPKLO, UUNPKHI, UUNPKLO	2	2	V	-
Zip/unzip	UZP1, UZP2, ZIP1, ZIP2	2	2	V	-

Notes:

1. 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).

2. Same as 1 except that for saturating instructions require an extra cycle of latency for late-forwarding accumulate operands.

3. Signed Additions need 2 cycles more

4. ASIMD FP convert operations are performed using several paths on the pipeline.

5. Rounding, saturating or accumulating shift operations play only on V1 pipe.

2.28 SVE Load instructions when not in Streaming SVE mode

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.

Table 2-27: SVE Load instructions when not in Streaming SVE mode

Instruction Group	SVE Instruction	Exec Latency	Execution Throughput	Utilized Pipelines	Notes
Load vector	LDR	6	3	L	-
Load predicate	LDR	5	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	7	3/4	L	-

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	6	4/5	L	-
Contiguous first faulting load, scalar + scalar	LDFF1B, LDFF1D, LDFF1H, LDFF1W, LDFF1SB, LDFF1SD, LDFF1SH LDFF1SW	6	3	L	-
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	2	V, L	-
Contiguous Load two structures to two vectors, scalar + scalar	LD2B, LD2D, LD2H, LD2W	8	2	V, L	-
Contiguous Load three structures to three vectors, scalar + imm	LD3D	8	2/3	V, L	-
Contiguous Load three structures to three vectors, scalar + imm	LD3B, LD3H, LD3W	10	1/3	V, L	
Contiguous Load three structures to three vectors, scalar + scalar	LD3D	9	2/3	V, L, I	-
Contiguous Load three structures to three vectors, scalar + scalar	LD3B, LD3W, LD3H	11	1/3	V, L, I	-
Contiguous Load four structures to four vectors, scalar + imm	LD4D	8	1/2	V, L	-
Contiguous Load four structures to four vectors, scalar + imm	LD4B, LD4H, LD4W	12	2/5	V, L	-
Contiguous Load four structures to four vectors, scalar + scalar	LD4D	9	1/2	L, V, I	-
Contiguous Load four structures to four vectors, scalar + scalar	LD4B, LD4H, LD4W	13	2/5	L, V, I	-
Gather load, vector + imm, 32-bit element size	LD1B, LD1H, LD1W, LD1SB, LD1SH, LD1SW, LDFF1B, LDFF1H, LDFF1W, LDFF1SB, LDFF1SH, LDFF1SW	7	3/4	L	-
Gather load, vector + imm, 64-bit element size	LD1B, LD1D, LD1H, LD1W, LD1SB, LD1SH, LD1SW, LDFF1B, LDFF1D LDFF1H, LDFF1W, LDFF1SB, LDFF1SD, LDFF1SH, LDFF1SW	6	4/5	L	-
Gather load, 32-bit scaled, unscaled offset	LD1H, LD1SH, LDFF1H, LDFF1SH, LD1W, LDFF1W, LDFF1SW	7	3/4	L	-
Gather load, 32-bit unpacked unscaled offset, 64 bit scaled, unscaled offset	LD1B, LD1SB, LDFF1B, LDFF1SB, LD1D, LDFF1D, LD1H, LD1SH, LDFF1H, LDFF1SH, LD1W, LD1SW, LDFF1W, LDFF1SW	6	4/5	L	-
Gather load, 32-bit unscaled offset	LD1B, LD1SB, LDFF1B, LDFF1SB	7	3/4	L	-
Gather load, 32-bit unpacked unscaled offset, 64 bit unscaled offset	LD1B, LD1SB, LDFF1B, LDFF1SB	6	4/5	L	-

2.29 SVE Store instructions when not in Streaming SVE mode

Table 2-28: SVE Store instructions when not in Streaming SVE mode

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, V	-
Contiguous store, scalar + imm	ST1B, ST1H, ST1D, ST1W	2	2	L01, V	-
Contiguous store, scalar + scalar	ST1H	2	2	L01, I, V	-
Contiguous store, scalar + scalar	ST1B, ST1D, ST1W	2	2	L01, V	-
Contiguous store two structures from two vectors, scalar + imm	ST2B, ST2H, ST2D, ST2W	2	2	L01, V	-
Contiguous store two structures from two vectors, scalar + scalar	ST2B, ST2D, ST2H, ST2W	2	2	L01, V	-
Contiguous store three structures from three vectors, scalar + imm	ST3B, ST3D, ST3H, ST3W	4	2/3	L01, V	-
Contiguous store three structures from three vectors, scalar + imm	ST3D	3	2/3	L01, V	-
Contiguous store three structures from three vectors, scalar + scalar	ST3B, ST3H, ST3W	4	2/3	L01, I, V	-
Contiguous store three structures from three vectors, scalar + scalar	ST3D	3	2/3	L01, I, V	-
Contiguous store four structures from four vectors, scalar + imm	ST4B, ST4H, ST4W	6	2/3	L01, V	-
Contiguous store four structures from four vectors, scalar + imm	ST4D	3	1/2	L01, V	-
Contiguous store four structures from four vectors, scalar + scalar	ST4D	3	1/2	L01, I, V	-
Contiguous store four structures from four vectors, scalar + scalar	ST4B, ST4H, ST4W	6	2/3	L01, I, V	-
Non temporal store, scalar + imm	STNT1B, STNT1D, STNT1H, STNT1W	2	2	L01, V	-
Non temporal store, scalar + scalar	STNT1B, STNT1D, STNT1H, STNT1W	2	2	L01, V	-
Scatter non temporal store, vector + scalar 32-bit element size	STNT1B, STNT1H, STNT1W	2	1	L01, V	-
Scatter non temporal store, vector + scalar 64-bit element size	STNT1B, STNT1D, STNT1H, STNT1W	2	2	L01, V	-
Scatter store vector + imm 32-bit element size	ST1B, ST1H, ST1W	2	1	L01, V	-
Scatter store vector + imm 64-bit element size	ST1B, ST1D, ST1H, ST1W	2	2	L01, V	-
Scatter store, 32-bit scaled offset	ST1H, ST1W	2	1	L01, V	-
Scatter store, 32-bit unpacked unscaled offset	ST1B, ST1D, ST1H, ST1W	2	2	L01, V	-
Scatter store, 32-bit unpacked scaled offset	ST1D, ST1H, ST1W	2	2	L01, V	-

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

2.30 SVE Miscellaneous instructions

Table 2-29: SVE miscellaneous instructions

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

Notes:

1. Operation are optimized in rename stage pipeline, execution latency and throughput are then not representative.

2.31 SVE Cryptographic instructions when not in Streaming SVE mode

Table 2-30: SVE cryptographic instructions when not in Streaming SVE mode

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

2.32 SVE instructions added by SME and available when not in Streaming SVE mode

Table 2-31: SME instructions available when not in Streaming SVE mode

Instruction Group	SME Instructions	Exec Latency	Execution Throughput	Utilized Pipelines	Notes
BFloat16 floating-point multiply-subtract long from single-precision vector and indexed forms	BFMLSLB, BFMLSLT	4	2	V	-
Floating-point clamp to minimum/maximum number	FCLAMP	2	2	V	-
Half-precision floating-point indexed or vector forms dot product	FDOT	4	2	V	-
Predicate select between predicate register or all-false	PSEL	2	2	M	-
Reverse 64-bit doublewords in elements	REVD	2	2	V	-
Range Prefetch Memory	RPRFM	4	3	L	-
Signed or unsigned clamp to minimum/maximum vector	SCLAMP, UCLAMP	2	2	V	-
Signed or unsigned integer indexed or vector forms dot product	SDOT, UDOT	3	2	V	-
Signed or unsigned saturating (unsigned) extract narrow and interleave	SQCVTN, SQCTUN, UQCVTN	2	2	V	-
Signed saturating rounding shift right (unsigned) narrow by immediate and interleave	SQRSHRN, SQRSHRUN	6	1	V1	-
Unsigned saturating rounding shift right narrow by immediate and interleave	UQRSHRN	4	1	V1	-
Loop control, based on GPR generating predicate pair	WHILEGE, WHILEGT, WHILEHI, WHILEHS, WHILELE, WHILELO, WHILELS, WHILELT	2	2	M	-

2.33 SVE instructions added by SME but not sent to CME when in Streaming SVE mode

Table 2-32: SME instructions not sent to CME when in Streaming SVE mode

Instruction Group	SME Instructions	Exec Latency	Execution Throughput	Utilized Pipelines	Notes
Add/Read multiple of Streaming SVE predicate/vector register size to scalar register	ADDSPL, ADDSVL, RDSVL	1	4	I	-
Predicate counting scalar to count from predicate-as-counter	CNTP	2	2	M	-
Set pair of predicates from predicate-as-counter	PEXT	4	2	M	-
Predicate as counter set	PTRUE	1	2	M	-

Instruction Group	SME Instructions	Exec Latency	Execution Throughput	Utilized Pipelines	Notes
Loop control, based on GPR generating predicate as counter	WHILEGE, WHILEGT, WHILEHI, WHILEHS, WHILELE, WHILELO, WHILELS, WHILELT	1	2	M	-

2.34 FP/ASIMD/SVE/SME instructions sent to CME when in Streaming SVE mode

The following instructions are sent to CME when in Streaming SVE mode thus are not executed in the out of order part of the machine unless explicitly using a pipeline on this side of the machine, for example, Loads and Stores. The execution latency is then not meaningful (NA = Not Applicable). The execution throughput relates to the bandwidth of execution in the Core and to the bandwidth of instructions that can be sent to CME. Most of the following instructions utilizes only the pipeline C and thus do not utilize the out of order scheduling and execution resources of the machine.

