



Arm® Cortex®-A520AE Core

Revision: r0p1

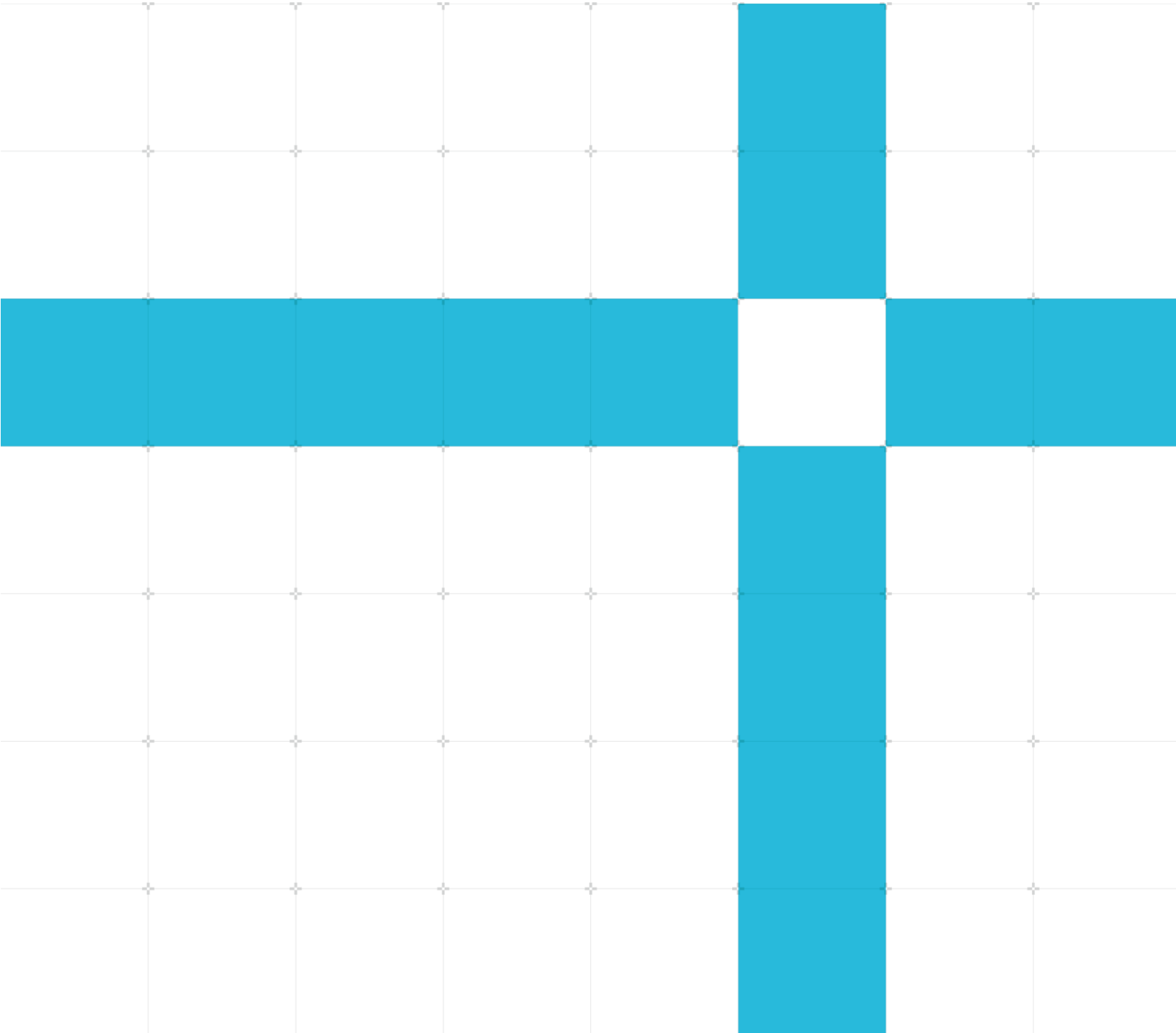
Software Optimization Guide

Non-Confidential

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

Issue 3.0

PJDOC-1505342170-694361



Arm® Cortex-A520AE Core

Software Optimization Guide

Copyright © 2022-2024 Arm Limited (or its affiliates). All rights reserved.

Release Information

Document history

| Issue | Date | Confidentiality | Change |
|-------|-------------|------------------|-------------------------|
| 1.0 | 16 Nov 2023 | Confidential | First release for r0p0 |
| 2.0 | 13 Mar 2024 | Confidential | Second release for r0p0 |
| 3.0 | 20 Dec 2024 | Non-Confidential | First release for r0p1 |

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, 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 <http://www.arm.com/company/policies/trademarks>. All rights reserved. Other brands and names mentioned in this document may be the trademarks of their respective owners.

Arm Limited. Company 02557590 registered in England.

110 Fulbourn Road, Cambridge, England CB1 9NJ.

PRE-1122-V1.0

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.

Feedback

Arm welcomes feedback on this product and its documentation. To provide feedback on Arm® Cortex-A520AE Core, create a ticket on <https://support.developer.arm.com>.

To provide feedback on the document, fill the following survey:
<https://developer.arm.com/documentation-feedback-survey>.

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 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 | 7 |
| 1.1 | Product revision status | 7 |
| 1.2 | Intended audience..... | 7 |
| 1.3 | Conventions..... | 7 |
| 1.3.1 | Glossary | 7 |
| 1.3.2 | Typographical conventions..... | 8 |
| 1.4 | Additional reading..... | 9 |
| 2 | Overview..... | 10 |
| 2.1 | Pipeline overview | 11 |
| 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 | 16 |
| 3.5 | Pointer authentication instructions..... | 17 |
| 3.6 | Miscellaneous data-processing instructions..... | 18 |
| 3.7 | Load instructions | 19 |
| 3.8 | Store instructions | 21 |
| 3.9 | Tag data processing..... | 23 |
| 3.10 | Tag load instructions..... | 23 |
| 3.11 | Tag store instructions | 23 |
| 3.12 | FP scalar data processing instructions | 25 |
| 3.13 | FP scalar miscellaneous instructions..... | 26 |
| 3.14 | FP scalar load instructions..... | 27 |
| 3.15 | FP scalar store instructions..... | 28 |
| 3.16 | ASIMD Integer instructions..... | 30 |
| 3.17 | ASIMD FP data processing instructions | 35 |
| 3.18 | ASIMD BFloat16 (BF16) instructions | 38 |
| 3.19 | ASIMD miscellaneous instructions..... | 39 |
| 3.20 | ASIMD load instructions | 41 |
| 3.21 | ASIMD store instructions | 44 |

| | | |
|----------|---|-----------|
| 3.22 | Cryptography extensions..... | 46 |
| 3.23 | CRC..... | 47 |
| 3.24 | SVE Predicate instructions..... | 48 |
| 3.25 | SVE Integer instructions..... | 51 |
| 3.26 | SVE FP data processing instructions | 60 |
| 3.27 | SVE BFloat16 (BF16) instructions | 64 |
| 3.28 | SVE Load instructions | 64 |
| 3.29 | SVE Store instructions | 69 |
| 3.30 | SVE Miscellaneous instructions | 71 |
| 3.31 | SVE Cryptography instructions..... | 72 |
| 4 | Special considerations | 73 |
| 4.1 | Issue constraints | 73 |
| 4.2 | Instruction fusion..... | 74 |
| 4.3 | Branch instruction alignment | 74 |
| 4.4 | Load / Store Alignment..... | 74 |
| 4.5 | A64 low latency pointer forwarding..... | 74 |
| 4.6 | AUT* RET forwarding..... | 75 |
| 4.7 | SIMD MAC forwarding..... | 75 |
| 4.8 | Memory Tagging Extensions..... | 75 |
| 4.9 | Memory routines | 75 |
| 4.10 | Cache maintenance operations | 77 |
| 4.11 | Cache access latencies..... | 78 |
| 4.12 | Shared VPU | 78 |
| 4.13 | AES encryption / decryption | 78 |

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: <https://developer.arm.com/glossary>.






