



Arm® Cortex®-A75 Core

Revision: r3p1

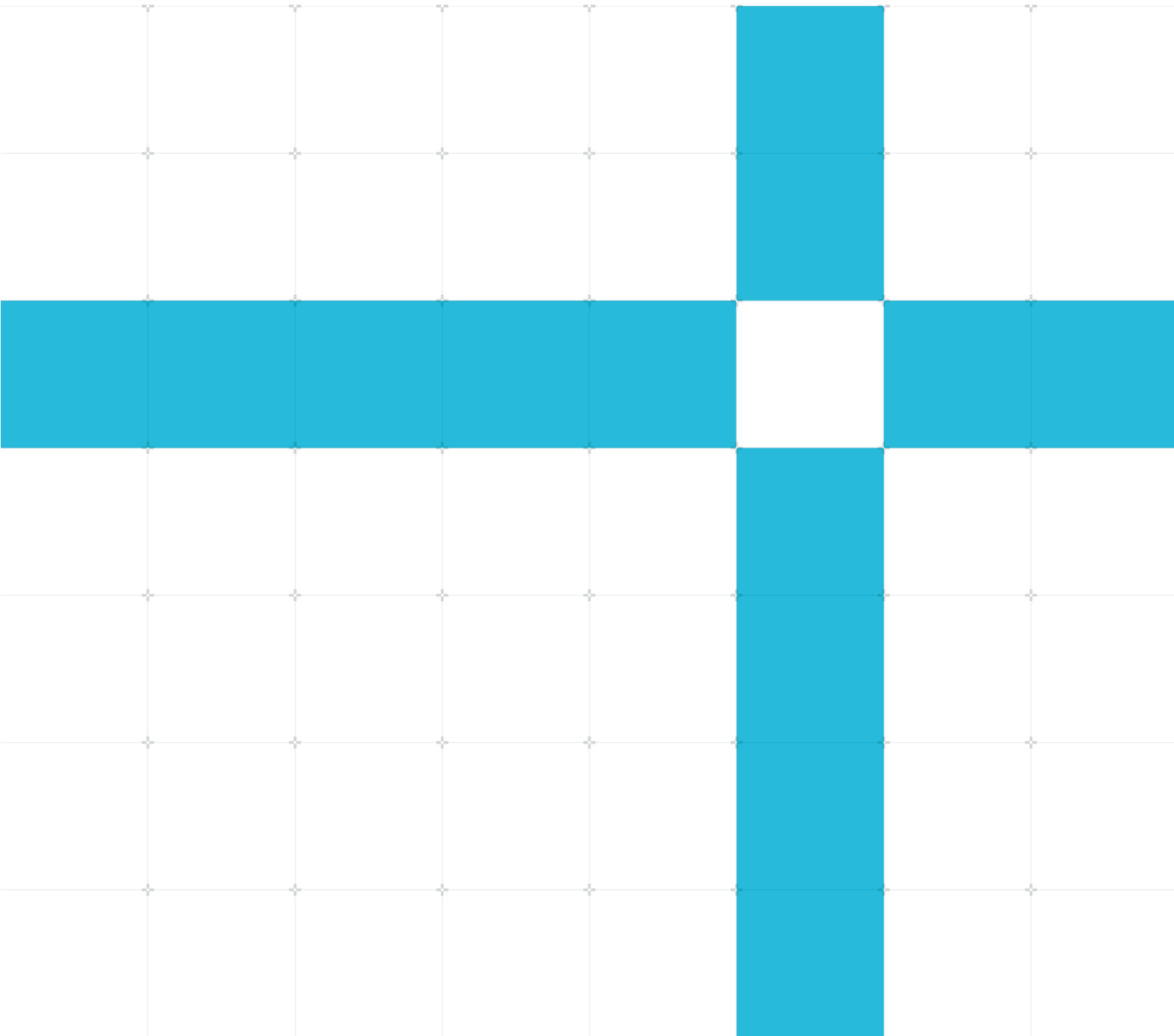
Software Optimization Guide

Non-Confidential

Issue 3.0

Copyright © 2018, 2024 Arm Limited (or its affiliates).
All rights reserved.

109757



Arm® Cortex®-A75 Core Software Optimization Guide

Copyright © 2018, 2024 Arm Limited (or its affiliates). All rights reserved.

Release information

Document history

Issue	Date	Confidentiality	Change
1.0	14 May 2018	Confidential	First release
2.0	31 May 2018	Non-Confidential	Editorial changes and confidentiality status change
3.0	06 June 2024	Non-Confidential	Editorial changes and document ID change

Proprietary Notice

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

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

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

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

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

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

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

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-1121-V1.0

Confidentiality Status

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

Unrestricted Access is an Arm internal classification.

Product Status

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

Web Address

developer.arm.com

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.

This document includes terms that can be offensive. We will replace these terms in a future issue of this document. If you find offensive terms in this document, please email terms@arm.com.

Contents

1 Introduction	6
1.1 Product revision status.....	6
1.2 Intended audience	6
1.3 Scope.....	6
1.4 Conventions	6
1.4.1 Glossary.....	6
1.4.2 Terms and abbreviations	6
1.4.3 Typographical conventions	8
1.5 Useful resources.....	9
1.6 Feedback.....	10
1.6.1 Feedback on this product	10
1.6.2 Feedback on content.....	10
2 Pipeline	11
2.1 Overview	11
3 Instruction characteristics	13
3.1 Instruction tables	13
3.2 Branch instructions.....	14
3.3 Arithmetic and logical instructions	15
3.4 Move and shift instructions.....	16
3.5 Saturating and parallel arithmetic instructions	17
3.6 Divide and multiply instructions.....	18
3.7 Miscellaneous data-processing instructions	20
3.8 Load instructions.....	22
3.9 Store instructions	25
3.10 Floating-point data processing instructions	28
3.11 Floating-point miscellaneous instructions.....	30
3.12 Floating-point load instructions	31
3.13 Floating-point store instructions	33
3.14 Advanced SIMD integer instructions.....	35
3.15 Advanced SIMD floating-point instructions	41
3.16 Advanced SIMD miscellaneous instructions	46

3.17 Advanced SIMD load instructions	50
3.18 Advanced SIMD store instructions	54
3.19 Cryptographic Extension	57
3.20 CRC	59
4 Special considerations	60
4.1 Dispatch constraints	60
4.2 Conditional ASIMD	60
4.3 Optimizing memory copy	60
4.4 Load/store alignment.....	61
4.5 AES encryption and decryption	61
4.6 Branch instruction alignment.....	62
4.7 Region-based fast forwarding.....	62
4.8 FPCR self-synchronization	62
4.9 Special register access.....	62
4.10 IT blocks	64

1 Introduction

1.1 Product revision status

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

rx

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

py

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

1.2 Intended audience

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

1.3 Scope

This document describes aspects of the Cortex-A75 core micro-architecture that influence software performance so that software and compilers can be optimized accordingly. Micro-architectural detail is limited to that which is useful for software optimization.

Documentation extends only to software visible behavior of the Cortex-A75 core and not to the hardware rationale behind the behavior.

1.4 Conventions

The following subsections describe conventions used in Arm documents.

1.4.1 Glossary

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







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

1.4.2 Terms and abbreviations

This document uses the following terms and abbreviations.

Term	Meaning
ALU	Arithmetic and Logical Unit
ASIMD	Advanced SIMD
MAC	Multiply-Accumulate
SQRT	Square Root
FP	Floating-point
CRC	Cyclic Redundancy Check

1.4.3 Typographical conventions

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

1.5 Useful resources

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

- Arm Non-Confidential documents are available on 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 products	Document ID	Confidentiality
Arm® Cortex®-A75 Core Technical Reference Manual	100403	Non-Confidential

Arm architecture and specifications	Document ID	Confidentiality
Arm® Architecture Reference Manual for A-profile architecture	DDI 0487	Non-Confidential



Arm tests its PDFs only in Adobe Acrobat and Acrobat Reader. Arm cannot guarantee the quality of its documents when used with any other PDF reader.
Adobe PDF reader products can be downloaded at <http://www.adobe.com>.

1.6 Feedback

Arm welcomes feedback on this product and its documentation.

1.6.1 Feedback on this product

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

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

1.6.2 Feedback on content

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

- The title Arm® Cortex®-A75 Core Software Optimization Guide.
- The number 109757.
- 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.



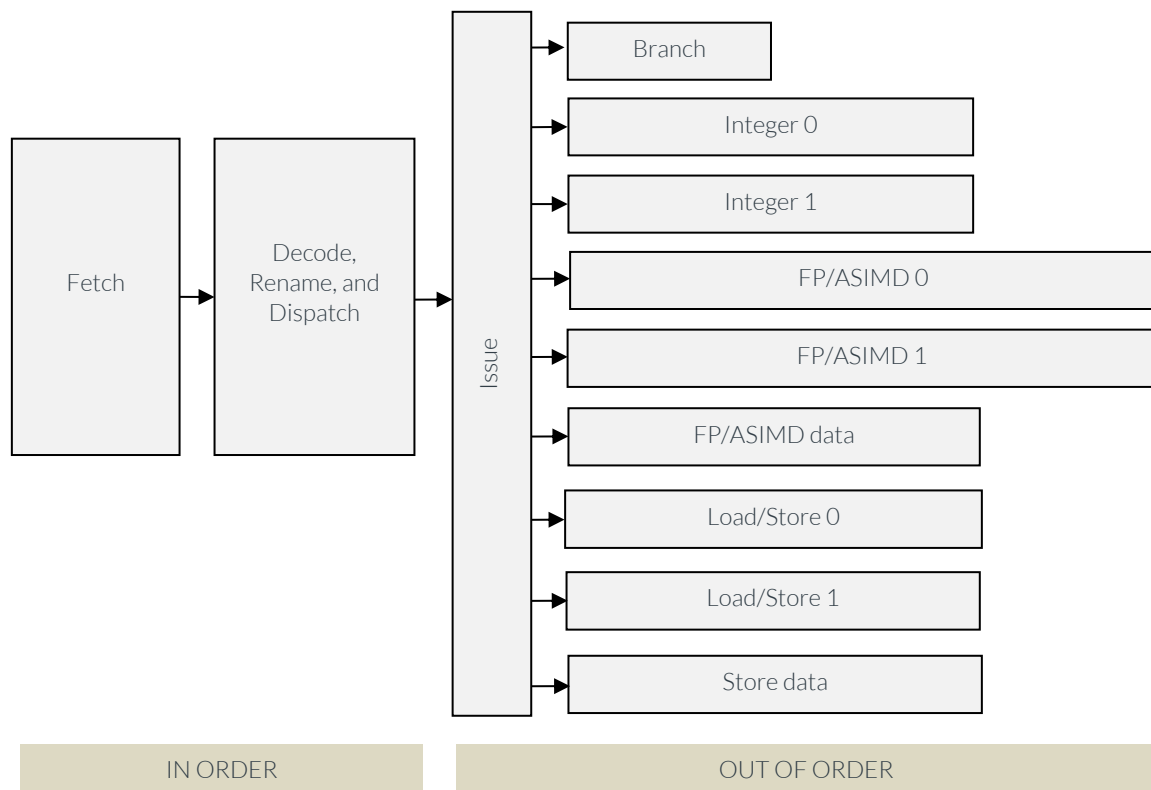
Arm tests the PDF only in Adobe Acrobat and Acrobat Reader and cannot guarantee the quality of the represented document when used with any other PDF reader.

2 Pipeline

2.1 Overview

The following figure shows a high-level Cortex-A75 instruction processing pipeline. Instructions are fetched and then decoded into internal micro-operations. From there, the micro-operations proceed through register renaming and dispatch stages. Once they are dispatched, the micro-operations wait for their operands and issue out of order to one of the execution pipelines. Each execution pipeline can accept and complete one micro-operation per cycle.

Figure 2-1 Cortex-A75 core pipeline



The following table shows the different types of operations that the execution pipelines support.

Table 2-1: Cortex-A75 core supported types of operations

Pipeline mnemonic	Supported functionality
Branch (B)	Branch micro-operations

Pipeline mnemonic	Supported functionality
Integer 0 (I0)	<ul style="list-style-type: none"> Single-cycle integer ALU micro-operations Two-cycle integer shift-ALU micro-operations System register micro-operations Multiply MAC0 CRC Sum-of-absolute-differences micro-operations
Integer 1 (I1)	<ul style="list-style-type: none"> Single-cycle integer ALU micro-operations Two-cycle integer shift-ALU micro-operations MAC1 Divide Sum-of-absolute-differences micro-operations
Load/Store 0/1 (LS)	<ul style="list-style-type: none"> Load Store address micro-operations Special memory micro-operations
Store data (D)	<ul style="list-style-type: none"> Store data micro-operations Register transfer
FP/ASIMD-0 (F0)	<ul style="list-style-type: none"> ASIMD ALU ASIMD miscellaneous ASIMD integer multiply FP convert FP miscellaneous FP add FP multiply Crypto micro-operations
FP/ASIMD-1 (F1)	<ul style="list-style-type: none"> ASIMD ALU ASIMD miscellaneous FP miscellaneous FP add FP multiply FP divide FP sqrt ASIMD shift micro-operations
FP/ASIMD data (FD)	<ul style="list-style-type: none"> ASIMD store data micro-operations FP store data micro-operations

Note:

- Most of branch instructions are decoded in one branch micro-operation and one ALU micro-operation going through I0/I1.
- MAC instructions are decoded in two micro-operations. The first one, MAC0, always goes through I0, and the second one, MAC1, always goes through I1. The ordering is kept between MAC0 and MAC1 in issue cycles.