2.34.1 FP data processing instructions when in Streaming SVE mode

Table 2-33: AArch64 FP data processing instructions when in Streaming SVE mode

Instruction Group	AArch64 Instructions	Exec Latency	Execution Throughput	Utilized Pipelines	Notes
FP absolute value	FABS, FABD	NA	4	C	-
FP arithmetic	FADD, FSUB	NA	4	C	-
FP compare	FCCMP{E}, FCMP{E}	NA	4	C	1
FP divide, H, S, D-form	FDIV	NA	4	C	-
FP min/max	FMIN, FMINNM, FMAX, FMAXNM	NA	4	C	-
FP multiply	FMUL, FNMUL	NA	4	C	-
FP multiply accumulate	FMADD, FMSUB, FNMADD, FNMSUB	NA	4	C	-
FP negate	FNEG	NA	4	C	-
FP round to integral	FRINTA, FRINTI, FRINTM, FRINTN, FRINTP, FRINTX, FRINTZ, FRINT32X, FRINT64X, FRINT32Z, FRINT64Z	NA	4	C	-
FP select	FCSEL	NA	2	S, C	-
FP square root, H, S, D-form	FSQRT	NA	4	C	-

Notes:

1. Those instructions sent to CME when in Streaming SVE mode are writing flags or general purpose or predicate registers and will impact the return bandwidth CME to Core, refer to the CME SWOG for more information.

Table 2-34: AArch64 FP miscellaneous instructions when in Streaming SVE mode

Instruction Group	AArch64 Instructions	Exec Latency	Execution Throughput	Utilized Pipelines	Notes
FP convert, from gen to vec reg	SCVTF, UCVTF	NA	2	M, C	-
FP convert, from vec to gen reg	FCVTAS, FCVTAU, FCVTMS, FCVTMU, FCVTNS, FCVTNU, FCVTPS, FCVTPU, FCVTZS, FCVTZU	NA	4	C	1
FP convert, from vec to vec reg	FCVT, FCVTXN	NA	4	C	-
FP move, immed	FMOV	NA	4	C	-
FP move, register	FMOV	NA	4	C	-
FP transfer, from gen to low half of vec reg	FMOV	NA	2	M, C	-
FP transfer, from gen to high half of vec reg	FMOV	NA	2	M, C	-
FP transfer, from vec to gen reg	FMOV	NA	4	C	-

Notes:

1. Those instructions sent to CME when in Streaming SVE mode are writing flags or general purpose or predicate registers and will impact the return bandwidth CME to Core, refer to the CME SWOG for more information.

2.34.2 FP load instructions when in Streaming SVE mode

Table 2-35: AArch64 FP load instructions when in Streaming SVE mode

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

2.34.3 FP store instructions when in Streaming SVE mode

Table 2-36: AArch64 FP store instructions when in Streaming SVE mode

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

2.34.4 ASIMD floating-point and miscellaneous instructions when in Streaming SVE mode

Table 2-37: AArch64 ASIMD integer instructions when in Streaming SVE mode

Instruction Group	AArch64 Instructions	Exec Latency	Execution Throughput	Utilized Pipelines	Notes
ASIMD FP multiply scalar forms	FMULX	NA	4	C	-
ASIMD reciprocal and square root estimate scalar forms	FRECPE, FRSQRTE	NA	4	C	-
ASIMD reciprocal exponent	FRECPX	NA	4	C	-
ASIMD reciprocal step scalar forms	FRECPS, FRSQRTS	NA	4	C	-
ASIMD transfer, element to gen reg	UMOV, SMOV	NA	4	C	1

Notes:

1. Those instructions sent to CME when in Streaming SVE mode are writing flags or general purpose or predicate registers and will impact the return bandwidth CME to Core, refer to the CME SWOG for more information.

2.34.5 SVE integer instructions when in Streaming SVE mode

Table 2-38: SVE integer instructions when in Streaming SVE mode

Instruction Group	SVE Instruction	Exec Latency	Execution Throughput	Utilized Pipelines	Notes
Arithmetic, absolute diff	SABD, UABD	NA	4	C	-
Arithmetic, absolute diff accum	SABA, UABA	NA	4	C	-
Arithmetic, absolute diff accum long	SABALB, SABALT, UABALB, UABALT	NA	4	C	-
Arithmetic, absolute diff long	SABDLB, SABDLT, UABDLB, UABDLT	NA	4	C	-
Arithmetic, basic	ABS, ADD, CNOT, NEG, SADDLB, SADDLBT, SADDLT, SADDWB, SADDWT, SHADD, SHSUB, SHSUBR, SSUBLB, SSUBLBT, SSUBLT, SSUBLTB, SSUBWB, SSUBWT, SUB, SUBHNB, SUBHNT, SUBR, UADDLB, UADDLT, UADDWB, UADDWT, UHADD, UHSUB, UHSUBR, USUBLB, USUBLT, USUBWB, USUBWT	NA	4	C	-
Arithmetic, complex	ADDHNB, ADDHNT, RADDHNB, RADDHNT, RSUBHNB, RSUBHNT, SQABS, SQADD, SQNEG, SQSUB, SQSUBR, SRHADD, SUQADD, UQADD, UQSUB, UQSUBR, USQADD, URHADD	NA	4	C	-
Arithmetic, large integer	ADCLB, ADCLT, SBCLB, SBCLT	NA	4	C	-
Arithmetic, pairwise add	ADDP	NA	4	C	-
Arithmetic, pairwise add and accum long	SADALP, UADALP	NA	4	C	-
Arithmetic, shift by vector or by wide elements	ASR, ASRR, LSL, LSLR, LSR, LSRR	NA	4	C	-
Arithmetic, shift by immediate predicated	ASR, LSL, LSR	NA	4	C	-
Arithmetic, shift by immediate unpredicated	ASR, LSL, LSR	NA	4	C	-
Arithmetic, shift and accumulate	SRSRA, SSRA, URSRA, USRA	NA	4	C	-
Arithmetic, shift by immediate	SSHLLB, SSHLLT, USHLLB, USHLLT	NA	4	C	-
Arithmetic, shift right narrow by immediate	SHRNB, SHRNT	NA	4	C	-
Arithmetic, shift by immediate and insert	SLI, SRI	NA	4	C	-
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	NA	4	C	-
Arithmetic, shift right for divide	ASRD	NA	4	C	-
Arithmetic, shift rounding	SRSHL, SRSHLR, SRSHR, URSHL, URSHLR, URSHR	NA	4	C	-
Bitwise select	BSL, BSL1N, BSL2N, NBSL	NA	4	C	-

