



# Arm<sup>®</sup> C1-Nano Core

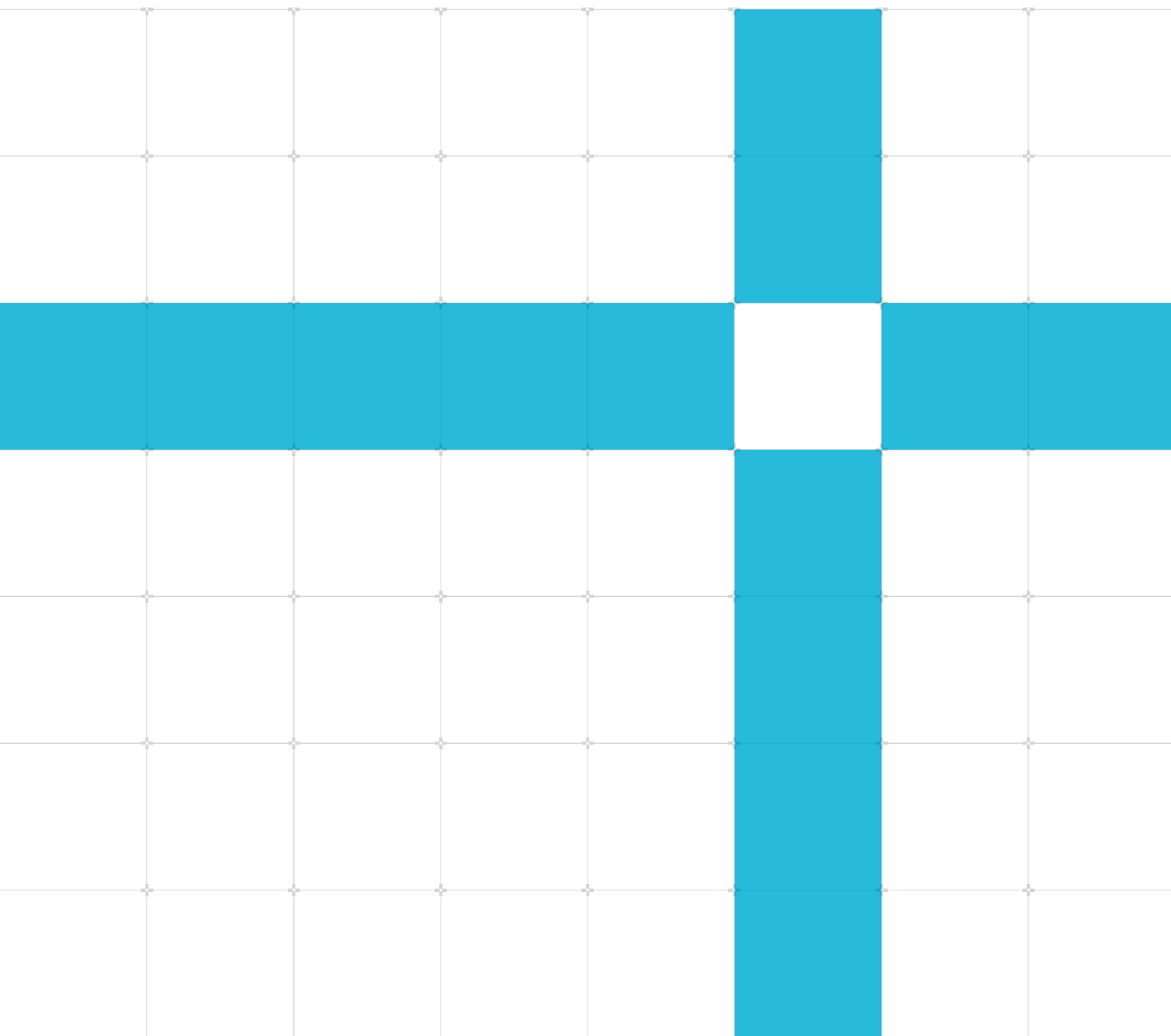
Revision: r0p1

## Software Optimization Guide

Non-Confidential

**Issue 04**

Copyright © 2024-2025 Arm Limited (or its affiliates). 109590\_0001\_04  
All rights reserved.



## Arm® C1-Nano Core Software Optimization Guide

This document is Confidential. This document may only be used and distributed in accordance with the terms of the agreement entered into by Arm and the party that Arm delivered this document to.

Copyright © 2024-2025 Arm Limited (or its affiliates). All rights reserved.

This document is protected by copyright and other intellectual property rights. Arm only permits use of this document if you have reviewed and accepted [Arm's Proprietary notice](#) found at the end of this document.

This document ( 109590\_0001\_04 ) was issued on 16th September 2025. There might be a later issue at <http://developer.arm.com/documentation/>

The product revision is rOp1.

See also: [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.

### 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](mailto: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

<b>1</b>	<b>Product Overview .....</b>	<b>6</b>
1.1	Pipeline overview .....	7
<b>2</b>	<b>Instruction characteristics.....</b>	<b>9</b>
2.1	Instruction tables .....	9
2.2	Branch Instructions.....	9
2.3	Arithmetic and logical instructions.....	10
2.4	Divide and multiply instructions.....	10
2.5	Pointer authentication instructions.....	11
2.6	Miscellaneous data-processing instructions .....	13
2.7	Load instructions.....	14
2.8	Store instructions .....	15
2.9	Tag data processing.....	16
2.10	Tag load instructions .....	17
2.11	Tag store instructions.....	17
2.12	FP scalar data processing instructions .....	18
2.13	FP scalar miscellaneous instructions .....	20
2.14	FP scalar load instructions.....	20
2.15	FP scalar store instructions.....	22
2.16	ASIMD Integer instructions.....	23
2.17	ASIMD FP data processing instructions.....	26
2.18	ASIMD BFloat16 (BF16) instructions .....	29
2.19	ASIMD miscellaneous instructions.....	29
2.20	ASIMD load instructions.....	31
2.21	ASIMD store instructions .....	33
2.22	Cryptography extensions.....	35
2.23	CRC.....	36
2.24	SVE Predicate instructions.....	36
2.25	SVE Integer instructions .....	38

2.26	SVE FP data processing instructions.....	46
2.27	SVE BFloat16 (BF16) instructions.....	49
2.28	SVE Load instructions.....	49
2.29	SVE Store instructions.....	53
2.30	SVE Miscellaneous instructions.....	55
2.31	SVE Cryptography instructions.....	55
2.32	MOPS instructions.....	56
2.33	SME instructions.....	59
2.33.1	Entering and leaving streaming mode.....	59
2.33.2	Predicate and flag related instructions.....	59
2.33.3	Load and store instructions.....	60
2.33.4	Data processing instructions.....	60
2.33.5	System register instructions.....	60
<b>3</b>	<b>Special considerations.....</b>	<b>61</b>
3.1	Issue constraints.....	61
3.2	Instruction fusion.....	62
3.3	Branch instruction alignment.....	62
3.4	Load / Store Alignment.....	62
3.5	A64 low latency pointer forwarding.....	63
3.6	AUT* RET forwarding.....	63
3.7	SIMD MAC forwarding.....	63
3.8	Memory Tagging Extensions.....	64
3.9	Memory routines.....	64
3.10	Cache maintenance operations.....	66
3.11	Cache access latencies.....	66
3.12	Shared VPU.....	67
3.13	AES encryption / decryption.....	67
	<b>Proprietary Notice.....</b>	<b>68</b>
	<b>Product and document information.....</b>	<b>70</b>
	Product status.....	70
	Revision history.....	70
	Conventions.....	72

**Useful resources ..... 75**

# 1 Product Overview

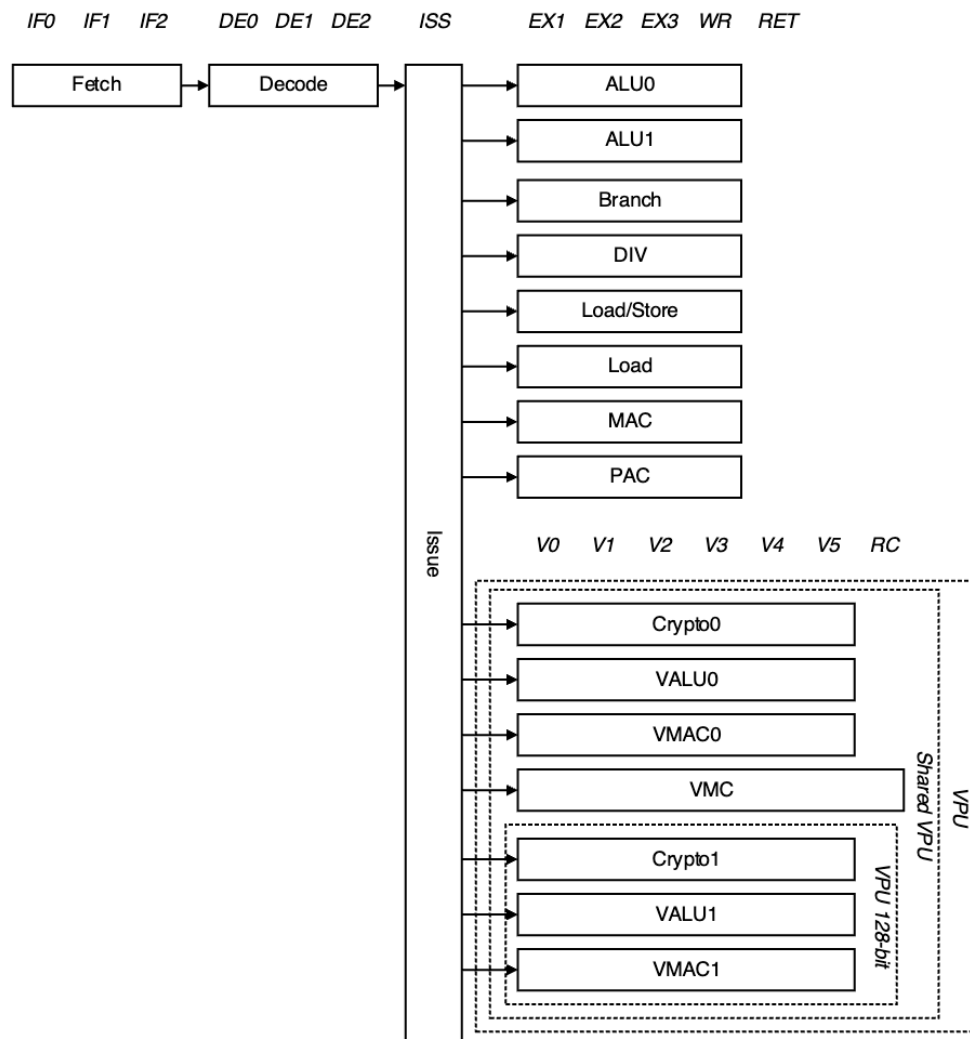
C1-Nano Core is a high-efficiency, low-power product that implements the Arm®v9.3-A architecture. The Arm®v9.3-A architecture extends the architecture defined in the Arm®v8-A architectures up to Arm®v8.9-A. The key features of C1-Nano Core are:

- Implementation of the Arm®v9.3-A A64 instruction set.
- AArch64 Execution state at all Exception levels, EL0 to EL3.
- Separate L1 data and instruction side memory systems with a Memory Management Unit (MMU).
- In-order pipeline with direct and indirect branch prediction.
- Generic Interrupt Controller (GIC) CPU interface to connect to an external interrupt distributor.
- Generic Timer interface that supports a 64-bit count input from an external system counter.
- Implementation of the Reliability, Availability, and Serviceability (RAS) Extension.
- 128-bit Scalable Vector Extension (SVE) and SVE2 SIMD instruction set, offering Advanced SIMD (ASIMD) and floating-point (FP) architecture support.
- Support for the optional Cryptographic Extension, which is licensed separately.
- Activity Monitoring Unit (AMU).
- Dual/Single Core configuration option: C1-Nano cores can be grouped into dual-core complexes or instantiated as single-core complexes. Dual-core complexes share the L2 cache and VPU, while single-core complexes have a dedicated L2 cache and VPU. Figure 1-1 highlights the VPU pipelines shared between C1-Nano cores in a complex.
- Configurable vector datapath size: The size of the vector datapaths can be 2x64 or 2x128-bit. The selected option applies to all cores in the complex. Figure 1-1 highlights the VPU pipelines that are only instantiated for a 2x128-bit configuration.

This document describes the elements of C1-Nano Core micro-architecture that influence the software performance so that software and compilers can be optimized accordingly.

## 1.1 Pipeline overview

Figure 1-1: C1-Nano Core pipeline.



