Arm® Cortex®-A55 Software Optimization Guide

Version 2.0

Cortex-A55

Software Optimization Guide

Copyright © 2017, 2018 Arm Limited (or its affiliates). All rights reserved.

Release Information

Document History

Version	Date	Confidentiality	Change
1.0	30 June 2017	Confidential	First release
2.0	26 January 2018	Non-Confidential	Editorial changes and change in confidentiality status

Non-Confidential Proprietary Notice

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

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

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

This document may include technical inaccuracies or typographical errors.

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

This document consists solely of commercial items. You shall be responsible for ensuring that any use, duplication or disclosure of this document complies fully with any relevant export laws and regulations to assure that this document or any portion thereof is not exported, directly or indirectly, in violation of such export laws. Use of the word "partner" in reference to Arm's customers is not intended to create or refer to any partnership relationship with any other company. Arm may make changes to this document at any time and without notice.

If any of the provisions contained in these terms conflict with any of the provisions of any click through or signed written agreement covering this document with Arm, then the click through or signed written agreement prevails over and supersedes the conflicting provisions of these terms. This document may be translated into other languages for convenience, and you agree that if there is any conflict between the English version of this document and any translation, the terms of the English version of the Agreement shall prevail.

The Arm corporate logo and words marked with ® or ™ are registered trademarks or trademarks of Arm Limited (or its subsidiaries) in the US and/or elsewhere. All rights reserved. Other brands and names mentioned in this document may be the trademarks of their respective owners. Please follow Arm's trademark usage guidelines at **http://www.arm.com/company/policies/trademarks**.

Copyright © 2017 Arm Limited (or its affiliates). All rights reserved.

Arm Limited. Company 02557590 registered in England.

110 Fulbourn Road, Cambridge, England CB1 9NJ.

LES-PRE-20349

Confidentiality Status

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

Unrestricted Access is an Arm internal classification.

Product Status

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

Web Address

http://www.arm.com

Contents

1 About this document 6
1.1. References 6
1.2. Terms and Abbreviations 6
1.3. Scope 6

2 Introduction 7

3 Pipeline 8
3.1. Overview 8
3.1.1 Forwarding paths 9
3.2. Dual-issue 9
3.3. Load/store and address generation 11
3.4. Integer divide and multiply-accumulate units 12
3.5. Floating-point and NEON instructions 12
3.5.1 Instructions with out-of-order completion 12
3.5.2 Cryptographic instructions 13

4 Instruction characteristics 14
4.1. Instruction tables 14
4.2. Branch instructions 14
4.3. Arithmetic and logical instructions 15
4.4. Move and shift instructions 15
4.5. Divide and multiply instructions 16
4.6. Saturating and parallel arithmetic instructions 17
4.7. Miscellaneous Data-processing instructions 18
4.8. Load instructions 18
4.9. Store instructions 22
4.10. Atomic instructions 23
4.11. Floating-point data processing instructions 25
4.12. Floating-point miscellaneous instructions 26
4.13. Floating-point load instructions 27
4.14. Floating-point store instructions 28
4.15. Advanced SIMD integer instructions 29
4.16. Advanced SIMD floating-point instructions 32
4.17. Advanced SIMD miscellaneous instructions 34
4.18. Advanced SIMD load instructions 36
4.19. Advanced SIMD store instructions 38

Copyright © 2017, 2018 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

4.20. Cryptographic Extension 40
4.21. CRC 41

5 General 42
5.1. Support for three outstanding loads 42
5.2. Automatic hardware-based prefetch 42
5.3. Software load prefetch performance 42
5.4. Non-temporal loads 42
5.5. Cache line size 43
5.6. Atomics 43
5.7. Similar instruction performance 43
5.8. MemCopy performance 43
5.9. Conditional execution 44
5.10. A64 low latency pointer forwarding 44
5.11. Flag-transfer cost 44

Copyright © 2017, 2018 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

1 About this document

This document contains a guide to the Cortex-A55 micro-architecture with a view to aiding software optimization.

1.1. References

Reference	Document	Author	Title
1	DDI 0487	Arm	*Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile*
2	100442	Arm	*Arm® Cortex®-A55 Core Technical Reference Manual*

1.2. Terms and Abbreviations

This document uses the following terms and abbreviations.

Term	Meaning
AGU	Address Generation Unit
ALU	Arithmetic and Logical Unit
ASIMD	Advanced SIMD
DIV	Divide
MAC	Multiply-Accumulate
SQRT	Square Root
T32	AArch32 Thumb® instruction set
FP	Floating-point

1.3. Scope

This document describes aspects of the Cortex-A55 micro-architecture that influence software performance. Micro-architectural detail is limited to that which is useful for software optimization.

Documentation extends only to software visible behavior of the Cortex-A55 core and not to the hardware rationale behind the behavior.

Copyright © 2017, 2018 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

2 Introduction

The Cortex-A55 core is a mid-range, low-power core that implements the Armv8-A architecture with support for the Armv8.1-A extension, the Armv8.2-A extension, the RAS extension, the Load acquire (LDAPR) instructions introduced in the Armv8.3-A extension, and the Dot Product instructions introduced in the Armv8.4-A extension.

All pipelines within the Cortex-A55 core have been designed to be optimal with both the AArch32 and AArch64 instruction sets. There is no bias towards one or other instruction set.

This document describes elements of the Cortex-A55 micro-architecture that influence software performance so that software and compilers can be optimized accordingly.

Copyright © 2017, 2018 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

3 Pipeline

3.1. Overview

The Cortex-A55 pipeline is 8-stages deep for integer instructions and 10-stages deep for *floating-point* (FP) and *Advanced SIMD* (ASIMD) instructions.

The Advanced SIMD architecture, its associated implementations, and supporting software, are also referred to as NEON™ technology.

The following figure shows the structure of the datapath.

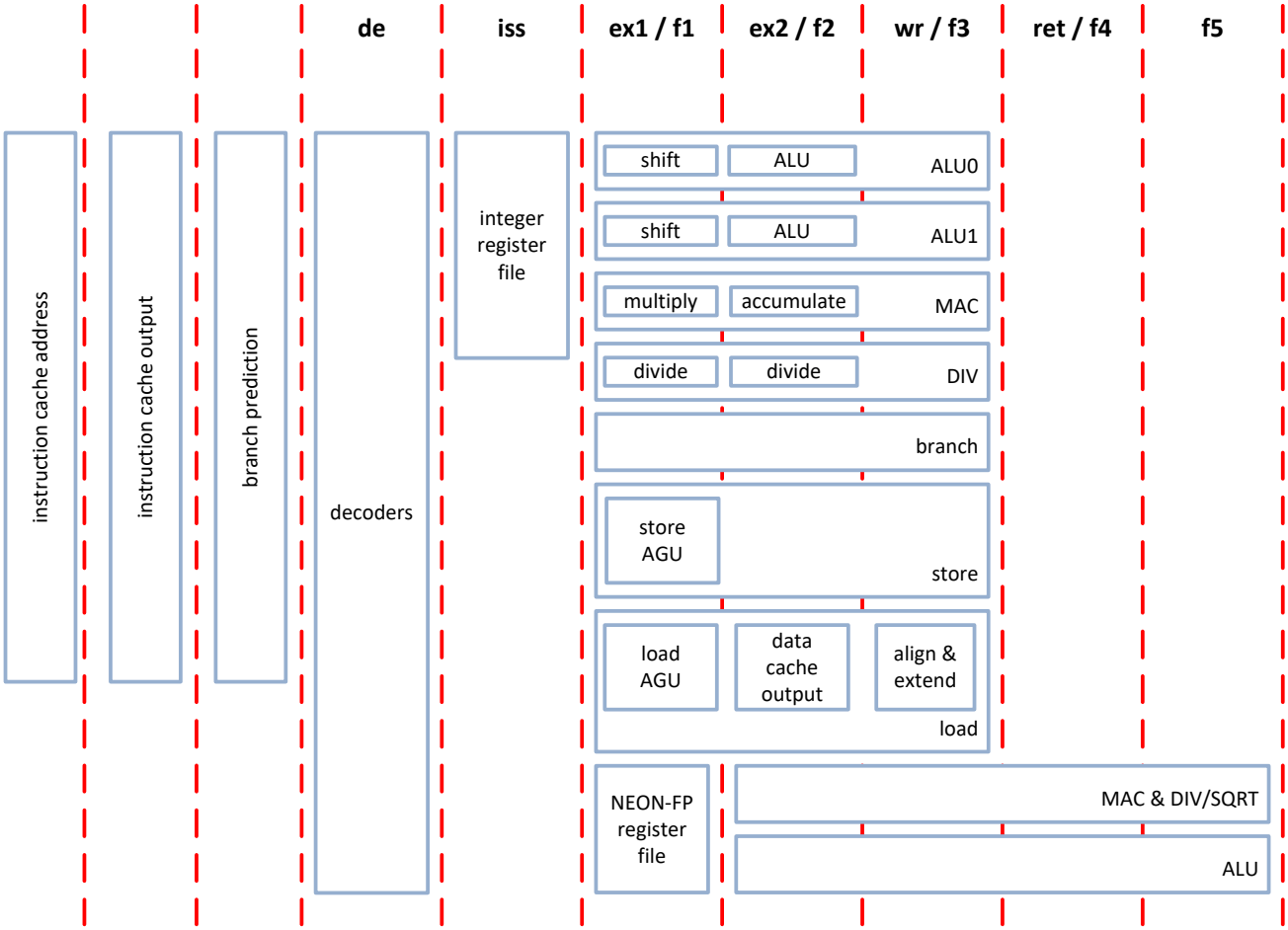

Figure 1 Cortex-A55 pipeline

The pipeline stages in the main datapath are *iss*, *ex1*, *ex2*, *wr*, and *ret*.

The pipeline stages in the NEON-FP datapath are *f1*, *f2*, *f3*, *f4*, and *f5*.

Integer instructions are issued in-order from the *iss* (issue) pipeline stage and complete in the *wr* (writeback) pipeline stage.

Floating-point or NEON instructions are issued from the *f5* pipeline stage.

Copyright © 2017, 2018 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Pipeline stages cannot be skipped. In the case of the branch and store pipelines there are still pipeline stages for *ex1* and *ex2* to ensure that instructions using those pipelines complete in-order.

3.1.1 Forwarding paths

Forwarding paths are implemented between almost all integer pipeline stages where operands can be consumed.

