Arm® Cortex®-A77

Revision: r1p1

Software Optimization Guide

Non-Confidential

Issue 4.0

PJDOC-466751330-8482

Arm® Cortex®-A77

Software Optimization Guide

Release information

Document history

Issue	Date	Confidentiality	Change
1.0	9 May 2018	Confidential	First release for r0p0
2.0	28 Sept 2018	Confidential	First release for r1p0
3.0	10 May 2019	Non-Confidential	First release for r1p1
4.0	16 Dec 2021	Non-Confidential	Second release for r1p1

Non-Confidential Proprietary Notice

Arm Limited. Company 02557590 registered in England.

110 Fulbourn Road, Cambridge, England CB1 9NJ.

LES-PRE-20349

Confidentiality Status

This document is Non-Confidential. The right to use, copy and disclose this document may be subject to license restrictions in accordance with the terms of the agreement entered into by Arm and the party that Arm delivered this document to.

Unrestricted Access is an Arm internal classification.

Product Status

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

Web Address

http://www.arm.com

Contents

1 Introduction

1.1 Product revision status

The *rmpn* identifier indicates the revision status of the product described in this book, for example, r*1*p*2*, where:

rm Identifies the major revision of the product, for example, r*1*.

pn Identifies the minor revision or modification status of the product, for example, p*2*.

1.2 Intended audience

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

1.3 Conventions

The following subsections describe conventions used in Arm documents.

1.3.1 Glossary

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

See the Arm® Glossary for more information.

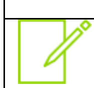
1.3.1.1 Terms and Abbreviations

This document uses the following terms and abbreviations.

Term	Meaning
ALU	Arithmetic and Logical Unit
ASIMD	Advanced SIMD
DSU	DynamIQ™ Shared Unit
MOP	Macro-OPeration
µOP	Micro-OPeration
SQRT	Square Root

Term	Meaning
T32	AArch32 Thumb® instruction set
FP	Floating-point

1.3.2 Typographical conventions

Convention	Use
italic	Introduces special terminology, denotes cross-references, and citations.
bold	Highlights interface elements, such as menu names. Denotes signal names. Also used for terms in descriptive lists, where appropriate.
`monospace`	Denotes text that you can enter at the keyboard, such as commands, file and program names, and source code.
`Monospace` **`bold`**	Denotes language keywords when used outside example code.
`monospace italic`	Denotes arguments to monospace text where the argument is to be replaced by a specific value.
`monospace` <u>`underline`</u>	Denotes a permitted abbreviation for a command or option. You can enter the underlined text instead of the full command or option name.
<and>	Encloses replaceable terms for assembler syntax where they appear in code or code fragments. For example: `MRC p15, 0, <Rd>, <CRn>, <CRm>, <Opcode_2>`
SMALL CAPITALS	Used in body text for a few terms that have specific technical meanings, that are defined in the Arm® Glossary. For example, IMPLEMENTATION DEFINED, IMPLEMENTATION SPECIFIC, UNKNOWN, and UNPREDICTABLE.
	Caution
	Warning
	Note

1.4 Additional reading

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

Table 1: Arm publications

Document name	Document ID	Licensee only Y/N
Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile	DDI 0487	N
Arm® Cortex®-A77 Core Technical Reference Manual	101111	N

1.5 Feedback

1.5.1 Feedback on this product

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

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

1.5.2 Feedback on content

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

- The title: *Arm® Cortex®-A77 Software Optimization Guide*.
- The number: PJDOC-466751330-8482.
- If applicable, the page number(s) to which your comments refer.
- A concise explanation of your comments.

Arm also welcomes general suggestions for additions and improvements.

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

2 About this document

This document contains a guide to the Cortex-A77 core micro-architecture with a view to aiding software optimization.

2.1 Scope

This document describes aspects of the Cortex-A77 core micro-architecture that influence software performance. Micro-architectural detail is limited to that which is useful for software optimization.

This documentation extends only to software visible behavior of the Cortex-A77 core and not to the hardware rationale behind the behavior.

2.2 Product overview

The Cortex-A77 core is a high-performance and low-power Arm product that implements the Armv8-A architecture with support for the Armv8.2-A extension, including the RAS extension, the Load acquire (LDAPR) instructions introduced in the Armv8.3-A extension, and the dot product instructions introduced in the Armv8.4-A extension.

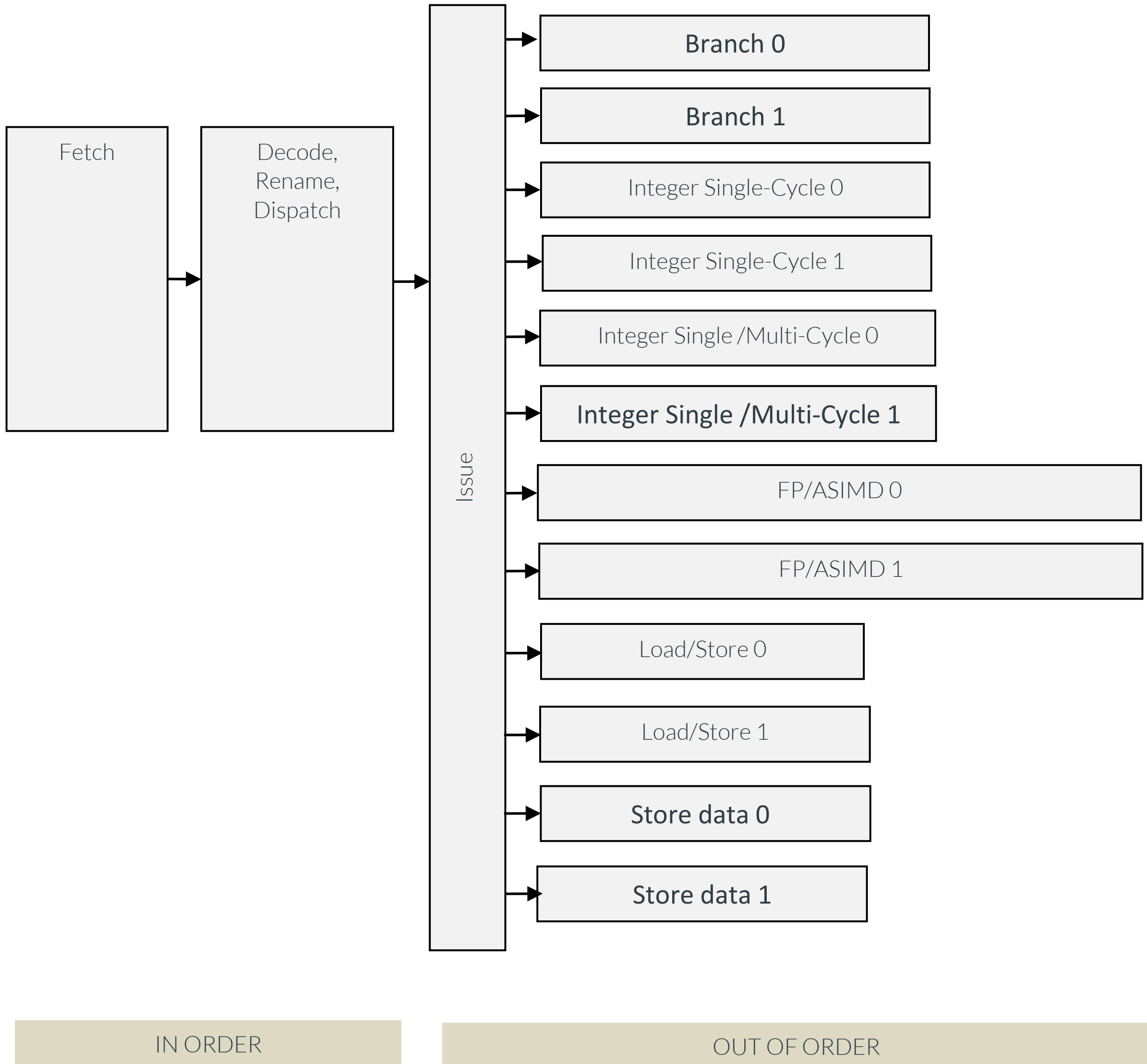
The Cortex-A77 core has a Level 1 (L1) memory system and a private, integrated Level 2 (L2) cache. It also includes a superscalar, variable-length, out-of-order pipeline.

The Cortex-A77 core is implemented inside the *DynamIQ™ Shared Unit* (DSU) cluster. For more information, see the *Arm® DynamIQ™ Shared Unit Technical Reference Manual*.

2.2.1 Pipeline overview

The following figure describes the high-level Cortex-A77 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 twelve issue pipelines. Each issue pipeline can accept one µOP per cycle.

Figure 1: Cortex-A77 core pipeline



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

Table 2: Cortex-A77 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, CRC and sum-of-absolute-differences µOPs
Load/Store 0/1	Load, Store address generation and special memory µOPs
Store data 0/1	Store data µOPs
FP/ASIMD-0	ASIMD ALU, ASIMD misc, ASIMD integer multiply, FP convert, FP misc, FP add, FP multiply, FP divide, FP sqrt, crypto µOPs, store data µOPs
FP/ASIMD-1	ASIMD ALU, ASIMD misc, FP misc, FP add, FP multiply, ASIMD shift µOPs, store data µOPs, crypto µOPs.

3 Instruction characteristics

3.1 Instruction tables

This chapter describes high-level performance characteristics for most Armv8.2-A A32, T32, and A64 instructions. A series of tables summarize the effective execution latency and throughput (instruction bandwidth per cycle), pipelines utilized, and special behaviors associated with each group of instructions. Utilized pipelines correspond to the execution pipelines described in chapter 2.

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

In the tables below, Execution Throughput is defined as the maximum throughput (in instructions per cycle) of the specified instruction group that can be achieved in the entirety of the Cortex-A77 microarchitecture.

