Arm® Cortex®-A725 Core

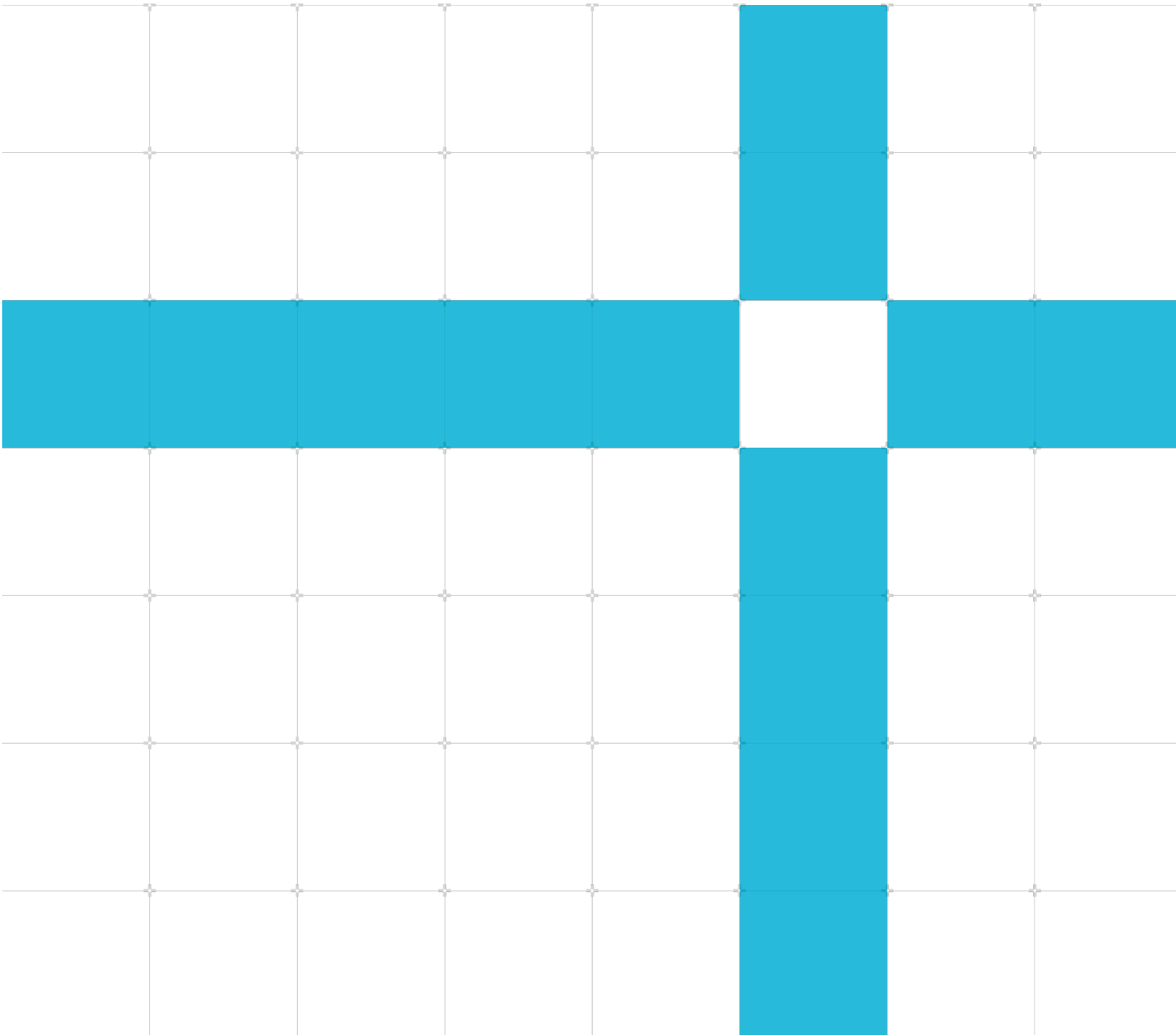
Revision: r0p2

Software Optimization Guide

Non-Confidential

Copyright © 2023-2024 Arm Limited (or its affiliates).
All rights reserved.

Issue 4.0
109824



Arm® Cortex®-A725 Core
Software Optimization Guide

Copyright © 2023-2024 Arm Limited (or its affiliates). All rights reserved.

Release information

Document history

Issue	Date	Confidentiality	Change
1.0	28 February 2023	Confidential	First limited access release for r0p0
2.0	31 July 2023	Confidential	First early access release for r0p1
3.0	29 July 2024	Non-Confidential	Second early access release for r0p1
4.0	29 November 2024	Non-Confidential	First release for r0p2

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**.

Copyright © 2023-2024 Arm Limited (or its affiliates). All rights reserved.

Contents

1 Introduction 7
1.1 Product revision status 7
1.2 Intended audience 7
1.3 Scope 7
1.4 Conventions 7
1.4.1 Glossary 7
1.4.2 Terms and abbreviations 7
1.4.3 Typographical conventions 9
1.5 Useful resources 10
1.6 Feedback 11
1.6.1 Feedback on this product 11
1.6.2 Feedback on content 11

2 Overview 12
2.1 Pipeline overview 14

3 Instruction characteristics 16
3.1 Instruction tables 16
3.2 Legend for reading the utilized pipelines 16
3.3 Branch instructions 17
3.4 Arithmetic and logical instructions 18
3.5 Divide and multiply instructions 19
3.6 Pointer Authentication Instructions 20
3.7 Miscellaneous data-processing instructions 21
3.8 Load instructions 21
3.9 Store instructions 23
3.10 Tag Load Instructions 23
3.11 Tag Store instructions 24
3.12 FP data processing instructions 24
3.13 FP miscellaneous instructions 25
3.14 FP load instructions 26
3.15 FP store instructions 27
3.16 ASIMD integer instructions 28

Copyright © 2023-2024 Arm Limited (or its affiliates). All rights reserved.

3.17 ASIMD floating-point instructions ... 31
3.18 ASIMD BFloat16 (BF16) instructions ... 34
3.19 ASIMD miscellaneous instructions ... 35
3.20 ASIMD load instructions ... 37
3.21 ASIMD store instructions ... 39
3.22 Cryptography extensions ... 40
3.23 CRC ... 41
3.24 SVE Predicate instructions ... 42
3.25 SVE integer instructions ... 43
3.26 SVE floating-point instructions ... 50
3.27 SVE BFloat16 (BF16) instructions ... 52
3.28 SVE Load instructions ... 53
3.29 SVE Store instructions ... 55
3.30 SVE Miscellaneous instructions ... 57
3.31 SVE Cryptographic instructions ... 58

4 Special considerations ... 59
4.1 Dispatch constraints ... 59
4.2 Optimizing memory routines ... 59
4.3 Load/Store alignment ... 61
4.4 Store to Load Forwarding ... 61
4.5 AES encryption/decryption ... 62
4.6 Region based fast forwarding ... 63
4.7 Branch instruction alignment ... 64
4.8 FPCR self-synchronization ... 65
4.9 Special register access ... 65
4.10 Instruction fusion ... 66
4.11 Zero Latency Instructions ... 67
4.12 TLB-access latencies ... 68
4.13 Cache-access latencies ... 68
4.14 Cache maintenance operation ... 69
4.15 Memory Tagging - Tagging Performance ... 69
4.16 Memory Tagging - Synchronous Mode ... 70
4.17 Complex ASIMD and SVE instructions ... 70
4.18 MOVPRFX fusion ... 71

Copyright © 2023-2024 Arm Limited (or its affiliates). All rights reserved.

Appendix A Revisions.....75

Copyright © 2023-2024 Arm Limited (or its affiliates). All rights reserved.

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®-A725 core micro-architecture that influence software performance. Micro-architectural detail is limited to that which is useful for software optimization.

Documentation extends only to software visible behavior of the Cortex®-A725 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.

Copyright © 2023-2024 Arm Limited (or its affiliates). All rights reserved.

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

Copyright © 2023-2024 Arm Limited (or its affiliates). All rights reserved.

1.4.3 Typographical conventions

Convention	Use
italic	Introduces citations.
bold	Highlights interface elements, such as menu names. Denotes signal names. Also used for terms in descriptive lists, where appropriate.
`monospace`	Denotes text that you can enter at the keyboard, such as commands, file and program names, and source code.
`monospace` **`bold`**	Denotes language keywords when used outside example code.
`monospace underline`	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.
	This represents a recommendation which, if not followed, might lead to system failure or damage.
	This represents a requirement for the system that, if not followed, might result in system failure or damage.
	This represents a requirement for the system that, if not followed, will result in system failure or damage.
	This represents an important piece of information that needs your attention.
	This represents a useful tip that might make it easier, better or faster to perform a task.
	This is a reminder of something important that relates to the information you are reading.

Copyright © 2023-2024 Arm Limited (or its affiliates). All rights reserved.

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®-A725 Core Technical Reference Manual	107652	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**.

Copyright © 2023-2024 Arm Limited (or its affiliates). All rights reserved.

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®-A725 Core Software Optimization Guide.
- The number 109824.
- 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.

Copyright © 2023-2024 Arm Limited (or its affiliates). All rights reserved.

2 Overview

The Cortex®-A725 core is a balanced-performance, low-power, and constrained area product that implements the Arm®v9.2-A architecture. The Arm®v9.2-A architecture extends the architecture defined in the Arm®v8-A architectures up to Arm®v8.7-A. It targets large screen compute applications as well as smartphone applications.