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

Table 1-1: C1-Nano Core Pipeline

Pipeline	Instructions
ALU0, ALU1	Arithmetic and logic
Branch	Branch
Crypto0	Cryptography

Pipeline	Instructions
	<p>Supports 1x128-bit operation.</p> <p>This pipeline is shared for dual core configuration.</p> <p>Present only for implementations configured with Cryptographic Extensions enabled.</p>
Crypto1	<p>Cryptography</p> <p>Supports 1x128-bit operation.</p> <p>This pipeline is shared for dual core configuration.</p> <p>Present only for implementations configured with Cryptographic Extensions enabled and a Vector datapath size of 2x128-bit.</p>
DIV	Integer scalar division (iterative)
Load/Store	Load and store
Load	Load
MAC	Multiply accumulate
PAC	Pointer Authentication
VALU0	<p>Addition, logic and shift for ASIMD, FP, Neon, and SVE</p> <p>Supports 2x64-bit or 1x128-bit operations.</p> <p>This pipeline is shared for dual core configuration.</p>
VALU1	<p>Addition, logic and shift for ASIMD, FP, Neon, and SVE</p> <p>Supports 2x64-bit or 1x128-bit operations.</p> <p>This pipeline is shared for dual core configuration.</p> <p>Present only for implementations configured with a Vector datapath size of 2x128-bit.</p>
VMAC0	<p>Multiply accumulate for ASIMD, FP, Neon, and SVE</p> <p>Supports 2x64-bit or 1x128-bit operations.</p> <p>This pipeline is shared for dual core configurations.</p>
VMAC1	<p>Multiply accumulate for ASIMD, FP, Neon, and SVE</p> <p>Supports 2x64-bit or 1x128-bit operations.</p> <p>This pipeline is shared for dual core configurations.</p> <p>Present only for implementations configured with a Vector datapath size of 2x128-bit configurations.</p>
VMC	<p>Cryptography and iterative multi cycle instruction (e.g. bit permutation, division, and square root)</p> <p>Supports 2x64-bit or 1x128-bit operations.</p> <p>This pipeline is shared for dual core configurations.</p>



## 2 Instruction characteristics

### 2.1 Instruction tables

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

- *Execution Latency* is the minimum latency seen by an operation dependent on an instruction in the described group.
- *Load Latency* is the minimum latency seen by an operation dependent on the load. It is assumed the memory access hits in the L1 Data Cache.
- *Execution Throughput* is maximum throughput (in instructions per cycle) of the specified instruction group that can be achieved in the entirety of C1-Nano Core microarchitecture.

The Vector datapath size may affect the operation of ASIMD, FP, Neon, and SVE instructions. In such cases the *Execution Latency* and *Execution Throughput* will be defined with two value, “A,B”. A is for a 2x128-bit configuration or a non-Q or scalar form of a 2x64-bit configuration. B is for a 2x64-bit configuration.

### 2.2 Branch Instructions

**Table 2-1: AArch64 Branch instructions.**

Instruction Group	AArch64 Instruction	Execution Latency	Execution Throughput	Utilized Pipeline
Branch, immed	B	-	1	Branch
Branch, register	BR, RET	-	1	Branch
Branch and link, immed	BL	1	1	Branch
Branch and link, register	BLR	1	1	Branch
Compare and branch	CBZ, CBNZ, TBZ, TBNZ	-	1	Branch

## 2.3 Arithmetic and logical instructions

**Table 2-2: AArch64 Branch instructions.**

Instruction Group	AArch64 Instruction	Execution Latency	Execution Throughput	Utilized Pipeline
Arithmetic, basic	ADD, ADC, SBC, SUB, NEG	1	2	ALU
Arithmetic, basic, flagset	ADDS, SUBS	1	2	ALU
Arithmetic, basic, carry, flagset	ADCS, SBCS	1	1	ALU
Arithmetic, extend and shift	ADD, ADDS, SUB, SUBS, NEG	1 <sup>[1]</sup>	2	ALU
Compare	CMN, CMP	1	2	ALU
Conditional compare	CCMN, CCMP	1	1	ALU
Conditional select	CSEL, CSINC, CSINV, CSNEG	1	2	ALU
Logical, basic	AND, ANDS, BIC, BICS, EON, EOR, ORN, ORR	1	2	ALU
Logical, shift	AND, ANDS, BIC, BICS, EON, EOR, ORN, ORR	1	2	ALU

## 2.4 Divide and multiply instructions

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.

**Table 2-3: AArch64 Divide and multiply instructions.**

Instruction Group	AArch64 Instruction	Execution Latency	Execution Throughput	Utilized Pipeline
Divide, W-form	SDIV, UDIV	12	1/12	DIV
Divide, X-form	SDIV, UDIV	20	1/20	DIV

<sup>[1]</sup> Latency=2 when the dependency is on Rm.

Instruction Group	AArch64 Instruction	Execution Latency	Execution Throughput	Utilized Pipeline
Multiply accumulate, W-form	MADD, MSUB, MUL	3	1	MAC
Multiply accumulate, X-form	MADD, MSUB, MUL	4	1/2	MAC
Multiply accumulate long	SMADDL, SMSUBL, UMADDL, UMSUBL	2	1	MAC
Multiply high	SMULH, UMULH	6	1/4	MAC

## 2.5 Pointer authentication instructions

**Table 2-4: AArch64 Pointer authentication instructions.**

Instruction Group	AArch64 Instruction	Execution Latency	Execution Throughput	Utilized Pipeline
Authenticate data address	AUTDA, AUTDB, AUTDZA, AUTDZB	4	1	PAC
Authenticate instruction address	AUTIA, AUTIB, AUTIA1716, AUTIB1716, AUTIASP, AUTIBSP, AUTIAZ, AUTIBZ, AUTIZA, AUTIZB	4	1	PAC
Branch and link, register, with pointer authentication	BLRAA, BLRAAZ, BLRAB, BLRABZ	1	1	Branch, PAC
Branch, register, with pointer authentication	BRAA, BRAAZ, BRAB, BRABZ	-	1	Branch, PAC
Branch, return, with pointer authentication	RETA, RETB	-	1	Branch
Compute pointer authentication code for data address	PACDA, PACDB, PACDZA, PACDZB	4	1	PAC
Compute pointer authentication code, using generic key	PACGA	5	1	PAC

Instruction Group	AArch64 Instruction	Execution Latency	Execution Throughput	Utilized Pipeline
Compute pointer authentication code for instruction address	PACIA, PACIB, PACIA1716, PACIB1716, PACIASP, PACIBSP, PACIAZ, PACIBZ, PACIZA, PACIZB	4	1	PAC
Load register, with pointer authentication, offset	LDRAA, LDRAB	2	2	PAC
Load register, with pointer authentication, pre-indexed	LDRAA, LDRAB	2	1	PAC
Strip pointer authentication code	XPACD, XPACI, XPACLRI	4	1	PAC



1. There is a dedicated forwarding path in the accumulate portion of the unit that allows the result of one MAC operation to be used as the accumulate operand of a following MAC operation with no interlock. Thanks to this, a typical sequence of multiply-accumulate instructions can issue one every 2 cycles). Accumulator forwarding is not supported for consumers of 64 bit multiply high operations.
2. Latency and throughput numbers given for SDIV and UDIV are the worst-case values. Early termination is possible, depending upon the data values (for example, degenerate cases such as divide by zero). Integer divides are performed using an iterative algorithm and block any subsequent divide operations until complete. The number of cycles needed to execute these instructions can be calculated using the formula  $[N + \text{bits}/4]$  ( $N=3$  for UDIV,  $N=4$  for SDIV, i.e. signed division takes one more cycle than unsigned division).

## 2.6 Miscellaneous data-processing instructions

**Table 2-5: AArch64 miscellaneous data-processing instructions.**

Instruction Group	AArch64 Instruction	Execution Latency	Execution Throughput	Utilized Pipeline
Address generation	ADR, ADRP	1	2	ALU
Bitfield extract	EXTR	2 <sup>[2]</sup>	2	ALU
Bitfield move, basic	SBFM, SBFIZ, SBFX, SXTH, SXTW, UBFM, UBFIZ, UBFX, UXTH	2 <sup>[3]</sup>	2	ALU
Bitfield move, insert	BFC, BFI, BFM	2	2	ALU
Convert floating-point condition flags	AXFLAG, XAFLAG	-	1/2	ALU
Flag set instructions	SETF8, SETF16	2	1/2	ALU
Flag manipulation instructions, rotate and select	RMIF	1	1	ALU
Flag manipulation instructions, invert carry	CFINV	1	1/2	ALU
Count leading	CLS, CLZ	1	2	ALU
Move	MOV, MOVN, MVN, MOVK, MOVZ	1	2	ALU
Reverse bytes	REV, REV16, REV32	1	2	ALU
Reverse bits	RBIT	1	2	ALU
Variable shift	ASR, ASRV, LSL, LSLV, LSR, LSRV, ROR, RORV	1	2	ALU
Extend, sign or zero	SXTB, UXTB	1	2	ALU

<sup>[2]</sup> Latency=1 for ROR (immediate) alias of EXTR.

<sup>[3]</sup> Latency=1 for LSL (immediate), LSR (immediate) and UXTB aliases of UBFM. Latency=1 for SXTB and ASR (immediate) aliases of SBFM.

## 2.7 Load instructions

The latencies shown in Table 2-6 assume the memory access hits in the Level 1 Data Cache. Base register updates are done in parallel to the operation.

**Table 2-6: AArch64 Load instructions.**

Instruction Group	AArch64 Instruction	Execution Latency	Execution Throughput	Utilized Pipeline
Load register, literal	LDR, LDRSW, PRFM	2	2	Load/Store, Load
Load register, unscaled immediate	LDUR, LDURB, LDURH, LDURSB, LDURSH, LDURSW, PRFUM	2	2	Load/Store, Load
Load register, immediate post-index	LDR, LDRB, LDRH, LDRSB, LDRSH, LDRSW	2	1	Load/Store, Load
Load register, immediate pre-index	LDR, LDRB, LDRH, LDRSB, LDRSH, LDRSW	2	1	Load/Store, Load
Load register, immediate unprivileged	LDTR, LDTRB, LDTRH, LDTRSB, LDTRSH, LDTRSW	2	2	Load/Store, Load
Load register, unsigned immediate	LDR, LDRB, LDRH, LDRSB, LDRSH, LDRSW, PRFM	2	2	Load/Store, Load
Load register, register offset, basic	LDR, LDRB, LDRH, LDRSB, LDRSH, LDRSW, PRFM	2	2	Load/Store, Load
Load register, register offset, scale	LDR, LDRB, LDRSB, LDRSW, PRFM	2	2	Load/Store, Load
Load register, register offset, scale, halfword	LDRH, LDRSH	2	2	Load/Store, Load
Load register, register offset, extend	LDR, LDRB, LDRH, LDRSB, LDRSH, LDRSW, PRFM	2	2	Load/Store, Load
Load register, register offset, extend, scaled	LDR, LDRB, LDRSW, LDRSB, PRFM	2	2	Load/Store, Load

Instruction Group	AArch64 Instruction	Execution Latency	Execution Throughput	Utilized Pipeline
Load register, register offset, extend, scaled, halfword	LDRH, LDRSH	2	2	Load/Store, Load
Load pair, signed immediate offset, normal, W-form	LDP, LDNP	2	2	Load/Store, Load
Load pair, signed immediate offset, normal, X-form	LDP, LDNP	2	2	Load/Store, Load
Load pair, signed immediate offset, signed words	LDPSW	2	2	Load/Store, Load
Load pair, immediate post-index or immediate pre-index, normal, W-form	LDP	2	1	Load/Store, Load
Load pair, immediate post-index or immediate pre-index, normal, X-form	LDP	2	1	Load/Store, Load
Load pair, immediate post-index or immediate pre-index, signed words	LDPSW	2	1	Load/Store, Load

## 2.8 Store instructions

Base register updates are done in parallel to the operation.

**Table 2-7: AArch64 Store instructions.**

Instruction Group	AArch64 Instruction	Execution Latency	Execution Throughput	Utilized Pipeline
Store register, unscaled immediate	STUR, STURB, STURH	-	1	Load/Store
Store register, immediate post-index	STR, STRB, STRH	-	1	Load/Store
Store register, immediate pre-index	STR, STRB, STRH	-	1	Load/Store
Store register, immediate unprivileged	STTR, STTRB, STTRH	-	1	Load/Store
Store register, unsigned immediate	STR, STRB, STRH	-	1	Load/Store

Instruction Group	AArch64 Instruction	Execution Latency	Execution Throughput	Utilized Pipeline
Store register, register offset, basic	STR, STRB, STRH	-	1	Load/Store
Store register, register offset, scaled	STR, STRB	-	1	Load/Store
Store register, register offset, scaled, halfword	STRH	-	1	Load/Store
Store register, register offset, extend	STR, STRB, STRH	-	1	Load/Store
Store register, register offset, extend, scaled	STR, STRB	-	1	Load/Store
Store register, register offset, extend, scaled, halfword	STRH	-	1	Load/Store
Store pair, immediate offset	STP, STNP	-	1	Load/Store
Store pair, immediate post-index	STP	-	1	Load/Store
Store pair, immediate pre-index	STP	-	1	Load/Store

## 2.9 Tag data processing

**Table 2-8: AArch64 Tag data processing instructions.**

Instruction Group	AArch64 Instruction	Execution Latency	Execution Throughput	Utilized Pipeline
Arithmetic, immediate to logical address tag	ADDG, SUBG	2	2	ALU
Insert Random Tags	IRG	4	1/3	ALU
Insert Tag Mask	GMI	2	2	ALU
Subtract Pointer	SUBP	2	2	ALU
Subtract Pointer, flagset	SUBPS	2	2	ALU



## 2.10 Tag load instructions

The latencies shown assume the memory access hits in the Level 1 Data Cache.