Terms and abbreviations


This document uses the following terms and abbreviations.

| Convention | Use |
|------------|---|
| 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 |

| Convention | Use |
|------------|------------------------|
| VPU | Vector Processing Unit |

1.3.2 Typographical conventions

| Convention | Use |
|--|---|
| <i>italic</i> | Citations. |
| bold | Interface elements, such as menu names. Signal names. Terms in descriptive lists, where appropriate. |
| monospace | Text that you can enter at the keyboard, such as commands, file and program names, and source code. |
| monospace bold | Language keywords when used outside example code. |
| monospace <u>underline</u> | A permitted abbreviation for a command or option. You can enter the underlined text instead of the full command or option name. |
| <and> | Encloses replaceable terms for assembler syntax where they appear in code or code fragments. For example: <code>MRC p15, 0, <Rd>, <CRn>, <CRm>, <Opcode_2></code> |
| SMALL CAPITALS | Terms that have specific technical meanings as defined in the Arm® Glossary. For example, IMPLEMENTATION DEFINED, IMPLEMENTATION SPECIFIC, UNKNOWN, and UNPREDICTABLE. |
|  Caution | Recommendations. Not following these recommendations might lead to system failure or damage. |
|  Warning | Requirements for the system. Not following these requirements might result in system failure or damage. |
|  Danger | Requirements for the system. Not following these requirements will result in system failure or damage. |
|  Note | An important piece of information that needs your attention. |
|  Tip | A useful tip that might make it easier, better, or faster to perform a task. |

| Convention | Use |
|---|--|
|  Remember | A reminder of something important that relates to the information you are reading. |

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 |
|--|-------------|-------------------|
| <i>Arm® Architecture Reference Manual for A-profile architecture profile</i> | DDI 0487 | N |
| <i>Arm® Cortex-A520AE Core Technical Reference Manual</i> | 107726 | N |
| <i>Arm® Cortex-A520AE Core Configuration and Integration Manual</i> | 107727 | Y |



Note

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>.

2 Overview

The Arm® Cortex-A520AE Core (MP187) is a high-efficiency AE Cortex-A CPU based on the Arm® v9.2-A architecture for safety critical applications. Cortex-A520AE is targeting L1 & L2 ADAS, and Gateway / V2X applications. It is considered a Cortex-A class little core that supports ASIL B and ASIL D systems.

- Implementation of the Arm®v9.2-A A64 instruction set
- AArch64 Execution state at all Exception levels, EL0 to EL3
- Separate L1 data and instruction side memory systems with a Memory Management Unit (MMU)
- In-order pipeline with direct and indirect branch prediction
- Generic Interrupt Controller (GIC) CPU interface to connect to an external interrupt distributor
- Generic Timer interface that supports a 64-bit count input from an external system counter
- Implementation of the Reliability, Availability, and Serviceability (RAS) Extension
- 128-bit Scalable Vector Extension (SVE) and SVE2 SIMD instruction set, offering Advanced SIMD (ASIMD) and floating-point (FP) architecture support
- Support for the optional Cryptographic Extension, which is licensed separately
- Activity Monitoring Unit (AMU)
- Dual/Single Core configuration option: Cortex-A520AE 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-A520AE cores in a complex.
- Configurable vector datapath size: The size of the vector datapaths can be 2x64 or 2x128-bit. The selected option applies to all cores in the complex. Figure 1 highlights the VPU pipelines that are only instantiated for a 2x128-bit configuration.

This document describes the elements of Cortex-A520AE Core micro-architecture that influence the software performance so that software and compilers can be optimized accordingly.

2.1 Pipeline overview

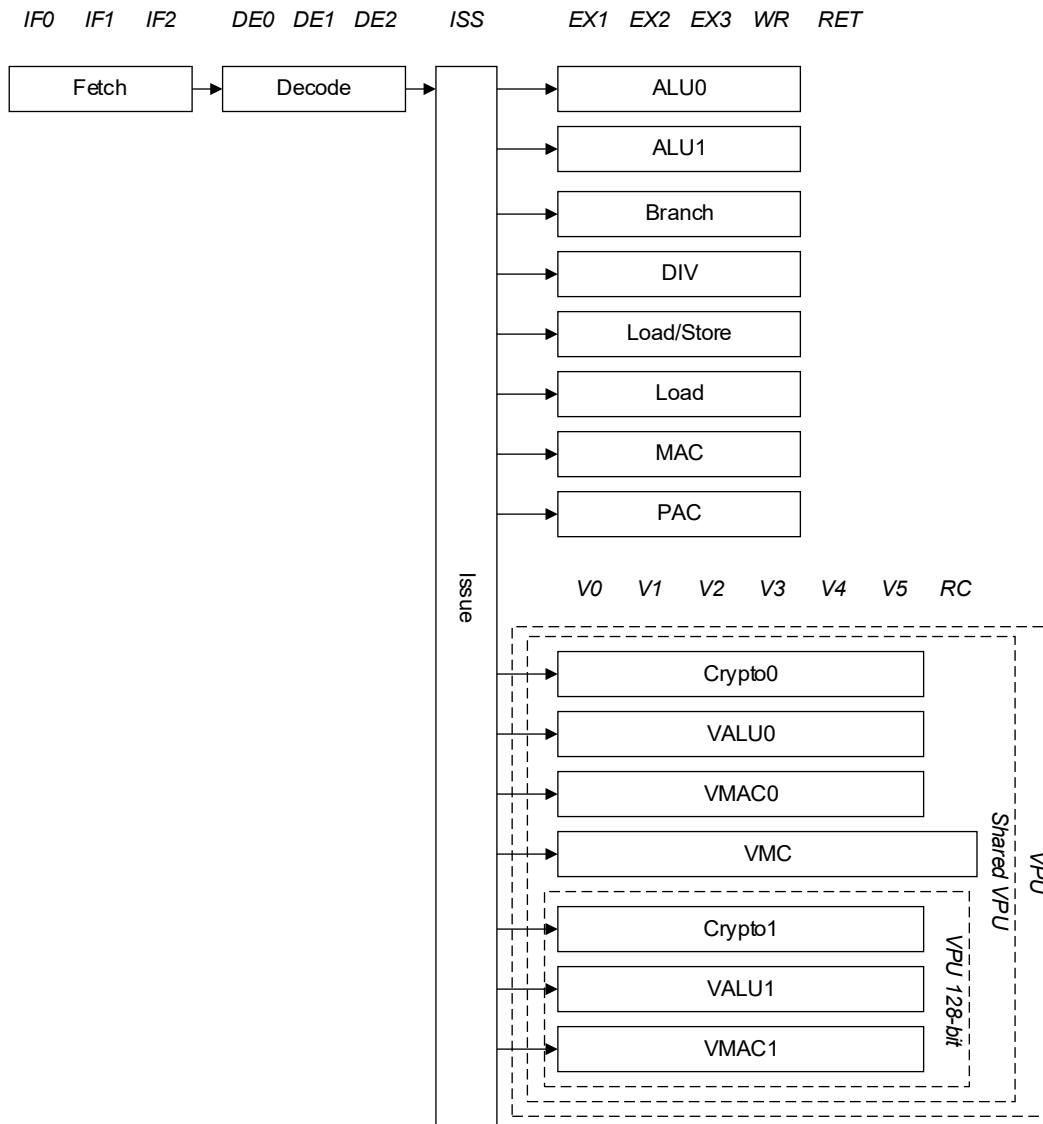


Figure 1 Cortex-A520AE Core pipeline

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

| Pipeline | Instructions |
|------------|----------------------|
| ALU0, ALU1 | Arithmetic and logic |
| Branch | Branch |