The key features of Cortex®-A725 core are:

- Implementation of the Arm®v9.2-A A64 instruction set
- AArch64 Execution state at all Exception levels, EL0 to EL3
- Memory Management Unit (MMU)
- 40-bit Physical Address (PA) and 48-bit Virtual Address (VA)
- Generic Interrupt Controller (GIC) CPU interface to connect to an external interrupt Distributor
- Generic Timers interface that supports 64-bit count input from an external system counter
- Implementation of the Reliability, Availability, and Serviceability (RAS) Extension
- Implementation of the Scalable Vector Extension (SVE) with a 128-bit vector length and Scalable Vector Extension 2 (SVE2)
- Integrated execution unit with Advanced Single Instruction Multiple Data (SIMD) and floating-point support
- Support for the optional Cryptographic Extension, which is licensed separately
- Activity Monitoring Unit (AMU)
- Separate L1 data and instruction caches
- Private, unified data and instruction L2 cache
- Optional error protection with parity or Error Correcting Code (ECC) allowing:

  - Single Error Correction and Double Error Detection (SECDED) on L1 data cache and L2 cache

  - Single Error Detection (SED) on L1 instruction cache and L2 Translation Lookaside Buffer (TLB)
- Support for Memory System Resource Partitioning and Monitoring (MPAM)

Debug features

- Arm®v9.2-A debug logic
- Performance Monitoring Unit (PMU)
- Embedded Trace Extension (ETE)
- Trace Buffer Extension (TRBE)
- Optional implementation of the Statistical Profiling Extension (SPE)
- Optional Embedded Logic Analyzer (ELA), ELA-600

Copyright © 2023-2024 Arm Limited (or its affiliates). All rights reserved.

This document describes elements of the Cortex®-A725 core micro-architecture that influence software performance so that software and compilers can be optimized accordingly.

Copyright © 2023-2024 Arm Limited (or its affiliates). All rights reserved.

2.1 Pipeline overview

The following figure describes the high-level Cortex®-A725 core instruction processing pipeline.

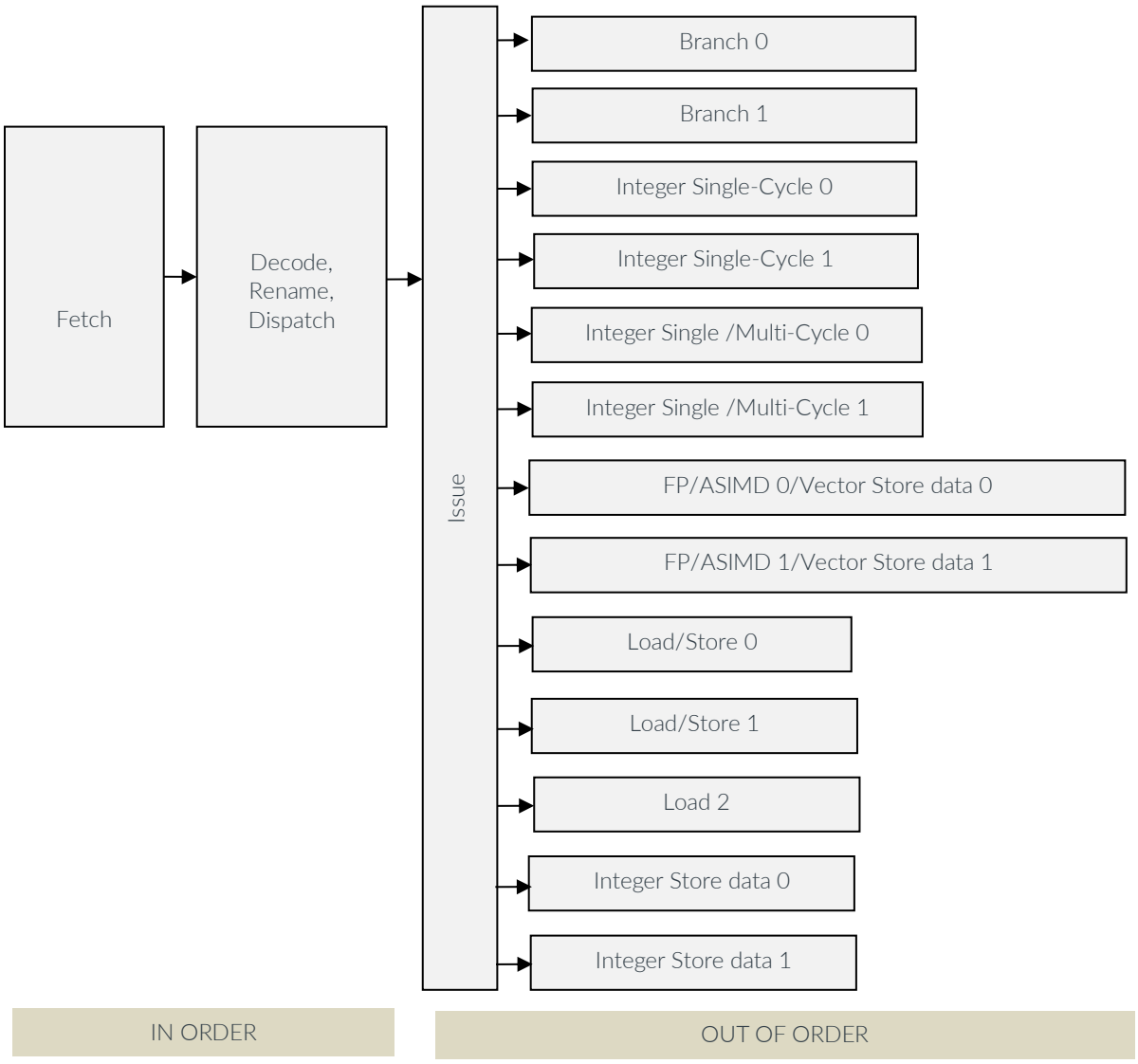
Instructions are first fetched and then decoded into internal Macro-OPerations (MOPs).

From there, the MOPs proceed through register renaming and dispatch stages.

A MOP can be split into two Micro-OPerations (µOPs) further down the pipeline after the decode stage. Once dispatched, µOPs wait for their operands and issue out-of-order to one of thirteen issue pipelines.

Each issue pipeline can accept one µOP per cycle.

Figure 2-1 Cortex®-A725 core pipeline



Copyright © 2023-2024 Arm Limited (or its affiliates). All rights reserved.

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

Table 2-1 Cortex®-A725 core operations

Instruction groups	Instructions
Branch 0/1	Branch µOPs
Integer Single-Cycle 0/1	Integer ALU µOPs
Integer Single/Multi-cycle 0/1	Integer shift-ALU, multiply, divide and CRC µOPs
Load/Store 0/1	Load, Store address generation and special memory µOPs
Load 2	Load µOPs
Integer Store data 0/1	Integer Store data µOPs
FP/ASIMD-0/Vector Store data 0	ASIMD ALU, ASIMD misc, ASIMD integer multiply, FP convert, FP misc, FP add, FP multiply, FP divide, FP sqrt, AES µOps, crypto µOPs, store data µOPs
FP/ASIMD-1/Vector Store data 1	ASIMD ALU, ASIMD misc, FP misc, FP add, FP multiply, ASIMD shift µOPs, ASIMD reduction µOPs, AES µOPs., store data µOPs

Copyright © 2023-2024 Arm Limited (or its affiliates). All rights reserved.

3 Instruction characteristics

3.1 Instruction tables

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

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

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®-A725 core microarchitecture.

3.2 Legend for reading the utilized pipelines

Table 3-1 Cortex®-A725 core pipeline names and symbols

Pipeline name	Symbol used in tables
Branch 0/1	B
Integer single Cycle 0/1	S
Integer single Cycle 0/1 and single/multicycle 0/1	I
Integer single/multicycle 0/1	M
Integer multicycle 0	M0
Load/Store 01	L01
Load/Store 0/1 and Load 2	L
Integer Store data 0/1	ID
FP/ASIMD/Vector Store data 0/1	V
FP/ASIMD/Vector Store data 0	V0
FP/ASIMD/Vector Store data 1	V1

Copyright © 2023-2024 Arm Limited (or its affiliates). All rights reserved.

3.3 Branch instructions

Table 3-2 AArch64 Branch instructions

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

Copyright © 2023-2024 Arm Limited (or its affiliates). All rights reserved.

3.4 Arithmetic and logical instructions

Table 3-3 AArch64 Arithmetic and logical instructions

Instruction Group	AArch64 Instructions	Exec Latency	Execution Throughput	Utilized Pipelines	Notes
ALU, basic	ADD, ADC, AND, BIC, EON, EOR, ORN, ORR, SUB, SBC	1	4	I	-
ALU, basic, flagset	ADDS, ADCS, ANDS, BICS, SUBS, SBCS	1	4	I	-
ALU, extend and shift	ADD{S}, SUB{S}	2	2	M	-
Arithmetic, LSL shift, shift <= 4	ADD, SUB	1	4	I	-
Arithmetic, flagset, LSL shift, shift <= 4	ADDS, SUBS	1	4	I	-
Arithmetic, LSR/ASR/ROR shift or LSL shift > 4	ADD{S}, SUB{S}	2	2	M	-
Arithmetic, immediate to logical address tag	ADDG, SUBG	1	4	I	-
Conditional compare	CCMN, CCMP	1	4	I	-
Conditional select	CSEL, CSINC, CSINV, CSNEG	1	4	I	-
Convert floating-point condition flags	AXFLAG, XAFLAG	1	4	I	-
Flag manipulation instructions	SETF8, SETF16, RMIF, CFINV	1	4	I	-
Insert Random Tags	IRG	2	1	M0	1
Insert Tag Mask	GMI	1	4	I	-
Logical, shift, no flagset	AND, BIC, EON, EOR, ORN, ORR	1	4	I	-
Logical, shift, flagset	ANDS, BICS	2	2	M	-
Subtract Pointer	SUBP	1	4	I	-
Subtract Pointer, flagset	SUBPS	1	3	I	-

Notes:

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

Copyright © 2023-2024 Arm Limited (or its affiliates). All rights reserved.

3.5 Divide and multiply instructions

Table 3-4 AArch64 Divide and multiply instructions

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

Notes:

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

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

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

Copyright © 2023-2024 Arm Limited (or its affiliates). All rights reserved.

3.6 Pointer Authentication Instructions

Table 3-5 AArch64 pointer authentication instructions

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

Notes:

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

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

Copyright © 2023-2024 Arm Limited (or its affiliates). All rights reserved.