**Table 2-9: The latencies shown assume the memory access hits in the Level 1 Data Cache.**

Instruction Group	AArch64 Instruction	Execution Latency	Execution Throughput	Utilized Pipeline
Load allocation tag	LDG	2	2	Load/Store, Load
Load multiple allocation tags	LDGM	2	1/4	Load/Store, Load

## 2.11 Tag store instructions

Base register updates are done in parallel to the operation.

**Table 2-10: AArch64 Tag store instructions.**

Instruction Group	AArch64 Instruction	Execution Latency	Execution Throughput	Utilized Pipeline
Store allocation tags to one granule, post-index	STG	-	1	Load/Store
Store allocation tags to two granules, post-index	ST2G	-	1/2	Load/Store
Store allocation tags to one granule, pre-index	STG	-	1	Load/Store
Store allocation tags to two granules, pre-index	ST2G	-	1/2	Load/Store
Store allocation tags to one granule, signed offset	STG	-	1	Load/Store
Store allocation tags to two granules, signed offset	ST2G	-	1/2	Load/Store
Store allocation tag to one granule, zeroing, post-index	STZG	-	1	Load/Store
Store allocation tag to two granules, zeroing, post-index	STZ2G	-	1/2	Load/Store
Store Allocation Tag to one granule, zeroing, pre-index	STZG	-	1	Load/Store

Instruction Group	AArch64 Instruction	Execution Latency	Execution Throughput	Utilized Pipeline
Store Allocation Tag to two granules, zeroing, pre-index	STZ2G	-	1/2	Load/Store
Store allocation tag to one granule, zeroing, signed offset	STZG	-	1	Load/Store
Store allocation tag to two granules, zeroing, signed offset	STZ2G	-	1/2	Load/Store
Store allocation tag and reg pair to memory, post-Index	STGP	-	1	Load/Store
Store allocation tag and reg pair to memory, pre-Index	STGP	-	1	Load/Store
Store allocation tag and reg pair to memory, signed offset	STGP	-	1	Load/Store
Store multiple allocation tags	STGM	-	1	Load/Store
Store multiple allocation tags, zeroing	STZGM	-	1	Load/Store

## 2.12 FP scalar data processing instructions

**Table 2-11: AArch64 FP data processing instructions.**

Instruction Group	AArch64 Instruction	Execution Latency	Execution Throughput	Utilized Pipeline
FP absolute value	FABS, FABD	4	2	VALU
FP arithmetic	FADD, FSUB, FADDP	4	2	VALU
FP conditional compare	FCCMP, FCCMPE	5	1/5	VALU
FP compare	FCMP, FCMPE	1	1	VALU
FP divide, H-form	FDIV	8	2/5	VMC
FP divide, S-form	FDIV	13	2/10	VMC
FP divide, D-form	FDIV	22	2/19	VMC
FP min/max	FMIN, FMINNM, FMAX, FMAXNM	4	2	VALU

Instruction Group	AArch64 Instruction	Execution Latency	Execution Throughput	Utilized Pipeline
FP max/min, pairwise	FMAXP, FMAXNMP, FMINP, FMINNMP	4	2	VALU
FP multiply	FMUL, FNMUL, FMULX	4	2	VMAC
FP multiply accumulate	FMADD, FMSUB, FNMADD, FNMSUB	4	2	VMAC
FP negate	FNEG	4	2	VALU
FP round to integral	FRINTA, FRINTI, FRINTM, FRINTN, FRINTP, FRINTX, FRINTZ, FRINT32X, FRINT64X, FRINT32Z, FRINT64Z	4	2	VALU
FP select	FCSEL	3	1	VALU
FP square root, H-form	FSQRT	11	2/5	VMC
FP square root, S-form	FSQRT	14	2/9	VMC
FP square root, D-form	FSQRT	25	2/19	VMC



Floating-point division operations may finish early if the divisor is a power of two (normal with a zero trailing significand).

## 2.13 FP scalar miscellaneous instructions

**Table 2-12: AArch64 FP miscellaneous instructions.**

Instruction Group	AArch64 Instruction	Execution Latency	Execution Throughput	Utilized Pipeline
FP convert, from gen to vec reg	SCVTF, UCVTF	4	2	VALU
FP convert, from vec to gen reg	FCVTAS, FCVTAU, FCVTMS, FCVTMU, FCVTNS, FCVTNU, FCVTPS, FCVTPU, FCVTZS, FCVTZU	4	2	VALU
FP convert, Javascript from vec to gen reg	FJCVTZS	4	1	VALU
FP convert, from vec to vec reg	FCVT, FCVTXN	4	2	VALU
FP move, immediate	FMOV	3	2	VALU
FP move, register	FMOV	2	2	VALU
FP transfer, from gen to vec reg	FMOV	-	2	VALU
FP transfer, from vec to gen reg	FMOV	-	2	VALU

## 2.14 FP scalar load instructions

The latencies shown assume the memory access hits in the Level 1 Data Cache. Base register updates are done in parallel to the operation.

**Table 2-13: AArch64 FP load instructions.**

Instruction Group	AArch64 Instruction	Execution Latency	Execution Throughput	Utilized Pipeline
Load vector reg, literal	LDR	3	2	Load/Store, Load
Load vector reg, unscaled immediate	LDUR	3	2	Load/Store, Load

Instruction Group	AArch64 Instruction	Execution Latency	Execution Throughput	Utilized Pipeline
Load vector reg, immediate post-index	LDR	3	1	Load/Store, Load
Load vector reg, immediate pre-index	LDR	3	1	Load/Store, Load
Load vector reg, unsigned immediate	LDR	3	2	Load/Store, Load
Load vector reg, register offset, basic	LDR	3	2	Load/Store, Load
Load vector reg, register offset, scale	LDR	3	2	Load/Store, Load
Load vector reg, register offset, extend	LDR	3	2	Load/Store, Load
Load vector reg, register offset, extend, scale	LDR	3	2	Load/Store, Load
Load vector pair, immediate offset, S/D-form	LDP, LDNP	3	1	Load/Store, Load
Load vector pair, immediate offset, Q-form	LDP, LDNP	3	1	Load/Store, Load
Load vector pair, immediate post-index, S/D-form	LDP	3	1	Load/Store, Load
Load vector pair, immediate post-index, Q-form	LDP	3	1	Load/Store, Load
Load vector pair, immediate pre-index, S/D-form	LDP	3	1	Load/Store, Load
Load vector pair, immediate pre-index, Q-form	LDP	3	1	Load/Store, Load

## 2.15 FP scalar store instructions

Base register updates are done in parallel to the operation.

**Table 2-14: AArch64 FP Store instructions.**

Instruction Group	AArch64 Instruction	Execution Latency	Execution Throughput	Utilized Pipeline
Store vector reg, unscaled immediate	STUR	-	1	Load/Store
Store vector reg, immediate post-index	STR	-	1	Load/Store
Store vector reg, immediate pre-index	STR	-	1	Load/Store
Store vector reg, unsigned immediate	STR	-	1	Load/Store
Store vector reg, register offset, basic	STR	-	1	Load/Store
Store vector reg, register offset, scale	STR	-	1	Load/Store
Store vector reg, register offset, extend	STR	-	1	Load/Store
Store vector reg, register offset, extend	STR	-	1	Load/Store
Store vector pair, immediate offset, S-form	STP, STNP	-	1	Load/Store
Store vector pair, immediate offset, D-form	STP, STNP	-	1	Load/Store
Store vector pair, immediate offset, Q-form	STP, STNP	-	1/2	Load/Store
Store vector pair, immediate post-index, S-form	STP	-	1	Load/Store
Store vector pair, immediate post-index, D-form	STP	-	1	Load/Store
Store vector pair, immediate post-index, Q-form	STP	-	1/2	Load/Store
Store vector pair, immediate pre-index, S-form	STP	-	1	Load/Store

Instruction Group	AArch64 Instruction	Execution Latency	Execution Throughput	Utilized Pipeline
Store vector pair, immediate pre-index, D-form	STP	-	1	Load/Store
Store vector pair, immediate pre-index, Q-form	STP	-	1/2	Load/Store

## 2.16 ASIMD Integer instructions

**Table 2-15: AArch64 ASIMD Integer instructions.**

Instruction Group	AArch64 Instruction	Execution Latency	Execution Throughput	Utilized Pipeline
ASIMD absolute diff	SABD, UABD	3	2	VALU
ASIMD absolute diff accum	SABA, UABA	5	1/3	VALU
ASIMD absolute diff accum long	SABAL2, UABAL2	5	1/3	VALU
ASIMD absolute diff long	SABDL2, UABDL2	3	2	VALU
ASIMD arith, basic	ABS, ADD, NEG, SHADD, SHSUB, SUB, UHADD, UHSUB	3	2	VALU
ASIMD arith, basic, long, saturate	SADDL, SADDL2, SADDW, SADDW2, SSUBL, SSUBL2, SSUBW, SSUBW2, UADDL, UADDL2, UADDW, UADDW2, USUBL, USUBL2, USUBW, USUBW2	3	2	VALU
ASIMD arith, complex	ADDHN, ADDHN2, SQABS, SQADD, SQNEG, SQSUB, SUBHN, SUBHN2, SUQADD, UQADD, UQSUB, USQADD	4	2	VALU

Instruction Group	AArch64 Instruction	Execution Latency	Execution Throughput	Utilized Pipeline
ASIMD arith, complex, rounding, add and subtract	RADDHN, RADDHN2, RSUBHN, RSUBHN2	6	1/3	VALU
ASIMD arith, complex, rounding halving addition	SRHADD, URHADD	2	2	VALU
ASIMD arith, pair-wise	ADDP, SADDLP, UADDLP	3	2	VALU
ASIMD arith, reduce, 4H/4S	ADDV, SADDLV, UADDLV	4	1	VALU
ASIMD arith, reduce	ADDV	3	1	VALU
ASIMD arith, reduce Long	SADDLV, UADDLV	4	1	VALU
ASIMD compare	CMEQ, CMGE, CMGT, CMHI, CMHS, CMLE, CMLT	3	2	VALU
ASIMD compare test	CMTST	3	2	VALU
ASIMD dot product	SDOT, UDOT	4	2	VMAC
ASIMD dot product using signed and unsigned integers	SUDOT, USDOT	4	2	VMAC
ASIMD logical	AND, BIC, EOR, MOV, MVN, NOT, ORN, ORR	3	2	VALU
ASIMD matrix multiply-accumulate	SMMLA, UMMLA, USMMLA	4	2	VALU
ASIMD max/min, basic and pair-wise	SMAX, SMAXP, SMIN, SMINP, UMAX, UMAXP, UMIN, UMINP	3	2	VALU
ASIMD max/min, reduce, B-form	SMAXV, SMINV, UMAXV, UMINV	4	1	VALU
ASIMD max/min, reduce, H-form	SMAXV, SMINV, UMAXV, UMINV	4	1	VALU
ASIMD max/min, reduce, S-form	SMAXV, SMINV, UMAXV, UMINV	4	1	VALU
ASIMD multiply	MUL, SQDMULH, SQRDMULH	4	2	VMAC



Instruction Group	AArch64 Instruction	Execution Latency	Execution Throughput	Utilized Pipeline
ASIMD multiply accumulate	MLA, MLS	4	2	VMAC
ASIMD multiply accumulate high	SQRDMLAH, SQRDMLSH	4	2	VMAC
ASIMD multiply accumulate long	SMLAL2, SMLSL2, UMLAL2, UMLSL2	4	2	VMAC
ASIMD multiply accumulate saturating long	SQDMLAL2, SQDMLSL2	4	2	VMAC
ASIMD multiply/multiply long (8x8) polynomial, D-form	PMUL, PMULL2	3	2	VALU
ASIMD multiply/multiply long (8x8) polynomial, Q-form	PMUL, PMULL2	3	2	VALU
ASIMD multiply long	SMULL, SMULL2, UMULL, UMULL2, SQDMULL, SQDMULL2	4	2	VMAC
ASIMD pairwise add and accumulate long	SADALP, UADALP	5	1/3	VALU
ASIMD shift and accumulate	SRSRA, URSRA	5	1/3	VALU
ASIMD rounding shift and accumulate	SSRA, USRA	3	2	VALU
ASIMD shift by immediate, basic	SHL, SHLL2, SSHLL2, SSHR, SXTL2, USHLL2, USHR, UXTL2	3	2	VALU
ASIMD shift by immediate, narrow	SHRN2	4	2	VALU
ASIMD shift by immediate and insert, basic	SLI, SRI	3	2	VALU
ASIMD shift by immediate, complex	RSHRN2, SQRSHRN2, SQRSHRUN2, SQSHL, SQSHLU, SQSHRN2, SQSHRUN2, UQRSHRN2, UQSHL, UQSHRN2	4	2	VALU

Instruction Group	AArch64 Instruction	Execution Latency	Execution Throughput	Utilized Pipeline
ASIMD shift by register, basic	SSHL, USHL, SRSHL, SRSHR, URSHL, URSRHR	3	2	VALU
ASIMD shift by register, complex	SQRSHL, SQSHL, UQRSHL, UQSHL	4	2	VALU