3.2 Legend for reading the utilized pipelines

Table 3: Cortex-A77 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 0/1	L
Store data 0/1	D
FP/ASIMD 0/1	V
FP/ASIMD 0	V0
FP/ASIMD 1	V1

3.3 Branch instructions

Table 4: AArch64 Branch instructions

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

Table 5: AArch32 Branch instructions

Instruction Group	AArch32 Instructions	Exec Latency	Execution Throughput	Utilized Pipelines	Notes
Branch, immed	B	1	2	B	-
Branch, register	BX	1	2	B	-
Branch and link, immed	BL, BLX	1	2	B	-
Branch and link, register	BLX	1	2	B	-
Compare and branch	CBZ, CBNZ	1	2	B	-

3.4 Arithmetic and logical instructions

Table 6: AArch64 Arithmetic and logical instructions

Instruction Group	AArch64 Instructions	Exec Latency	Execution Throughput	Utilized Pipelines	Notes
Arithmetic, basic	ADD, ADC, SUB, SBC	1	4	I	-
Arithmetic, basic, flag set	ADDS, ADCS, SUBS, SBCS	1	3	I	-
Arithmetic, extend and shift	ADD{S}, SUB{S}	2	2	M	-

Instruction Group	AArch64 Instructions	Exec Latency	Execution Throughput	Utilized Pipelines	Notes
Arithmetic, LSL shift, shift <= 4	ADD, SUB	1	4	I	-
Arithmetic, flag set, LSL shift, shift <= 4	ADDS, SUBS	1	3	I	-
Arithmetic, LSR/ASR/ROR shift or LSL shift > 4	ADD{S}, SUB{S}	2	2	M	-
Conditional compare	CCMN, CCMP	1	3	I	-
Conditional select	CSEL, CSINC, CSINV, CSNEG	1	3	I	-
Logical, basic	AND{S}, BIC{S}, EON, EOR, ORN, ORR	1	3	I	-
Logical, shift, no flagset	AND, BIC, EON, EOR, ORN, ORR	1	4	I	-
Logical, shift, flagset	ANDS, BICS	2	2	M	-

Table 7: AArch32 Arithmetic and logical instructions

Instruction Group	AArch32 Instructions	Exec Latency	Execution Throughput	Utilized Pipelines	Notes
ALU, basic, no flagset	ADD, ADC, ADR, AND, BIC, EOR, ORN, ORR, RSB, RSC, SUB, SBC	1	4	I	-

Instruction Group	AArch32 Instructions	Exec Latency	Execution Throughput	Utilized Pipelines	Notes
ALU, basic, flagset	ADDS, ADCS, ANDS, BICS, CMN, CMP, EORS, ORNS, ORRS, RSBS, RSCS, SUBS, SBCS, TEQ, TST	1	3	I	-
ALU, basic, shift by register, conditional	(same as ALU basic, flagset and no flagset)	2	1	I, M0	-
ALU, basic, shift by register, unconditional, flagset	(same as ALU, basic, flagset)	2	1	M0	-
Arithmetic, shift by register, unconditional, no flagset	ADD, ADC, RSB, RSC, SUB, SBC	2	1	M0	-
Logical, shift by register, unconditional, no flagset	AND, BIC, EOR, ORN, ORR	1	1	M0	-
Arithmetic, LSL shift by immed, shift <= 4, unconditional, no flagset	ADD, ADC, RSB, RSC, SUB, SBC	1	4	I	-
Arithmetic, LSL shift by immed, shift <= 4, unconditional, flagset	ADDS, ADCS, RSBS, RSCS, SUBS, SBCS	1	3	I	-
Arithmetic, LSL shift by immed, shift <= 4, conditional	ADD{S}, ADC{S}, RSB{S}, RSC{S}, SUB{S}, SBC{S}	1	1	M0	-
Arithmetic, LSR/ASR/ROR shift by immed or LSL shift by immed > 4, unconditional	ADD{S}, ADC{S}, RSB{S}, RSC{S}, SUB{S}, SBC{S}	2	2	M	-

Instruction Group	AArch32 Instructions	Exec Latency	Execution Throughput	Utilized Pipelines	Notes
Arithmetic, LSR/ASR/ROR shift by immed or LSL shift by immed > 4, conditional	ADD{S}, ADC{S}, RSB{S}, RSC{S}, SUB{S}, SBC{S}	2	1	M0	-
Logical, shift by immed, no flagset, unconditional	AND, BIC, EOR, ORN, ORR	1	4	I	-
Logical, shift by immed, no flagset, conditional	AND, BIC, EOR, ORN, ORR	1	1	M0	-
Logical, shift by immed, flagset, unconditional	ANDS, BICS, EORS, ORNS, ORRS	2	2	M	-
Logical, shift by immed, flagset, conditional	ANDS, BICS, EORS, ORNS, ORRS	2	1	M0	-
Test/Compare, shift by immed	CMN, CMP, TEQ, TST	2	2	M	-
Branch forms		+1	2	+B	1

Branch forms are possible when the instruction destination register is the PC. For those cases, an additional branch μOP is required. This adds 1 cycle to the latency.

3.5 Move and shift instructions

Table 8: AArch32 Move and shift instructions

Instruction Group	AArch32 Instructions	Exec Latency	Execution Throughput	Utilized Pipelines	Notes
Move, basic	MOV, MOVW, MVN	1	4	I	-
Move, basic, flagset	MOVS, MVNS	1	3	I	

Instruction Group	AArch32 Instructions	Exec Latency	Execution Throughput	Utilized Pipelines	Notes
Move, shift by immed, no flagset	ASR, LSL, LSR, ROR, RRX, MVN	1	4	I	-
Move, shift by immed, flagset	ASRS, LSLS, LSRS, RORS, RRXS, MVNS	2	2	M	-
Move, shift by register, no flagset, unconditional	ASR, LSL, LSR, ROR, RRX, MVN	1	4	I	-
Move, shift by register, no flagset, conditional	ASR, LSL, LSR, ROR, RRX, MVN	2	2	I	-
Move, shift by register, flagset	ASRS, LSLS, LSRS, RORS, RRXS, MVNS	2	1	M0	-
Move, top	MOVT	1	4	I	-
Move, branch forms		+1	2	+B	-

3.6 Divide and multiply instructions

Table 9: 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-form	MADD, MSUB	2(1)	1	M0	2
Multiply accumulate, X-form	MADD, MSUB	2(1)	1	M0	2
Multiply accumulate long	SMADDL, SMSUBL, UMADDL, UMSUBL	2(1)	1	M0	2
Multiply high	SMULH, UMULH	3	1	M0	2

Table 10: AArch32 Divide and multiply instructions

Instruction Group	AArch32 Instructions	Exec Latency	Execution Throughput	Utilized Pipelines	Notes
Divide	SDIV, UDIV	5 to 12	1/12 to 1/5	M0	1
Multiply	MUL, SMULBB, SMULBT, SMULTB, SMULTT, SMULWB, SMULWT, SMMUL{R}, SMUAD{X}, SMUSD{X}	2	1	M0	-
Multiply accumulate, conditional	MLA, MLS, SMLABB, SMLABT, SMLATB, SMLATT, SMLAWB, SMLAWT, SMLAD{X}, SMLSD{X}, SMMLA{R}, SMMLS{R}	3	1	M0, I	-
Multiply accumulate, unconditional	MLA, MLS, SMLABB, SMLABT, SMLATB, SMLATT, SMLAWB, SMLAWT, SMLAD{X}, SMLSD{X}, SMMLA{R}, SMMLS{R}	2(1)	1	M0	2
Multiply accumulate accumulate long, conditional	UMAAL	4	1	I, M0	-
Multiply accumulate accumulate long, unconditional	UMAAL	3	1	I, M0	-

Instruction Group	AArch32 Instructions	Exec Latency	Execution Throughput	Utilized Pipelines	Notes
Multiply accumulate long, no flagset	SMLAL, SMLALBB, SMLALBT, SMLALTB, SMLALTT, SMLALD{X}, SMLSLD{X}, UMLAL	2	1	M0, I	-
Multiply accumulate long, flagset	SMLAL, SMLALBB, SMLALBT, SMLALTB, SMLALTT, SMLALD{X}, SMLSLD{X}, UMLAL	2	1	M0, I, M	-
Multiply long, unconditional, no flagset	SMULL, UMULL	2	1	M0	-
Multiply long, unconditional, flagset	SMULLS, UMULLS	3	1	M0, I	-
Multiply long, conditional	SMULL{S}, UMULL{S}	3	1	M0, I	-

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. Multiplies that set the condition flags require an additional integer µOP.

3.7 Saturating and parallel arithmetic instructions

Table 11: AArch32 Saturating and parallel arithmetic instructions

Instruction Group	AArch32 Instructions	Exec Latency	Execution Throughput	Utilized Pipelines	Notes
Parallel arith, unconditional	SADD16, SADD8, SSUB16, SSUB8, UADD16, UADD8, USUB16, USUB8	2	1	M	-
Parallel arith, conditional	SADD16, SADD8, SSUB16, SSUB8, UADD16, UADD8, USUB16, USUB8	2(4)	1	M0, I	1
Parallel arith with exchange, unconditional	SASX, SSAX, UASX, USAX	3	2	I, M	-
Parallel arith with exchange, conditional	SASX, SSAX, UASX, USAX	3(5)	1	I, M0	1
Parallel halving arith, unconditional	SHADD16, SHADD8, SHSUB16, SHSUB8, UHADD16, UHADD8, UHSUB16, UHSUB8	2	2	M	-
Parallel halving arith, conditional	SHADD16, SHADD8, SHSUB16, SHSUB8, UHADD16, UHADD8, UHSUB16, UHSUB8	2	1	M0	-

Instruction Group	AArch32 Instructions	Exec Latency	Execution Throughput	Utilized Pipelines	Notes
Parallel halving arith with exchange	SHASX, SHSAX, UHASX, UHSAX	3	1	I, M0	-
Parallel saturating arith, unconditional	QADD16, QADD8, QSUB16, QSUB8, UQADD16, UQADD8, UQSUB16, UQSUB8	2	2	M	-
Parallel saturating arith, conditional	QADD16, QADD8, QSUB16, QSUB8, UQADD16, UQADD8, UQSUB16, UQSUB8	2	1	M0	-
Parallel saturating arith with exchange, unconditional	QASX, QSAX, UQASX, UQSAX	3	2	I, M	-
Parallel saturating arith with exchange, conditional	QASX, QSAX, UQASX, UQSAX	3(5)	1	I, M0	-
Saturate, unconditional	SSAT, SSAT16, USAT, USAT16	2	2	M	-
Saturate, conditional	SSAT, SSAT16, USAT, USAT16	2	1	M0	-
Saturating arith, unconditional	QADD, QSUB	2	2	M	-
Saturating arith, conditional	QADD, QSUB	2	1	M0	-
Saturating doubling arith, unconditional	QDADD, QDSUB	4	1	M, M	-

Instruction Group	AArch32 Instructions	Exec Latency	Execution Throughput	Utilized Pipelines	Notes
Saturating doubling arith conditional	QDADD, QDSUB	4	1	M, M0	-

Branch forms are possible when the instruction destination register is the PC. For those cases, an additional branch µOP is required. This adds 1 cycle to the latency.