3 Instruction characteristics

3.1 Instruction tables

This chapter describes high-level performance characteristics for most Armv8-A, Armv8.1-A, and Armv8.2-A A32, T32, and A64 instructions. It includes a series of tables that summarize the effective execution latency and throughput, pipelines used, and special behaviors associated with each group of instructions.

In the following tables:

- *Execution latency*, unless otherwise specified, is defined as the minimum latency seen by an operation dependent on an instruction in the described group.
- *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-A75 micro-architecture.
- *Used pipelines* correspond to the execution pipelines described in Pipeline.

3.2 Branch instructions

Table 3-1: AArch32 branch instructions

Instruction group	AArch32 instructions	Execution latency	Execution throughput	Used pipelines	Notes
Branch, immed	B	1	1	B	-
Branch, register	BX	1	1	I0/I1 + B	-
Branch and link, immed	BL, BLX	1	1	I0/I1 + B	-
Branch and link, register	BLX	1	1	I0/I1 + B	-
Compare and branch	CBZ, CBNZ	1	1	I0/I1 + B	-

Table 3-2: AArch64 branch instructions

Instruction group	AArch64 instructions	Execution latency	Execution throughput	Used pipelines	Notes
Branch, immed	B	1	1	B	-
Branch, register	BR, RET	1	1	I0/I1 + B	-
Branch and link, immed	BL	1	1	I0/I1 + B	-
Branch and link, register	BLR	1	1	I0/I1 + B	-
Compare and branch	CBZ, CBNZ, TBZ, TBNZ	1	1	I0/I1 + B	-

3.3 Arithmetic and logical instructions

Table 3-3: AArch32 arithmetic and logical instructions

Instruction group	AArch32 instructions	Execution latency	Execution throughput	Used pipelines	Notes
ALU, basic	ADD{S}, ADC{S}, ADR,	1	2	I0/I1	-
ALU, shift by immed	AND{S}, BIC{S}, CMN,	2	2	I0/I1	See ¹
ALU, shift by register	CMP, EOR{S}, ORN{S}, ORR{S}, RSB{S}, RSC{S}, SUB{S}, SBC{S}, TEQ, TST	2	1	I0/I1	
ALU, branch forms	-	+2	2	+B	See ²

Table 3-4: AArch64 arithmetic and logical instructions

Instruction group	AArch64 instructions	Execution latency	Execution throughput	Used pipelines	Notes
ALU, basic, include flag setting	ADD{S}, ADC{S}, AND{S}, BIC{S}, EON, EOR, ORN, ORR, SUB{S}, SBC{S}	1	2	I0/I1	-
ALU, extend and/or shift	ADD{S}, AND{S}, BIC{S}, EON, EOR, ORN, ORR, SUB{S}	2	2	I0/I1	See ¹
Conditional compare	CCMN, CCMF	1	2	I0/I1	-
Conditional select	CSEL, CSINC, CSINV, CSNEG	1	2	I0/I1	-

¹ Late forwarding allows having once-cycle latency for back-to-back instructions with dependency on unshifted operands.

² Branch forms are possible when the instruction destination register is the *Program Counter* (PC). In this case, an additional branch micro-operation is required, which adds two cycles to latency.

3.4 Move and shift instructions

Table 3-5: AArch32 move and shift instructions

Instruction group	AArch32 instructions	Execution latency	Execution throughput	Used pipelines	Notes
Move, basic	MOV{S}, MOVW, MOVT, MVN{S}	1	2	I0/I1	See ³
Move, shift by immed	ASR{S}, LSL{S}, LSR{S}, ROR{S}, RRX{S}	1	2	I0/I1	-
MVN, shift by immed	MVN{S}	2	2	I0/I1	-
Move, shift by register	ASR{S}, LSL{S}, LSR{S}, ROR{S}, RRX{S}	1	2	I0/I1	-
MVN, shift by register	MVN{S}	2	1	I0/I1	-
(Move, branch forms)	-	+2	2	+B	See ⁴

Table 3-6: AArch64 move and shift instructions

Instruction group	AArch64 instructions	Execution latency	Execution throughput	Used pipelines	Notes
Address generation	ADR, ADRP	1	2	I0/I1	See ⁵
Move immed	MOVN, MOVK, MOVZ	1	2	I0/I1	See ³
Variable shift	ASRV, LSLV, LSRV, RORV	1	2	I0/I1	-

³ Sequential MOVW/MOVT (AArch32) instruction pairs and some MOVZ/MOVK and MOVK/MOVK (AArch64) instruction pairs can be executed with one-cycle execute latency and four instructions per cycle execution throughput in I0/I1. See IT blocks for more information on the instruction pairs that can be merged.

⁴ Branch forms are possible when the instruction destination register is the *Program Counter* (PC). In this case, an additional branch micro-operation is required, which adds two cycles to latency.

⁵ Sequential ADRP/ADD instruction pairs can be executed with one-cycle execute latency and four-instruction-per-cycle execution throughput in I0/I1. See IT blocks for more information on the instruction pairs that can be merged.

3.5 Saturating and parallel arithmetic instructions

Table 3-7: AArch32 saturating and parallel arithmetic instructions

Instruction group	AArch32 instructions	Execution latency	Execution throughput	Used pipelines	Notes
Parallel arith with exchange, unconditional	SADD16, SADD8, SSUB16, SSUB8, UADD16, UADD8, USUB16, USUB8	2	2	I0/I1	-
Parallel arith with exchange, conditional	SADD16, SADD8, SSUB16, SSUB8, UADD16, UADD8, USUB16, USUB8	2	2	I0/I1	-
Parallel halving arith	SASX, SSAX, UASX, USAX	2	2	I0/I1	-
Parallel halving arith with exchange	SASX, SSAX, UASX, USAX	3	2	I0/I1	-
Parallel saturating arith	SHADD16, SHADD8, SHSUB16, SHSUB8, UHADD16, UHADD8, UHSUB16, UHSUB8	2	2	I0/I1	-
Parallel saturating arith with exchange	SHASX, SHSAX, UHASX, UHSAX	3	2	I0/I1	-
Saturate, basic	QADD16, QADD8, QSUB16, QSUB8, UQADD16, UQADD8, UQSUB16, UQSUB8	1	2	I0/I1	-
Saturate, shift by immed	QASX, QSAX, UQASX, UQSAX	2	2	I0/I1	-
Saturating arith	SSAT, SSAT16, USAT, USAT16	2	2	I0/I1	-
Saturating doubling arith	SSAT, USAT	3	2	I0/I1	-
Parallel arith with exchange, unconditional	QADD, QSUB	2	2	I0/I1	-
Parallel arith with exchange, conditional	QDADD, QDSUB	2	2	I0/I1	-

3.6 Divide and multiply instructions

Table 3-8: AArch32 divide and multiply instructions

Instruction group	AArch32 instructions	Execution latency	Execution throughput	Used pipelines	Notes
Divide	SDIV, UDIV	4 - 12	1/12 - 1/4	I1	See ⁶
Multiply	MUL, SMULBB, SMULBT, SMULTB, SMULTT, SMULWB, SMULWT, SMMUL{R}, SMUAD{X}, SMUSD{X}	3	1	I0	-
Multiply accumulate	MLA, MLS, SMLABB, SMLABT, SMLATB, SMLATT, SMLAWB, SMLAWT, SMLAD{X}, SMLSD{X}, SMMLA{R}, SMMLS{R}	3 (1)	1	I0/I1	See ⁷
Multiply accumulate long	SMLAL, SMLALBB, SMLALBT, SMLALTB, SMLALTT, SMLALD{X}, SMLSLD{X}, UMLAL	4 (2)	1/2	I0/I1	See ⁷ and ⁸
Multiply long	SMULL, UMULL	4	1/2	I0/I1	See ⁸
(Multiply, setflags forms)	-	+3	1/4 - 1/5	+I1	See ⁹

Table 3-9: AArch64 divide and multiply instructions

Instruction group	AArch64 instructions	Execution latency	Execution throughput	Used pipelines	Notes
Divide, W-form	SDIV, UDIV	4 - 12	1/12 - 1/4	I1	See ⁶
Divide, X-form	SDIV, UDIV	4 - 20	1/20 - 1/4	I1	
Multiply accumulate, W-form	MADD, MSUB	3 (1)	1	I0+I1	See ⁷

⁶ Integer divides are performed using an iterative algorithm and block any subsequent divide operations until they are complete. Early termination is possible depending on the data values.

⁷ Multiply-accumulate pipelines support late forwarding of accumulate operands from similar micro-operations, allowing a typical sequence of multiply-accumulate micro-operations to issue one every N cycle (accumulate latency N is shown between brackets).

⁸ Long-form multiplies, which produce two result registers, stall the multiplier pipeline for one extra cycle.

⁹ Multiplies that set the condition flags require three additional cycles.

Multiply accumulate, X-form, non-null Rm most significant 32 bits	MADD, MSUB	5 (3)	1/3	0+ 1	See ⁷ and ¹⁰
Multiply accumulate, X-form, null Rm most significant 32 bits	MADD, MSUB	4 (2)	1/2	0+ 1	
Multiply accumulate long	SMADDL, SMSUBL, UMADDL, UMSUBL	3 (1)	1	0+ 1	See ⁷
Multiply high	SMULH, UMULH	6 (4)	1/4	0+ 1	See ¹¹

¹⁰ X-form multiply accumulates stall the multiplier pipeline for two extra cycles, except if the Rm highest 32 bits are zero (four-cycle latency instead of five-cycle latency).

¹¹ Multiply-high operations stall the multiplier pipeline for N extra cycles before any other M-type micro-operation can be issued to that pipeline (N is shown between brackets).

3.7 Miscellaneous data-processing instructions

Table 3-10: AArch32 miscellaneous data-processing instructions

Instruction group	AArch32 instructions	Execution latency	Execution throughput	Used pipelines	Notes
Bit field extract	SBFX, UBFX	1	2	I0/I1	-
Bit field insert/clear	BFI, BFC	1	2	I0/I1	-
Count leading zeros	CLZ	2	2	I0/I1	-
Pack halfword	PKH	2	2	I0/I1	-
Reverse bits/bytes	RBIT, REV, REV16, REVSH	1	2	I0/I1	-
Select bytes, unconditional	SEL	1	2	I0/I1	-
Sign/zero extend, normal	SXTB, SXTB, UXTB, UXTB	1	2	I0/I1	See ¹²
Sign/zero extend, parallel	SXTB16, UXTB16	1	2	I0/I1	
Sign/zero extend and add, normal	SXTAB, SXTAB, UXTAB, UXTAB	2	1	I0/I1	
Sign/zero extend and add, parallel	SXTAB16, UXTAB16	2	1	I0/I1	
Sum of absolute differences	USAD8, USADA8	3	1	I0+I1	-

Table 3-11: AArch64 miscellaneous data-processing instructions

Instruction group	AArch64 instructions	Execution latency	Execution throughput	Used pipelines	Notes
Bitfield extract, one reg	EXTR	1	2	I0/I1	-
Bitfield extract, two regs	EXTR	2	2	I0/I1	-
Bitfield move	BFM, SBFM, UBFM	1	2	I0/I1	-

¹² Instruction is decoded as two micro-operations.

Count leading	CLS, CLZ	2	2	I0/I1	-
Reverse bits/bytes	RBIT, REV, REV16, REV32	1	2	I0/I1	-

3.8 Load instructions

Latencies shown in the following table assume that memory access hits in the Level 1 data cache.

Notes:

- All forms of load that imply write back of the baser register also require a micro-operation that makes use of the I0/I1 pipeline, which is not shown in the following tables.
- The Cortex-A75 core can return two registers (both X-form and W-form) per cycle, sustained. In a single cycle, it can return four registers (both forms) from a maximum of two micro-operations.
- Load instructions with execution latency of four cycles and returning a single register might expose latency of three cycles if:
 - It executes in Load/Store unit 0.
 - The consuming instruction is either a data processing instruction or a load or store instruction, which depends on the load instruction for the base address register.