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

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 Cortex-A520AE Core 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 |

| Instruction Group | AArch64 Instruction | Exec Latency | Execution Throughput | Utilized Pipeline |
|---------------------------|---------------------|--------------|----------------------|-------------------|
| Branch, immed | B | - | 1 | Branch |
| Branch, register | BX | - | 1 | Branch |
| Branch and link, immed | BL, BLX | 1 | 1 | Branch |
| Branch and link, register | BLX | 1 | 1 | Branch |
| Compare and branch | CBZ, CBNZ | - | 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 |
|---|---------------------|--------------|----------------------|-------------------|
| Arithmetic, basic | ADD, ADC, SUB, SBC | 1 | 2 | ALU |
| Arithmetic, basic, flagset ^[1] | ADDS, SUBS | 1 | 2 | ALU |
| | ADCS, SBCS | 1 | 1 | |

| Instruction Group | AArch64 Instruction | Exec Latency | Execution Throughput | Utilized Pipeline |
|------------------------------|---|------------------|----------------------|-------------------|
| Arithmetic, extend and shift | ADD{S}, SUB{S} | 1 ^[1] | 2 | ALU |
| Conditional compare | CCMN, CCMP | 1 | 1 | ALU |
| Conditional select | CSEL, CSINC, CSINV, CSNEG | 1 | 2 | ALU |
| Logical, basic | AND{S}, BIC{S}, EOR, ORR | 1 | 2 | ALU |
| Logical, shift | AND{S}, BIC{S}, EON, EOR, ORN, ORR | 1 | 2 | ALU |

Notes:

1. Latency=2 when the dependency is on Rm.

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.

Table 3-3 AArch64 Divide and multiply instructions¹

| Instruction Group | AArch64 Instruction | Exec Latency | Execution Throughput | Utilized Pipeline |
|-----------------------------|--------------------------------|--------------|----------------------|-------------------|
| Divide, W-form | SDIV ² , UDIV | 12 | 1/12 | DIV |
| Divide, X-form | SDIV, UDIV | 20 | 1/20 | DIV |
| Multiply accumulate, W-form | MADD, MSUB | 3 | 1 | MAC |
| | MUL | 3 | 1 | |
| Multiply accumulate, X-form | MADD, MSUB, MUL | 4 | 1/2 | MAC |
| Multiply accumulate long | SMADDL, SMSUBL, UMADDL, UMSUBL | 2 | 1 | MAC |
| Multiply high | SMULH, UMULH | 6 | 1/4 | MAC |

Notes:

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

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 | 5 | 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, PACIZA, PACIZB | 5 | 1 | PAC |

| Instruction Group | AArch64 Instruction | Exec Latency | Execution Throughput | Utilized Pipeline |
|---|--|--------------|----------------------|-------------------|
| | PACIA171, PACIB1716, PACIAZ, PACIASP, PACIBSP, PACIBZ | | | |
| Load register, with pointer authentication, offset | LDRAA, LDRAB | 2 | 2 | PAC |
| Load register, with pointer authentication, pre-indexed | LDRAA, LDRAB | 2 | 1 | PAC |
| Strip pointer authentication code | XPACD, XPACI, XPACLRI | 5 | 1/5 | 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 | 2 | ALU |
| Bitfield extract | EXTR | 2 ^[1] | 2 | ALU |
| Bitfield move, basic | SBFM, SBFIZ, SBFX, SXTH, SXTW, UBFM, UBFIZ, UBFX, UXTH | 2 ^[2] | 2 | ALU |
| Bitfield move, insert | BFM | 2 | 2 | ALU |
| Convert floating-point condition flags | AXFLAG, XAFLAG | - | 1/2 | ALU |

| Instruction Group | AArch64 Instruction | Exec Latency | Execution Throughput | Utilized Pipeline |
|--------------------------------|------------------------|--------------|----------------------|-------------------|
| Flag manipulation instructions | SETF8, SETF16 | 2 | 1/2 | ALU |
| | RMIF, | 1 | 1 | |
| | CFINV | | 1/2 | |
| Count leading | CLS, CLZ | 1 | 2 | ALU |
| Move immed | MOVN, MOVK, MOVZ | 1 | 2 | ALU |
| Reverse bits/bytes | REV, REV16, REV32 | 1 | 2 | ALU |
| | RBIT | 2 | 2 | |
| Variable shift | ASRV, LSLV, LSRV, RORV | 1 | 2 | ALU |

Notes:

1. Latency=1 for ROR (immediate) alias of EXTR
2. Latency=1 for LSL (immediate), LSR (immediate) and UXTB aliases of UBFM

Latency=1 for SXTB and ASR (immediate) aliases of SBFM

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 |

| Instruction Group | AArch64 Instruction | Load Latency | Execution Throughput | Utilized Pipeline |
|--|---|--------------|----------------------|-------------------|
| 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 |
| 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 |

| Instruction Group | AArch64 Instruction | Load Latency | Execution Throughput | Utilized Pipeline |
|--|--|--------------|----------------------|-------------------|
| 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 | 2 | ALU |
| Insert Random Tags | IRG | 3 | 1/3 | ALU |
| Insert Tag Mask | GMI | 2 | 2 | ALU |
| Subtract Pointer | SUBP | 2 | 2 | ALU |
| Subtract Pointer, flagset | SUBPS | 2 | 2 | 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 | 1/4 | 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 | - | 1 | Load/Store |
| | ST2G | | 1/2 | |

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

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}, | 5 | 1/5 | VALU |
| | FCMP{E} | 1 | 1 | |
| 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 | 1 | VALU |
| FP square root, H-form | FSQRT | 11 | 2/5 | VMC |
| FP square root, S-form | FSQRT | 14 | 2/9 | VMC |

| Instruction Group | AArch64 Instruction | Exec Latency | Execution Throughput | Utilized Pipeline |
|------------------------|---------------------|--------------|----------------------|-------------------|
| FP square root, D-form | FSQRT | 25 | 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 | 1 | VALU |
| FP convert, Javascript from vec to gen reg | FJCVTZS | 4 | 1 | VALU |
| FP convert, from vec to vec reg | FCVT, FCVTXN | 4 | 2 | VALU |
| FP move, immed | FMOV | 3 | 2 | VALU |
| FP move, register | FMOV | 3 | 1 | VALU |
| FP transfer, from gen to vec reg | FMOV | 3 | 2 | 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 |
| 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 |

| Instruction Group | AArch64 Instruction | Load Latency | Execution Throughput | Utilized Pipeline |
|--|---------------------|--------------|----------------------|-------------------|
| 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 |
| Store vector reg, immed pre-index, B/H/S/D-form | STR | - | 1 | Load/Store |

| Instruction Group | AArch64 Instructions | Exec Latency | Execution Throughput | Utilized Pipeline |
|--|----------------------|--------------|----------------------|-------------------|
| 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 |

| Instruction Group | AArch64 Instructions | Exec Latency | Execution Throughput | Utilized Pipeline |
|--|----------------------|--------------|----------------------|-------------------|
| 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 |
| 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 | 2,1 | VALU |

| Instruction Group | AArch64 Instruction | Exec Latency | Execution Throughput | Utilized Pipeline |
|------------------------------------|---|--------------|----------------------|-------------------|
| ASIMD absolute diff accum | SABA, UABA | 6 | 2/7,1/7 | VALU |
| ASIMD absolute diff accum long | SABAL(2), UABAL(2) | 6 | 1/2,1/4 | VALU |
| ASIMD absolute diff long | SABDL(2), UABDL(2) | 3 | 2,1 | VALU |
| ASIMD arith, basic | ABS, ADD, NEG, SHADD, SHSUB, SUB, UHADD, UHSUB, | 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) | 3 | 2,1 | VALU |
| ASIMD arith, complex | ADDHN(2), RSUBHN(2), SQABS, SQADD, SQNEG, SQSUB, SUBHN(2), SUQADD, UQADD, UQSUB, USQADD | 4 | 2,1 | VALU |
| | RADDHN(2) | 8 | 2/5,1/5 | |
| | SRHADD, URHADD | 3 | 2,1 | |
| ASIMD arith, pair-wise | ADDP, SADDLP, UADDLP | 3 | 2,1 | VALU |

