



Arm® Cortex®-A510 Core

Revision: r0p3

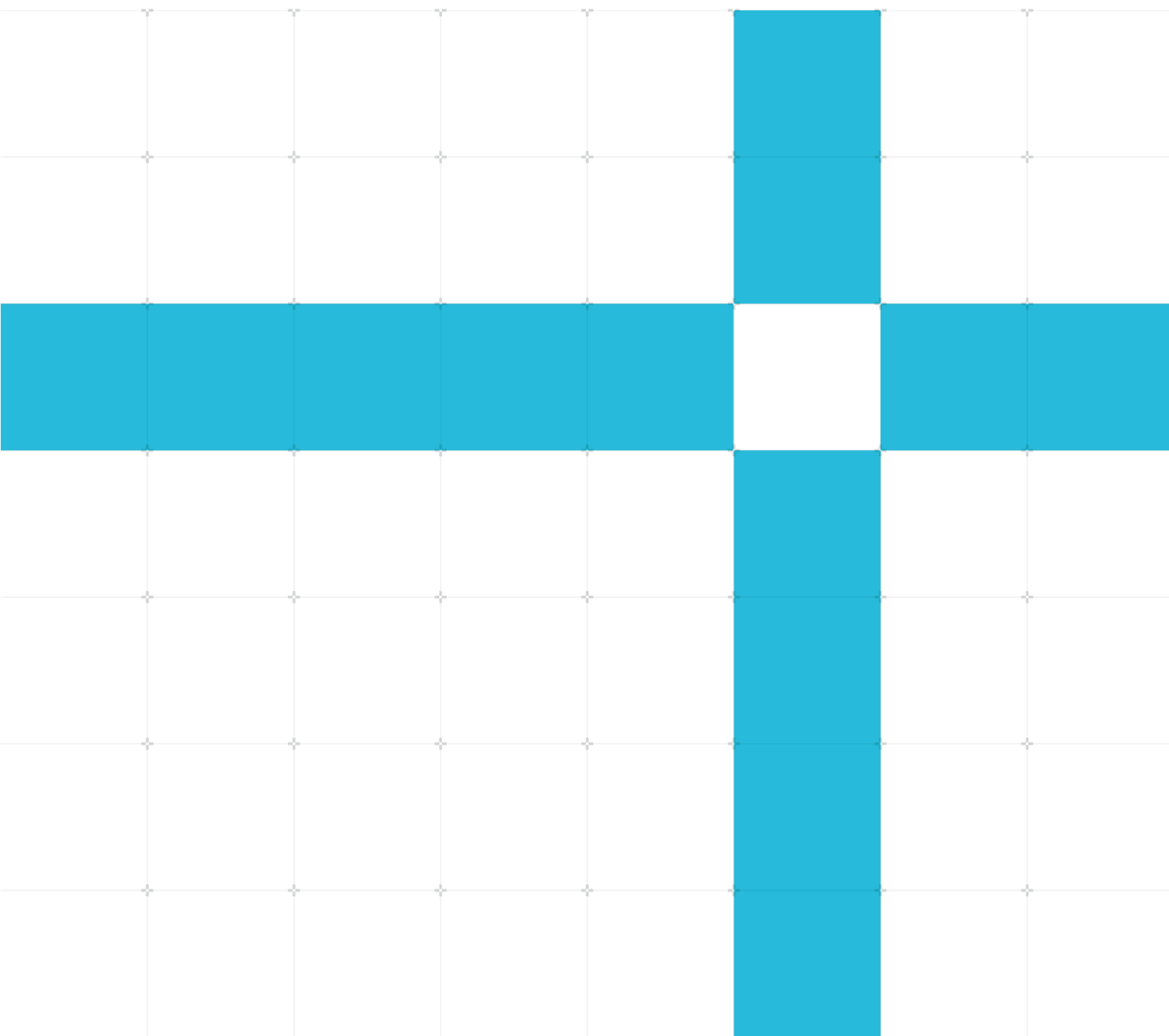
Software Optimization Guide

Non-Confidential

Copyright © 2020-2021 Arm Limited (or its affiliates).
All rights reserved.

Issue 3.0

PJDOC-466751330-536816



Arm® Cortex-A510 Core Software Optimization Guide

Copyright © 2020-2021 Arm Limited (or its affiliates). All rights reserved.

Release information

Document history

Issue	Date	Confidentiality	Change
1.0	12 Jan 2021	Confidential	First release for r0p2
2.0	25 May 2021	Confidential	First release for r0p3
3.0	17 Dec 2021	Non-Confidential	Second release for r0p3

Non-Confidential Proprietary Notice

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

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

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, has undertaken no analysis to identify or understand the scope and content of, patents, copyrights, trade secrets, or other rights.

This document may include technical inaccuracies or typographical errors.

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.

This document consists solely of commercial items. You shall be responsible for ensuring that any 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 the Agreement shall prevail.

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. All rights reserved. Other brands and names mentioned in this document may be the trademarks of their respective owners. Please follow Arm's trademark usage guidelines at <https://www.arm.com/company/policies/trademarks>.

Copyright © 2020-2021 Arm Limited (or its affiliates). All rights reserved.

Arm Limited. Company 02557590 registered in England.
110 Fulbourn Road, Cambridge, England CB1 9NJ.
(LES-PRE-20349)

Confidentiality Status

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

Unrestricted Access is an Arm internal classification.

Product Status

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

Inclusive language commitment

This document includes language that can be offensive. We will replace this language in a future issue of this document. To report offensive language in this document, email terms@arm.com.

Contents

1	Introduction	6
1.1	Product revision status.....	6
1.2	Intended audience	6
1.3	Conventions	6
1.3.1	Glossary	6
1.3.2	Terms and abbreviations.....	6
1.3.3	Typographical conventions	7
1.4	Additional reading	7
1.5	Feedback	8
1.5.1	Feedback on this product.....	8
1.5.2	Feedback on content	8
2	Overview	9
2.1	Pipeline overview.....	10
2.2	Configuration	11
2.2.1	Cryptographic extensions.....	11
2.2.2	Dual or single core	12
2.2.3	Vector datapath size	12
3	Instruction characteristics	13
3.1	Instruction tables	13
3.2	Branch Instructions.....	13
3.3	Arithmetic and logical instructions	14
3.4	Divide and multiply instructions	14
3.5	Pointer authentication instructions	15
3.6	Miscellaneous data-processing instructions.....	16
3.7	Load instructions.....	16
3.8	Store instructions.....	17
3.9	Tag data processing.....	18
3.10	Tag load instructions.....	19
3.11	Tag store instructions.....	19
3.12	FP scalar data processing instructions	20
3.13	FP scalar miscellaneous instructions.....	21

3.14	FP scalar load instructions.....	21
3.15	FP scalar store instructions.....	22
3.16	ASIMD Integer instructions	24
3.17	ASIMD FP data processing instructions	26
3.18	ASIMD BFloat16 (BF16) instructions.....	29
3.19	ASIMD miscellaneous instructions	29
3.20	ASIMD load instructions	30
3.21	ASIMD store instructions	32
3.22	Cryptography extensions	34
3.23	CRC.....	35
3.24	SVE Predicate instructions.....	35
3.25	SVE Integer instructions.....	37
3.26	SVE FP data processing instructions	43
3.27	SVE BFloat16 (BF16) instructions.....	45
3.28	SVE Load instructions.....	45
3.29	SVE Store instructions	47
3.30	SVE Miscellaneous instructions	49
3.31	SVE Cryptography instructions.....	49
4	Special considerations	51
4.1	Issue constraints.....	51
4.2	Instruction fusion.....	51
4.3	Branch instruction alignment.....	52
4.4	Load / Store Alignment	52
4.5	Memory Tagging Extensions	52
4.6	Memory routines.....	52
4.7	Cache maintenance operations.....	54
4.8	Shared VPU	54
4.9	AES encryption / decryption	54

1 Introduction

1.1 Product revision status

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

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

pn 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 Conventions

The following subsections describe conventions used in Arm documents.

1.3.1 Glossary




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

See the [Arm® Glossary](#) for more information.

1.3.2 Terms and abbreviations

Term	Meaning
ALU	Arithmetic and Logical Unit
ASIMD	Advanced SIMD
FP	Floating-point
GPR	General Purpose Register
SQRT	Square Root
SVE	Scalable Vector instruction Extension (SVE or SVE2)
VPR	Vector Processing Register; FP/ASIMD/SVE registers
VPU	Vector Processing Unit

1.3.3 Typographical conventions

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

1.4 Additional reading

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

Table 1-1 Arm publications

Document name	Document ID	Licensee only Y/N
Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile	DDI 0487	N
Arm® Cortex-A510 Core Technical Reference Manual	101604	N
Arm® Cortex-A510 Core Configuration and Integration Manual	101605	Y

1.5 Feedback

Arm welcomes feedback on this product and its documentation.

1.5.1 Feedback on this product

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

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

1.5.2 Feedback on content

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

- The title Arm® Cortex-A510 Core Software Optimization Guide.
- The number PJDOC-466751330-536816.
- If applicable, the page number(s) to which your comments refer.
- A concise explanation of your comments.

Arm also welcomes general suggestions for additions and improvements.



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

2 Overview

The Cortex-A510 core is a high-efficiency, low-power product that implements the Arm®v9.0-A architecture. The Arm®v9.0-A architecture extends the architecture defined in the Arm®v8-A architectures up to Arm®v8.5-A.