For example:

- The *Arithmetic and Logical Unit* (ALU) pipeline can forward results from the end of *ex1* and *ex2* to earlier stages of both ALU pipelines, and to the *iss* stage of the *Divide* (DIV) and *Multiply-Accumulate* (MAC) pipelines and the load/store *Address Generation Units* (AGUs). There is also a dedicated forwarding path from the *ex1* stage of the ALU0 to the *ex1* stage of the ALU1 and from the *ex2* stage of the ALU0 to the *ex2* stage of the store pipeline.
- The DIV pipeline can only accept operands in the *iss* stage and will only forward results from the *wr* stage.
- The MAC pipeline can only accept multiply operands in the *iss* stage and the accumulate operand in the *ex2* stage. There is a dedicated forwarding path from the *wr* stage to the *ex2* stage for accumulator forwarding within the MAC pipeline. Multiply and MAC results can be forwarded from the *wr* stage.
- Load-Store instructions require their address operands at the *iss* stage. There is a dedicated forwarding path to forward the address result back to the AGU base operand. There is also a dedicated forwarding path to support pointer-chasing of a load data at *wr* to AGU base operand at *ex1*.
- With the exception of system register read results, all integer results can be forwarded from the *wr* stage.

Forwarding does not contain bubbles so if a result can be forwarded from the end of the *ex1* stage it can also be forwarded from the *ex2* and *wr* stages. Similarly, if the latest a result can be consumed in *ex2*, it can also be consumed in *ex1* or *iss* if the result is available earlier.

Forwarding in the FP-NEON pipelines is more complex and dependent on whether the instruction is passing through the MAC & DIV/SQRT pipeline or the ALU pipeline.

3.2. Dual-issue

The Cortex-A55 core dual-issues under most circumstances. An outline of the rules required to achieve dual-issue are described in the following two tables. In these tables instruction-0 is the instruction that would otherwise be single-issued (also known as the older instruction) and instruction-1 (also known as the younger instruction) can only dual-issue if instruction-0 also supports dual-issue.

Instruction groups	Instructions	Notes
Data-processing	All integer data-processing instructions (including flag setting instructions) can be dual issued except: • Instructions that want to write to the program counter. • Special cases detailed in sections 4.3 - 4.7.	-
Load/store	All load/store instructions can be dual issued except: • Instructions that want to write to the program counter. • Load/store multiple instructions. • 32-bit load double/pair instructions. • Atomic compare swap pair. • Special cases detailed in sections 4.8 - 4.10.	-

Copyright © 2017, 2018 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Instruction groups	Instructions	Notes
Floating-point	All floating-point instructions can be dual issued except: • All load/store multiple instructions. • All load/store double instructions. • VMRS/VMSR instructions. • Special cases detailed in sections 4.11 - 4.14.	-
Advanced SIMD	All Advanced SIMD instructions can be dual issued except: • Load/store instructions of more than one cycle. • Certain multi-cycle data-processing instructions. • Special cases detailed in sections 4.15 - 4.19.	-
Branches	Most branches can be dual-issued from this position except indirect branches.	-
Miscellaneous	Control instructions cannot be dual issued. These include MRC/MCR, MRS/MSR, WFI, WFE, CPS, and barriers. In particular, IT cannot be dual issued from this position.	-

Table 1 Instruction-0 dual issue conditions

Instruction groups	Instructions	Notes
Data-processing	All data-processing instructions (including flag setting instructions) can be dual-issued except: • Instructions that want to write to the program counter. • Divide instructions. • Special cases detailed in sections 4.3 - 4.7.	Flag setting and Non-flag setting supported
Load/store	All load/store instructions can be dual issued except: • Instructions that want to write to the program counter. • Load/store multiple instructions. • Load double/pair instructions. • Special cases detailed in sections 4.8 - 4.10.	Providing there is not a structural hazard (loads cannot be dual issued with loads, and stores cannot be dual issued with stores)
Floating-point	Most floating-point instructions can be dual-issued from this position except: • Special cases detailed in sections 4.11 - 4.14.	-
Advanced SIMD	Most data-processing Advanced SIMD instructions can be dual-issued from this position except: • Special cases detailed in sections 4.15 - 4.19.	-
Branches	Most branches can be dual-issued from this position except compare and branch instructions.	Providing there is not a structural hazard (branches cannot be dual issued with branches)

Copyright © 2017, 2018 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Instruction groups	Instructions	Notes
Conditional	Conditional (flag-dependent) instructions can be dual issued with a flag setting instruction-0 except: • Instructions that execute an RRX operation. • Arithmetic with carry instructions. • Instruction-0 is MULS/MLAS. Instruction-0 is a NEON instruction.	-
Miscellaneous	IT, Architectural NOP.	-

Table 2 Instruction-1 dual issue conditions

3.3. Load/store and address generation

The Cortex-A55 load/store pipeline supports reads of up to 64 bits wide and writes of up to 128 bits wide. Providing the memory address is aligned, this allows instructions such as A32/T32 LDRD, STRD and A64 STP to be issued in a single cycle and occupy only one stage as the instruction passes through the pipeline. STM instructions in A32/T32 can only consume a maximum of 64 bits of the store pipeline and not the full 128-bit width.

The alignment requirements for load/store instructions to avoid a performance penalty are:

- 8-bit loads: Never a penalty cycle.
- 16-bit, 32-bit loads: Address must not cross a 64-bit boundary.
- 64-bit, 128-bit loads: Address must be 64-bit aligned.
- 8-bit stores: Never a penalty cycle.
- 16-bit, 32-bit, 64-bit stores: Address must not cross a 128-bit boundary.
- 128-bit store: Address must be 128-bit aligned.

If the memory address is not aligned, then providing the instruction passes its alignment checks a penalty cycle is incurred. If the next immediate cycle is a load or store operation, it will take a penalty cycle even if its access is aligned. According to this:

- For the 64-bit LDP instruction which splits into two load operations, it will take 2 cycles penalty if it is not 64-bit aligned.
- LDM/STM instructions that are 32-bit aligned, but not 64-bit aligned can take one more cycle if the number of registers are less than three or two more cycles otherwise to complete than LDM/STM instructions that are 64-bit aligned. For example, an LDM of 8-registers will take four cycles to complete if the address is 64-bit aligned and six cycles if the address is not 64-bit aligned.

The load-use latency from the data of a load instruction to the ALU of a dependent datapath instruction is two cycles. This means that in a back-to-back LDR-ADD sequence the ADD instruction would stall for one cycle.

The first stage (*ex1*) of both the load and store pipeline contains an AGU. To lower the latency on pointer chasing operations to 2-cycle, load data from a limited set of load instructions can be forwarded from the beginning of the *wr* pipeline stage to either the load or store AGU base operand. In general, this is limited to load instructions that do not require sign/zero extension, but more detail is provided in the following table.

Load instruction	Limitation
LDR & LDRT/LDTR (all variants)	32-bit aligned addresses only or 64-bit aligned addresses if 64-bit load (little endian)
LDRD/LDP (all variants)	Only the first register of the pair if 32-bit load or the second register of the pair if 64-bit load (little endian)

Copyright © 2017, 2018 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Load instruction	Limitation
LDM (all variants)	Only the first register of the last transfer (little endian)

Table 3 AGU pointer chasing

Load instructions that do not have a low-latency path in to the AGUs for pointer chasing incur an extra cycle penalty.

Finally, while the Cortex-A55 AGUs can calculate the address of all A64 and T32 load/store instructions in a single cycle, the performance on some rarely used A32 load/store instructions is compromised. Namely, A32 instructions that require the offset register to be subtracted from the base register to calculate the address or require the offset register to be shifted by a value other than 0, 1, 2, or 3 (as supported by T32) take a two-cycle penalty while the offset operand is formatted. Since these instructions are rare, it is unlikely that any performance impact will be noticed.

3.4. Integer divide and multiply-accumulate units

The Cortex-A55 core contains an integer divide unit for executing the UDIV and SDIV instructions. Integer divide instructions are serializing and do not allow younger instructions to retire underneath to ensure that the integer divide results is retired in-order. The divide iteration will terminate as soon as the result has been calculated.

The MAC unit in the Cortex-A55 core can sustain one 32-bit x 32-bit multiply or MAC operation per-cycle. 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.

Flag setting multiply operations do not take any longer than non-flag setting multiply operations. However, if there is a flag setting MUL immediately ahead of a conditional instruction then a single interlock cycle is forced to ensure that there is a cycle to move the flags from the multiply pipeline to the flag testing logic.

The latency for integer divide and multiply instructions are:

- In AArch32:
  - All multiplies take one cycle, but the UMAAL instruction takes two cycles due to the extra accumulation.
  - Divides take up to 12 cycles.
- In AArch64:
  - 32-bit multiplies take one cycle.
  - 64-bit multiplies take two or three cycles, depending on whether both operands contain ‘1’s in their top 32 bits.
  - SMULH/UMULH takes four cycles.
  - 32-bit divides take up to 12 cycles.
  - 64-bit divides take up to 20 cycles.

3.5. Floating-point and NEON instructions

3.5.1 Instructions with out-of-order completion

While the Cortex-A55 core only issues instructions in-order, due to the number of cycles required to complete more complex floating-point and NEON instructions, out-of-order retire is allowed on the instructions described in this section. The nature of the Cortex-A55 microarchitecture is such that NEON and floating-point instructions of the same type have the same timing characteristics.

Copyright © 2017, 2018 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

The out-of-order instructions are detailed in the following table.

Instructions (FP or NEON)	FP/ASIMD (half-precision)		FP/ASIMD (single-precision)		FP/ASIMD (double-precision)	
	Hazard	Latency	Hazard	Latency	Hazard	Latency
VDIV/FDIV	5[ac]	8[c]	10[ac]	13[c]	19[ac]	22[ac]
VSQRT/FSQRT	5[ac]	8[c]	9[ac]	12[c]	19[ac]	22[ac]
VMLA/VNMLA, VMLS/VNMLS, VRECPS, VRSQRTS	1	8 (4[b])	1	8 (4[b])	1	8 (4[b])
VFMA/FMADD, VFNMA/FNMADD, VFMS/FMSUB, VFNMS/FNMSUB	1	4	1	4	1	4

Table 4 Out-of-order FP/NEON instruction characteristics

The following information describes how to decode the information in the table:

- *Hazard (structural):* The number of cycles that the datapath resource is unavailable to another instruction that wants to use it. For example, a VDIV instruction after a VSRQT instruction must wait for the datapath resource to free up. But a VMUL instruction behind a VSQRT instruction is not blocked. A value of 1-cycle indicates that the resource will not block and supports single cycle back-to-back operation.
  - a Indicates the hazard is applicable to instructions that wish to use the divide or square-root resource.
- *Latency:* The number of cycles between when the operands are required and the result is available for forwarding.
  - b Indicates the number of cycles between multiply-accumulate instructions if the only dependency is the accumulate operand. In other words, the accumulate forwarding latency.
- c Indicates that the number quoted is for normal inputs. Each denormal input operand adds an additional hazard and latency cycle.