## 2.17 ASIMD FP data processing instructions

**Table 2-16: AArch64 ASIMD Floating-point instructions.**

Instruction Group	AArch64 Instruction	Execution Latency	Execution Throughput	Utilized Pipeline
ASIMD FP absolute value/difference	FABS, FABD	4	2	VALU
ASIMD FP arith, normal	FADD, FSUB, FADDP	4	2	VALU
ASIMD FP compare	FACGE, FACGT, FCMEQ, FCMGE, FCMGT, FCMLE, FCMLT	3	2	VALU
ASIMD FP complex add	FCADD	4	2	VMAC
ASIMD FP complex multiply add	FCMLA	4	2	VMAC
ASIMD FP convert, long (F16 to F32)	FCVTL, FCVTL2	4	2	VALU
ASIMD FP convert, long (F32 to F64)	FCVTL, FCVTL2	4	2	VALU
ASIMD FP convert, narrow (F32 to F16)	FCVTN, FCVTN2	4	2	VALU
ASIMD FP convert, narrow (F64 to F32)	FCVTN, FCVTN2, FCVTXN2	4	2	VALU
ASIMD FP convert, from gen to vec reg	SCVTF, UCVTF	4	2	VALU

Instruction Group	AArch64 Instruction	Execution Latency	Execution Throughput	Utilized Pipeline
ASIMD FP convert, other, F16	FCVTAS, FCVTAU, FCVTMS, FCVTMU, FCVTNS, FCVTNU, FCVTPS, FCVTPU, FCVTZS, FCVTZU	4	2	VALU
ASIMD FP convert, other, F32	FCVTAS, FCVTAU, FCVTMS, FCVTMU, FCVTNS, FCVTNU, FCVTPS, FCVTPU, FCVTZS, FCVTZU	4	2	VALU
ASIMD FP convert, other, F64	FCVTAS, FCVTAU, FCVTMS, FCVTMU, FCVTNS, FCVTNU, FCVTPS, FCVTPU, FCVTZS, FCVTZU	4	2	VALU
ASIMD FP divide, D-form, F16	FDIV	8	2/5	VMC
ASIMD FP divide, D-form, F32	FDIV	13	1/5	VMC
ASIMD FP divide, Q-form, F16	FDIV	8	1/5	VMC
ASIMD FP divide, Q-form, F32	FDIV	13	1/10	VMC
ASIMD FP divide, Q-form, F64	FDIV	22	1/19	VALU
ASIMD FP max/min, normal	FMAX, FMAXNM, FMIN, FMINNM	4	2	VALU
ASIMD FP max/min, pairwise	FMAXP, FMAXNMP, FMINP, FMINNMP	4	2	VALU
ASIMD FP max/min, reduce	FMAXV, FMAXNMV, FMINV, FMINNMV	4	1	VALU
ASIMD FP multiply	FMUL, FMULX	4	2	VMAC

Instruction Group	AArch64 Instruction	Execution Latency	Execution Throughput	Utilized Pipeline
ASIMD FP multiply accumulate	FMLA, FMLS	4	2	VMAC
ASIMD FP multiply accumulate long	FMLAL, FMLAL2, FMLSL, FMLSL2	4	2	VMAC
ASIMD FP negate	FNEG	4	2	VALU
ASIMD FP round, F16	FRINTA, FRINTI, FRINTM, FRINTN, FRINTP, FRINTX, FRINTZ, FRINT32X, FRINT64X, FRINT32Z, FRINT64Z	4	2	VALU
ASIMD FP round, F32	FRINTA, FRINTI, FRINTM, FRINTN, FRINTP, FRINTX, FRINTZ, FRINT32X, FRINT64X, FRINT32Z, FRINT64Z	4	2	VALU
ASIMD FP round, F64	FRINTA, FRINTI, FRINTM, FRINTN, FRINTP, FRINTX, FRINTZ, FRINT32X, FRINT64X, FRINT32Z, FRINT64Z	4	2	VALU
ASIMD FP square root, D-form, F16	FSQRT	8	2/5	VMC
ASIMD FP square root, D-form, F32	FSQRT	12	2/9	VMC
ASIMD FP square root, Q-form, F16	FSQRT	8	1/5	VMC
ASIMD FP square root, Q-form, F32	FSQRT	12	1/9	VMC
ASIMD FP square root, Q-form, F64	FSQRT	22	1/19	VMC



Note

Floating-point division operations may finish early if the divisor is a power of two.

## 2.18 ASIMD BFloat16 (BF16) instructions

**Table 2-17: AArch64 ASIMD BFloat16 (BF16) instructions.**

Instruction Group	AArch64 Instruction	Execution Latency	Execution Throughput	Utilized Pipeline
ASIMD convert, F32 to BF16	BFCVTN, BFCVTN2	4	2	VALU
ASIMD dot product	BFDOT	10	2	VMAC,VALU
ASIMD matrix multiply accumulate	BFMMLA	14	1	VMAC,VALU
ASIMD multiply accumulate long	BFMLALB, BFMLALT	4	2	VMAC
Scalar convert, F32 to BF16	BFCVT	4	2	VALU

## 2.19 ASIMD miscellaneous instructions

**Table 2-18: AArch64 ASIMD miscellaneous instructions.**

Instruction Group	AArch64 Instruction	Execution Latency	Execution Throughput	Utilized Pipeline
ASIMD bit reverse	RBIT	3	2	VALU
ASIMD bitwise insert	BIF, BIT, BSL	3	2	VALU
ASIMD count	CLS, CLZ, CNT	3	2	VALU
ASIMD duplicate, gen reg	DUP	3	2	VALU
ASIMD duplicate, element	DUP	3	2	VALU
ASIMD extract	EXT	3	2	VALU
ASIMD extract narrow	XTN	4	2	VALU

Instruction Group	AArch64 Instruction	Execution Latency	Execution Throughput	Utilized Pipeline
ASIMD extract narrow, saturating	SQXTN, SQXTN2, SQXTUN, SQXTUN2, UQXTN, UQXTN2	4	2	VALU
ASIMD insert, element to element	INS	3	2	VALU
ASIMD move, FP immediate	MOV	3	2	VALU
ASIMD FP convert, from vec to vec reg	FCVT, FCVTXN	4	2	VALU
ASIMD move, FP immediate	FMOV	3	2	VALU
ASIMD move, FP register	FMOV	3	2	VALU
ASIMD move, FP transfer, from gen to vec reg	FMOV	3	2	VALU
ASIMD move, integer immediate	MOVI, MVNI	3	2	VALU
ASIMD reciprocal estimate, F16	FRECPE, FRECPX, FRSQRTE, URECPE, URSQRTE	4	2	VMAC
ASIMD reciprocal estimate, F32	FRECPE, FRECPX, FRSQRTE, URECPE, URSQRTE	4	2	VMAC
ASIMD reciprocal estimate, F64	FRECPE, FRECPX, FRSQRTE, URECPE, URSQRTE	4	2	VMAC
ASIMD reciprocal step	FRECPS, FRSQRTS	4	2	VMAC
ASIMD reverse	REV16, REV32, REV64	3	2	VALU
ASIMD table lookup, 1 table regs	TBL	4	2	VALU
ASIMD table lookup, 2 table regs	TBL	5	1/2	VALU
ASIMD table lookup, 3 table regs	TBL	6	1/3	VALU
ASIMD table lookup, 4 table regs	TBL	7	1/4	VALU
ASIMD table lookup extension, 1 table reg	TBX	5	1/2	VALU

Instruction Group	AArch64 Instruction	Execution Latency	Execution Throughput	Utilized Pipeline
ASIMD table lookup extension, 2 table regs	TBX	6	1/3	VALU
ASIMD table lookup extension, 3 table regs	TBX	7	1/4	VALU
ASIMD table lookup extension, 4 table regs	TBX	8	1/5	VALU
ASIMD transfer, element to gen reg	UMOV, SMOV	3	2	VALU
ASIMD transfer, gen reg to element	INS	3	2	VALU
ASIMD transpose	TRN1, TRN2	3	2	VALU
ASIMD unzip/zip	UZIP1, UZIP2, ZIP1, ZIP2	3	2	VALU

## 2.20 ASIMD load instructions

The latencies shown assume the memory access hits in the Level 1 Data Cache. Base register updates are done in parallel to the operation.

**Table 2-19: AArch64 ASIMD load instructions.**

Instruction Group	AArch64 Instruction	Execution Latency	Execution Throughput	Utilized Pipeline
ASIMD load, 1 element, multiple, 1 reg, D-form	LD1	3	2	Load/Store, Load
ASIMD load, 1 element, multiple, 1 reg, Q-form	LD1	3	2	Load/Store, Load
ASIMD load, 1 element, multiple, 2 reg, D-form	LD1	3	1	Load/Store
ASIMD load, 1 element, multiple, 2 reg, Q-form	LD1	3	1	Load/Store
ASIMD load, 1 element, multiple, 3 reg, D-form	LD1	4	1/2	Load/Store
ASIMD load, 1 element, multiple, 3 reg, Q-form	LD1	4	1/2	Load/Store

Instruction Group	AArch64 Instruction	Execution Latency	Execution Throughput	Utilized Pipeline
ASIMD load, 1 element, multiple, 4 reg, D-form	LD1	4	1/2	Load/Store
ASIMD load, 1 element, multiple, 4 reg, Q-form	LD1	4	1/2	Load/Store
ASIMD load, 1 element, one lane, B/H/S	LD1	3	1	Load/Store, Load
ASIMD load, 1 element, one lane, D	LD1	3	1	Load/Store, Load
ASIMD load, 1 element, all lanes, D-form	LD1R	3	2	Load/Store, Load
ASIMD load, 1 element, all lanes, Q-form	LD1R	3	2	Load/Store, Load
ASIMD load, 2 element, multiple, D-form	LD2	4	1	Load/Store
ASIMD load, 2 element, multiple, Q-form	LD2	4	1	Load/Store
ASIMD load, 2 element, one lane, B/H/S	LD2	4	1/4	Load/Store
ASIMD load, 2 element, one lane, D	LD2	4	1/4	Load/Store
ASIMD load, 2 element, all lanes, D-form	LD2R	3	1	Load/Store
ASIMD load, 2 element, all lanes, Q-form	LD2R	3	1	Load/Store
ASIMD load, 3 element, multiple, D-form	LD3	5	1/3	Load/Store
ASIMD load, 3 element, multiple, Q-form	LD3	5	1/3	Load/Store
ASIMD load, 3 element, one lane, B/H/S	LD3	5	1/5	Load/Store
ASIMD load, 3 element, one lane, D	LD3	5	1/5	Load/Store
ASIMD load, 3 element, all lanes, D-form	LD3R	4	1/2	Load/Store
ASIMD load, 3 element, all lanes, Q-form	LD3R	4	1/2	Load/Store



Instruction Group	AArch64 Instruction	Execution Latency	Execution Throughput	Utilized Pipeline
ASIMD load, 4 element, multiple, D-form	LD4	5	1/3	Load/Store
ASIMD load, 4 element, multiple, Q-form	LD4	5	1/3	Load/Store
ASIMD load, 4 element, one lane, B/H/S	LD4	6	1/5	Load/Store
ASIMD load, 4 element, one lane, D	LD4	6	1/5	Load/Store
ASIMD load, 4 element, all lanes, D-form	LD4R	4	1/2	Load/Store
ASIMD load, 4 element, all lanes, Q-form	LD4R	4	1/2	Load/Store

## 2.21 ASIMD store instructions

Base register updates are done in parallel to the operation.

**Table 2-20: AArch64 ASIMD store instructions.**

Instruction Group	AArch64 Instruction	Execution Latency	Execution Throughput	Utilized Pipeline
ASIMD store, 1 element, multiple, 1 reg, D-form	ST1	-	1	Load/Store
ASIMD store, 1 element, multiple, 1 reg, Q-form	ST1	-	1	Load/Store
ASIMD store, 1 element, multiple, 2 reg, D-form	ST1	-	1	Load/Store
ASIMD store, 1 element, multiple, 2 reg, Q-form	ST1	-	1/2	Load/Store
ASIMD store, 1 element, multiple, 3 reg, D-form	ST1	-	1/2 <sup>[4]</sup>	Load/Store
ASIMD store, 1 element, multiple, 3 reg, Q-form	ST1	-	1/3	Load/Store
ASIMD store, 1 element, multiple, 4 reg, D-form	ST1	-	1/2	Load/Store

<sup>[4]</sup> Throughput=1/3 when the access is aligned and crosses 16B boundary, one more cycle is needed.