The key features of the Cortex-A510 core are:

- Implementation of the Arm®v9.0-A A64 instruction set
- AArch64 Execution state at all Exception levels, EL0 to EL3
- Separate L1 data and instruction side memory systems with a Memory Management Unit (MMU)
- In-order pipeline with direct and indirect branch prediction
- Generic Interrupt Controller (GIC) CPU interface to connect to an external interrupt distributor
- Generic Timer interface that supports a 64-bit count input from an external system counter
- Implementation of the Reliability, Availability, and Serviceability (RAS) Extension
- 128-bit Scalable Vector Extension (SVE) and SVE2 SIMD instruction set, offering Advanced SIMD (ASIMD) and floating-point (FP) architecture support
- Support for the optional Cryptographic Extension, which is licensed separately
- Activity Monitoring Unit (AMU)

This document describes the elements of the Cortex-A510 core micro-architecture that influence the software performance so that software and compilers can be optimized accordingly.

2.1 Pipeline overview

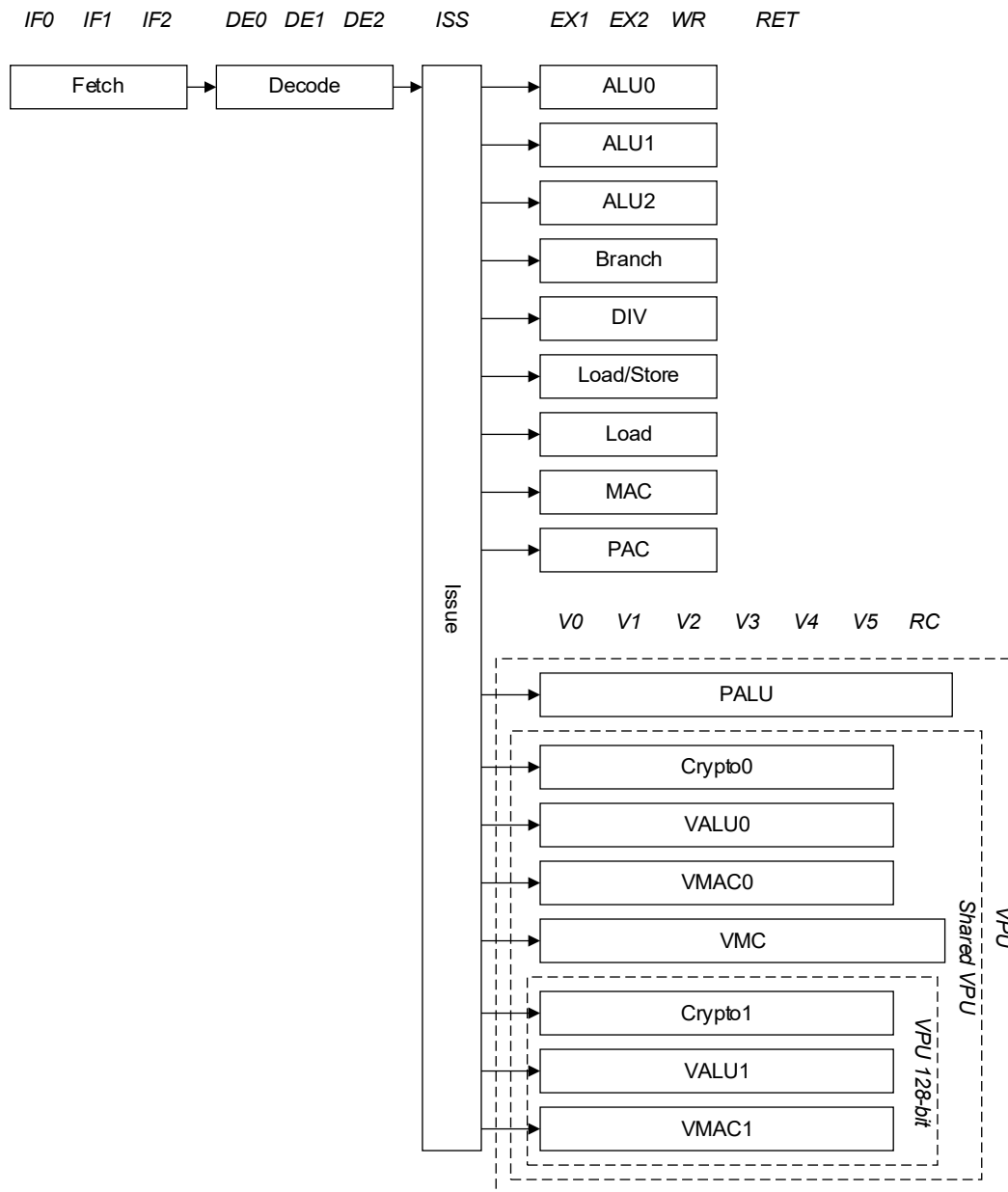


Figure 1 Cortex-A510 core pipeline

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

Pipeline	Instructions
ALU0, ALU1, ALU2	Arithmetic and logic
Branch	Branch

Pipeline	Instructions
Crypto0	Cryptography Supports 1x128-bit operation. This pipeline is shared for dual core configuration. Present only for implementations configured with Cryptographic extensions enabled.
Crypto1	Cryptography Supports 1x128-bit operation. This pipeline is shared for dual core configuration. Present only for implementations configured with Cryptographic extensions enabled and a Vector datapath size of 2x128-bit.
DIV	Integer scalar division (iterative)
Load/Store	Load and store
Load	Load
MAC	Multiply accumulate
PAC	Pointer Authentication
PALU	Predicate register arithmetic and logic
VALU0	Addition, logic and shift for ASIMD, FP, Neon, and SVE Supports 2x64-bit or 1x128-bit operations. This pipeline is shared for dual core configuration.
VALU1	Addition, logic and shift for ASIMD, FP, Neon, and SVE Supports 2x64-bit or 1x128-bit operations. This pipeline is shared for dual core configuration. Present only for implementations configured with a Vector datapath size of 2x128-bit.
VMAC0	Multiply accumulate for ASIMD, FP, Neon, and SVE Supports 2x64-bit or 1x128-bit operations. This pipeline is shared for dual core configurations.
VMAC1	Multiply accumulate for ASIMD, FP, Neon, and SVE Supports 2x64-bit or 1x128-bit operations. This pipeline is shared for dual core configurations. Present only for implementations configured a Vector datapath size of 2x128-bit configurations.
VMC	Cryptography and iterative multi cycle instruction (e.g. bit permutation, division, and square root) Supports 2x64-bit or 1x128-bit operations. This pipeline is shared for dual core configurations.

2.2 Configuration

2.2.1 Cryptographic extensions

Cryptographic extensions can be included in the VPU.

See `ID_AA64ISAR0_EL1` to determine if Cryptographic extensions are enabled.

2.2.2 Dual or single core

Cortex-A510 cores can be grouped into dual-core complexes, or instantiated as single-core complexes. Dual-core complexes share the L2 cache and VPU, while single-core complexes have a dedicated L2 cache and VPU.

Figure 1 highlights the VPU pipelines shared between Cortex-A510 cores in a complex.

See `IMP_CPUCFR_EL1.Cores` to determine the number of cores in the complex.

2.2.3 Vector datapath size

The size of the vector datapaths can be 2×64-bit or 2×128-bit. The selected option applies to all cores in the complex.

Figure 1 highlights the VPU pipelines that are only instantiated for a 2x128-bit configuration.

See `IMP_CPUCFR_EL1.VPU` to determine the configured VPU datapath width.

3 Instruction characteristics

3.1 Instruction tables

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

In the tables below:

- *Exec Latency* is the minimum latency seen by an operation dependent on an instruction in the described group.
- *Load Latency* is the minimum latency seen by an operation dependent on the load. It is assumed the memory access hits in the L1 Data Cache.
- *Execution Throughput* is maximum throughput (in instructions per cycle) of the specified instruction group that can be achieved in the entirety of the Cortex-A510 microarchitecture.

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

3.2 Branch Instructions

Table 3-1 AArch64 Branch instructions

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

3.3 Arithmetic and logical instructions

Table 3-2 AArch64 Arithmetic and logical instructions

Instruction Group	AArch64 Instruction	Exec Latency	Execution Throughput	Utilized Pipeline
ALU, basic ^[1]	ADD, ADC, AND, BIC, EON, EOR, ORN, ORR, SUB, SBC	1	3	ALU
ALU, basic, flagset ^[1]	ADDS, ADCS, ANDS, BICS, SUBS, SBCS	1	3	ALU
ALU, extend and shift	ADD{S}, AND{S}, BIC{S}, EON, EOR, ORN, ORR, SUB{S}	2	3	ALU
Conditional compare ^[1]	CCMN, CCMP	1	3	ALU
Conditional select ^[1]	CSEL, CSINC, CSINV, CSNEG	1	3	ALU
Logical, shift, no flagset ^[1]	AND, BIC, EON, EOR, ORN, ORR	1	3	ALU
Logical, shift, flagset ^[1]	AND{S}, BIC{S}	1	3	ALU
Shift ^[1]	ASL, ASR, LSL, LSR, ROR	1	3	ALU

Notes:

1. Instructions with dependencies may be co-issued

3.4 Divide and multiply instructions

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

Multiply-accumulate pipelines support late-forwarding of accumulate operands from similar instructions allowing a typical sequence of multiply-accumulate instructions 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.