3.5.2 Cryptographic instructions

All cryptographic instructions issue in a single cycle. To optimize the AES algorithm, dependent pairs of AES/AESMC or AESD/AESIMC can be dual-issued, if they meet the following template:

```
AESE Vn, _
AESMC Vn, Vn
```

Copyright © 2017, 2018 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

4 Instruction characteristics

4.1. Instruction tables

This chapter describes high-level performance characteristics for most Armv8-A A32, T32, and A64 instructions. A series of tables summarize the effective execution latency and throughput (instruction bandwidth per cycle), pipelines utilized, and special behaviors associated with each group of instructions. Dual-issue corresponds to the execution pipelines described in chapter 3.

In the following tables:

- *Exec latency*, unless otherwise specified, is defined as the minimum latency seen by an operation dependent on an instruction in the described group.
- *Execution throughput* is defined as the maximum throughput (in instructions per cycle) of the specified instruction group that can be achieved in the entirety of the Cortex-A55 microarchitecture.
- *Dual-issue* is interpreted as:
  - `00` not dual-issuable.
  - `01` dual-issuable from slot 0.
  - `10` dual-issuable from slot 1.
  - `11` dual-issuable from both slots.

4.2. Branch instructions

Instruction group	AArch32 instructions	Exec latency	Execution throughput	Dual-issue	Notes
Branch, immed	`B`	`1`	`1`	`11`	`-`
Branch, register	`BX`	`1`	`1`	`10`	`-`
Branch and link, immed	`BL, BLX`	`1`	`1`	`11`	`-`
Branch and link, register	`BLX`	`1`	`1`	`10`	`-`
Compare and branch	`CBZ, CBNZ`	`1`	`1`	`01`	`-`

Instruction group	AArch64 instructions	Exec latency	Execution throughput	Dual-issue	Notes
Branch, immed	`B`	`1`	`1`	`11`	`-`
Branch, register	`BR, RET`	`1`	`1`	`10`	`-`
Branch and link, immed	`BL`	`1`	`1`	`11`	`-`
Branch and link, register	`BLR`	`1`	`1`	`10`	`-`
Compare and branch	`CBZ, CBNZ, TBZ, TBNZ`	`1`	`1`	`01`	`-`

Copyright © 2017, 2018 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

4.3. Arithmetic and logical instructions

Instruction group	AArch32 instructions	Exec latency	Execution throughput	Dual-issue	Notes
ALU, basic, include flag setting	ADD{S}, ADC{S}, ADR, AND{S}, BIC{S}, CMN, CMP, EOR{S}, ORN{S}, ORR{S}, RSB{S}, RSC{S}, SUB{S}, SBC{S}, TEQ, TST	1	2	11	-
ALU, shift by immed	(same as above)	2	2	11	-
ALU, shift by register	(same as above)	2	1	11	-
ALU, branch forms	-	8	1/8	00	1

Notes:

1. Branch form of ALU instructions always causes a flush when retired.

Instruction group	AArch64 instructions	Exec latency	Execution throughput	Dual-issue	Notes
ALU, basic, include flag setting	ADD{S}, ADC{S}, AND{S}, BIC{S}, EON, EOR, ORN, ORR, SUB{S}, SBC{S}	1	2	11	-
ALU, extend and/or shift	ADD{S}, AND{S}, BIC{S}, EON, EOR, ORN, ORR, SUB{S}	2	2	11	-
ALU, Conditional compare	CCMN, CCMP	1	2	11	-
ALU, Conditional select	CSEL, CSINC, CSINV, CSNEG	1	2	11	-

4.4. Move and shift instructions

Instruction group	AArch32 instructions	Exec latency	Execution throughput	Dual-issue	Notes
Move, basic	MOV{S}, MOVW, MOVT, MVN{S}	1	2	11	-
Move, shift by immed	ASR{S}, LSL{S}, LSR{S}, ROR{S}, RRX{S}	1	2	11	-
MVN, shift by immed	MVN{S}	2	2	11	-
Move, shift by register	ASR{S}, LSL{S}, LSR{S}, ROR{S}	1	1	11	-
MVN, shift by register	MVN	2	1	11	-
(Move, branch forms)	-	8	1/8	00	1

Copyright © 2017, 2018 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Instruction group	AArch64 instructions	Exec latency	Execution throughput	Dual-issue	Notes
Address generation	ADR, ADRP	1	2	11	-
Move immed	MOVN, MOVK, MOVZ	1	2	11	-
Variable shift	ASRV, LSLV, LSRV, RORV	1	1	11	-

Notes:

1. Branch form of ALU instructions always causes a flush when retired.

4.5. Divide and multiply instructions

Instruction group	AArch32 instructions	Exec latency	Execution throughput	Dual-issue	Notes
Divide	SDIV, UDIV	3 - 12 (11)	1/12 (11) - 1/3	01	1
Multiply	MUL{S}, SMULBB, SMULBT, SMULTB, SMULTT, SMULWB, SMULWT, SMMUL{R}, SMUAD{X}, SMUSD{X}	3	1	11	-
Multiply accumulate	MLA, MLS, SMLABB, SMLABT, SMLATB, SMLATT, SMLAWB, SMLAWT, SMLAD{X}, SMLSD{X}, SMMLA{R}, SMMLS{R}	3 (1)	1	11	2
Multiply accumulate long	SMLAL, SMLALBB, SMLALBT, SMLALTB, SMLALTT, SMLALD{X}, SMLSLD{X}, UMLAL	3 (1)	1	00	2
Multiply Accumulate Accumulate Long	UMAAL	4 (2)	1/2	01	2
Multiply long	SMULL, UMULL	3	1	11	-

Instruction group	AArch64 instructions	Exec latency	Execution throughput	Dual-issue	Notes
Divide, W-form	SDIV, UDIV	3 - 12 (11)	1/12 (11) - 1/3	01	1
Divide, X-form	SDIV, UDIV	3 - 20 (19)	1/20 (19) - 1/3	01	1
Multiply accumulate (32-bit)	MADD, MSUB	3 (1)	1	11	2
Multiply accumulate (64-bit)	MADD, MSUB	4-5 (2-3)	1/3 – 1/2	11	2

Copyright © 2017, 2018 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Instruction group	AArch64 instructions	Exec latency	Execution throughput	Dual-issue	Notes
Multiply accumulate long	SMADDL, SMSUBL, UMADDL, UMSUBL	3 (1)	1	11	2
Multiply high	SMULH, UMULH	6	1/3	11	-

Notes:

1. Integer divides are performed using an iterative algorithm and block any subsequent divide operations until complete. Early termination is possible, depending upon the data values. Signed division takes one more cycle than unsigned division for non-zero division.
2. 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.

4.6. Saturating and parallel arithmetic instructions

Instruction group	AArch32 instructions	Exec latency	Execution throughput	Dual-issue	Notes
Parallel arith	SADD16, SADD8, SSUB16, SSUB8, UADD16, UADD8, USUB16, USUB8	1	2	11	-
Parallel arith with exchange	SASX, SSAX, UASX, USAX	1	2	11	-
Parallel halving arith	SHADD16, SHADD8, SHSUB16, SHSUB8, UHADD16, UHADD8, UHSUB16, UHSUB8	1	2	11	-
Parallel halving arith with exchange	SHASX, SHSAX, UHASX, UHSAX	1	2	11	-
Parallel saturating arith	QADD16, QADD8, QSUB16, QSUB8, UQADD16, UQADD8, UQSUB16, UQSUB8	2	2	11	-
Parallel saturating arith with exchange	QASX, QSAX, UQASX, UQSAX	2	2	11	-
Saturate, basic	SSAT, SSAT16, USAT, USAT16	1	2	11	-
Saturate, LSL by immed or ASR	SSAT, USAT	2	2	11	-
Saturating arith	QADD, QSUB	2	2	11	-
Saturating doubling arith	QDADD, QDSUB	3	2	11	-

Copyright © 2017, 2018 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

4.7. Miscellaneous Data-processing instructions

Instruction group	AArch32 instructions	Exec latency	Execution throughput	Dual-issue	Notes
Bit field extract	SBFX, UBFX	2	2	11	-
Bit field insert/clear	BFI, BFC	2	2	11	-
Count leading zeros	CLZ	1	2	11	-
Pack halfword	PKH	2	2	11	-
Reverse bits	RBIT	2	2	11	-
Reverse bytes	REV, REV16, REVSH	1	2	11	-
Select bytes, unconditional	SEL	1	2	01	-
Sign/zero extend	SXTB, SXTH, UXTB, UXTH, SXTB16, UXTB16	1	2	11	-
Sign/zero extend and add	SXTAB, SXTAH,UXTAB, UXTAH, SXTAB16, UXTAB16	2	1	11	-
Sum of absolute differences	USAD8, USADA8	3	1	11	-

Instruction group	AArch64 instructions	Exec latency	Execution throughput	Dual-issue	Notes
Bitfield extract	EXTR	2	2	11	-
Sign/zero extend	SXTB, UXTB	1	2	11	-
Bitfield move, basic	SBFM, UBFM	2	2	11	-
Bitfield move, insert	BFM	2	2	11	-
Count leading	CLS, CLZ	1	2	11	-
Reverse bits	RBIT	2	2	11	-
Reverse bytes	REV, REV16, REVSH	1	2	11	-

4.8. Load instructions

- The latencies shown assume the memory access hits in the *Level 1* (L1) data cache.
- Latencies correspond to “correctly” aligned accesses. There is one cycle penalty for unaligned loads that cross a 64-bit boundary.

Copyright © 2017, 2018 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Instruction group	AArch32 instructions	Exec latency	Execution throughput	Dual-issue	Notes
Load, immed offset	LDR{T}, LDRB{T}, LDRH{T}, LDRSB{T}, LDRSH{T}, LDRD	3 (2)	1	11*	1, 3
Load, register offset, plus, unscaled	LDR, LDRB, LDRH, LDRSB, LDRSH, LDRD	3 (2)	1	11*	1, 3
Load, register offset, plus, LSL imm < 4	LDR, LDRB	3 (2)	1	11	1
Load, register offset, plus, others	LDR, LDRB	5 (4)	1/3	01	1
Load, register offset, minus	LDR, LDRB, LDRH, LDRSB, LDRSH, LDRD	5 (4)	1/3	01	1
Load, immed pre-indexed	LDR, LDRB, LDRH, LDRSB, LDRSH, LDRD	3 (2), 1	1	11*	1, 2, 3
Load, register pre-indexed, plus, unscaled	LDR, LDRB, LDRH, LDRSB, LDRSH, LDRD	3 (2), 1	1	11*	1, 2, 3
Load, register pre-indexed, plus, LSL imm < 4	LDR, LDRB	3 (2), 1	1	11	1, 2
Load, register pre-indexed, plus, others	LDR, LDRB	5 (4), 3	1/3	01	1, 2
Load, register pre-indexed, minus	LDR, LDRB, LDRH, LDRSB, LDRSH, LDRD	5 (4), 3	1/3	01*	1, 2, 3
Load, immed post-indexed	LDR{T}, LDRB{T}, LDRH{T}, LDRSB{T}, LDRSH{T}, LDRD	3 (2), 1	1	11*	1, 2, 3
Load, register post-indexed, unscaled	LDR{T}, LDRB{T}, LDRH{T}, LDRSB{T}, LDRSH{T}, LDRD	3 (2), 1	1	11*	1, 2, 3
Load, register post-indexed, scaled	LDR{T}, LDRB{T}	3 (2), 1	1	11	1, 2
Preload, immed	PLD, PLDW, PLI	1	1	11	-