Table 3-12: AArch32 load instructions

Instruction group	AArch32 instructions	Execution latency	Execution throughput	Used pipelines	Notes
Load, immed offset	LDR{ <i>T</i> }, LDRB{ <i>T</i> }, LDRD, LDRH{ <i>T</i> }, LDRSB{ <i>T</i> }, LDRSH{ <i>T</i> }	4	2	LS	-
Load, register offset, plus	LDR, LDRB, LDRD, LDRH, LDRSB, LDRSH	4	2	LS	-
Load, register offset, minus	LDR, LDRB, LDRD, LDRH, LDRSB, LDRSH	4	2	LS	-
Load, scaled register offset, plus, scale by 4/8	LDR, LDRB	4	2	LS	-
Load, scaled register offset, other	LDR, LDRB, LDRH, LDRSB, LDRSH	5	1	LS	See ¹³
Load, immed pre-indexed	LDR, LDRB, LDRD, LDRH, LDRSB, LDRSH	4	2	LS	-
Load, register pre-indexed, shift Rm, plus and minus	LDR, LDRB, LDRH, LDRSB, LDRSH	5	1	LS	See ¹³
Load, register pre-indexed	LDRD	4	2	LS	-

¹³ These instructions iterate two cycles in the load/store pipeline. Two of these instructions can be dispatched at the same cycle to Load/Store 0 and Load/Store 1, however the sustained throughput is one every other cycle on each LS.

Load, register pre-indexed, cond	LDRD	4	2	LS	-
Load, scaled register pre-indexed, plus, scale by 4/8	LDR, LDRB	4	2	LS	-
Load, scaled register pre-indexed, unshifted	LDR, LDRB	4	2	LS	-
Load, immed post-indexed	LDR{T}, LDRB{T}, LDRD, LDRH{T}, LDRSB{T}, LDRSH{T}	4	2	LS	-
Load, register post-indexed	LDR, LDRB, LDRH{T}, LDRSB{T}, LDRSH{T}	4	2	LS	-
Load, register post-indexed	LDRD	4	2	LS	-
Load, register post-indexed	LDRT, LDRBT	4	2	LS	-
Load, scaled register post-indexed	LDR, LDRB	4	2	LS	-
Load, scaled register post-indexed	LDRT, LDRBT	4	2	LS	-
Preload, all forms	PLD, PLDW	4	1	IO	-
Load multiple, no writeback, base reg not in list	LDMIA, LDMIB, LDMDA, LDMDB	N	2/R	LS	See ¹⁴
Load multiple, no writeback, base reg in list	LDMIA, LDMIB, LDMDA, LDMDB	N	2/R	LS	
Load multiple, writeback	LDMIA, LDMIB, LDMDA, LDMDB, POP	N	2/R	LS	
(Load, all branch forms)	-	+1	-	+ B	See ¹⁵

Table 3-13: AArch64 load instructions

Instruction group	AArch64 instructions	Execution latency	Execution throughput	Used pipelines	Notes
Load register, literal	LDR, LDRSW	4	2	LS	-
Load register, unscaled immed	LDUR, LDURB, LDURH, LDURSB, LDURSH, LDURSW	4	2	LS	-

¹⁴ N is floor((num_reg+3)/4) and R is floor((num_reg +1)/2).¹⁵ Branch forms are possible when the instruction destination register is the *Program Counter* (PC). In this case, an additional branch micro-operation is required, which adds one cycle to latency.

Load register, immed post- index	LDR, LDRB, LDRH, LDRSB, LDRSH, LDRSW	4	2	LS	-
Load register, immed pre- index	LDR, LDRB, LDRH, LDRSB, LDRSH, LDRSW	4	2	LS	-
Load register, immed unprivileged	LDTR, LDTRB, LDTRH, LDTRSB, LDTRSH, LDTRSW	4	2	LS	-
Load register, unsigned immed	LDR, LDRB, LDRH, LDRSB, LDRSH, LDRSW	4	2	LS	-
Load register, register offset, basic	LDR, LDRB, LDRH, LDRSB, LDRSH, LDRSW	4	2	LS	-
Load register, register offset, scale by 4/8	LDR, LDRSW	4	2	LS	-
Load register, register offset, scale by 2	LDRH, LDRSH	5	1	LS	See ¹³
Load register, register offset, extend	LDR, LDRB, LDRH, LDRSB, LDRSH, LDRSW	4	2	LS	-
Load register, register offset, extend, scale by 4/8	LDR, LDRSW	4	2	LS	-
Load register, register offset, extend, scale by 2	LDRH, LDRSH	5	1	LS	See ¹³
Load pair, signed immed offset, normal, W-form	LDP, LDNP	4	2	LS	-
Load pair, signed immed offset, normal, X-form	LDP, LDNP	5	1	LS	See ¹³
Load pair, signed immed offset, signed words, base != SP	LDPSW	4	1	LS	See ¹⁶
Load pair, signed immed offset, signed words, base = SP	LDPSW	4	1	LS	
Load pair, immed post-index, normal	LDP	5	1	LS	See ¹³
Load pair, immed post-index, signed words	LDPSW	4	1	LS	See ¹⁶
Load pair, immed pre-index, normal	LDP	5	1	LS	See ¹³

¹⁶ These instructions are split into two micro-operations which can be sent to both Load/Store units. If both micro-operations are dispatched at the same cycle, then execution latency is four cycles. If only one Load/Store unit is available, then latency is five cycles.

Load pair, immed pre-index, signed words	LDPSW	4	1	LS	See ¹⁶
Preload, all forms	PRFM, PRFUM	4	1	10	-

3.9 Store instructions

The following tables describe performance characteristics for standard store instructions. Store micro-operations can issue after their address operands become available and do not need to wait for data operands. Once executed, stores are buffered and committed in the background.

Note:

The Cortex-A75 core features two store units for address generation and one store data unit. This is shown in the following tables with the micro-operations throughput provided for both address and data in the Execution throughput column.

Table 3-14: AArch32 store instructions

Instruction group	AArch32 instructions	Execution latency	Execution throughput (Store address/ store data)	Used pipelines	Notes
Store, immed offset	STR{T}, STRB{T}, STRD, STRH{T}	1	2/1	LS, D	-
Store, register offset, plus	STR, STRB, STRD, STRH	1	2/1	LS, D	-
Store, register offset, minus	STR, STRB, STRD, STRH	1	2/1	LS, D	-
Store, register offset, no shift, plus	STR, STRB	1	2/1	LS, D	-
Store, scaled register offset, plus LSL2, LSL3	STR, STRB	1	2/1	LS, D	-
Store, scaled register offset, other	STR, STRB	2	1/1	LS, D	See ¹⁷
Store, scaled register offset, minus	STR, STRB	2	1/1	LS, D	
Store, immed pre-indexed	STR, STRB, STRD, STRH	1	2/1	LS, D	-
Store, register pre-indexed, plus, no shift	STR, STRB, STRD, STRH	1	2/1	LS, D	-
Store, register pre-indexed, minus	STR, STRB, STRD, STRH	2	1/1	LS, D	See ¹⁷

¹⁷ These instructions iterate two cycles in the load/store pipeline. Two of these instructions can be dispatched at the same cycle to Load/Store 0 and Load/Store 1, however the sustained throughput is one every other cycle on each LS.

Store, scaled register pre-indexed, plus LSL2, LSL3	STR, STRB	1	2/1	LS, D	-
Store, scaled register pre-indexed, other	STR, STRB	2	1/1	LS, D	See ¹⁷
Store, immed post-indexed	STR{T}, STRB{T}, STRD, STRH{T}	1	2/1	LS, D	-
Store, register post-indexed	STRH{T}, STRD	1	2/1	LS, D	-
Store, register post-indexed	STR{T}, STRB{T}	1	2/1	LS, D	-
Store, scaled register post-indexed	STR{T}, STRB{T}	1	2/1	LS, D	-
Store multiple, no writeback	STMIA, STMIB, STMDA, STMDB	N	1/N	LS, D	See ¹⁸
Store multiple, writeback	STMIA, STMIB, STMDA, STMDB, PUSH	N	1/N	LS, D	

Table 3-15: AArch64 store instructions

Instruction group	AArch64 instructions	Execution latency	Execution throughput	Used pipelines	Notes
Store register, unscaled immed	STUR, STURB, STURH	1	2/1	LS, D	-
Store register, immed post-index	STR, STRB, STRH	1	2/1	LS, D	-
Store register, immed pre-index	STR, STRB, STRH	1	2/1	LS, D	-
Store register, immed unprivileged	STTR, STTRB, STTRH	1	2/1	LS, D	-
Store register, unsigned immed	STR, STRB, STRH	1	2/1	LS, D	-
Store register, register offset, basic	STR, STRB, STRH	1	2/1	LS, D	-
Store register, register offset, scaled by 4/8	STR	1	2/1	LS, D	-
Store register, register offset, scaled by 2	STRH	2	1/1	LS, D	See ¹⁷
Store register, register offset, extend	STR, STRB, STRH	1	2/1	LS, D	-
Store register, register offset, extend, scale by 4/8	STR	1	2/1	LS, D	-

¹⁸ For store multiple instructions, $N = \text{floor}((\text{num_regs} + 3) / 4)$.

Store register, register offset, extend, scale by 2	STRH	2	1/1	LS, D	See ¹⁷
Store pair, immed offset, W-form	STP, STNP	1	2/1	LS, D	-
Store pair, immed offset, X-form	STP, STNP	1	2/1	LS, D	-
Store pair, immed post-index, W- form	STP	1	2/1	LS, D	-
Store pair, immed post-index, X- form	STP	1	2/1	LS, D	-
Store pair, immed pre-index, W- form	STP	1	2/1	LS, D	-
Store pair, immed pre-index, X- form	STP	1	2/1	LS, D	-

3.10 Floating-point data processing instructions

Table 3-16: AArch32 floating-point data processing instructions

Instruction group	AArch32 instructions	Execution latency	Execution throughput	Used pipelines	Notes
FP absolute value	VABS	2	2	F0/F1	-
FP arith	VADD, VSUB	3	2	F0/F1	See ¹⁹
FP compare	VCMP, VCMPE	4	1	F0	See ²⁰
FP compare and write flags	VCMP, VCMPE followed by VMRS APSR_nzcv, FPSCR	6	1	F0	See ²¹
FP convert	VCVT{R}, VCVTb, VCVTT, VCVTA, VCVTM, VCVTN, VCVTP	3	1	F0	-
FP round to integral	VRINTA, VRINTM, VRINTN, VRINTP, VRINTR, VRINTX, VRINTZ	3	1	F0	-
FP divide, H-form	VDIV	6-8	1/4-1/3	F1	See ²²
FP divide, S-form	VDIV	6-10	1/5-1/3	F1	
FP divide, D-form	VDIV	6-15	1/15-1/6	F1	
FP max/min	VMAXNM, VMINNM	3	2	F0/F1	
FP multiply	VMUL, VNMUL	3	2	F0/F1	See ¹⁹ and ²³
FP multiply non-fused accumulate	VMLA, VMLS, VNMLA, VNMLS	6 (3)	2	F0/F1	See ¹⁹ and ²⁴

¹⁹ FP add and multiply pipelines use 2.5 cycles to calculate the results, which allows 0-cycle forward. Execution latency can reach three only with 0-cycle forward. Similarly, the combined multiply-accumulate pipelines would have one more cycle in execution latency without 0-cycle forward.

²⁰ Latency corresponds to FPSCR flags forward to a VMRS APSR_nzcv, FPSCR instruction.

²¹ Latency corresponds to the sequence FCMP, VMRS APSR_nzcv, FPSCR to a conditional instruction.

²² FP divide and square root operations are performed using an iterative algorithm and block subsequent similar operations to the same pipeline until complete. The minimum execution latency and maximum execution throughput are achieved by power-of-two early termination. In a normal case, the minimum execution latency is 6 for H-form, 8 for S-form, and 13 for D-form, and the maximum execution throughput is 1/3 for H-form, 1/4 for S-form, and 1/13 for D-form.

²³ FP multiply-accumulate pipelines support late forwarding of the result from FP multiply micro-operations to the accumulate operands of an FP multiply-accumulate micro-operation. The latter can be issued one cycle after the FP multiply micro-operation is issued.

²⁴ FP multiply-accumulate pipelines support late forwarding of accumulate operands from similar micro-operations, allowing a typical sequence of multiply-accumulate micro-operations to issue one every N cycles (accumulate latency N is shown between brackets).