Table 3-3 AArch64 Divide and multiply instructions

Instruction Group	AArch64 Instruction	Exec Latency	Execution Throughput	Utilized Pipeline
Divide, W-form	SDIV, UDIV	8	1/8	DIV
Divide, X-form	SDIV, UDIV	16	1/16	DIV
Multiply accumulate, W-form	MADD, MSUB, MUL	3 (1)	1	MAC
Multiply accumulate, X-form	MADD, MSUB, MUL	4-5 (2-3)	1/3 - 1/2	MAC

Instruction Group	AArch64 Instruction	Exec Latency	Execution Throughput	Utilized Pipeline
Multiply accumulate long	SMADDL, SMSUBL, UMADDL, UMSUBL	3 (1)	1	MAC
Multiply high	SMULH, UMULH	6	1/4	MAC

3.5 Pointer authentication instructions

Table 3-4 AArch64 Pointer authentication instructions

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

3.6 Miscellaneous data-processing instructions

Table 3-5 AArch64 miscellaneous data-processing instructions

Instruction Group	AArch64 Instruction	Exec Latency	Execution Throughput	Utilized Pipeline
Address generation	ADR, ADRP	1	3	ALU
Bitfield extract, one reg	EXTR	1	3	ALU
Bitfield extract, two regs	EXTR	2	3	ALU
Bitfield move, basic	SBFM, UBFM	2	3	ALU
Bitfield move, insert	BFM	2	3	ALU
Convert floating-point condition flags ^[1]	AXFLAG, XAFLAG	1	3	ALU
Flag manipulation instructions	SETF8, SETF16, RMIF, CFINV	1	3	ALU
Count leading	CLS, CLZ	1	3	ALU
Move immed ^[1]	MOVN, MOVK, MOVZ	1	3	ALU
Reverse bits/bytes ^[1]	RBIT, REV, REV16, REV32	1	3	ALU
Rotate, mask, insert flags	RMIF	2	3	ALU
Variable shift	ASRV, LSLV, LSRV, RORV	1	3	ALU

Notes:

1. Instructions with dependencies may be co-issued

3.7 Load instructions

The latencies shown in Table 3-6 assume the memory access hits in the Level 1 Data Cache.

Base register updates are done in parallel to the operation.

Table 3-6 AArch64 Load instructions

Instruction Group	AArch64 Instruction	Load Latency	Execution Throughput	Utilized Pipeline
Load register, literal	LDR, LDRSW, PRFM	2	2	Load/Store, Load
Load register, unscaled immed	LDUR, LDURB, LDURH, LDURSB, LDURSH, LDURSW, PRFUM	2	2	Load/Store, Load
Load register, immed post-index	LDR, LDRB, LDRH, LDRSB, LDRSH, LDRSW	2	2	Load/Store, Load

Instruction Group	AArch64 Instruction	Load Latency	Execution Throughput	Utilized Pipeline
Load register, immed pre-index	LDR, LDRB, LDRH, LDRSB, LDRSH, LDRSW	2	2	Load/Store, Load
Load register, immed unprivileged	LDTR, LDTRB, LDTRH, LDTRSB, LDTRSH, LDTRSW	2	2	Load/Store, Load
Load register, unsigned immed	LDR, LDRB, LDRH, LDRSB, LDRSH, LDRSW, PRFM	2	2	Load/Store, Load
Load register, register offset, basic	LDR, LDRB, LDRH, LDRSB, LDRSH, LDRSW, PRFM	2	2	Load/Store, Load
Load register, register offset, scale by 4/8	LDR, LDRSW, PRFM	2	2	Load/Store, Load
Load register, register offset, scale by 2	LDRH, LDRSH	2	2	Load/Store, Load
Load register, register offset, extend	LDR, LDRB, LDRH, LDRSB, LDRSH, LDRSW, PRFM	2	2	Load/Store, Load
Load register, register offset, extend, scale by 4/8	LDR, LDRSW, PRFM	2	2	Load/Store, Load
Load register, register offset, extend, scale by 2	LDRH, LDRSH	2	2	Load/Store, Load
Load pair, signed immed offset, normal, W-form	LDP, LDNP	2	2	Load/Store, Load
Load pair, signed immed offset, normal, X-form	LDP, LDNP	2	2	Load/Store, Load
Load pair, signed immed offset, signed words	LDPSW	2	2	Load/Store, Load
Load pair, immed post-index or immed pre-index, normal, W-form	LDP	2	1	Load/Store, Load
Load pair, immed post-index or immed pre-index, normal, X-form	LDP	2	1	Load/Store, Load
Load pair, immed post-index, signed words	LDPSW	2	1	Load/Store, Load

3.8 Store instructions

Base register updates are done in parallel to the operation.

Table 3-7 AArch64 Store instructions

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

3.9 Tag data processing

Table 3-8 AArch64 Tag data processing instructions

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

3.10 Tag load instructions

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

Table 3-9 AArch64 Tag load instructions

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

3.11 Tag store instructions

Base register updates are done in parallel to the operation.

Table 3-10 AArch64 Tag store instructions

Instruction Group	AArch64 Instruction	Exec Latency	Execution Throughput	Utilized Pipeline
Store allocation tags to one or two granules, post-index	STG, ST2G	-	1	Load/Store
Store allocation tags to one or two granules, pre-index	STG, ST2G	-	1	Load/Store
Store allocation tags to one or two granules, signed offset	STG, ST2G	-	1	Load/Store
Store allocation tag to one or two granules, zeroing, post-index	STZG, STZ2G	-	1	Load/Store
Store Allocation Tag to one or two granules, zeroing, pre-index	STZG, STZ2G	-	1	Load/Store
Store allocation tag to two granules, zeroing, signed offset	STZG, STZ2G	-	1	Load/Store
Store allocation tag and reg pair to memory, post-Index	STGP	-	1	Load/Store
Store allocation tag and reg pair to memory, pre-Index	STGP	-	1	Load/Store
Store allocation tag and reg pair to memory, signed offset	STGP	-	1	Load/Store
Store multiple allocation tags	STGM	-	1	Load/Store

Instruction Group	AArch64 Instruction	Exec Latency	Execution Throughput	Utilized Pipeline
Store multiple allocation tags, zeroing	STZGM	-	1	Load/Store

3.12 FP scalar data processing instructions

Table 3-11 AArch64 FP data processing instructions

Instruction Group	AArch64 Instruction	Exec Latency	Execution Throughput	Utilized Pipeline
FP absolute value	FABS, FABD	4	2	VALU
FP arithmetic	FADD, FSUB	4	2	VALU
FP compare	FCCMP{E}, FCMP{E}	3	2	VALU
FP divide, H-form ¹	FDIV	8	2/5	VMC
FP divide, S-form ¹	FDIV	13	2/10	VMC
FP divide, D-form ¹	FDIV	22	2/19	VMC
FP min/max	FMIN, FMINNM, FMAX, FMAXNM	4	2	VALU
FP multiply	FMUL, FNMUL	4	2	VMAC
FP multiply accumulate	FMADD, FMSUB, FNMADD, FNMSUB	4	2	VMAC
FP negate	FNEG	4	2	VALU
FP round to integral	FRINTA, FRINTI, FRINTM, FRINTN, FRINTP, FRINTX, FRINTZ, FRINT32X, FRINT64X, FRINT32Z, FRINT64Z	4	2	VALU
FP select	FCSEL	3	2	VALU
FP square root, H-form	FSQRT	8	2/5	VMC
FP square root, S-form	FSQRT	12	2/9	VMC
FP square root, D-form	FSQRT	22	2/19	VMC

Notes:

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

3.13 FP scalar miscellaneous instructions

Table 3-12 AArch64 FP miscellaneous instructions

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

3.14 FP scalar load instructions

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

Base register updates are done in parallel to the operation.

Table 3-13 AArch64 FP load instructions

Instruction Group	AArch64 Instruction	Load Latency	Execution Throughput	Utilized Pipeline
Load vector reg, literal, S/D/Q forms	LDR	3	2	Load/Store, Load
Load vector reg, unscaled immed	LDUR	3	2	Load/Store, Load
Load vector reg, immed post-index	LDR	3	2	Load/Store, Load
Load vector reg, immed pre-index	LDR	3	2	Load/Store, Load
Load vector reg, unsigned immed	LDR	3	2	Load/Store, Load

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

3.15 FP scalar store instructions

Base register updates are done in parallel to the operation.