Copyright © 2017, 2018 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Instruction group	AArch32 instructions	Exec latency	Execution throughput	Dual-issue	Notes
Preload, register offset, plus, unscaled or LSL imm < 4	PLD, PLDW, PLI	1	1	11	-
Preload, register offset, plus, others	PLD, PLDW, PLI	3	1/3	01	-
Preload, register offset, minus	PLD, PLDW, PLI	3	1/3	01	-
Load acquire	LDA, LDAB, LDAH	3 (2)	1	01	1
Load acquire exclusive	LDAEX, LDAEXB, LDAEXH	3 (2)	1	01	1
Load acquire exclusive, doubleword	LDAEXD	3 (2)	1	00	1
Load multiple, no writeback	LDMIA, LDMIB, LDMDA, LDMDB	3 (2) + N -1	1/N	00	1, 4
Load multiple, writeback	LDMIA, LDMIB, LDMDA, LDMDB, POP	3 (2) + N -1, N	1/N	00	1, 2, 4
Load multiple, branch forms	LDMIA, LDMIB, LDMDA, LDMDB, POP	8 + N-1	1/(8+N-1)	00	5
Load, branch forms with addressing mode as register offset, pre-indexed, minus or plus scaled and not LSL imm < 4	-	10	1/10	00	5
(Load, branch forms)	-	8	1/8	00	5

Copyright © 2017, 2018 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Instruction group	AArch64 instructions	Exec latency	Execution throughput	Dual-issue	Notes
Load register, literal	LDR, LDRSW	3 (2)	1	11	1
Load register, unscaled immed	LDUR, LDURB, LDURH, LDURSB, LDURSH, LDURSW	3 (2)	1	11	1
Load register, immed, pre/post-indexed	LDR, LDRB, LDRH, LDRSB, LDRSH, LDRSW	3 (2), 1	1	11	1, 2
Load register, immed unprivileged	LDTR, LDTRB, LDTRH, LDTRSB, LDTRSH, LDTRSW	3 (2)	1	11	1
Load register, unsigned immed	LDR, LDRB, LDRH, LDRSB, LDRSH, LDRSW	3 (2)	1	11	1
Load register, register offset	LDR, LDRB, LDRH, LDRSB, LDRSH, LDRSW	3 (2)	1	11	1
Preload	PRFM	1	1	11	-
Load acquire	LDAR, LDARB, LDARH, LDLAR, LDLARB, LDLARH	3 (2)	1	01	1
Load acquire exclusive	LDAXR, LDAXRB, LDAXRH	3 (2)	1	01	1
Load acquire exclusive, pair	LDAXP	3(2)	1	00	1
Load pair, W-form, immed offset, normal	LDP, LDNP	3 (2)	1	00	1
Load pair, X-form, immed offset, normal	LDP, LDNP	4 (3)	1/2	01	1
Load pair, signed words	LDPSW	3	1	00	-
Load pair, W-form, immed pre/post-index, normal	LDP	3 (2), 1	1	00	1, 2
Load pair, X-form, immed pre/post-index, normal	LDP	4 (3), 2	1/2	00	1, 2

Notes:

1. A fast forward path from load data to address (pointer chasing) can be activated in some cases (short latency show in parentheses). See section 3.3.
2. Base register updates are typically completed in parallel with the load operation and with shorter latency (update latency shown after the comma).
3. LDRD is only single issued.
4. For load multiple instructions, `N=floor((num_regs+1)/2)`.
5. Branch form of the load instructions always causes a flush when retired.

Copyright © 2017, 2018 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

4.9. Store instructions

The following table describes performance characteristics for standard store instructions. Stores may issue to L1 at *iss* once their address operands are available and do not need to wait for data operands (which are required at *wr*). Once executed, stores are buffered and committed in the background.

Instruction group	AArch32 instructions	Exec latency	Execution throughput (T)	Dual-issue	Notes
Store, immed offset	STR{T}, STRB{T}, STRD, STRH{T}	1	1	11	-
Store, register offset, plus, unscaled	STR, STRB, STRD, STRH	1	1	11	1
Store, register offset, minus	STR, STRB, STRD, STRH	3	1/3	01	1
Store, register offset, plus, LSL imm < 4	STR, STRB	1	1	11	-
Store, register offset, plus, other	STR, STRB	3	1/3	01	-
Store, immed pre-indexed	STR, STRB, STRD, STRH	1	1	11	-
Store, register pre-indexed, plus, unscaled	STR, STRB, STRD, STRH	1	1	11	1
Store, register pre-indexed, minus	STR, STRB, STRD, STRH	3	1/3	01	1
Store, register pre-indexed, plus, LSL imm < 4	STR, STRB	1	1	11	-
Store, register pre-indexed, plus, other	STR, STRB	3	1/3	01	-
Store, immed post-indexed	STR{T}, STRB{T}, STRH{T}	1	1	11	-
Store dual, register post-indexed	STRD	1	1	00	-
Store, register post-indexed	STR{T}, STRB{T}, STRH	1	1	11	-
Store release	STL, STLB, STLH	2	1/2	01	-
Store release exclusive	STLEX, STLEXB, STLEXH, STLEXD	4	1/2	01	-
Store multiple	STMIA, STMIB, STMDA, STMDB	N	1/N	11	2
(Store, writeback form)	-	(1/T)	Same as before	-	3

Copyright © 2017, 2018 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Instruction group	AArch64 instructions	Exec latency	Execution throughput (T)	Dual-issue	Notes
Store register, unscaled immed	STUR, STURB, STURH	1	1	11	-
Store register, immed pre/post-index	STR, STRB, STRH	1	1	11	-
Store register, immed unprivileged	STTR, STTRB, STTRH	1	1	11	-
Store register, unsigned immed	STR, STRB, STRH	1	1	11	-
Store register, register offset	STR, STRB, STRH	1	1	11	-
Store release	STLR, STLRB, STLRH, STLLR, STLLRB, STLLRH	2	1/2	01	-
Store release exclusive	STLXR, STLXRB, STLXRH, STLXP	4	1/2	01	-
Store pair, immed, all addressing modes	STP, STNP	1	1	11	-
(Store, writeback form)	-	(1/T)	Same as before		3

Notes:

1. STRD (register) is only single-issued.
2. For store multiple instructions, N=floor((num_regs+1)/2).
3. Writeback forms of store instructions require an extra operation to update the base address. This update is typically performed in parallel with the store operation (update latency shown in parentheses).

4.10. Atomic instructions

Instruction group	AArch32/AArch64 instructions	Exec latency	Execution throughput	Dual-issue	Notes
LD<OP>, without release semantics	LDADD{A}, LDADD{A}B, LDADD{A}H, LDCLR{A}, LDCLR{A}B, LDCLR{A}H, LDEOR{A}, LDEOR{A}B, LDEOR{A}H, LDSET{A}, LDSET{A}B, LDSET{A}H, LDSMAX{A}, LDSMAX{A}B, LDSMAX{A}H, LDSMIN{A}, LDSMIN{A}B, LDSMIN{A}H, LDUMAX{A}, LDUMAX{A}B, LDUMAX{A}H, LDUMIN{A}, LDUMIN{A}B, LDUMIN{A}H	2	1/2	01	-

Copyright © 2017, 2018 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Instruction group	AArch32/AArch64 instructions	Exec latency	Execution throughput	Dual-issue	Notes
LD<OP>, with release semantics	LDADDL, LDADDLB, LDADDLH, LDCLRL, LDCLRLB, LDCLRLH, LDEORL, LDEORLB, LDEORLH, LDSETL, LDSETLB, LDSETLH, LDSMAXL, LDSMAXLB, LDSMAXLH, LDSMINL, LDSMINLB, LDSMINLH, LDUMAXL, LDUMAXLB, LDUMAXLH, LDUMINL, LDUMINLB, LDUMINLH, LDADDAL, LDADDALB, LDADDALH, LDCLRAL, LDCLRALB, LDCLRALH, LDEORAL, LDEORALB, LDEORALH, LDSETAL, LDSETALB, LDSETALH, LDSMAXAL, LDSMAXALB, LDSMAXALH, LDSMINAL, LDSMINALB, LDSMINALH, LDUMAXAL, LDUMAXALB, LDUMAXALH, LDUMINAL, LDUMINALB, LDUMINALH	3	1/3	01	-
ST<OP>, without release semantics	STADD{A}, STADD{A}B, STADD{A}H, STCLR{A}, STCLR{A}B, STCLR{A}H, STEOR{A}, STEOR{A}B, STEOR{A}H, STSET{A}, STSET{A}B, STSET{A}H, STSMAX{A}, STSMAX{A}B, STSMAX{A}H, STSMIN{A}, STSMIN{A}B, STSMIN{A}H, STUMAX{A}, STUMAX{A}B, STUMAX{A}H, STUMIN{A}, STUMIN{A}B, STUMIN{A}H	1	1	01	-
ST<OP>, with release semantics	STADDL, STADDLB, STADDLH, STCLRL, STCLRLB, STCLRLH, STEORL, STEORLB, STEORLH, STSETL, STSETLB, STSETLH, STSMAXL, STSMAXLB, STSMAXLH, STSMINL, STSMINLB, STSMINLH, STUMAXL, STUMAXLB, STUMAXLH, STUMINL, STUMINLB, STUMINLH, STADDAL, STADDALB, STADDALH, STCLRAL, STCLRALB, STCLRALH, STEORAL, STEORALB, STEORALH, STSETAL, STSETALB, STSETALH, STSMAXAL, STSMAXALB, STSMAXALH, STSMINAL, STSMINALB, STSMINALH, STUMAXAL, STUMAXALB, STUMAXALH, STUMINAL, STUMINALB, STUMINALH	2	1/2	01	-
Compare and swap, without release semantics	CAS{A}, CAS{A}B, CAS{A}H	4	1/4	01	-
Compare and swap, with release semantics	CASL, CASLB, CASLH, CASAL, CASALB, CASALH	3	1/3	01	-
Compare and swap, pair, without release semantics	CASP{A}	5	1/5	00	-
Compare and swap, pair, with release semantics	CASPL, CASPAL	4	1/4	00	-

Copyright © 2017, 2018 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Instruction group	AArch32/AArch64 instructions	Exec latency	Execution throughput	Dual-issue	Notes
Swap, without release semantics	SWP{A}, SWP{A}B, SWP{A}H	2	1/2	01	-
Swap, with release semantics	SWPL, SWPLB, SWPLH, SWPAL, SWPALB, SWPALH	3	1/3	01	-