Instruction Group	SVE Instruction	Exec Latency	Execution Throughput	Utilized Pipelines	Notes
Count/reverse bits	CLS, CLZ, CNT, RBIT	NA	4	C	-
Broadcast logical bitmask immediate to vector	DUPM, MOV	NA	4	C	-
Compare and set flags	CMPEQ, CMPGE, CMPGT, CMPHI, CMPHS, CMPLE, CMPLO, CMPLS, CMPLT, CMPNE	NA	4	C	1
Complex add	CADD, SQCADD	NA	4	C	-
Complex dot product 8-bit and 16 bit elements vector and indexed forms	CDOT	NA	4	C	-
Complex multiply-add B, H, S element size	CMLA	NA	4	C	-
Complex multiply-add D element size	CMLA	NA	4	C	-
Conditional extract operations, scalar form	CLASTA, CLASTB	NA	4	C	1
Conditional extract operations, SIMD&FP scalar and vector forms	CLASTA, CLASTB, SPLICE	NA	4	C	-
Convert to floating point, 64b to float or convert to double, 32b to single or half, 16b to half	SCVTF, UCVTF	NA	4	C	-
Copy, scalar	CPY	NA	2	M, C	-
Copy, scalar SIMD&FP or imm	CPY	NA	4	C	-
Divides, 32 bit, 64 bit	SDIV, SDIVR, UDIV, UDIVR	NA	4	C	-
Dot product, 8 bit and 16 bit vector and indexed forms	SDOT, UDOT	NA	4	C	-
Dot product, 8 bit, using signed and unsigned integers	SUDOT, USDOT	NA	4	C	-
Duplicate, immediate and indexed form	DUP, MOV	NA	4	C	-
Duplicate, scalar form	DUP, MOV	NA	2	M, C	-
Extend, sign or zero	SXTB, SXTH, SXTW, UXTB, UXTH, UXTW	NA	4	C	-
Extract	EXT	NA	4	C	-
Extract narrow saturating	SQXTNB, SQXTNT, SQXTUNB, SQXTUNT, UQXTNB, UQXTNT	NA	4	C	-
Extract/insert operation, SIMD and FP scalar form	LASTA, LASTB, INSR	NA	4	C	-
Extract/insert operation, scalar	LASTA, LASTB, INSR	NA	4	C	1
Horizontal operations, B, H, S, D form, immediate operands only	INDEX	NA	4	C	-

Instruction Group	SVE Instruction	Exec Latency	Execution Throughput	Utilized Pipelines	Notes
Horizontal operations, B, H, S, D form, scalar, immediate operands)/ scalar operands only / immediate, scalar operands	INDEX	NA	2	M, C	-
Logical	AND, BIC, EON, EOR, EORBT, EORTB, MOV, NOT, ORN, ORR	NA	4	C	-
Max/min, basic and pairwise	SMAX, SMAXP, SMIN, SMINP, UMAX, UMAXP UMIN, UMINP	NA	4	C	-
Move prefix	MOVPRFX	NA	4	C	-
Multiply, B, H, S, D element size	MUL, SMULH, UMULH	NA	4	C	-
Multiply long	SMULLB, SMULLT, UMULLB, UMULLT	NA	4	C	-
Multiply accumulate, B, H, S, D element size	MLA, MLS, MAD, MSB	NA	4	C	-
Multiply accumulate long	SMLALB, SMLALT, SMLSLB, SMLSLT, UMLALB, UMLALT, UMLSLB, UMLSLT	NA	4	C	-
Multiply accumulate saturating doubling long regular	SQDMLALB, SQDMLALT, SQDMLALBT, SQDMLSLB, SQDMLSLT, SQDMLSLBT	NA	4	C	-
Multiply saturating doubling high, B, H, S, D element size	SQDMULH	NA	4	C	-
Multiply saturating doubling long	SQDMULLB, SQDMULLT	NA	4	C	-
Multiply saturating rounding doubling regular/complex accumulate, B, H, S, D element size	SQRDMLAH, SQRDMLSH, SQRDCMLAH	NA	4	C	-
Multiply saturating rounding doubling regular/complex, B, H, S, D element size	SQRDMULH	NA	4	C	-
Multiply/multiply long, (8x8) polynomial	PMUL	NA	4	C	-
Predicate counting vector	CNT, DECB, DECH, DECW, DECD, INCB, INCH, INCW, INCD, SQDECB, SQDECH, SQDECW, SQDECD, SQINCB, SQINCH, SQINCW, SQINCD, UQDECB, UQDECH, UQDECW, UQDECD, UQINCB, UQINCH, UQINCW, UQINCD	NA	4	C	-
Reciprocal estimate for B and H	URECPE, URSQRTE	NA	4	C	-
Reduction, arithmetic, B, H, S form	SADDV, UADDV, SMAXV, SMINV, UMAXV, UMINV	NA	4	C	-
Reduction, logical	ANDV, EORV, ORV	NA	4	C	-
Reverse, vector	REV, REVB, REVH, REVW	NA	4	C	-
Select, vector form	MOV, SEL	NA	4	C	-
Table lookup	TBL	NA	4	C	-

Instruction Group	SVE Instruction	Exec Latency	Execution Throughput	Utilized Pipelines	Notes
Table lookup extension	TBX	NA	4	C	-
Transpose, vector form	TRN1, TRN2	NA	4	C	-
Unpack and extend	SUNPKHI, SUNPKLO, UUNPKHI, UUNPKLO	NA	4	C	-
Zip/unzip	UZP1, UZP2, ZIP1, ZIP2	NA	4	C	-

Notes:

1. Those instructions sent to CME when in Streaming SVE mode are writing flags or general purpose or predicate registers and will impact the return bandwidth CME to Core, refer to the CME SWOG for more information.

2.34.6 SVE floating-point instructions when in Streaming SVE mode

Table 2-39: SVE floating-point instructions when in Streaming SVE mode

Instruction Group	SVE Instruction	Exec Latency	Execution Throughput	Utilized Pipelines	Notes
Floating point absolute value/difference	FABD, FABS	NA	4	C	-
Floating point arithmetic	FADD, FNEG, FSUB, FSUBR	NA	4	C	-
Floating point compare	FACGE, FACGT, FACLE, FACLT, FCMEQ, FCMGE, FCMGT, FCMLE, FCMLT, FCMNE, FCMUO	NA	4	C	1
Floating point complex add	FCADD	NA	4	C	-
Floating point complex multiply add	FCMLA	NA	4	C	-
Floating point convert, long or narrow ((F16 to F64, F32 to F64, F64 to F32, F64 to F16, F16 to F32 or F32 to F16)	FCVT, FCVTLT, FCVTNT	NA	4	C	-
Floating point convert, round to odd	FCVTX, FCVTXNT	NA	4	C	-
Floating point base2 log, F16, F32, F64	FLOGB	NA	4	C	-
Floating point convert to integer, F16, F32, F64	FCVTZS, FCVTZU	NA	4	C	-
Floating point copy	FCPY, FDUP, FMOV	NA	4	C	-
Floating point divide, F16, F32, F64	FDIV, FDIVR	NA	4	C	-
Floating point arith, min/max pairwise	FADDP, FMAXP, FMAXNMP, FMINP, FMINNMP	NA	4	C	-
Floating point min/max	FMAX, DMIN, FMAXNM, FMINNM	NA	4	C	-
Floating point multiply	FSCALE, FMUL, FMULX	NA	4	C	-
Floating point multiply accumulate	FMLA, FMLS, FMAD, FMSB, FNMAD, FNMLA, FNMLS, FNMSB	NA	4	C	-
Floating point multiply add/sub accumulate long	FMLALB, FMLALT, FMLSLB, FMLSLT	NA	4	C	-
Floating point reciprocal estimate, F16, F32, F64	FRECPE, FRECPX, FRSQRTE	NA	4	C	-
Floating point reciprocal step	FRECPS, FRSQRTS	NA	4	C	-

Instruction Group	SVE Instruction	Exec Latency	Execution Throughput	Utilized Pipelines	Notes
Floating point reduction, F16, F32, F64	FADDV, FMAXNMV, FMAXV, FMINNMV, FMINV	NA	4	C	-
Floating point round to integral, F16, F32, F64	FRINTA, FRINTM, FRINTN, FRINTP, FRINTX, FRINTZ	NA	4	C	-
Floating point square root, F16, F32, F64	FSQRT	NA	4	C	-

Notes:

1. Those instructions sent to CME when in Streaming SVE mode are writing flags or general purpose or predicate registers and will impact the return bandwidth CME to Core, refer to the CME SWOG for more information.