3.7 Miscellaneous data-processing instructions

Table 3-6 AArch64 Miscellaneous data-processing instructions

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

3.8 Load instructions

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

Table 3-7 AArch64 Load instructions

Instruction Group	AArch64 Instructions	Exec Latency	Execution Throughput	Utilized Pipelines	Notes
Load register, literal	LDR, LDRSW, PRFM	5	2	L, S	-
Load register, unscaled immed	LDUR, LDURB, LDURH, LDURSB, LDURSH, LDURSW, PRFUM	4	3	L	-
Load register, immed post-index	LDR, LDRB, LDRH, LDRSB, LDRSH, LDRSW	4	3	L, I	-
Load register, immed pre-index	LDR, LDRB, LDRH, LDRSB, LDRSH, LDRSW	4	3	L, I	1
Load register, immed unprivileged	LDTR, LDTRB, LDTRH, LDTRSB, LDTRSH, LDTRSW	4	3	L	-

Copyright © 2023-2024 Arm Limited (or its affiliates). All rights reserved.

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

Notes:

1. Only Immed pre-index with write back use I pipes
2. Execution Latency is 5 and Utilized Pipelines are L, I when scale with aligned offset of 128 bits

Copyright © 2023-2024 Arm Limited (or its affiliates). All rights reserved.

3.9 Store instructions

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

Table 3-8 AArch64 Store instructions

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

3.10 Tag Load Instructions

Table 3-9 AArch64 Tag load instructions

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

Copyright © 2023-2024 Arm Limited (or its affiliates). All rights reserved.

3.11 Tag Store instructions

Table 3-10 AArch64 Tag store instructions

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

3.12 FP data processing instructions

Table 3-11 AArch64 FP data processing instructions

Instruction Group	AArch64 Instructions	Exec Latency	Execution Throughput	Utilized Pipelines	Notes
FP absolute value	FABS, FABD	2	2	V	-
FP arithmetic	FADD, FSUB	2	2	V	-
FP compare	FCCMP{E}, FCMP{E}	2	2	V	-
FP divide, H-form	FDIV	5	1	V0	1

Copyright © 2023-2024 Arm Limited (or its affiliates). All rights reserved.

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

Notes:

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

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

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

3.13 FP miscellaneous instructions

Table 3-12 AArch64 FP miscellaneous instructions

Instruction Group	AArch64 Instructions	Exec Latency	Execution Throughput	Utilized Pipelines	Notes
FP convert, from gen to vec reg	SCVTF, UCVTF	3	1	M0	-
FP convert, from vec to gen reg	FCVTAS, FCVTAU, FCVTMS, FCVTMU, FCVTNS, FCVTNU, FCVTPS, FCVTPU, FCVTZS, FCVTZU	3	1	V0, S0	-

Copyright © 2023-2024 Arm Limited (or its affiliates). All rights reserved.

Instruction Group	AArch64 Instructions	Exec Latency	Execution Throughput	Utilized Pipelines	Notes
FP convert, Javascript from vec to gen reg	FJCVTZS	3	1	V0, S0	-
FP convert, from vec to vec reg	FCVT, FCVTXN	3	1	V0	-
FP move, immed	FMOV	2	2	V	1
FP move, register	FMOV	2	2	V	1
FP transfer, from gen to low half of vec reg	FMOV	3	1	M0	-
FP transfer, from gen to high half of vec reg	FMOV	5	1	M0, V	-
FP transfer, from vec to gen reg	FMOV	3	2	V, S	-

Notes:

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

3.14 FP load instructions

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

Table 3-13 AArch64 FP load instructions

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

Copyright © 2023-2024 Arm Limited (or its affiliates). All rights reserved.

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

3.15 FP store instructions

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

Table 3-14 AArch64 FP store instructions

Instruction Group	AArch64 Instructions	Exec Latency	Execution Throughput	Utilized Pipelines	Notes
Store vector reg, unscaled immed, B/H/S/D-form	STUR	2	2	L01, V	-
Store vector reg, unscaled immed, Q-form	STUR	2	2	L01, V	-
Store vector reg, immed post-index, B/H/S/D-form	STR	2	2	L01, V, I	-
Store vector reg, immed post-index, Q-form	STR	2	2	L01, V, I	-
Store vector reg, immed pre-index, B/H/S/D-form	STR	3	2	L01, V, I	-
Store vector reg, immed pre-index, Q-form	STR	2	2	L01, V, I	-
Store vector reg, unsigned immed, B/H/S/D-form	STR	2	2	L01, V	-
Store vector reg, unsigned immed, Q-form	STR	2	2	L01, V	-
Store vector reg, register offset, basic, B/H/S/D-form	STR	2	2	L01, V	-
Store vector reg, register offset, basic, Q-form	STR	2	2	L01, V	-

Copyright © 2023-2024 Arm Limited (or its affiliates). All rights reserved.

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

3.16 ASIMD integer instructions

Table 3-15 AArch64 ASIMD integer instructions

Instruction Group	AArch64 Instructions	Exec Latency	Execution Throughput	Utilized Pipelines	Notes
ASIMD absolute diff	SABD, UABD	2	2	V	-

Copyright © 2023-2024 Arm Limited (or its affiliates). All rights reserved.

Instruction Group	AArch64 Instructions	Exec Latency	Execution Throughput	Utilized Pipelines	Notes
ASIMD absolute diff accum	SABA, UABA	4(1)	1	V1	2
ASIMD absolute diff accum long	SABAL(2), UABAL(2)	4(1)	1	V1	2
ASIMD absolute diff long	SABDL(2), UABDL(2)	2	2	V	-
ASIMD arith, basic	ABS, ADD, NEG, SADDL(2), SADDW(2), SHADD, SHSUB, SSUBL(2), SSUBW(2), SUB, UADDL(2), UADDW(2), UHADD, UHSUB, USUBL(2), USUBW(2)	2	2	V	-
ASIMD arith, complex	ADDHN(2), RADDHN(2), RSUBHN(2), SQABS, SQADD, SQNEG, SQSUB, SRHADD, SUBHN(2), SUQADD, UQADD, UQSUB, URHADD, USQADD	2	2	V	-
ASIMD arith, pair-wise	ADDP, SADDLP, UADDLP	2	2	V	-
ASIMD arith, reduce, 4H/4S	ADDV, SADDLV, UADDLV	3	1	V1	-
ASIMD arith, reduce, 8B/8H	ADDV, SADDLV, UADDLV	5	1	V1, V	-
ASIMD arith, reduce, 16B	ADDV, SADDLV, UADDLV	6	1/2	V1	-
ASIMD compare	CMEQ, CMGE, CMGT, CMHI, CMHS, CMLE, CMLT, CMTST	2	2	V	-
ASIMD dot product	SDOT, UDOT	3 (1)	2	V	2
ASIMD dot product using signed and unsigned integers	SUDOT, USDOT	3(1)	2	V	2
ASIMD logical	AND, BIC, EOR, MOV, MVN, NOT, ORN, ORR	2	2	V	-
ASIMD matrix multiply-accumulate	SMMLA, UMMLA, USMMLA	3(1)	2	V	2

Copyright © 2023-2024 Arm Limited (or its affiliates). All rights reserved.

Instruction Group	AArch64 Instructions	Exec Latency	Execution Throughput	Utilized Pipelines	Notes
ASIMD max/min, basic and pair-wise	SMAX, SMAXP, SMIN, SMINP, UMAX, UMAXP, UMIN, UMINP	2	2	V	-
ASIMD max/min, reduce, 4H/4S	SMAXV, SMINV, UMAXV, UMINV	3	1	V1	-
ASIMD max/min, reduce, 8B/8H	SMAXV, SMINV, UMAXV, UMINV	5	1	V1, V	-
ASIMD max/min, reduce, 16B	SMAXV, SMINV, UMAXV, UMINV	6	1/2	V1	-
ASIMD multiply	MUL, SQDMULH, SQRDMULH	4	1	V0	-
ASIMD multiply accumulate	MLA, MLS	4(1)	1	V0	1
ASIMD multiply accumulate high	SQRDMLAH, SQRDMLSH	4(2)	1	V0	1
ASIMD multiply accumulate long	SMLAL(2), SMLSL(2), UMLAL(2), UMLSL(2)	4(1)	1	V0	1
ASIMD multiply accumulate saturating long	SQDMLAL(2), SQDMLSL(2)	4(2)	1	V0	1
ASIMD multiply/multiply long (8x8) polynomial, D-form	PMUL, PMULL(2)	2	1	V0	3
ASIMD multiply/multiply long (8x8) polynomial, Q-form	PMUL, PMULL(2)	2	1	V0	3
ASIMD multiply long	SMULL(2), UMULL(2), SQDMULL(2)	4	1	V0	-
ASIMD pairwise add and accumulate long	SADALP, UADALP	4(1)	1	V1	2
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	-

Copyright © 2023-2024 Arm Limited (or its affiliates). All rights reserved.

Instruction Group	AArch64 Instructions	Exec Latency	Execution Throughput	Utilized Pipelines	Notes
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	-

Notes:

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

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

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

3.17 ASIMD floating-point instructions

Table 3-16 AArch64 ASIMD floating-point instructions

Instruction Group	AArch64 Instructions	Exec Latency	Execution Throughput	Utilized Pipelines	Notes
ASIMD FP absolute value/difference	FABS, FABD	2	2	V	-
ASIMD FP arith, normal	FADD, FSUB	2	2	V	-
ASIMD FP compare	FACGE, FACGT, FCMEQ, FCMGE, FCMGT, FCMLE, FCMLT	2	2	V	-
ASIMD FP complex add	FCADD	3	2	V	-
ASIMD FP complex multiply add	FCMLA	4(2)	2	V	1
ASIMD FP convert, long (F16 to F32)	FCVTL(2)	4	1/2	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	-

Copyright © 2023-2024 Arm Limited (or its affiliates). All rights reserved.