FP multiply fused accumulate	VFMA, VFMS, VFNMA, VFNMS	5 (3)	2	F0/F1	See ¹⁹ and ²⁴
FP negate	VNEG	2	2	F0/F1	-
FP select	VSELEQ, VSELGE, VSELGT, VSELVS	2	2	F0/F1	-
FP square root, H-form	VSQRT	6-7	2/7-1/3	F1	See ²²
FP square root, S-form	VSQRT	6-11	2/11-1/3	F1	
FP square root, D-form	VSQRT	6-18	1/18-1/6	F1	

Table 3-17: AArch64 floating-point data processing instructions

Instruction group	AArch64 instructions	Execution latency	Execution throughput	Used pipelines	Notes
FP absolute value	FABS	2	2	F0/F1	-
FP arithmetic	FADD, FSUB	3	2	F0/F1	See ¹⁹
FP compare	FCCMP{E}, FCMP{E}	3	1	F0	-
FP divide, H-form	FDIV	6-8	1/4-1/3	F1	See ²²
FP divide, S-form	FDIV	6-10	1/5-1/3	F1	
FP divide, D-form	FDIV	6-15	1/15-1/6	F1	
FP min/max	FMIN, FMINNM, FMAX, FMAXNM	3	2	F0/F1	-
FP multiply	FMUL, FNMUL	3	2	F0/F1	See ¹⁹ and ²³
FP multiply accumulate	FMADD, FMSUB, FNMADD, FNMSUB	5 (3)	2	F0/F1	See ¹⁹ and ²⁴
FP negate	FNEG	2	2	F0/F1	-
FP round to integral	FRINTA, FRINTI, FRINTM, FRINTN, FRINTP, FRINTX, FRINTZ	3	1	F0	-
FP select	FCSEL	2	2	F0/F1	-
FP square root, H-form	FSQRT	6-7	2/7-1/3	F1	See ²²
FP square root, S-form	FSQRT	6-11	2/11-1/3	F1	
FP square root, D-form	FSQRT	6-18	1/18-1/6	F1	

3.11 Floating-point miscellaneous instructions

Table 3-18: AArch32 floating-point miscellaneous instructions

Instruction group	AArch32 instructions	Execution latency	Execution throughput	Used pipelines	Notes
FP move, immed	VMOV	3	2	F0/F1	-
FP move, register	VMOV	3	2	F0/F1	-
FP move, extraction or insertion	VMOVX, VINS	3	2	F0/F1	-
FP transfer, vfp to core reg	VMOV	3	1	F0	-
FP transfer, core reg to upper or lower half of vfp D-reg	VMOV	4	1	LS0, F0/F1	-

Table 3-19: AArch64 floating-point miscellaneous instructions

Instruction group	AArch64 instructions	Execution latency	Execution throughput	Used pipelines	Notes
FP transfer, core reg to vfp	VMOV	4	1	LS0	-
FP convert, from vec to vec reg	FCVT, FCVTXN	3	1	F0	-
FP convert, from gen to vec reg	SCVTF, UCVTF	6	1	LS0, F0	-
FP convert, from vec to gen reg	FCVTAS, FCVTAU, FCVTMS, FCVTMU, FCVTNS, FCVTNU, FCVTPS, FCVTPU, FCVTZS, FCVTZU	5	1	F0	-
FP move, immed	FMOV	3	2	F0/F1	-
FP move, register	FMOV	3	2	F0/F1	-
FP transfer, from gen to vec reg	FMOV	4	1	LS0	-
FP transfer, from vec to gen reg	FMOV	4	1	F0	-

3.12 Floating-point load instructions

The latencies shown assume that 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.

Latencies also assume that 64-bit element loads are 64-bit aligned. If this is not the case, an extra cycle is required.

Table 3-20: AArch32 floating-point load instructions

Instruction group	AArch32 instructions	Execution latency	Execution throughput	Used pipelines	Notes
FP load, register, unconditional	VLDR	5	2	LS0/LS1	-
FP load, register, conditional	VLDR	5	2	LS0/LS1 F0/F1	-
FP load multiple, unconditional	VLDmia, VLDMDB, VPOP	4 + N	2/N	LS0/LS1	See ²⁵
FP load multiple, conditional	VLDmia, VLDMDB, VPOP	4 + N	2/N	LS0/LS1 F0/F1	See ²⁶
(FP load, writeback forms)	-	(1)	Same as before	+IO/I1	See ²⁷

Table 3-21: AArch64 floating-point load instructions

Instruction group	AArch64 instructions	Execution latency	Execution throughput	Used pipelines	Notes
Load vector reg, literal	LDR	5	2	LS0/LS1	-
Load vector reg, unscaled immed	LDUR	5	2	LS0/LS1	-
Load vector reg, immed post-index	LDR	5 (1)	2	LS0/LS1 IO/I1	See ²⁷
Load vector reg, immed pre-index	LDR	5 (1)	2	LS0/LS1 IO/I1	
Load vector reg, unsigned immed	LDR	5	2	LS0/LS1	-

²⁵ N=num_regs for Double-precision registers and N=floor((num_regs+1)/2) for Single-precision registers.

²⁶ This is assuming that the condition is resolved maximum once cycle after the Issue stage.

²⁷ Writeback forms of load instructions require an extra micro-operation to update the base address. This update is typically performed in parallel with or prior to the load micro-operation (update latency is shown between brackets).

Load vector reg, register offset, basic	LDR	5	2	LS0/LS1	-
Load vector reg, register offset, scale, S/D-form	LDR	4	2	LS0/LS1	-
Load vector reg, register offset, scale, H-form	LDR	6	1	LS0/LS1 IO/I1	-
Load vector reg, register offset, scale, Q-form	LDR	7	1	LS0/LS1 IO/I1	-
Load vector reg, register offset, extend	LDR	5	2	LS0/LS1	-
Load vector reg, register offset, extend, scale, S/D- form	LDR	5	2	LS0/LS1	-
Load vector reg, register offset, extend, scale, H- form	LDR	6	1	LS0/LS1 IO/I1	-
Load vector reg, register offset, extend, scale, Q- form	LDR	7	1	LS0/LS1 IO/I1	-
Load vector pair, immed offset, S-form	LDP, LDNP	5	2	L	-
Load vector pair, immed offset, D-form	LDP, LDNP	6	1	L	-
Load vector pair, immed offset, Q-form	LDP, LDNP	6	1/2	L	-
Load vector pair, immed post-index, S-form	LDP	5 (1)	2	LS0/LS1 IO/I1	See ²⁷
Load vector pair, immed post-index, D-form	LDP	6 (1)	1	LS0/LS1 IO/I1	
Load vector pair, immed post-index, Q-form	LDP	6 (1)	1/2	LS0/LS1 IO/I1	
Load vector pair, immed pre-index, S-form	LDP	5 (1)	1	LS0/LS1 IO/I1	
Load vector pair, immed pre-index, D-form	LDP	6 (1)	1	LS0/LS1 IO/I1	
Load vector pair, immed pre-index, Q-form	LDP	6 (1)	1/2	LS0/LS1 IO/I1	

3.13 Floating-point store instructions

Stores micro-operations can issue after their address operands become available and do not need to wait for data operands. After they are executed, stores are buffered and committed in the background.

Table 3-22: AArch32 floating-point store instructions

Instruction group	AArch32 instructions	Execution latency	Execution throughput	Used pipelines	Notes
FP store, immed offset	VSTR	1	2	LS0/LS1	-
FP store multiple, S-form	VSTMIA, VSTMDB, VPUSH	N	2/N	LS0/LS1	See ²⁸
FP store multiple, D-form	VSTMIA, VSTMDB, VPUSH	N	2/N	LS0/LS1	See ²⁹
(FP store, writeback forms)	-	(1)	Same as before	+I0/I1	See ³⁰

Table 3-23: AArch64 floating-point store instructions

Instruction group	AArch64 instructions	Execution latency	Execution throughput	Used pipelines	Notes
Store vector reg, unscaled immed, B/H/S/D-form	STUR	1	2	LS0/LS1	-
Store vector reg, unscaled immed, Q-form	STUR	2	1	LS0/LS1	-
Store vector reg, immed post- index, B/H/S/D-form	STR	1 (1)	2	LS0/LS1, I0/I1	See ³⁰
Store vector reg, immed post- index, Q-form	STR	2 (1)	1	LS0/LS1, I0/I1	
Store vector reg, immed pre- index, B/H/S/D-form	STR	1 (1)	2	S, I0/I1	
Store vector reg, immed pre- index, Q-form	STR	2 (1)	1	LS0/LS1	
Store vector reg, unsigned immed, B/H/S/D-form	STR	1	2	LS0/LS1	-

²⁸ $N = \text{floor}((\text{num_regs} + 1) / 2)$.

²⁹ $N = (\text{num_regs})$.

³⁰ Writeback forms of store instructions require an extra micro-operation to update the base address. This update is typically performed in parallel with, or prior to, the store micro-operation (address update latency is shown between brackets).

Store vector reg, unsigned immed, Q-form	STR	2	1	IO/I1, LS0/LS1	-
Store vector reg, register offset, basic, B/H/S/D-form	STR	1	2	LS0/LS1	-
Store vector reg, register offset, basic, Q-form	STR	2	1	LS0/LS1, IO/I1	-
Store vector reg, register offset, scale, H-form	STR	2	1	LS0/LS1, IO/I1	-
Store vector reg, register offset, scale, S/D-form	STR	1	2	LS0/LS1	-
Store vector reg, register offset, scale, Q-form	STR	2	1	LS0/LS1, IO/I1	-
Store vector reg, register offset, extend, B/H/S/D-form	STR	1	2	LS0/LS1	-
Store vector reg, register offset, extend, Q-form	STR	2	1	LS0/LS1	-
Store vector reg, register offset, extend, scale, H-form	STR	2	1	LS0/LS1	-
Store vector reg, register offset, extend, scale, S/D-form	STR	1	2	LS0/LS1	-
Store vector reg, register offset, extend, scale, Q-form	STR	2	1	LS0/LS1, IO/I1	-
Store vector pair, immed offset, S-form	STP	1	2	LS0/LS1	-
Store vector pair, immed offset, D-form	STP	2	1	LS0/LS1	-
Store vector pair, immed offset, Q-form	STP	2	1/2	LS0/LS1, IO/I1	-
Store vector pair, immed post- index, S-form	STP	1 (1)	2	LS0/LS1, IO/I1	See ³⁰
Store vector pair, immed post- index, D-form	STP	2 (1)	1	LS0/LS1, IO/I1	
Store vector pair, immed post- index, Q-form	STP	2 (1)	1/2	LS0/LS1, IO/I1	
Store vector pair, immed pre- index, S-form	STP	1 (1)	2	LS0/LS1, IO/I1	
Store vector pair, immed pre- index, D-form	STP	2 (1)	1	LS0/LS1, IO/I1	

Store vector pair, immed pre- index, Q-form	STP	2 (1)	1/2	LS0/LS1, IO/I1	
---	-----	-------	-----	----------------	--

3.14 Advanced SIMD integer instructions

Table 3-24: AArch32 advanced SIMD integer instructions

Instruction group	AArch32 instructions	Execution latency	Execution throughput	Used pipelines	Notes
ASIMD absolute diff, D-form	VABD	3	2	F0/F1	-
ASIMD absolute diff, Q-form	VABD	3	3/2	F0/F1	-
ASIMD absolute diff accum, D-form	VABA	4 (1)	2	F0/F1	See ³¹
ASIMD absolute diff accum, Q-form	VABA	4 (1)	3/2	F0/F1	
ASIMD absolute diff accum long	VABAL	4 (1)	3/2	F0/F1	
ASIMD absolute diff long	VABDL	3	3/2	F0/F1	-
ASIMD arith, basic, D-form	VADD, VNEG, VPADD, VSUB	3	2	F0/F1	-
ASIMD arith, basic, Q-form	VADD, VNEG, VPADD, VSUB	3	3/2	F0/F1	-
ASIMD arith, basic, long or wide	VADDL, VADDW, VSUBL, VSUBW	3	3/2	F0/F1	-
ASIMD arith, Vector Pairwise Add Long, D-form	VPADDL	3	2	F0/F1	-
ASIMD arith, Vector Pairwise Add Long, Q-form	VPADDL	3	3/2	F0/F1	-
ASIMD arith, complex, D-form	VABS, VHADD, VHSUB, VQABS, VQADD, VQNEG, VQSUB, VRHADD	3	2	F0/F1	-
ASIMD arith, complex, Q-form	VABS, VHADD, VHSUB, VQABS, VQADD, VQNEG, VQSUB, VRADDHN, VRHADD, VRSUBHN	3	3/2	F0/F1	-
ASIMD arith, complex, narrow	VADDHN, VRADDHN, VRSUBHN, VSUBHN	3	2	F0/F1	-

³¹ Other accumulate pipelines also support late forwarding of accumulate operands from similar micro-operations, allowing a typical sequence of such micro-operations to issue one every cycle (accumulate latency is shown between brackets).