2.34.7 SVE BFloat16 (BF16) instructions when in Streaming SVE mode

Table 2-40: SVE Bfloat16 (BF16) instructions when in Streaming SVE mode

Instruction Group	SVE Instruction	Exec Latency	Execution Throughput	Utilized Pipelines	Notes
Convert, F32 to BF16	BFCVT, BFCVTNT	NA	4	C	-
Dot product	BFDOT	NA	4	C	-
Multiply accumulate long	BFMLALB, BFMLALT	NA	4	C	-

2.34.8 SVE Load instructions when in Streaming SVE mode

Table 2-41: SVE Load instructions when in Streaming SVE mode

Instruction Group	SVE Instruction	Exec Latency	Execution Throughput	Utilized Pipelines	Notes
Load vector	LDR	NA	3	L, C	-
Load predicate	LDR	5	3	L, C	1
Contiguous load, scalar + imm	LD1B, LD1D, LD1H, LD1W, LD1SB, LD1SH, LD1SW,	NA	3	L, C	-
Contiguous load, scalar + scalar	LD1B, LD1D, LD1H, LD1W, LD1SB, LD1SH LD1SW	NA	3	L, C	-
Non temporal load, scalar + imm	LDNT1B, LDNT1D, LDNT1H, LDNT1W	NA	3	L, C	-
Non temporal load, scalar + scalar	LDNT1B, LDNT1D, LDNT1H, LDNT1W	NA	3	L, C	-
Contiguous Load two structures to two vectors, scalar + imm	LD2B, LD2D, LD2H, LD2W	NA	3	L, C	-
Contiguous Load two structures to two vectors, scalar + scalar	LD2B, LD2D, LD2H, LD2W	NA	3	L, C	-
Contiguous Load three structures to three vectors, scalar + imm	LD3B, LD3H, LD3W, LD3D	NA	3	L, C	-

Instruction Group	SVE Instruction	Exec Latency	Execution Throughput	Utilized Pipelines	Notes
Contiguous Load three structures to three vectors, scalar + scalar	LD3B, LD3W, LD3H, LD3D	NA	3	L, C	-
Contiguous Load four structures to four vectors, scalar + imm	LD4B, LD4H, LD4W, LD4D	NA	3	L, C	-
Contiguous Load four structures to four vectors, scalar + scalar	LD4B, LD4H, LD4W, LD4D	NA	3	L, C	-

Notes:

1. Those instructions sent to CME when in Streaming SVE mode are writing flags or general purpose or predicate registers and will impact the return bandwidth CME to Core, refer to the CME SWOG for more information.

2.34.9 SVE Store instructions when in Streaming SVE mode

Table 2-42: SVE Store instructions

Instruction Group	SVE Instruction	Exec Latency	Execution Throughput	Utilized Pipelines	Notes
Store from predicate reg	STR	1	2	L01, C	-
Store from vector reg	STR	NA	2	L01, C	-
Contiguous store, scalar + imm	ST1B, ST1H, ST1D, ST1W	NA	2	L01, C	-
Contiguous store, scalar + scalar	ST1B, ST1H, ST1D, ST1W	NA	2	L01, C	-
Contiguous store two structures from two vectors, scalar + imm	ST2B, ST2H, ST2D, ST2W	NA	2	L01, C	-
Contiguous store two structures from two vectors, scalar + scalar	ST2B, ST2D, ST2H, ST2W	NA	2	L01, C	-
Contiguous store three structures from three vectors, scalar + imm	ST3B, ST3D, ST3H, ST3W, ST3D	NA	2	L01, C	-
Contiguous store three structures from three vectors, scalar + scalar	ST3B, ST3H, ST3W, ST3D	NA	2	L01, C	-
Contiguous store four structures from four vectors, scalar + imm	ST4B, ST4H, ST4W, ST4D	NA	2	L01, C	-
Contiguous store four structures from four vectors, scalar + scalar	ST4B, ST4H, ST4W, ST4D	NA	2	L01, C	-
Non temporal store, scalar + imm	STNT1B, STNT1D, STNT1H, STNT1W	NA	2	L01, C	-
Non temporal store, scalar + scalar	STNT1B, STNT1D, STNT1H, STNT1W	NA	2	L01, C	-

2.34.10 SVE Cryptographic instructions when in Streaming SVE mode

Table 2-43: SVE cryptographic instructions when in Streaming SVE mode

Instruction Group	SVE Instructions	Exec Latency	Execution Throughput	Utilized Pipelines	Notes
Crypto SHA3 ops	BCAX, EOR3, XAR	NA	4	C	-

2.34.11 SVE2 and base A64 instructions added by SME when in Streaming SVE mode

Table 2-44: SVE2 and base A64 instructions added by SME when in Streaming SVE mode

Instruction Group	SME Instructions	Exec Latency	Execution Throughput	Utilized Pipelines	Notes
BFloat16 floating-point multiply-subtract long from single-precision vector and indexed forms	BFMLSLB, BFMLSLT	NA	4	C	-
Floating-point clamp to minimum/maximum number	FCLAMP	NA	4	C	-
Half-precision floating-point indexed or vector forms dot product	FDOT	NA	4	C	-
Reverse 64-bit doublewords in elements	REVD	NA	4	C	-
Range Prefetch Memory	RPRFM	NA	4	C	-
Signed or unsigned clamp to minimum/maximum vector	SCLAMP, UCLAMP	NA	4	C	-
Signed or unsigned integer indexed or vector forms dot product	SDOT, UDOT	NA	4	C	-
Signed or unsigned saturating (unsigned) extract narrow and interleave	SQCVTN, SQCTUN, UQCVTN	NA	4	C	-
Signed saturating rounding shift right (unsigned) narrow by immediate and interleave	SQRSHRN, SQRSHRUN,	NA	4	C	-
Unsigned saturating rounding shift right narrow by immediate and interleave	UQRSHRN	NA	4	C	-

2.34.12 SME and SME2 processing instructions when in Streaming SVE mode

Table 2-45: SME and SME2 instructions when in Streaming SVE mode

Instruction Group	SME Instructions	Exec Latency	Execution Throughput	Utilized Pipelines	Notes
Add multi-vector to ZA array vector accumulators or to multi-vector with ZA array vector results	ADD, SUB	NA	4	C	-
Add replicated single vector to multi-vector with ZA array vector results or to multi-vector with multi-vector result	ADD	NA	4	C	-
Add vector to array	ADDHA, ADDVA	NA	4	C	-
Multi-vector floating-point convert from 8-bit floating-point to (deinterleaved) BFloat16	BF1CVT, BF1CVTL, BF2CVT, BF2CVTL	NA	4	C	-

Instruction Group	SME Instructions	Exec Latency	Execution Throughput	Utilized Pipelines	Notes
BFloat16 floating-point add/subtract multi-vector to ZA array vector accumulators	BFADD, BFSUB	NA	4	C	-
Multi-vector BFloat16 floating-point clamp to minimum/ maximum number	BFCLAMP	NA	4	C	-
Multi-vector floating-point convert from/to BFloat16, single-precision/packed 8-bit floating point	BFCVT, BFCVTN	NA	4	C	-
Multi-vector BFloat16 floating-point dot-product all forms	BFDOT	NA	4	C	-
Multi-vector BFloat16 floating-point maximum/minimum all forms	BFMAX, BFMAXNM, BFMIN, BFMINNM	NA	4	C	-
Multi-vector BFloat16 floating-point fused multiply-add, multiply-subtract (long) all forms	BFMLA, BFMLAL, BFMLS, BFMLSL	NA	4	C	-
BFloat16 floating-point outer product and accumulate/ subtract	BFMOPA, BFMOPS	NA	4	C	-
Multi-vector BFloat16 floating-point vertical dot-product	BFVDOT	NA	4	C	-
Bitwise exclusive NOR population count outer product and accumulate/subtract	BMOPA, BMOPS	NA	4	C	-
Multi-vector floating-point convert from 8-bit floating-point to (deinterleaved) half-precision	F1CVT, F2CVT, F1CVTL, F2CVTL	NA	4	C	-
Floating-point add/subtract multi-vector to ZA array vector accumulators.	FADD/ FSUB	NA	4	C	-
Multi-vector floating-point absolute maximum/minimum	FAMAX, FAMIN	NA	4	C	-
Multi-vector floating-point clamp to minimum/maximum number	FCLAMP	NA	4	C	-
Multi-vector floating-point convert all forms	FCVT, FCVTL, FCVTN, FCVTZS, FCVTZU	NA	4	C	-
Half-precision floating-point indexed or vector forms dot product	FDOT	NA	4	C	-
Multi-vector floating-point maximum/minimum all forms	FMAX, FMAXNM, FMIN, FMINNM	NA	4	C	-
Multi-vector floating-point fused multiply-add/subtract all forms	FMLA, FMLAL, FMLALL, FMLS, FMLSL	NA	4	C	-
Floating-point sum of outer products and accumulate/ subtract all forms	FMOPA, FMOPS	NA	4	C	-
Multi-vector floating-point round to integral value all forms	FRINTA, FRINTM, FRINTN, FRINTP	NA	4	C	-
Multi-vector floating-point adjust exponent all forms	FSCALE, FSCALE	NA	4	C	-
Multi-vector floating-point vertical dot-product all forms	FVDOT, FVDOTB, FVDOTT	NA	4	C	-
Lookup table read with 2-bit indexes/4-bit indexes all forms	LUTI2, LUTI4	NA	4	C	-
Move into/from array (zeroing) all forms	MOV, MOVA, MOVAZ	NA	4	C	-
Move 8 bytes from general-purpose register to ZT0	MOVT	NA	2	M, C	-
Move 8 bytes from ZT0 to general-purpose register	MOVT	NA	4	C	1
Move vector register to ZT0	MOVT	NA	4	C	-
Multi-vector signed or unsigned clamp to minimum/ maximum vector	SCLAMP, UCLAMP	NA	4	C	-