3.8 Miscellaneous data-processing instructions

Table 12: AArch64 Miscellaneous data-processing instructions

Instruction Group	AArch64 Instructions	Exec Latency	Execution Throughput	Utilized Pipelines	Notes
Address generation	ADR, ADRP	1	4	I	-
Bitfield extract, one reg	EXTR	1	4	I	-
Bitfield extract, two regs	EXTR	3	2	I, M	-
Bitfield move, basic	SBFM, UBFM	1	4	I	-
Bitfield move, insert	BFM	2	2	M	-
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	-

Table 13: AArch32 Miscellaneous data-processing instructions

Instruction Group	AArch32 Instructions	Exec Latency	Execution Throughput	Utilized Pipelines	Notes
Bit field extract	SBFX, UBFX	1	4	I	-
Bit field insert/clear, unconditional	BFI, BFC	2	2	M	-

Instruction Group	AArch32 Instructions	Exec Latency	Execution Throughput	Utilized Pipelines	Notes
Bit field insert/clear, conditional	BFI, BFC	2	1	M0	-
Count leading zeros	CLZ	1	4	I	-
Pack halfword, unconditional	PKH	2	2	M	-
Pack halfword, conditional	PKH	2	1	M0	-
Reverse bits/bytes	RBIT, REV, REV16, REVSH	1	4	I	-
Select bytes, unconditional	SEL	1	4	I	-
Select bytes, conditional	SEL	2	2	I	-
Sign/zero extend, normal	SXTB, SXTH, UXTB, UXTH	1	4	I	-
Sign/zero extend, parallel, unconditional	SXTB16, UXTB16	2	2	M	-
Sign/zero extend, parallel, conditional	SXTB16, UXTB16	2	1	M0	-
Sign/zero extend and add, normal, unconditional	SXTAB, SXTAH, UXTAB, UXTAH	2	2	M	-
Sign/zero extend and add, normal, conditional	SXTAB, SXTAH, UXTAB, UXTAH	2	1	M0	-
Sign/zero extend and add, parallel, unconditional	SXTAB16, UXTAB16	4	1/2	M	-
Sign/zero extend and add, parallel, conditional	SXTAB16, UXTAB16	4	1/2	M, M0	-
Sum of absolute differences, unconditional	USAD8, USADA8	2	1	M0	-
Sum of absolute differences, conditional	USAD8, USADA8	2	1	M0, I	-

3.9 Load instructions

Table 14: AArch64 Load instructions

Instruction Group	AArch64 Instructions	Exec Latency	Execution Throughput	Utilized Pipelines	Notes
Load register, literal	LDR, LDRSW, PRFM	4	2	L	-
Load register, unscaled immed	LDUR, LDURB, LDURH, LDURSB, LDURSH, LDURSW, PRFUM	4	2	L	-
Load register, immed post-index	LDR, LDRB, LDRH, LDRSB, LDRSH, LDRSW	4	2	L, I	-
Load register, immed pre-index	LDR, LDRB, LDRH, LDRSB, LDRSH, LDRSW	4	2	L, I	-
Load register, immed unprivileged	LDTR, LDTRB, LDTRH, LDTRSB, LDTRSH, LDTRSW	4	2	L	-
Load register, unsigned immed	LDR, LDRB, LDRH, LDRSB, LDRSH, LDRSW, PRFM	4	2	L	-
Load register, register offset, basic	LDR, LDRB, LDRH, LDRSB, LDRSH, LDRSW, PRFM	4	2	L	-

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

Table 15: AArch32 Load instructions

Instruction Group	AArch32 Instructions	Exec Latency	Execution Throughput	Utilized Pipelines	Notes
Load, immed offset	LDR{T}, LDRB{T}, LDRD, LDRH{T}, LDRSB{T}, LDRSH{T}	4	2	L	1,2
Load, register offset, plus	LDR, LDRB, LDRD, LDRH, LDRSB, LDRSH	4	2	L	1.2
Load, register offset, minus	LDR, LDRB, LDRD, LDRH, LDRSB, LDRSH	5	2	I, L	1,2
Load, scaled register offset, plus, LSL2	LDR, LDRB	4	2	L	1
Load, scaled register offset, other	LDR, LDRB, LDRH, LDRSB, LDRSH	5	2	I, L	1
Load, immed pre-indexed	LDR, LDRB, LDRD, LDRH, LDRSB, LDRSH	4	2	L, I	1,2
Load, register pre-indexed, shift Rm, plus and minus	LDR, LDRB, LDRH, LDRSB, LDRSH	5	2	I, L, M	3
Load, register pre-indexed	LDRD	4	2	L, I	-
Load, register pre-indexed, cond	LDRD	5	1 1/2	L, I	-
Load, scaled register pre-indexed, plus, LSL2	LDR, LDRB	4	2	L, I	1
Load, scaled register pre-indexed, unshifted	LDR, LDRB	4	2	L, I	-

Instruction Group	AArch32 Instructions	Exec Latency	Execution Throughput	Utilized Pipelines	Notes
Load, immed post-indexed	LDR{T}, LDRB{T}, LDRD, LDRH{T}, LDRSB{T}, LDRSH{T}	4	2	L, I	1,2
Load, register post-indexed	LDR, LDRB, LDRH{T}, LDRSB{T}, LDRSH{T}	5	2	I, L	-
Load, register post-indexed	LDRD	4	2	L, I	-
Load, register post-indexed	LDRT, LDRBT	5	2	I, L	-
Load, scaled register post-indexed	LDR, LDRB	4	2	L, M	3
Load, scaled register post-indexed	LDRT, LDRBT	4	2	L, M	3
Preload, immed offset	PLD, PLDW	4	2	L	-
Preload, register offset, plus	PLD, PLDW	4	2	L	-
Preload, register offset, minus	PLD, PLDW	5	2	I, L	-
Preload, scaled register offset, plus LSL2	PLD, PLDW	5	2	I, L	-
Preload, scaled register offset, other	PLD, PLDW	5	2	I, L	-
Load multiple, no writeback, base reg not in list	LDMIA, LDMIB, LDMDA, LDMDB	N	2/R	L	1, 4, 5
Load multiple, no writeback, base reg in list	LDMIA, LDMIB, LDMDA, LDMDB	1+ N	2/R	I, L	1, 4, 5
Load multiple, writeback	LDMIA, LDMIB, LDMDA, LDMDB, POP	1+N	2/R	L, I	1, 4, 5

Instruction Group	AArch32 Instructions	Exec Latency	Execution Throughput	Utilized Pipelines	Notes
(Load, all branch forms)	-	+1	-	+ B	6

1. Condition loads have an extra µOP which goes down pipeline I and have 1 cycle extra latency compared to their unconditional counterparts.
2. The throughput of conditional LDRD is 1 as compared to a throughput of 2 for unconditional LDRD.
3. The address update op for addressing forms which use reg scaled reg, or reg extend goes down pipeline 'I' if the shift is LSL where the shift value is less than or equal to 4.
4. N is floor [(num_reg+3)/4].
5. R is floor [(num_reg +1)/2].
6. Branch forms are possible when the instruction destination register is the PC. For those cases, an additional branch µOP is required. This adds 1 cycle to the latency.

3.10 Store instructions

The following tables 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 16: AArch64 Store instructions

Instruction Group	AArch64 Instructions	Exec Latency	Execution Throughput	Utilized Pipelines	Notes
Store register, unscaled immed	STUR, STURB, STURH	1	2	L, D	-
Store register, immed post-index	STR, STRB, STRH	1	2	L, D	-
Store register, immed pre-index	STR, STRB, STRH	1	2	L, D	-
Store register, immed unprivileged	STTR, STTRB, STTRH	1	2	L, D	-
Store register, unsigned immed	STR, STRB, STRH	1	2	L, D	-
Store register, register offset, basic	STR, STRB, STRH	1	2	L, D	-

Instruction Group	AArch64 Instructions	Exec Latency	Execution Throughput	Utilized Pipelines	Notes
Store register, register offset, scaled by 4/8	STR	1	2	L, D	-
Store register, register offset, scaled by 2	STRH	2	3/2	I, L, D	-
Store register, register offset, extend	STR, STRB, STRH	1	2	L, D	-
Store register, register offset, extend, scale by 4/8	STR	1	2	L, D	-
Store register, register offset, extend, scale by 1	STRH	2	3/2	I, L, D	-
Store pair, immed offset, W-form	STP, STNP	1	2	L, D	-
Store pair, immed offset, X-form	STP, STNP	1	1	L, D	-
Store pair, immed post-index, W-form	STP	1	1	L, D	-
Store pair, immed post-index, X-form	STP	1	1	L, D	-
Store pair, immed pre-index, W-form	STP	1	1	L, D	-
Store pair, immed pre-index, X-form	STP	1	1	L, D	-

Table 17: AArch32 Store instructions

Instruction Group	AArch32 Instructions	Exec Latency	Execution Throughput	Utilized Pipelines	Notes
Store, immed offset	STR{T}, STRB{T}, STRD, STRH{T}	1	2	L, D	-
Store, register offset, plus	STR, STRB, STRD, STRH	1	2	L, D	-
Store, register offset, minus	STR, STRB, STRD, STRH	1	2	L, D	-

Instruction Group	AArch32 Instructions	Exec Latency	Execution Throughput	Utilized Pipelines	Notes
Store, register offset, no shift, plus	STR, STRB	1	2	L, D	-
Store, scaled register offset, plus LSL2	STR, STRB	1	2	L, D	-
Store, scaled register offset, other	STR, STRB	2	3/2	I, L, D	-
Store, scaled register offset, minus	STR, STRB	2	3/2	I, L, D	-
Store, immed pre-indexed	STR, STRB, STRD, STRH	1	3/2	I, L, D	-
Store, register pre-indexed, plus, no shift	STR, STRB, STRD, STRH	1	3/2	L, D	-
Store, register pre-indexed, minus	STR, STRB, STRD, STRH	2	1	I, L, D	-
Store, scaled register pre-indexed, plus LSL2	STR, STRB	1	3/2	L, D	-
Store, scaled register pre-indexed, other	STR, STRB	2	1	I, L, D, M	1
Store, immed post-indexed	STR{T}, STRB{T}, STRD, STRH{T}	1	3/2	L, D	-
Store, register post-indexed	STRH{T}, STRD	1	3/2	L, D	-
Store, register post-indexed	STR{T}, STRB{T}	1	3/2	L, D	-
Store, scaled register post-indexed	STR{T}, STRB{T}	1	3/2	L, D	-
Store multiple, no writeback	STMIA, STMIB, STMDA, STMDB	N	1/N	L, D	2
Store multiple, writeback	STMIA, STMIB, STMDA, STMDB, PUSH	N	1/N	L, D	2

1. The address update op for addressing forms which use reg scaled reg, or reg extend goes down pipeline 'I' if the shift is LSL where the shift value is less than or equal to 4.
2. For store multiple instructions, N=floor((num_regs+3)/4).