ASIMD compare, D-form	VCEQ, VCGE, VCGT, VCLE, VTST	3	2	F0/F1	-
ASIMD compare, Q-form	VCEQ, VCGE, VCGT, VCLE, VTST	3	3/2	F0/F1	-
ASIMD logical, D-form	VAND, VBIC, VMVN, VORR, VORN, VEOR	3	2	F0/F1	-
ASIMD logical, Q-form	VAND, VBIC, VMVN, VORR, VORN, VEOR	3	3/2	F0/F1	-
ASIMD max/min, D-form	VMAX, VMIN, VPMAX, VPMIN	3	2	F0/F1	-
ASIMD max/min, Q-form	VMAX, VMIN, VPMAX, VPMIN	3	3/2	F0/F1	-
ASIMD multiply, D-form	VMUL, VQDMULH, VQDMULH	4	1	F0	-
ASIMD multiply, Q-form	VMUL, VQDMULH, VQDMULH,	5	1/2	F0	-
ASIMD multiply accumulate, D- form	VMLA, VMLS	4 (1)	1	F0	See ³²
ASIMD multiply accumulate, Q- form	VMLA, VMLS	5 (2)	1/2	F0	
ASIMD multiply accumulate long	VMLAL, VMLSL	4 (1)	1	F0	
ASIMD multiply accumulate saturating long	VQDMLAL, VQDMLSL	4 (2)	1	F0	
ASIMD multiply long	VMULL.S, VMULL.I, VQDMULL	4	1	F0	-
ASIMD multiply long, polynomial	VMULL.P8	3	1	F0	-
ASIMD pairwise add and accumulate	VPADAL	4 (1)	1	F1	See ³¹
ASIMD rounding double multiply accumulate, D-form	VQRDMLAH, VQRDMLSH	4	1	F0	-
ASIMD rounding double multiply accumulate, Q-form	VQRDMLAH, VQRDMLSH	5	1/2	F0	-
ASIMD shift accumulate	VSRA, VRSRA	4 (1)	1	F1	See ³¹
ASIMD shift by immed, basic	VMOVL, VSHL, VSHLL, VSHR, VSHRN	3	1	F1	-

³² Multiply-accumulate pipelines support late forwarding of accumulate operands from similar micro-operations, allowing a typical sequence of integer multiply-accumulate micro-operations to issue one every cycle or one every other cycle (accumulate latency is shown between brackets, and might be further limited to throughput according to destination register size).

ASIMD shift by immed, complex	VQSRHN, VQSRHN, VQSHL{U}, VQSHRN, VQSHRUN, VRSHR, VRSHRN	4	1	F1	-
ASIMD shift by immed and insert, basic, D-form	VSLI, VSRI	3	1	F1	-
ASIMD shift by immed and insert, basic, Q-form	VSLI, VSRI	4	1/2	F1	-
ASIMD shift by register, basic, D- form	VSHL	3	1	F1	-
ASIMD shift by register, basic, Q- form	VSHL	4	1/2	F1	-
ASIMD shift by register, complex, D-form	VQSHL, VQSHL, VRSHL	4	1	F1	-
ASIMD shift by register, complex, Q-form	VQSHL, VQSHL, VRSHL	5	1/2	F1	-

Table 3-25: AArch64 advanced SIMD integer instructions

Instruction group	AArch64 instructions	Execution latency	Execution throughput	Used pipelines	Notes
ASIMD absolute diff, D-form	SABD, UABD	3	2	F0/F1	-
ASIMD absolute diff, Q-form	SABD, UABD	3	2	F0/F1	-
ASIMD absolute diff accum, D-form	SABA, UABA	4 (1)	2	F0/F1	See ³¹
ASIMD absolute diff accum, Q-form	SABA, UABA	5 (2)	3/2	F0/F1	
ASIMD absolute diff accum long	SABAL (2) , UABAL (2)	4 (1)	3/2	F0/F1	
ASIMD absolute diff long	SABDL, UABDL	3	3/2	F0/F1	-
ASIMD arith, basic, D-form	ABS, ADD, ADDP, NEG, SADDLP, SHADD, SHSUB, SUB, UADDLP, UHADD, UHSUB	3	2	F0/F1	-
ASIMD arith, basic, Q-form	ABS, ADD, ADDP, NEG, SADDLP, SHADD, SHSUB, SUB, UADDLP, UHADD, UHSUB	3	3/2	F0/F1	-

ASIMD arith, basic, long or wide	SADDL (2) , SADDW (2) , SSUBL (2) , SSUBW (2) , UADDL (2) , UADDW (2) , USUBL (2) , USUBW (2)	3	2	F0/F1	-
ASIMD arith, complex, D-form	SQABS, SQADD, SQNEG, SQSUB, SRHADD, SUQADD, UQADD, UQSUB, URHADD, USQADD	3	2	F0/F1	-
ASIMD arith, complex, Q-form	SQABS, SQADD, SQNEG, SQSUB, SRHADD, SUQADD, UQADD, UQSUB, URHADD, USQADD	3	3/2	F0/F1	-
ASIMD arith, complex, narrow	ADDHN (2) , RADDHN (2) , RSUBHN (2) , SUBHN (2)	3	2	F0/F1	-
ASIMD arith, reduce, 4H/4S	SADDLV, UADDLV	3	1	F1	-
ASIMD arith, reduce, 8B/8H	SADDLV, UADDLV	6	1	F1, F0/F1	-
ASIMD arith, reduce, 16B	SADDLV, UADDLV	6	1/2	F1	-
ASIMD compare, D-form	CMEQ, CMGE, CMGT, CMHI, CMHS, CMLE, CMLT, CMTST	3	2	F0/F1	-
ASIMD compare, Q-form	CMEQ, CMGE, CMGT, CMHI, CMHS, CMLE, CMLT, CMTST	3	3/2	F0/F1	-
ASIMD logical, D-form	AND, BIC, EOR, MOV, MVN, ORN, ORR	3	2	F0/F1	-
ASIMD logical, Q-form	AND, BIC, EOR, MOV, MVN, ORN, ORR	3	3/2	F0/F1	-
ASIMD max/min, basic, D-form	SMAX, SMAXP, SMIN, SMINP, UMAX, UMAXP, UMIN, UMINP	3	2	F0/F1	-
ASIMD max/min, basic, Q-form	SMAX, SMAXP, SMIN, SMINP, UMAX, UMAXP, UMIN, UMINP	3	3/2	F0/F1	-
ASIMD max/min, reduce, 4H/4S	SMAXV, SMINV, UMAXV, UMINV	3	1	F1	-
ASIMD max/min, reduce, 8B/8H	SMAXV, SMINV, UMAXV, UMINV	6	1	F1, F0/F1	-

ASIMD max/min, reduce, 16B	SMAV, SMINV, UMAXV, UMINV	6	1/2	F1	-
ASIMD multiply, D-form	MUL, PMUL, SQDMULH, SQRDMULH	4	1	F0	-
ASIMD multiply, Q-form	MUL, PMUL, SQDMULH, SQRDMULH	5	1/2	F0	-
ASIMD multiply accumulate, D- form	MLA, MLS	4 (1)	1	F0	See ³²
ASIMD multiply accumulate, Q- form	MLA, MLS	5 (2)	1/2	F0	
ASIMD multiply accumulate long	SMLAL (2) , SMLSL (2) , UMLAL (2) , UMLSL (2)	4 (1)	1	F0	
ASIMD multiply accumulate saturating long	SQDMLAL (2) , SQDMLSL (2)	4 (2)	1	F0	
ASIMD multiply long	SMULL (2) , UMULL (2) , SQDMULL (2)	4	1	F0	-
ASIMD rounding double multiply accumulate, D-form	SQRDMLAH, SQRDMLSH	4	1	F0	-
ASIMD rounding double multiply accumulate, Q-form	SQRDMLAH, SQRDMLSH	5	1/2	F0	-
ASIMD polynomial (8x8) multiply long	PMULL.8B, PMULL2.16B	3	1	F0	See ³³
ASIMD pairwise add and accumulate	SADALP, UADALP	4 (1)	1	F1	See ³¹
ASIMD shift accumulate	SRA, SRSRA, USRA, URSRA	4 (1)	1	F1	
ASIMD shift by immed, basic	SHL, SHLL (2) , SHRN (2) , SLI, SRI, SSHLL (2) , SSHR, SXTL (2) , USHLL (2) , USHR, UXTL (2)	3	1	F1	-
ASIMD shift by immed and insert, basic, D-form	SLI, SRI	3	1	F1	-
ASIMD shift by immed and insert, basic, Q-form	SLI, SRI	4	1/2	F1	-
ASIMD shift by immed, complex	RSHRN (2) , RSRSHR,	4	1	F1	-

³³ This category includes instructions of the form PMULL Vd.8H, Vn.8B, Vm.8B and PMULL2 Vd.8H, Vn.16B, Vm.16B.

	SQSHL{U}, SQRSHRN(2), SQRSHRUN(2), SQSHRN(2), SQSHRUN(2), URSHR, UQSHL, UQRSHRN(2), UQSHRN(2)				
ASIMD shift by register, basic, D-form	SSHL, USHL	3	1	F1	-
ASIMD shift by register, basic, Q-form	SSHL, USHL	4	1/2	F1	-
ASIMD shift by register, complex, D-form	SRSHL, SQRSHL, SQSHL, URSHL, UQRSHL, UQSHL	4	1	F1	-
ASIMD shift by register, complex, Q-form	SRSHL, SQRSHL, SQSHL, URSHL, UQRSHL, UQSHL	5	1/2	F1	-

3.15 Advanced SIMD floating-point instructions

Table 3-26: AArch32 advanced SIMD floating-point instructions

Instruction group	AArch32 instructions	Execution latency	Execution throughput	Used pipelines	Notes
ASIMD FP absolute value, D- form	VABS	2	2	F0/F1	-
ASIMD FP absolute value, Q- form	VABS	2	3/2	F0/F1	-
ASIMD FP arith, D-form	VABD, VADD, VPADD, VSUB	3	2	F0/F1	See ³⁴
ASIMD FP arith, Q-form	VABD, VADD, VSUB	3	1	F0/F1	
ASIMD FP compare, D-form	VACGE, VACGT, VACLE, VACLT, VCEQ, VCGE, VCGT, VCLE	3	2	F0/F1	-
ASIMD FP compare, Q-form	VACGE, VACGT, VACLE, VACLT, VCEQ, VCGE, VCGT, VCLE	3	1	F0/F1	-
ASIMD FP convert, integer, D- form, 16-bit elements	VCVT, VCVTA, VCVTM, VCVTN, VCVTP	4	1/2	F0	-
ASIMD FP convert, integer, D- form, non-16-bit elements	VCVT, VCVTA, VCVTM, VCVTN, VCVTP	3	1	F0	-
ASIMD FP convert, integer, Q- form, 16-bit elements	VCVT, VCVTA, VCVTM, VCVTN, VCVTP	6	1/4	F0	-
ASIMD FP convert, integer, Q- form, 32-bit elements	VCVT, VCVTA, VCVTM, VCVTN, VCVTP	4	1/2	F0	-
ASIMD FP convert, integer, Q- form, 64-bit elements	VCVT, VCVTA, VCVTM, VCVTN, VCVTP	3	1	F0	-
ASIMD FP convert, fixed, D- form, 16-bit elements	VCVT	4	1/2	F0	-
ASIMD FP convert, fixed, D- form, non-16-bit elements	VCVT	3	1	F0	-
ASIMD FP convert, fixed, Q- form, 16-bit elements	VCVT	6	1/4	F0	-

³⁴ FP add and multiply pipelines uses 2.5 cycles to calculate the results, which allows 0-cycle forward. Execution latency can reach 3 only with 0-cycle forward. Similarly, all other instructions using these pipelines would have a corresponding execution latency increase without 0-cycle forward.

ASIMD FP convert, fixed, Q- form, 32-bit elements	VCVT	4	1/2	FO	-
ASIMD FP convert, fixed, Q- form, 64-bit elements	VCVT	3	1	FO	-
ASIMD FP convert, between single-precision and half- precision	VCVT	7	1/2	FO, FO/F1	-
ASIMD FP max/min, D-form	VMAX, VMIN, VPMAX, VPMIN, VMAXNM, VMINNM	3	2	FO/F1	See ³⁴
ASIMD FP max/min, Q-form	VMAX, VMIN, VMAXNM, VMINNM	3	1	FO/F1	
ASIMD FP multiply, D-form	VMUL	3	2	FO/F1	See ³⁴ and ³⁵
ASIMD FP multiply, Q-form	VMUL	3	1	FO/F1	
ASIMD FP non-fused multiply accumulate, D-form	VMLA, VMLS	6 (3)	2	FO/F1	See ³⁴ and ³⁶
ASIMD FP fused multiply accumulate, D-form	VFMA, VFMS	5 (3)	2	FO/F1	
ASIMD FP non-fused multiply accumulate, Q-form	VMLA, VMLS	6 (3)	1	FO/F1	
ASIMD FP fused multiply accumulate, Q-form	VFMA, VFMS	5 (3)	1	FO/F1	
ASIMD FP negate, D-form	VNEG	2	2	FO/F1	-
ASIMD FP negate, Q-form	VNEG	2	3/2		-
ASIMD FP round to integral, D- form, 16-bit elements	VRINTA, VRINTM, VRINTN, VRINTP, VRINTX, VRINTZ	4	1/2	FO	-
ASIMD FP round to integral, D- form, 32-bit elements	VRINTA, VRINTM, VRINTN, VRINTP, VRINTX, VRINTZ	3	1	FO	-
ASIMD FP round to integral, Q- form, 16-bit elements	VRINTA, VRINTM, VRINTN, VRINTP, VRINTX, VRINTZ	6	1/4	FO	-

³⁵ ASIMD multiply-accumulate pipelines support late forwarding of the result from ASIMD FP multiply micro-operations to the accumulate operands of an ASIMD FP multiply-accumulate micro-operation. The latter can be issued one cycle after the ASIMD FP multiply micro-operation is issued.