Table 3-14 AArch64 FP Store instructions

Instruction Group	AArch64 Instructions	Exec Latency	Execution Throughput	Utilized Pipeline
Store vector reg, unscaled immed, B/H/S/D-form	STUR	-	1	Load/Store
Store vector reg, unscaled immed, Q-form	STUR	-	1	Load/Store
Store vector reg, immed post-index, B/H/S/D-form	STR	-	1	Load/Store
Store vector reg, immed post-index, Q-form	STR	-	1	Load/Store

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

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

3.16 ASIMD Integer instructions

Table 3-15 AArch64 ASIMD Integer instructions

Instruction Group	AArch64 Instruction	Exec Latency	Execution Throughput	Utilized Pipeline
ASIMD absolute diff	SABD, UABD	3,3	2,1	VALU
ASIMD absolute diff accum	SABA, UABA	8,8	2/7,1/7	VALU
ASIMD absolute diff accum long	SABAL(2), UABAL(2)	8,8	2/7,1/7	VALU
ASIMD absolute diff long	SABDL(2), UABDL(2)	3,3	2,1	VALU
ASIMD arith, basic	ABS, ADD, NEG, SHADD, SHSUB, SUB, UHADD, UHSUB,	3,3	2,1	VALU
ASIMD arith, basic, long, saturate	SADDL(2), SADDW(2), SSUBL(2), SSUBW(2), UADDL(2), UADDW(2), USUBL(2), USUBW(2)	4,4	2,1	VALU
ASIMD arith, complex	ADDHN(2), RADDHN(2), RSUBHN(2), SQABS, SQADD, SQNEG, SQSUB, SRHADD, SUBHN(2), SUQADD, UQADD, UQSUB, URHADD, USQADD	4,4	2,1	VALU
ASIMD arith, pair-wise	ADDP, SADDLP, UADDLP	3,3	2,1	VALU
ASIMD arith, reduce, 4H/4S	ADDV, SADDLV, UADDLV	4,4	2,1	VALU
ASIMD arith, reduce	ADDV	3,3	2,1	VALU
ASIMD arith, reduce	SADDLV, UADDLV	4,4	2,1	VALU

Instruction Group	AArch64 Instruction	Exec Latency	Execution Throughput	Utilized Pipeline
ASIMD compare	CMEQ, CMGE, CMGT, CMHI, CMHS, CMLE, CMLT, CMTST	3,3	2,1	VALU
ASIMD compare test	CMTST	4,4	2,1	VALU
ASIMD dot product	SDOT, UDOT	4,4	2,1	VMAC
ASIMD dot product using signed and unsigned integers	SUDOT, USDOT	4,4	2,1	VMAC
ASIMD logical	AND, BIC, EOR, MOV, MVN, NOT, ORN, ORR	3,3	2,1	VALU
ASIMD matrix multiply-accumulate	SMMLA, UMMLA, USMMLA	4,4	2,1	VALU
ASIMD max/min, basic and pair-wise	SMAX, SMAXP, SMIN, SMINP, UMAX, UMAXP, UMIN, UMINP	3,3	2,1	VALU
ASIMD max/min, reduce, 4H/4S	SMAXV, SMINV, UMAXV, UMINV	4,4	2,1	VALU
ASIMD max/min, reduce, 8B/8H	SMAXV, SMINV, UMAXV, UMINV	4,4	2,1	VALU
ASIMD max/min, reduce, 16B	SMAXV, SMINV, UMAXV, UMINV	4,4	2,1	VALU
ASIMD multiply	MUL, SQDMULH, SQRDMULH	4,4	2,1	VMAC
ASIMD multiply accumulate	MLA, MLS	4,4	2,1	VMAC
ASIMD multiply accumulate high, D-form	SQRDMLAH, SQRDMLSH	4,4	2,1	VMAC
ASIMD multiply accumulate high, Q-form	SQRDMLAH, SQRDMLSH	4,4	2,1	VMAC
ASIMD multiply accumulate long	SMLAL(2), SMLSL(2), UMLAL(2), UMLSL(2)	4,4	2,1	VMAC
ASIMD multiply accumulate saturating long	SQDMLAL(2), SQDMLSL(2)	4,4	2,1	VMAC
ASIMD multiply/multiply long (8x8) polynomial, D-form	PMUL, PMULL(2)	4,4	2,1	VALU
ASIMD multiply/multiply long (8x8) polynomial, Q-form	PMUL, PMULL(2)	4,4	2,1	VALU
ASIMD multiply long	SMULL(2), UMULL(2), SQDMULL(2)	4,4	2,1	VMAC
ASIMD pairwise add and accumulate long	SADALP, UADALP	8,8	2/5,1/5	VALU

Instruction Group	AArch64 Instruction	Exec Latency	Execution Throughput	Utilized Pipeline
ASIMD shift accumulate	SSRA, SRSRA, USRA, URSRA	8,8	2/5,1/5	VALU
ASIMD shift by immed, basic	SHL, SHLL(2), SSHLL(2), SSHR, SXTL(2), USHLL(2), USHR, UXTL(2)	3,3	2,1	VALU
ASIMD shift by immed, basic	SHRN(2),	4,4	2,1	VALU
ASIMD shift by immed and insert, basic	SLI, SRI	3,3	2,1	VALU
ASIMD shift by immed, complex	RSHRN(2), QQRSHRN(2), QQRSHRUN(2), SQSHL{U}, SQSHRN(2), SQSHRUN(2), UQRSHRN(2), UQSHL, UQSHRN(2),	4,4	2,1	VALU
ASIMD shift by register, basic	SSHL, USHL, SRSHL, SRSHR, URSHL, URSR	3,3	2,1	VALU
ASIMD shift by register, complex	SQRSHL, SQSHL, , UQRSHL, UQSHL	4,4	2,1	VALUE

3.17 ASIMD FP data processing instructions

Table 3-16 AArch64 ASIMD Floating-point instructions

Instruction Group	AArch64 Instruction	Exec Latency	Execution Throughput	Utilized Pipeline
ASIMD FP absolute value/difference	FABS, FABD	4,4	2,1	VALU
ASIMD FP arith, normal	FABD, FADD, FSUB, FADDP	4,4	2,1	VALU
ASIMD FP compare	FACGE, FACGT, FCMEQ, FCMGE, FCMGT, FCMLE, FCMLT	3,3	2,1	VALU
ASIMD FP complex add	FCADD	4,4	2,1	VMAC
ASIMD FP complex multiply add	FCMLA	4,4	2,1	VMAC

Instruction Group	AArch64 Instruction	Exec Latency	Execution Throughput	Utilized Pipeline
ASIMD FP convert, long (F16 to F32)	FCVTL(2)	4,4	2,1	VALU
ASIMD FP convert, long (F32 to F64)	FCVTL(2)	4,4	2,1	VALU
ASIMD FP convert, narrow (F32 to F16)	FCVTN(2)	4,4	2,1	VALU
ASIMD FP convert, narrow (F64 to F32)	FCVTN(2), FCVTXN(2)	4,4	2,1	VALU
ASIMD FP convert, other, D-form F32 and Q-form F64	FCVTAS, FCVTAU, FCVTMS, FCVTMU, FCVTNS, FCVTNU, FCVTPS, FCVTPU, FCVTZS, FCVTZU, SCVTF, UCVTF	4,4	2,1	VALUE
ASIMD FP convert, other, D-form F16 and Q-form F32	FCVTAS, VCVTAU, FCVTMS, FCVTMU, FCVTNS, FCVTNU, FCVTPS, FCVTPU, FCVTZS, FCVTZU, SCVTF, UCVTF	4,4	2,1	VALU
ASIMD FP convert, other, Q-form F16	FCVTAS, VCVTAU, FCVTMS, FCVTMU, FCVTNS, FCVTNU, FCVTPS, FCVTPU, FCVTZS, FCVTZU, SCVTF, UCVTF	4,4	2,1	VALU
ASIMD FP divide, D-form, F16	FDIV	8,8	2/5,2/5	VMC
ASIMD FP divide, D-form, F32 ¹	FDIV	13,13	2/10,2/10	VMC
ASIMD FP divide, Q-form, F16 ¹	FDIV	8,8	1/5,1/5	VMC
ASIMD FP divide, Q-form, F32 ¹	FDIV	13,13	1/10,1/10	VMC
ASIMD FP divide, Q-form, F64	FDIV	22,22	1/19,1/19	VALU
ASIMD FP max/min, normal	FMAX, FMAXNM, FMIN, FMINNM	4,4	2,1	VALU
ASIMD FP max/min, pairwise	FMAXP, FMAXNMP, FMINP, FMINNMP	4,4	2,1	VALU
ASIMD FP max/min, reduce	FMAXV, FMAXNMV, FMINV, FMINNMV	4,4	2,1	VALU
ASIMD FP max/min, reduce, Q-form F16	FMAXV, FMAXNMV, FMINV, FMINNMV	4,4	2,1	VALU

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

Notes:

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

3.18 ASIMD BFloat16 (BF16) instructions

Table 3-17 AArch64 ASIMD BFloat16 (BF16) instructions

Instruction Group	AArch64 Instruction	Exec Latency	Execution Throughput	Utilized Pipeline
ASIMD convert, F32 to BF16	BFCVTN, BFCVTN2	4,4	2,1	VALU
ASIMD dot product	BFDOT	10,10	2,1	VMAC, VALU
ASIMD matrix multiply accumulate	BFMMLA	14, 15	1,1/2	VMAC, VALU
ASIMD multiply accumulate long	BFMLALB, BFMLALT	4,4	2,1	VMAC
Scalar convert, F32 to BF16	BFCVT	4,4	2,1	VALU

3.19 ASIMD miscellaneous instructions

Table 3-18 AArch64 ASIMD miscellaneous instructions

Instruction Group	AArch64 Instruction	Exec Latency	Execution Throughput	Utilized Pipeline
ASIMD bit reverse	RBIT	3,3	2,1	VALU
ASIMD bitwise insert	BIF, BIT, BSL	3,3	2,1	VALU
ASIMD count	CLS, CLZ, CNT	3,3	2,1	VALU
ASIMD duplicate, gen reg	DUP	3,3	2,1	VALU
ASIMD duplicate, element	DUP	3,3	2,1	VALU
ASIMD extract	EXT	3,3	2,1	VALU
ASIMD extract narrow	XTN	4,4	2,1	VALU
ASIMD extract narrow, saturating	SQXTN(2), SQXTUN(2), UQXTN(2)	4,4	2,1	VALU
ASIMD insert, element to element	INS	4,4	2,1	VALU
ASIMD move, FP immed	FMOV	3,3	2,1	VALU
ASIMD move, integer immed	MOVI, MVNI	3,3	2,1	VALU
ASIMD reciprocal estimate, D-form F32 and F64	FRECPE, FRECPX, FRSQRTE, URECPE, URSQRTE	4,4	2,1	VMAC
ASIMD reciprocal estimate, D-form F16 and Q-form F32	FRECPE, FRECPX, FRSQRTE, URECPE, URSQRTE	4,4	2,1	VMAC