Instruction Group	AArch64 Instruction	Execution Latency	Execution Throughput	Utilized Pipeline
ASIMD store, 1 element, multiple, 4 reg, Q-form	ST1	-	1/4	Load/Store
ASIMD store, 1 element, one lane, B/H/S	ST1	-	1	Load/Store
ASIMD store, 1 element, one lane, D	ST1	-	1	Load/Store
ASIMD store, 2 element, multiple, D-form	ST2	-	1	Load/Store
ASIMD store, 2 element, multiple, Q-form	ST2	-	1/2	Load/Store
ASIMD store, 2 element, one lane, B/H/S	ST2	-	1	Load/Store
ASIMD store, 2 element, one lane, D	ST2	-	1	Load/Store
ASIMD store, 3 element, multiple, D-form, B/H/S	ST3	-	1/4	Load/Store
ASIMD store, 3 element, multiple, Q-form, B/H/S	ST3	-	1/6	Load/Store
ASIMD store, 3 element, multiple, Q-form, D	ST3	-	1/3	Load/Store
ASIMD store, 3 element, one lane, B/H/S	ST3	-	1/2	Load/Store
ASIMD store, 3 element, one lane, D	ST3	-	1/2	Load/Store
ASIMD store, 4 element, multiple, D-form, B/H/S	ST4	-	1/4	Load/Store
ASIMD store, 4 element, multiple, Q-form, B/H/S	ST4	-	1/8	Load/Store
ASIMD store, 4 element, multiple, Q-form, D	ST4	-	1/4	Load/Store
ASIMD store, 4 element, one lane, B/H/S	ST4	-	1/2	Load/Store
ASIMD store, 4 element, one lane, D	ST4	-	1/2	Load/Store

## 2.22 Cryptography extensions

**Table 2-21: AArch64 Cryptography instructions.**

Instruction Group	AArch64 Instruction	Execution Latency	Execution Throughput	Utilized Pipeline
Crypto AES ops	AESD, AESE, AESIMC, AESMC	3	2	Crypto
Crypto polynomial (64x64) multiply long	PMULL, PMULL2	3	2	VMC
Crypto SHA1 hash acceleration op	SHA1H	3	1	VALU
Crypto SHA1 hash acceleration ops	SHA1C, SHA1M, SHA1P	4	2	VMC
Crypto SHA1 schedule acceleration ops	SHA1SU0, SHA1SU1	3	2	VMC
Crypto SHA256 hash acceleration ops	SHA256H, SHA256H2	4	2	VMC
Crypto SHA256 schedule acceleration ops	SHA256SU0, SHA256SU1	4	2	VMC
Crypto SHA512 hash acceleration ops	SHA512H, SHA512H2, SHA512SU0, SHA512SU1	9	1/7	VMC
Crypto SHA3 ops	BCAX, EOR3	3	2	VALU
Crypto SHA3 ops, exclusive Or and rotate	XAR	4	2	VALU
Crypto SHA3 ops, rotate and exclusive Or	RAX1	3	2	VMC
Crypto SM3 ops	SM3PARTW1, SM3PARTW2, SM3SS1, SM3TT1A, SM3TT1B, SM3TT2A, SM3TT2B	9	1/7	VMC
Crypto SM4 ops	SM4E, SM4EKEY	9	1/7	VMC

## 2.23 CRC

**Table 2-22: AArch64 CRC instructions**

Instruction Group	AArch64 Instruction	Execution Latency	Execution Throughput	Utilized Pipeline
CRC checksum ops	CRC32, CRC32C, CRC32B, CRC32CB, CRC32CH, CRC32CW, CRC32CX, CRC32H, CRC32W, CRC32X	2	1	MAC

## 2.24 SVE Predicate instructions

**Table 2-23: SVE Predicate instructions.**

Instruction Group	AArch64 Instruction	Execution Latency	Execution Throughput	Utilized Pipeline
Loop control, based on predicate	BRKA, BRKB	2	1	PALU
Loop control, based on predicate and flag setting	BRKAS, BRKBS	2	1	PALU
Loop control, propagating	BRKN, BRKPA, BRKPB	2	1	PALU
Loop control, propagating and flag setting	BRKNS, BRKPAS, BRKPBS	2	1	PALU
Loop control, based on GPR	WHILEGE, WHILEGT, WHILEHI, WHILEHS, WHILELE, WHILELO, WHILELS, WHILELT, WHILERW, WHILEWR	2	1	PALU

Instruction Group	AArch64 Instruction	Execution Latency	Execution Throughput	Utilized Pipeline
Loop terminate	CTERMEQ, CTERMNE	1	1	ALU
Predicate counting scalar, add	ADDPL, ADDVL, RDVL	1	2	ALU
Predicate counting scalar	CNTB, CNTH, CNTW, CNTD, DECB, DECH, DECW, DECD, INCB, INCH, INCW, INCD	1	1	ALU
Predicate counting scalar, saturate	SQDECB, SQDECH, SQDECW, SQDECD, SQINCB, SQINCH, SQINCW, SQINCD, UQDECB, UQDECH, UQDECW, UQDECD, UQINCB, UQINCH, UQINCW, UQINCD	5	1	ALU
Predicate counting scalar, active predicate	CNTP, DECP, INCP	1	1	PALU
Predicate counting scalar, active predicate, saturating, 64-bit	SQDECP, SQINCP, UQDECP, UQINCP	2	1	VALU
Predicate counting scalar, active predicate, saturating, 32-bit	SQDECP, SQINCP	1	1	VALU
Predicate counting scalar, active predicate, saturating, 32-bit	UQDECP, UQINCP	2	1	VALU
Predicate counting vector, active predicate	CNTP, DECP, INCP	3	2	PALU
Predicate counting vector, active predicate, saturating	SQDECP, SQINCP, UQDECP, UQINCP	4	2	VALU
Predicate logical	AND, BIC, EOR, MOV, NAND, NOR, NOT, ORN, ORR	2	1	PALU

Instruction Group	AArch64 Instruction	Execution Latency	Execution Throughput	Utilized Pipeline
Predicate logical, flag setting	ANDS, BICS, EORS, MOVS, NANDS, NORs, NOTS, ORNS, ORRS	2	1	PALU
Predicate reverse	REV	1	1	PALU
Predicate select	SEL	2	1	PALU
Predicate set	PFALSE, PTRUE	1	1	PALU
Predicate set/initialize, set flags	PTRUES	2	1	PALU
Predicate find first/next	PFIRST, PNEXT	2	1	PALU
Predicate test	PTEST	1	1	PALU
Predicate transpose	TRN1, TRN2	1	1	PALU
Predicate unpack and widen	PUNPKHI, PUNPKLO	1	1	PALU
Predicate zip/unzip	ZIP1, ZIP2, UZP1, UZP2	1	1	PALU



Note

Instructions with dependencies may be co-issue.

## 2.25 SVE Integer instructions

**Table 2-24: SVE integer instructions.**

Instruction Group	AArch64 Instruction	Execution Latency	Execution Throughput	Utilized Pipeline
Arithmetic, absolute diff	SABD, UABD	3	2	VALU
Arithmetic, absolute diff accum	SABA, UABA	5	1/3	VALU
Arithmetic, absolute diff accum long	SABALB, SABALT, UABALB, UABALT	5	1/3	VALU

Instruction Group	AArch64 Instruction	Execution Latency	Execution Throughput	Utilized Pipeline
Arithmetic, absolute diff long	SABDLB, SABDLT, UABDLB, UABDLT	3	2	VALU
Arithmetic, basic	ABS, ADD, ADR, CNOT, NEG, SHADD, SHSUB, SHSUBR, SRHADD, SUB, UADDWB, UADDWT, UHADD, UHSUB, UHSUBR, URHADD	3	2	VALU
Arithmetic, basic	SUBHNB, SUBHNT, SUBR, USUBWB, USUBWT	4	2	VALU
Arithmetic, basic	SADDLB, SADDLBT, SADDLT, SADDWB, SADDWT, SSUBLB, SSUBLBT, SSUBLT, SSUBLTB, SSUBWB, SSUBWT, UADDLB, UADDLT, USUBLB, USUBLT	4	2	VALU
Arithmetic, complex	ADDHNB, ADDHNT, SQABS, SQADD, SQNEG, SQSUB, QSUBR, SUQADD, UQADD, UQSUB, UQSUBR, USQADD	4	2	VALU
Arithmetic, complex	RADDHNB, RADDHNT, RSUBHNB, RSUBHNT	6	1/3	VALU
Arithmetic, large integer	ADCLB, ADCLT, SBCLB, SBCLT	4	2	VALU
Arithmetic, pairwise add	ADDP	3	2	VALU

Instruction Group	AArch64 Instruction	Execution Latency	Execution Throughput	Utilized Pipeline
Arithmetic, pairwise add and accum long	SADALP, UADALP	6	1/4	VALU
Arithmetic, shift	ASR, ASRR, LSL, LSLR, LSR, LSRR	3	2	VALU
Arithmetic, shift and accumulate	USRA	3	2	VALU
Arithmetic, shift and accumulate complex, round	SRSRA, URSRA	5	1/3	VALU
Arithmetic, shift and accumulate complex	SSRA	3	2	VALU
Arithmetic, shift by immediate	SHRNB, SHRNT, SSHLLB, SSHLLT, USHLLB, USHLLT	3	2	VALU
Arithmetic, shift by immediate and insert	SLI, SRI	3	2	VALU
Arithmetic, shift complex	RSHRNB, RSHRNT, SQRSHL, SQRSHLR, SQRSHRNB, SQRSHRNT, SQRSHRUNB, SQRSHRUNT, SQSHL, SQSHLR, SQSHLU, SQSHRNB, SQSHRNT, SQSHRUNB, SQSHRUNT, UQRSHL, UQRSHLR, UQRSHRNB, UQRSHRNT, UQSHL, UQSHLR, UQSHRNB, UQSHRNT	4	2	VALU
Arithmetic, shift right for divide	ASRD	4	2	VALU



Instruction Group	AArch64 Instruction	Execution Latency	Execution Throughput	Utilized Pipeline
Arithmetic, shift rounding	SRSHL, SRSHLR, SRSHR, URSHL, URSHLR, URSHR	4	2	VALU
Bit manipulation (B)	BDEP, BEXT, BGRP	13	1/11	VMC
Bit manipulation (H)	BDEP, BEXT, BGRP	21	1/19	VMC
Bit manipulation (S)	BDEP, BEXT, BGRP	37	1/35	VMC
Bit manipulation (D)	BDEP, BEXT, BGRP	68	1/66	VMC
Bitwise select	BSL, BSL1N, BSL2N, NBSL	3	2	VALU
Count/reverse bits	CLS, CLZ, RBIT	3	2	VALU
Count (B,H)	CNT	3	2	VALU
Count (S)	CNT	6	1/4	VALU
Count (D)	CNT	9	1/7	VALU
Broadcast logical bitmask immediate to vector	DUPM	4	2	VALU
Compare and set flags	CMPEQ, CMPGE, CMPGT, CMPHI, CMPHS, CMPLE, CMPLO, CMPLS, CMPLT, CMPNE	5	1	VALU
Complex add	CADD	3	2	VALU
Complex add saturating	SQCADD	4	2	VALU
Complex dot product 8-bit element	CDOT	4	2	VMAC
Complex dot product 16-bit element	CDOT	4	2	VMAC
Complex multiply-add B, H, S element size	CMLA	4	2	VMAC
Complex multiply-add D element size	CMLA	4	2	VMAC
Conditional extract operations, general purpose register	CLASTA, CLASTB	4	1/4	VALU
Conditional extract operations, SIMD&FP scalar and vector forms	CLASTA, CLASTB, COMPACT, SPLICE	4	2	VALU

Instruction Group	AArch64 Instruction	Execution Latency	Execution Throughput	Utilized Pipeline
Convert to floating point, 64b to float or convert to double	SCVTF, UCVTF	4	2	VALU
Convert to floating point, 32b to single or half	SCVTF, UCVTF	4	2	VALU
Convert to floating point, 16b to half	SCVTF, UCVTF	4	2	VALU
Copy	CPY	3	2	VALU
Divides, 32 bit	SDIV, SDIVR, UDIV, UDIVR	15	1/12	VMC
Divides, 64 bit	SDIV, SDIVR, UDIV, UDIVR	26	1/23	VMC
Dot product, 8 bit	SDOT, UDOT	4	2	VMAC
Dot product, 8 bit, using signed and unsigned integers	SUDOT, USDOT	4	2	VMAC
Dot product, 16 bit	SDOT, UDOT	4	2	VMAC
Duplicate, immediate and indexed form	DUP	3	2	VALU
Duplicate, indexed > elem	DUP	3	2	VALU
Duplicate, scalar form	DUP	3	2	VALU
Extend, sign or zero	SXTB, SXTH, SXTW, UXTB, UXTH, UXTW	3	2	VALU
Extract	EXT	3	2	VALU
Extract narrow saturating	SQXTNB, SQXTNT, SQXTUNB, SQXTUNT, UQXTNB, UQXTNT, UQXTUNB, UQXTUNT	4	2	VALU
Extract/insert operation, SIMD and FP scalar form	LASTA, LASTB, INSR	4	2	VALU
Extract operation, scalar	LASTA, LASTB	8	1/4	VALU
Insert operation, scalar	INSR	4	2	VALU
Histogram operations	HISTCNT, HISTSEG	6	1/4	VALU0

Instruction Group	AArch64 Instruction	Execution Latency	Execution Throughput	Utilized Pipeline
Horizontal operations, B, H, S form, immediate operands only	INDEX	4	2	VMAC
Horizontal operations, B, H, S form, scalar, immediate operands or immediate, scalar operands	INDEX	4	1	VMAC
Horizontal operations, D form, immediate operands only	INDEX	4	2	VMAC
Horizontal operations, D form, scalar, immediate operands or immediate, scalar operands	INDEX	4	1	VMAC
Logical ops	AND, BIC, EON, EOR, MOV, NOT, ORN, ORR	3	2	VALU
Logical, exclusive or bottom-top and top-bottom	EORBT, EORTB	4	2	VALU
Max/min, basic and pairwise	SMAX, SMAXP, SMIN, SMINP, UMAX, UMAXP, UMIN, UMINP	3	2	VALU
Matching operations	MATCH, NMATCH	8	1/4	VALU
Matrix multiply-accumulate	SMMLA, UMMLA, USMMLA	4	2	VMAC
Move prefix	MOVPRFX	3	2	VALU
Multiply, B, H, S element size	MUL, SMULH, UMULH	4	2	VMAC
Multiply, D element size	MUL, SMULH, UMULH	4	2	VMAC
Multiply long	SMULLB, SMULLT, UMULLB, UMULLT	4	2	VMAC
Multiply accumulate, B, H, S element size	MLA, MLS, MAD, MSB	4	2	VMAC
Multiply accumulate, D element size	MLA, MLS, MAD, MSB	4	2	VMAC