3.11 FP data processing instructions

Table 18: AArch64 FP data processing instructions

Instruction Group	AArch64 Instructions	Exec Latency	Execution Throughput	Utilized Pipelines	Notes
FP absolute value	FABS	2	2	V	-
FP arithmetic	FADD, FSUB	2	2	V	-
FP compare	FCCMP{E}, FCMP{E}	2	1	V0	-
FP divide, H-form	FDIV	7	4/7	V0	1
FP divide, S-form	FDIV	7 to 10	4/9 to 4/7	V0	1
FP divide, D-form	FDIV	7 to 15	1/7 to 2/7	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	3	1	V	-
FP select	FCSEL	2	2	V	-
FP square root, H-form	FSQRT	7	4/7	V0	1
FP square root, S-form	FSQRT	7 to 10	4/9 to 4/7	V0	1
FP square root, D-form	FSQRT	7 to 17	1/8 to 2/7	V0	1

Table 19: AArch32 FP data processing instructions

Instruction Group	AArch32 Instructions	Exec Latency	Execution Throughput	Utilized Pipelines	Notes
VFP absolute value	VABS	2	2	V	-
VFP arith	VADD, VSUB	2	2	V	-
VFP compare, unconditional	VCMP, VCMPE	2	1	V0	-
VFP compare, conditional	VCMP, VCMPE	4	1	V, V0	-
VFP convert	VCVT{R}, VCVTB, VCVTT, VCVTA, VCVTM, VCVTN, VCVTP	3	1	V0	-
VFP divide, H-form	VDIV	7	4/7	V0	1
VFP divide, S-form	VDIV	7 to 10	4/9 to 4/7	V0	1
VFP divide, D-form	VDIV	7 to 15	1/7 to 2/7	V0	1
VFP max/min	VMAXNM, VMINNM	2	2	V	-
VFP multiply	VMUL, VNMUL	3	2	V	2
VFP multiply accumulate (chained)	VMLA, VMLS, VNMLA, VNMLS	5 (2)	2	V	3
VFP multiply accumulate (fused)	VFMA, VFMS, VFNMA, VFNMS	4 (2)	2	V	3
VFP negate	VNEG	2	2	V	-

Instruction Group	AArch32 Instructions	Exec Latency	Execution Throughput	Utilized Pipelines	Notes
VFP round to integral	VRINTA, VRINTM, VRINTN, VRINTP, VRINTR, VRINTX, VRINTZ	3	1	V0	-
VFP select	VSELEQ, VSELGE, VSELGT, VSELVS	2	2	V	-
VFP square root, H-form	VSQRT	7	4/7	V0	1
VFP square root, S-form	VSQRT	7 to 10	4/9 to 4/7	V0	1
VFP square root, D-form	VSQRT	7 to 17	1/8 to 2/7	V0	1

1. FP divide and square root operations are performed using an iterative algorithm and block subsequent similar operations to the same pipeline until complete.
2. FP multiply-accumulate pipelines support late forwarding of the result from FP multiply µOPs to the accumulate operands of an FP multiply-accumulate µOP. The latter can potentially be issued 1 cycle after the FP multiply µOP has been issued.
3. FP multiply-accumulate pipelines support late-forwarding of accumulate operands from similar µOPs, allowing a typical sequence of multiply-accumulate µOPs to issue one every N cycles (accumulate latency N shown in parentheses).

3.12 FP miscellaneous instructions

Table 20: AArch64 FP miscellaneous instructions

Instruction Group	AArch64 Instructions	Exec Latency	Execution Throughput	Utilized Pipelines	Notes
FP convert, from vec to vec reg	FCVT, FCVTXN	3	1	V0	-
FP convert, from gen to vec reg	SCVTF, UCVTF	6	1	M0, V0	-

Instruction Group	AArch64 Instructions	Exec Latency	Execution Throughput	Utilized Pipelines	Notes
FP convert, from vec to gen reg	FCVTAS, FCVTAU, FCVTMS, FCVTMU, FCVTNS, FCVTNU, FCVTPS, FCVTPU, FCVTZS, FCVTZU	4	1	V0, V1	-
FP move, immed	FMOV	2	2	V	-
FP move, register	FMOV	2	2	V	-
FP transfer, from gen to vec reg	FMOV	3	1	M0	-
FP transfer, from vec to gen reg	FMOV	2	1	V1	-

Table 21: AArch32 FP miscellaneous instructions

Instruction Group	AArch32 Instructions	Exec Latency	Execution Throughput	Utilized Pipelines	Notes
VFP move, immed	VMOV	2	2	V	-
VFP move, register	VMOV	2	2	V	-
VFP move, insert	VINS	2	2	V	-
VFP move, extraction	VMOVX	2	2	V	-
VFP transfer, core to vfp, single reg to S-reg, cond	VMOV	5	1	M0, V	-
VFP transfer, core to vfp, single reg to S-reg, uncond	VMOV	3	1	M0	-
VFP transfer, core to vfp, single reg to upper/lower half of D-reg	VMOV	5	1	M0, V	-
VFP transfer, core to vfp, 2 regs to 2 S-regs, cond	VMOV	6	1/2	M0, V	-

Instruction Group	AArch32 Instructions	Exec Latency	Execution Throughput	Utilized Pipelines	Notes
VFP transfer, core to vfp, 2 regs to 2 S-regs, uncond	VMOV	4	1/2	M0	-
VFP transfer, core to vfp, 2 regs to D-reg, cond	VMOV	5	1	M0, V	-
VFP transfer, core to vfp, 2 regs to D-reg, uncond	VMOV	3	1	M0	-
VFP transfer, vfp S-reg or upper/lower half of vfp D-reg to core reg, cond	VMOV	3	1	V1, I	-
VFP transfer, vfp S-reg or upper/lower half of vfp D-reg to core reg, uncond	VMOV	2	1	V1	-
VFP transfer, vfp 2 S-regs or D-reg to 2 core regs, cond	VMOV	3	1	V1, I	-
VFP transfer, vfp 2 S-regs or D-reg to 2 core regs, uncond	VMOV	2	1	V1	-

3.13 FP load instructions

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

Table 22: AArch64 FP load instructions

Instruction Group	AArch64 Instructions	Exec Latency	Execution Throughput	Utilized Pipelines	Notes
Load vector reg, literal, S/D/Q forms	LDR	-	2	L	-
Load vector reg, unscaled immed	LDUR	5	2	L	-
Load vector reg, immed post-index	LDR	5	2	L, I	-
Load vector reg, immed pre-index	LDR	5	2	L, I	-

Instruction Group	AArch64 Instructions	Exec Latency	Execution Throughput	Utilized Pipelines	Notes
Load vector reg, unsigned immed	LDR	5	2	L, I	-
Load vector reg, register offset, basic	LDR	5	2	L, I	-
Load vector reg, register offset, scale, S/D-form	LDR	5	2	L, I	-
Load vector reg, register offset, scale, H/Q-form	LDR	6	2	I, L	-
Load vector reg, register offset, extend	LDR	5	2	L, I	-
Load vector reg, register offset, extend, scale, S/D-form	LDR	5	2	L, I	-
Load vector reg, register offset, extend, scale, H/Q-form	LDR	6	2	I, L	-
Load vector pair, immed offset, S/D-form	LDP, LDNP	5	1	L, I	-
Load vector pair, immed offset, Q-form	LDP, LDNP	7	1	L	-
Load vector pair, immed post-index, S/D-form	LDP	5	1	I, L	-
Load vector pair, immed post-index, Q-form	LDP	7	1	L, I	-
Load vector pair, immed pre-index, S/D-form	LDP	5	1	I, L	-
Load vector pair, immed pre-index, Q-form	LDP	7	1	L, I	-

Table 23: AArch32 FP load instructions

Instruction Group	AArch32 Instructions	Exec Latency	Execution Throughput	Utilized Pipelines	Notes
FP load, register	VLDR	4	2	L	1
FP load multiple, S form	VLDMIA, VLDMDB, VPOP	N	2/R	L	1, 2, 3

Instruction Group	AArch32 Instructions	Exec Latency	Execution Throughput	Utilized Pipelines	Notes
FP load multiple, D form	VLDMIA, VLDMDB, VPOP	N + 2	1/R	L, V	1, 2, 3
(FP load, writeback forms)	-	(1)	-	+ I	4

1. Condition loads have an extra µOP which goes down pipeline V and have 2 cycle extra latency compared to their unconditional counterparts.
2. N is floor[(num_reg+3)/4].
3. R is floor[(num_reg+1)/2].
4. Writeback forms of load instructions require an extra µOP to update the base address. This update is typically performed in parallel with or prior to the load µOP (update latency shown in parentheses).