Instruction Group	AArch64 Instruction	Exec Latency	Execution Throughput	Utilized Pipeline
ASIMD reciprocal estimate, Q-form F16	FRECPE, FRECPX, FRSQRTE, URECPE, URSQRTE	4,4	2,1	VMAC
ASIMD reciprocal step	FRECPS, FRSQRTS	4,4	2,1	VMAC
ASIMD reverse	REV16, REV32, REV64	3,3	2,1	VALU
ASIMD table lookup, 1 table regs	TBL	4,4	2,1	VALU
ASIMD table lookup, 2 table regs	TBL	8,8	1/2, 1/2	VALU
ASIMD table lookup, 3 table regs	TBL	12,12	1/5, 1/5	VALU
ASIMD table lookup, 4 table regs	TBL	16,16	1/9, 1/9	VALU
ASIMD table lookup extension, 1 table reg	TBX	8,8	1/2, 1/2	VALU
ASIMD table lookup extension, 2 table reg	TBX	12,12	1/5, 1/5	VALU
ASIMD table lookup extension, 3 table reg	TBX	16,16	1/9, 1/9	VALU
ASIMD table lookup extension, 4 table reg	TBX	20,20	1/13, 1/13	VALU
ASIMD transfer, element to gen reg	UMOV, SMOV	3	1	VALU
ASIMD transfer, gen reg to element	INS	3,3	2,1	VALU
ASIMD transpose	TRN1, TRN2	4,4	2,1	VALU
ASIMD unzip/zip	UZP1, UZP2, ZIP1, ZIP2	4,4	2,1	VALU

3.20 ASIMD load instructions

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

Base register updates are done in parallel to the operation.

Table 3-19 AArch64 load instructions

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

Instruction Group	AArch64 Instruction	Load Latency	Execution Throughput	Utilized Pipeline
ASIMD load, 1 element, multiple, 1 reg, Q-form	LD1	3	2	Load/Store, Load
ASIMD load, 1 element, multiple, 2 reg, D-form	LD1	3	1	Load/Store
ASIMD load, 1 element, multiple, 2 reg, Q-form	LD1	3	1	Load/Store
ASIMD load, 1 element, multiple, 3 reg, D-form	LD1	4	1/2	Load/Store
ASIMD load, 1 element, multiple, 3 reg, Q-form	LD1	4	1/2	Load/Store
ASIMD load, 1 element, multiple, 4 reg, D-form	LD1	4	½	Load/Store
ASIMD load, 1 element, multiple, 4 reg, Q-form	LD1	4	1/2	Load/Store
ASIMD load, 1 element, one lane, B/H/S	LD1	3	2	Load/Store
ASIMD load, 1 element, one lane, D	LD1	3	2	Load/Store, Load
ASIMD load, 1 element, all lanes, D-form, B/H/S	LD1R	3	2	Load/Store, Load
ASIMD load, 1 element, all lanes, D-form, D	LD1R	3	2	Load/Store, Load
ASIMD load, 1 element, all lanes, Q-form	LD1R	3	2	Load/Store, Load
ASIMD load, 2 element, multiple, D-form, B/H/S	LD2	4	½	Load/Store
ASIMD load, 2 element, multiple, Q-form, B/H/S	LD2	4	1/2	Load/Store
ASIMD load, 2 element, multiple, Q-form, D	LD2	4	1/2	Load/Store
ASIMD load, 2 element, one lane, B/H	LD2	4	1/2	Load/Store
ASIMD load, 2 element, one lane, S	LD2	4	1/2	Load/Store
ASIMD load, 2 element, one lane, D	LD2	4	1/2	Load/Store
ASIMD load, 2 element, all lanes, D-form, B/H/S	LD2R	3	1	Load/Store
ASIMD load, 2 element, all lanes, D-form, D	LD2R	3	1	Load/Store
ASIMD load, 2 element, all lanes, Q-form	LD2R	3	1	Load/Store
ASIMD load, 3 element, multiple, D-form, B/H/S	LD3	5	1/3	Load/Store

Instruction Group	AArch64 Instruction	Load Latency	Execution Throughput	Utilized Pipeline
ASIMD load, 3 element, multiple, Q-form, B/H/S	LD3	5	1/3	Load/Store
ASIMD load, 3 element, multiple, Q-form, D	LD3	5	1/3	Load/Store
ASIMD load, 3 element, one lane, B/H	LD3	5	1/3	Load/Store
ASIMD load, 3 element, one lane, S	LD3	5	1/3	Load/Store
ASIMD load, 3 element, one lane, D	LD3	5	1/3	Load/Store
ASIMD load, 3 element, all lanes, D-form, B/H/S	LD3R	4	1/2	Load/Store
ASIMD load, 3 element, all lanes, D-form, D	LD3R	4	1/2	Load/Store
ASIMD load, 3 element, all lanes, Q-form, B/H/S	LD3R	4	1/2	Load/Store
ASIMD load, 3 element, all lanes, Q-form, D	LD3R	4	1/2	Load/Store
ASIMD load, 4 element, multiple, D-form, B/H/S	LD4	5	1/3	Load/Store
ASIMD load, 4 element, multiple, Q-form, B/H/S	LD4	5	1/3	Load/Store
ASIMD load, 4 element, multiple, Q-form, D	LD4	5	1/4	Load/Store
ASIMD load, 4 element, one lane, B/H	LD4	6	1/4	Load/Store
ASIMD load, 4 element, one lane, S	LD4	6	1/4	Load/Store
ASIMD load, 4 element, one lane, D	LD4	6	1/4	
ASIMD load, 4 element, all lanes, D-form, B/H/S	LD4R	4	1/2	Load/Store
ASIMD load, 4 element, all lanes, D-form, D	LD4R	4	1/2	Load/Store
ASIMD load, 4 element, all lanes, Q-form, B/H/S	LD4R	4	1/2	Load/Store
ASIMD load, 4 element, all lanes, Q-form, D	LD4R	4	1/2	Load/Store

3.21 ASIMD store instructions

Base register updates are done in parallel to the operation.

Table 3-20 AArch64 ASIMD store instructions

Instruction Group	AArch64 Instruction	Exec Latency	Execution Throughput	Utilized Pipeline
ASIMD store, 1 element, multiple, 1 reg, D-form	ST1	-	1	Load/Store
ASIMD store, 1 element, multiple, 1 reg, Q-form	ST1	-	1	Load/Store
ASIMD store, 1 element, multiple, 2 reg, D-form	ST1	-	1	Load/Store
ASIMD store, 1 element, multiple, 2 reg, Q-form	ST1	-	1/2	Load/Store
ASIMD store, 1 element, multiple, 3 reg, D-form	ST1	-	1/3	Load/Store
ASIMD store, 1 element, multiple, 3 reg, Q-form	ST1	-	1/3	Load/Store
ASIMD store, 1 element, multiple, 4 reg, D-form	ST1	-	1/2	Load/Store
ASIMD store, 1 element, multiple, 4 reg, Q-form	ST1	-	1/4	Load/Store
ASIMD store, 1 element, one lane, B/H/S	ST1	-	1	Load/Store
ASIMD store, 1 element, one lane, D	ST1	-	1	Load/Store
ASIMD store, 2 element, multiple, D-form, B/H/S	ST2	-	1/6	Load/Store
ASIMD store, 2 element, multiple, Q-form, B/H/S	ST2	-	1/11	Load/Store
ASIMD store, 2 element, multiple, Q-form, D	ST2	-	1/11	Load/Store
ASIMD store, 2 element, one lane, B/H/S	ST2	-	1/6	Load/Store
ASIMD store, 2 element, one lane, D	ST2	-	1	Load/Store
ASIMD store, 3 element, multiple, D-form, B/H/S	ST3	-	1/19	Load/Store
ASIMD store, 3 element, multiple, Q-form, B/H/S	ST3	-	1/28	Load/Store
ASIMD store, 3 element, multiple, Q-form, D	ST3	-	1/3	Load/Store
ASIMD store, 3 element, one lane, B/H/S	ST3	-	1/11	Load/Store
ASIMD store, 3 element, one lane, D	ST3	-	1/2	Load/Store
ASIMD store, 4 element, multiple, D-form, B/H/S	ST4	-	1/33	Load/Store

Instruction Group	AArch64 Instruction	Exec Latency	Execution Throughput	Utilized Pipeline
ASIMD store, 4 element, multiple, Q-form, B/H/S	ST4	-	1/65	Load/Store
ASIMD store, 4 element, multiple, Q-form, D	ST4	-	1/4	Load/Store
ASIMD store, 4 element, one lane, B/H/S	ST4	-	1/10	Load/Store
ASIMD store, 4 element, one lane, D	ST4	-	½	Load/Store