Instruction Group	AArch64 Instructions	Exec Latency	Execution Throughput	Utilized Pipelines	Notes
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	-
ASIMD FP convert, other, D-form F16 and Q-form F32	FCVTAS, VCVTAU, FCVTMS, FCVTMU, FCVTNS, FCVTNU, FCVTPS, FCVTPU, FCVTZS, FCVTZU, SCVTF, UCVTF	4	1/2	V0	-
ASIMD FP convert, other, Q-form F16	FCVTAS, VCVTAU, FCVTMS, FCVTMU, FCVTNS, FCVTNU, FCVTPS, FCVTPU, FCVTZS, FCVTZU, SCVTF, UCVTF	6	1/4	V0	-
ASIMD FP divide, D-form, F16	FDIV	8	1/4	V0	3
ASIMD FP divide, D-form, F32	FDIV	8	1/2	V0	3
ASIMD FP divide, Q-form, F16	FDIV	12	1/8	V0	3
ASIMD FP divide, Q-form, F32	FDIV	10	1/4	V0	3
ASIMD FP divide, Q-form, F64	FDIV	13	1/2	V0	3
ASIMD FP max/min, normal	FMAX, FMAXNM, FMIN, FMINNM	2	2	V	-
ASIMD FP arith, max/min, pairwise	FADDP, FMAXP, FMAXNMP, FMINP, FMINNMP	3	2	V	-
ASIMD FP max/min, reduce, F32 and D-form F16	FMAXV, FMAXNMV, FMINV, FMINNMV	4	1	V	-

Copyright © 2023-2024 Arm Limited (or its affiliates). All rights reserved.

Instruction Group	AArch64 Instructions	Exec Latency	Execution Throughput	Utilized Pipelines	Notes
ASIMD FP max/min, reduce, Q-form F16	FMAXV, FMAXNMV, FMINV, FMINNMV	6	2/3	V	-
ASIMD FP multiply	FMUL, FMULX	3	2	V	2
ASIMD FP multiply accumulate	FMLA, FMLS	4(2)	2	V	1
ASIMD FP multiply accumulate long	FMLAL(2), FMLSL(2)	4(2)	2	V	1
ASIMD FP negate	FNEG	2	2	V	-
ASIMD FP round, D-form F32 and Q-form F64	FRINTA, FRINTI, FRINTM, FRINTN, FRINTP, FRINTX, FRINTZ, FRINT32X, FRINT64X, FRINT32Z, FRINT64Z	3	1	V0	-
ASIMD FP round, D-form F16 and Q-form F32	FRINTA, FRINTI, FRINTM, FRINTN, FRINTP, FRINTX, FRINTZ, FRINT32X, FRINT64X, FRINT32Z, FRINT64Z	4	1/2	V0	-
ASIMD FP round, Q-form F16	FRINTA, FRINTI, FRINTM, FRINTN, FRINTP, FRINTX, FRINTZ, FRINT32X, FRINT64X, FRINT32Z, FRINT64Z	6	1/4	V0	-
ASIMD FP square root, D-form, F16	FSQRT	8	1/4	V0	3
ASIMD FP square root, D-form, F32	FSQRT	8	1/2	V0	3
ASIMD FP square root, Q-form, F16	FSQRT	12	1/8	V0	3
ASIMD FP square root, Q-form, F32	FSQRT	10	1/4	V0	3
ASIMD FP square root, Q-form, F64	FSQRT	13	1/2	V0	3

Notes:

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

Copyright © 2023-2024 Arm Limited (or its affiliates). All rights reserved.

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

3.18 ASIMD BFloat16 (BF16) instructions

Table 3-17 AArch64 ASIMD BFloat (BF16) instructions

Instruction Group	AArch64 Instructions	Exec Latency	Execution Throughput	Utilized Pipelines	Notes
ASIMD convert, F32 to BF16	BFCVTN, BFCVTN2	4	1/2	V0	-
ASIMD dot product	BFDOT	4(2)	2	V	1
ASIMD matrix multiply accumulate	BFMMLA	5(3)	2	V	1
ASIMD multiply accumulate long	BFMLALB, BFMLALT	4(2)	2	V	1
Scalar convert, F32 to BF16	BFCVT	3	1	V0	-

Notes:

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

Copyright © 2023-2024 Arm Limited (or its affiliates). All rights reserved.

3.19 ASIMD miscellaneous instructions

Table 3-18 AArch64 ASIMD miscellaneous instructions

Instruction Group	AArch64 Instructions	Exec Latency	Execution Throughput	Utilized Pipelines	Notes
ASIMD bit reverse	RBIT	2	2	V	2
ASIMD bitwise insert	BIF, BIT, BSL	2	2	V	
ASIMD count	CLS, CLZ, CNT	2	2	V	-
ASIMD duplicate, gen reg	DUP	3	1	M0	-
ASIMD duplicate, element	DUP	2	2	V	2
ASIMD extract	EXT	2	2	V	2
ASIMD extract narrow	XTN(2)	2	2	V	
ASIMD extract narrow, saturating	SQXTN(2), SQXTUN(2), UQXTN(2)	4	1	V1	-
ASIMD insert, element to element	INS	2	2	V	2
ASIMD move, FP immed	FMOV	2	2	V	1
ASIMD move, integer immed	MOVI, MVNI	2	2	V	-
ASIMD reciprocal and square root estimate, D-form U32	URECPE, URSQRTE	3	1	V0	-
ASIMD reciprocal and square root estimate, Q-form U32	URECPE, URSQRTE	4	1/2	V0	-
ASIMD reciprocal and square root estimate, D-form F32 and scalar forms	FRECPE, FRSQRTE	3	1	V0	-
ASIMD reciprocal and square root estimate, D-form F16 and Q-form F32	FRECPE, FRSQRTE	4	1/2	V0	-
ASIMD reciprocal and square root estimate, Q-form F16	FRECPE, FRSQRTE	6	1/4	V0	-
ASIMD reciprocal exponent	FRECPX	3	1	V0	
ASIMD reciprocal step	FRECPS, FRSQRTS	4	2	V	-
ASIMD reverse	REV16, REV32, REV64	2	2	V	2
ASIMD table lookup, 1 or 2 table regs	TBL	2	2	V	2
ASIMD table lookup, 3 table regs	TBL	4	1	V	2
ASIMD table lookup, 4 table regs	TBL	4	2/3	V	2
ASIMD table lookup extension, 1 table reg	TBX	2	2	V	2

Copyright © 2023-2024 Arm Limited (or its affiliates). All rights reserved.

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

Notes:

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

2 PERM instructions part of a particular region forwarding

Copyright © 2023-2024 Arm Limited (or its affiliates). All rights reserved.

3.20 ASIMD load instructions

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

Table 3-19 AArch64 ASIMD load instructions

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

Copyright © 2023-2024 Arm Limited (or its affiliates). All rights reserved.

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

Copyright © 2023-2024 Arm Limited (or its affiliates). All rights reserved.

Instruction Group	AArch64 Instructions	Exec Latency	Execution Throughput	Utilized Pipelines	Notes
ASIMD load, 4 element, all lanes, Q-form, B/H/S	LD4R	8	2/3	L, V	-
ASIMD load, 4 element, all lanes, Q-form, D	LD4R	8	1/2	L, V	-
(ASIMD load, writeback form)	-	-	-	I	1

Notes:

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

3.21 ASIMD store instructions

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

Table 3-20 AArch64 ASIMD store instructions

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

Copyright © 2023-2024 Arm Limited (or its affiliates). All rights reserved.

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

Notes:

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

3.22 Cryptography extensions

Table 3-21 AArch64 Cryptography extensions

Instruction Group	AArch64 Instructions	Exec Latency	Execution Throughput	Utilized Pipelines	Notes
Crypto AES ops	AESD, AESE, AESIMC, AESMC	2	2	V	-

Copyright © 2023-2024 Arm Limited (or its affiliates). All rights reserved.

Instruction Group	AArch64 Instructions	Exec Latency	Execution Throughput	Utilized Pipelines	Notes
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	-
Crypto SHA512 hash acceleration ops	SHA512H, SHA512H2, SHA512SU0, SHA512SU1	2	1	V0	-
Crypto SHA3 ops	BCAX, EOR3, RAX1, XAR	2	2	V	2
Crypto SM3 ops	SM3PARTW1, SM3PARTW2SM3SS1, SM3TT1A, SM3TT1B, SM3TT2A, SM3TT2B	2	1	V0	-
Crypto SM4 ops	SM4E, SM4EKEY	4	1	V0	-

Notes:

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

2. SHA3 ops are executed from the ALU pipeline

3.23 CRC

Table 3-22 AArch64 CRC

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

Notes:

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

Copyright © 2023-2024 Arm Limited (or its affiliates). All rights reserved.

3.24 SVE Predicate instructions

Table 3-23 SVE Predicate Instructions

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

Copyright © 2023-2024 Arm Limited (or its affiliates). All rights reserved.

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

Notes:

1. Operation leading to all, none element active are optimized in rename stage pipeline, execution latency and throughput are then not representative.

3.25 SVE integer instructions

Table 3-24 SVE integer instructions

Instruction Group	SVE Instruction	Exec Latency	Execution Throughput	Utilized Pipelines	Notes
Arithmetic, absolute diff	SABD, UABD	2	2	V	-
Arithmetic, absolute diff accum	SABA, UABA	4(1)	1	V1	1

Copyright © 2023-2024 Arm Limited (or its affiliates). All rights reserved.

Instruction Group	SVE Instruction	Exec Latency	Execution Throughput	Utilized Pipelines	Notes
Arithmetic, absolute diff accum long	SABALB, SABALT, UABALB, UABALT	4(1)	1	V1	1
Arithmetic, absolute diff long	SABDLB, SABDLT, UABDLB, UABDLT	2	2	V	-
Arithmetic, basic	ABS, ADD, ADR, CNOT, NEG, SADDLB, SADDLBT, SADDLT, SADDWB, SADDWT, SHADD, SHSUB, SHSUBR, SSUBLB, SSUBLBT, SSUBLT, SSUBLTB, SSUBWB, SSUBWT, SUB, SUBHNB, SUBHNT, SUBR, UADDLB, UADDLT, UADDWB, UADDWT, UHADD, UHSUB, UHSUBR, USUBLB, USUBLT, USUBWB, USUBWT	2	2	V	-
Arithmetic, complex	ADDHNB, ADDHNT, RADDHNB, RADDHNT, RSUBHNB, RSUBHNT, SQABS, SQADD, SQNEG, SQSUB, SQSUBR, SRHADD, SUQADD, UQADD, UQSUB, UQSUBR, USQADD, URHADD	2	2	V	-
Arithmetic, large integer	ADCLB, ADCLT, SBCLB, SBCLT	2	2	V	-
Arithmetic, pairwise add	ADDP	2	2	V	-

Copyright © 2023-2024 Arm Limited (or its affiliates). All rights reserved.