Instruction Group	SME Instructions	Exec Latency	Execution Throughput	Utilized Pipelines	Notes
Multi-vector signed or unsigned integer convert to floating-point	SCVTF, UCVTF	NA	4	C	-
Multi-vector signed or unsigned integer dot product all forms	SDOT, UDOT	NA	4	C	-
Multi-vector conditionally select elements from two vectors	SEL	NA	4	C	-
Multi-vector signed or unsigned maximum/minimum all forms	SMAX, SMIN, UMAX, UMIN	NA	4	C	-
Multi-vector signed or unsigned integer multiply-add/subtract long(-long)	SMLAL, SMLALL, SMLSL, SMLSLL, UMLAL, UMLALL, UMLSL, UMLSLL	NA	4	C	-
Signed or unsigned integer sum of outer products and accumulate/subtract	SMOPA, SMOPS, UMOPA, UMOPS	NA	4	C	-
Multi-vector signed or unsigned saturating (unsigned) extract narrow and interleave	SQCVT, SQCVTN, SQCTU, SQCTUN, UQCVT, UQCVTN	NA	4	C	-
Multi-vector signed saturating doubling multiply high all forms	SQDMULH, SQDMULH	NA	4	C	-
Multi-vector signed saturating rounding shift right (unsigned) narrow by immediate (and interleaved)	SQRSHR, SQRSHRN, SQRSHRU, SQRSHRUN, UQRSHR, UQRSHRN	NA	4	C	-
Multi-vector signed rounding shift left all forms	SRSHL, URSHL	NA	4	C	-
Multi-vector signed/unsigned by unsigned/signed integer dot-product all forms	SUDOT, USDOT.	NA	4	C	-
Multi-vector signed or unsigned by unsigned integer multiply-add long-long all forms	SUMLALL, USMLALL	NA	4	C	-
Signed by unsigned/Unsigned by signed integer sum of outer products and accumulate/subtract	SUMOPA, SUMOPS, USMOPA, USMOPS	NA	4	C	-
Unpack and sign-extend/zero-extend multi-vector elements	SUNPK, UUNPK	NA	4	C	-
Multi-vector signed by unsigned/unsigned by signed integer vertical dot-product	SUVDOT, UVDOT	NA	4	C	-
Multi-vector signed integer vertical dot-product	SVDOT	NA	4	C	-
Multi-vector unsigned by signed integer vertical dot-product by indexed element	USVDOT	NA	4	C	-
Concatenate elements from four vectors/two vectors	UZP	NA	4	C	-
Zero all forms	ZERO	NA	4	C	-
Interleave elements from four vectors/two vectors	ZIP	NA	4	C	-

Notes:

1. Those instructions sent to CME when in Streaming SVE mode are writing flags or general purpose or predicate registers and will impact the return bandwidth CME to Core, refer to the CME SWOG for more information.

2.34.13 SME Load instructions when in Streaming SVE mode

Table 2-46: SME Load instructions when in Streaming SVE mode

Instruction Group	SVE Instruction	Exec Latency	Execution Throughput	Utilized Pipelines	Notes
Load ZA array vector	LDR	NA	3	L, C	-
Load ZT0 register	LDR	NA	3	L, C	-
Contiguous load all forms	LD1B, LD1H, LD1W, LD1D, LD1Q	NA	3	L, C	-
Contiguous load non temporal all forms	LDNT1B, LDNT1H, LDNT1W, LDNT1D	NA	3	L, C	-

2.34.14 SME Store instructions when in Streaming SVE mode

Table 2-47: SME Store instructions when in Streaming SVE mode

Instruction Group	SVE Instruction	Exec Latency	Execution Throughput	Utilized Pipelines	Notes
Store ZA array vector	STR	NA	2	L01, C	-
Store ZT0 register	STR	NA	2	L01, C	-
Contiguous store all forms	ST1B, ST1H, ST1W, ST1D, ST1Q	NA	2	L01, C	-
Contiguous store non temporal all forms	STNT1B, STNT1H, STNT1W, STNT1D	NA	2	L01, C	-

3. Special considerations

3.1 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 C1-Pro core.

The dispatch stage can process up to 5 MOPs per cycle and dispatch up to 10 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 4 µOPs utilizing the L01 pipelines
- Up to 4 µOPs utilizing the ID 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.

3.2 CME Dispatch constraints

Dispatch of CME MOPs staying in-order are directly sent to the C pipeline. The C pipeline can receive up to 5 MOPs directly from the rename stage.

3.3 Optimizing memory routines

The C1-Pro core implements FEAT_MOPS, a feature that optimizes memory copying and setting operations by proposing microarchitecture-independent instruction sequences. For each invocation of a memcpy, memmove or memset routine, three instructions (a prologue, main, and epilogue) should be used consecutively. The C1-Pro core implements Option A for all instructions of FEAT_MOPS, those are referenced as Memory Copy and Memory Set instructions in the Arm®v9.3-A architecture which exhaustively describes all supported instructions, such as non-temporal versions.

Table 3-1: FEAT_MOPS bandwidth

Operation	FEAT_MOPS Instructions	Operation Bandwidth
Memory copying (memcpy, memmove)	CPY*	32 bytes/cycle
Memory setting (memset) to 0	SET*	64 bytes/cycle
Memory setting to non-zero value	SET*	32 bytes/cycle

The bandwidth achievable with SET* is equal to that with DC ZVA, given the same alignment and data size conditions. Thus SET* should be used in all cases; there is no need to use DC ZVA for optimal memset to zero.

3.4 Load/Store alignment

The Armv8-A architecture allows many types of load and store accesses to be arbitrarily aligned. The C1-Pro 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 64B boundary.
- Store operations that cross a 32B boundary.

There is extra penalty for accesses (load and store) that cross 4kB boundary.

3.5 Store to Load Forwarding

The C1-Pro 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 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 single register, plus LDP W can get their data forwarded from only 1 store.
- LDP instructions other than LDP W can get their data forwarded from independent stores (for a total of 2)

3.6 AES encryption/decryption

The C1-Pro core can issue two AESE/AESMC/AESD/AESIMC instruction every cycle (fully pipelined) with an execution latency of two cycles. Note, 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 since they are fused (see Section [FPASIMD fusion] on Instruction Fusion). This means encryption or decryption for at least four data chunks should be interleaved for maximum performance, reaching then virtually 4 instructions issue rate in this case:

```
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 data0, key_reg
AESMC data0, data0
...
```

3.7 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.

This is defined in the following table.

Table 3-2: Optimized INT forwarding regions

Region	Instruction Types	Notes
1	ASIMD/SVE integer ALU, ASIMD/SVE integer shift, ASIMD/scalar insert and move, ASIMD/SVE integer abs/cmp/max/min, ASIMD/SVE AES, ASIMD/SVE polynomial multiply, ASIMD/SVE integer reduction, SHA3 and PERM instructions in part [ASIMD miscellaneous in structions] see Note 2	1
2	ASIMD/SVE integer mul/mac	2
3	ASIMD/SVE Crypto, SHA1/SHA256	1

Table 3-3: Optimized FP forwarding regions

Region	Instruction Types	Notes
1	FP/ASIMD/SVE floating-point multiply, FP/ASIMD/SVE floating point multiply-accumulate, FP/ASIMD/SVE compare, FP/ASIMD/SVE add/sub and PERM instructions in part [ASIMD miscell aneous instructions] see Note 2	1
2	ASIMD/SVE BFDOT and BFMMLA instructions	

Notes:

1. ASIMD/SVE extract narrow, saturating instructions are excluded from this region and ASIMD/SVE integer reduction are only consumer forward from this region

2. ASIMD/SVE INT multiply accumulate only fast forward to accumulation source

The following instructions are not part of any region:

- FP/ASIMD/SVE convert and rounding instructions that do not write to general purpose registers
- FP div/sqrt
- SVE sdiv, udiv

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

In addition to the regions mentioned in the table above, all instructions in regions INT1 and FP1 can fast forward to FP/ASIMD/SVE stores plus FP/ASIMD vector to integer register transfers, ASIMD converts that write to general purpose registers and PERM instructions in part ASIMD miscellaneous instructions see Note 2.