3.22 Cryptography extensions

Table 3-21 AArch64 Cryptography instructions

Instruction Group	AArch64 Instruction	Exec Latency	Execution Throughput	Utilized Pipeline
Crypto AES ops	AESD, AESE, AESIMC, AESMC	3,3	2,1	Crypto
Crypto polynomial (64x64) multiply long	PMULL (2)	8	1	VMC
Crypto SHA1 hash acceleration op	SHA1H	3,3	2,1	VALU
Crypto SHA1 hash acceleration ops	SHA1C, SHA1M, SHA1P	8	1	VMC
Crypto SHA1 schedule acceleration ops	SHA1SU0, SHA1SU1	8	1	VMC
Crypto SHA256 hash acceleration ops	SHA256H, SHA256H2	8	1	VMC
Crypto SHA256 schedule acceleration ops	SHA256SU0, SHA256SU1	8	1	VMC
Crypto SHA512 hash acceleration ops	SHA512H, SHA512H2, SHA512SU0, SHA512SU1	8	1	VMC
Crypto SHA3 ops	BCAX, EOR3, XAR	3,3	2,1	VALU
Crypto SHA3 ops RAX1	RAX1	8	1	VMC
Crypto SM3 ops	SM3PARTW1, SM3PARTW2SM3S1, SM3TT1A, SM3TT1B, SM3TT2A, SM3TT2B	8	1	VMC
Crypto SM4 ops	SM4E, SM4EKEY	8	1	VMC

3.23 CRC

Table 3-22 AArch64 CRC instructions

Instruction Group	AArch64 Instruction	Exec Latency	Execution Throughput	Utilized Pipeline
CRC checksum ops	CRC32, CRC32C	2	1	MAC

3.24 SVE Predicate instructions

Table 3-23 SVE Predicate instructions

Instruction Group	SVE Instruction	Exec Latency	Execution Throughput	Utilized Pipeline
Loop control, based on predicate	BRKA, BRKB	6	1	PALU
Loop control, based on predicate and flag setting	BRKAS, BRKBS	6	1	PALU
Loop control, propagating	BRKN, BRKPA, BRKPB	6	1	PALU
Loop control, propagating and flag setting	BRKNS, BRKPAS, BRKPBS	6	1	PALU
Loop control, based on GPR	WHILEGE, WHILEGT, WHILEHI, WHILEHS, WHILELE, WHILELO, WHILELS, WHILELT, WHILERW, WHILEWR	6	1	PALU
Loop terminate ^[1]	CTERMEQ, CTERMNE	1	3	ALU

Instruction Group	SVE Instruction	Exec Latency	Execution Throughput	Utilized Pipeline
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	3	ALU
Predicate counting scalar, ALL, {1,2,4}	INC, DEC	1	3	ALU
Predicate counting scalar, active predicate	CNTP, DECP, INCP	6	1	PALU
Predicate counting scalar, active predicate, saturating	SQDECP, SQINCP, UQDECP, UQINCP	8,8	2,1	VALU
Predicate logical	AND, BIC, EOR, MOV, NAND, NOR, NOT, ORN, ORR	6	1	PALU
Predicate logical, flag setting	ANDS, BICS, EORS, MOV, NANDS, NORS, NOTS, ORNS, ORRS	6	1	PALU
Predicate reverse	REV	6	1	PALU
Predicate select	SEL	6	1	PALU
Predicate set	PFALSE, PTRUE	6	1	PALU
Predicate set/initialize, set flags	PTRUES	6	1	PALU
Predicate find first/next	PFIRST, PNEXT	6	1	PALU
Predicate test	PTEST	6	1	PALU
Predicate transpose	TRN1, TRN2	6	1	PALU
Predicate unpack and widen	PUNPKHI, PUNPKLO	6	1	PALU
Predicate zip/unzip	ZIP1, ZIP2, UZP1, UZP2	6	1	PALU

Notes:

1. Instructions with dependencies may be co-issued

3.25 SVE Integer instructions

Table 3-24 SVE integer instructions

Instruction Group	SVE Instruction	Exec Latency	Execution Throughput	Utilized Pipeline
Arithmetic, absolute diff	SABD, UABD	3,3	2,1	VALU
Arithmetic, absolute diff accum	SABA, UABA	8,8	2/5,1/5	VALU
Arithmetic, absolute diff accum long	SABALB, SABALT, UABALB, UABALT	8,8	2/5,1/5	VALU
Arithmetic, absolute diff long	SABDLB, SABDLT, UABDLB, UABDLT	3,3	2,1	VALU
Arithmetic, basic	ABS, ADD, ADR, CNOT, NEG, SHADD, SHSUB, SHSUBR, SRHADD, SUB, SUBHNB, SUBHNT, SUBR, UADDWB, UADDWT, UHADD, UHSUB, UHSUBR, , URHADD USUBWB, USUBWT	3,3	2,1	VALU
Arithmetic, basic	SADDLB, SADDLBT, SADDLT, SADDWB, SADDWT, SSUBLB, SSUBLBT, SSUBLT, SSUBLTB, SSUBWB, SSUBWT, UADDLB, UADDLT, USUBLB, USUBLT,	4,4	2,1	VALU
Arithmetic, complex	ADDHNB, ADDHNT, RADDHNB, RADDHNT, RSUBHNB, RSUBHNT, SQABS, SQADD, SQNEG, SQSUB, SQSUBR, SUQADD, UQADD, UQSUB, UQSUBR, USQADD,	4,4	2,1	VALU
Arithmetic, large integer	ADCLB, ADCLT, SBCLB, SBCLT	4,4	2,1	VALU
Arithmetic, pairwise add	ADDP	3,3	2,1	VALU
Arithmetic, pairwise add and accum long	SADALP, UADALP	8,8	2/5,1/5	VALU

Instruction Group	SVE Instruction	Exec Latency	Execution Throughput	Utilized Pipeline
Arithmetic, shift	ASR, ASRR, LSL, LSLR, LSR, LSRR	3,3	2,1	VALU
Arithmetic, shift and accumulate	USRA	4,4	2,1	VALU
Arithmetic, shift and accumulate complex	SRSRA, SSRA, URSRA,	7,7	2,1	VALU
Arithmetic, shift by immediate	SHRNB, SHRNT, SSHLLB, SSHLLT, USHLLB, USHLLT	3,3	2,1	VALU
Arithmetic, shift by immediate and insert	SLI, SRI	3,3	2,1	VALU
Arithmetic, shift complex	RSHRNB, RSHRNT, SQRSHL, SQRSHLR, SQRSHRNB, SQRSHRNT, SQRSHRUNB, SQRSHRUNT, SQSHL, SQSHLR, SQSHLU, SQSHRNB, SQSHRNT, SQSHRUNB, SQSHRUNT, UQRSHL, UQRSHLR, UQRSHRNB, UQRSHRNT, UQSHL, UQSHLR, UQSHRNB, UQSHRNT	4,4	2,1	VALU
Arithmetic, shift right for divide	ASRD	4,4	2,1	VALU
Arithmetic, shift rounding	SRSHL, SRSHLR, SRSHR, URSHL, URSHLR, URSHR	4,4	2,1	VALU
Bit manipulation (B)	BDEP, BEXT, BGRP	14	1/14	VMC
Bit manipulation (H)	BDEP, BEXT, BGRP	22	1/22	VMC
Bit manipulation (S)	BDEP, BEXT, BGRP	38	1/38	VMC
Bit manipulation (D)	BDEP, BEXT, BGRP	70	1/70	VMC
Bitwise select	BSL, BSL1N, BSL2N, NBSL	3,3	2,1	VALU
Count/reverse bits	CLS, CLZ, RBIT	4,4	2,1	VALU
Count (B,H)	CNT	4,4	2,1	VALU
Count (S)	CNT	8,8	2/5,1/5	VALU
Count (D)	CNT	12,12	2/9,1/9	VALU
Broadcast logical bitmask immediate to vector	DUPM, MOV	4,4	2,1	VALU

Instruction Group	SVE Instruction	Exec Latency	Execution Throughput	Utilized Pipeline
Compare and set flags	CMPEQ, CMPGE, CMPGT, CMPHI, CMPHS, CMPLE, CMPLO, CMPLS, CMPLT, CMPNE	3,3	2,1	VALU
Complex add	CADD	3,3	2,1	VALU
Complex add saturating	SQCADD	4,4	2,1	VALU
Complex dot product 8-bit element	CDOT	4,4	2,1	VMAC
Complex dot product 16-bit element	CDOT	4,4	2,1	VMAC
Complex multiply-add B, H, S element size	CMLA	4,4	2,1	VMAC
Complex multiply-add D element size	CMLA	4,4	2,1	VMAC
Conditional extract operations, general purpose register	CLASTA, CLASTB	8,8	1,1	VALU
Conditional extract operations, SIMD&FP scalar and vector forms	CLASTA, CLASTB, COMPACT, SPLICE	4,4	2,1	VALU
Convert to floating point, 64b to float or convert to double	SCVTF, UCVTF	4,4	2,1	VALU
Convert to floating point, 32b to single or half	SCVTF, UCVTF	4,4	2,1	VALU
Convert to floating point, 16b to half	SCVTF, UCVTF	4,4	2,1	VALU
Copy, scalar	CPY	3,3	2,1	VALU
Copy, scalar SIMD&FP or imm	CPY	3,3	2,1	VALU
Divides, 32 bit	SDIV, SDIVR, UDIV, UDIVR	15	1/12	VMC
Divides, 64 bit	SDIV, SDIVR, UDIV, UDIVR	26	1/23	VMC
Dot product, 8 bit	SDOT, UDOT	4,4	2,1	VMAC
Dot product, 8 bit, using signed and unsigned integers	SUDOT, USDOT	4,4	2,1	VMAC
Dot product, 16 bit	SDOT, UDOT	4,4	2,1	VMAC
Duplicate, immediate and indexed form	DUP, MOV	3,3	2,1	VALU
Duplicate, indexed > elem	DUP	4,4	2,1	VALU
Duplicate, scalar form	DUP, MOV	3,3	2,1	VALU