Instruction Group	AArch64 Instruction	Execution Latency	Execution Throughput	Utilized Pipeline
Multiply accumulate long	SMLALB, SMLALT, SMLS LB, SMLS LT, UMLALB, UMLALT, UMLS LB, UMLS LT	4	2	VMAC
Multiply accumulate saturating doubling long regular	SQDMLALB, SQDMLALT, SQDMLALBT, SQDMLS LB, SQDMLS LT, SQDMLS LBT	4	2	VMAC
Multiply saturating doubling high, B, H, S element size	SQDMULH	4	2	VMAC
Multiply saturating doubling high, D element size	SQDMULH	4	2	VMAC
Multiply saturating doubling long	SQDMULLB, SQDMULLT	4	2	VMAC
Multiply saturating rounding doubling regular/complex accumulate, B, H, S element size	SQRDMLAH, SQRDMLSH, SQRDCMLAH	4	2	VMAC
Multiply saturating rounding doubling regular/complex accumulate, D element size	SQRDMLAH, SQRDMLSH, SQRDCMLAH	4	2	VMAC
Multiply saturating rounding doubling regular/complex, B, H, S element size	SQRDMULH	4	2	VMAC
Multiply saturating rounding doubling regular/complex, D element size	SQRDMULH	4	2	VMAC
Multiply/multiply long, (8, 16, 32) polynomial	PMUL, PMULLB, PMULLT	3	2	VALU
Multiply/multiply long, (64) polynomial	PMULLB, PMULLT	9	1/7	VMC
Predicate counting vector	DECB, DECH, DECW, DECD, INCB, INCH, INCW, INCD	4	2	VALU

Instruction Group	AArch64 Instruction	Execution Latency	Execution Throughput	Utilized Pipeline
Predicate counting vector, saturating	SQDECB, SQDECH, SQDECW, SQDECD, SQINCB, SQINCH, SQINCW, SQINCD, UQDECB, UQDECH, UQDECW, UQDECD, UQINCB, UQINCH, UQINCW, UQINCD	4	2	VALU
Reciprocal estimate	URECPE, URSQRTE	4	2	VMAC
Reduction, arithmetic, B form	SADDV, UADDV, SMAXV, SMINV, UMAXV, UMINV	4	1	VALU0
Reduction, arithmetic, H form	SADDV, UADDV, SMAXV, SMINV, UMAXV, UMINV	4	1	VALU0
Reduction, arithmetic, S form	SADDV, UADDV, SMAXV, SMINV, UMAXV, UMINV	4	1	VALU0
Reduction, arithmetic, D form	SADDV, UADDV, SMAXV, SMINV, UMAXV, UMINV	4	1	VALU0
Reduction, logical	ANDV, EORV, ORV	4	1	VALU0
Reverse, vector	REV, REVB, REVH, REVW	3	2	VALU
Select, vector form	SEL	2	2	VALU
Table lookup	TBL	4	2	VALU
Table lookup, double table	TBL	8	1/5	VALU
Table lookup extension	TBX	4	2	VALU
Transpose, vector form	TRN1, TRN2	3	2	VALU
Unpack and extend	SUNPKHI, SUNPKLO, UUNPKHI, UUNPKLO	4	2	VALU
Zip/unzip	UZP1, UZP2, ZIP1, ZIP2	3	2	VALU

## 2.26 SVE FP data processing instructions

**Table 2-25: SVE Floating-point instructions.**

Instruction Group	AArch64 Instruction	Execution Latency	Execution Throughput	Utilized Pipeline
Floating point absolute value/difference	FABD, FABS	4	2	VALU
Floating point arithmetic	FADD, FADDP, FNEG, FSUB, FSUBR	4	2	VALU
Floating point associative add, F16	FADDA	32	1/25	VALU
Floating point associative add, F32	FADDA	16	1/9	VALU
Floating point associative add, F64	FADDA	8	2/5	VALU
Floating point compare	FACGE, FACGT, FACLE, FACLT, FCMEQ, FCMGE, FCMGT, FCMLE, FCMLT, FCMNE, FCMUO	4	1	VALU
Floating point complex add	FCADD	4	2	VALU
Floating point complex multiply add	FCMLA	4	2	VMAC
Floating point convert, long to narrow	FCVT, FCVTLT, FCVTNT	4	2	VALU
Floating point convert, round to odd	FCVTX, FCVTXNT	4	2	VALU
Floating point base2 log, F16	FLOGB	4	2	VMAC
Floating point base2 log, F32	FLOGB	4	2	VMAC
Floating point base2 log, F64	FLOGB	4	2	VMAC
Floating point convert to integer, F16	FCVTZS, FCVTZU	4	2	VALU
Floating point convert to integer, F32	FCVTZS, FCVTZU	4	2	VALU

Instruction Group	AArch64 Instruction	Execution Latency	Execution Throughput	Utilized Pipeline
Floating point convert to integer, F64	FCVTZS, FCVTZU	4	2	VALU
Floating point copy	FCPY, FDUP, FMOV	3	2	VALU
Floating point divide, F16	FDIV, FDIVR	8	1/5	VMC
Floating point divide, F32	FDIV, FDIVR	13	1/10	VMC
Floating point divide, F64	FDIV, FDIVR	22	1/19	VMC
Floating point min/max pairwise	FMAXP, FMAXNMP, FMINP, FMINNMP	4	2	VALU
Floating point min/max	FMAX, FMIN, FMAXNM, FMINNM	4	2	VALU
Floating point multiply	FSCALE, FMUL, FMULX	4	2	VMAC
Floating point multiply accumulate	FMLA, FMLS, FMAD, FMSB, FNMAD, FNMLA, FNMLS, FNMSB	4	2	VMAC
Floating point multiply add/sub accumulate long	FMLALB, FMLALT, FMLSLB, FMLSLT	4	2	VMAC
Floating point reciprocal estimate, F16	FRECPE, FRECPX, FRSQRTE	4	2	VMAC
Floating point reciprocal estimate, F32	FRECPE, FRECPX, FRSQRTE	4	2	VMAC
Floating point reciprocal estimate, F64	FRECPE, FRECPX, FRSQRTE	4	2	VMAC
Floating point reciprocal step	FRECPS, FRSQRTS	4	2	VMAC
Floating point max/min reduction	FMAXNMV, FMAXV, FMINNMV, FMINV	4	1	VALU0
Floating point reduction, F16	FADDV	12	1/5	VALU0
Floating point reduction, F32	FADDV	8	2/5	VALU0
Floating point reduction, F64	FADDV	4	2	VALU0

Instruction Group	AArch64 Instruction	Execution Latency	Execution Throughput	Utilized Pipeline
Floating point round to integral, F16	FRINTA, FRINTI, FRINTM, FRINTN, FRINTP, FRINTX, FRINTZ	4	2	VALU
Floating point round to integral, F32	FRINTA, FRINTI, FRINTM, FRINTN, FRINTP, FRINTX, FRINTZ	4	2	VALU
Floating point round to integral, F64	FRINTA, FRINTI, FRINTM, FRINTN, FRINTP, FRINTX, FRINTZ	4	2	VALU
Floating point square root, F16	FSQRT	8	1/5	VMC
Floating point square root, F32	FSQRT	12	1/9	VMC
Floating point square root F64	FSQRT	22	1/19	VMC
Floating point trigonometric exponentiation	FEXPA	4	2	VMAC
Floating point trigonometric multiply add	FTMAD	4	2	VMAC
Floating point trigonometric starting value	FTSMUL	4	2	VMAC
Floating point trigonometric select coefficient	FTSSEL	3	2	VALU



Floating-point division operations may finish early if the divisor is a power of two.



## 2.27 SVE BFloat16 (BF16) instructions

**Table 2-26: SVE Bfloat16 (BF16) instructions.**

Instruction Group	AArch64 Instruction	Execution Latency	Execution Throughput	Utilized Pipeline
Convert, F32 to BF16	BFCVT, BFCVTNT	4	2	VALU
Dot product	BFDOT	10	2	VMAC,VALU
Matrix multiply accumulate	BFMMLA	14	1	VMAC,VALU
Multiply accumulate long	BFMLALB, BFMLALT	4	2	VMAC

## 2.28 SVE Load instructions

The latencies shown in Table 2-27 assume the memory access hits in the Level 1 Data Cache. Base register updates are done in parallel to the operation.

**Table 2-27: SVE Load instructions.**

Instruction Group	AArch64 Instruction	Execution Latency	Execution Throughput	Utilized Pipeline
Load vector	LDR	3	1	Load/Store, Load
Load predicate	LDR	3	1	Load/Store
Contiguous load, scalar + imm	LD1B, LD1D, LD1H, LD1W, LD1SB, LD1SH, LD1SW	3	1	Load/Store, Load
Contiguous load, scalar + scalar	LD1B, LD1D, LD1H, LD1W, LD1SB, LD1SH, LD1SW	3	1	Load/Store, Load
Contiguous load broadcast, scalar + imm	LD1RB, LD1RH, LD1RD, LD1RW, LD1RSB, LD1RSH, LD1RSW, LD1RQB, LD1RQD, LD1RQH, LD1RQW	3	1	Load/Store, Load

Instruction Group	AArch64 Instruction	Execution Latency	Execution Throughput	Utilized Pipeline
Contiguous load broadcast, scalar + scalar	LD1RQB, LD1RQD, LD1RQH, LD1RQW	3	1	Load/Store, Load
Non-temporal load, scalar + imm	LDNT1B, LDNT1D, LDNT1H, LDNT1W	3	1	Load/Store, Load
Non-temporal load, scalar + scalar	LDNT1B, LDNT1D, LDNT1H, LDNT1W	3	1	Load/Store, Load
Non-temporal gather load, vector + scalar 32-bit element size	LDNT1B, LDNT1H, LDNT1W, LDNT1SB, LDNT1SH	9	1/7	Load/Store
Non-temporal gather load, vector + scalar 64-bit element size	LDNT1B, LDNT1D, LDNT1H, LDNT1W, LDNT1SB, LDNT1SH, LDNT1SW	7	1/6	Load/Store
Contiguous first faulting load, scalar + scalar	LDFF1B, LDFF1D, LDFF1H, LDFF1W, LDFF1SB, LDFF1SD, LDFF1SH, LDFF1SW	3	1	Load/Store, Load
Contiguous non-faulting load, scalar + imm	LDNF1B, LDNF1D, LDNF1H, LDNF1W, LDNF1SB, LDNF1SH, LDNF1SW	3	1	Load/Store, Load
Contiguous load two structures to two vectors, scalar + imm	LD2B, LD2D, LD2H, LD2W	3	1	Load/Store
Contiguous load two structures to two vectors, scalar + scalar	LD2B, LD2D, LD2H, LD2W	3	1/2	Load/Store
Contiguous load three structures to three vectors, scalar + imm	LD3B, LD3D, LD3H, LD3W	5	1/3	Load/Store

Instruction Group	AArch64 Instruction	Execution Latency	Execution Throughput	Utilized Pipeline
Contiguous load three structures to three vectors, scalar + scalar	LD3B, LD3D, LD3H, LD3W	5	1/4	Load/Store
Contiguous load four structures to four vectors, scalar + imm	LD4B, LD4D, LD4H, LD4W	5	1/3	Load/Store
Contiguous load four structures to four vectors, scalar + scalar	LD4B, LD4D, LD4H, LD4W	5	1/4	Load/Store
Gather load, vector + imm, 32-bit element size	LD1B, LD1H, LD1W, LD1SB, LD1SH, LD1SW, LDFF1B, LDFF1H, LDFF1W, LDFF1SB, LDFF1SH, LDFF1SW	9	1/7	Load/Store
Gather load, vector + imm, 64-bit element size	LD1B, LD1D, LD1H, LD1W, LD1SB, LD1SH, LD1SW, LDFF1B, LDFF1D, LDFF1H, LDFF1W, LDFF1SB, LDFF1SD, LDFF1SH, LDFF1SW	7	1/6	Load/Store
Gather load, 32-bit scaled offset	LD1H, LD1SH, LDFF1H, LDFF1SH, LD1W, LDFF1W, LDFF1SW	7	1/7	Load/Store
Gather load, 32-bit unpacked unscaled offset	LD1B, LD1SB, LDFF1B, LDFF1SB, LD1D, LDFF1D, LD1H, LD1SH, LDFF1H, LDFF1SH, LD1W, LD1SW, LDFF1W, LDFF1SW	7	1/6	Load/Store

Instruction Group	AArch64 Instruction	Execution Latency	Execution Throughput	Utilized Pipeline
Gather load, 32-bit unscaled offset	LD1B, LD1H, LD1W, LDFF1B, LDFF1H, LDFF1SB, LDFF1SH, LDFF1W	7	1/7	Load/Store
Gather load, 32-bit unpacked scaled offset	LD1B, LD1SB, LDFF1B, LDFF1SB, LD1D, LDFF1D, LD1H, LD1SH, LDFF1H, LDFF1SH, LD1W, LD1SW, LDFF1W, LDFF1SW	7	1/6	Load/Store
Gather load, 64-bit unscaled offset	LD1B, LD1D, LD1H, LD1SB, LD1SH, LD1SW, LD1W, LDFF1B, LDFF1D, LDFF1H, LDFF1SB, LDFF1SH, LDFF1SW, LDFF1W	7	1/6	Load/Store
Gather load, 64-bit scaled offset	LD1B, LD1D, LD1H, LD1SB, LD1SH, LD1SW, LD1W, LDFF1B, LDFF1D, LDFF1H, LDFF1SB, LDFF1SH, LDFF1SW, LDFF1W	7	1/6	Load/Store

## 2.29 SVE Store instructions

Base register updates are done in parallel to the operation.