Instruction Group	SVE Instruction	Exec Latency	Execution Throughput	Utilized Pipelines	Notes
Arithmetic, pairwise add and accum long	SADALP, UADALP	4(1)	1	V1	1
Arithmetic, shift	ASR, ASRR, LSL, LSLR, LSR, LSRR	2	1	V1	-
Arithmetic, shift and accumulate	SRSRA, SSRA, URSRA, USRA	4(1)	1	V1	1
Arithmetic, shift by immediate	SHRNB, SHRNT, SSHLLB, SSHLLT, USHLLB, USHLLT	2	1	V1	-
Arithmetic, shift by immediate and insert	SLI, SRI	2	1	V1	-
Arithmetic, shift complex	RSHRNB, RSHRNT, SQRSHL, SQRSHLR, SQRSHRNB, SQRSHRNT, SQRSHRUNB, SQRSHRUNT, SQSHL, SQSHLR, SQSHLU, SQSHRNB, SQSHRNT, SQSHRUNB, SQSHRUNT, UQRSHL, UQRSHLR, UQRSHRNB, UQRSHRNT, UQSHL, UQSHLR, UQSHRNB, UQSHRNT	4	1	V1	-
Arithmetic, shift right for divide	ASRD	4	1	V1	-
Arithmetic, shift rounding	SRSHL, SRSHLR, SRSHR, URSHL, URSHLR, URSHR	4	1	V1	-
Bit manipulation	BDEP, BEXT, BGRP	4	1/2	V0	-
Bitwise select	BSL, BSL1N, BSL2N, NBSL	2	2	V	-
Count/reverse bits	CLS, CLZ, CNT, RBIT	2	2	V	-
Broadcast logical bitmask immediate to vector	DUPM, MOV	2	2	V	-
Compare and set flags	CMPEQ, CMPGE, CMPGT, CMPHI, CMPHS, CMPLE, CMPLO, CMPLS, CMPLT, CMPNE	2	2	V	

Copyright © 2023-2024 Arm Limited (or its affiliates). All rights reserved.

Instruction Group	SVE Instruction	Exec Latency	Execution Throughput	Utilized Pipelines	Notes
Complex add	CADD, SQCADD	2	2	V	-
Complex dot product 8-bit element	CDOT	3(1)	2	V	1
Complex dot product 16-bit element	CDOT	4(1)	1	V0	1
Complex multiply-add B, H, S element size	CMLA	4(1)	1	V0	1
Complex multiply-add D element size	CMLA	5(3)	1/2	V0	1
Conditional extract operations, scalar form	CLASTA, CLASTB	8	1	M0, V	-
Conditional extract operations, SIMD&FP scalar and vector forms	CLASTA, CLASTB, COMPACT, SPLICE	2	2	V	-
Convert to floating point, 64b to float or convert to double	SCVTF, UCVTF	3	1	V0	-
Convert to floating point, 32b to single or half	SCVTF, UCVTF	4	1/2	V0	-
Convert to floating point, 16b to half	SCVTF, UCVTF	6	1/4	V0	-
Copy, scalar	CPY	5	1	M0, V	
Copy, scalar SIMD&FP or imm	CPY	2	2	V	
Divides, 32 bit	SDIV, SDIVR, UDIV, UDIVR	8	1/8	V0	2
Divides, 64 bit	SDIV, SDIVR, UDIV, UDIVR	16	1/16	V0	2
Dot product, 8 bit	SDOT, UDOT	3(1)	2	V	1
Dot product, 8 bit, using signed and unsigned integers	SUDOT, USDOT	3(1)	2	V	1
Dot product, 16 bit	SDOT, UDOT	4(1)	1	V0	1
Duplicate, immediate and indexed form	DUP, MOV	2	2	V	-
Duplicate, scalar form	DUP, MOV	3	1	M0	-
Extend, sign or zero	SXTB, SXTH, SXTW, UXTB, UXTH, UXTW	2	2	V	-
Extract	EXT	2	2	V	-
Extract narrow saturating	SQXTNB, SQXTNT, SQXTUNB, SQXTUNT, UQXTNB, UQXTNT	4	1	V1	-

Copyright © 2023-2024 Arm Limited (or its affiliates). All rights reserved.

Instruction Group	SVE Instruction	Exec Latency	Execution Throughput	Utilized Pipelines	Notes
Extract/insert operation, SIMD and FP scalar form	LASTA, LASTB, INSR	2	2	V	-
Extract/insert operation, scalar	LASTA, LASTB, INSR	5	2	V	-
Histogram operations	HISTCNT, HISTSEG	2	2	V	-
Horizontal operations, B, H, S form, immediate operands only	INDEX	2	2	V	-
Horizontal operations, B, H, S form, scalar, immediate operands)/ scalar operands only / immediate, scalar operands	INDEX	5	1	M0, V	-
Horizontal operations, D form, immediate operands only	INDEX	2	2	V	-
Horizontal operations, D form, scalar, immediate operands)/ scalar operands only / immediate, scalar operands	INDEX	5	1	M0, V	-
Logical	AND, BIC, EON, EOR, EORBT, EORTB, MOV, NOT, ORN, ORR	2	2	V	-
Max/min, basic and pairwise	SMAX, SMAXP, SMIN, SMINP, UMAX, UMAXP UMIN, UMINP	2	2	V	-
Matching operations	MATCH, NMATCH	2	2	V	
Matrix multiply-accumulate	SMMLA, UMMLA, USMMLA	3(1)	2	V	1
Move prefix	MOVPRFX	2	2	V	-
Multiply, B, H, S element size	MUL, SMULH, UMULH	4	1	V0	-
Multiply, D element size	MUL, SMULH, UMULH	5	1/2	V0	-
Multiply long	SMULLB, SMULLT, UMULLB, UMULLT	4	1	V0	-
Multiply accumulate, B, H, S element size	MLA, MLS	4(1)	1	V0	1
Multiply accumulate, D element size	MLA, MLS, MAD, MSB,	5(3)	1/2	V0	1

Copyright © 2023-2024 Arm Limited (or its affiliates). All rights reserved.

Instruction Group	SVE Instruction	Exec Latency	Execution Throughput	Utilized Pipelines	Notes
Multiply accumulate long	SMLALB, SMLALT, SMLSLB, SMLSLT, UMLALB, UMLALT, UMLSLB, UMLSLT	4(1)	1	V0	1
Multiply accumulate saturating doubling long regular	SQDMLALB, SQDMLALT, SQDMLALBT, SQDMLSLB, SQDMLSLT, SQDMLSLBT	4(2)	1	V0	3
Multiply saturating doubling high, B, H, S element size	SQDMULH	4	1	V0	-
Multiply saturating doubling high, D element size	SQDMULH	5	1/2	V0	-
Multiply saturating doubling long	SQDMULLB, SQDMULLT	4	1	V0	-
Multiply saturating rounding doubling regular/complex accumulate, B, H, S element size	SQRDMLAH, SQRDMLSH, SQRDCMLAH	4(2)	1	V0	3
Multiply saturating rounding doubling regular/complex accumulate, D element size	SQRDMLAH, SQRDMLSH, SQRDCMLAH	5(3)	1/2	V0	3
Multiply saturating rounding doubling regular/complex, B, H, S element size	SQRDMULH	4	1	V0	-
Multiply saturating rounding doubling regular/complex, D element size	SQRDMULH	5	1/2	V0	-
Multiply/multiply long, (8x8) polynomial	PMUL, PMULLB, PMULLT	2	1	V0	-

Copyright © 2023-2024 Arm Limited (or its affiliates). All rights reserved.

Instruction Group	SVE Instruction	Exec Latency	Execution Throughput	Utilized Pipelines	Notes
Predicate counting vector	CNT, DECB, DECH, DECW, DECD, INCB, INCH, INCW, INCD, SQDECB, SQDECH, SQDECW, SQDECD, SQINCB, SQINCH, SQINCW, SQINCD, UQDECB, UQDECH, UQDECW, UQDECD, UQINCB, UQINCH, UQINCW, UQINCD	2	2	V	-
Reciprocal estimate for B	URECPE, URSQRTE	4	1	V0	
Reciprocal estimate for H	URECPE, URSQRTE	6	1/2	V0	
Reduction, arithmetic, B form	SADDV, UADDV, SMAXV, SMINV, UMAXV, UMINV	8	1/2	V, V1	4
Reduction, arithmetic, H form	SADDV, UADDV, SMAXV, SMINV, UMAXV, UMINV	7	1	V, V1	4
Reduction, arithmetic, S form	SADDV, UADDV, SMAXV, SMINV, UMAXV, UMINV	4	2	V	
Reduction, logical	ANDV, EORV, ORV	5	1	V, V1	-
Reverse, vector	REV, REVB, REVH, REVW	2	2	V	-
Select, vector form	MOV, SEL	2	2	V	-
Table lookup	TBL	2	2	V	-
Table lookup extension	TBX	2	2	V	-
Transpose, vector form	TRN1, TRN2	2	2	V	-
Unpack and extend	SUNPKHI, SUNPKLO, UUNPKHI, UUNPKLO	2	2	V	-
Zip/unzip	UZP1, UZP2, ZIP1, ZIP2	2	2	V	-

Notes:

Copyright © 2023-2024 Arm Limited (or its affiliates). All rights reserved.