³⁶ ASIMD multiply-accumulate pipelines support late forwarding of accumulate operands from similar micro-operations, allowing a typical sequence of floating-point multiply-accumulate micro-operations to issue one every N cycles (accumulate latency N is shown between brackets).

ASIMD FP round to integral, Q- form, 32-bit elements	VRINTA, VRINTM, VRINTN, VRINTP, VRINTX, VRINTZ	4	1/2	F0	-
--	--	---	-----	----	---

Table 3-27: AArch64 advanced SIMD floating-point instructions

Instruction group	AArch64 instructions	Execution latency	Execution throughput	Used pipelines	Notes
ASIMD FP absolute value, D- form	FABS	2	2	F0/F1	-
ASIMD FP absolute value, Q- form	FABS	2	3/2	F0/F1	-
ASIMD FP arith, normal, D-form	FABD, FADD, FSUB	3	2	F0/F1	See ³⁴
ASIMD FP arith, normal, Q-form	FABD, FADD, FSUB	3	1	F0/F1	
ASIMD FP arith, pairwise, D- form	FADDP	3	2	F0/F1	
ASIMD FP arith, pairwise, Q- form	FADDP	5	1	F0/F1	
ASIMD FP compare, D-form	FACGE, FACGT, FCMEQ, FCMGE, FCMGT, FCMLE, FCMLT	3	2	F0/F1	-
ASIMD FP compare, Q-form	FACGE, FACGT, FCMEQ, FCMGE, FCMGT, FCMLE, FCMLT	3	1	F0/F1	-
ASIMD FP convert, long (F16 to F32)	FCVTL (2)	7	1/2	F0, F0/F1	-
ASIMD FP convert, long (F32 to F64)	FCVTL (2)	3	1	F0	-
ASIMD FP convert, narrow (F32 to F16)	FCVTN (2) , FCVTXN (2)	7	1/2	F0, F0/F1	-
ASIMD FP convert, narrow (F64 to F32)	FCVTN (2) , FCVTXN (2)	3	1	F0	-
ASIMD FP convert, other, D- form F32 and Q-form F64	FCVTAS, VCVTAU, FCVTMS, FCVTMU, FCVTNS, FCVTNU, FCVTPS, FCVTPU, FCVTZS, FCVTZU, SCVTF, UCVTF	3	1	F0	-

ASIMD FP convert, other, Q- form F32 and D-form F16	FCVTAS, VCVTAU, FCVTMS, FCVTMU, FCVTNS, FCVTNU, FCVTPS, FCVTPU, FCVTZS, FCVTZU, SCVTF, UCVTF	4	1/2	F0	-
ASIMD FP convert, other, Q- form F16	FCVTAS, VCVTAU, FCVTMS, FCVTMU, FCVTNS, FCVTNU, FCVTPS, FCVTPU, FCVTZS, FCVTZU, SCVTF, UCVTF	6	1/4	F0	-
ASIMD FP divide, D-form, F16	FDIV	12-16	1/16-1/12	F1	See ³⁷ and ³⁸
ASIMD FP divide, Q-form, F16	FDIV	24-32	1/32-1/24	F1	
ASIMD FP divide, D-form, F32	FDIV	6-10	1/10-1/6	F1	
ASIMD FP divide, Q-form, F32	FDIV	12-20	1/20-1/12	F1	
ASIMD FP divide, Q-form, F64	FDIV	6-15	1/15-1/6	F1	
ASIMD FP max/min, normal, D- form	FMAX, FMAXNM, FMIN, FMINNM	3	2	F0/F1	-
ASIMD FP max/min, normal, Q- form	FMAX, FMAXNM, FMIN, FMINNM	3	3/2	F0/F1	-
ASIMD FP max/min, pairwise, D-form	FMAXP, FMAXNMP, FMINP, FMINNMP	3	2	F0/F1	-
ASIMD FP max/min, pairwise, Q-form	FMAXP, FMAXNMP, FMINP, FMINNMP	5	1	F0/F1	-
ASIMD FP max/min, reduce, D- form, F16	FMAXV, FMAXNMV, FMINV, FMINNMV	6	2/3	F0/F1	-
ASIMD FP max/min, reduce, Q- form, F16	FMAXV, FMAXNMV, FMINV, FMINNMV	9	1	F0/F1	-
ASIMD FP max/min, reduce, Q- form, F32	FMAXV, FMAXNMV, FMINV, FMINNMV	6	2/3	F0/F1	-
ASIMD FP multiply, D-form	FMUL, FMULX	3	2	F0/F1	See ³⁴ and ³⁵
ASIMD FP multiply, Q-form	FMUL, FMULX	3	1	F0/F1	

³⁷ ASIMD divide operations are performed using an iterative algorithm and block subsequent similar operations to the same pipeline until complete.

³⁸ FP divide and square root operations are performed using an iterative algorithm and block subsequent similar operations to the same pipeline until complete. Minimum execution latency and maximum execution throughput are achieved by power-of-two early termination. In a normal case, minimum execution latency is 6 for H-form, 8 for S-form, and 13 for D-form. It would also be possible to execute 2 H-form, 2 S-form, or only 1 D-form in parallel.

ASIMD FP multiply accumulate, D-form	FMLA, FMLS	5 (3)	2	F0/F1	See 34 and 36
ASIMD FP multiply accumulate, Q-form	FMLA, FMLS	5(3)	1	F0/F1	
ASIMD FP negate, D-form	FNEG	2	2	F0/F1	-
ASIMD FP negate, Q-form	FNEG	2	3/2	F0/F1	-
ASIMD FP round, D-form F32 and Q-form F64	FRINTA, FRINTI, FRINTM, FRINTN, FRINTP, FRINTX, FRINTZ	3	1	F0	-
ASIMD FP round, Q-form F32 and D-form F16	FRINTA, FRINTI, FRINTM, FRINTN, FRINTP, FRINTX, FRINTZ	4	1/2	F0	-
ASIMD FP round, Q-form F16	FRINTA, FRINTI, FRINTM, FRINTN, FRINTP, FRINTX, FRINTZ	6	1/4	F0	-

3.16 Advanced SIMD miscellaneous instructions

Table 3-28: AArch32 advanced SIMD miscellaneous instructions

Instruction group	AArch32 instructions	Execution latency	Execution throughput	Used pipelines	Notes
ASIMD bitwise insert, D-form	VBIF, VBIT, VBSL	3	2	F0/F1	-
ASIMD bitwise insert, Q-form	VBIF, VBIT, VBSL	3	3/2	F0/F1	-
ASIMD count, D-form	VCLS, VCLZ, VCNT	3	2	F0/F1	-
ASIMD count, Q-form	VCLS, VCLZ, VCNT	3	3/2	F0/F1	-
ASIMD duplicate, core reg	VDUP	7	1	LSO, F0/F1	-
ASIMD duplicate, scalar, D-form	VDUP	2	2	F0/F1	See ³⁹
ASIMD duplicate, scalar, Q-form	VDUP	2	3/2	F0/F1	
ASIMD extract, D-form	VEXT	2	2	F0/F1	
ASIMD extract, Q-form	VEXT	2	3/2	F0/F1	
ASIMD move, immed, D-form	VMOV	2	2	F0/F1	
ASIMD move, immed, Q-form	VMOV	2	3/2	F0/F1	
ASIMD move, register, D-form	VMOV	2	2	F0/F1	
ASIMD move, register, Q-form	VMOV	2	3/2	F0/F1	
ASIMD move, narrowing	VMOVN	2	2	F0/F1	-
ASIMD move, saturating	VQMOVN, VQMOVUN	4	1	F1	-
ASIMD reciprocal estimate, D- form, 16-bit elements	VRECPE, VRSQRTE	4	1/2	F0	-
ASIMD reciprocal estimate, D- form, 32-bit elements	VRECPE, VRSQRTE	3	1	F0	-
ASIMD reciprocal estimate, Q- form, 16-bits elements	VRECPE, VRSQRTE	6	1/4	F0	-

³⁹ FP add and multiply pipelines uses 2.5 cycles to calculate the results, which allows 0-cycle forward. These instructions are treated as multiply-accumulate which uses the FP add and multiply pipelines. Similarly, some permutation instructions can be 0-cycle forward to all permutation modules. Without 0-cycle forward, they would have one more cycle in execution latency.

ASIMD reciprocal estimate, Q-form, 32-bits elements	VRECPE, VRSQRTE	4	1/2	F0	-
ASIMD reciprocal step, D-form	VRECPS, VRSQRTS	6	2	F0/F1	See ³⁹
ASIMD reciprocal step, Q-form	VRECPS, VRSQRTS	6	1	F0/F1	
ASIMD reverse, D-form	VREV16, VREV32, VREV64	2	2	F0/F1	
ASIMD reverse, Q-form	VREV16, VREV32, VREV64	2	3/2	F0/F1	
ASIMD swap, D-form	VSWP	2	2	F0/F1	
ASIMD swap, Q-form	VSWP	2	1	F0/F1	
ASIMD table lookup, 1 reg	VTBL, VTBX	3	2	F0/F1	-
ASIMD table lookup, 2 reg	VTBL, VTBX	3	2	F0/F1	-
ASIMD table lookup, 3 reg	VTBL, VTBX	6	2	F0/F1	-
ASIMD table lookup, 4 reg	VTBL, VTBX	6	2	F0/F1	-
ASIMD transfer, scalar to core reg, word	VMOV	4	1	F0	-
ASIMD transfer, scalar to core reg, byte/hword	VMOV	4	1	F0	-
ASIMD transfer, core reg to scalar	VMOV	7	1	LSO, F0/F1	-
ASIMD transpose, D-form	VTRN	2	2	F0/F1	See ³⁹
ASIMD transpose, Q-form	VTRN	2	1	F0/F1	
ASIMD unzip/zip, D-form	VUZP, VZIP	2	2	F0/F1	
ASIMD unzip/zip, Q-form	VUZP, VZIP	6	2/3	F0/F1	

Table 3-29: AArch64 advanced SIMD miscellaneous instructions

Instruction group	AArch64 instructions	Execution latency	Execution throughput	Used pipelines	Notes
ASIMD bit reverse, D-form	RBIT	2	2	F0/F1	See ³⁹
ASIMD bit reverse, Q-form	RBIT	2	3/2	F0/F1	

ASIMD bitwise insert, D-form	BIF, BIT, BSL	3	2	F0/F1	-
ASIMD bitwise insert, Q-form	BIF, BIT, BSL	3	1	F0/F1	-
ASIMD count, D-form	CLS, CLZ, CNT	3	2	F0/F1	-
ASIMD count, Q-form	CLS, CLZ, CNT	3	1	F0/F1	-
ASIMD duplicate, gen reg	DUP	8	1	L, F0/F1	-
ASIMD duplicate, element, D- form	DUP	2	2	F0/F1	See ³⁹
ASIMD duplicate, element, Q- form	DUP	2	3/2	F0/F1	
ASIMD extract, D-form	EXT	2	2	F0/F1	
ASIMD extract, Q-form	EXT	2	3/2	F0/F1	
ASIMD extract narrow	XTN	3	2	F0/F1	-
ASIMD extract narrow, saturating	SQXTN (2) , SQXTUN (2) , UQXTN (2)	4	1	F1	-
ASIMD insert, element to element	INS	2	2	F0/F1	See ³⁹
ASIMD move, integer immed, D- form	MOVI	2	2	F0/F1	
ASIMD move, integer immed, Q- form	MOVI	2	3/2	F0/F1	
ASIMD move, FP immed, D-form	FMOV	2	2	F0/F1	
ASIMD move, FP immed, Q-form	FMOV	2	3/2	F0/F1	
ASIMD reciprocal estimate, D- form, 16-bit elements	FRECPE, FRECPX, FRSQRTE	4	1/2	F0	-
ASIMD reciprocal estimate, D- form, 32-bit elements	FRECPE, FRECPX, FRSQRTE, URECPE, URSQRTE	3	1	F0	-
ASIMD reciprocal estimate, Q- form, 16-bit elements	FRECPE, FRECPX, FRSQRTE	6	1/4	F0	-
ASIMD reciprocal estimate, Q- form, 32-bit elements	FRECPE, FRECPX, FRSQRTE, URECPE, URSQRTE	4	1/2	F0	-

ASIMD reciprocal estimate, Q- form, 64-bit elements	FRECPE, FRECPX, FRSQRTE	3	1	F0	-
ASIMD reciprocal step, D-form	FRECPS, FRSQRTS	5	2	F0/F1	See ³⁹
ASIMD reciprocal step, Q-form	FRECPS, FRSQRTS	5	1	F0/F1	
ASIMD reverse, D-form	REV16, REV32, REV64	2	2	F0/F1	
ASIMD reverse, Q-form	REV16, REV32, REV64	2	3/2	F0/F1	
ASIMD table lookup, D-form	TBL, TBX	3xN		F0/F1	See ⁴⁰
ASIMD table lookup, Q-form	TBL, TBX	3xN + 3		F0/F1	See ⁴⁰
ASIMD transfer, element to gen reg, word or dword	UMOV	4	1	F0	-
ASIMD transfer, element to gen reg, others	SMOV, UMOV	4	1	F0	-
ASIMD transfer, gen reg to element	INS	7	1	LS0, F0/F1	-
ASIMD transpose, D-form	TRN1, TRN2	2	2	F0/F1	See ³⁹
ASIMD unzip/zip, D-form	UZP1, UZP2, ZIP1, ZIP2	2	2	F0/F1	

⁴⁰ For table branches (TBL and TBX), N shows the number of registers in the table.