| Instruction Group | AArch64 Instruction | Exec Latency | Execution Throughput | Utilized Pipeline |
|--|--|--------------|----------------------|-------------------|
| ASIMD arith, reduce, 4H/4S | ADDV, SADDLV, UADDLV | 4 | 1 | VALU |
| ASIMD arith, reduce | ADDV | 3 | 1 | VALU |
| ASIMD arith, reduce | SADDLV, UADDLV | 4 | 1 | VALU |
| ASIMD compare | CMEQ, CMGE, CMGT, CMHI, CMHS, CMLE, CMLT | 3 | 2,1 | VALU |
| ASIMD compare test | CMTST | 4 | 2,1 | VALU |
| ASIMD dot product | SDOT, UDOT | 4 | 2,1 | VMAC |
| ASIMD dot product using signed and unsigned integers | SUDOT, USDOT | 4 | 2,1 | VMAC |
| ASIMD logical | AND, BIC, EOR, MOV, MVN, NOT, ORN, ORR | 3 | 2,1 | VALU |
| ASIMD matrix multiply-accumulate | SMMLA, UMMLA, USMMLA | 4 | 2,1 | VALU |
| ASIMD max/min, basic and pair-wise | SMAX, SMAXP, SMIN, SMINP, UMAX, UMAXP, UMIN, UMINP | 3 | 2,1 | VALU |
| ASIMD max/min, reduce, 4H/4S | SMAXV, SMINV, UMAXV, UMINV | 4 | 1 | VALU |

| Instruction Group | AArch64 Instruction | Exec Latency | Execution Throughput | Utilized Pipeline |
|---|--|--------------|----------------------|-------------------|
| ASIMD max/min, reduce, 8B/8H | SMAXV, SMINV, UMAXV, UMINV | 4 | 1 | VALU |
| ASIMD max/min, reduce, 16B | SMAXV, SMINV, UMAXV, UMINV | 4 | 1 | VALU |
| ASIMD multiply | MUL, SQDMULH, SQRDMULH | 4 | 2,1 | VMAC |
| ASIMD multiply accumulate | MLA, MLS | 4 | 2,1 | VMAC |
| ASIMD multiply accumulate high, D-form | SQRDMLAH, SQRDMLSH | 4 | 1 | VMAC |
| ASIMD multiply accumulate high, Q-form | SQRDMLAH, SQRDMLSH | 4 | 1 | VMAC |
| ASIMD multiply accumulate long | SMLAL(2), SMLSL(2), UMLAL(2), UMLSL(2) | 4 | 2,1 | VMAC |
| ASIMD multiply accumulate saturating long | SQDMLAL(2), SQDMLSL(2) | 4 | 2,1 | VMAC |
| ASIMD multiply/multiply long (8x8) polynomial, D-form | PMUL, PMULL(2) | 4 | 2,1 | VALU |
| ASIMD multiply/multiply long (8x8) polynomial, Q-form | PMUL, PMULL(2) | 4 | 2,1 | VALU |
| ASIMD multiply long | SMULL(2), UMULL(2), SQDMULL(2) | 4 | 2,1 | VMAC |
| ASIMD pairwise add and accumulate long | SADALP, UADALP | 6 | 2/5,1/5 | VALU |

| Instruction Group | AArch64 Instruction | Exec Latency | Execution Throughput | Utilized Pipeline |
|--|---|--------------|----------------------|-------------------|
| ASIMD shift accumulate | SRSRA, URSRA | 8 | 2/5,1/5 | VALU |
| | SSRA, USRA | 3 | 2,1 | |
| ASIMD shift by immed, basic | SHL, SHLL(2), SSHLL(2), SSHR, SXTL(2), USHLL(2), USHR, UXTL(2) | 3 | 2,1 | VALU |
| ASIMD shift by immed, basic | SHRN(2), | 4 | 2,1 | VALU |
| ASIMD shift by immed and insert, basic | SLI, SRI | 3 | 2,1 | VALU |
| ASIMD shift by immed, complex | RSHRN(2), SQRSHRN(2), SQRSHRUN(2), SQSHL{U}, SQSHRN(2), SQSHRUN(2), UQRSHRN(2), UQSHL, UQSHRN(2), | 4 | 2,1 | VALU |
| ASIMD shift by register, basic | SSHL, USHL, SRSHL, SRSHR, URSHL, URSHR | 3 | 2,1 | VALU |
| ASIMD shift by register, complex | SQRSHL, SQSHL, UQRSHL, UQSHL | 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 | 2,1 | VALU |
| ASIMD FP arith, normal | FABD, FADD, FSUB, FADDP | 4 | 2,1 | VALU |
| ASIMD FP compare | FACGE, FACGT, FCMEQ, FCMGE, FCMGT, FCMLE, FCMLT | 3 | 2,1 | VALU |
| ASIMD FP complex add | FCADD | 4 | 2,1 | VMAC |
| ASIMD FP complex multiply add | FCMLA | 4 | 2,1 | VMAC |
| ASIMD FP convert, long (F16 to F32) | FCVTL(2) | 4 | 2,1 | VALU |
| ASIMD FP convert, long (F32 to F64) | FCVTL(2) | 4 | 2,1 | VALU |
| ASIMD FP convert, narrow (F32 to F16) | FCVTN(2) | 4 | 2,1 | VALU |
| ASIMD FP convert, narrow (F64 to F32) | FCVTN(2), FCVTXN(2) | 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 | 2,1 | VALUE |

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

| Instruction Group | AArch64 Instruction | Exec Latency | Execution Throughput | Utilized Pipeline |
|---|--|--------------|----------------------|-------------------|
| ASIMD FP max/min, reduce | FMAXV, FMAXNMV, FMINV, FMINNMV | 4 | 1 | VALU |
| ASIMD FP max/min, reduce, Q-form F16 | FMAXV, FMAXNMV, FMINV, FMINNMV | 4 | 1 | VALU |
| ASIMD FP multiply | FMUL, FMULX | 4 | 2,1 | VMAC |
| ASIMD FP multiply accumulate | FMLA, FMLS | 4 | 2,1 | VMAC |
| ASIMD FP multiply accumulate long | FMLAL(2), FMLSL(2) | 4 | 2,1 | VMAC |
| ASIMD FP negate | FNEG | 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 | 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 | 2,1 | VALU |

| Instruction Group | AArch64 Instruction | Exec Latency | Execution Throughput | Utilized Pipeline |
|-----------------------------------|--|--------------|----------------------|-------------------|
| ASIMD FP round, Q-form F16 | FRINTA, FRINTI, FRINTM, FRINTN, FRINTP, FRINTX, FRINTZ, FRINT32X, FRINT64X, FRINT32Z, FRINT64Z | 4 | 2,1 | VALU |
| ASIMD FP square root, D-form, F16 | FSQRT | 8 | 2/5 | VMC |
| ASIMD FP square root, D-form, F32 | FSQRT | 12 | 2/9 | VMC |
| ASIMD FP square root, Q-form, F16 | FSQRT | 8 | 1/5 | VMC |
| ASIMD FP square root, Q-form, F32 | FSQRT | 12 | 1/9 | VMC |
| ASIMD FP square root, Q-form, F64 | FSQRT | 22 | 1/19 | VMC |

Notes:

1. 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 | 2,1 | VALU |
| ASIMD dot product | BFDOT | 10 | 2,1 | VMAC, VALU |
| ASIMD matrix multiply accumulate | BFMMLA | 14, 15 | 1,1/2 | VMAC, VALU |
| ASIMD multiply accumulate long | BFMLALB, BFMLALT | 4 | 2,1 | VMAC |

| Instruction Group | AArch64 Instruction | Exec Latency | Execution Throughput | Utilized Pipeline |
|-----------------------------|---------------------|--------------|----------------------|-------------------|
| Scalar convert, F32 to BF16 | BFCVT | 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 | 2,1 | VALU |
| ASIMD bitwise insert | BIF, BIT, BSL | 3 | 2,1 | VALU |
| ASIMD count | CLS, CLZ, CNT | 3 | 2,1 | VALU |
| ASIMD duplicate, gen reg | DUP | 3 | 1 | VALU |
| ASIMD duplicate, element | DUP | 3 | 2,1 | VALU |
| ASIMD extract | EXT | 3 | 2,1 | VALU |
| ASIMD extract narrow | XTN | 4 | 2,1 | VALU |
| ASIMD extract narrow, saturating | SQXTN(2), SQXTUN(2), UQXTN(2) | 4 | 2,1 | VALU |
| ASIMD insert, element to element | INS | 4 | 2,1 | VALU |
| ASIMD move, FP immed | FMOV | 3 | 2,1 | VALU |
| ASIMD move, integer immed | MOVI, MVNI | 3 | 2,1 | VALU |
| ASIMD reciprocal estimate, D-form F32 and F64 | FRECPE, FRECPX, FRSQRTE, URECPE, URSQRTE | 4 | 2,1 | VMAC |

| Instruction Group | AArch64 Instruction | Exec Latency | Execution Throughput | Utilized Pipeline |
|--|--|--------------|----------------------|-------------------|
| ASIMD reciprocal estimate, D-form F16 and Q-form F32 | FRECPE, FRECPX, FRSQRTE, URECPE, URSQRTE | 4 | 2,1 | VMAC |
| ASIMD reciprocal estimate, Q-form F16 | FRECPE, FRECPX, FRSQRTE, URECPE, URSQRTE | 4 | 2,1 | VMAC |
| ASIMD reciprocal step | FRECPS, FRSQRSTS | 4 | 2,1 | VMAC |
| ASIMD reverse | REV16, REV32, REV64 | 3 | 2,1 | VALU |
| ASIMD table lookup, 1 table regs | TBL | 4 | 2,1 | VALU |
| ASIMD table lookup, 2 table regs | TBL | 8 | 2/5 | VALU |
| ASIMD table lookup, 3 table regs | TBL | 12 | 1/5 | VALU |
| ASIMD table lookup, 4 table regs | TBL | 16 | 1/9 | VALU |
| ASIMD table lookup extension, 1 table reg | TBX | 8 | 2/5 | VALU |
| ASIMD table lookup extension, 2 table reg | TBX | 12 | 1/5 | VALU |
| ASIMD table lookup extension, 3 table reg | TBX | 16 | 1/9 | VALU |
| ASIMD table lookup extension, 4 table reg | TBX | 20 | 1/13 | VALU |
| ASIMD transfer, element to gen reg | UMOV, SMOV | 3 | 1 | VALU |
| ASIMD transfer, gen reg to element | INS | 3 | 1 | VALU |

| Instruction Group | AArch64 Instruction | Exec Latency | Execution Throughput | Utilized Pipeline |
|-------------------|--------------------------|--------------|----------------------|-------------------|
| ASIMD transpose | TRN1, TRN2 | 3 | 2,1 | VALU |
| ASIMD unzip/zip | UZIP1, UZIP2, ZIP1, ZIP2 | 3 | 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 |
| 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 | 1/2 | Load/Store |
| ASIMD load, 1 element, multiple, 4 reg, Q-form | LD1 | 4 | 1/2 | Load/Store |

| Instruction Group | AArch64 Instruction | Load Latency | Execution Throughput | Utilized Pipeline |
|---|---------------------|--------------|----------------------|-------------------|
| 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 | 1 | 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 | 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 |

| Instruction Group | AArch64 Instruction | Load Latency | Execution Throughput | Utilized Pipeline |
|---|---------------------|--------------|----------------------|-------------------|
| ASIMD load, 3 element, multiple, D-form, B/H/S | LD3 | 5 | 1/3 | Load/Store |
| 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 |

| Instruction Group | AArch64 Instruction | Load Latency | Execution Throughput | Utilized Pipeline |
|---|---------------------|--------------|----------------------|-------------------|
| 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 |

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

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

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 | 2,1 | Crypto |
| Crypto polynomial (64x64) multiply long | PMULL (2) | 4 | 2 | VMC |
| Crypto SHA1 hash acceleration op | SHA1H | 3 | 1,1/2 | VALU |
| Crypto SHA1 hash acceleration ops | SHA1C, SHA1M, SHA1P | 4 | 2 | VMC |
| Crypto SHA1 schedule acceleration ops | SHA1SU0, SHA1SU1 | 3 | 2 | VMC |

| Instruction Group | AArch64 Instruction | Exec Latency | Execution Throughput | Utilized Pipeline |
|---|--|--------------|----------------------|-------------------|
| Crypto SHA256 hash acceleration ops | SHA256H, SHA256H2 | 4 | 2 | VMC |
| Crypto SHA256 schedule acceleration ops | SHA256SU0, SHA256SU1 | 4 | 2 | VMC |
| Crypto SHA512 hash acceleration ops | SHA512H, SHA512H2, SHA512SU0, SHA512SU1 | 9 | 1/9 | VMC |
| Crypto SHA3 ops | BCAX, EOR3, | 3 | 2,1 | VALU |
| | XAR | 4 | | |
| Crypto SHA3 ops RAX1 | RAX1 | 9 | 1/9 | VMC |
| Crypto SM3 ops | SM3PARTW1, SM3PARTW2, SM3SS1, SM3TT1A, SM3TT1B, SM3TT2A, SM3TT2B | 9 | 1/9 | VMC |
| Crypto SM4 ops | SM4E, SM4EKEY | 9 | 1/9 | 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 | 2 | 1 | ALU |
| Loop control, based on predicate and flag setting | BRKAS, BRKBS | 2 | 1 | ALU |
| Loop control, propagating | BRKN, BRKPA, BRKPB | 2 | 1 | ALU |
| Loop control, propagating and flag setting | BRKNS, BRKPAS, BRKPBS | 2 | 1 | ALU |
| Loop control, based on GPR | WHILEGE, WHILEGT, WHILEHI, WHILEHS, WHILELE, WHILELO, WHILELS, WHILELT, WHILERW, WHILEWR | 2 | 1 | ALU |
| Loop terminate ^[1] | CTERMEQ, CTERMNE | 1 | 1 | ALU |
| Predicate counting scalar | ADDPL, ADDVL, RDVL, | 1 | 2 | ALU |

| Instruction Group | SVE Instruction | Exec Latency | Execution Throughput | Utilized Pipeline |
|---|---|--------------|----------------------|-------------------|
| | CNTB, CNTH, CNTW, CNTD, DECB, DECH, DECW, DECD, INCB, INCH, INCW, INCD, | 1 | 1 | ALU |
| | SQDECB, SQDECH, SQDECW, SQDECD, SQINCB, SQINCH, SQINCW, SQINCD, UQDECB, UQDECH, UQDECW, UQDECD, UQINCB, UQINCH, UQINCW, UQINCD | 5 | 1 | |
| Predicate counting scalar, active predicate | CNTP, DECP, INCP | 6 | 1 | ALU |
| Predicate counting scalar, active predicate, saturating, 64-bit | SQDECP, SQINCP, UQDECP, UQINCP | 9 | 1/3,1/6 | ALU |
| Predicate counting scalar, active predicate, saturating, 32-bit | SQDECP, SQINCP, | 3 | 1/3,1/6 | ALU |
| | UQDECP, UQINCP | 9 | | |

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

Notes:

1. Instructions with dependencies may be co-issue.

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 | 2,1 | VALU |
| Arithmetic, absolute diff accum | SABA, UABA | 6 | 1/2,1/4 | VALU |
| Arithmetic, absolute diff accum long | SABALB, SABALT, UABALB, UABALT | 6 | 1/2,1/4 | VALU |
| Arithmetic, absolute diff long | SABDLB, SABDLT, UABDLB, UABDLT | 3 | 2,1 | VALU |
| Arithmetic, basic | ABS, ADD, ADR, CNOT, NEG, SHADD, SHSUB, SHSUBR, SRHADD, SUB, UADDWB, UADDWT, UHADD, UHSUB, UHSUBR, URHADD, | 3 | 2,1 | VALU |
| | SUBHNB, SUBHNT, SUBR, USUBWB, USUBWT | 4 | | |

| Instruction Group | SVE Instruction | Exec Latency | Execution Throughput | Utilized Pipeline |
|---|--|--------------|----------------------|-------------------|
| Arithmetic, basic | SADDLB, SADDLBT, SADDLT, SADDWB, SADDWT, SSUBLB, SSUBLBT, SSUBLT, SSUBLTB, SSUBWB, SSUBWT, UADDLB, UADDLT, USUBLB, USUBLT, | 4 | 2,1 | VALU |
| Arithmetic, complex | ADDHNB, ADDHNT, SQABS, SQADD, SQNEG, SQSUB, SQSUBR, SUQADD, UQADD, UQSUB, UQSUBR, USQADD, | 4 | 2,1 | VALU |
| | RADDHNB, RADDHNT, RSUBHNB, RSUBHNT | 8 | 2/5,1/5 | |
| Arithmetic, large integer | ADCLB, ADCLT, SBCLB, SBCLT | 4 | 2,1 | VALU |
| Arithmetic, pairwise add | ADDP | 3 | 2,1 | VALU |
| Arithmetic, pairwise add and accum long | SADALP, UADALP | 7 | 2/5,1/5 | VALU |
| Arithmetic, shift | ASR, ASRR, LSL, LSLR, LSR, LSRR | 3 | 2,1 | VALU |
| Arithmetic, shift and accumulate | USRA | 4 | 2,1 | VALU |

| Instruction Group | SVE Instruction | Exec Latency | Execution Throughput | Utilized Pipeline |
|---|--|--------------|----------------------|-------------------|
| Arithmetic, shift and accumulate complex | SRSRA, URSRA, | 7 | 2/5,1/5 | VALU |
| | SSRA | 4 | 2,1 | |
| Arithmetic, shift by immediate | SHRNB, SHRNT, SSHLLB, SSHLLT, USHLLB, USHLLT | 3 | 2,1 | VALU |
| Arithmetic, shift by immediate and insert | SLI, SRI | 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 | 2,1 | VALU |
| Arithmetic, shift right for divide | ASRD | 4 | 2,1 | VALU |
| Arithmetic, shift rounding | SRSHL, SRSHLR, SRSHR, URSHL, URSHLR, URSHR | 4 | 2,1 | VALU |
| Bit manipulation (B) | BDEP, BEXT, BGRP | 14 | 1/14 | VMC |

| Instruction Group | SVE Instruction | Exec Latency | Execution Throughput | Utilized Pipeline |
|---|--|--------------|----------------------|-------------------|
| 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 | 2,1 | VALU |
| Count/reverse bits | CLS, CLZ, RBIT | 3 | 2,1 | VALU |
| Count (B,H) | CNT | 3 | 2,1 | VALU |
| Count (S) | CNT | 8 | 2/5,1/5 | VALU |
| Count (D) | CNT | 12 | 1/5,1/10 | VALU |
| Broadcast logical bitmask immediate to vector | DUPM, MOV | 4 | 2,1 | VALU |
| Compare and set flags | CMPEQ, CMPGE, CMPGT, CMPHI, CMPHS, CMPLE, CMPLO, CMPLS, CMPLT, CMPNE | 5 | 2,1 | VALU |
| Complex add | CADD | 3 | 2,1 | VALU |
| Complex add saturating | SQCADD | 4 | 2,1 | VALU |
| Complex dot product 8-bit element | CDOT | 4 | 2,1 | VMAC |
| Complex dot product 16-bit element | CDOT | 4 | 2,1 | VMAC |
| Complex multiply-add B, H, S element size | CMLA | 4 | 2,1 | VMAC |
| Complex multiply-add D element size | CMLA | 4 | 2,1 | VMAC |

| Instruction Group | SVE Instruction | Exec Latency | Execution Throughput | Utilized Pipeline |
|---|------------------------------------|--------------|----------------------|-------------------|
| Conditional extract operations, general purpose register | CLASTA, CLASTB | 8 | 1,1 | VALU |
| Conditional extract operations, SIMD&FP scalar and vector forms | CLASTA, CLASTB, COMPACT, SPLICE | 4 | 2,1 | VALU |
| Convert to floating point, 64b to float or convert to double | SCVTF, UCVTF | 4 | 2,1 | VALU |
| Convert to floating point, 32b to single or half | SCVTF, UCVTF | 4 | 2,1 | VALU |
| Convert to floating point, 16b to half | SCVTF, UCVTF | 4 | 2,1 | VALU |
| Copy, scalar | CPY | 3 | 2,1 | VALU |
| Copy, scalar SIMD&FP or imm | CPY | 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 | 2,1 | VMAC |
| Dot product, 8 bit, using signed and unsigned integers | SUDOT, USDOT | 4 | 2,1 | VMAC |
| Dot product, 16 bit | SDOT, UDOT | 4 | 2,1 | VMAC |
| Duplicate, immediate and indexed form | DUP, MOV | 3 | 2,1 | VALU |
| Duplicate, indexed > elem | DUP | 3 | 2,1 | VALU |
| Duplicate, scalar form | DUP, MOV | 3 | 2,1 | VALU |
| Extend, sign or zero | SXTB, SXTH, SXTW, UXTB, UXTH, UXTW | 3 | 2,1 | VALU |

| Instruction Group | SVE Instruction | Exec Latency | Execution Throughput | Utilized Pipeline |
|---|---|--------------|----------------------|-------------------|
| Extract | EXT | 3 | 2,1 | VALU |
| Extract narrow saturating | SQXTNB, SQXTNT, SQXTUNB, SQXTUNT, UQXTNB, UQXTNT | 4 | 2,1 | VALU |
| Extract/insert operation, SIMD and FP scalar form | LASTA, LASTB, INSR | 4 | 2,1 | VALU |
| Extract/insert operation, scalar | LASTA, LASTB, | 8,8 | 1/3,1/3 | VALU0 |
| | INSR | 4 | 2,1 | |
| Histogram operations | HISTCNT, HISTSEG | 8 | 2/5 | VALU0 |
| Horizontal operations, B, H, S form, immediate operands only | INDEX | 4 | 2,1 | VMAC |
| Horizontal operations, B, H, S form, scalar, immediate operands)/ scalar operands only / immediate, scalar operands | INDEX | 4 | 1,1 | VMAC |
| Horizontal operations, D form, immediate operands only | INDEX | 4 | 2,1 | VMAC |
| Horizontal operations, D form, scalar, immediate operands)/ scalar operands only / immediate, scalar operands | INDEX | 4 | 1,1 | VMAC |
| Logical | AND, BIC, EON, EOR, MOV, NOT, ORN, ORR | 3 | 2,1 | VALU |

| Instruction Group | SVE Instruction | Exec Latency | Execution Throughput | Utilized Pipeline |
|--|---|--------------|----------------------|-------------------|
| Logical | EORBT, EORTB, | 4 | 2,1 | VALU |
| Max/min, basic and pairwise | SMAX, SMAXP, SMIN, SMINP, UMAX, UMAXP UMIN, UMINP | 3 | 2,1 | VALU |
| Matching operations | MATCH, NMATCH | 7,7 | 1/4,1/4 | VALU |
| Matrix multiply-accumulate | SMMLA, UMMLA, USMMLA | 4 | 2,1 | VMAC |
| Move prefix | MOVPRFX | 3 | 2,1 | VALU |
| Multiply, B, H, S element size | MUL, SMULH, UMULH | 4 | 2,1 | VMAC |
| Multiply, D element size | MUL, SMULH, UMULH | 4 | 2,1 | VMAC |
| Multiply long | SMULLB, SMULLT, UMULLB, UMULLT | 4 | 2,1 | VMAC |
| Multiply accumulate, B, H, S element size | MLA, MLS | 4 | 2,1 | VMAC |
| Multiply accumulate, D element size | MLA, MLS, MAD, MSB, | 4 | 2,1 | VMAC |
| Multiply accumulate long | SMLALB, SMLALT, SMLSBLB, SMLSBLT, UMLALB, UMLALT, UMLSBLB, UMLSBLT | 4 | 2,1 | VMAC |
| Multiply accumulate saturating doubling long regular | SQDMLALB, SQDMLALT, SQDMLALBT, SQDMLSBLB, SQDMLSBLT, SQDMLSBLT | 4 | 2,1 | VMAC |