**Table 2-28: SVE Store instructions.**

Instruction Group	AArch64 Instruction	Execution Latency	Execution Throughput	Utilized Pipeline
Store from predicate reg	STR	-	1	Load/Store
Store from vector reg	STR	-	1	Load/Store
Contiguous store, scalar + imm	ST1B, ST1H, ST1D, ST1W	-	1	Load/Store
Contiguous store, scalar + scalar	ST1H, ST1B, ST1D, ST1W	-	1	Load/Store
Contiguous store two structures from two vectors, scalar + imm	ST2B, ST2H, ST2D, ST2W	-	1/2	Load/Store
Contiguous store two structures from two vectors, scalar + scalar	ST2H, ST2B, ST2D, ST2W	-	1/2	Load/Store
Contiguous store three structures from three vectors, scalar + imm	ST3B, ST3H, ST3W	-	1/6	Load/Store
Contiguous store three structures from three vectors, scalar + imm, doubleword	ST3D	-	1/3	Load/Store
Contiguous store three structures from three vectors, scalar + scalar	ST3B, ST3H, ST3W	-	1/6	Load/Store
Contiguous store three structures from three vectors, scalar + scalar, doubleword	ST3D	-	1/3	Load/Store
Contiguous store four structures from four vectors, scalar + imm	ST4B, ST4H, ST4W	-	1/8	Load/Store
Contiguous store four structures from four vectors, scalar + imm, doubleword	ST4D	-	1/4	Load/Store

Instruction Group	AArch64 Instruction	Execution Latency	Execution Throughput	Utilized Pipeline
Contiguous store four structures from four vectors, scalar + scalar	ST4B, ST4H, ST4W	-	1/8	Load/Store
Contiguous store four structures from four vectors, scalar + scalar, doubleword	ST4D	-	1/4	Load/Store
Non-temporal store, scalar + imm	STNT1B, STNT1D, STNT1H, STNT1W	-	1	Load/Store
Non-temporal store, scalar + scalar	STNT1H, STNT1B, STNT1D, STNT1W	-	1	Load/Store
Scatter non-temporal store, vector + scalar 32-bit element size	STNT1B, STNT1H, STNT1W	-	1/9	Load/Store
Scatter non-temporal store, vector + scalar 64-bit element size	STNT1B, STNT1D, STNT1H, STNT1W	-	1/7	Load/Store
Scatter store vector + imm 32-bit element size	ST1B, ST1H, ST1W	-	1/9	Load/Store
Scatter store vector + imm 64-bit element size	ST1B, ST1D, ST1H, ST1W	-	1/7	Load/Store
Scatter store, 32-bit scaled offset	ST1H, ST1W	-	1/9	Load/Store
Scatter store, 32-bit unpacked unscaled offset	ST1B, ST1D, ST1H, ST1W	-	1/7	Load/Store
Scatter store, 32-bit unpacked scaled offset	ST1D, ST1H, ST1W	-	1/7	Load/Store
Scatter store, 32-bit unscaled offset	ST1B, ST1H, ST1W	-	1/9	Load/Store
Scatter store, 64-bit unscaled offset	ST1B, ST1D, ST1H, ST1W	-	1/7	Load/Store
Scatter store, 64-bit scaled offset	ST1D, ST1H, ST1W	-	1/7	Load/Store

## 2.30 SVE Miscellaneous instructions

**Table 2-29: SVE Miscellaneous instructions**

Instruction Group	AArch64 Instruction	Execution Latency	Execution Throughput	Utilized Pipeline
Read first fault register, unpredicated	RDFFR	1	1	Load/Store
Read first fault register, predicated	RDFFR	3	1	Load/Store
Read first fault register and set flags	RDFFRS	3	1	Load/Store
Set first fault register	SETFFR	1	1	Load/Store
Write to first fault register	WRFFR	1	1	Load/Store

## 2.31 SVE Cryptography instructions

**Table 2-30: SVE cryptography instructions.**

Instruction Group	AArch64 Instruction	Execution Latency	Execution Throughput	Utilized Pipeline
Crypto AES ops	AESD, AESE, AESIMC, AESMC	3	2	Crypto
Crypto SHA3 ops	BCAX, EOR3	3	2	VALU
Crypto SHA3 ops, exclusive Or and rotate	XAR	4	2	VALU
Crypto SHA3 ops RAX1	RAX1	3	2	VALU
Crypto SM4 ops	SM4E, SM4EKEY	9	1/7	VMC

## 2.32 MOPS instructions

**Table 2-31: MOPS instructions.**

Instruction Group	AArch64 Instruction	Execution Latency	Execution Throughput	Utilized Pipeline
Memory Copy Forward-only Prologue	CPYFP, CPYFPN, CPYFPRN, CPYFPRT, CPYFPRTN, CPYFPRTRN, CPYFPRTWN, CPYFPT, CPYFPTN, CPYFPTRN, CPYFPTWN, CPYFPWN, CPYFPWT, CPYFPWTN, CPYFPWTRN, CPYFPWTWN	2	1/2	ALU, Load/Store
Memory Copy Forward-only Main	CPYFM, CPYFMN, CPYFMRN, CPYFMRT, CPYFMRTN, CPYFMRTRN, CPYFMRTWN, CPYFMT, CPYFMTN, CPYFMTRN, CPYFMTWN, CPYFMWN, CPYFMWT, CPYFMWTN, CPYFMWTRN, CPYFMWTWN	1 <sup>[5]</sup>	1	ALU, Load/Store

<sup>[5]</sup> Actual execution latency depends on  $Xn$ . For  $Xn > 16$ , latency will be  $\lfloor \frac{Xn-16}{16} \rfloor$ .



<sup>[6]</sup> Actual execution latency depends on  $Xn$ . For  $Xn > 16$ , latency will be  $\lfloor \frac{Xn-16}{16} \rfloor$ .

Instruction Group	AArch64 Instruction	Execution Latency	Execution Throughput	Utilized Pipeline
Memory Copy Epilogue	CPYE, CPYEN, CPYERN, CPYERT, CPYERTN, CPYERTRN, CPYERTWN, CPYET, CPYETN, CPYETRN, CPYETWN, CPYEWN, CPYEWT, CPYEWTN, CPYEWTRN, CPYEWTRWN	1	1	ALU, Load/Store
Memory Set Prologue	SETP, SETPN, SETPT, SETPTN	2	1/2	ALU, Load/Store
Memory Set Main	SETM, SETMN, SETMT, SETMTN	1 <sup>[7]</sup>	1	ALU, Load/Store
Memory Set Epilogue	SETE, SETEN, SETET, SETETN	1	1	ALU, Load/Store
Memory Set with tag setting Prologue	SETGP, SETGPN, SETGPT, SETGPTN	2	1/2	ALU, Load/Store
Memory Set with tag setting Main	SETGM, SETGMN, SETGMT, SETGMTN	1 <sup>[8]</sup>	1	ALU, Load/Store
Memory Set with tag setting Epilogue	SETGE, SETGEN, SETGET, SETGETN	1	1	ALU, Load/Store

<sup>[7]</sup> Actual execution latency depends on  $Xn$ . For  $Xn > 16$ , latency will be  $\lfloor \frac{Xn-16}{16} \rfloor$ .

<sup>[8]</sup> Actual execution latency depends on  $Xn$ . For  $Xn > 16$ , latency will be  $\lfloor \frac{Xn-16}{16} \rfloor$ .

## 2.33 SME instructions

SME instructions are instructions within the SME extensions 2-31.

SME instructions are decoded on the CPU and then issued to the C1-SME2. The overall performance of the instructions rely not only on the CPU, but also the C1-SME2 unit and transport layer.

This section only describes the performance from the CPU point of view.

The CPU has a maximum bandwidth of three instructions to the C1-SME2 unit. With the use of **MOVPRFX** fusion this can be increased to four.

There are five different classes of SME instructions from a performance point of view in the CPU.

The instructions related to entering and exiting streaming mode, system related instructions e.g. FPSR updates, load store related instructions, predicate and flag related instructions and finally data processing instructions.

Instruction fusion in the form of **MOVPRFX** is supported in the same way as in SVE mode 2-31.

### 2.33.1 Entering and leaving streaming mode

To enter and leave streaming mode use the **SMSTART** and **SMSTOP** instructions. The **MSR SVCR, Xn** versions incur a penalty in terms of a flush. **SMSTART** and **SMSTOP** are single issued.

### 2.33.2 Predicate and flag related instructions

Predicate only instructions, where the producer and consumers are predicate and or integer, have the same performance as in SVE. With the exception of the **WHILE** instructions which have one less cycle of throughput.

Instructions which produce the predicate value based on vector registers, e.g. a **CMPEQ <Pd>.<T>**, **<Pg>/Z**, **<Zn>.<T>**, **<Zm>.<T>** are executed on the C1-SME2 unit. Therefore any instruction consuming the same predicate on the CPU, e.g. predicated load / stores or predicate operation will stall until the result has been produced.

These code constructs should be avoided.

The CPU has a mechanism to deal with multiple outstanding writes to the same predicate register. With a maximum of 16 outstanding writes. For example if a **CMP** is executed in a loop and the predicate is used only as a consumer in subsequent data processing instructions executed on the C1-SME2, there is no need for the CPU to stall as there is no consumer of the predicate in the CPU itself. This mechanism is the same for the **NZCV** flags which also supports a maximum of 16 outstanding writes.

The 16 outstanding writes are only tracked for a single predicated register at a time. Any other predicate register which is outstanding at the same time would incur stall penalties until the result is produced.

### 2.33.3 Load and store instructions

Load and store instructions have a bandwidth of two instead of three as the bandwidth to the memory system is two wide. They can be issued together with data processing instructions.

### 2.33.4 Data processing instructions

These instructions can use the full bandwidth of three instructions per cycle.

### 2.33.5 System register instructions

System instructions e.g. MSR FPSR, Xn are single issue only

## 3 Special considerations

### 3.1 Issue constraints

The issue queue has space for three instructions that support a maximum of (excluding Floating-Point. Predicate, SIMD, SVE register accesses):

- Four general purpose destination registers.
- Six general purpose source registers.

An instruction will occupy two entries when it has either:

- Three or more general purpose destination registers.
- Three or more general purpose source registers.

An instruction will stall if insufficient space is available in the issue queue.

AES instructions will stall until there is at least one other instruction available to be issued (see 3.2 Instruction fusion).

A maximum of three issue queue entries can be co-issued per cycle (ignoring hazards) consisting of at most:

- Two ALU instructions.
- Two load instructions.
- One store instruction.
- Two VPU data processing instructions.

Multicycle entries disable co-issuing for all cycles of the operation but the last.

The following are multicycle:

- Atomic instructions with *Acquire* or *Release* semantics.
- Loads that load more than 256-bit of data.
- Stores that store more than 128-bits of data.
- Stores with *Release* semantics.
- RDFFRS instructions.

## 3.2 Instruction fusion

C1-Nano Core can accelerate key instruction pairs in an operation called fusion.

The following instruction pairs can be fused for increased execution efficiency:

- 'AESE + AESMC' and 'AESD + AESIMC' (see 3.3)
- MOVPRFX fusion: C1-Nano Core implements instruction fusion for MOVPRFX instructions followed by SVE data processing instructions in all cases where the instruction pair is defined as architecturally predictable other than those listed below, and the fused pair will execute with the latency of the SVE data processing instruction.

Due to microarchitectural limitations, the following instructions will not fuse with an unpredicated MOVPRFX: FCMLA, FMAD, FMLA, FMLS, FNMAD, FNMLA, FNMLS, FNMSB, MAD, MLA, MLS, MSB, UDOT, BFMLALB, BFMLALT, SMMLA, UMMLA, USMMLA, USDOT, SUDOT.

The following instructions will not fuse with a predicated or unpredicated MOVPRFX: CNT, SABA, SABALB, SABALT, UABA, UABALB, UABALT, URSRA.

## 3.3 Branch instruction alignment

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



Note

For best case performance, avoid placing more than one conditional branch instructions within an aligned 16-byte instruction memory region.

## 3.4 Load / Store Alignment

The Armv8-A architecture allows many types of load and store accesses to be arbitrarily aligned. C1-Nano Core handles most unaligned accesses without performance penalties. However, there are cases which could reduce bandwidth or incur additional latency, as described below.