More special notes about the forwarding region in [special-considerations:vxint-forwarding]:

- Complex shift by immediate/register and shift accumulate instructions cannot be producers (see sections [ASIMD interger instructions] and SVE integer instructions when not in Streaming SVE mode) in region INT1.
- Extract narrow, saturating instructions cannot be producers (see sections ASIMD miscellaneous instructions and SVE integer instructions when not in Streaming SVE mode) in region INT1.
- Absolute difference accumulate and pairwise add and accumulate instructions cannot be producers (see sections [ASIMD interger instructions] and SVE integer instructions when not in Streaming SVE mode) in region INT1.

More special notes about the forwarding region in [special-considerations:vxfp-forwarding]:

- Element sources (the non-vector operand in "by element" multiplies) used by ASIMD/SVE floating-point multiply and multiply-accumulate operations cannot be consumers.
- For floating-point producer-consumer pairs, the precision of the instructions should match (single, double, or half) in region FP1.
- Pair-wise floating-point instructions cannot be producers or consumers in region FP1.

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 for [special-considerations:vxint-forwarding] and [special-considerations:vxfp-forwarding]). For example, the code below interleaves producers and consumers from regions INT1 and INT2. This will result in an additional latency of cycle as seen by MUL.

```
INS   v27[1], v20[1] // Region INT1 producer but not a region INT2 consumer
MUL   v26, v27, v6 // Region INT2
```

These fast-forwarding regions described in [special-considerations:vxint-forwarding] and [special-considerations:vxfp-forwarding] are forming two clusters: cluster FP and cluster INT. Inter-cluster communication requires one cycle penalty. For example, the code below

```
FADD   v20.2s, v28.2s, v20.2s // Region FP1
ADD    v27, v20, v20 // Region INT1 producer but not a region FP1 consumer
```

3.8 Branch instruction alignment

Branch instruction and branch target instruction alignment and density can affect performance.

For best performance, prefer placing taken branches towards the end of an aligned 32-byte instruction memory region and prefer to have branch target pointing toward the beginning of an aligned 32-byte instruction memory region.

The C1-Pro core prediction is optimized to handle aligned 32-byte instruction region containing no branches.

It is preferable to have an aligned 32-byte instruction region containing two branches, than having two 32-byte regions containing one branch each.

To avoid branch prediction limitation, avoid placing a branch as the last instruction of a 4MB aligned instruction region of code.

3.9 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.

3.10 Special register access

The C1-Pro 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 3-4: Special-purpose register read accesses

Register Read	Non-Speculative	In-Order	Flush Side-Effect	Notes
CurrentEL	No	Yes	No	-
DAIF	No	Yes	No	-
DLR_EL0	No	Yes	No	-

Register Read	Non-Speculative	In-Order	Flush Side-Effect	Notes
DSPSR_EL0	No	Yes	No	-
ELR_*	No	Yes	No	-
FPCR	No	Yes	No	-
FPSR	Yes	Yes	No	2
NZCV	No	No	No	1
SP_*	No	No	No	1
SPSel	No	Yes	No	-
SPSR_*	No	Yes	No	-
SVCR	No	No	No	-
FFR	No	Yes	No	-

Notes:

1. The NZCV and SP registers are fully renamed.

2. FPSR reads must wait for all prior instructions that may update the status flags to execute and retire.

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

Table 3-5: Special-purpose register write accesses

Register Write	Non-Speculative	In-Order	Flush Side-Effect	Notes
DAIF	Yes	Yes	No	-
DLR_EL0	Yes	Yes	No	-
DSPSR_EL0	Yes	Yes	No	-
ELR_*	Yes	Yes	No	-
FPCR	Yes	Yes	See Notes	2
FPSR	Yes	Yes	No	3
NZCV	No	No	No	1
SP_*	No	No	No	1
SPSel	Yes	Yes	Yes	-
SPSR_*	Yes	Yes	No	-
SVCR	Yes	Yes	Yes	4
SVCR.SM	No	No	No	5
SVCR.ZA	No	No	No	5
SVCR.SMZA	No	No	No	5
SETFFR	No	No	No	-
WRFFR	Yes	Yes	Yes	6

Notes:

1. The NZCV and SP registers are fully renamed.

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

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

4. The special register write is performed by MSR SVCR (register) and uses the VX pipeline.

5. The special register write is performed by MSR SVCR (immediate) ie. SMSTART/SMSTOP. SMSTART SM or SMSTART.SMZA uses the VX pipeline.

6. A flush side-effect is only generated if the new FFR is different from the previous value.

3.11 Instruction fusion

The C1-Pro core can accelerate certain instruction pairs in an operation called fusion. Specific instruction pairs that can be fused are described in following sections.

3.11.1 Branch and Integer fusion

These instruction pairs must be adjacent to each other in program code. For CMP, CMN, TST fusion is allowed for shifted and/or extended register forms. For CMP, CMN, TST and BICS, there are restrictions on immediate values for both instructions of the pair for which fusion is supported. Other restrictions apply on instruction fusion.

- CMP/CMN (immediate) + B.cond/BC.cond
- CMP/CMN (register Rn != ZR) + B.cond/BC.cond
- TST (immediate) + B.cond/BC.cond
- TST (register) + B.cond/BC.cond
- BICS ZR (register) + B.cond/BC.cond
- CMP (immediate) + CSEL
- CMP (register) + CSEL
- CMP (immediate) + CSET
- CMP (register) + CSET
- BTI + Integer DP instructions with some restrictions which apply on instruction fusion : ADD, ADC, ADCS, ADR, ADRP, AND, ANDS, ASRV, BFM, CCMN, CCMP, CLS, CLZ, CRC, CSEL, CSINC, CSINV, CSNEG, DIV, EOR, EXTR, GMI, LSLV, LSLRV, MOVK, MOVN, MOVZ, MUL, MULL, MULLH, ORR, RBIT, REV, REV16, REV32, RMIF, RORV, SBC, SBCS, SBFM, SETF8, SETF16, SUB, SUBP, UBFM
- BTI + Branch instructions with some restrictions which apply on instruction fusion: BR, BL, BLR, B UNCOND, CBNZ, CBZ, RET, TBNZ, TBZ

3.11.2 FP/ASIMD fusion

These instruction pairs must be adjacent to each other in program code. Other restrictions apply on instruction fusion. Those fusions are possible only when not in Streaming SVE mode.

- AESE + AESMC (see Section AES encryption/decryption)
- AESD + AESIMC (see Section AES encryption/decryption)
- SHL + SRI (both scalar or both vector)
- FCMP + AXFLAG

3.11.3 MOVPRFX fusion

Those fusions are possible regardless of Streaming SVE mode.

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, BF16 or SME instruction.

- MOVPRFX + supported SVE and SME instructions

The list of SVE and SME instructions and the conditions under which this fusion can be applied is mentioned in the tables below.

Table 3-6: MOVPRFX unpredicated fusion

Instruction Group	SVE Instruction	Notes
Integer Instructions		
Arithmetic, absolute difference	SABD, UABD	-
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 and accum long	SADALP, UADALP	-
Arithmetic, shift	ASR, ASRR, LSL, LSLR, LSR, LSRR	For ASR, LSL and LSR, only the wide elements predicated and vector forms are fusible.
Arithmetic, shift and accumulate	SRSRA, SSRA, URSRA, USRA	-
Arithmetic, shift complex	SQRSHL, SQRSHLR, SQSHL, SQSHLR, UQRSHL, UQRSHLR, UQSHL, UQSHLR	SQSHL and UQSHL only vector form is fusible