| Instruction Group | SVE Instruction | Exec Latency | Execution Throughput | Utilized Pipeline |
|--|------------------------------------|--------------|----------------------|-------------------|
| Multiply saturating doubling high, B, H, S element size | SQDMULH | 4 | 2,1 | VMAC |
| Multiply saturating doubling high, D element size | SQDMULH | 4 | 2,1 | VMAC |
| Multiply saturating doubling long | SQDMULLB, SQDMULLT | 4 | 2,1 | VMAC |
| Multiply saturating rounding doubling regular/complex accumulate, B, H, S element size | SQRDMLAH, SQRDMLSH, SQRDCMLAH | 4 | 1,1 | VMAC |
| Multiply saturating rounding doubling regular/complex accumulate, D element size | SQRDMLAH, SQRDMLSH, SQRDCMLAH | 4 | 1,1 | VMAC |
| Multiply saturating rounding doubling regular/complex, B, H, S element size | SQRDMULH | 4 | 2,1 | VMAC |
| Multiply saturating rounding doubling regular/complex, D element size | SQRDMULH | 4 | 2,1 | VMAC |
| Multiply/multiply long, (8, 16, 32) polynomial | PMUL, PMULLB, PMULLT | 4 | 2,1 | VALU |
| Multiply/multiply long, (64) polynomial | PMULLB, PMULLT | 9 | 1/9 | VMC |
| Predicate counting vector | DECH, DECW, DECD, INCH, INCW, INCD | 3 | 2,1 | VALU |

| Instruction Group | SVE Instruction | Exec Latency | Execution Throughput | Utilized Pipeline |
|---------------------------------------|--|--------------|----------------------|-------------------|
| Predicate counting vector, saturating | SQDECH, SQDECW, SQDECD, SQINCH, SQINCW, SQINCD, UQDECH, UQDECW, UQDECD, UQINCH, UQINCW, UQINCD | 4 | 2,1 | VALU |
| Reciprocal estimate | URECPE, URSQRTE | 4 | 2,1 | VMAC |
| Reduction, arithmetic, B form | SADDV, UADDV, SMAXV, SMINV, UMAXV, UMINV | 4 | 1 | VALU0 |
| Reduction, arithmetic, H form | SADDV, UADDV, SMAXV, SMINV, UMAXV, UMINV | 4 | 1 | VALU0 |
| Reduction, arithmetic, S form | SADDV, UADDV, SMAXV, SMINV, UMAXV, UMINV | 4 | 1 | VALU0 |
| Reduction, logical | ANDV, EORV, ORV | 4 | 1 | VALU0 |
| Reverse, vector | REV, REVB, REVH, REVW | 3,3 | 2,1 | VALU |
| Select, vector form | MOV, SEL | 3 | 2,1 | VALU |
| Table lookup | TBL | 4 | 2,1 | VALU |
| Table lookup, double table | TBL | 8 | 2/5,1/5 | VALU |
| Table lookup extension | TBX | 4 | 2,1 | VALU |
| Transpose, vector form | TRN1, TRN2 | 3 | 2,1 | VALU |

| Instruction Group | SVE Instruction | Exec Latency | Execution Throughput | Utilized Pipeline |
|-------------------|---|--------------|----------------------|-------------------|
| Unpack and extend | SUNPKHI, SUNPKLO, UUNPKHI, UUNPKLO | 4 | 2,1 | VALU |
| Zip/unzip | UZP1, UZP2, ZIP1, ZIP2 | 3 | 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 | 2,1 | VALU |
| Floating point arithmetic | FADD, FADDP, FNEG, FSUB, FSUBR | 4 | 2,1 | VALU |
| Floating point associative add, F16 | FADDA | 32 | 1/25 | VALU |
| Floating point associative add, F32 | FADDA | 16 | 1/9,1/18 | VALU |
| Floating point associative add, F64 | FADDA | 8 | 2/5,1/5 | VALU |
| Floating point compare | FACGE, FACGT, FACLE, FACLT, FCMEQ, FCMGE, FCMGT, FCMLE, FCMLT, FCMNE, FCMUO | 4 | 1,1/2 | VALU |
| Floating point complex add | FCADD | 4 | 2,1 | VALU |

| Instruction Group | SVE Instruction | Exec Latency | Execution Throughput | Utilized Pipeline |
|---|----------------------|--------------|----------------------|-------------------|
| Floating point complex multiply add | FCMLA | 4 | 2,1 | VMAC |
| Floating point convert, long or narrow (F16 to F32 or F32 to F16) | FCVT, FCVTLT, FCVTNT | 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 | 2,1 | VALU |
| Floating point convert, round to odd | FCVTX, FCVTXNT | 4 | 2,1 | VALU |
| Floating point base2 log, F16 | FLOGB | 4 | 2,1 | VMAC |
| Floating point base2 log, F32 | FLOGB | 4 | 2,1 | VMAC |
| Floating point base2 log, F64 | FLOGB | 4 | 2,1 | VMAC |
| Floating point convert to integer, F16 | FCVTZS, FCVTZU | 4 | 2,1 | VALU |
| Floating point convert to integer, F32 | FCVTZS, FCVTZU | 4 | 2,1 | VALU |
| Floating point convert to integer, F64 | FCVTZS, FCVTZU | 4 | 2,1 | VALU |
| Floating point copy | FCPY, FDUP, FMOV | 3 | 2,1 | VALU |
| Floating point divide, F16 ¹ | FDIV, FDIVR | 8 | 1/5 | VMC |
| Floating point divide, F32 ¹ | FDIV, FDIVR | 13 | 1/10 | VMC |
| Floating point divide, F64 ¹ | FDIV, FDIVR | 22 | 1/19 | VMC |

| Instruction Group | SVE Instruction | Exec Latency | Execution Throughput | Utilized Pipeline |
|---|--|--------------|----------------------|-------------------|
| Floating point min/max pairwise | FMAXP, FMAXNMP, FMINP, FMINNMP | 4 | 2,1 | VALU |
| Floating point min/max | FMAX, FMIN, FMAXNM, FMINNM | 4 | 2,1 | VALU |
| Floating point multiply | FSCALE, FMUL, FMULX | 4 | 2,1 | VMAC |
| Floating point multiply accumulate | FMLA, FMLS, FMAD, FMSB, FNMAD, FNMLA, FNMLS, FNMSB | 4 | 2,1 | VMAC |
| Floating point multiply add/sub accumulate long | FMLALB, FMLALT, FMLS LB, FMLS LT | 4 | 2,1 | VMAC |
| Floating point reciprocal estimate, F16 | FRECPE, FRECPX, FRSQRTE | 4 | 2,1 | VMAC |
| Floating point reciprocal estimate, F32 | FRECPE, FRECPX, FRSQRTE | 4 | 2,1 | VMAC |
| Floating point reciprocal estimate, F64 | FRECPE, FRECPX, FRSQRTE | 4 | 2,1 | VMAC |
| Floating point reciprocal step | FRECPS, FRSQRTS | 4 | 2,1 | VMAC |
| Floating point reduction, F16 | FMAXNMV, FMAXV, FMINNMV, FMINV | 4 | 1 | VALU0 |
| Floating point reduction, F16 | FADDV | 12 | 1/5 | VALU0 |