3.14 FP store instructions

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

Table 24: 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	L, V	-
Store vector reg, unscaled immed, Q-form	STUR	2	1	L, V	-
Store vector reg, immed post-index, B/H/S/D-form	STR	2	2	L, V	-
Store vector reg, immed post-index, Q-form	STR	2	1	L, V	-
Store vector reg, immed pre-index, B/H/S/D-form	STR	2	2	L, V	-
Store vector reg, immed pre-index, Q-form	STR	2	1	L, V	-

Instruction Group	AArch64 Instructions	Exec Latency	Execution Throughput	Utilized Pipelines	Notes
Store vector reg, unsigned immed, B/H/S/D-form	STR	2	2	L, V	-
Store vector reg, unsigned immed, Q-form	STR	2	1	L, V	-
Store vector reg, register offset, basic, B/H/S/D-form	STR	2	2	L, V	-
Store vector reg, register offset, basic, Q-form	STR	2	1	L, V	-
Store vector reg, register offset, scale, H-form	STR	2	2	I, L, V	-
Store vector reg, register offset, scale, S/D-form	STR	2	2	L, V	-
Store vector reg, register offset, scale, Q-form	STR	2	1	I, L, V	-
Store vector reg, register offset, extend, B/H/S/D-form	STR	2	2	L, V	-
Store vector reg, register offset, extend, Q-form	STR	2	1	L, V	-
Store vector reg, register offset, extend, scale, H-form	STR	2	2	I, L, V	-
Store vector reg, register offset, extend, scale, S/D-form	STR	2	2	L, V	-
Store vector reg, register offset, extend, scale, Q-form	STR	2	1	I, L, V	-
Store vector pair, immed offset, S-form	STP, STNP	2	2	L, V	-
Store vector pair, immed offset, D-form	STP, STNP	2	1	L, V	-

Instruction Group	AArch64 Instructions	Exec Latency	Execution Throughput	Utilized Pipelines	Notes
Store vector pair, immed offset, Q-form	STP, STNP	3	1/2	L, V	-
Store vector pair, immed post-index, S-form	STP	2	1	L, V	-
Store vector pair, immed post-index, D-form	STP	2	1	L, V	-
Store vector pair, immed post-index, Q-form	STP	3	1	L, V	-
Store vector pair, immed pre-index, S-form	STP	2	1	L, V	-
Store vector pair, immed pre-index, D-form	STP	2	1	L, V	-
Store vector pair, immed pre-index, Q-form	STP	3	1/2	L, V	-

Table 25: AArch32 FP store instructions

Instruction Group	AArch32 Instructions	Exec Latency	Execution Throughput	Utilized Pipelines	Notes
FP store, immed offset	VSTR	2	2	L V	-
FP store multiple, S-form	VSTMIA, VSTMDB, VPUSH	N+1	2/R	L, V	1, 3
FP store multiple, D-form	VSTMIA, VSTMDB, VPUSH	P + 1	1/R	L, V	2, 3
(FP store, writeback forms)	-	(1)	-	+ I	4

1. For store multiple instructions, N=floor((num_regs+3)/4).
2. For store multiple instructions, P=floor((num_regs+1)/2).
3. R=floor[(num_regs + 1)/2].
4. Writeback forms of store instructions require an extra µOP to update the base address. This update is typically performed in parallel with or prior to the store µOP (update latency shown in parentheses).

3.15 ASIMD integer instructions

Table 26: 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	-

Instruction Group	AArch64 Instructions	Exec Latency	Execution Throughput	Utilized Pipelines	Notes
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	2	2	V	-
ASIMD logical	AND, BIC, EOR, MOV, MVN, ORN, ORR, NOT	2	2	V	-
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	-

Instruction Group	AArch64 Instructions	Exec Latency	Execution Throughput	Utilized Pipelines	Notes
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, D-form	MUL, SQDMULH, SQRDMULH	4	1	V0	-
ASIMD multiply, Q-form	MUL, SQDMULH, SQRDMULH	5	1/2	V0	-
ASIMD multiply accumulate, D-form	MLA, MLS	4(1)	1	V0	1
ASIMD multiply accumulate, Q-form	MLA, MLS	5(2)	1/2	V0	1
ASIMD multiply accumulate high, D-form	SQRDMLAH, SQRDMLSH	4	1	V0	-
ASIMD multiply accumulate high, Q-form	SQRDMLAH, SQRDMLSH	5	1/2	V0	-
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	1	V0	-
ASIMD multiply/multiply long (8x8) polynomial, D-form	PMUL, PMULL(2)	3	1	V0	3
ASIMD multiply/multiply long (8x8) polynomial, Q-form	PMUL, PMULL(2)	4	1/2	V0	3

Instruction Group	AArch64 Instructions	Exec Latency	Execution Throughput	Utilized Pipelines	Notes
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
ASIMD shift accumulate	SSRA, SRSRA, USRA, URSRA	4(1)	1	V1	2
ASIMD shift by immed, basic	SHL, SHLL(2), SHRN(2), SSHLL(2), SSHR, SXTL(2), USHLL(2), USHR, UXTL(2)	2	1	V1	-
ASIMD shift by immed and insert, basic	SLI, SRI	2	1	V1	-
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	-
ASIMD shift by register, basic	SSHL, USHL	2	1	V1	-
ASIMD shift by register, complex	SRSHL, SQRSHL, SQSHL, URSHL, UQRSHL, UQSHL	4	1	V1	-

Table 27: AArch32 ASIMD integer instructions

Instruction Group	AArch32 Instructions	Exec Latency	Execution Throughput	Utilized Pipelines	Notes
ASIMD absolute diff	VABD	2	2	V	-
ASIMD absolute diff accum	VABA	4(1)	1	V1	2
ASIMD absolute diff accum long	VABAL	4(1)	1	V1	2
ASIMD absolute diff long	VABDL	2	2	V	-
ASIMD arith, basic	VADD, VADDL, VADDW, VNEG, VSUB, VSUBL, VSUBW	2	2	V	-
ASIMD arith, complex	VABS, VADDHN, VHADD, VHSUB, VQABS, VQADD, VQNEG, VQSUB, VRADDHN, VRHADD, VRSUBHN, VSUBHN	2	2	V	-
ASIMD arith, dot product	VSDOT, VUDOT	2	2	V	-
ASIMD arith, pair-wise	VPADD, VPADDL	2	2	V	-
ASIMD compare	VCEQ, VCGE, VCGT, VCLE, VTST	2	2	V	-

Instruction Group	AArch32 Instructions	Exec Latency	Execution Throughput	Utilized Pipelines	Notes
ASIMD logical	VAND, VBIC, VMVN, VORR, VORN, VEOR	2	2	V	-
ASIMD max/min	VMAX, VMIN, VPMAX, VPMIN	2	2	V	-
ASIMD multiply, D-form	VMUL, VQDMULH, VQRDMULH	4	1	V0	-
ASIMD multiply, Q-form	VMUL, VQDMULH, VQRDMULH	5	1/2	V0	-
ASIMD multiply accumulate, D-form	VMLA, VMLS	4(1)	1	V0	1
ASIMD multiply accumulate, Q-form	VMLA, VMLS	5(2)	1/2	V0	1
ASIMD multiply accumulate high, D-form	VQRDMLAH, VQRDMLSH	4	1	V0	-
ASIMD multiply accumulate high, Q-form	VQRDMLAH, VQRDMLSH	5	1/2	V0	-
ASIMD multiply accumulate long	VMLAL, VMLSL	4(1)	1	V0	1
ASIMD multiply accumulate saturating long	VQDMLAL, VQDMLSL	4	1	V0	-
ASIMD multiply/multiply long (8x8) polynomial, D-form	VMUL (.P8), VMULL (.P8)	3	1	V0	-
ASIMD multiply (8x8) polynomial, Q-form	VMUL (.P8)	4	1/2	V0	-

Instruction Group	AArch32 Instructions	Exec Latency	Execution Throughput	Utilized Pipelines	Notes
ASIMD multiply long	VMULL (.S, .I), VQDMULL	4	1	V0	-
ASIMD pairwise add and accumulate	VPADAL	4(1)	1	V1	1
ASIMD shift accumulate	VSRA, VRSRA	4(1)	1	V1	1
ASIMD shift by immed, basic	VMOVL, VSHL, VSHLL, VSHR, VSHRN	2	1	V1	-
ASIMD shift by immed and insert, basic	VSLI, VSRI	2	1	V1	-
ASIMD shift by immed, complex	VQRSHRN, VQRSHRUN, VQSHL{U}, VQSHRN, VQSHRUN, VRSHR, VRSHRN	4	1	V1	-
ASIMD shift by register, basic	VSHL	2	1	V1	-
ASIMD shift by register, complex	VQRSHL, VQSHL, VRSHL	4	1	V1	-

1. Multiply-accumulate pipelines support late-forwarding of accumulate operands from similar µOPs, allowing a typical sequence of integer multiply-accumulate µOPs to issue one every cycle or one every other cycle (accumulate latency shown in parentheses).
2. Other accumulate pipelines also support late-forwarding of accumulate operands from similar µOPs, allowing a typical sequence of such µOPs to issue one every cycle (accumulate latency shown in parentheses).
3. This category includes instructions of the form "PMULL Vd.8H, Vn.8B, Vm.8B" and "PMULL2 Vd.8H, Vn.16B, Vm.16B".