1. SVE accumulate pipelines support late-forwarding of accumulate operands from similar µOPs, allowing a typical sequence of such µOPs to issue one every N cycles (accumulate latency N shown in parentheses).
2. SVE integer divide operations are now performed using a fully pipelined data path.
3. Same as 2 except that for saturating instructions require an extra cycle of latency for late-forwarding accumulate operands.
4. Signed Additions need 2 cycles more

3.26 SVE floating-point instructions

Table 3-25 SVE floating-point instructions

Instruction Group	SVE Instruction	Exec Latency	Execution Throughput	Utilized Pipelines	Notes
Floating point absolute value/difference	FABD, FABS	2	2	V	-
Floating point arithmetic	FADD, FNEG, FSUB, FSUBR	2	2	V	-
Floating point associative add, F16	FADDA	16	1/4	V	-
Floating point associative add, F32	FADDA	8	1/2	V	-
Floating point associative add, F64	FADDA	4	1	V	-
Floating point compare	FACGE, FACGT, FACLE, FACLT, FCMEQ, FCMGE, FCMGT, FCMLE, FCMLT, FCMNE, FCMUO	2	2	V	-
Floating point complex add	FCADD	3	2	V	-
Floating point complex multiply add	FCMLA	4(2)	2	V	1
Floating point convert, long or narrow (F16 to F32 or F32 to F16)	FCVT, FCVTLT, FCVTNT	4	1/2	V0	-
Floating point convert, long or narrow (F16 to F64, F32 to F64, F64 to F32 or F64 to F16)	FCVT, FCVTLT, FCVTNT	3	1	V0	-
Floating point convert, round to odd	FCVTX, FCVTXNT	3	1	V0	-
Floating point base2 log, F16	FLOGB	6	1/4	V0	
Floating point base2 log, F32	FLOGB	4	1/2	V0	
Floating point base2 log, F64	FLOGB	3	1	V0	
Floating point convert to integer, F16	FCVTZS, FCVTZU	6	1/4	V0	-
Floating point convert to integer, F32	FCVTZS, FCVTZU	4	1/2	V0	-

Copyright © 2023-2024 Arm Limited (or its affiliates). All rights reserved.

Instruction Group	SVE Instruction	Exec Latency	Execution Throughput	Utilized Pipelines	Notes
Floating point convert to integer, F64	FCVTZS, FCVTZU	3	1	V0	-
Floating point copy	FCPY, FDUP, FMOV	2	2	V	-
Floating point divide, F16	FDIV, FDIVR	12	1/8	V0	2
Floating point divide, F32	FDIV, FDIVR	10	1/4	V0	2
Floating point divide, F64	FDIV, FDIVR	13	1/2	V0	2
Floating point arith, min/max pairwise	FADDP, FMAXP, FMAXNMP, FMINP, FMINNMP	3	2	V	
Floating point min/max	FMAX, DMIN, FMAXNM, FMINNM	2	2	V	-
Floating point multiply	FSCALE, FMUL, FMULX	3	2	V	-
Floating point multiply accumulate	FMLA, FMLS, FMAD, FMSB, FNMAD, FNMLA, FNMLS, FNMSB	4(2)	2	V	1
Floating point multiply add/sub accumulate long	FMLALB, FMLALT, FMLSLB, FMLSLT	4(2)	2	V	1
Floating point reciprocal estimate, F16	FRECPE, FRECPX, FRSQRTE	6	1/4	V0	-
Floating point reciprocal estimate, F32	FRECPE, FRECPX, FRSQRTE	4	1/2	V0	-
Floating point reciprocal estimate, F64	FRECPE, FRECPX, FRSQRTE	3	1	V0	-
Floating point reciprocal step	FRECPS, FRSQRTS	4	2	V	-
Floating point reduction, F16	FADDV, FMAXNMV, FMAXV, FMINNMV, FMINV	6	2/3	V	-
Floating point reduction, F32	FADDV, FMAXNMV, FMAXV, FMINNMV, FMINV	4	1	V	-

Copyright © 2023-2024 Arm Limited (or its affiliates). All rights reserved.

Instruction Group	SVE Instruction	Exec Latency	Execution Throughput	Utilized Pipelines	Notes
Floating point reduction, F64	FADDV, FMAXNMV, FMAXV, FMINNMV, FMINV	2	2	V	-
Floating point round to integral, F16	FRINTA, FRINTM, FRINTN, FRINTP, FRINTX, FRINTZ	6	1/4	V0	-
Floating point round to integral, F32	FRINTA, FRINTM, FRINTN, FRINTP, FRINTX, FRINTZ	4	1/2	V0	-
Floating point round to integral, F64	FRINTA, FRINTM, FRINTN, FRINTP, FRINTX, FRINTZ	3	1	V0	-
Floating point square root, F16	FSQRT	12	1/8	V0	2
Floating point square root, F32	FSQRT	10	1/4	V0	2
Floating point square root F64	FSQRT	13	1/2	V0	2
Floating point trigonometric exponentiation	FEXPA	2	2	V	
Floating point trigonometric multiply add	FTMAD	4	2	V	
Floating point trigonometric, miscellaneous	FTSMUL, FTSSEL	3	2	V	-

Notes:

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

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

3.27 SVE BFloat16 (BF16) instructions

Table 3-26 SVE Bfloat16 (BF16) instructions

Instruction Group	SVE Instruction	Exec Latency	Execution Throughput	Utilized Pipelines	Notes
Convert, F32 to BF16	BFCVT, BFCVTNT	4	1/2	V0	-
Dot product	BFDOT	4(2)	2	V	1
Matrix multiply accumulate	BFMMLA	5(3)	2	V	1
Multiply accumulate long	BFMLALB, BFMLALT	4(2)	2	V	1

Notes:

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

Copyright © 2023-2024 Arm Limited (or its affiliates). All rights reserved.

3.28 SVE Load instructions

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

Table 3-27 SVE Load instructions

Instruction Group	SVE Instruction	Exec Latency	Execution Throughput	Utilized Pipelines	Notes
Load vector	LDR	6	3	L	-
Load predicate	LDR	7	2	L, M	-
Contiguous load, scalar + imm	LD1B, LD1D, LD1H, LD1W, LD1SB, LD1SH, LD1SW,	6	3	L	-
Contiguous load, scalar + scalar	LD1B, LD1D, LD1H, LD1W, LD1SB, LD1SH LD1SW	6	3	L	-
Contiguous load broadcast, scalar + imm	LD1RB, LD1RH, LD1RD, LD1RW, LD1RSB, LD1RSH, LD1RSW, LD1RQB, LD1RQD, LD1RQH, LD1RQW	6	3	L	-
Contiguous load broadcast, scalar + scalar	LD1RQB, LD1RQD, LD1RQH, LD1RQW	6	3	L	-
Non temporal load, scalar + imm	LDNT1B, LDNT1D, LDNT1H, LDNT1W	6	3	L	-
Non temporal load, scalar + scalar	LDNT1B, LDNT1D, LDNT1H, LDNT1W	6	3	L	-
Non temporal gather load, vector + scalar 32-bit element size	LDNT1B, LDNT1H, LDNT1W, LDNT1SB, LDNT1SH	7	3/4	L	-
Non temporal gather load, vector + scalar 64-bit element size	LDNT1B, LDNT1D, LDNT1H, LDNT1W, LDNT1SB, LDNT1SH, LDNT1SW	6	4/5	L	-

Copyright © 2023-2024 Arm Limited (or its affiliates). All rights reserved.

Instruction Group	SVE Instruction	Exec Latency	Execution Throughput	Utilized Pipelines	Notes
Contiguous first faulting load, scalar + scalar	LDFF1B, LDFF1D, LDFF1H, LDFF1W, LDFF1SB, LDFF1SD, LDFF1SH LDFF1SW	6	3	L	-
Contiguous non faulting load, scalar + imm	LDNF1B, LDNF1D, LDNF1H, LDNF1W, LDNF1SB, LDNF1SH, LDNF1SW	6	3	L	-
Contiguous Load two structures to two vectors, scalar + imm	LD2B, LD2D, LD2H, LD2W	8	2	V, L	-
Contiguous Load two structures to two vectors, scalar + scalar	LD2B, LD2D, LD2H, LD2W	8	2	V, L	
Contiguous Load three structures to three vectors, scalar + imm	LD3D	8	2/3	V, L	-
Contiguous Load three structures to three vectors, scalar + imm	LD3B, LD3H, LD3W	10	1/3	V, L	
Contiguous Load three structures to three vectors, scalar + scalar	LD3D	9	2/3	V, L, I	-
Contiguous Load three structures to three vectors, scalar + scalar	LD3B, LD3W, LD3H	11	1/3	V, L, I	-
Contiguous Load four structures to four vectors, scalar + imm	LD4D	8	1/2	V, L	-
Contiguous Load four structures to four vectors, scalar + imm	LD4B, LD4H, LD4W	12	2/5	V, L	-
Contiguous Load four structures to four vectors, scalar + scalar	LD4D	9	1/2	L, V, I	-
Contiguous Load four structures to four vectors, scalar + scalar	LD4B, LD4H, LD4W	13	2/5	L, V, I	-

Copyright © 2023-2024 Arm Limited (or its affiliates). All rights reserved.

Instruction Group	SVE Instruction	Exec Latency	Execution Throughput	Utilized Pipelines	Notes
Gather load, vector + imm, 32-bit element size	LD1B, LD1H, LD1W, LD1SB, LD1SH, LD1SW, LDFF1B, LDFF1H, LDFF1W, LDFF1SB, LDFF1SH, LDFF1SW	7	3/4	L	-
Gather load, vector + imm, 64-bit element size	LD1B, LD1D, LD1H, LD1W, LD1SB, LD1SH, LD1SW, LDFF1B, LDFF1D LDFF1H, LDFF1W, LDFF1SB, LDFF1SD, LDFF1SH, LDFF1SW	6	4/5	L	-
Gather load, 32-bit scaled, unscaled offset	LD1H, LD1SH, LDFF1H, LDFF1SH, LD1W, LDFF1W, LDFF1SW	7	3/4	L	-
Gather load, 32-bit unpacked unscaled offset, 64 bit scaled, unscaled offset	LD1B, LD1SB, LDFF1B, LDFF1SB, LD1D, LDFF1D, LD1H, LD1SH, LDFF1H, LDFF1SH, LD1W, LD1SW, LDFF1W, LDFF1SW	6	4/5	L	-
Gather load, 32-bit unscaled offset	LD1B, LD1SB, LDFF1B, LDFF1SB	7	3/4	L	
Gather load, 32-bit unpacked unscaled offset, 64 bit unscaled offset	LD1B, LD1SB, LDFF1B, LDFF1SB	6	4/5	L	-

3.29 SVE Store instructions

Table 3-28 SVE Store instructions

Instruction Group	SVE Instruction	Exec Latency	Execution Throughput	Utilized Pipelines	Notes
Store from predicate reg	STR	1	2	L01	-

Copyright © 2023-2024 Arm Limited (or its affiliates). All rights reserved.