| Instruction Group | SVE Instruction | Exec Latency | Execution Throughput | Utilized Pipeline |
|---|--|--------------|----------------------|-------------------|
| Floating point reduction, F32 | FADDV | 8 | 2/5 | VALU0 |
| Floating point reduction, F64 | FADDV | 4 | 2 | VALU0 |
| Floating point round to integral, F16 | FRINTA, FRINTI, FRINTM, FRINTN, FRINTP, FRINTX, FRINTZ | 4 | 2,1 | VALU |
| Floating point round to integral, F32 | FRINTA, FRINTI, FRINTM, FRINTN, FRINTP, FRINTX, FRINTZ | 4 | 2,1 | VALU |
| Floating point round to integral, F64 | FRINTA, FRINTI, FRINTM, FRINTN, FRINTP, FRINTX, FRINTZ | 4 | 2,1 | VALU |
| Floating point square root, F16 | FSQRT | 8 | 1/5 | VMC |
| Floating point square root, F32 | FSQRT | 12 | 1/9 | VMC |
| Floating point square root F64 | FSQRT | 22 | 1/19 | VMC |
| Floating point trigonometric exponentiation | FEXPA | 4 | 2,1 | VMAC |
| Floating point trigonometric multiply add | FTMAD | 4 | 2,1 | VMAC |
| Floating point trigonometric starting value | FTSMUL | 4 | 2,1 | VMAC |

| Instruction Group | SVE Instruction | Exec Latency | Execution Throughput | Utilized Pipeline |
|---|-----------------|--------------|----------------------|-------------------|
| Floating point trigonometric select coefficient | FTSSEL | 33 | 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 | 2, 1 | VALU |
| Dot product | BFDOT | 10 | 2, 1 | VMAC, VALU |
| Matrix multiply accumulate | BFMMLA | 14, 15 | 1, 1/2 | VMAC, VALU |
| Multiply accumulate long | BFMLALB, BFMLALT | 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 |

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

| Instruction Group | SVE Instruction | Load Latency | Execution Throughput | Utilized Pipeline |
|--|--|--------------|----------------------|-------------------|
| 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 | 1/2 | 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 | 1/4 | Load/Store |
| Contiguous Load four structures to four vectors, scalar + imm | LD4B, LD4D, LD4H, LD4W | 5 | 1/3 | Load/Store |

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

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

| Instruction Group | SVE Instruction | Load Latency | Execution Throughput | Utilized Pipeline |
|-------------------------------------|---|--------------|----------------------|-------------------|
| Gather load, 64-bit unscaled offset | LD1B, LD1D, LD1H, LD1W, LDFF1B, LDFF1D, LDFF1H, LDFF1SB, LDFF1SH, LDFF1SW, LDFF1W | 7 | 1/7 | Load/Store |
| Gather load, 64-bit scaled offset | LD1D, LD1H, LD1W, LDFF1D, LDFF1H, LDFF1SH, LDFF1SW, LDFF1W | 7 | 1/7 | Load/Store |

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 |
| Contiguous store, scalar + scalar | ST1H, ST1B, ST1D, ST1W | - | 1 | Load/Store |

| Instruction Group | SVE Instructions | Exec Latency | Execution Throughput | Utilized Pipeline |
|---|--------------------------------|--------------|----------------------|-------------------|
| Contiguous store two structures from two vectors, scalar + imm | ST2B, ST2H, ST2D, ST2W | - | 1/2 | Load/Store |
| Contiguous store two structures from two vectors, scalar + scalar | ST2H, ST2B, ST2D, ST2W | - | 1/2 | Load/Store |
| Contiguous store three structures from three vectors, scalar + imm | ST3B, ST3H, ST3W | - | 1/6 | Load/Store |
| | ST3D | | 1/3 | Load/Store |
| Contiguous store three structures from three vectors, scalar + scalar | ST3B, ST3H, ST3W | - | 1/6 | Load/Store |
| | ST3D | - | 1/3 | Load/Store |
| Contiguous store four structures from four vectors, scalar + imm | ST4B, ST4H, ST4W | - | 1/8 | Load/Store |
| | ST4D | | 1/4 | Load/Store |
| Contiguous store four structures from four vectors, scalar + scalar | ST4B, ST4H, ST4W | - | 1/8 | Load/Store |
| | ST4D | - | 1/4 | Load/Store |
| Non temporal store, scalar + imm | STNT1B, STNT1D, STNT1H, STNT1W | - | 1 | Load/Store |
| Non temporal store, scalar + scalar | STNT1H, STNT1B, STNT1D, STNT1W | - | 1 | Load/Store |
| | | | | |
| Scatter non temporal store, vector + scalar 32-bit element size | STNT1B, STNT1H, STNT1W | - | 1/9 | Load/Store |
| Scatter non temporal store, vector + scalar 64-bit element size | STNT1B, STNT1D, STNT1H, STNT1W | - | 1/7 | Load/Store |

| Instruction Group | SVE Instructions | Exec Latency | Execution Throughput | Utilized Pipeline |
|--|------------------------|--------------|----------------------|-------------------|
| Scatter store vector + imm 32-bit element size | ST1B, ST1H, ST1W | - | 1/9 | Load/Store |
| Scatter store vector + imm 64-bit element size | ST1B, ST1D, ST1H, ST1W | - | 1/7 | Load/Store |
| Scatter store, 32-bit scaled offset | ST1H, ST1W | - | 1/8 | Load/Store |
| Scatter store, 32-bit unpacked unscaled offset | ST1B, ST1D, ST1H, ST1W | - | 1/8 | Load/Store |
| Scatter store, 32-bit unpacked scaled offset | ST1D, ST1H, ST1W | - | 1/8 | Load/Store |
| Scatter store, 32-bit unscaled offset | ST1B, ST1H, ST1W | - | 1/8 | Load/Store |
| Scatter store, 64-bit scaled offset | ST1D, ST1H, ST1W | - | 1/8 | Load/Store |
| Scatter store, 64-bit unscaled offset | ST1B, ST1D, ST1H, ST1W | - | 1/8 | 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 | 3 | 1 | Load/Store |
| Read first fault register and set flags | RDFFRS | 3 | 1 | Load/Store |
| Set first fault register | SETFFR | 1 | 1 | Load/Store |

| Instruction Group | SVE Instruction | Exec Latency | Execution Throughput | Utilized Pipeline |
|-------------------------------|-----------------|--------------|----------------------|-------------------|
| 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 | 2,1 | Crypto |
| Crypto SHA3 ops | BCAX, EOR3, XAR | 4 | 2,1 | VALU |
| Crypto SHA3 ops RAX1 | RAX1 | 9 | 1/9 | VMC |
| Crypto SM4 ops | SM4E, SM4EKEY | 9 | 1/9 | 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:

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

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

The following are multicycle:

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

4.2 Instruction fusion

Cortex-A520AE Core can accelerate key instruction pairs in an operation called fusion.

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

- 'AESE + AESMC' and 'AESD + AESIMC' (see 4.13)
- MOVPRFX fusion: Cortex-A520AE Core implements instruction fusion for MOVPRFX instructions followed by SVE data processing instructions in all cases where the instruction pair is defined as architecturally predictable other than those listed below, and the fused pair will execute with the latency of the SVE data processing instruction. Due to microarchitectural limitations, the following instructions will not fuse with an unpredicated MOVPRFX: FCMLA, FMAD, FMLA, FMLS, FNMAD, FNMLA, FNMLS, FNMSB, MAD, MLA, MLS, MSB, UDOT, BFMLALB, BFMLALT, SMMLA, UMMLA, USMMLA, USDOT, SUDOT.
The following instructions will not fuse with a predicated or unpredicated MOVPRFX: CNT, SABA, SABALB, SABALT, UABA, UABALB, UABALT, URSRA.

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. Cortex-A520AE 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 A64 low latency pointer forwarding

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

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

In Cortex-A520AE Core, there are dedicated forwarding paths that always allow this sequence to be executed without incurring a dependency-based stall.

4.6 AUT* RET forwarding

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

4.7 SIMD MAC forwarding

For the following integer SIMD instructions:

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

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

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

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

4.8 Memory Tagging Extensions

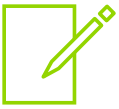
Enabling precise tag checking can prevent Cortex-A520AE Core from entering write-streaming mode. This can reduce performance and increase power for larger writes, and memset or memcpy-like workloads.

4.9 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

4.10 Cache maintenance operations

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

4.11 Cache access latencies

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

Table 4-1: Cortex-A520AE cache access latencies

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

4.12 Shared VPU

Cortex-A520AE Core shares a VPU between all Cortex-A520AE 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.

4.13 AES encryption / decryption

Cortex-A520AE Core 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