3.16 ASIMD floating-point instructions

Table 28: 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	FABD, FADD, FSUB, FADDP	2	2	V	-
ASIMD FP compare	FACGE, FACGT, FCMEQ, FCMGE, FCMGT, FCMLE, FCMLT	2	2	V	-
ASIMD FP convert, long (F16 to F32)	FCVTL(2)	4	1/2	V0	-
ASIMD FP convert, long (F32 to F64)	FCVTL(2)	3	1	V0	-
ASIMD FP convert, narrow (F32 to F16)	FCVTN(2)	4	1/2	V0	-
ASIMD FP convert, narrow (F64 to F32)	FCVTN(2), FCVTXN(2)	3	1	V0	-
ASIMD FP convert, other, D-form F32 and Q-form F64	FCVTAS, FCVTAU, FCVTMS, FCVTMU, FCVTNS, FCVTNU, FCVTPS, FCVTPU, FCVTZS, FCVTZU, SCVTF, UCVTF	3	1	V0	-

Instruction Group	AArch64 Instructions	Exec Latency	Execution Throughput	Utilized Pipelines	Notes
ASIMD FP convert, other, D-form F16 and Q-form F32	FCVTAS, FCVTAU, FCVTMS, FCVTMU, FCVTNS, FCVTNU, FCVTPS, FCVTPU, FCVTZS, FCVTZU, SCVTF, UCVTF	4	1/2	V0	-
ASIMD FP convert, other, Q-form F16	FCVTAS, FCVTAU, FCVTMS, FCVTMU, FCVTNS, FCVTNU, FCVTPS, FCVTPU, FCVTZS, FCVTZU, SCVTF, UCVTF	6	1/4	V0	-
ASIMD FP divide, D-form, F16	FDIV	7	1/7	V0	3
ASIMD FP divide, D-form, F32	FDIV	7 to 10	2/9 to 2/7	V0	3
ASIMD FP divide, Q-form, F16	FDIV	10 to 13	1/13 to 1/10	V0	3
ASIMD FP divide, Q-form, F32	FDIV	7 to 10	1/9 to 1/7	V0	3
ASIMD FP divide, Q-form, F64	FDIV	7 to 15	1/14 to 1/7	V0	3
ASIMD FP max/min, normal	FMAX, FMAXNM, FMIN, FMINNM	2	2	V	-

Instruction Group	AArch64 Instructions	Exec Latency	Execution Throughput	Utilized Pipelines	Notes
ASIMD FP max/min, pairwise	FMAXP, FMAXNMP, FMINP, FMINNMP	2	2	V	-
ASIMD FP max/min, reduce	FMAXV, FMAXNMV, FMINV, FMINNMV	5	2	V	-
ASIMD FP max/min, reduce, Q-form F16	FMAXV, FMAXNMV, FMINV, FMINNMV	8	2/3	V	-
ASIMD FP multiply	FMUL, FMULX	3	2	V	2
ASIMD FP multiply accumulate	FMLA, FMLS	4 (2)	2	V	1
ASIMD FP multiply accumulate long	FMLAL(2), FMLSL(2)	5(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	3	1	V0	-
ASIMD FP round, D-form F16 and Q-form F32	FRINTA, FRINTI, FRINTM, FRINTN, FRINTP, FRINTX, FRINTZ	4	1/2	V0	-
ASIMD FP round, Q-form F16	FRINTA, FRINTI, FRINTM, FRINTN, FRINTP, FRINTX, FRINTZ	6	1/4	V0	-

Instruction Group	AArch64 Instructions	Exec Latency	Execution Throughput	Utilized Pipelines	Notes
ASIMD FP square root, D-form, F16	FSQRT	7	1/7	V0	3
ASIMD FP square root, D-form, F32	FSQRT	7 to 10	2/9 to 2/7	V0	3
ASIMD FP square root, Q-form, F16	FSQRT	11 to 13	1/13 to 1/11	V0	3
ASIMD FP square root, Q-form, F32	FSQRT	7 to 10	1/9 to 1/7	V0	3
ASIMD FP square root, Q-form, F64	FSQRT	7 to 17	1/16 to 1/7	V0	3

Table 29: AArch32 ASIMD integer instructions

Instruction Group	AArch32 Instructions	Exec Latency	Execution Throughput	Utilized Pipelines	Notes
ASIMD FP absolute value	VABS	2	2	V	-
ASIMD FP arith	VABD, VADD, VPADD, VSUB	2	2	V	-
ASIMD FP compare	VACGE, VACGT, VACLE, VACLT, VCEQ, VCGE, VCGT, VCLE	2	2	V	-
ASIMD FP convert, integer, D-form	VCVT, VCVTA, VCVTM, VCVTN, VCVTP	3	1	V0	-
ASIMD FP convert, integer, Q-form	VCVT, VCVTA, VCVTM, VCVTN, VCVTP	4	1/2	V0	-

Instruction Group	AArch32 Instructions	Exec Latency	Execution Throughput	Utilized Pipelines	Notes
ASIMD FP convert, fixed, D-form	VCVT	3	1	V0	-
ASIMD FP convert, fixed, Q-form	VCVT	4	1/2	V0	-
ASIMD FP convert, half-precision	VCVT	4	1/2	V0	-
ASIMD FP max/min	VMAX, VMIN, VPMAX, VPMIN, VMAXNM, VMINNM	2	2	V	-
ASIMD FP multiply	VMUL, VNMUL	3	2	V	2
ASIMD FP chained multiply accumulate	VMLA, VMLS	5(2)	2	V	1
ASIMD FP fused multiply accumulate	VFMA, VFMS	4(2)	2	V	1
ASIMD FP negate	VNEG	2	2	V	
ASIMD FP round to integral, D-form	VRINTA, VRINTM, VRINTN, VRINTP, VRINTX, VRINTZ	3	1	V0	-
ASIMD FP round to integral, Q-form	VRINTA, VRINTM, VRINTN, VRINTP, VRINTX, VRINTZ	4	1/2	V0	-

1. ASIMD multiply-accumulate pipelines support late-forwarding of accumulate operands from similar µOPs, allowing a typical sequence of floating-point multiply-accumulate µOPs to issue one every N cycles (accumulate latency N shown in parentheses).
2. ASIMD multiply-accumulate pipelines support late forwarding of the result from ASIMD FP multiply µOPs to the accumulate operands of an ASIMD FP multiply-accumulate µOP. The latter can potentially be issued 1 cycle after the ASIMD FP multiply µOP has been issued.
3. ASIMD divide and square root operations are performed using an iterative algorithm and

block subsequent similar operations to the same pipeline until complete.

3.17 ASIMD miscellaneous instructions

Table 30: AArch64 ASIMD miscellaneous instructions

Instruction Group	AArch64 Instructions	Exec Latency	Execution Throughput	Utilized Pipelines	Notes
ASIMD bit reverse	RBIT	2	2	V	-
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	-
ASIMD extract	EXT	2	2	V	-
ASIMD extract narrow	XTN	2	2	V	-
ASIMD extract narrow, saturating	SQXTN(2), SQXTUN(2), UQXTN(2)	4	1	V1	-
ASIMD insert, element to element	INS	2	2	V	-
ASIMD move, FP immed	FMOV	2	2	V	-
ASIMD move, integer immed	MOVI, MVNI	2	2	V	-
ASIMD reciprocal estimate, D-form F32 and F64	FRECPE, FRECPX, FRSQRTE, URECPE, URSQRTE	3	1	V0	-
ASIMD reciprocal estimate, D-form F16 and Q-form F32	FRECPE, FRECPX, FRSQRTE, URECPE, URSQRTE	4	1/2	V0	-

Instruction Group	AArch64 Instructions	Exec Latency	Execution Throughput	Utilized Pipelines	Notes
ASIMD reciprocal estimate, Q-form F16	FRECPE, FRECPX, FRSQRTE, URECPE, URSQRTE	6	1/4	V0	-
ASIMD reciprocal step	FRECPS, FRSQRTS	4	2	V	-
ASIMD reverse	REV16, REV32, REV64	2	2	V	-
ASIMD table lookup, 1 or 2 table regs	TBL	2	2	V	-
ASIMD table lookup, 3 table regs	TBL	4	1/2	V	-
ASIMD table lookup, 4 table regs	TBL	4	2/3	V	-
ASIMD table lookup extension, 1 table reg	TBX	2	2	V	-
ASIMD table lookup extension, 2 table reg	TBX	4	1/2	V	-
ASIMD table lookup extension, 3 table reg	TBX	6	2/3	V	-
ASIMD table lookup extension, 4 table reg	TBX	6	2/5	V	-
ASIMD transfer, element to gen reg	UMOV, SMOV	2	1	V1	-
ASIMD transfer, gen reg to element	INS	5	1	M0, V	-
ASIMD transpose	TRN1, TRN2	2	2	V	-
ASIMD unzip/zip	UZP1, UZP2, ZIP1, ZIP2	2	2	V	-

Table 31: AArch32 ASIMD miscellaneous instructions

Instruction Group	AArch32 Instructions	Exec Latency	Execution Throughput	Utilized Pipelines	Notes
ASIMD bitwise insert	VBIF, VBIT, VBSL	2	2	V	-
ASIMD count	VCLS, VCLZ, VCNT	2	2	V	-
ASIMD duplicate, core reg	VDUP	3	1	M0	-
ASIMD duplicate, scalar	VDUP	2	2	V	-
ASIMD extract	VEXT	2	2	V	-
ASIMD move, immed	VMOV	2	2	V	-
ASIMD move, register	VMOV	2	2	V	-
ASIMD move, narrowing	VMOVN	2	2	V	-
ASIMD move, saturating	VQMOVN, VQMOVUN	4	1	V1	-
ASIMD reciprocal estimate, D-form	VRECPE, VRSQRTE	3	1	V0	-
ASIMD reciprocal estimate, Q-form	VRECPE, VRSQRTE	4	1/2	V0	-
ASIMD reciprocal step	VRECPS, VRSQRTS	5	2	V	-
ASIMD reverse	VREV16, VREV32, VREV64	2	2	V	-
ASIMD swap	VSWP	4	2/3	V	-
ASIMD table lookup, 1 or 2 table regs	VTBL	2	2	V	-
ASIMD table lookup, 3 table regs	VTBL	4	1/2	V	-
ASIMD table lookup, 4 table regs	VTBL	4	2/3	V	-
ASIMD table lookup extension, 1 reg	VTBX	2	2	V	-
ASIMD table lookup extension, 2 table reg	VTBX	4	1/2	V	-

Instruction Group	AArch32 Instructions	Exec Latency	Execution Throughput	Utilized Pipelines	Notes
ASIMD table lookup extension, 3 table reg	VTBX	6	2/3	V	-
ASIMD table lookup extension, 4 table reg	VTBX	6	2/5	V	-
ASIMD transfer, scalar to core reg, word	VMOV	2	1	V1	-
ASIMD transfer, scalar to core reg, byte/hword	VMOV	3	1	V1, I	-
ASIMD transfer, core reg to scalar	VMOV	5	1	M0, V	-
ASIMD transpose	VTRN	4	2/3	V	-
ASIMD unzip/zip	VUZP, VZIP	4	2/3	V	-