4.11. Floating-point data processing instructions

Instruction group	AArch32 instructions	Exec latency	Execution throughput	Dual-issue	Notes
FP absolute value	VABS	4	2	11	-
FP arith	VADD, VSUB	4	2	11	-
FP compare	VCMP, VCMPE	1	1	11	1
FP compare and write flags	VCMP,VCMPE followed by VMRS APSR_nzcv, FPSCR	2	1	11	2
FP convert	VCVT{R}, VCVTB, VCVTT, VCVTA, VCVTM, VCVTN, VCVTP	4	2	11	-
FP round to integral	VRINTA, VRINTM, VRINTN, VRINTP, VRINTR, VRINTX, VRINTZ	4	2	11	-
FP divide, H-form	VDIV	8	1/5	01	3
FP divide, S-form	VDIV	13	1/10	01	3
FP divide, D-form	VDIV	22	1/19	01	3
FP max/min	VMAXNM, VMINNM	4	2	11	-
FP multiply	VMUL, VNMUL	4	2	11	-
FP multiply accumulate	VMLA, VMLS, VNMLA, VNMLS	8 (4)	2	11	3
FP multiply accumulate	VFMA VFNMA VFMS VFNMS	4	2	11	3
FP negate	VNEG	4	2	11	-
FP select	VSELEQ, VSELGE, VSELGT, VSELVS	2	2	01	-
FP square root, H-form	VSQRT	8	1/5	01	3
FP square root, S-form	VSQRT	12	1/9	01	3
FP square root, D-form	VSQRT	22	1/19	01	3

Instruction group	AArch64 instructions	Exec latency	Execution throughput	Dual-issue	Notes
FP absolute value	FABS	4	2	11	-
FP arithmetic	FADD, FSUB	4	2	11	-

Copyright © 2017, 2018 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Instruction group	AArch64 instructions	Exec latency	Execution throughput	Dual-issue	Notes
FP compare	FCCMP{E}, FCMP{E}	1	1	11	-
FP divide, H-form	FDIV	8	1/5	01	3
FP divide, S-form	FDIV	13	1/10	01	3
FP divide, D-form	FDIV	22	1/19	01	3
FP min/max	FMIN, FMINNM, FMAX, FMAXNM	4	2	11	-
FP multiply	FMUL, FNMUL	4	2	11	-
FP multiply accumulate	FMADD, FMSUB, FNMADD, FNMSUB	4	2	11	-
FP negate	FNEG	4	2	11	-
FP round to integral	FRINTA, FRINTI, FRINTM, FRINTN, FRINTP, FRINTX, FRINTZ	4	2	11	-
FP select	FCSEL	2	1	01	-
FP square root, H-form	FSQRT	8	1/5	01	3
FP square root, S-form	FSQRT	12	1/9	01	3
FP square root, D-form	FSQRT	22	1/19	01	3

Notes:

1. The latency corresponds to FPSCR flags forward to a VMRS APSR_nzcv, FPSCR instruction.
2. The latency corresponds to the sequence FCMP, VMRS APSR_nzcv, FPSCR to a conditional instruction.
3. Refer to section 3.5.1 for details.

4.12. Floating-point miscellaneous instructions

Instruction group	AArch32 instructions	Exec latency	Execution throughput	Dual-issue	Notes
FP move, immed	VMOV	1	2	11	-
FP move, register	VMOV	1	2	11	-
FP transfer, single to core	VMOV	2	2	11	1
FP transfer, two singles to core	VMOV	3	1	01	1
FP transfer, double/half to core	VMOV	3	2	11	1
FP transfer, core to half/single/double	VMOV	2	2	11	1
FP transfer, core to two singles	VMOV	2	1	01	1

Copyright © 2017, 2018 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Instruction group	AArch64 instructions	Exec latency	Execution throughput	Dual-issue	Notes
FP convert, from vec to vec reg	FCVT	4	2	11	-
FP convert, from vec to vec reg	FCVTXN	4	1	01	-
FP convert, from gen to vec reg	SCVTF, UCVTF	5	2	11	-
FP convert, from vec to gen reg	FCVTAS, FCVTAU, FCVTMS, FCVTMU, FCVTNS, FCVTNU, FCVTPS, FCVTPU, FCVTZS, FCVTZU	3	2	11	1
FP move, immed	FMOV	1	2	11	-
FP move, register	FMOV	1	2	11	-
FP transfer, from gen to half/double/single	FMOV	2	2	11	1
FP transfer, from double/single to gen reg	FMOV	3	2	11	1
FP transfer, from half to gen reg	FMOV	3	2	11	1

Notes:

1. Latency number refers to the worst-case latency from the result to a dependent instruction.

4.13. Floating-point load instructions

FP load data is available for forwarding from *f4*. The latency numbers shown indicate the worst-case load-use latency from the load data to a dependent instruction. Latencies assume the memory access hits in the L1 data cache. Latencies also assume that 64-bit element loads are aligned to 64-bit. If this is not the case, one extra cycle is required.

Instruction group	AArch32 instructions	Exec latency	Execution throughput	Dual-issue	Notes
FP load, register	VLDR	3	1	11	-
FP load multiple	VLDMIA, VLDMDB, VPOP	3 + N - 1	1/N	00	1
FP load multiple, writeback	VLDMIA, VLDMDB, VPOP	3 + N -1, N	1/N	00	1,2

Instruction group	AArch64 instructions	Exec latency	Execution throughput	Dual-issue	Notes
Load vector reg, literal, S/D-form	LDR	3	1	11	-
Load vector reg, literal, Q-form	LDR	4	1/2	01	-
Load vector reg, unscaled immed, B/H-form	LDUR	3	1	01	-
Load vector reg, unscaled immed, S/D-form	LDUR	3	1	11	-
Load vector reg, unscaled immed, Q-form	LDUR	4	1/2	01	-

Copyright © 2017, 2018 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Instruction group	AArch64 instructions	Exec latency	Execution throughput	Dual-issue	Notes
Load vector reg, immed pre/post-index, B/H-form	LDR	3, 1	1	01	2
Load vector reg, immed pre/post-index, S/D-form	LDR	3, 1	1	11	2
Load vector reg, immed pre/post-index, Q-form	LDR	4, 2	1/2	01	2
Load vector reg, unsigned immed / register offset, B/H-form	LDR	3	1	01	-
Load vector reg, unsigned immed / register offset, S/D-form	LDR	3	1	11	-
Load vector reg, unsigned immed / register offset, Q-form	LDR	4	1/2	01	-
Load vector pair, immed offset, S-form	LDP, LDNP	3	1	01	-
Load vector pair, immed offset, D-form	LDP, LDNP	4	1/2	01	-
Load vector pair, immed offset, Q-form	LDP, LDNP	6	1/4	01	-
Load vector pair, immed pre/post-index, S-form	LDP, LDNP	3, 1	1	01	2
Load vector pair, immed pre/post-index, D-form	LDP, LDNP	4, 2	1/2	01	2
Load vector pair, immed pre/post-index, Q-form	LDP, LDNP	6, 4	1/4	01	2

Notes:

1. For FP load multiple instructions:
   - `N=num_regs` for double-precision registers.
   - `N=floor((num_regs+1)/2)` for single-precision registers.
2. Writeback forms of load instructions require an extra operation to update the base address. This update is typically performed in parallel with or prior to the load operation (update latency shown after the comma).

4.14. Floating-point store instructions

Stores may issue to L1 at *iss* once their address operands are available and do not need to wait for data operands (which are required at *f2*). Once executed, stores are buffered and committed in the background.

Instruction group	AArch32 instructions	Exec latency	Execution throughput (T)	Dual-issue	Notes
FP store, immed offset	VSTR	1	1	11	-
FP store multiple	VSTMIA, VSTMDB, VPUSH	N	1/N	00	1
(FP store, writeback form)	-	(1/T)	Same as before	-	2

Copyright © 2017, 2018 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Instruction group	AArch64 instructions	Exec latency	Execution throughput (T)	Dual-issue	Notes
Store vector reg, unscaled immed	STUR	1	1	11	-
Store vector reg, immed	STR	1	1	11	-
Store vector reg, register offset	STR	1	1	11	-
Store vector pair, immed, S/D-form	STP	1	1	11	-
Store vector pair, immed, Q-form	STP	2	1/2	01	-
(FP store, writeback form)	-	(1/T)	Same as before		2

Notes:

1. For single-precision store multiple instructions, `N=floor((num_regs+1)/2)`.

   For double-precision store multiple instructions, `N=(num_regs)`.
2. Writeback forms of store instructions require an extra operation to update the base address. This update is typically performed in parallel with or prior to the store operation (address update latency shown in the parentheses).

4.15. Advanced SIMD integer instructions

Instruction group	AArch32 instructions	Exec latency	Execution throughput	Dual-issue	Notes
ASIMD absolute diff	VABD	3	2*	11*	1
ASIMD absolute diff accum	VABA, VABAL	4	1/2	01	-
ASIMD absolute diff long	VABDL	3	1	01	-
ASIMD arith	VADD, VHADD, VNEG, VSUB, VHSUB, VRHADD	2	2*	11*	1
ASIMD arith	VADDL, VADDW, VSUBL, VSUBW, VPADDL, VADDHN , VSUBHN	3	1	01	-
ASIMD arith	VABS, VQADD, VQNEG, VQSUB, VPADD	3	2*	11*	1
ASIMD arith	VQABS	4	2*	11*	1
ASIMD arith	VRADDHN, VRSUBHN	4	1/2	01	-
ASIMD compare	VCEQ, VCGE, VCGT, VCLE, VTST	2	2*	11*	1
ASIMD logical	VAND, VBIC, VMVN, VORR, VORN, VEOR	1	2*	11*	1
ASIMD max/min	VMAX, VMIN, VPMAX, VPMIN	2	2*	11*	1

Copyright © 2017, 2018 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Instruction group	AArch32 instructions	Exec latency	Execution throughput	Dual-issue	Notes
ASIMD multiply	VMUL, VQDMULH, VQRDMULH	4	2*	11*	1
ASIMD multiply, by scalar	VMUL, VQDMULH, VQRDMULH	4	1	01	-
ASIMD multiply accumulate	VMLA, VMLS	4 (1)	2*	11*	1, 2
ASIMD multiply accumulate, by scalar	VMLA, VMLS	4 (1)	1	01	2
ASIMD multiply accumulate half	VQRDMLAH, VQRDMLSH	4	1	01	-
ASIMD multiply accumulate long	VQDMLAL, VQDMLSL	4	1	01	-
ASIMD multiply accumulate long	VMLAL, VMLSL	4 (1)	1	01	2
ASIMD dot product	VUDOT, VSDOT	4 (1)	2*	11*	1,3
ASIMD dot product, by scalar	VUDOT, VSDOT	4 (1)	1	01	3
ASIMD multiply long, integer	VMULL, VQDMULL	4	1	01	-
ASIMD multiply long, polynomial	VMULL.P8	3	1	01	-
ASIMD pairwise add and accumulate	VPADAL	4	1/2	01	-
ASIMD shift accumulate	VSRA, VRSRA	3	2*	11*	1
ASIMD shift by immed	VMOVL, VSHLL	2	1	01	-
ASIMD shift by immed	VSHL, VSHR, VSHRN	2	2*	11*	1
ASIMD shift by immed	VQRSHRN, VQRSHRUN, VQSHL{U}, VQSHRN, VQSHRUN	4	2*	11*	1
ASIMD shift by immed	VRSHR, VRSHRN	3	2*	11*	1
ASIMD shift by immed and insert, basic	VSLI, VSRI	2	2*	11*	1
ASIMD shift by register	VSHL	2	2*	11*	1
ASIMD shift by register	VRSHL	3	2*	11*	1
ASIMD shift by register	VQRSHL, VQSHL	4	2*	11*	1