Instruction Group	SVE Instruction	Notes
Arithmetic, shift rounding	SRSHL, SRSHLR, URSHL, URSHLR	-
Bitwise select	BSL, BSL1N, BSL2N, NBSL	-
Count/reverse bits	CLS, CLZ, CNT, RBIT	-
Complex add	CADD, SQCADD	-
Complex dot product	CDOT	Vector and indexed forms are fusible.
Complex multiply-add	CMLA	Vector and indexed forms are fusible.
Conditional extract operations	CLASTA, CLASTB	Only the vector forms are fusible.
Convert to floating point	SCVTF, UCVTF	-
Copy	CPY	Only the SIMD&FP scalar and immediate merging forms are fusible
Divides	SDIV, SDIVR, UDIV, UDIVR	-
Dot product	SDOT, UDOT, SUDOT, USDOT	For SUDOT only the indexed form is fusible
Extract/insert operation	INSR	Only the SIMD&FP scalar form is fusible
Logical	AND, BIC, EON, EOR, EORBT, EORTB, MOV, ORN, ORR	For AND, BIC, EOR and ORR, only the immediate and vector, predicated forms are fusible For MOV only the SIMD&FP scalar form predicated is fusible For ORN only the immediate form is fusible
Max/min, basic and pairwise	SMAX, SMIN, UMAX, UMIN	Only the immediate and vector, predicated forms are fusible
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, MAD, MSB	Vector and indexed forms are fusible
Multiply accumulate long	SMLALB, SMLALT, SMLSLB, SMLSLT, UMLALB, UMLALT, UMLSLB, UMLSLT	Vector and indexed forms are fusible
Multiply accumulate saturating doubling long regular	SQDMLALB, SQDMLALT, SQDMLALBT, SQDMLSLB, SQDMLSLT, SQDMLSLBT	Vector and indexed forms are fusible
Multiply saturating rounding doubling regular/complex accumulate	SQRDMLAH, SQRDMLSH, SQRDCMLAH	Vector and indexed forms are fusible
Predicate counting, vector form	DECH, DECW, DECD, INCH, INCW, INCD, SQDECH, SQDECW, SQDECD, SQINCH, SQINCW, SQINCD, UQDECH, UQDECW, UQDECD, UQINCH, UQINCW, UQINCD	Only the vector form is fusible
Reciprocal estimate	URECPE, URSQRTE	-
Reverse, vector	REVB, REVH, REVW	-

Instruction Group	SVE Instruction	Notes
Floating point Instructions		
Floating point absolute value/difference	FABD, FABS	-
Floating point arithmetic	FADD, FNEG, FSUB, FSUBR	For FADD, FSUB, FSUBR only the immediate and vector, predicated forms are fusible.
Floating point complex add	FCADD	-
Floating point complex multiply add	FCMLA	Vector and indexed forms are fusible
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 form is fusible
Floating point divide	FDIV, FDIVR	-
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	-
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	Vector and indexed forms are fusible
Matrix multiply accumulate	BFMMLA	-
Multiply accumulate long	BFMLALB, BFMLALT	Vector and indexed forms are fusible
Scalar convert, F32 to BF16	BFCVT	-
Cryptographic Instructions		
Crypto SHA3 ops	BCAX, EOR3, XAR	-

Instruction Group	Instruction added by SME	Notes
BFloat16 floating-point multiply-subtract long from single-precision vector and indexed forms	BFMLSLB, BFMLSLT	-
Floating-point clamp to minimum/maximum number	FCLAMP	-
Half-precision floating-point indexed or vector forms dot product	FDOT	Index form is fusible
Signed or unsigned clamp to minimum/maximum vector	SCLAMP, UCLAMP	-
Signed or unsigned integer indexed or vector forms dot product	SDOT, UDOT	-

Table 3-8: MOVPRFX predicated fusion

Instruction Group	SVE Instruction	Notes
Integer Instructions		
Arithmetic, absolute difference	SABD, UABD	-
Arithmetic, basic	ABS, ADD, CNOT, NEG, SHADD, SHSUB, SHSUBR, SUB, SUBR, UHADD, UHSUB, UHSUBR	For ADD and SUB, only the vector, predicated form is fusible.
Arithmetic, complex	SQABS, SQADD, SQNEG, SQSUB, SQSUBR, SRHADD, SUQADD, UQADD, UQSUB, UQSUBR, URHADD, USQADD	For SQABS, SQSUB, UQADD and UQSUB, only the vector, predicated form is fusible.
Arithmetic, shift	ASR, ASRR, LSL, LSLR, LSR, LSRR	For ASR, LSL and LSR, only the wide elements predicated and vector forms are fusible.
Count/reverse bits	CLS, CLZ, CNT, RBIT	-
Divides	SDIV, SDIVR, UDIV, UDIVR	-
Logical	AND, BIC, EOR, ORR	For AND, BIC, EOR and ORR, only the vector, predicated form is fusible
Max/min, basic and pairwise	SMAX, SMIN, UMAX, UMIN	Only the vector form is fusible
Multiply	MUL, SMULH, UMULH	For MUL, only the vector, predicated form is fusible. For the others, only the predicated form is fusible.
Reverse, vector	REVB, REVH, REVW	-
Floating point Instructions		
Floating point absolute value/ difference	FABD, FABS	-
Floating point arithmetic	FADD, FNEG, FSUB, FSUBR	For FADD, FSUB, FSUBR only the immediate and vector, predicated forms are fusible.
Floating point complex add	FCADD	-
Floating point divide	FDIV, FDIVR	-
Floating point min/ max	FMAX, FMIN, FMAXNM, FMINNM	-
Floating point multiply	FMUL, FMULX	For FMUL, only the vector, predicated form is fusible
Floating point square root	FSQRT	-

3.12 Zero Latency Instructions

3.12.1 Move and similar instructions

A subset of register-to-register move operations, move immediate operations, predicates 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, #{8{0b0},imm[7:0]}
- MOV Xd, XZR
- MOV Wd, #{8{0b0},imm[7:0]}
- 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
- FMOV Sd, Sn
- FMOV Dd, Dn
- MOV Vd, Vn (vector)
- MOV Zd.D, Zn.D
- PTRUE when not in Streaming SVE mode
- PFALSE when not in Streaming SVE mode
- SETFFR

The MOV Wd, Wn, MOV Xd, Xn and FMOV Sd, Sn, FMOV Dd, Dn, MOV Vd, Vn (vector), MOV Zd.D, Zn.D instructions may not be executed with zero latency under certain conditions.

3.12.2 Add immediate instructions

The C1-Pro core can optimize certain Add immediate instructions. Those instructions can be optimized when they are dependent on other Zero Latency instructions and will then operate as well as Zero Latency instruction ie .these instructions do not utilize the out of order scheduling and execution resources of the machine.

The following instructions are eligible to optimization:

- ADD Xd, Xn, #{4{0b0},imm[7:0]}
- ADD Wd, Wn, #{4{0b0},imm[7:0]}

When they are dependent on those instructions :

- MOV Xd, #{8{0b0},imm[7:0]}
- MOV Xd, XZR
- MOV Wd, #{8{0b0},imm[7:0]}
- MOV Wd, WZR
- ADD Xd, Xn, #{4{0b0},imm[7:0]}, effectively optimized to Zero Latency
- ADD Wd, Wn, #{4{0b0},imm[7:0]}, effectively optimized to Zero Latency

Those ADD immediate may not be executed with zero latency under certain other conditions.

3.12.3 Branch instructions

Branch immediate operations are executed with zero latency. These instructions do not utilize the out of order scheduling and execution resources of the machine. These are as follows:

- B, BC

Branch and Link immediate operations do not execute with zero latency. but these instructions do not utilize the branch execution resources of the machine. These are as follows:

- BL

3.13 TLB-access Latencies

A hit in the L1 instruction *Translation Lookaside Buffer* (TLB) provides a single CLK cycle access to the translation and returns the *Physical Address* (PA) to the instruction cache for comparison. It also checks the access permissions to signal an Instruction Abort.

A hit in the L1 data TLB provides a single CLK cycle access to the translation and returns the PA to the data cache for comparison. It also checks the access permissions to signal a Data Abort.

A miss in the L1 data TLB followed by a hit in the L2 TLB has a 4-cycle penalty compared to a hit in the L1 data TLB. This penalty can be increased depending on the arbitration of pending requests.

3.14 Cache-access Latencies

The C1-Pro core pipeline is optimized for low latency and high bandwidth. The following table lists the latencies for the different levels of cache.

Table 3-9: C1-Pro core cache access latencies

Scenario	Cycle count
Level-1 Cache Hit	4 core cycles
Level-2 Cache Hit	9 core cycles if 128kB, 256kB, 512kB L2 size 10 core cycles if 1MB L2 size
Level-3 Cache Hit	19.5 core cycles + 14.5 DSU cycles
Level-1 Cache Hit in another C1-Pro core in t he same cluster	38 core cycles + 22.5 DSU cycles
Level-2 Cache Hit in another C1-Pro core in t he same cluster	32 core cycles if 128kB, 256kB, 512kB L2 size + 22.5 DSU cycles
Level-3 Cache Miss, DMC access	19.5 core cycles + 15.5 DSU cycles + 2 SYS cycles + system latency

The information in the table above is based on the assumptions that:

- Asynchronous bridges are present between core and DSU with 2-stage synchronizers in each clock domain. Latencies that include crossing the asynchronous boundary to the DSU use average latencies through the asynchronous bridge.
- The Level-3 cache data RAM latency configuration is the default 1-cycle in, 2-cycles out.
- DSU frequency is 2GHz, asynchronous to 3GHz CPU frequency. Higher DSU frequency might require extra flops that increase the latency to L3.
- The cluster contains 1-4 cores. Additional cores might require register slices that increase the latency to L3.