- Quad-word load operations that are not 4-byte aligned.
- Load operations that cross a 32-byte boundary.
- Store operations that cross a 16-byte boundary.

## 3.5 A64 low latency pointer forwarding

In the A64 instruction set the following pointer sequence is expected to be common to generate load-store addresses:

```
adrp x0, <const>
ldrp x0, [x0, #1012 <const>]
```

In C1-Nano Core, there are dedicated forwarding paths that always allow this sequence to be executed without incurring a dependency-based stall.

## 3.6 AUT\* RET forwarding

In the A64 instruction set any variant of the AUT instruction will be dual issued with the directly following RET instruction. The latency of the AUT instruction for the dependency of the LR does not apply for these cases.

## 3.7 SIMD MAC forwarding

For the following integer SIMD instructions:

MUL, MLA, MLS, UMULL, UMULL2, SMULL, SMULL2, UMLAL, UMLAL2, SMLAL, SMLAL2, UMLSL, UMLSL2, SMLSL, SMLAL2, UDOT, SDOT

A dedicated MAC accumulator forwarding path is present. This forwarding path will be triggered only when two consecutive instructions satisfy the following conditions:

- Both instructions read from/write to the same destination/accumulator register.
- Both instructions use the same destination element size.
- The instructions target the same destination register size (128-bit or 64-bit).

When this forwarding path is active, the latency between the above instructions will be 1 cycle.

## 3.8 Memory Tagging Extensions

Enabling precise tag checking can prevent C1-Nano Core from entering write-streaming mode. This can reduce performance and increase power for larger writes, and memset or memcpy-like workloads.

## 3.9 Memory routines

C1-Nano 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. C1-Nano Core implements Option B for all instructions of FEAT\_MOPS. Those are referenced as Memory Copy and Memory Set instructions in the Armv9.3-A architecture which exhaustively describes all supported instructions, such as nontemporal versions.

**Table 3-1: C1-Nano FEAT\_MOPS bandwidth**

Operation	FEAT_MOPS Instructions	Operation Bandwidth
Memory copying (memcpy, memmove)	CPY*	16 bytes/cycles
Memory setting (memset) to 0	SET*	16 bytes/cycles
Memory setting to non-zero value	SET*	16 bytes/cycles

The bandwidth achievable with SET\* is less than that with DC ZVA, given the same alignment and data size conditions. Therefore, DC ZVA should be used for optimal memset to zero. An example routine of memset to zero using DC ZVA is shown in Figure 3-4.

In case one does not want to use FEAT\_MOPS instructions, legacy memcpy and memset routines can be used. These routines and corresponding recommendations are described below.

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

- Unroll the loop to include multiple load and store operations per iteration, minimizing the overheads of looping.
- Stores should be aligned on a 16-byte boundary wherever possible.
- Loads should not cross a 32-byte boundary as they incur a penalty.



Note

Updated optimized routines, including those utilizing FEAT\_MOPS instructions, are available:

<https://github.com/ARM-software/optimized-routines/tree/master/string/aarch64>



Figure 3-1 shows a code snippet from the inner loop of memory copy routine that copies at least 128 bytes. The loop copies 64 bytes per iteration and prefetches one iteration ahead.

**Figure 3-1: Code Snippet from memcpy routine - large copy inner loop.**

```
L(loop64_simd):
    str A_q, [dst, 16]
    ldr A_q, [src, 16]
    str B_q, [dst, 32]
    ldr B_q, [src, 32]
    str C_q, [dst, 48]
    ldr C_q, [src, 48]
    str D_q, [dst, 64]!
    ldr D_q, [src, 64]!
    subs count, count, 64
    b.hi L(loop64_simd)
```

Figure 3-2 shows a code snippet from the inner loop memory copy routine that copies 0 to 16 bytes.

**Figure 3-2: Code Snippet from memcpy routine - small copy inner loop.**

```
.p2align 4
/* Small copies: 0..16 bytes. */
L(copy16_simd):
/* 8-15 bytes. */
cmp count, 8
b.lo 1f
ldr A_l, [src]
ldr A_h, [srcend, -8]
str A_l, [dstin]
str A_h, [dstend, -8]
ret
.p2align 4
1:
/* 4-7 bytes. */
tbz count, 2, 1f
ldr A_lw, [src]
ldr A_hw, [srcend, -4]
str A_lw, [dstin]
str A_hw, [dstend, -4]
ret
---
bic src, src, 15
```

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

Unroll the loop to include multiple store operations per iteration, minimizing the overheads of looping. Figure 3-3 shows code from the memset routine to set 17 to 96 bytes.

**Figure 3-3: Code snippet from memset routine.**

```

L(set_medium):
    str q0, [dstin]
    tbnz count, 6, L(set96)
    str q0, [dstend, -16]
    tbz count, 5, 1f
    str q0, [dstin, 16]
    str q0, [dstend, -32]
1:    ret

```

To achieve maximum performance on memset to zero, it is recommended that one use DC ZVA instead of STP/SET\*. Figure 3-4 shows code from the memset routine to illustrate the usage of DC ZVA.

**Figure 3-4: Code snippet from memset to zero routine.**

```

L(zva_loop):
    add dst, dst, 64
    dc zva, dst
    subs count, count, 64
    b.hi L(zva_loop)
    stp q0, q0, [dstend, -64]
    stp q0, q0, [dstend, -32]
    ret

```

## 3.10 Cache maintenance operations

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

## 3.11 Cache access latencies

The latency numbers for load instructions given in Instruction characteristics section assume the ideal case. It should be noted that more cycles will be added to these access delays depending on which level of cache is accessed. Table 4-1 lists the latencies for the different levels of cache.

**Table 3-2: C1-Nano cache access latencies**

Scenario	Cycle count
L1 cache hit	2-4 cycles (2 is best case, 4 is normal case)
L2 cache hit	10-12 cycles (10 is best case, 11-12 is normal case)

## 3.12 Shared VPU

C1-Nano Core shares a VPU between all C1-Nano cores in a complex. The VPU is used to execute ASIMD, FP, Neon, and SVE instructions. Instructions being executed on VPU pipelines by one core may reduce performance of the instructions executed on the VPU by the other core.

## 3.13 AES encryption / decryption

C1-Nano Core implements instruction fusion for AES instructions (see section 3.2). It is recommended instructions pairs be interleaved in groups of three or more for the following: AESE, AESMC, AESD, AESIMC.

**Figure 3-5: Code snippet for AES instruction fusion.**

```
AESE  data0, key_reg
AESMC data0, data0
AESE  data1, key_reg
AESMC data1, data1
AESE  data2, key_reg
AESMC data2, data2...
```

# Proprietary Notice

This document is **NON-CONFIDENTIAL** and any use by you is subject to the terms of the agreement between you and Arm Limited (“Arm”) or the terms of the agreement between you and the party authorized by Arm to disclose this document to you.

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. **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, without obtaining Arm’s prior written consent, you will not use or permit others to use the information: **(i)** for the purposes of determining whether the subject matter of this document infringes any third party patents; **(ii)** for developing technology or products which avoid any of Arm’s intellectual property; **(iii)** as a reference for modifying existing patents or patent applications or creating any continuation, continuation in part, or extension of existing patents or patent applications; or **(iv)** for generating data for publication or disclosure to third parties, which compares the performance or functionality of the Arm technology described in this document with any other products created by you or a third party.

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 export laws and regulations to assure that this document or any portion thereof is not exported, directly or indirectly, in violation of such export laws. Use of the word “partner” in reference to Arm’s customers is not intended to create or refer to any partnership relationship with any other company. Arm may make changes to this document at any time and without notice.

If any of the provisions contained in these terms conflict with any of the provisions of any click through or signed written agreement covering this document with Arm, then the click through or signed written agreement prevails over and supersedes the conflicting provisions of these terms.

This document may be translated into other languages for convenience, and you agree that if there is any conflict between the English version of this document and any translation, the terms of the English version of this document shall prevail.

The validity, construction and performance of this notice shall be governed by English Law.

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

Arm Limited. Company 02557590 registered in England.  
110 Fulbourn Road, Cambridge, England CB1 9NJ.  
(PRE-1122-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 the 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

This product is r0p1, which indicates the revision status of the product described in this manual, where:

**r(value)** Identifies the major revision of the product, for example, r1.

**p(value)** Identifies the minor revision or modification status of the product, for example, p2.

## 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
0001-04	16 September 2025	Non-Confidential	Documentation update.
0001-03	10 September 2025	Non-Confidential	Second early access release for r0p1.
0001-02	7 May 2024	Confidential	First early access release for r0p1.
0000-01	26 February 2024	Confidential	First limited access release for r0p0.

---

## Change history

The first table is for the first release. Then, each table compares the new issue of the manual with the last released issue of the manual. Issue numbers match the revision history in Release Information.

**Table 3-4: Issue 0000-01**

Change	Location
First limited access release for r0p0	-

**Table 3-5: Issue 0001-02**

Change	Location
First early access release for r0p1	-
Editorial changes	Throughout document

**Table 3-6: Issue 0001-03**

Change	Location
Second early access release for r0p1	-
Updated product name to C1-Nano	Throughout document
Memory routines updated to include FEAT_MOPS	Section 3.9
Editorial changes	Throughout document

**Table 3-7: Issue 0001-04**

Change	Location
Correction to supported Arm Architecture version	Section 1

## Conventions

The following subsections describe conventions used in Arm documents.


### Glossary

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






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

### Typographical conventions

Arm documentation uses typographical conventions to convey specific meaning.

Convention	Use
<i>italic</i>	Citations.
<b>bold</b>	Terms in descriptive lists, where appropriate.
<code>monospace</code>	Text that you can enter at the keyboard, such as commands, file and program names, and source code.
<u><code>monospace underlined</code></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: <code>MRC p15, 0, &lt;Rd&gt;, &lt;CRn&gt;, &lt;CRm&gt;, &lt;Opcode_2&gt;</code>
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.

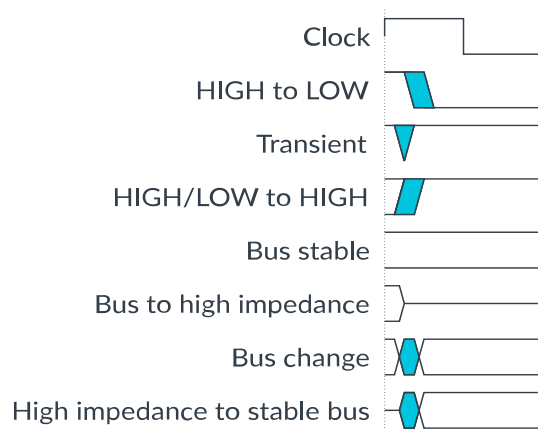


Convention	Use
 Warning	Your system requires the following. If you do not follow these requirements your system will not work.
 Danger	You are at risk of causing permanent damage to your system or your equipment, or of harming yourself.
 Note	This information is important and needs your attention.
 Tip	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.



## 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 relevant information.

Access to Arm documents depends on their confidentiality:

- Arm Non-Confidential documents are available at <https://developer.arm.com/documentation>. Each document link in the tables below provides direct access to the online version of the document.
- Arm Confidential documents are available to licensees only through the product package.

Arm product resources	Document ID	Confidentiality
<i>Arm® C1-Scalable Matrix Extension 2 Configuration and Integration Manual</i>	107832	Confidential
<i>Arm® C1-Scalable Matrix Extension 2 Technical Reference Manual</i>	107831	Non-Confidential
<i>Arm® CoreSight™ ELA-600 Embedded Logic Analyzer Configuration and Integration Manual</i>	101089	Confidential
<i>Arm® CoreSight™ ELA-600 Embedded Logic Analyzer Technical Reference Manual</i>	101088	Non-Confidential
<i>Arm® C1-Nano Core Cryptographic Extension Technical Reference Manual</i>	107755	Confidential
<i>Arm® C1-Nano Core iBEP User Guide</i>	PJDOC-1505342170-693760	Confidential
<i>Arm® C1-Nano Core Release Note</i>	109356	Confidential
<i>Arm® C1-Nano Core Configuration and Integration Manual</i>	107754	Confidential
<i>Arm® C1-Nano Core Technical Reference Manual</i>	107753	Non-Confidential
<i>Arm® C1-DynamiQ™ Shared Unit Configuration and Integration Manual</i>	107805	Confidential
<i>Arm® C1-DynamiQ™ Shared Unit Technical Reference Manual</i>	107804	Non-Confidential

Arm architecture and specifications	Document ID	Confidentiality
<i>Arm® Architecture Reference Manual for A-profile architecture profile</i>	DDI 0487	Non-Confidential
<i>AMBA® 5 CHI Architecture Specification</i>	IHI 0050	Non-Confidential
<i>Arm® CoreSight™ Architecture Specification v3.0</i>	IHI 0029	Non-Confidential

Non-Arm resources	Document ID	Organization
<i>IEEE, Standard for Access and Control of Instrumentation Embedded within a Semiconductor Device</i>	1687-2014	IEEE
<i>IEEE, Standard for Design and Verification of Low Power Integrated Circuits</i>	1801-2009	IEEE

Non-Arm resources	Document ID	Organization
IEEE, <i>Standard Test Access Port and Boundary Scan Architecture</i>	1149.1-2001	IEEE