Instruction Group	SVE Instruction	Exec Latency	Execution Throughput	Utilized Pipelines	Notes
Store from vector reg	STR	2	2	L01, V	-
Contiguous store, scalar + imm	ST1B, ST1H, ST1D, ST1W	2	2	L01, V	-
Contiguous store, scalar + scalar	ST1H	2	2	L01, I, V	-
Contiguous store, scalar + scalar	ST1B, ST1D, ST1W	2	2	L01, V	-
Contiguous store two structures from two vectors, scalar + imm	ST2B, ST2H, ST2D, ST2W	2	2	L01, V	-
Contiguous store two structures from two vectors, scalar + scalar	ST2B, ST2D, ST2H, ST2W	2	2	L01, V	-
Contiguous store three structures from three vectors, scalar + imm	ST3B, ST3D, ST3H, ST3W	4	2/3	L01, V	-
Contiguous store three structures from three vectors, scalar + imm	ST3D	3	2/3	L01, V	
Contiguous store three structures from three vectors, scalar + scalar	ST3B, ST3H, ST3W	4	2/3	L01, I, V	-
Contiguous store three structures from three vectors, scalar + scalar	ST3D	3	2/3	L01, I, V	
Contiguous store four structures from four vectors, scalar + imm	ST4B, ST4H, ST4W	6	2/3	L01, V	-
Contiguous store four structures from four vectors, scalar + imm	ST4D	3	1/2	L01, V	
Contiguous store four structures from four vectors, scalar + scalar	ST4D	3	1/2	L01, I, V	
Contiguous store four structures from four vectors, scalar + scalar	ST4B, ST4H, ST4W	6	2/3	L01, I, V	-
Non temporal store, scalar + imm	STNT1B, STNT1D, STNT1H, STNT1W	2	2	L01, V	-
Non temporal store, scalar + scalar	STNT1B, STNT1D, STNT1H, STNT1W	2	2	L01, V	-
Scatter non temporal store, vector + scalar 32-bit element size	STNT1B, STNT1H, STNT1W	2	1	L01, V	-

Copyright © 2023-2024 Arm Limited (or its affiliates). All rights reserved.

Instruction Group	SVE Instruction	Exec Latency	Execution Throughput	Utilized Pipelines	Notes
Scatter non temporal store, vector + scalar 64-bit element size	STNT1B, STNT1D, STNT1H, STNT1W	2	2	L01, V	-
Scatter store vector + imm 32-bit element size	ST1B, ST1H, ST1W	2	1	L01, V	-
Scatter store vector + imm 64-bit element size	ST1B, ST1D, ST1H, ST1W	2	2	L01, V	-
Scatter store, 32-bit scaled offset	ST1H, ST1W	2	1	L01, V	-
Scatter store, 32-bit unpacked unscaled offset	ST1B, ST1D, ST1H, ST1W	2	2	L01, V	-
Scatter store, 32-bit unpacked scaled offset	ST1D, ST1H, ST1W	2	2	L01, V	-
Scatter store, 32-bit unscaled offset	ST1B, ST1H, ST1W	2	1	L01, V	-
Scatter store, 64-bit scaled offset	ST1D, ST1H, ST1W	2	2	L01, V	-
Scatter store, 64-bit unscaled offset	ST1B, ST1D, ST1H, ST1W	2	2	L01, V	-

3.30 SVE Miscellaneous instructions

Table 3-29 SVE miscellaneous instructions

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

Notes:

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

Copyright © 2023-2024 Arm Limited (or its affiliates). All rights reserved.

3.31 SVE Cryptographic instructions

Table 3-48 SVE cryptographic instructions

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

Copyright © 2023-2024 Arm Limited (or its affiliates). All rights reserved.

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®-A725 core.

The dispatch stage can process up to 5 MOPs per cycle and dispatch up to 10 µOPs per cycle, with the following limitations on the number of µOPs of each type that may be simultaneously dispatched.

- Up to 4 µOPs utilizing the S or B pipelines
- Up to 4 µOPs utilizing the M pipelines
- Up to 2 µOPs utilizing the M0 pipelines
- Up to 2 µOPs utilizing the V0 pipeline
- Up to 2 µOPs utilizing the V1 pipeline
- Up to 5 µOPs utilizing the L pipelines

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

4.2 Optimizing memory routines

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

- Unroll the loop to include multiple load and store operations per iteration, minimizing the overheads of looping.
- Align stores on 32B boundary wherever possible.
- Use non-writeback forms of LDP and STP instructions interleaving them like shown in the example below:

```
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
```

Copyright © 2023-2024 Arm Limited (or its affiliates). All rights reserved.

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

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

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

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

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

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

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

Copyright © 2023-2024 Arm Limited (or its affiliates). All rights reserved.

4.3 Load/Store alignment

The Armv8-A architecture allows many types of load and store accesses to be arbitrarily aligned. The Cortex®-A725 core handles most unaligned accesses without performance penalties. However, there are cases which could reduce bandwidth or incur additional latency, as described below.

- Load operations that cross a cache-line (64-byte) boundary.
- Quad-word load operations that are not 4B aligned.
- Store operations that cross a 32B boundary.

4.4 Store to Load Forwarding

The Cortex®-A725 core allows data to be forwarded from store instructions to a load instruction with the restrictions mentioned below:

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

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

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

Copyright © 2023-2024 Arm Limited (or its affiliates). All rights reserved.

4.5 AES encryption/decryption

The Cortex®-A725 core can issue two AESE/AESMC/AESD/AESIMC instruction every cycle (fully pipelined) with an execution latency of two cycles. Plus note, pairs of dependent AESE/AESMC and AESD/AESIMC instructions are higher performance when they are adjacent in the program code and both instructions use the same destination register since they are fused (see Section 4.11 on Instruction Fusion). This means encryption or decryption for at least four data chunks should be interleaved for maximum performance, reaching then virtually 4 instructions issue rate in this case:

```
AESE  data0, key_reg
AESMC data0, data0
AESE  data1, key_reg
AESMC data1, data1
AESE  data2, key_reg
AESMC data2, data2
AESE  data3, key_reg
AESMC data3, data3
AESE  data0, key_reg
AESMC data0, data0
...
```

Copyright © 2023-2024 Arm Limited (or its affiliates). All rights reserved.

4.6 Region based fast forwarding

The forwarding logic in the V pipelines is optimized to provide optimal latency for instructions which are expected to commonly forward to one another.

This is defined in the following table.

Table 4-1 Optimized INT forwarding regions

Region	Instruction Types	Notes
1	ASIMD/SVE integer ALU, ASIMD/SVE integer shift, ASIMD/scalar insert and move, ASIMD/SVE integer abs/cmp/max/min, ASIMD/SVE AES, ASIMD/SVE polynomial multiply, ASIMD/SVE integer reduction, SHA3 and PERM instructions in part 3.19 see Note 2	1
2	ASIMD/SVE integer mul/mac	2
3	ASIMD/SVE Crypto, SHA1/SHA256	1

Table 4-2 Optimized FP forwarding regions

Region	Instruction Types	Notes
1	FP/ASIMD/SVE floating-point multiply, FP/ASIMD/SVE floating point multiply-accumulate, FP/ASIMD/SVE compare, FP/ASIMD/SVE add/sub and PERM instructions in part 3.19 see Note 2	1
2	ASIMD/SVE BFDOT and BFMMLA instructions	

Notes:

1. ASIMD/SVE extract narrow, saturating instructions are excluded from this region and ASIMD/SVE integer reduction are only consumer forward from this region
2. ASIM/SVE INT multiply accumulate only fast forward to accumulation source

The following instructions are not part of any region:

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

In addition to the regions mentioned in the table above, all instructions in regions INT1 and FP1 can fast forward to FP/ASIMD/SVE stores plus FP/ASIMD vector to integer register transfers, ASIMD converts that write to general purpose registers and PERM instructions in part 3.19 see Note 2.

More special notes about the forwarding region in Table 4-1 Optimized INT forwarding regions:

- Complex shift by immediate/register and shift accumulate instructions cannot be producers (see sections 3.16 and 3.25) in region INT1.
- Extract narrow, saturating instructions cannot be producers (see sections 3.19 and 3.25) in region INT1.

Copyright © 2023-2024 Arm Limited (or its affiliates). All rights reserved.

- Absolute difference accumulate and pairwise add and accumulate instructions cannot be producers (see sections 3.16 and 3.25) in region INT1.

More special notes about the forwarding region in Table 4-2 Optimized FP forwarding regions:

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

It is not advisable to interleave instructions belonging to different regions. Also, certain instructions can only be producers or consumers in a particular region but not both (see footnote for Table 4-1 Optimized INT forwarding regions and Table 4-2 Optimized FP forwarding regions). For example, the code below interleaves producers and consumers from regions INT1 and INT2. This will result in an additional latency of 1 cycle as seen by MUL.

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

These fast forwarding regions described in Table 4-1 Optimized INT forwarding regions and Table 4-2 Optimized FP forwarding regions are forming two clusters: cluster FP and cluster INT. Intercluster communication requires one cycle penalty. For example, the code below

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

4.7 Branch instruction alignment

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



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

Cortex®-A725 core prediction is optimized to handle aligned 32-byte instruction region containing no branches. For best performance and power efficiency, avoid diluting branches over aligned instruction regions.

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

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

Copyright © 2023-2024 Arm Limited (or its affiliates). All rights reserved.

4.8 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.

4.9 Special register access

The Cortex®-A725 core performs register renaming for general purpose registers to enable speculative and out-of-order instruction execution. But most special-purpose registers are not renamed. Instructions that read or write non-renamed registers are subjected to one or more of the following additional execution constraints.

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

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

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

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

Table 4-3 Special-purpose register read accesses

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

Notes:

1. The NZCV and SP registers are fully renamed.

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

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

Copyright © 2023-2024 Arm Limited (or its affiliates). All rights reserved.