3.17 Advanced SIMD load instructions

Advanced SIMD has a load to use four-cycle latency. The latency numbers shown indicate the worst-case load-use latency from the load data to a dependent instruction. The latencies shown assume the memory access hits in the L1 data cache.

Table 3-30: AArch32 advanced SIMD load instructions

Instruction group	AArch32 instructions	Execution latency	Execution throughput	Used pipelines	Notes
ASIMD load, 1 element, multiple, 1 reg	VLD1	5	2	LS0/LS1	-
ASIMD load, 1 element, multiple, 2 reg	VLD1	6	1	LS0/LS1	-
ASIMD load, 1 element, multiple, 3 reg	VLD1	6	2/3	LS0/LS1	-
ASIMD load, 1 element, multiple, 4 reg	VLD1	6	1/2	LS0/LS1	-
ASIMD load, 1 element, one lane	VLD1	8	2	LS0/LS1, FO/F1	-
ASIMD load, 1 element, all lanes	VLD1	8	2	LS0/LS1, FO/F1	-
ASIMD load, 2 element, multiple, 2 reg	VLD2	8	2	LS0/LS1, FO/F1	-
ASIMD load, 2 element, multiple, 4 reg	VLD2	9	1	LS0/LS1, FO/F1	-
ASIMD load, 2 element, one lane, size 32	VLD2	8	2	LS0/LS1, FO/F1	-
ASIMD load, 2 element, one lane, size 8/16	VLD2	8	2	LS0/LS1, FO/F1	-
ASIMD load, 2 element, all lanes	VLD2	8	2	LS0/LS1, FO/F1	-
ASIMD load, 3 element, multiple, 3 reg	VLD3	9	1	LS0/LS1, FO/F1	-
ASIMD load, 3 element, one lane, size 32	VLD3	8	1	LS0/LS1, FO/F1	-
ASIMD load, 3 element, one lane, size 8/16	VLD3	9	1	LS0/LS1, FO/F1	-
ASIMD load, 3 element, all lanes	VLD3	8	2	LS0/LS1, FO/F1	-
ASIMD load, 4 element, multiple, 4 reg	VLD4	9	1	LS0/LS1, FO/F1	-

ASIMD load, 4 element, one lane, size 32	VLD4	8	1	LS0/LS1, F0/F1	-
ASIMD load, 4 element, one lane, size 8/16	VLD4	9	1	LS0/LS1, F0/F1	-
ASIMD load, 4 element, all lanes	VLD4	8	1	LS0/LS1, F0/F1	-
(ASIMD load, writeback form)	-	(1)	Same as before	+I0/I1	See ⁴¹

Table 3-31: AArch64 advanced SIMD load instructions

Instruction group	AArch64 instructions	Execution latency	Execution throughput	Used pipelines	Notes
ASIMD load, 1 element, multiple, 1 reg, D-form	LD1	5	2	LS0/LS1	-
ASIMD load, 1 element, multiple, 1 reg, Q-form	LD1	6	1	LS0/LS1	-
ASIMD load, 1 element, multiple, 2 reg, D-form	LD1	6	1	LS0/LS1	-
ASIMD load, 1 element, multiple, 2 reg, Q-form	LD1	6	1	LS0/LS1	-
ASIMD load, 1 element, multiple, 3 reg, D-form	LD1	7	2/3	LS0/LS1	-
ASIMD load, 1 element, multiple, 3 reg, Q-form	LD1	7	1/3	LS0/LS1	-
ASIMD load, 1 element, multiple, 4 reg, D-form	LD1	6	1/2	LS0/LS1	-
ASIMD load, 1 element, multiple, 4 reg, Q-form	LD1	8	1/4	LS0/LS1	-
ASIMD load, 1 element, one lane, B/H/S	LD1	8	2	LS0/LS1, F0/F1	-
ASIMD load, 1 element, one lane, D	LD1	5	2	LS0/LS1	-
ASIMD load, 1 element, all lanes, D-form, B/H/S	LD1R	8	2	LS0/LS1, F0/F1	-

⁴¹ Writeback forms of load instructions require an extra micro-operation to update the base address. This update is typically performed in parallel with the load micro-operation (update latency is shown between brackets).

ASIMD load, 1 element, all lanes, D-form, D	LD1R	8	2	LS0/LS1, FO/F1	-
ASIMD load, 1 element, all lanes, Q-form	LD1R	8	2	LS0/LS1, FO/F1	-
ASIMD load, 2 element, multiple, D-form, B/H/S	LD2	8	2	LS0/LS1, FO/F1	-
ASIMD load, 2 element, multiple, Q-form, B/H/S	LD2	8	2	LS0/LS1, FO/F1	-
ASIMD load, 2 element, multiple, Q-form, D	LD2	8	2	LS0/LS1	-
ASIMD load, 2 element, one lane, B/H	LD2	8	2	LS0/LS1, FO/F1	-
ASIMD load, 2 element, one lane, S	LD2	8	2	LS0/LS1, FO/F1	-
ASIMD load, 2 element, one lane, D	LD2	6	1	LS0/LS1	-
ASIMD load, 2 element, all lanes, D-form, B/H/S	LD2R	8	2	LS0/LS1, FO/F1	-
ASIMD load, 2 element, all lanes, D-form, D	LD2R	5	2	LS0/LS1	-
ASIMD load, 2 element, all lanes, Q-form	LD2R	9	2	LS0/LS1, FO/F1	-
ASIMD load, 3 element, multiple, D-form, B/H/S	LD3	9	1	LS0/LS1, FO/F1	-
ASIMD load, 3 element, multiple, Q-form, B/H/S	LD3	10	2/3	LS0/LS1, FO/F1	-
ASIMD load, 3 element, multiple, Q-form, D	LD3	6	2/3	LS0/LS1	-
ASIMD load, 3 element, one lane, B/H	LD3	9	2/3	LS0/LS1, FO/F1	-
ASIMD load, 3 element, one lane, S	LD3	9	1	LS0/LS1, FO/F1	-
ASIMD load, 3 element, one lane, D	LD3	6	2/3	LS0/LS1	-
ASIMD load, 3 element, all lanes, D-form, B/H/S	LD3R	9	2/3	LS0/LS1, FO/F1	-
ASIMD load, 3 element, all lanes, D-form, D	LD3R	6	2/3	LS0/LS1	-

ASIMD load, 3 element, all lanes, Q-form, B/H/S	LD3R	10	1/2	LS0/LS1, FO/F1	-
ASIMD load, 3 element, all lanes, Q-form, D	LD3R	9	2/3	LS0/LS1, FO/F1	-
ASIMD load, 4 element, multiple, D-form, B/H/S	LD4	10	1/2	LS0/LS1, FO/F1	-
ASIMD load, 4 element, multiple, Q-form, B/H/S	LD4	12	1/4	LS0/LS1, FO/F1	-
ASIMD load, 4 element, multiple, Q-form, D	LD4	9	1/2	LS0/LS1	-
ASIMD load, 4 element, one lane, B/H	LD4	9	1/2	LS0/LS1, FO/F1	-
ASIMD load, 4 element, one lane, S	LD4	9	1	LS0/LS1, FO/F1	-
ASIMD load, 4 element, one lane, D	LD4	7	1/2	LS0/LS1	-
ASIMD load, 4 element, all lanes, D-form, B/H/S	LD4R	9	1/2	LS0/LS1, FO/F1	-
ASIMD load, 4 element, all lanes, D-form, D	LD4R	7	1/2	LS0/LS1	-
ASIMD load, 4 element, all lanes, Q-form, B/H/S	LD4R	9	1/2	LS0/LS1, FO/F1	-
ASIMD load, 4 element, all lanes, Q-form, D	LD4R	8	1/2	LS0/LS1, FO/F1	-
(ASIMD load, writeback form)	-	(1)	Same as before	+I0/I1	See ⁴¹

3.18 Advanced SIMD store instructions

Stores micro-operations can issue after their address operands are available and do not need to wait for data operands. After they are executed, stores are buffered and committed in the background.

Table 3-32: AArch32 advanced SIMD store instructions

Instruction group	AArch32 instructions	Execution latency	Execution throughput	Used pipelines	Notes
ASIMD store, 1 element, multiple, 1 reg	VST1	1	2	LS0/1,FD	-
ASIMD store, 1 element, multiple, 2 reg	VST1	2	1	LS0/1,FD	-
ASIMD store, 1 element, multiple, 3 reg	VST1	3	2/3	LS0/1,FD	-
ASIMD store, 1 element, multiple, 4 reg	VST1	4	1/2	LS0/1,FD	-
ASIMD store, 1 element, one lane	VST1	3	2	F0/F1, LS0/1,FD	-
ASIMD store, 2 element, multiple, 2 reg	VST2	3	1	F0/F1, LS0/1,FD	-
ASIMD store, 2 element, multiple, 4 reg	VST2	4	1/2	F0/F1, LS0/1,FD	-
ASIMD store, 2 element, one lane	VST2	3	1	F0/F1, LS0/1,FD	-
ASIMD store, 3 element, multiple, 3 reg	VST3	3	2/3	F0/F1, LS0/1,FD	-
ASIMD store, 3 element, one lane, size 32	VST3	3	1	F0/F1, LS0/1,FD	-
ASIMD store, 3 element, one lane, size 8/16	VST3	3	2	F0/F1, LS0/1,FD	-
ASIMD store, 4 element, multiple, 4 reg	VST4	4	1/2	F0/F1, LS0/1,FD	-
ASIMD store, 4 element, one lane, size 32	VST4	3	1	F0/F1, LS0/1,FD	-
ASIMD store, 4 element, one lane, size 8/16	VST4	3	2	F0/F1, LS0/1,FD	-
(ASIMD store, writeback form)	-	-	+1	+I0/I1	See ⁴²

⁴² Writeback forms of store instructions require an extra micro-operation to update the base address. This update is typically performed in parallel with the store micro-operation (update latency is shown between brackets).