Instruction Group	SVE Instruction	Exec Latency	Execution Throughput	Utilized Pipeline
Extend, sign or zero	SXTB, SXTH, SXTW, UXTB, UXTH, UXTW	3,3	2,1	VALU
Extract	EXT	3,3	2,1	VALU
Extract narrow saturating	SQXTNB, SQXTNT, SQXTUNB, SQXTUNT, UQXTNB, UQXTNT	4,4	2,1	VALU
Extract/insert operation, SIMD and FP scalar form	LASTA, LASTB, INSR	4,4	2,1	VALU
Extract/insert operation, scalar	LASTA, LASTB, INSR	8,8	1/5,1/5	VALU0
Histogram operations	HISTCNT, HISTSEG	8	1/5	VALU0
Horizontal operations, B, H, S form, immediate operands only	INDEX	4,4	2,1	VMAC
Horizontal operations, B, H, S form, scalar, immediate operands) / scalar operands only / immediate, scalar operands	INDEX	4,4	2,1	VMAC
Horizontal operations, D form, immediate operands only	INDEX	4,4	2,1	VMAC
Horizontal operations, D form, scalar, immediate operands) / scalar operands only / immediate, scalar operands	INDEX	4,4	2,1	VMAC
Logical	AND, BIC, EON, EOR, MOV, NOT, ORN, ORR	3,3	2,1	VALU
Logical	EORBT, EORTB,	4,4	2,1	VALU
Max/min, basic and pairwise	SMAX, SMAXP, SMIN, SMINP, UMAX, UMAXP, UMIN, UMINP	3,3	2,1	VALU
Matching operations	MATCH, NMATCH	7,7	1/4,1/4	VALU
Matrix multiply-accumulate	SMMLA, UMMLA, USMMLA	4,4	2,1	VMAC
Move prefix	MOVPRFX	3,3	2,1	VALU
Multiply, B, H, S element size	MUL, SMULH, UMULH	4,4	2,1	VMAC
Multiply, D element size	MUL, SMULH, UMULH	4,4	2,1	VMAC

Instruction Group	SVE Instruction	Exec Latency	Execution Throughput	Utilized Pipeline
Multiply long	SMULLB, SMULLT, UMULLB, UMULLT	4,4	2,1	VMAC
Multiply accumulate, B, H, S element size	MLA, MLS	4,4	2,1	VMAC
Multiply accumulate, D element size	MLA, MLS, MAD, MSB,	4,4	2,1	VMAC
Multiply accumulate long	SMLALB, SMLALT, SMLSLB, SMLSLT, UMLALB, UMLALT, UMLSLB, UMLSLT	4,4	2,1	VMAC
Multiply accumulate saturating doubling long regular	SQDMLALB, SQDMLALT, SQDMLALBT, SQDMLSLB, SQDMLSLT, SQDMLSLBT	4,4	2,1	VMAC
Multiply saturating doubling high, B, H, S element size	SQDMULH	4,4	2,1	VMAC
Multiply saturating doubling high, D element size	SQDMULH	4,4	2,1	VMAC
Multiply saturating doubling long	SQDMULLB, SQDMULLT	4,4	2,1	VMAC
Multiply saturating rounding doubling regular/complex accumulate, B, H, S element size	SQRDMLAH, SQRDMLSH, SQRDCMLAH	4,4	2,1	VMAC
Multiply saturating rounding doubling regular/complex accumulate, D element size	SQRDMLAH, SQRDMLSH, SQRDCMLAH	4,4	2,1	VMAC
Multiply saturating rounding doubling regular/complex, B, H, S element size	SQRDMULH	4,4	2,1	VMAC
Multiply saturating rounding doubling regular/complex, D element size	SQRDMULH	4,4	2,1	VMAC
Multiply/multiply long, (8, 16, 32) polynomial	PMUL, PMULLB, PMULLT	4,4	2,1	VALU
Multiply/multiply long, (64) polynomial	PMULLB, PMULLT	6	1	VMC
Predicate counting vector	CNT, DECB, DECH, DECW, DECD, INCB, INCH, INCW, INCD	3,3	2,1	VALU

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

3.26 SVE FP data processing instructions

Table 3-25 SVE Floating-point instructions

Instruction Group	SVE Instruction	Exec Latency	Execution Throughput	Utilized Pipeline
Floating point absolute value/difference	FABD, FABS	4,4	2,1	VALU
Floating point arithmetic	FADD, FADDP, FNEG, FSUB, FSUBR	4,4	2,1	VALU
Floating point associative add, F16	FADDA	32,32	2/29,1/29	VALU
Floating point associative add, F32	FADDA	16,16	2/13,1/13	VALU
Floating point associative add, F64	FADDA	8,8	2/5,1/5	VALU
Floating point compare	FACGE, FACGT, FACLE, FACLT, FCMEQ, FCMGE, FCMGT, FCMLE, FCMLT, FCMNE, FCMUO	4,4	2,1	VALU
Floating point complex add	FCADD	4,4	2,1	VALU
Floating point complex multiply add	FCMLA	4,4	2,1	VMAC
Floating point convert, long or narrow (F16 to F32 or F32 to F16)	FCVT, FCVTLT, FCVTNT	4,4	2,1	VALU
Floating point convert, long or narrow (F16 to F64, F32 to F64, F64 to F32 or F64 to F16)	FCVT, FCVTLT, FCVTNT	4,4	2,1	VALU
Floating point convert, round to odd	FCVTX, FCVTXNT	4,4	2,1	VALU
Floating point base2 log, F16	FLOGB	4,4	2,1	VMAC
Floating point base2 log, F32	FLOGB	4,4	2,1	VMAC
Floating point base2 log, F64	FLOGB	4,4	2,1	VMAC
Floating point convert to integer, F16	FCVTZS, FCVTZU	4,4	2,1	VALU
Floating point convert to integer, F32	FCVTZS, FCVTZU	4,4	2,1	VALU
Floating point convert to integer, F64	FCVTZS, FCVTZU	4,4	2,1	VALU
Floating point copy	FCPY, FDUP, FMOV	3,3	2,1	VALU
Floating point divide, F16 ¹	FDIV, FDIVR	8	1/5	VMC

Instruction Group	SVE Instruction	Exec Latency	Execution Throughput	Utilized Pipeline
Floating point divide, F32 ¹	FDIV, FDIVR	13	1/10	VMC
Floating point divide, F64 ¹	FDIV, FDIVR	22	1/19	VMC
Floating point min/max pairwise	FMAXP, FMAXNMP, FMINP, FMINNMP	4,4	2,1	VALU
Floating point min/max	FMAX, FMIN, FMAXNM, FMINNM	4,4	2,1	VALU
Floating point multiply	FSCALE, FMUL, FMULX	4,4	2,1	VMAC
Floating point multiply accumulate	FMLA, FMLS, FMAD, FMSB, FNMAD, FNMLA, FNMLS, FNMSB	4,4	2,1	VMAC
Floating point multiply add/sub accumulate long	FMLALB, FMLALT, FMLSLB, FMLSLT	4,4	2,1	VMAC
Floating point reciprocal estimate, F16	FRECPE, FRECPX, FRSQRTE	4,4	2,1	VMAC
Floating point reciprocal estimate, F32	FRECPE, FRECPX, FRSQRTE	4,4	2,1	VMAC
Floating point reciprocal estimate, F64	FRECPE, FRECPX, FRSQRTE	4,4	2,1	VMAC
Floating point reciprocal step	FRECPS, FRSQRSTS	4,4	2,1	VMAC
Floating point reduction, F16	FMAXNMV, FMAXV, FMINNMV, FMINV	4	1	VALU0
Floating point reduction, F16	FADDV	12	1/13	VALU0
Floating point reduction, F32	FADDV	8	1/5	VALU0
Floating point reduction, F64	FADDV	4	1	VALU0
Floating point round to integral, F16	FRINTA, FRINTM, FRINTN, FRINTP, FRINTX, FRINTZ	4,4	2,1	VALU
Floating point round to integral, F32	FRINTA, FRINTM, FRINTN, FRINTP, FRINTX, FRINTZ	4,4	2,1	VALU
Floating point round to integral, F64	FRINTA, FRINTM, FRINTN, FRINTP, FRINTX, FRINTZ	4,4	2,1	VALU
Floating point square root, F16	FSQRT	8	1/5	VMC
Floating point square root, F32	FSQRT	12	1/9	VMC

Instruction Group	SVE Instruction	Exec Latency	Execution Throughput	Utilized Pipeline
Floating point square root F64	FSQRT	22	1/19	VMC
Floating point trigonometric exponentiation	FEXPA	4,4	2,1	VMAC
Floating point trigonometric multiply add	FTMAD	4,4	2,1	VMAC
Floating point trigonometric starting value	FTSMUL	4,4	2,1	VMAC
Floating point trigonometric select coefficient	FTSSEL	4,4	2,1	VALU

Notes:

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

3.27 SVE BFloat16 (BF16) instructions

Table 3-26 SVE Bfloat16 (BF16) instructions

Instruction Group	SVE Instruction	Exec Latency	Execution Throughput	Utilized Pipeline
Convert, F32 to BF16	BFCVT, BFCVTNT	4,4	2, 1	VALU
Dot product	BFDOT	10,10	2, 1	VMAC, VALU
Matrix multiply accumulate	BFMMLA	14, 15	1, 1/2	VMAC, VALU
Multiply accumulate long	BFMLALB, BFMLALT	4, 4	2, 1	VMAC

3.28 SVE Load instructions

The latencies shown in Table 3-27 assume the memory access hits in the Level 1 Data Cache.

Base register updates are done in parallel to the operation.