Table 4-3 Special-purpose register write accesses

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

Notes:

1. The NZCV and SP registers are fully renamed.

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

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

4.10 Instruction fusion

The Cortex®-A725 core can accelerate certain instruction pairs in an operation called fusion. Specific instruction pairs that can be fused are as follows:

AESE + AESMC (see Section 4.6 on AES Encryption/Decryption)
AESD + AESIMC (see Section 4.6 on AES Encryption/Decryption)
CMP/CMN (immediate) + B.cond
CMP/CMN (register Rn != ZR) + B.cond
TST (immediate) + B.cond
TST (register) + B.cond
BICS ZR (register) + B.cond
CMP (immediate) + CSEL
CMP (register) + CSEL
CMP (immediate) + CSET
CMP (register) + CSET
BTI + Integer DP/BR/BLR/RET/B uncond/CBZ/TBZ
SHL + SRI (both scalar or both vector)
FCMP + AXFLAG
MOVPRFX + supported SVE instruction

These instruction pairs must be adjacent to each other in program code. For CMP, CMN, TST fusion is allowed for shifted and/or extended register forms. For CMP, CMN, TST and BICS, there

Copyright © 2023-2024 Arm Limited (or its affiliates). All rights reserved.

are restrictions on immediate values for both instructions of the pair for which fusion is supported. Other particular restrictions apply on instruction fusion.

4.11 Zero Latency Instructions

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

- MOV Xd, #{12{1'b0},imm[3:0]}
- MOV Xd, XZR
- MOV Wd, #{12{1'b0},imm[3:0]}
- MOV Wd, WZR
- MOV Hd, WZR
- MOV Hd, XZR
- MOV Sd, WZR
- MOV Dd, XZR
- MOVI Dd, #0
- MOVI Vd.2D, #0
- MOV Wd, Wn
- MOV Xd, Xn
- FMOV Sd, Sn
- FMOV Dd, Dn
- MOV Vd, Vn (vector)
- MOV Zd.D, Zn.D
- PTRUE
- PFALSE
- SETFFR

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

Copyright © 2023-2024 Arm Limited (or its affiliates). All rights reserved.

4.12 TLB-access latencies

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

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

A miss in the L1 data TLB followed by a hit in the L2 TLB has a 5-cycle penalty compared to a hit in the L1 data TLB. This penalty can be increased depending on the arbitration of pending requests.

4.13 Cache-access latencies

The Cortex®-A725 core pipeline is optimized for low latency and high bandwidth. The following table lists the latencies for the different levels of cache.

Table 4-4 Cortex®-A725 core cache access latencies

Scenario	Cycle count
Level-1 Cache Hit	4 core cycles
Level-2 Cache Hit	9 core cycles if 128kB, 256kB, 512kB L2 size 10 core cycles if 1MB L2 size
Level-3 Cache Hit	19.5 core cycles + 14.5 DSU cycles
Level-1 Cache Hit in another Cortex®-A725 core in the same cluster	38 core cycles + 22.5 DSU cycles
Level-2 Cache Hit in another Cortex®-A725 core in the same cluster	32 core cycles if 128kB, 256kB, 512kB L2 size + 22.5 DSU cycles 33 core cycles if 1MB L2 size + 22.5 DSU cycles
Level-3 Cache Miss, DMC access	19.5 core cycles + 15.5 DSU cycles + 2 SYS cycles + system latency

The information in Table 4-4 Cortex®-A725 core cache access latencies is based on the assumptions that:

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

Copyright © 2023-2024 Arm Limited (or its affiliates). All rights reserved.

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

4.14 Cache maintenance operation

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

4.15 Memory Tagging - Tagging Performance

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

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

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

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

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

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

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

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

Copyright © 2023-2024 Arm Limited (or its affiliates). All rights reserved.

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

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

4.16 Memory Tagging - Synchronous Mode

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

It is recommended to use asynchronous mode for better performance.

4.17 Complex ASIMD and SVE instructions

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

ASIMD

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

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

LD4, multiple 4-element structures, quad form, element size less than 64b.

LD4, multiple 4-element structures, quad form, element size less than 64b, , post indexed addressing mode.

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

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

SVE

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

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

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

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

Copyright © 2023-2024 Arm Limited (or its affiliates). All rights reserved.

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

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

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

4.18 MOVPRFX fusion

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

Table 4-5 MOVPRFX unpredicated fusion

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

Copyright © 2023-2024 Arm Limited (or its affiliates). All rights reserved.

Instruction Group	SVE Instruction	Notes
Copy	CPY	Only the SIMD&FP scalar and immediate merging forms are fusible
Divides	SDIV, SDIVR, UDIV, UDIVR	-
Dot product	SDOT, UDOT, SUDOT, USDOT	Only the vector form is fusible
Extract/insert operation	INSR	Only the SIMD&FP scalar form is fusible
Logical	AND, BIC, EON, EOR, EORBT, EORTB, MOV, ORN, ORR	For AND, BIC, EOR and ORR, only the immediate and vector, predicated forms are fusible For MOV onty the SIMD&FP scalar form predicated is fusible For ORN only the immediate form is fusible
Max/min, basic and pairwise	SMAX, SMIN, UMAX, UMIN	Only the immediate and vector, predicated forms are fusible
Matrix multiply-accumulate	SMMLA, UMMLA, USMMLA	-
Multiply	MUL, SMULH, UMULH	For MUL, only the immediate and vector, predicated forms are fusible. For the others, only the predicated form is fusible.
Multiply accumulate	MLA, MLS, MAD, MSB	For MLA, MLS only the vector forms are fusible
Multiply accumulate long	SMLALB, SMLALT, SMLSLB, SMLSLT, UMLALB, UMLALT, UMLSLB, UMLSLT	Only the vector form is fusible
Multiply accumulate saturating doubling long regular	SQDMLALB, SQDMLALT, SQDMLALBT, SQDMLSLB, SQDMLSLT, SQDMLSLBT	For SQDMLALB, SQDMLALT, SQDMLSLB, SQDMLSLT only the vector forms are fusible
Multiply saturating rounding doubling regular/complex accumulate	SQRDMLAH, SQRDMLSH, SQRDCMLAH	Only the vector form is fusible
Predicate counting, vector form	DECH, DECW, DECD, INCH, INCW, INCD, SQDECH, SQDECW, SQDECD, SQINCH, SQINCW, SQINCD, UQDECH, UQDECW, UQDECD, UQINCH, UQINCW, UQINCD	Only the vector form is fusible
Reciprocal estimate	URECPE, URSQRTE	-
Reverse, vector	REVB, REVH, REVW	-
Floating point Instructions		
Floating point absolute value/difference	FABD, FABS	-
Floating point arithmetic	FADD, FNEG, FSUB, FSUBR	For FADD, FSUB, FSUBR only the immediate and vector, predicated forms are fusible.
Floating point complex add	FCADD	-
Floating point complex multiply add	FCMLA	Only the vector form is fusible

Copyright © 2023-2024 Arm Limited (or its affiliates). All rights reserved.

Instruction Group	SVE Instruction	Notes
Floating point convert	FCVT, FCVTX	-
Floating point base2 log	FLOGB	-
Floating point convert to integer	FCVTZS, FCVTZU	-
Floating point copy	FCPY, FMOV	Only the predicated form is fusible
Floating point divide	FDIV, FDIVR	-
Floating point min/max	FMAX, FMIN, FMAXNM, FMINNM	-
Floating point multiply	FSCALE, FMUL, FMULX	For FMUL, only the immediate and vector, predicated forms are fusible
Floating point multiply accumulate	FMLA, FMLS, FMAD, FMSB, FNMAD, FNMLA, FNMLS, FNMSB	For FMLA, FMLS only the vector forms are fusible
Floating point multiply add/sub accumulate long	FMLALB, FMLALT, FMLSLB, FMLSLT	Only the vector form is fusible
Floating point reciprocal estimate	FRECPX	-
Floating point round to integral	FRINTA, FRINTI, FRINTM, FRINTN, FRINTP, FRINTX, FRINTZ	-
Floating point square root	FSQRT	-
Floating point trigonometric multiply add	FTMAD	-
BF16 Instructions		
Dot product	BFDOT	Only the vector form is fusible
Matrix multiply accumulate	BFMMLA	-
Multiply accumulate long	BFMLALB, BFMLALT	Only the vector form is fusible
Scalar convert, F32 to BF16	BFCVT	-
Cryptographic Instructions		
Crypto SHA3 ops	BCAX, EOR3, XAR	-

Copyright © 2023-2024 Arm Limited (or its affiliates). All rights reserved.

Table 4-6 MOVPRFX predicated fusion

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

Copyright © 2023-2024 Arm Limited (or its affiliates). All rights reserved.

Appendix A Revisions

This appendix describes the technical changes between released issues of this document.

Table A-1: Issue 1.0

Change	Location	Affects
First Confidential limited access release for r0p0	-	r0p0

Table A-2: Issue 2.0

Change	Location	Affects
First Confidential early access release for r0p1	-	r0p1

Table A-3: Issue 3.0

Change	Location	Affects
Second Non-Confidential early access release for r0p1	-	r0p1
Updated product name	Throughout document	r0p1
Removed fixes of PMULL instructions played in AES module since played in PMUL	Section 3.16 and 3.22	r0p1
Updated AES encryption/decryption description	Section 4.5	r0p1
Updated FCMLA latency	Section 3.17	r0p1
Fixed AUT and LDRA latencies	Section 3.6	r0p1
Fixed core cycles for Level-3 Cache accesses	Section 4.13	r0p1
Removed few unsupported MOVPRFX fusions	Section 4.18	r0p1
Fixed FCVT and MOV instructions from vec to gen	Section 3.13 and 3.19	r0p1

Table A-4: Issue 4.0

Change	Location	Affects
First Non-Confidential release for r0p2	-	r0p2
Updated revision value	-	r0p2
Removed 'Optimizing general-purpose register spills and fills' section	Section 4.2	r0p1, r0p2

Copyright © 2023-2024 Arm Limited (or its affiliates). All rights reserved.

Copyright © 2023-2024 Arm Limited (or its affiliates). All rights reserved.