Instruction group	AArch64 instructions	Exec latency	Execution throughput	Dual-issue	Notes
ASIMD absolute diff	SABD, UABD	3	2*	11*	1
ASIMD absolute diff accum	SABA, UABA, SABAL(2), UABAL(2)	4	1/2	01	-
ASIMD absolute diff long	SABDL, UABDL	3	1	01	-
ASIMD arith	ADD, SUB, NEG, SHADD, UHADD, SHSUB, UHSUB, SRHADD, URHADD	2	2*	11*	1

Copyright © 2017, 2018 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Instruction group	AArch64 instructions	Exec latency	Execution throughput	Dual-issue	Notes
ASIMD arith	ABS, ADDP, SADDLP, UADDLP, SQADD, UQADD, SQNEG, SQSUB, UQSUB, SUQADD, USQADD	3	2*	11*	1
ASIMD arith	SADDL(2), UADDL(2), SADDW(2), UADDW(2), SSUBL(2), USUBL(2), SSUBW(2), USUBW(2), ADDHN(2), SUBHN(2),	3	1	01	-
ASIMD arith	SQABS	4	2*	11*	1
ASIMD arith	RADDHN(2), RSUBHN(2)	4	1/2	01	-
ASIMD arith, reduce	ADDV, SADDLV, UADDLV	3	1	01	-
ASIMD compare	CMEQ, CMGE, CMGT, CMHI, CMHS, CMLE, CMLT	2	2*	11*	1
ASIMD compare	CMTST	3	2*	11*	1
ASIMD logical	AND, BIC, EOR, MVN, ORN, ORR	1	2*	11*	1
ASIMD logical	MOV	2	2*	11*	1
ASIMD max/min, basic	SMAX, SMAXP, SMIN, SMINP, UMAX, UMAXP, UMIN, UMINP	2	2*	11*	1
ASIMD max/min, reduce	SMAXV, SMINV, UMAXV, UMINV	4	1	01	-
ASIMD multiply	MUL, SQDMULH, SQRDMULH	4	2*	11*	1
ASIMD multiply, by element	MUL, SQDMULH, SQRDMULH	4	1	01	-
ASIMD multiply	PMUL	3	2*	11*	1
ASIMD multiply accumulate	MLA, MLS	4 (1)	2*	11*	1,2
ASIMD multiply accumulate, by element	MLA, MLS	4 (1)	1	01	2
ASIMD multiply accumulate half	SQRDMLAH, SQRDMLSH	4	1	01	-
ASIMD multiply accumulate long	SMLAL(2), SMLSL(2), UMLAL(2), UMLSL(2)	4 (1)	1	01	2
ASIMD multiply accumulate long	SQDMLAL(2), SQDMLSL(2)	4	1	01	-
ASIMD dot product	UDOT, SDOT	4 (1)	2*	11*	1,3

Copyright © 2017, 2018 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Instruction group	AArch64 instructions	Exec latency	Execution throughput	Dual-issue	Notes
ASIMD dot product, by scalar	UDOT, SDOT	4 (1)	1	01	3
ASIMD multiply long	SMULL(2), UMULL(2), SQDMULL(2)	4	1	01	-
ASIMD polynomial (8x8) multiply long	PMULL.8B, PMULL2.16B	3	1	01	4
ASIMD pairwise add and accumulate	SADALP, UADALP	4	1/2	01	-
ASIMD shift accumulate	SRA, USRA	3	2*	11*	1
ASIMD shift accumulate	SRSRA, URSRA	4	1/2	01	-
ASIMD shift by immed	SHL, SHRN(2), SLI, SRI, SSHR, USHR	2	2	11	-
ASIMD shift by immed and insert	SLI, SRI	2	2*	11*	1
ASIMD shift by immed	SHLL(2), SSHLL(2), USHLL(2), SXTL(2), UXTL(2)	2	1	01	-
ASIMD shift by immed	RSHRN(2), SRSHR, URSHR, SQSHRN(2), UQSHRN(2)	3	2*	11*	1
ASIMD shift by immed	SQSHL{U}, UQSHL, SQRSHRN(2), UQRSHRN(2), SQRSHRUN(2), SQSHRUN(2)	4	2*	11*	1
ASIMD shift by register	SSHL, USHL	2	2*	11*	1
ASIMD shift by register	SRSHL, URSHL	3	2*	11*	1
ASIMD shift by register	SQSHL, UQSHL, SQRSHL, UQRSHL	4	2*	11*	1

Notes:

1. If the instruction has Q-form, the Q-form of the instruction can only be dual issued as instruction 0 and execution throughput is 1.
2. Multiply-accumulate pipelines support forwarding of accumulate operands from similar instructions, allowing a typical sequence of integer multiply-accumulate instructions to issue every cycle (accumulate latency shown in parentheses).
3. Multiply-accumulate pipelines support forwarding of accumulate operands between Dot Product instructions, allowing a sequence of Dot Product instructions to issue every cycle (accumulate latency shown in parentheses).
4. This category includes instructions of the form “PMULL Vd.8H, Vn.8B, Vm.8B” and “PMULL2 Vd.8H, Vn.16B, Vm.16B”.

4.16. Advanced SIMD floating-point instructions

Instruction group	AArch32 instructions	Exec latency	Execution throughput	Dual-issue	Notes
ASIMD FP arith	VABS, VABD, VADD, VPADD, VSUB	4	2*	11*	1

Copyright © 2017, 2018 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Instruction group	AArch32 instructions	Exec latency	Execution throughput	Dual-issue	Notes
ASIMD FP compare	VACGE, VACGT, VACLE, VACLT, VCEQ, VCGE, VCGT, VCLE, VCLT	2	2*	11*	1
ASIMD FP convert, integer	VCVT, VCVTA, VCVTM, VCVTN, VCVTP	4	2*	11*	1
ASIMD FP convert, fixed	VCVT	4	2*	11*	1
ASIMD FP convert, half-precision	VCVT	4	1	01	
ASIMD FP max/min	VMAX, VMIN, VPMAX, VPMIN, VMAXNM, VMINNM	4	2*	11*	1
ASIMD FP multiply	VMUL	4	2*	11*	1
ASIMD FP multiply, by scalar	VMUL	4	1	01	
ASIMD FP multiply accumulate	VMLA, VMLS	8 (4)	2*	11*	1,2
ASIMD FP multiply accumulate, by scalar	VMLA, VMLS	8 (4)	1	01	1,2
ASIMD FP multiply accumulate	VFMA, VFMS	4	2*	11*	1,2
ASIMD FP negate	VNEG	4	2*	11*	1
ASIMD FP round to integral	VRINTA, VRINTM, VRINTN, VRINTP, VRINTX, VRINTZ	4	2*	11*	1

Instruction group	AArch64 instructions	Exec latency	Execution throughput	Dual-issue	Notes
ASIMD FP arith	FABS, FABD, FADD, FSUB, FADDP	4	2*	11*	1
ASIMD FP compare	FACGE, FACGT, FCMEQ, FCMGE, FCMGT, FCMLE, FCMLT	2	2*	11*	1
ASIMD FP convert, long	FCVTL(2)	4	1	01	-
ASIMD FP convert, narrow	FCVTN(2), FCVTXN(2)	4	1	01	-
ASIMD FP convert, other	FCVTAS, VCVTAU, FCVTMS, FCVTMU, FCVTNS, FCVTNU, FCVTPS, FCVTPU, FCVTZS, FCVTZU, SCVTF, UCVTF	4	2*	11*	1
ASIMD FP divide, H-form	FDIV	8	1/5	01	2
ASIMD FP divide, S-form	FDIV	13	1/10	01	2
ASIMD FP divide, D-form	FDIV	22	1/19	01	2

Copyright © 2017, 2018 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Instruction group	AArch64 instructions	Exec latency	Execution throughput	Dual-issue	Notes
ASIMD FP max/min, normal	FMAX, FMAXNM, FMIN, FMINNM	4	2*	11*	1
ASIMD FP max/min, pairwise	FMAXP, FMAXNMP, FMINP, FMINNMP	4	2*	11*	1
ASIMD FP max/min, reduce	FMAXV, FMAXNMV, FMINV, FMINNMV	4	1	01	-
ASIMD FP multiply	FMUL, FMULX	4	2*	11*	1
ASIMD FP multiply, by element	FMUL, FMULX	4	1	01	
ASIMD FP multiply accumulate	FMLA, FMLS	4	2*	11*	1,2
ASIMD FP multiply accumulate, by element	FMLA, FMLS	4	1	01	-
ASIMD FP negate	FNEG	4	2*	11*	1
ASIMD FP round	FRINTA, FRINTI, FRINTM, FRINTN, FRINTP, FRINTX, FRINTZ	4	2*	11*	1

Notes:

1. If the instruction has Q-form, the Q-form of the instruction can only be dual issued as instruction 0 and execution throughput is 1.
2. Refer to section 3.5.1 for more details.