Table 3-27 SVE Load instructions

Instruction Group	SVE Instruction	Load Latency	Execution Throughput	Utilized Pipeline
Load vector	LDR	3	2	Load/Store, Load
Load predicate	LDR	3	1	Load/Store
Contiguous load, scalar + imm	LD1B, LD1D, LD1H, LD1W, LD1SB, LD1SH, LD1SW,	3	2	Load/Store, Load

Instruction Group	SVE Instruction	Load Latency	Execution Throughput	Utilized Pipeline
Contiguous load, scalar + scalar	LD1B, LD1D, LD1H, LD1W, LD1SB, LD1SH LD1SW	3	2	Load/Store, Load
Contiguous load broadcast, scalar + imm	LD1RB, LD1RH, LD1RD, LD1RW, LD1RSB, LD1RSH, LD1RSW, LD1RQB, LD1RQD, LD1RQH,	3	2	Load/Store, Load
Contiguous load broadcast, scalar + scalar	LD1RQB, LD1RQD, LD1RQH, LD1RQW	3	2	Load/Store, Load
Non temporal load, scalar + imm	LDNT1B, LDNT1D, LDNT1H, LDNT1W	3	2	Load/Store, Load
Non temporal load, scalar + scalar	LDNT1B, LDNT1D, LDNT1H LDNT1W	3	2	Load/Store, Load
Non temporal gather load, vector + scalar 32-bit element size	LDNT1B, LDNT1H, LDNT1W, LDNT1SB, LDNT1SH	9	1/9	Load/Store
Non temporal gather load, vector + scalar 64-bit element size	LDNT1B, LDNT1D, LDNT1H, LDNT1W, LDNT1SB, LDNT1SH, LDNT1SW	7	1/7	Load/Store
Contiguous first faulting load, scalar + scalar	LDFF1B, LDFF1D, LDFF1H, LDFF1W, LDFF1SB, LDFF1SD, LDFF1SH LDFF1SW	3	2	Load/Store, Load
Contiguous non faulting load, scalar + imm	LDNF1B, LDNF1D, LDNF1H, LDNF1W, LDNF1SB, LDNF1SH, LDNF1SW	3	2	Load/Store, Load
Contiguous Load two structures to two vectors, scalar + imm	LD2B, LD2D, LD2H, LD2W	3	1	Load/Store
Contiguous Load two structures to two vectors, scalar + scalar	LD2B, LD2D, LD2H, LD2W	3	1	Load/Store
Contiguous Load three structures to three vectors, scalar + imm	LD3B, LD3D, LD3H, LD3W	5	1/3	Load/Store
Contiguous Load three structures to three vectors, scalar + scalar	LD3B, LD3D, LD3H, LD3W	5	1/3	Load/Store

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

3.29 SVE Store instructions

Base register updates are done in parallel to the operation.

Table 3-28 SVE Store instructions

Instruction Group	SVE Instructions	Exec Latency	Execution Throughput	Utilized Pipeline
Store from predicate reg	STR	-	1	Load/Store
Store from vector reg	STR	-	1	Load/Store
Contiguous store, scalar + imm	ST1B, ST1H, ST1D, ST1W	-	1	Load/Store

Instruction Group	SVE Instructions	Exec Latency	Execution Throughput	Utilized Pipeline
Contiguous store, scalar + scalar	ST1H	-	1	Load/Store
Contiguous store, scalar + scalar	ST1B, ST1D, ST1W	-	1	Load/Store
Contiguous store two structures from two vectors, scalar + imm	ST2B, ST2H, ST2D, ST2W	-	1/2	Load/Store
Contiguous store two structures from two vectors, scalar + scalar	ST2H	-	1/2	Load/Store
Contiguous store two structures from two vectors, scalar + scalar	ST2B, ST2D, ST2W	-	1/2	Load/Store
Contiguous store three structures from three vectors, scalar + imm	ST3B, ST3D, ST3H, ST3W	-	1/28	Load/Store
Contiguous store three structures from three vectors, scalar + scalar	ST3H	-	1/28	Load/Store
Contiguous store three structures from three vectors, scalar + scalar	ST3B, ST3D, ST3W	-	1/28	Load/Store
Contiguous store four structures from four vectors, scalar + imm	ST4B, ST4D, ST4H, ST4W	-	1/65	Load/Store
Contiguous store four structures from four vectors, scalar + scalar	ST4H	-	1/65	Load/Store
Contiguous store four structures from four vectors, scalar + scalar	ST4B, ST4D, ST4W	-	1/65	Load/Store
Non temporal store, scalar + imm	STNT1B, STNT1D, STNT1H, STNT1W	-	1	Load/Store
Non temporal store, scalar + scalar	STNT1H	-	1	Load/Store
Non temporal store, scalar + scalar	STNT1B, STNT1D, STNT1W	-	1	Load/Store
Scatter non temporal store, vector + scalar 32-bit element size	STNT1B, STNT1H, STNT1W	-	1/9	Load/Store
Scatter non temporal store, vector + scalar 64-bit element size	STNT1B, STNT1D, STNT1H, STNT1W	-	1/7	Load/Store
Scatter store vector + imm 32-bit element size	ST1B, ST1H, ST1W	-	1/9	Load/Store
Scatter store vector + imm 64-bit element size	ST1B, ST1D, ST1H, ST1W	-	1/7	Load/Store

Instruction Group	SVE Instructions	Exec Latency	Execution Throughput	Utilized Pipeline
Scatter store, 32-bit scaled offset	ST1H, ST1W	-	1/9	Load/Store
Scatter store, 32-bit unpacked unscaled offset	ST1B, ST1D, ST1H, ST1W	-	1/9	Load/Store
Scatter store, 32-bit unpacked scaled offset	ST1D, ST1H, ST1W	-	1/9	Load/Store
Scatter store, 32-bit unscaled offset	ST1B, ST1H, ST1W	-	1/9	Load/Store
Scatter store, 64-bit scaled offset	ST1D, ST1H, ST1W	-	1/7	Load/Store
Scatter store, 64-bit unscaled offset	ST1B, ST1D, ST1H, ST1W	-	1/7	Load/Store

3.30 SVE Miscellaneous instructions

Table 3-29 SVE Miscellaneous instructions

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

3.31 SVE Cryptography instructions

Table 3-30 SVE cryptography instructions

Instruction Group	SVE Instructions	Exec Latency	Execution Throughput	Utilized Pipeline
Crypto AES ops	AESD, AESE, AESIMC, AESMC	3,3	2,1	Crypto
Crypto SHA3 ops	BCAX, EOR3, XAR	3,3	2,1	VALU
Crypto SHA3 ops RAX1	RAX1	8	1	VMC

Crypto SM4 ops	SM4E, SM4EKEY	8	1	VMC
----------------	---------------	---	---	-----

4 Special considerations

4.1 Issue constraints

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

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

An instruction will occupy two entries when it has either:

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

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

AES instructions will stall until there is at least one other instruction available to be issued (see 4.2 *Instruction fusion*).

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

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

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

The following are multicycle:

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

4.2 Instruction fusion

The Cortex-A510 can accelerate key instruction pairs in an operation called fusion.

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

- AES + AESMC (see 4.9 AES encryption / decryption)
- AUT* + BR
- AUT* + LDR

These instruction pairs must be adjacent to each other in program code.

4.3 Branch instruction alignment

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



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

4.4 Load / Store Alignment

The Armv8-A architecture allows many types of load and store accesses to be arbitrarily aligned. The Cortex-A510 core handles most unaligned accesses without performance penalties. However, there are cases which could reduce bandwidth or incur additional latency, as described below.

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

4.5 Memory Tagging Extensions

Enabling precise tag checking can prevent the Cortex-A510 core from entering write-streaming mode. This can reduce performance and increase power when writes miss in the L1 or L2 caches.

4.6 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
- Stores should be aligned on a 16-byte boundary wherever possible
- Loads should not cross a 32-byte boundary as they incur a penalty



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

Figure 2 shows a code snippet from the inner loop of memory copy routine that copies at least 128 bytes. The loop copies 64 bytes per iteration and prefetches one iteration ahead.

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

Figure 2 Code Snippet from memcpy routine - large copy inner loop

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

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

Figure 3 Code Snippet from memcpy routine - small copy inner loop

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

Unroll the loop to include multiple store operations per iteration, minimizing the overheads of looping. Figure 4 shows code from the memset routine to set 17 to 96 bytes.

```
L(set_medium):
    str q0, [dstin]
    tbnz count, 6, L(set96)
```

```

str    q0, [dstend, -16]
tbz    count, 5, 1f
str    q0, [dstin, 16]
str    q0, [dstend, -32]
1: ret

```

Figure 4 Code snippet from memset routine

To achieve maximum performance on memset to zero, it is recommended that one use DC ZVA instead of STP. Figure 5 shows code from the memset routine to illustrate the usage of DC ZVA..

```

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

```

Figure 5 Code snippet from memset to zero routine

4.7 Cache maintenance operations

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

4.8 Shared VPU

Cortex-A510 shares a VPU between all Cortex-A510 cores in a complex. The VPU is used to execute ASIMD, FP, Neon, and SVE instructions. Instructions being executed on VPU pipelines by one core may reduce performance of the instructions executed on the VPU by the other core.

4.9 AES encryption / decryption

Cortex-A510 implements instruction fusion for AES instructions (see section 4.2). It is recommended instructions pairs be interleaved in groups of three or more for the following: AESE, AESMC, AESD, AESIMC.

```

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

```

Figure 6 Code snippet for AES instruction fusion