3.18 ASIMD load instructions

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

Table 32: 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	5	2	L	-
ASIMD load, 1 element, multiple, 1 reg, Q-form	LD1	5	2	L	-
ASIMD load, 1 element, multiple, 2 reg, D-form	LD1	5	1	L	-
ASIMD load, 1 element, multiple, 2 reg, Q-form	LD1	5	1	L	-
ASIMD load, 1 element, multiple, 3 reg, D-form	LD1	6	2/3	L	-
ASIMD load, 1 element, multiple, 3 reg, Q-form	LD1	6	2/3	L	-
ASIMD load, 1 element, multiple, 4 reg, D-form	LD1	6	1/2	L	-
ASIMD load, 1 element, multiple, 4 reg, Q-form	LD1	6	1/2	L	-

Instruction Group	AArch64 Instructions	Exec Latency	Execution Throughput	Utilized Pipelines	Notes
ASIMD load, 1 element, one lane, B/H/S	LD1	7	2	L, V	-
ASIMD load, 1 element, one lane, D	LD1	7	2	L, V	-
ASIMD load, 1 element, all lanes, D-form, B/H/S	LD1R	7	2	L, V	-
ASIMD load, 1 element, all lanes, D-form, D	LD1R	7	2	L, V	-
ASIMD load, 1 element, all lanes, Q-form	LD1R	7	2	L, V	-
ASIMD load, 2 element, multiple, D-form, B/H/S	LD2	7	1	L, V	-
ASIMD load, 2 element, multiple, Q-form, B/H/S	LD2	7	1	L, V	-
ASIMD load, 2 element, multiple, Q-form, D	LD2	7	1	L, V	-
ASIMD load, 2 element, one lane, B/H	LD2	7	1	L, V	-
ASIMD load, 2 element, one lane, S	LD2	7	1	L, V	-
ASIMD load, 2 element, one lane, D	LD2	7	1	L, V	-
ASIMD load, 2 element, all lanes, D-form, B/H/S	LD2R	7	1	L, V	-
ASIMD load, 2 element, all lanes, D-form, D	LD2R	7	1	L, V	-
ASIMD load, 2 element, all lanes, Q-form	LD2R	7	1	L, V	-
ASIMD load, 3 element, multiple, D-form, B/H/S	LD3	8	1/2	L, V	-
ASIMD load, 3 element, multiple, Q-form, B/H/S	LD3	8	1/2	L, V	-
ASIMD load, 3 element, multiple, Q-form, D	LD3	8	1/2	L, V	-
ASIMD load, 3 element, one lane, B/H	LD3	7	1/2	L, V	-

Instruction Group	AArch64 Instructions	Exec Latency	Execution Throughput	Utilized Pipelines	Notes
ASIMD load, 3 element, one lane, S	LD3	7	1/2	L, V	-
ASIMD load, 3 element, one lane, D	LD3	7	1/2	L, V	-
ASIMD load, 3 element, all lanes, D-form, B/H/S	LD3R	7	1/2	L, V	-
ASIMD load, 3 element, all lanes, D-form, D	LD3R	7	1/2	L, V	-
ASIMD load, 3 element, all lanes, Q-form, B/H/S	LD3R	7	1/2	L, V	-
ASIMD load, 3 element, all lanes, Q-form, D	LD3R	7	1/2	L, V	-
ASIMD load, 4 element, multiple, D-form, B/H/S	LD4	8	2/7	L, V	-
ASIMD load, 4 element, multiple, Q-form, B/H/S	LD4	10	1/5	L, V	-
ASIMD load, 4 element, multiple, Q-form, D	LD4	10	1/5	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	1/2	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	1/2	L, V	-
ASIMD load, 4 element, all lanes, Q-form, D	LD4R	8	1/2	L, V	-
(ASIMD load, writeback form)	-	(1)	-	+ I	1

Table 33: AArch32 ASIMD load instructions

Instruction Group	AArch32 Instructions	Exec Latency	Execution Throughput	Utilized Pipelines	Notes
ASIMD load, 1 element, multiple, 1 reg	VLD1	5	2	L	-
ASIMD load, 1 element, multiple, 2 reg	VLD1	5	2	L	-
ASIMD load, 1 element, multiple, 3 reg	VLD1	5	1	L	-
ASIMD load, 1 element, multiple, 4 reg	VLD1	5	1	L	-
ASIMD load, 1 element, one lane	VLD1	7	2	L, V	-
ASIMD load, 1 element, all lanes, 1 reg	VLD1	7	2	L V	-
ASIMD load, 1 element, all lanes, 2 reg	VLD1	7	2/3	L, V	-
ASIMD load, 2 element, multiple, 2 reg	VLD2	7	2/3	L, V	-
ASIMD load, 2 element, multiple, 4 reg	VLD2	8	1/2	L, V	-
ASIMD load, 2 element, one lane, size 32	VLD2	7	1	L, V	-
ASIMD load, 2 element, one lane, size 8/16	VLD2	7	1	L, V	-
ASIMD load, 2 element, all lanes	VLD2	7	1	L, V	-
ASIMD load, 3 element, multiple, 3 reg	VLD3	8	2/3	L, V	-
ASIMD load, 3 element, one lane, size 32	VLD3	8	2/3	L, V	-
ASIMD load, 3 element, one lane, size 8/16	VLD3	8	2/3	L, V	-
ASIMD load, 3 element, all lanes	VLD3	8	2/3	L, V	-
ASIMD load, 4 element, multiple, 4 reg	VLD4	8	1/2	L, V	-
ASIMD load, 4 element, one lane, size 32	VLD4	8	1/2	L, V	-

Instruction Group	AArch32 Instructions	Exec Latency	Execution Throughput	Utilized Pipelines	Notes
ASIMD load, 4 element, one lane, size 8/16	VLD4	8	1/2	L, V	-
ASIMD load, 4 element, all lanes	VLD4	8	1/2	L, V	-
(ASIMD load, writeback form)	-	(1)	-	+I	1

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

3.19 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 34: 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	L, V	-
ASIMD store, 1 element, multiple, 1 reg, Q-form	ST1	2	1	L, V	-
ASIMD store, 1 element, multiple, 2 reg, D-form	ST1	2	1	L, V	-
ASIMD store, 1 element, multiple, 2 reg, Q-form	ST1	3	1/2	L, V	-
ASIMD store, 1 element, multiple, 3 reg, D-form	ST1	3	2/3	L, V	-
ASIMD store, 1 element, multiple, 3 reg, Q-form	ST1	4	1/3	L, V	-
ASIMD store, 1 element, multiple, 4 reg, D-form	ST1	3	1/2	L, V	-
ASIMD store, 1 element, multiple, 4 reg, Q-form	ST1	5	1/4	L, V	-
ASIMD store, 1 element, one lane, B/H/S	ST1	4	1	V, L	-

Instruction Group	AArch64 Instructions	Exec Latency	Execution Throughput	Utilized Pipelines	Notes
ASIMD store, 1 element, one lane, D	ST1	4	1	V, L	-
ASIMD store, 2 element, multiple, D-form, B/H/S	ST2	4	1	V, L	-
ASIMD store, 2 element, multiple, Q-form, B/H/S	ST2	5	1/2	V, L	-
ASIMD store, 2 element, multiple, Q-form, D	ST2	5	1/2	V, L	-
ASIMD store, 2 element, one lane, B/H/S	ST2	4	1	V, L	-
ASIMD store, 2 element, one lane, D	ST2	4	1	V, L	-
ASIMD store, 3 element, multiple, D-form, B/H/S	ST3	5	1/2	V, L	-
ASIMD store, 3 element, multiple, Q-form, B/H/S	ST3	6	1/3	V, L	-
ASIMD store, 3 element, multiple, Q-form, D	ST3	6	1/3	V, L	-
ASIMD store, 3 element, one lane, B/H	ST3	4	1/2	V, L	-
ASIMD store, 3 element, one lane, S	ST3	4	1/2	V, L	-
ASIMD store, 3 element, one lane, D	ST3	5	1/2	V, L	-
ASIMD store, 4 element, multiple, D-form, B/H/S	ST4	7	1/3	V, L	-
ASIMD store, 4 element, multiple, Q-form, B/H/S	ST4	9	1/6	V, L	-
ASIMD store, 4 element, multiple, Q-form, D	ST4	6	1/4	V, L	-
ASIMD store, 4 element, one lane, B/H	ST4	5	-	V, L	-
ASIMD store, 4 element, one lane, S	ST4	-	2/3	V, L	-
ASIMD store, 4 element, one lane, D	ST4	-	-	V, L	-

Instruction Group	AArch64 Instructions	Exec Latency	Execution Throughput	Utilized Pipelines	Notes
(ASIMD store, writeback form)	-	(1)	-	Add I	1

Table 35: AArch32 ASIMD store instructions

Instruction Group	AArch32 Instructions	Exec Latency	Execution Throughput	Utilized Pipelines	Notes
ASIMD store, 1 element, multiple, 1 reg	VST1	2	2	L, V	-
ASIMD store, 1 element, multiple, 2 reg	VST1	2	2	L, V	-
ASIMD store, 1 element, multiple, 3 reg	VST1	3	2/3	L, V	-
ASIMD store, 1 element, multiple, 4 reg	VST1	3	1/2	L, V	-
ASIMD store, 1 element, one lane	VST1	4	2	V, L	-
ASIMD store, 2 element, multiple, 2 reg	VST2	4	1	V, L	-
ASIMD store, 2 element, multiple, 4 reg	VST2	5	1/2	V, L	-
ASIMD store, 2 element, one lane	VST2	4	2	V, L	-
ASIMD store, 3 element, multiple, 3 reg	VST3	5	2/3	V, L	-
ASIMD store, 3 element, one lane, size 32	VST3	4	1	V, L	-
ASIMD store, 3 element, one lane, size 8/16	VST3	4	1	V, L	-
ASIMD store, 4 element, multiple, 4 reg	VST4	8	1/2	V, L	-
ASIMD store, 4 element, one lane, size 32	VST4	7	2	V, L	-
ASIMD store, 4 element, one lane, size 8/16	VST4	7	2	V, L	-
(ASIMD store, writeback form)	-	(1)	-	+I	1

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

3.20 Cryptography extensions

Table 36: 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 op	SHA1H	2	1	V0	-
Crypto SHA1 hash acceleration ops	SHA1C, SHA1M, SHA1P	4	1	V0	-
Crypto SHA1 schedule acceleration ops	SHA1SU0, SHA1SU1	2	1	V0	-
Crypto SHA256 hash acceleration ops	SHA256H, SHA256H2	4	1	V0	-
Crypto SHA256 schedule acceleration ops	SHA256SU0, SHA256SU1	2	1	V0	-

Table 37: AArch32 Cryptography extensions

Instruction Group	AArch32 Instructions	Exec Latency	Execution Throughput	Utilized Pipelines	Notes
Crypto AES ops	AESD, AESE, AESIMC, AESMC	2	2	V	1
Crypto polynomial (64x64) multiply long	VMULL.P64	2	1	V0	-
Crypto SHA1 hash acceleration op	SHA1H	2	1	V0	-

Instruction Group	AArch32 Instructions	Exec Latency	Execution Throughput	Utilized Pipelines	Notes
Crypto SHA1 hash acceleration ops	SHA1C, SHA1M, SHA1P	4	1	V0	-
Crypto SHA1 schedule acceleration ops	SHA1SU0, SHA1SU1	2	1	V0	-
Crypto SHA256 hash acceleration ops	SHA256H, SHA256H2	4	1	V0	-
Crypto SHA256 schedule acceleration ops	SHA256SU0, SHA256SU1	2	1	V0	-

Adjacent AESE/AESMC instruction pairs and adjacent AESD/AESIMC instruction pairs will exhibit the performance characteristics described in Section 4.6.