4.17. Advanced SIMD miscellaneous instructions

Instruction group	AArch32 instructions	Exec latency	Execution throughput	Dual-issue	Notes
ASIMD bitwise insert	VBIF, VBIT, VBSL	2	2*	11*	1
ASIMD count	VCLZ, VCNT	2	2*	11*	1
ASIMD count	VCLS	3	2*	11*	1
ASIMD duplicate, core reg	VDUP	3	2	11	3
ASIMD duplicate, scalar	VDUP	2	2*	11*	1
ASIMD extract	VEXT	2	2*	11*	1
ASIMD move, immed	VMOV	1	2*	11*	1
ASIMD move, register	VMOV	1	2*	11*	1
ASIMD move, narrowing	VMOVN	2	2	11	-
ASIMD move, extract/insert	VMOVX, VINS	2	2	11	-
ASIMD move, saturating	VQMOVN, VQMOVUN	4	2	11	-
ASIMD reciprocal estimate	VRECPE, VRSQRTE	4	2*	11*	1

Copyright © 2017, 2018 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Instruction group	AArch32 instructions	Exec latency	Execution throughput	Dual-issue	Notes
ASIMD reciprocal step	VRECPS, VRSQRTS	8 (4)	2*	11*	1,2
ASIMD reverse	VREV16, VREV32, VREV64	2	2*	11*	1
ASIMD swap, D-form	VSWP	2	1	01	-
ASIMD swap, Q-form	VSWP	2	1/2	01	-
ASIMD table lookup, 1/2 reg	VTBL, VTBX	2	1	01	-
ASIMD table lookup, 3/4 reg	VTBL, VTBX	3	1/2	01	-
ASIMD transfer, scalar to core reg	VMOV	3	2	11	3
ASIMD transfer, core reg to scalar	VMOV	2	2	11	3
ASIMD transpose	VTRN	3	1/2	01	-
ASIMD unzip	VUZP	3	1/2	01	-
ASIMD zip, D-form	VZIP	2	1	01	-
ASIMD zip, Q-form	VZIP	3	1/2	01	-

Instruction group	AArch64 instructions	Exec latency	Execution throughput	Dual-issue	Notes
ASIMD bit reverse	RBIT	2	2*	11*	1
ASIMD bitwise insert	BIF, BIT, BSL	2	2*	11*	1
ASIMD count	CLZ, CNT	2	2*	11*	1
ASIMD count	CLS	3	2*	11*	1
ASIMD duplicate, gen reg	DUP	3	2	11	3
ASIMD duplicate, element	DUP	2	2*	11*	1
ASIMD extract	EXT	2	2*	11*	1
ASIMD extract narrow	XTN	2	2	11	-
ASIMD extract narrow, saturating	SQXTN(2), SQXTUN(2), UQXTN(2)	4	2	11	-
ASIMD insert, element to element	INS	2	2	11	-
ASIMD move, integer immed	MOVI	1	2*	11*	1
ASIMD move, FP immed	FMOV	1	2	11	-
ASIMD reciprocal estimate	FRECPE, FRECPX, FRSQRTE, URECPE, URSQRTE	4	2*	11*	1
ASIMD reciprocal step	FRECPS, FRSQRTS	4	2*	11*	1
ASIMD reverse	REV16, REV32, REV64	2	2*	11*	1
ASIMD table lookup	TBL	2+N-1	1/N	01	2
ASIMD table lookup	TBX	2+N	1/(N+1)	01	2

Copyright © 2017, 2018 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Instruction group	AArch64 instructions	Exec latency	Execution throughput	Dual-issue	Notes
ASIMD transfer, element to gen reg	SMOV, UMOV	2	2	11	-
ASIMD transfer, gen reg to element	INS	2	2	11	3
ASIMD transpose, 64-bit (.2D)	TRN1, TRN2	2	1	01	-
ASIMD transpose, other	TRN1, TRN2	2	2*	11*	1
ASIMD unzip/zip	UZP1, UZP2, ZIP1, ZIP2	2	2*	11*	1

Notes:

1. If the instruction has Q-form, the Q-form of the instruction can only be dual issued as instruction 0 and execution throughput is 1.
2. For table branches (TBL and TBX), N denotes the number of registers in the table.
3. Latency number refers to the worst-case latency from the result to a dependent instruction.

4.18. Advanced SIMD load instructions

Advanced SIMD load data is available for forwarding from *f4*. The latency numbers shown indicate the worst-case load-use latency from the load data to a dependent instruction. The latencies shown assume the memory access hits in the L1 data cache.

Instruction group	AArch32 instructions	Exec latency	Execution throughput (T)	Dual-issue	Notes
ASIMD load, 1 element, multiple, 1 reg	VLD1	3	1	11	-
ASIMD load, 1 element, multiple, 2 reg	VLD1	4	1/2	01	-
ASIMD load, 1 element, multiple, 3 reg	VLD1	5	1/3	01	-
ASIMD load, 1 element, multiple, 4 reg	VLD1	6	1/4	01	-
ASIMD load, 1 element, one lane	VLD1	3	1	01	-
ASIMD load, 1 element, all lanes	VLD1	3	1	01	-
ASIMD load, 2 element, multiple, 2 reg	VLD2	4	1/2	01	-
ASIMD load, 2 element, multiple, 4 reg	VLD2	6	1/4	01	-
ASIMD load, 2 element, one lane	VLD2	3	1	01	-
ASIMD load, 2 element, all lanes	VLD2	3	1	01	-
ASIMD load, 3 element, multiple, 3 reg, size 8/16	VLD3	6	1/4	01	-
ASIMD load, 3 element, multiple, 3 reg, size 32	VLD3	5	1/3	01	-
ASIMD load, 3 element, one lane	VLD3	4	1/2	01	-
ASIMD load, 3 element, all lanes	VLD3	4	1/2	01	-

Copyright © 2017, 2018 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Instruction group	AArch32 instructions	Exec latency	Execution throughput (T)	Dual-issue	Notes
ASIMD load, 4 element, multiple, 4 reg, size 8/16	VLD4	7	1/5	01	-
ASIMD load, 4 element, multiple, 4 reg, size 32	VLD4	6	1/4	01	-
ASIMD load, 4 element, one lane	VLD4	4	1/2	01	-
ASIMD load, 4 element, all lanes	VLD4	4	1/2	01	-
(ASIMD load, writeback form)	-	(1/T)	Same as before	01	1

Instruction group	AArch64 instructions	Exec latency	Execution throughput (T)	Dual-issue	Notes
ASIMD load, 1 element, multiple, 1 reg, D-form	LD1	3	1	11	-
ASIMD load, 1 element, multiple, 1 reg, Q-form	LD1	4	1/2	01	-
ASIMD load, 1 element, multiple, 2 reg, D-form	LD1	4	1/2	01	-
ASIMD load, 1 element, multiple, 2 reg, Q-form	LD1	6	1/4	01	-
ASIMD load, 1 element, multiple, 3 reg, D-form	LD1	5	1/3	01	-
ASIMD load, 1 element, multiple, 3 reg, Q-form	LD1	8	1/6	01	-
ASIMD load, 1 element, multiple, 4 reg, D-form	LD1	6	1/4	01	-
ASIMD load, 1 element, multiple, 4 reg, Q-form	LD1	10	1/8	01	-
ASIMD load, 1 element, one lane	LD1	3	1	01	-
ASIMD load, 1 element, all lanes	LD1R	3	1	01	-
ASIMD load, 2 element, multiple, D-form	LD2	4	1/2	01	-
ASIMD load, 2 element, multiple, Q-form	LD2	6	1/4	01	-
ASIMD load, 2 element, one lane, B/H/S	LD2	3	1	01	-
ASIMD load, 2 element, one lane, D	LD2	4	1/2	01	-
ASIMD load, 2 element, all lanes, B/H/S	LD2R	3	1	01	-
ASIMD load, 2 element, all lanes, D	LD2R	4	1/2	01	-

Copyright © 2017, 2018 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Instruction group	AArch64 instructions	Exec latency	Execution throughput (T)	Dual-issue	Notes
ASIMD load, 3 element, multiple, D-form, B/H	LD3	6	1/4	01	-
ASIMD load, 3 element, multiple, Q-form, B/H	LD3	9	1/7	01	-
ASIMD load, 3 element, multiple, D-form, S	LD3	5	1/3	01	-
ASIMD load, 3 element, multiple, Q-form, S/D	LD3	8	1/6	01	-
ASIMD load, 3 element, one lane, B/H/S	LD3	4	1/2	01	-
ASIMD load, 3 element, one lane, D	LD3	5	1/3	01	-
ASIMD load, 3 element, all lanes, B/H/S	LD3R	4	1/2	01	-
ASIMD load, 3 element, all lanes, D	LD3R	5	1/3	01	-
ASIMD load, 4 element, multiple, D-form, B/H	LD4	7	1/5	01	-
ASIMD load, 4 element, multiple, Q-form, B/H	LD4	11	1/9	01	-
ASIMD load, 4 element, multiple, D-form, S	LD4	6	1/4	01	-
ASIMD load, 4 element, multiple, Q-form, S/D	LD4	10	1/8	01	-
ASIMD load, 4 element, one lane, B/H/S	LD4	4	1/2	01	-
ASIMD load, 4 element, one lane, D	LD4	6	1/4	01	-
ASIMD load, 4 element, all lanes, B/H/S	LD4R	4	1/2	01	-
ASIMD load, 4 element, all lanes, D	LD4R	6	1/4	01	-
(ASIMD load, writeback form)	-	(1/T)	Same as before	01	1

Notes:

1. Base register updates are typically completed in parallel with the load operation and with shorter latency.

4.19. Advanced SIMD store instructions

Store instructions may issue once their address operands are available and do not need to wait for data operands. Once executed, stores are buffered and committed in the background.

Instruction group	AArch32 instructions	Exec latency	Execution throughput (T)	Dual-issue	Notes
ASIMD store, 1 element, multiple, 1 reg	VST1	1	1	11	-
ASIMD store, 1 element, multiple, 2 reg	VST1	1	1	01	-

Copyright © 2017, 2018 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Instruction group	AArch32 instructions	Exec latency	Execution throughput (T)	Dual-issue	Notes
ASIMD store, 1 element, multiple, 3 reg	VST1	2	1/2	01	-
ASIMD store, 1 element, multiple, 4 reg	VST1	2	1/2	01	-
ASIMD store, 1 element, one lane	VST1	1	1	01	-
ASIMD store, 2 element, multiple, 2 reg	VST2	1	1	01	-
ASIMD store, 2 element, multiple, 4 reg	VST2	2	1/2	01	-
ASIMD store, 2 element, one lane	VST2	1	1	01	-
ASIMD store, 3 element, multiple, 3 reg	VST3	3	1/3	01	-
ASIMD store, 3 element, one lane	VST3	2	1/2	01	-
ASIMD store, 4 element, multiple, 4 reg	VST4	3	1/3	01	-
ASIMD store, 4 element, one lane	VST4	2	1/2	01	-
(ASIMD store, writeback form)	-	(1/T)	Same as before	-	1

Instruction group	AArch64 instructions	Exec latency	Execution throughput (T)	Dual-issue	Notes
ASIMD store, 1 element, multiple, 1 reg	ST1	1	1	11	-
ASIMD store, 1 element, multiple, 2 reg, D-form	ST1	1	1	01	-
ASIMD store, 1 element, multiple, 2 reg, Q-form	ST1	2	1/2	01	-
ASIMD store, 1 element, multiple, 3 reg, D-form	ST1	2	1/2	01	-
ASIMD store, 1 element, multiple, 3 reg, Q-form	ST1	3	1/3	01	-
ASIMD store, 1 element, multiple, 4 reg, D-form	ST1	2	1/2	01	-
ASIMD store, 1 element, multiple, 4 reg, Q-form	ST1	4	1/4	01	-
ASIMD store, 1 element, one lane	ST1	1	1	01	-
ASIMD store, 2 element, multiple, D-form	ST2	1	1	01	-
ASIMD store, 2 element, multiple, Q-form	ST2	2	1/2	01	-
ASIMD store, 2 element, one lane, B/H/S	ST2	1	1	01	-
ASIMD store, 2 element, one lane, D	ST2	2	1/2	01	-
ASIMD store, 3 element, multiple, B/H/S	ST3	3	1/3	01	-
ASIMD store, 3 element, multiple, D	ST3	3	1/3	01	-