Latencies are specified as load-to-use. This measurement represents the number of cycles from when a load instruction is in a given pipeline stage to when a dependent instruction is in the same pipeline stage.

3.15 Cache maintenance operation

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 outer loop.

3.16 Memory Tagging - Tagging Performance

The C1-Pro core implements FEAT_MTE and FEAT_MOPS. For each invocation of a memset routine, three instructions (a prologue, main, and epilogue) should be used consecutively. Those are referenced as Memory Set instructions in the Arm® v9.3-A architecture which exhaustively describes all supported instructions, such as non-temporal versions.

Table 3-10: FEAT_MOPS SETG bandwidth

Operation	FEAT_MOPS Instructions	Operation Bandwidth
Memory setting with tag (memset) to 0	SETG*	64 bytes/cycle
Memory setting with tag to non-zero value	SETG*	32 bytes/cycle

To achieve maximum throughput for setting 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-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
```

3.17 Memory Tagging - Synchronous Mode

In synchronous tag checking mode, each store must complete a tag check before the next store can be executed. Thus, performance of stores in synchronous tag checking mode will be diminished.

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

3.18 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, element size less than 64b.
    - LD4, multiple 4-element structures, quad form, element size less than 64b, , post indexed addressing mode.
    - 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
    - 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.
    - LD3[B/H] contiguous (scalar + scalar addressing).
    - LD4[B/H/W] contiguous (scalar + immediate addressing).
    - LD4[B/H/W] contiguous (scalar + scalar addressing).
    - 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.
    - ST3[B/H/W/D] contiguous (scalar + scalar addressing).
    - ST4[B/H/D/W] contiguous (scalar + scalar addressing).

Proprietary Notice

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

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

The content of this document is informational only. Any solutions presented herein are subject to changing conditions, information, scope, and data. This document was produced using reasonable efforts based on information available as of the date of issue of this document. The scope of information in this document may exceed that which Arm is required to provide, and such additional information is merely intended to further assist the recipient and does not represent Arm's view of the scope of its obligations. You acknowledge and agree that you possess the necessary expertise in system security and functional safety and that you shall be solely responsible for compliance with all legal, regulatory, safety and security related requirements concerning your products, notwithstanding any information or support that may be provided by Arm herein. In addition, you are responsible for any applications which are used in conjunction with any Arm technology described in this document, and to minimize risks, adequate design and operating safeguards should be provided for by you.

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

TO THE EXTENT NOT PROHIBITED BY LAW, IN NO EVENT WILL ARM BE LIABLE FOR ANY DAMAGES, INCLUDING WITHOUT LIMITATION ANY DIRECT, INDIRECT, SPECIAL, INCIDENTAL, PUNITIVE, OR CONSEQUENTIAL DAMAGES, HOWEVER CAUSED AND REGARDLESS OF THE THEORY OF LIABILITY, ARISING OUT OF ANY USE OF THIS DOCUMENT, EVEN IF ARM HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

Reference by Arm to any third party's products or services within this document is not an express or implied approval or endorsement of the use thereof.

This document consists solely of commercial items. You shall be responsible for ensuring that any permitted use, duplication, or disclosure of this document complies fully with any relevant

Arm Limited. Company 02557590 registered in England.

110 Fulbourn Road, Cambridge, England CB1 9NJ.

PRE-1121-V1.0

Product and document information

Read the information in these sections to understand the release status of the product and documentation, and the conventions used in Arm documents.

Product status

All products and services provided by Arm require deliverables to be prepared and made available at different levels of completeness. The information in this document indicates the appropriate level of completeness for the associated deliverables.

Product completeness status

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

Product revision status

The r1p2 identifier indicates the revision status of the product described in this manual, where:

r*x* Identifies the major revision of the product.
p*y* Identifies the minor revision or modification status of the product.

Revision history

These sections can help you understand how the document has changed over time.

Document release information

The Document history table gives the issue number and the released date for each released issue of this document.

Document history

Issue	Date	Confidentiality	Change
0102-05	10 September 2025	Non-Confidential	Second early access release for r1p2
0102-04	30 April 2025	Confidential	First early access release for r1p2
0101-03	30 August 2024	Confidential	First early access release for r1p1
0100-02	31 July 2024	Confidential	First early access release for r1p0
0000-01	28 February 2024	Confidential	First limited access release for r0p0

Change history

The Change history tables describe the technical changes between released issues of this document in reverse order. Issue numbers match the revision history in Document release information on page 73.

Table 2: Issue 0000-01

Change	Location
First Confidential limited access release for r0p0	

Table 3: Issue 0100-02

Change	Location
Remove few unsupported MOVPRFX fusions	MOVPRFX fusion
Fix FCVT and MOV instructions from vec to gen	FP miscellaneous instructions when not in Streaming SVE mode and ASIMD miscellaneous instructions
Describe accumulation latency which is provided in parenthesis	Instruction tables
List instructions for BTI fusion	Branch and Integer fusion
First Confidential early access release for r1p0	

Table 4: Issue 0101-03

Change	Location
Remove Optimizing general-purpose register spills and fills recommendation	
First Confidential early access release for r1p1	

Table 5: Issue 0102-04

Change	Location
Change Format of the document	
Fix some SVE reduction latencies	SVE floating-point instructions when not in Streaming SVE mode and SVE integer instructions when not in Streaming SVE mode
Fix information on Branch instructions	Branch instructions
First Confidential early access release for r1p2	

Table 6: Issue 0102-05

Change	Location
Second Non-Confidential early access release for r1p2	-
Editorial changes	Throughout document

Conventions

The following subsections describe conventions used in Arm documents.

Glossary

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

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

Typographic conventions

Arm documentation uses typographical conventions to convey specific meaning.

Convention	Use
italic	Citations.
bold	Terms in descriptive lists, where appropriate.
monospace	Text that you can enter at the keyboard, such as commands, file and program names, and source code.
monospace <u>underline</u>	A permitted abbreviation for a command or option. You can enter the underlined text instead of the full command or option name.
<and>	Encloses replaceable terms for assembler syntax where they appear in code or code fragments. For example: MRC p15, 0, <Rd>, <CRn>, <CRm>, <Opcode_2>
SMALL CAPITALS	Terms that have specific technical meanings as defined in the *Arm® Glossary*. For example, IMPLEMENTATION DEFINED, IMPLEMENTATION SPECIFIC, UNKNOWN, and UNPREDICTABLE.

Caution

We recommend the following. If you do not follow these recommendations your system might not work.

Warning

Your system requires the following. If you do not follow these requirements your system will not work.

Danger

You are at risk of causing permanent damage to your system or your equipment, or of harming yourself.

Note

This information is important and needs your attention.

Tip

This information might help you perform a task in an easier, better, or faster way.

Remember

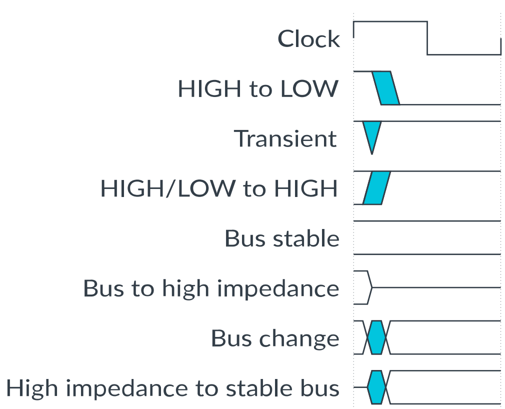
This information reminds you of something important relating to the current content.

Timing diagrams

The following figure explains the components used in timing diagrams. Variations, when they occur, have clear labels. You must not assume any timing information that is not explicit in the diagrams.

Shaded bus and signal areas are undefined, so the bus or signal can assume any value within the shaded area at that time. The actual level is unimportant and does not affect normal operation.

Figure 1: Key to timing diagram conventions



Signals

The signal conventions are:

Signal level

The level of an asserted signal depends on whether the signal is active-HIGH or active-LOW. Asserted means:

- HIGH for active-HIGH signals.
- LOW for active-LOW signals.

Lowercase n

At the start or end of a signal name, n denotes an active-LOW signal.

Useful resources

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

Access to Arm documents depends on their confidentiality:

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

Arm product resources	Document ID	Confidentiality
Arm® Architecture Reference Manual for A-profile architecture	DDI 0487	Non-Confidential
Arm® C1-Pro Core Configuration and Integration Manual	107770	Confidential
Arm® C1-Pro Core Technical Reference Manual	107771	Non-Confidential