3.21 CRC

Table 38: AArch64 CRC

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

Table 39: AArch32 CRC

Instruction Group	AArch32 Instructions	Exec Latency	Execution Throughput	Utilized Pipelines	Notes
CRC checksum ops	CRC32, CRC32C	2	1	M0	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.

4 Special considerations

4.1 Dispatch constraints

Dispatch of µOPs from the in-order portion to the out-of-order portion of the microarchitecture includes several constraints. It is important to consider these constraints during code generation to maximize the effective dispatch bandwidth and subsequent execution bandwidth of Cortex-A77.

The dispatch stage can process up to 6 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 L pipelines
- Up to 4 µOPs utilizing the D pipelines

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

4.2 Dispatch stall

In the event of a V-pipeline µOP containing more than 1 quad-word register source, a portion or all of which was previously written as one or multiple single words, that µOP will stall in dispatch for three cycles. This stall occurs only on the first such instance, and subsequent consumers of the same register will not experience this stall.

4.3 Optimizing general-purpose register spills and fills

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

4.4 Optimizing memory copy

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

- Unroll the loop to include multiple load and store operations per iteration, minimizing the overheads of looping.
- Align stores on 16B boundary wherever possible.
- Use non-writeback forms of LDP and STP/STR instructions interleaving them like shown in the examples below:

For forward copies:

```
Loop_start:
        SUBS    X2, X2, #96
        LDP     Q3, Q4, [x1, #0]
        STP     Q3, Q4, [x0, #0]
        LDP     Q3, Q4, [x1, #32]
        STP     Q3, Q4, [x0, #32]
        LDP     Q3, Q4, [x1, #64]
        STP     Q3, Q4, [x0, #64]
        ADD     X1, X1, #96
        ADD     X0, X0, #96
        BGT     Loop_start
```

For backward copies:

```
Loop_start:
        SUBS    X2,X2,#96
        LDP     Q4,Q3,[x1,#-32]
        STR     Q3,,[x0,#-16]
        STR     Q4,,[x0,#-32]
        LDP     Q4,Q3,[x1,#-64]
        STR     Q3,[x0,#-48]
        STR     Q4,[x0,#-64]
        LDP     Q4,Q3,[x1,#-96]
        STR     Q3,[x0,#-80]
        STR     Q4,[x0,#-96]
        SUB     X1,X1,#96
        SUB     X0,X0,#96
        BGT     Loop_start
```

A recommended copy routine for AArch32 would look like the sequence above but would use LDRD/STRD instructions. Avoid load-/store-multiple instruction encodings (such as LDM and STM).

4.5 Load/Store alignment

The Armv8.2-A architecture allows many types of load and store accesses to be arbitrarily aligned. The Cortex-A77 core handles most unaligned accesses without performance penalties.

However, there are cases which reduce bandwidth or incur additional latency, as described below.

- Load operations that cross a cache-line (64-byte) boundary.
- Quad-word load operations that are not 4B aligned.
- Store operations that cross a 16B boundary.

4.6 Store to Load Forwarding

The Cortex-A77 core allows data to be forwarded from store instructions to a load instruction with the restrictions mentioned below:

- Load start address should align with the start or middle address of the older store
- Loads of size greater than or equal to 8 bytes can get the data forwarded from a maximum of 2 stores. If there are 2 stores, then each store should forward to either first or second half of the load
- Loads of size less than or equal to 4 bytes can get their data forwarded from only 1 store

4.7 AES encryption/decryption

Cortex-A77 can issue two AESE/AESMC/AESD/AESIMC instruction every cycle (fully pipelined) with an execution latency of two cycles. This means encryption or decryption for at least four data chunks should be interleaved for maximum performance:

```
AESE  data0, key0
AESMC data0, data0
AESE  data1, key0
AESMC data1, data1
AESE  data2, key0
AESMC data2, data2
AESE  data3, key1
AESMC data3, data3
AESE  data0, key0
AESMC data0, data0
...
```

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

4.8 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. These optimized forwarding regions are defined in the following table.

Table 40: Optimized forwarding regions

Region	Instruction Types
1	ASIMD ALU, ASIMD shift and certain ASIMD Miscellaneous.
2	FP multiply, FP multiply-accumulate, FP compare, FP add/sub and certain ASIMD Miscellaneous.
3	Cryptography Extensions.

In addition to the regions mentioned in the table above, all floating point and ASIMD instructions can fast forward to FP and ASIMD stores.

Exceptions to these forwarding regions are as follows:

- Fast forwarding will not occur in AArch32 mode if the consuming register's width is greater than that of the producer.
- Element sources used by FP multiply and multiply-accumulate operations cannot be consumers.
- Complex ASIMD shift by immediate/register and shift accumulate instructions cannot be producers (see section 3.14) in region 1.
- ASIMD extract narrow, saturating instructions cannot be producers (see section 3.16) in region 1.
- ASIMD absolute difference accumulate and pairwise add and accumulate instructions cannot be producers (see section 3.14) in region 1.
- For FP producer-consumer pairs, the precision of the instructions should match (single, double or half) in region 2.
- Pair-wise FP instructions cannot be consumers in region 2.

The effective latency of FP and ASIMD instructions as described in section 3 is increased by one cycle if the producer and consumer instructions are not part of the same forwarding region, or if they are affected by the exceptions described above. ASIMD integer multiply/ multiply-accumulate, ASIMD max/min reduction, FP divide and square root, FP convert/round and reciprocal/reciprocal sqrt estimate instructions are not subject to this behavior.

4.9 Branch instruction alignment

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

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

4.10 FPCR self-synchronization

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

4.11 Special register access

The Cortex-A77 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 41: Special-purpose register read accesses

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

Register Read	Non-Speculative	In-Order	Flush Side-Effect	Notes
SPSR_*	No	Yes	No	-

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

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

Table 42: Special-purpose register write accesses

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

1. The NZCV and SP registers are fully renamed.
2. If the FPCR/FPSCR write is predicted to change the control field values, it will introduce a barrier which prevents subsequent instructions from executing. If the FPCR/FPSCR write is predicted to not change the control field values, it will execute without a barrier but trigger a flush if the values change.
3. FPSR/FPSCR writes must stall at dispatch if another FPSR/FPSCR write is still pending.
4. APSR writes that set the Q bit will introduce a barrier which prevents subsequent

instructions from executing until the write completes.

4.12 Register forwarding hazards

The Armv8-A architecture allows FP/ASIMD instructions to read and write 32-bit S-registers. In AArch32, Each S-register corresponds to one half (upper or lower) of an overlaid 64-bit D-register. A Q register in turn consists of two overlaid D register. Register forwarding hazards may occur when one µOP reads a Q-register operand that has recently been written with one or more S-register result. Consider the following scenario.

```
    VADD   S0, S1, S2
    VADD   Q6, Q5, Q0
```

The first instruction writes S0, which correspond to the lowest part of Q0. The second instruction then requires Q0 as an input operand. In this scenario, there is a dependency RAW dependency between the first and the second instructions. In most cases, Cortex-A77 performs slightly worse in such situations.

Cortex-A77 is able to avoid this register-hazard condition for certain cases. The following rules describe the conditions under which a register-hazard can occur.

- The producer writes an S-register (not a D[x] scalar)
- The consumer reads an overlapping Q-register (not as a D[x] scalar)
- The consumer is a FP/ASIMD µOP (not a store or MOV µOP)

To avoid unnecessary hazards, it is recommended that the programmer use D[x] scalar writes when populating registers prior to ASIMD operations. For example, either of the following instruction forms would safely prevent a subsequent hazard.

```
VLD1.32    D0[x], [address]
VADD       Q1, Q0, Q2F
```

4.13 IT blocks

The Armv8-A architecture performance deprecates some uses of the IT instruction in such a way that software may be written using multiple naïve single instruction IT blocks. It is preferred that software instead generate multi instruction IT blocks rather than single instruction blocks.

4.14 Instruction fusion

The Cortex-A77 core can accelerate certain instruction pairs in an operation called fusion. The following instruction pairs can be fused in Aarch64 mode only:

1. CMP/CMN (immediate) + B.cond
2. CMP/CMN (register) + B.cond
3. TST (immediate) + B.cond

4. TST (register) + B.cond
5. BICS (register) + B.cond
6. NOP + Any instruction

The following instruction pairs are fused in both Aarch32 and Aarch64 modes:

1. AESE + AESMC (see Section 4.6 on AES Encryption/Decryption)
2. AESD + AESIMC (see Section 4.6 on AES Encryption/Decryption)

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

4.15 Mixing Arm and Thumb state

Mixing Arm and Thumb instructions in the same cache-line should be avoided. In particular old-style interworking veneers to switch from Thumb to Arm state using BX pc may be very slow. This overhead can be reduced by inserting a direct branch or return between indirect branches in one state and code in the other state. For example:

```
BX pc   // Thumb to Arm veneer
B.-2    // never executed
… Arm code
```

However, it is preferable to remove the indirect branch by using only Thumb-2 or Arm code for each veneer.