Table 3-33: AArch64 advanced SIMD store instructions

Instruction group	AArch64 instructions	Execution latency	Execution throughput	Used pipelines	Notes
ASIMD store, 1 element, multiple, 1 reg, D-form	ST1	1	2	LS0/LS1/D	-
ASIMD store, 1 element, multiple, 1 reg, Q-form	ST1	2	1	LS0/LS1/D	-
ASIMD store, 1 element, multiple, 2 reg, D-form	ST1	2	1	LS0/LS1/D	-
ASIMD store, 1 element, multiple, 2 reg, Q-form	ST1	4	1/2	LS0/LS1/D	-
ASIMD store, 1 element, multiple, 3 reg, D-form	ST1	3	2/3	LS0/LS1/D	-
ASIMD store, 1 element, multiple, 3 reg, Q-form	ST1	6	1/3	LS0/LS1/D	-
ASIMD store, 1 element, multiple, 4 reg, D-form	ST1	4	1/2	LS0/LS1/D	-
ASIMD store, 1 element, multiple, 4 reg, Q-form	ST1	8	1/4	LS0/LS1/D	-
ASIMD store, 1 element, one lane, B/H/S	ST1	3	2	F0/F1, LS0/1,FD	-
ASIMD store, 1 element, one lane, D	ST1	1	2	LS0/LS1/D	-
ASIMD store, 2 element, multiple, D-form, B/H/S	ST2	3	1	F0/F1, LS0/1,FD	-
ASIMD store, 2 element, multiple, Q-form, B/H/S	ST2	4	1/2	F0/F1, LS0/1,FD	-
ASIMD store, 2 element, multiple, Q-form, D	ST2	1	1	LS0/LS1/D	-
ASIMD store, 2 element, one lane, B/H/S	ST2	3	2	F0/F1, LS0/1,FD	-
ASIMD store, 2 element, one lane, D	ST2	2	1	LS0/LS1/D	-
ASIMD store, 3 element, multiple, D-form, B/H/S	ST3	3	2/3	F0/F1, LS0/1,FD	-
ASIMD store, 3 element, multiple, Q-form, B/H/S	ST3	6	1/3	F0/F1, LS0/1,FD	-
ASIMD store, 3 element, multiple, Q-form, D	ST3	6	1/3	LS0/LS1/D	-
ASIMD store, 3 element, one lane, B/H	ST3	3	2	F0/F1, LS0/1,FD	-

ASIMD store, 3 element, one lane, S	ST3	3	1	F0/F1, LS0/1,FD-	
ASIMD store, 3 element, one lane, D	ST3	3	2/3	LS0/LS1/D	-
ASIMD store, 4 element, multiple, D-form, B/H/S	ST4	4	1/2	F0/F1, LS0/1,FD-	
ASIMD store, 4 element, multiple, Q-form, B/H/S	ST4	8	1/4	F0/F1, LS0/1,FD-	
ASIMD store, 4 element, multiple, Q-form, D	ST4	8	1/4	LS0/LS1/D	-
ASIMD store, 4 element, one lane, B/H	ST4	3	2	F0/F1, LS0/1,FD-	
ASIMD store, 4 element, one lane, S	ST4	3	1	F0/F1, LS0/1,FD-	
ASIMD store, 4 element, one lane, D	ST4	4	1/2	LS0/LS1/D	-
(ASIMD store, writeback form)	-	(1)	Same as before	+I0/I1	See ⁴²

3.19 Cryptographic Extension

Table 3-34: AArch32 Cryptographic Extension instructions

Instruction group	AArch32 instructions	Execution latency	Execution throughput	Used pipelines	Notes
Crypto AES ops	AESD, AESE, AESIMC, AESMC	2	1	F0	See ⁴³ , ⁴⁴ , and ⁴⁵
Crypto polynomial (64x64) multiply long	VMULL.P64	2	1	F0	See ⁴⁴ and ⁴⁵
Crypto SHA1 xor ops	SHA1SU0	6	3/2	F0/F1	-
Crypto SHA1 fast ops	SHA1H, SHA1SU1	2	1	F0	See ⁴⁴ and ⁴⁵
Crypto SHA1 slow ops	SHA1C, SHA1M, SHA1P	4	1/2	F0	
Crypto SHA256 fast ops	SHA256SU0	2	1	F0	
Crypto SHA256 slow ops	SHA256H, SHA256H2, SHA256SU1	4	1/2	F0	

Table 3-35: AArch64 Cryptographic Extension instructions

Instruction group	AArch64 instructions	Execution latency	Execution throughput	Used pipelines	Notes
Crypto AES ops	AESD, AESE, AESIMC, AESMC	2	1	F0	See ⁴³ , ⁴⁴ , and ⁴⁵
Crypto polynomial (64x64) multiply long	PMULL (2)	2	1	F0	See ⁴⁴ and ⁴⁵
Crypto SHA1 xor ops	SHA1SU0	6	3/2	F0/F1	-
Crypto SHA1 schedule acceleration ops	SHA1H, SHA1SU1	2	1	F0	See ⁴⁴ and ⁴⁵
Crypto SHA1 hash acceleration ops	SHA1C, SHA1M, SHA1P	4	1/2	F0	

⁴³ Adjacent AESE/AESMC instruction pairs and adjacent AESD/AESIMC instruction pairs exhibit the described performance characteristics. See **AES encryption and decryption** for more information.

⁴⁴ Cryptographic execution supports late forwarding of the result from a producer micro-operation to a consumer micro-operation. This results in a reduction of one cycle in latency, as seen by the consumer.

⁴⁵ Some cryptographic instructions use two or four cycles to calculate their results, which allows 0-cycle forward. Without 0-cycle forward, they would have one more cycle in execution latency.

Crypto SHA256 schedule acceleration op (1 μ op)	SHA256SU0	2	1	FO	
Crypto SHA256 schedule acceleration op (2 μ ops)	SHA256SU1	4	1/2	FO	
Crypto SHA256 hash acceleration ops	SHA256H, SHA256H2	4	1/2	FO	

3.20 CRC

Table 3-36: AArch32 CRC instructions

Instruction group	AArch32 instructions	Execution latency	Execution throughput	Used pipelines	Notes
CRC checksum ops	CRC32, CRC32C	2	1	10	See ⁴⁶

Table 3-37: AArch64 CRC instructions

Instruction group	AArch64 instructions	Execution latency	Execution throughput	Used pipelines	Notes
CRC checksum ops	CRC32, CRC32C	2	1	10	See ⁴⁶

⁴⁶ CRC execution supports late forwarding of the result from a producer CRC micro-operation to a consumer CRC micro-operation. This results in a reduction of one cycle in latency, as seen by the consumer.

4 Special considerations

4.1 Dispatch constraints

Dispatch of micro-operations from the in-order portion to the out-of-order portion of the micro-architecture includes some constraints. It is important to consider these constraints during code generation to maximize the effective dispatch bandwidth and subsequent execution bandwidth of the Cortex-A75 core.

The dispatch stage can process up to eight micro-operations per cycle, with the following limitations on the number of micro-operations of each type that might be simultaneously dispatched:

- Up to two micro-operations using the I0 pipelines.
- Up to two micro-operations using the I1 pipelines.
- Up to three micro-operations using the LS pipelines.
- Up to three micro-operations using the F0/F1 pipelines.
- Up to three micro-operations using the Branch pipeline.

If there are more micro-operations available to be dispatched in a given cycle than the constraints above can support, then the micro-operations are dispatched in oldest-to-youngest age order, to the extent allowed by the constraints above.

4.2 Conditional ASIMD

Conditional execution is architecturally possible for some ASIMD instructions in Thumb state using IT blocks. However, this type of encoding is considered abnormal and is not recommended for the Cortex-A75 core. It is likely to perform worse than the equivalent unconditional encodings.

4.3 Optimizing memory copy

The Cortex-A75 core features two load/store pipelines able to execute two micro-operations per cycle of either type. To achieve maximum throughput for memory copy (or similar loops), you must:

- Unroll the loop to include multiple load and store operations per iteration, minimizing the overheads of looping.
- Use discrete, non-writeback forms of load and store instructions (such as `LDP` and `STP`), interleaving them so that two load operations and one store operation might be performed each cycle.

The following example shows a recommended instruction sequence for a long memory copy in AArch64 state:

```

Loop_start:
    SUBS X2,X2,#192
    LDP   Q3,Q4,[x1,#0]
    LDP   Q5,Q6,[x1,#32]
    LDP   Q7,Q8,[x1,#64]
    STP   Q3,Q4,[x0,#0]
    STP   Q5,Q6,[x0,#32]
    STP   Q7,Q8,[x0,#64]
    LDP   Q3,Q4,[x1,#96]
    LDP   Q5,Q6,[x1,#128]
    LDP   Q7,Q8,[x1,#160]
    STP   Q3,Q4,[x0,#96]
    STP   Q5,Q6,[x0,#128]
    STP   Q7,Q8,[x0,#160]
    ADD   X1,X1,#192
    ADD   X0,X0,#192
    BGT   Loop_start

```

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

4.4 Load/store alignment

The Armv8.2-A architecture allows many types of load and store accesses to be arbitrarily aligned. On the Cortex-A75 core, the following cases reduce bandwidth or incur additional latency:

- Load operations that cross a 64-bit boundary.
- In AArch64, all stores that cross a 128-bit boundary.
- In AArch32, all stores that cross a 64-bit boundary.

4.5 AES encryption and decryption

The Cortex-A75 core can issue one AESE, AESMC, AESD, or AESIMC instruction every cycle (fully pipelined) with an execution latency of two cycles. This means encryption or decryption for at least two data chunks should be interleaved for maximum performance:

```

AESE data0, key0
AESMC data0, data0
AESE data1, key0
AESMC data1, data1
AESE data0, key0
AESMC data0, data0
AESE data1, key1
AESMC data1, data1
...

```

Pairs of dependent AESE/AESMC or AESD/AESIMC instructions provide higher performance when they are adjacent and in the described order in the program code.

4.6 Branch instruction alignment

Branch instruction and branch target instruction alignment and density can affect performance. For best-case performance, consider the following guidelines:

- Avoid placing more than three branch instructions within an aligned 16-byte instruction memory region.
- When possible, a branch and its target should be located within the same 4MB-aligned memory region.
- Consider aligning subroutine entry points and branch targets to 16-byte boundaries, within the bounds of the code-density requirements of the program. This ensures that the subsequent fetch can maximize bandwidth following the taken branch by bringing in all useful instructions.
- For loops which comprise 16 or fewer instruction bytes, it is preferred that the loop is located entirely within a single aligned 16-byte instruction memory region.

4.7 Region-based fast forwarding

Forwarding logic in the F0/F1 pipelines is such that it allows optimal latency for the most frequent instruction pairs. These optimized forwarding regions are defined as follows:

- From all `FADD`, `FMUL`, and `FMLA` 32-bit instructions to all `FADD`, `FMUL`, and `FMLA` 32-bit instructions.
- From all `FADD`, `FMUL`, and `FMLA` 64-bit instructions to all `FADD`, `FMUL`, and `FMLA` 64-bit instructions.
- From `PERM` instructions to all `FADD`, `FMUL`, or `FMLA` 32-bit or 64-bit instructions.
- From all `CRYPT` to all `CRYPT` instructions (AES, polynomial (64x64) multiply long, and SHA).
- From all `PERM` and `CRYPT` instructions to all `PERM`, `iALU`, and `iSHF` instructions.

4.8 FPCR self-synchronization

Programmers and compiler writers should note that writes to the `FPCR` register are self-synchronizing. This implies that writes to the `FPCR` register do not need a context synchronizing operation to have a visible effect on subsequent instructions.

4.9 Special register access

The Cortex-A75 core performs register renaming for general purpose registers to enable speculative and out-of-order instruction execution. However, most special-purpose registers are not renamed. Instructions that read or write non-renamed registers are subject to one or more of the following additional execution constraints:

- Non-speculative execution

- Instructions might 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 following table shows various special-purpose register read accesses and the associated execution constraints or side-effects.

Table 4-1: Special-purpose register read accesses

Register read	Non-speculative	In-order	Flush side-effect	Notes
APSR	Yes	Yes	No	See ⁴⁷
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	See ⁴⁸
FPSR	Yes	Yes	No	
NZCV	No	No	No	See ⁴⁹
SP_*	No	No	No	
SPSel	No	Yes	No	-
SPSR_*	No	Yes	No	-

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

⁴⁷ APSR reads must wait for all prior instructions that might set the Q bit to execute and retire.

⁴⁸ FPSR and FPSCR reads must wait for all prior instructions that might update the status flags to execute and retire.

⁴⁹ The NZCV and SP registers are fully renamed.

Table 4-2: Special-purpose register write accesses

Register write	Non-speculative	In-order	Flush side-effect	Notes
APSR	Yes	Yes	No	See ⁵⁰
DAIF	Yes	Yes	No	-
DLR_EL0	Yes	Yes	No	-
DSPSR_EL0	Yes	Yes	No	-
ELR_*	Yes	Yes	No	-
FPCR	Yes	Yes	Maybe	See ⁵¹
FPSCR	Yes	Yes	Maybe	See ⁵¹ and ⁵²
FPSR	Yes	Yes	No	See ⁵²
NZCV	No	No	No	See ⁴⁹
SP_*	No	No	No	
SPSel	Yes	Yes	Yes	-
SPSR_*	Yes	Yes	No	-

4.10 IT blocks

The Armv8-A architecture performance deprecates some uses of the IT instruction in such a way that software might be written using multiple naïve single instruction IT blocks. Instead, it is preferred that software generates multi-instruction IT blocks.

⁵⁰ APSR writes that set the Q bit introduce a barrier which prevents subsequent instructions from executing until the write completes.

⁵¹ If the FPCR or FPSCR write is predicted to change the control field values, then it introduces a barrier which prevents subsequent instructions from executing. If the FPCR or FPSCR write is not predicted to change the control field values, then it executes without a barrier but triggers a flush if the values change.

⁵² If another FPSR or FPSCR write is still pending, then FPSR and FPSCR writes must stall at dispatch.