Copyright © 2017, 2018 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Instruction group	AArch64 instructions	Exec latency	Execution throughput (T)	Dual-issue	Notes
ASIMD store, 3 element, one lane	ST3	2	1/2	01	-
ASIMD store, 4 element, multiple, D-form, B/H/S	ST4	3	1/3	01	-
ASIMD store, 4 element, multiple, Q-form, B/H/S	ST4	5	1/5	01	-
ASIMD store, 4 element, multiple, Q-form, D	ST4	4	1/4	01	-
ASIMD store, 4 element, one lane, B/H	ST4	2	1/2	01	-
(ASIMD store, writeback form)	-	(1/T)	Same as before	-	1

Notes:

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

4.20. Cryptographic Extension

Instruction group	AArch32 instructions	Exec latency	Execution throughput	Dual-issue	Notes
Crypto AES ops	AESD, AESE	2	1	01	1
Crypto AES ops	AESIMC, AESMC	2	1	11	-
Crypto polynomial (64x64) multiply long	VMULL.P64	2	1	01	-
Crypto SHA1 xor ops	SHA1SU0	2	1	01	-
Crypto SHA1 fast ops	SHA1H, SHA1SU1	2	1	01	-
Crypto SHA1 slow ops	SHA1C, SHA1M, SHA1P	4	1	01	-
Crypto SHA256 fast ops	SHA256SU0	3	1	01	-
Crypto SHA256 slow ops	SHA256H, SHA256H2, SHA256SU1	4	1	01	-

Instruction group	AArch64 instructions	Exec latency	Execution throughput	Dual-issue	Notes
Crypto AES ops	AESD, AESE	2	1	01	1
Crypto AES ops	AESIMC, AESMC	2	1	11	-
Crypto polynomial (64x64) multiply long	PMULL(2)	2	1	01	-
Crypto SHA1 xor ops	SHA1SU0	2	1	01	-
Crypto SHA1 schedule acceleration ops	SHA1H, SHA1SU1	2	1	01	-

Copyright © 2017, 2018 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Instruction group	AArch64 instructions	Exec latency	Execution throughput	Dual-issue	Notes
Crypto SHA1 hash acceleration ops	SHA1C, SHA1M, SHA1P	4	1	01	-
Crypto SHA256 schedule acceleration op	SHA256SU0	3	1	01	-
Crypto SHA256 schedule acceleration op	SHA256SU1	4	1	01	-
Crypto SHA256 hash acceleration ops	SHA256H, SHA256H2	4	1	01	-

Notes:

1. Adjacent AESE/AESMC instruction pairs and adjacent AESD/AESIMC instruction pairs can be dual-issued together.

4.21. CRC

Instruction group	AArch32 instructions	Exec latency	Execution throughput	Dual-issue	Notes
CRC checksum ops	CRC32B, CRC32H, CRC32CB, CRC32CH	2	2	11	-
CRC checksum ops	CRC32W, CRC32CW	1	2	11	-

Instruction group	AArch64 instructions	Exec latency	Execution throughput	Dual-issue	Notes
CRC checksum ops	CRC32B, CRC32H, CRC32CB, CRC32CH, CRC32X, CRC32CX	2	2	11	-
CRC checksum ops	CRC32W, CRC32CW	1	2	11	-

Copyright © 2017, 2018 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

5 General

This section covers aspects of the micro-architecture which are not related to the pipeline or branch prediction, but will improve performance if the software is optimized accordingly.

5.1. Support for three outstanding loads

While the Cortex-A55 core does not support any instruction reordering at issue or hit-under-miss, it can support three outstanding data cache misses. Providing that three load instructions are within four pipeline stages of each other, if the first load misses the data cache the second and third can also lookup and, if they both miss, generate a request to the *Level 2* (L2) cache or main memory.

5.2. Automatic hardware-based prefetch

The Cortex-A55 core has a data prefetch mechanism that looks for cache line fetches with regular patterns and automatically starts prefetching ahead. Prefetches will end once the pattern is broken, a `DSB` is executed, or a `WFI` or `WFE` is executed.

For read streams the prefetcher is based on virtual addresses and so can cross page boundaries provided that the new page is still cacheable and has read permission. Write streams are based on physical address and so cannot cross page boundaries, however if full cache line writes are being performed then the prefetcher will not activate and write streaming mode will be used instead.

The Cortex-A55 core is capable of tracking multiple streams in parallel.

For some types of pattern, once the prefetcher is confident in the stream it can start progressively increasing the prefetch distance ahead of the current accesses, and these accesses will start to allocate to the *Level 3* (L3) cache rather than L1. This allows better utilization of the larger resources available at L3, and also reduces the amount of pollution of the L1 cache if the stream ends or is incorrectly predicted. If the prefetching to L3 was accurate then the line will be removed from L3 and allocated to L1 when the stream reaches that address.

5.3. Software load prefetch performance

The Cortex-A55 core supports all load prefetching instructions (such as `PLD` and `PLI`). When executed load prefetches are non-blocking so they do not stall while the data is being fetched:

- Data fetched due to a `PLD` is placed in the L1 data cache.
- Data fetched by a `PLI` is placed in the L2 cache.
- Data fetched by a `PRFM` instruction is placed in the cache level encoded in the instruction.

On the Cortex-A55 core it is not advisable to use explicit load prefetch instructions if the access pattern falls within the capabilities of the hardware based prefetcher since load prefetch instructions consume an issue slot.

5.4. Non-temporal loads

The Cortex-A55 core supports the Non-temporal load and store instructions in the AArch64 instruction set. Non-temporal loads will allocate the line to the L1 cache as normal, but when the line is evicted from L1 if it is clean it will be discarded rather than allocating to L2. Non-temporal store instructions will update the cache if they hit, but will not cause an L1 allocation if they miss. They will allocate in L2 cache.

Any memory pages marked as transient in the page tables will behave as if all accesses to that page were Non-temporal.

Copyright © 2017, 2018 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

5.5. Cache line size

All caches in the Cortex-A55 core implement a 64-byte cache line.

5.6. Atomics

The Cortex-A55 core supports the atomic instructions added in Armv8.1-A of the Arm architecture. For atomics to cacheable memory, they can be performed as either near atomics or far atomics, depending on where the cache line containing the data resides. If the atomic hits in the L1 data cache in a unique state then it will be performed as a near atomic in the L1 memory system. If the atomic operation misses in the L1 cache, or the line is shared with another core then the atomic is sent as a far atomic out to the L3 cache. If the operation misses everywhere within the cluster, and the system interconnect supports far atomics, then the atomic will be passed on to the interconnect to perform the operation. If the operation hits anywhere inside the cluster, or the interconnect does not support atomics, then the L3 memory system will perform the atomic operation and allocated the line into the L3 cache if it is not already there. Therefore if software wants to ensure the atomic is performed as a near atomic then it should precede the atomic instruction with a `PLDW` or `PRFM PSTL1KEEP` instruction.

5.7. Similar instruction performance

Some instructions in the Armv8-A architecture can have similar or identical behavior as other instructions in the ISA. For example, an `LDM` of two registers is functionally the same as an `LDRD`. In the Cortex-A55 core, from an instruction timing perspective, there are no known cases where two similar instructions with the same result behave differently.

5.8. MemCopy performance

As the store pipeline width is 128 bits, the Cortex-A55 core will provide the highest performance if store instructions are used that can utilize the full width of this interface. In A64 the `STP` instruction can consume all 128 bits in a single-cycle and in A32/T32 only the `VSTM` instruction can consume all 128 bits in a single-cycle and only if the address is 128-bits aligned.

As the load datapath width is 64 bits, all load instructions that read 64 bits of data take a single cycle to issue if the address is 64-bit aligned and all load instructions that read 128-bits of data take two cycles to issue if the address is 64-bit aligned. Load multiple instructions such as `LDM` and `VLDM` which architecturally operate on 32-bit quantities can read 64-bits of data per-cycle providing the address is 64-bit aligned.

The Cortex-A55 core includes separate load and store pipelines, which allow it to execute a load and a store instruction in the same cycle.

To achieve maximum throughput for memory copy (or similar loops):

- Unroll the loop to include multiple load and store operations for each iteration, minimizing the overheads of looping.
- Use discrete, non-writeback forms of load and store instructions (such as `LDRD` and `STRD`), interleaving them so that one load and one store operation can be performed each cycle. Avoid load-multiple/store-multiple instruction encodings (such as `LDM` and `STM`), which lead to separated bursts of load and store operations which might not allow concurrent use of both the load and store pipelines.
- Separate the load and corresponding store instruction by at least one other instruction to avoid interlocks on the store source registers.

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

```
; x0 = destination pointer, aligned to 64 bytes
```

Copyright © 2017, 2018 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

```
; x1 = source pointer, aligned to 64 bytes
; x2 = number of bytes to copy, multiple of 64 bytes
      LDP  x3, x4, [x1,#0x0]
      LDP  x5, x6, [x1,#0x10]
      LDP  x7, x8, [x1,#0x20]
      LDP  x9, x10,[x1,#0x30]
      ADD  x0, x0, #0x30
      ADD  x1, x1, #0x70
loop_start:
      STP  x3, x4, [x0, #-0x30]
      SUBS x2, x2, #0x40
      LDP  x3, x4, [x1, #-0x30]
      STP  x5, x6, [x0, #-0x20]
      LDP  x5, x6, [x1, #-0x20]
      STP  x7, x8, [x0, #-0x10]
      LDP  x7, x8, [x1, #-0x10]
      STP  x9, x10, [x0], #0x40
      LDP  x9, x10, [x1], #0x40
      B.NE    loop_start
```

A recommended copy routine for the AArch32 state would look similar to the sequence above, but would use the LDRD/STRD instructions.

5.9. Conditional execution

With the exception of conditional MUL instructions (refer to section 3.4), conditional instructions have the same performance as non-conditional instructions.

5.10. 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, #lo12 <const>]
```

In the Cortex-A55 core there are dedicated forwarding paths that always allow this sequence to be executed without incurring a dependency based stall.

5.11. Flag-transfer cost

Flag transfer from the floating-point flags to the integer flags takes a single cycle. This prevents dual-issue of a VMRS instruction-0 with an integer flag testing instruction-1, but the integer flag testing instruction can be issued in the next cycle.

Copyright © 2017, 2018 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential