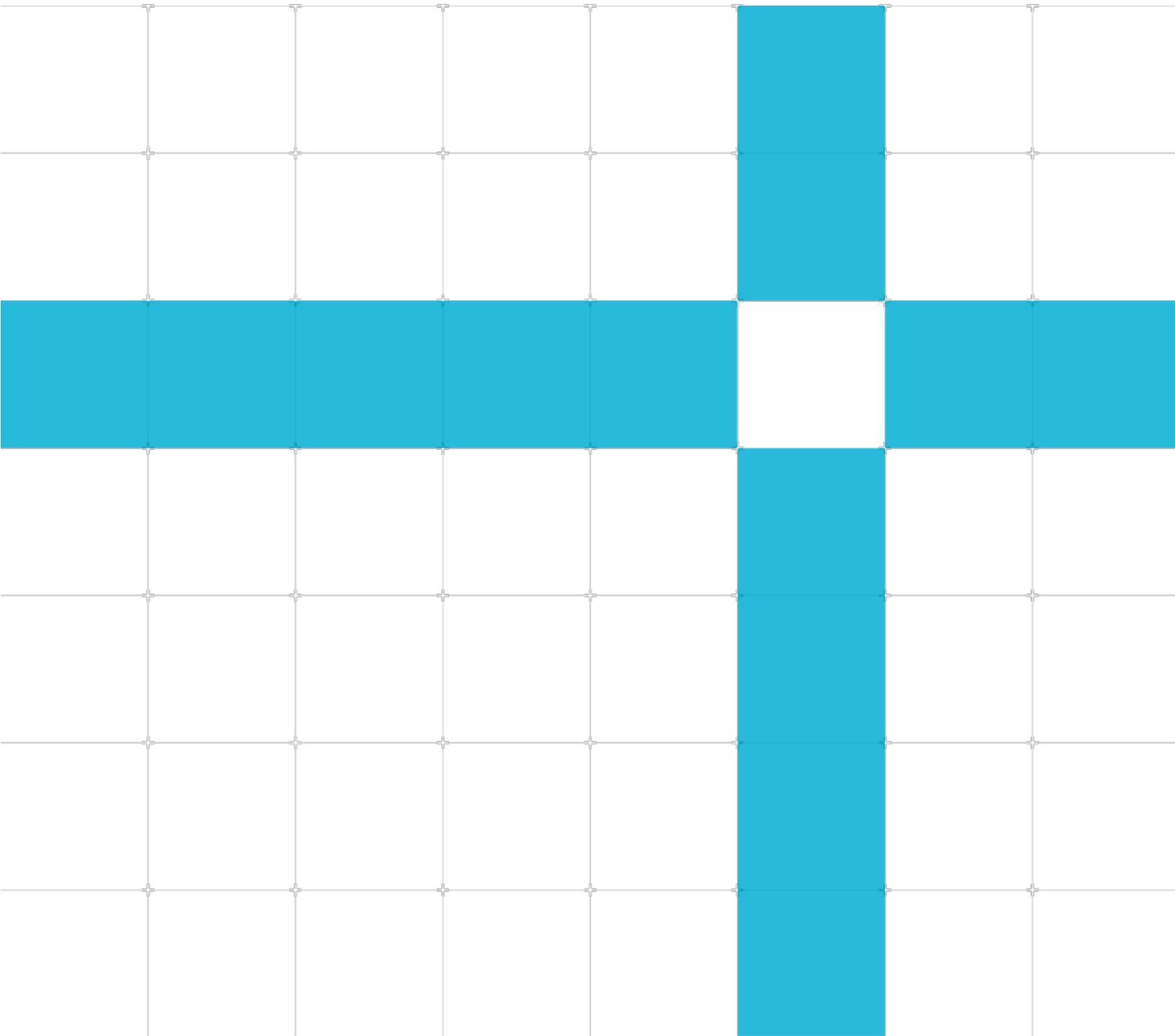
Arm® Cortex®-A65 Core

Revision: r1p1

Software Optimization Guide

Issue 1.0
PJDOC-466751330-10045



Arm® Cortex®-A65 Core

Software Optimization Guide

Release information

Document history

Issue	Date	Confidentiality	Change
1.0	20 Mar 2019	Non-Confidential	First release for r1p1

Non-Confidential Proprietary Notice

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.

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 Introduction **6**
1.1 Product revision status 6
1.2 Intended audience 6
1.3 Conventions 6
1.3.1 Glossary 6
1.3.2 Typographical conventions 7
1.4 Additional reading 7
1.5 Feedback 8
1.5.1 Feedback on this product 8
1.5.2 Feedback on content 8

2 About this document **9**
2.1 Scope 9
2.2 Product overview 9
2.2.1 Pipeline overview 9

3 Instruction characteristics **12**
3.1 Instruction tables 12
3.2 Legend for reading the utilized pipelines 12
3.3 Branch instructions 12
3.4 Arithmetic and logical instructions 13
3.5 Move and shift instructions 14
3.6 Divide and multiply instructions 14
3.7 Miscellaneous data-processing instructions 15
3.8 Load instructions 15
3.9 Store instructions 16
3.10 FP data processing instructions 17
3.11 FP miscellaneous instructions 18
3.12 FP load instructions 19
3.13 FP store instructions 20
3.14 ASIMD integer instructions 21
3.15 ASIMD floating-point instructions 24
3.16 ASIMD miscellaneous instructions 26
3.17 ASIMD load instructions 27

3.18 ASIMD store instructions 29
3.19 Cryptography extensions 31
3.20 CRC 32

4 Special considerations 33
4.1 SMT resource sharing 33
4.2 Optimizing memory copy 33
4.3 Load/Store alignment 34
4.4 Hardware data prefetch 34
4.5 Software prefetch performance 34
4.6 Non-temporal and transient memory hints 34
4.7 Branch instruction alignment 35
4.8 Instruction fusion 35

1 Introduction

1.1 Product revision status

The *rmpn* identifier indicates the revision status of the product described in this book, for example, r*1*p*2*, where:

r*m* Identifies the major revision of the product, for example, r*1*.

p*n* Identifies the minor revision or modification status of the product, for example, p*2*.

1.2 Intended audience

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

1.3 Conventions

The following subsections describe conventions used in Arm documents.

1.3.1 Glossary

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

See the Arm® Glossary for more information.

1.3.1.1 Terms and Abbreviations

This document uses the following terms and abbreviations.

Term	Meaning
A64	AAarch64 Instruction State
ALU	Arithmetic and Logical Unit
ASIMD	Advanced SIMD
µOP	Micro-Operation
FP	Floating-point

1.3.2 Typographical conventions

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

1.4 Additional reading

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

Table 1: Arm publications

Document name	Document ID	Licensee only Y/N
Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile	DDI 0487	N
Arm® Cortex®-A65 Core Technical Reference Manual	100439	N

1.5 Feedback

1.5.1 Feedback on this product

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

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

1.5.2 Feedback on content

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

- The title: Arm® Cortex®-A65 Core Software Optimization Guide.
- The number: PJDOC-TBD.
- If applicable, the page number(s) to which your comments refer.
- A concise explanation of your comments.

Arm also welcomes general suggestions for additions and improvements.

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

2 About this document

This document contains a guide to the Cortex-A65 core micro-architecture with a view to aiding software optimization.

2.1 Scope

This document describes aspects of the Cortex-A65 core micro-architecture that influence software performance. Micro-architectural detail is limited to that which is useful for software optimization.

This documentation extends only to software visible behavior of the Cortex-A65 core and not to the hardware rationale behind the behavior.

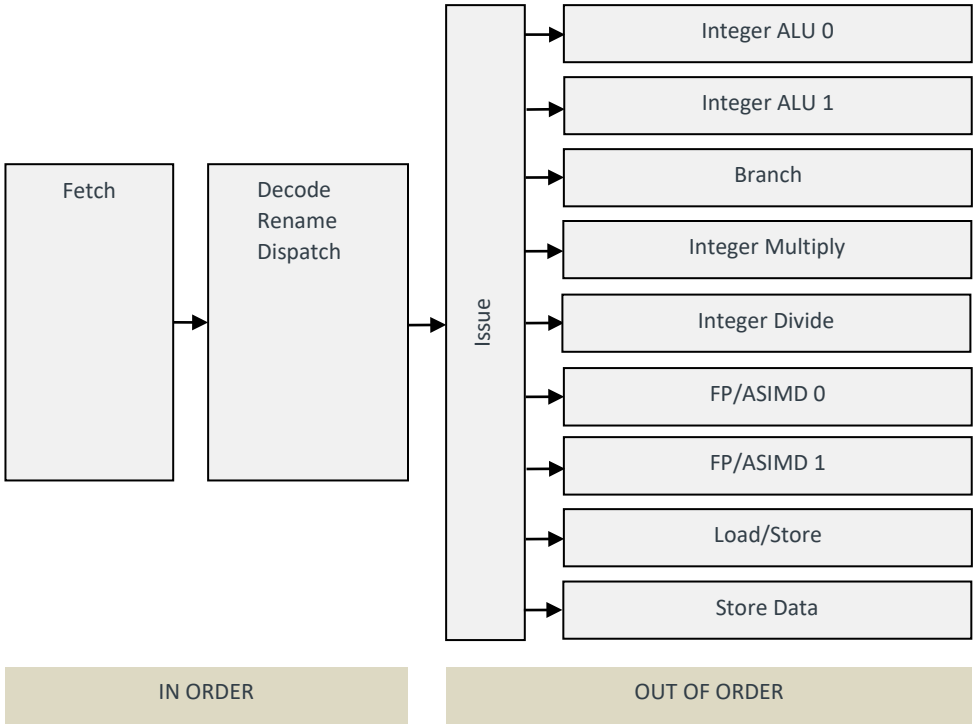
2.2 Product overview

The Cortex-A65 core is a power-efficient, mid-range core that implements the AArch64 execution state of the Armv8-A architecture. It supports the Armv8.2-A extension, including 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. Cortex-A65 is a Simultaneously Multi-Threaded (SMT) core with support for two threads of execution, each implementing a separate architectural Processing Element (PE).

2.2.1 Pipeline overview

The Cortex-A65 pipeline is 10 stages deep for integer instructions and 12 stages deep for floating-point and NEON instructions. The following figure illustrates the pipeline organization:

Figure 1: Cortex-A65 E1 core pipeline



Cortex-A65 fetches A64 instructions and decodes them, renames their register operands to allow out-of-order execution, and dispatches them to the issue stage. One or more corresponding Micro-OPerations (µOPs) per instruction are issued to one of nine execution pipelines once their operands are available.

The in-order portion of the Cortex-A65 pipeline can process a maximum of two instructions per cycle. Instructions requiring more than three source operands or more than three destination operands can only be processed one at a time.

Up to four µOPs can be issued in a cycle, constrained by type as follows:

- One may issue to the Load/Store pipeline.
- Two may issue to any of the remaining pipelines as follows:
    - Only one µOP total may issue to any of the Branch, Integer Multiply, Integer Divide, or Store Data pipelines.
    - Only Integer ALU and FP/ASIMD µOPs may issue two per cycle.
- Only Integer ALU µOPs associated with branch and link instructions or writeback forms of store instructions may issue as a fourth µOP, provided an Integer ALU pipe is available.

Instructions and µOps from either or both threads of execution may be processed throughout the front end and execution pipelines in any given CPU cycle. The threads dynamically share pipeline

bandwidth and related micro-architecture resources with policies designed to maximize aggregate compute throughput while maintaining fairness.

3 Instruction characteristics

3.1 Instruction tables

This chapter describes high-level performance characteristics for most Armv8 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.

In the tables below, Exec Latency 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-A65 microarchitecture.

Separate hardware threads running simultaneously on Cortex-A65 can increase latencies and reduce instruction bandwidths perceived by each thread, while also making opportunistic use of cycles which would generally be lost in a single-threaded CPU due to latencies and other inefficiencies preventing available execution bandwidth from being consumed. The Execution Latency and Execution Throughput figures that follow are respectively the realistic minimums and maximums encountered by instruction sequences from a thread regardless of the effects of SMT execution, and software scheduling will benefit from being optimized accordingly.

3.2 Legend for reading the utilized pipelines

Table 2: Cortex-A65 core pipeline names and symbols

Pipeline name	Symbol used in tables
Integer ALU 0/1	I
Branch	B
Integer multiply	M
Integer divide	D
FP/ASIMD 0/1	V
Load/Store	LS
Store data	SD

3.3 Branch instructions

Table 3: Branch instructions

Instruction Group	Instructions	Exec Latency	Execution Throughput	Utilized Pipelines	Notes

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

3.4 Arithmetic and logical instructions

Table 4: Arithmetic and logical instructions

Instruction Group	Instructions	Exec Latency	Execution Throughput	Utilized Pipelines	Notes
ALU, basic, includes flag setting	ADD{S}, ADC{S}, AND{S}, BIC{S}, EON, EOR, ORN, ORR, SUB{S}, SBC{S}	1	2	I	-
ALU, extend and/or shift	ADD{S}, AND{S}, BIC{S}, EON, EOR, ORN, ORR, SUB{S}	2 (1)	2	I	1
ALU, Conditional compare	CCMN, CCMP	1	1	I	-
ALU, Conditional select	CSEL, CSINC, CSINV, CSNEG	1	2	I	-

1. Late forwarding to the Xn/Wn operand allows a lower effective latency than for the shifted/extended Xm/Wm operand (shown in parentheses).

3.5 Move and shift instructions

Table 5: Move and shift instructions

Instruction Group	Instructions	Exec Latency	Execution Throughput	Utilized Pipelines	Notes
Address generation	ADR, ADRP	1	2	I	-
Move immed	MOVN, MOVK, MOVZ	1	2	I	-
Variable shift	ASRV, LSLV, LSRV, RORV	1	2	I	-

3.6 Divide and multiply instructions

Table 6: Divide and multiply instructions

Instruction Group	Instructions	Exec Latency	Execution Throughput	Utilized Pipelines	Notes
Signed divide, W-form	SDIV	5 to 11	1/14 to 1/8	D	1
Signed divide, X-form	SDIV	5 to 19	1/22 to 1/8	D	1
Unsigned divide, W-form	UDIV	5 to 10	1/13 to 1/8	D	1
Unsigned divide, X-form	UDIV	5 to 18	1/21 to 1/8	D	1
Multiply accumulate (32-bit)	MADD, MSUB	3 (2)	1	M	2
Multiply accumulate (64-bit)	MADD, MSUB	5 (4)	1/3	M	2
Multiply accumulate long	SMADDL, SMSUBL, UMADDL, UMSUBL	3 (2)	1	M	2
Multiply high	SMULH, UMULH	6	1/4	M	-

1. Integer divides are performed using an iterative algorithm and block any subsequent divide operations until complete. Early termination is possible, depending upon the data values.
2. Multiply-accumulate pipelines support late-forwarding of accumulate operands from similar µOPs, allowing a typical sequence of multiply-accumulate µOPs to issue one every N cycles (accumulate latency N shown in parentheses).

3.7 Miscellaneous data-processing instructions

Table 7: Miscellaneous data-processing instructions

Instruction Group	Instructions	Exec Latency	Execution Throughput	Utilized Pipelines	Notes
Bitfield extract	EXTR	2	2	I	-
Bitfield move, basic	SBFM, UBFM	2	2	I	-
Bitfield move, insert	BFM	2	2	I	-
Count leading	CLS, CLZ	1	2	I	-
Reverse bits	RBIT	2	2	I	-
Reverse bytes	REV, REV16, REV32	2	2	I	-

3.8 Load instructions

Table 8: Load instructions

Instruction Group	Instructions	Exec Latency	Execution Throughput	Utilized Pipelines	Notes
Load register, literal	LDR, LDRSW, PRFM	3 (2)	1	LS	1
Load register, unscaled immed	LDUR, LDURB, LDURH, LDURSB, LDURSH, LDURSW, PRFUM	3 (2)	1	LS	1
Load register, immed, pre-/post-indexed	LDR, LDRB, LDRH, LDRSB, LDRSH, LDRSW	3 (2)	1	LS	1
Load register, immed unprivileged	LDTR, LDTRB, LDTRH, LDTRSB, LDTRSH, LDTRSW	3 (2)	1	LS	1

Instruction Group	Instructions	Exec Latency	Execution Throughput	Utilized Pipelines	Notes
Load register, unsigned immed	LDR, LDRB, LDRH, LDRSB, LDRSH, LDRSW, PRFM	3 (2)	1	LS	1
Load register, register offset	LDR, LDRB, LDRH, LDRSB, LDRSH, LDRSW, PRFM	3 (2)	1	LS	1
Load pair, W-form, immed offset, normal	LDP, LDNP	3 (2)	1	LS	1
Load pair, X-form, immed offset, normal	LDP, LDNP	3 (2)	1	LS	1
Load pair, immed offset, signed words	LDPSW	3 (2)	1	LS	-
Load pair, W-form, immed pre/post-index, normal	LDP	3 (2)	1	LS	1
Load pair, X-form, immed pre/post-index, normal	LDP	3 (2)	1	LS	1
Load pair, immed pre/post-index, signed words	LDPSW	3 (2)	1	LS	1
(Load, writeback form)	-	(1)	-	+ I	2

1. Aligned doubleword-sized load data to dependent load and store instruction base address operands have reduced latency (shown in parentheses).
2. Writeback forms of load instructions require an extra µOP to update the base address. This update is typically performed in parallel with the store µOP (address update latency shown in parentheses).

3.9 Store instructions

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

Table 9: Store instructions

Instruction Group	Instructions	Exec Latency	Execution Throughput	Utilized Pipelines	Notes
Store register, unscaled immed	STUR, STURB, STURH	1	1	LS, SD	-
Store register, immed pre/post-index	STR, STRB, STRH	1	1	LS, SD	-
Store register, immed unprivileged	STTR, STTRB, STTRH	1	1	LS, SD	-
Store register, unsigned immed	STR, STRB, STRH	1	1	LS, SD	-
Store register, register offset	STR, STRB, STRH	1	1	LS, SD	-
Store pair, immed, all addressing modes	STP, STNP	1	1	LS, SD	-
(Store, writeback form)	-	(1)	-	+ I	1

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

3.10 FP data processing instructions

Table 10: FP data processing instructions

Instruction Group	Instructions	Exec Latency	Execution Throughput	Utilized Pipelines	Notes
FP absolute value	FABS	4	2	V	-
FP arithmetic	FADD, FSUB	4	2	V	-
FP compare	FCMP{E}	1	2	V, I	-
FP compare	FCCMP{E}	1	1	V, I	-
FP divide, H-form	FDIV	10	1/3	V	-
FP divide, S-form	FDIV	14	1/5	V	-
FP divide, D-form	FDIV	23	2/19	V	-
FP min/max	FMIN, FMINNM, FMAX, FMAXNM	4	2	V	-

Instruction Group	Instructions	Exec Latency	Execution Throughput	Utilized Pipelines	Notes
FP multiply	FMUL, FNMUL	4	2	V	-
FP multiply accumulate	FMADD, FMSUB, FNMADD, FNMSUB	4	2	V	-
FP negate	FNEG	4	2	V	-
FP round to integral	FRINTA, FRINTI, FRINTM, FRINTN, FRINTP, FRINTX, FRINTZ	4	2	V	-
FP select	FCSEL	2	2	V	-
FP square root, H-form	FSQRT	10	1/3	V	-
FP square root, S-form	FSQRT	13	2/9	V	-
FP square root, D-form	FSQRT	23	2/19	V	-

3.11 FP miscellaneous instructions

Table 11: FP miscellaneous instructions

Instruction Group	Instructions	Exec Latency	Execution Throughput	Utilized Pipelines	Notes
FP convert, from vec to vec reg	FCVT	4	2	V	-
FP convert, from vec to vec reg	FCVTXN	4	2	V	-
FP convert, from gen to vec reg	SCVTF, UCVTF	4	2	V, I	-

Instruction Group	Instructions	Exec Latency	Execution Throughput	Utilized Pipelines	Notes
FP convert, from vec to gen reg	FCVTAS, FCVTAU, FCVTMS, FCVTMU, FCVTNS, FCVTNU, FCVTPS, FCVTPU, FCVTZS, FCVTZU	3	2	V	-
FP move, immed	FMOV	2	2	V	-
FP move, register	FMOV	2	2	V	-
FP transfer, from gen reg to half/single/double	FMOV	2	2*	V, I	1
FP transfer, from half/single/double to gen reg	FMOV	3	2	V	-

1. For the 64-bit to top half of 128-bit variant of FMOV (FMOV <Vd>.D[1], <Xn>), Execution Throughput is half of the value shown.

3.12 FP load instructions

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

Table 12: FP load instructions

Instruction Group	Instructions	Exec Latency	Execution Throughput	Utilized Pipelines	Notes
Load vector reg, literal, S/D/Q-form	LDR	2	1	LS	-
Load vector reg, unscaled immed, B/H/S/D/Q-form	LDUR	2	1	LS	-
Load vector reg, immed pre/post-index, B/H/S/D/Q-form	LDR	2, 1	1	LS	-
Load vector reg, unsigned immed / register offset, B/H/S/D/Q-form	LDR	2	1	LS	-

Instruction Group	Instructions	Exec Latency	Execution Throughput	Utilized Pipelines	Notes
Load vector pair, immed offset, S/D-form	LDP, LDNP	2	1	LS	-
Load vector pair, immed offset, Q-form	LDP, LDNP	3	1/2	LS	-
Load vector pair, immed pre/post-index, S/D-form	LDP, LDNP	2, 1	1	LS	-
Load vector pair, immed pre/post-index, Q-form	LDP, LDNP	3, 1	1/2	LS	-
(FP load, writeback form)	-	(1)	-	+ I	1

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

3.13 FP store instructions

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

Table 13: FP store instructions

Instruction Group	Instructions	Exec Latency	Execution Throughput	Utilized Pipelines	Notes
Store vector reg, unscaled immed	STUR	1	1	LS, SD	-
Store vector reg, immed	STR	1	1	LS, SD	-
Store vector reg, register offset	STR	1	1	LS, SD	-
Store vector pair, immed, S/D-form	STP	1	1	LS, SD	-
Store vector pair, immed, Q-form	STP	2	1/2	LS, SD	-
(FP store, writeback form)	-	(1)	-	+ I	1

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

3.14 ASIMD integer instructions

Table 14: ASIMD integer instructions

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

Instruction Group	Instructions	Exec Latency	Execution Throughput	Utilized Pipelines	Notes
ASIMD compare	CMEQ, CMGE, CMGT, CMHI, CMHS, CMLE, CMLT	2	2*	V	1
ASIMD compare	CMTST	3	2*	V	1
ASIMD logical	AND, BIC, EOR, MVN, ORN, ORR	1	2*	V	1
ASIMD logical	MOV	2	2	V	-
ASIMD max/min, basic	SMAX, SMAXP, SMIN, SMINP, UMAX, UMAXP, UMIN, UMINP	2	2*	V	1
ASIMD max/min, reduce	SMAXV, SMINV, UMAXV, UMINV	4	1	V	-
ASIMD multiply	MUL, SQDMULH, SQRDMULH	4	2*	V	1
ASIMD multiply	PMUL	3	2*	V	1
ASIMD multiply accumulate	MLA, MLS	4	2*	V	1
ASIMD multiply accumulate half	SQRDMLAH, SQRDMLSH	4	2*	V	1
ASIMD multiply accumulate long	SMLAL(2), SMLSL(2), UMLAL(2), UMLSL(2)	4	1	V	-
ASIMD multiply accumulate long, vector	SQDMLAL(2), SQDMLSL(2)	4	1	V	-
ASIMD multiply accumulate long, scalar	SQDMLAL(2), SQDMLSL(2)	4	2	V	-
ASIMD dot product	UDOT, SDOT	4	2*	V	1
ASIMD dot product, by scalar	UDOT, SDOT	4	2*	V	1
ASIMD multiply long	SMULL(2), UMULL(2)	4	1	V	1

Instruction Group	Instructions	Exec Latency	Execution Throughput	Utilized Pipelines	Notes
ASIMD multiply long, scalar	SQDMULL	4	2	V	
ASIMD multiply long, vector	SQDMULL(2)	4	1	V	
ASIMD polynomial (8x8) multiply long	PMULL.8B, PMULL2.16B	3	1		3
ASIMD pairwise add and accumulate	SADALP, UADALP	7	2/3*	V	1
ASIMD shift accumulate	SSRA, USRA	3 (2)	2*	V	1, 4
ASIMD shift accumulate	SRSRA, URSRA	7	2/3*	V	1
ASIMD shift by immed	SHL, SHRN(2), SLI, SRI, SSHR, USHR	2	2*	V	1
ASIMD shift by immed and insert	SLI, SRI	2	2*	V	1
ASIMD shift by immed	SHLL(2), SSHLL(2S), USHLL(2), SXTL(2), UXTL(2)	2	1	V	-
ASIMD shift by immed	RSHRN(2), SRSHR, URSHR	3	2*	V	1
ASIMD shift by immed	SQSHRN(2), UQSHRN(2)	4	2*	V	1
ASIMD shift by immed	SQSHL{U}, UQSHL, SQRSHRN(2), UQRSHRN(2), SQRSHRUN(2), SQSHRUN(2)	4	2*	V	1
ASIMD shift by register	SSHL, USHL	2	2*	V	1
ASIMD shift by register	SRSHL, URSHL	3	2*	V	1

Instruction Group	Instructions	Exec Latency	Execution Throughput	Utilized Pipelines	Notes
ASIMD shift by register	SQSHL, UQSHL, SQRSHL, UQRSHL	4	2*	V	1

1. If the instruction has Q-form, the Q-form of the instruction has Execution Throughput of half of the value shown.
2. These instructions support late forwarding of their Vn operands from similar µOPs, allowing a typical sequence of µOPs to issue one every N cycles (accumulate latency N shown in parentheses).
3. This category includes instructions of the form “PMULL Vd.8H, Vn.8B, Vm.8B” and “PMULL2 Vd.8H, Vn.16B, Vm.16B”.
4. These instructions support late forwarding of accumulate operands from similar µOPs, allowing a typical sequence of µOPs to issue one every N cycles (accumulate latency N shown in parentheses).

3.15 ASIMD floating-point instructions

Table 15: ASIMD integer instructions

Instruction Group	Instructions	Exec Latency	Execution Throughput	Utilized Pipelines	Notes
ASIMD FP arith	FABS, FABD, FADD, FSUB, FADDP	4	2*	V	1
ASIMD FP compare	FACGE, FACGT, FCMEQ, FCMGE, FCMGT, FCMLE, FCMLT	2	2*	V	1
ASIMD FP convert, long	FCVTL(2)	4	1	V	-
ASIMD FP convert, narrow, vector	FCVTN(2), FCVTXN(2)	4	1	V	1
ASIMD FP convert, narrow, scalar	FCVTXN	4	2	V	1

Instruction Group	Instructions	Exec Latency	Execution Throughput	Utilized Pipelines	Notes
ASIMD FP convert, other	FCVTAS, FCVTAU, FCVTMS, FCVTMU, FCVTNS, FCVTNU, FCVTPS, FCVTPU, FCVTZS, FCVTZU, SCVTF, UCVTF	4	2*	V	1
ASIMD FP divide, H-form	FDIV	10	1/3	V	-
ASIMD FP divide, S-form	FDIV	14	1/5	V	-
ASIMD FP divide, D-form	FDIV	23	2/19	V	-
ASIMD FP max/min, normal	FMAX, FMAXNM, FMIN, FMINNM	4	2*	V	1
ASIMD FP max/min, pairwise	FMAXP, FMAXNMP, FMINP, FMINNMP	4	2*	V	1
ASIMD FP max/min, reduce	FMAXV, FMAXNMV, FMINV, FMINNMV	4	1	V	-
ASIMD FP multiply	FMUL, FMULX	4	2*	V	1
ASIMD FP multiply accumulate	FMLA, FMLS	4	2*	V	1
ASIMD FP multiply accumulate, by element	FMLA, FMLS	4	2	V	-
ASIMD FP negate	FNEG	4	2*	V	1

Instruction Group	Instructions	Exec Latency	Execution Throughput	Utilized Pipelines	Notes
ASIMD FP round	FRINTA, FRINTI, FRINTM, FRINTN, FRINTP, FRINTX, FRINTZ	4	2*	V	1
ASIMD FP square root, H-form	FSQRT	10	1/3	V	-
ASIMD FP square root, S-form	FSQRT	13	2/9*	V	1
ASIMD FP square root, D-form	FSQRT	23	2/19	V	-

1. If the instruction has Q-form, the Q-form of the instruction has Execution Throughput of half of the value shown.

3.16 ASIMD miscellaneous instructions

Table 16: ASIMD miscellaneous instructions

Instruction Group	Instructions	Exec Latency	Execution Throughput	Utilized Pipelines	Notes
ASIMD bit reverse	RBIT	2	2*	V	1
ASIMD bitwise insert	BIF, BIT, BSL	1	2*	V	1
ASIMD count	CLZ, CNT	2	2*	V	1
ASIMD count	CLS	3	2*	V	1
ASIMD duplicate, gen reg	DUP	2	2	V, I	-
ASIMD duplicate, element	DUP	2	2	V	-
ASIMD extract	EXT	2	2*	V	1
ASIMD extract narrow	XTN(2)	4	2*	V	1
ASIMD extract narrow, saturating	SQXTN(2), SQXTUN(2), UQXTN(2)	4	2*	V	1
ASIMD insert, element to element	INS	2	1	V	-

Instruction Group	Instructions	Exec Latency	Execution Throughput	Utilized Pipelines	Notes
ASIMD move, integer immed	MOVI, MVNI	2	2*	V	1
ASIMD move, FP immed	FMOV	2	2*	V	1
ASIMD reciprocal estimate	FRECPE, FRECPX, FRSQRTE, URECPE, URSQRTE	4	2*	V	1
ASIMD reciprocal step	FRECPS, FRSQRTS	4	2*	V	1
ASIMD reverse	REV16, REV32, REV64	2	2*	V	1
ASIMD table lookup, 1 table reg	TBL	6	1/11	D	-
ASIMD table lookup, 2+ table regs	TBL	N+7	1/(N+12)	D	2
ASIMD table lookup extension, 1 table reg	TBX	7	1/12	D	-
ASIMD table lookup extension, 2+ table regs	TBX	N+8	1/(N+13)	D	2
ASIMD transfer, element to gen reg	SMOV, UMOV	3	2	V	-
ASIMD transfer, gen reg to element	INS	2	1	V, I	-
ASIMD transpose	TRN1, TRN2	2	2*	V	1
ASIMD unzip/zip	UZP1, UZP2, ZIP1, ZIP2	2	2*	V	1

1. If the instruction has Q-form, the Q-form of the instruction has Execution Throughput of half of the value shown.
2. N denotes the number of registers in the table.

3.17 ASIMD load instructions

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

Table 17: ASIMD load instructions

Instruction Group	Instructions	Exec Latency	Execution Throughput	Utilized Pipelines	Notes
ASIMD load, 1 element, multiple, 1 reg, D-form	LD1	2	1	LS	-
ASIMD load, 1 element, multiple, 1 reg, Q-form	LD1	2	1	LS	-
ASIMD load, 1 element, multiple, 2 reg, D-form	LD1	2	1	LS	-
ASIMD load, 1 element, multiple, 2 reg, Q-form	LD1	3	1/2	LS	-
ASIMD load, 1 element, multiple, 3 reg, D-form	LD1	3	1/2	LS	-
ASIMD load, 1 element, multiple, 3 reg, Q-form	LD1	4	1/3	LS	-
ASIMD load, 1 element, multiple, 4 reg, D-form	LD1	4	1/2	LS	-
ASIMD load, 1 element, multiple, 4 reg, Q-form	LD1	6	1/4	LS	-
ASIMD load, 1 element, one lane	LD1	6	1/2	LS, SD	-
ASIMD load, 1 element, all lanes, B/H/S	LD1R	2	1	LS	-
ASIMD load, 1 element, all lanes, D	LD1R	3	1	LS	-
ASIMD load, 2 element, multiple, D-form	LD2	2	1	LS	-
ASIMD load, 2 element, multiple, Q-form	LD2	6	1/2	LS	-
ASIMD load, 2 element, one lane, B/H/S	LD2	8	1/4	LS, SD	-
ASIMD load, 2 element, one lane,	LD2	9	1/4	LS, SD	-
ASIMD load, 2 element, all lanes, B/H/S	LD2R	2	1	LS	-
ASIMD load, 2 element, all lanes, D	LD2R	3	1	LS	-
ASIMD load, 3 element, multiple, D-form	LD3	4	1/2	LS	-

Instruction Group	Instructions	Exec Latency	Execution Throughput	Utilized Pipelines	Notes
ASIMD load, 3 element, multiple, Q-form, B/H/S	LD3	8	1/3	LS	-
ASIMD load, 3 element, multiple, Q-form, D	LD3	7	1/3	LS	-
ASIMD load, 3 element, one lane, B/H/S	LD3	10	1/6	LS, SD	-
ASIMD load, 3 element, one lane, D	LD3	11	1/6	LS, SD	-
ASIMD load, 3 element, all lanes, B/H/S	LD3R	3	1/2	LS	-
ASIMD load, 3 element, all lanes, D	LD3R	4	1/2	LS	-
ASIMD load, 4 element, multiple, D-form	LD4	4	1/2	LS	-
ASIMD load, 4 element, multiple, Q-form, B/H/S	LD4	9	1/4	LS	-
ASIMD load, 4 element, multiple, Q-form, D	LD4	8	1/4	LS	-
ASIMD load, 4 element, one lane, B/H/S	LD4	12	1/9	LS, SD	-
ASIMD load, 4 element, one lane, D	LD4	13	1/9	LS, SD	-
ASIMD load, 4 element, all lanes, B/H/S	LD4R	3	1/2	LS	-
ASIMD load, 4 element, all lanes, D	LD4R	4	1/2	LS	-
(ASIMD load, writeback form)	-	(1)	-	+ I	1

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

3.18 ASIMD store instructions

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

Table 18: ASIMD store instructions

Instruction Group	Instructions	Exec Latency	Execution Throughput	Utilized Pipelines	Notes
ASIMD store, 1 element, multiple, 1 reg, D-form	ST1	1	1	LS, SD	-
ASIMD store, 1 element, multiple, 1 reg, Q-form	ST1	3/2	2/3	LS, SD	-
ASIMD store, 1 element, multiple, 2 reg, D-form	ST1	1	1	LS, SD	-
ASIMD store, 1 element, multiple, 2 reg, Q-form	ST1	2	1/2	LS, SD	-
ASIMD store, 1 element, multiple, 3 reg, D-form	ST1	3	1/3	LS, SD	-
ASIMD store, 1 element, multiple, 3 reg, Q-form	ST1	5	1/5	LS, SD	-
ASIMD store, 1 element, multiple, 4 reg, D-form	ST1	3	1/3	LS, SD	-
ASIMD store, 1 element, multiple, 4 reg, Q-form	ST1	6	1/6	LS, SD	-
ASIMD store, 1 element, one lane	ST1	1	1	LS, SD	-
ASIMD store, 2 element, multiple, D-form	ST2	1	1	LS, SD	-
ASIMD store, 2 element, multiple, Q-form	ST2	2	1/2	LS, SD	-
ASIMD store, 2 element, one lane	ST2	1	1	LS, SD	-
ASIMD store, 3 element, multiple, D-form	ST3	4	1/4	LS, SD	-
ASIMD store, 3 element, multiple, Q-form	ST3	6	1/6	LS, SD	-
ASIMD store, 3 element, one lane	ST3	2	1/2	LS, SD	-
ASIMD store, 4 element, multiple, D-form	ST4	3	1/3	LS, SD	-
ASIMD store, 4 element, multiple, Q-form	ST4	8	1/8	LS, SD	-
ASIMD store, 4 element, one lane	ST4	3	1/3	LS, SD	-

Instruction Group	Instructions	Exec Latency	Execution Throughput	Utilized Pipelines	Notes
(ASIMD store, writeback form)	-	(1)	-	+ I	1

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

3.19 Cryptography extensions

Table 19: Cryptography extensions

Instruction Group	Instructions	Exec Latency	Execution Throughput	Utilized Pipelines	Notes
Crypto AES ops	AESD, AESE	2	1	V	-
Crypto AES ops	AESIMC, AESMC	2	1	V	-
Crypto polynomial (64x64) multiply long	PMULL(2)	2	1	V	-
Crypto SHA1 xor ops	SHA1SU0	2	1	V	-
Crypto SHA1 schedule acceleration ops	SHA1H, SHA1SU1	2	1	V	1
Crypto SHA1 hash acceleration ops	SHA1C, SHA1M, SHA1P	5	1	V	-
Crypto SHA256 schedule acceleration op	SHA256SU0	3	1	V	-
Crypto SHA256 schedule acceleration op	SHA256SU1	5	1	V	-
Crypto SHA256 hash acceleration ops	SHA256H, SHA256H2	5	1	V	-

1. Adjacent AESE/AESMC instruction pairs and adjacent AESD/AESIMC instruction pairs can benefit performance as described in Section 4.8.

3.20 CRC

Table 20: CRC

Instruction Group	Instructions	Exec Latency	Execution Throughput	Utilized Pipelines	Notes
CRC checksum ops	CRC32B, CRC32H, CRC32CB, CRC32CH, CRC32X, CRC32CX	2 (1)	2	I	1
CRC checksum ops	CRC32W, CRC32CW	1	2	I	-

1. CRC µOPs support late forwarding of results to their Rn operands, resulting in a reduced latency shown in parentheses.

4 Special considerations

4.1 SMT resource sharing

Software threads scheduled to run together on a single Cortex-A65 core share hardware resources throughout the CPU, including pipeline bandwidth, memory bandwidth at the core interface, and capacity in the core's private caches and TLBs. It is useful to consider this in addition to any existing strategies in place for sharing multiple cores and system resources among threads.

Cortex-A65 is designed to maximize throughput and maintain fairness regardless of workload, but the beneficial effects of SMT can be maximized by maintaining an awareness of each Cortex-A65 core's two PEs as the closest level of hardware affinity in the system. Ideally, scheduling of particularly compatible pairs of threads should be favored if such distinctions exist in target workloads. For example, suitable threads might have:

- a relatively high degree of sharing of working sets in memory
- dissimilar needs for core-bound compute capacity or memory bandwidth
- differences in timing of peak compute needs

Scenarios in which one or both threads are frequently memory-bound will likely allow Cortex-A65 to hide memory latency by allowing high-IPC progress in one thread while the other is stalled waiting for a data to return.

4.2 Optimizing memory copy

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.
- Use non-writeback forms of LDP and STP instructions interleaving them like shown in the example below:

```
Loop_start:
      SUBS   X2,X2,#96
      LDP    X3,X4,[x1,#0]
      STP    X3,X4,[x0,#0]
      LDP    X3,X4,[x1,#16]
      STP    X3,X4,[x0,#16]
      LDP    X3,X4,[x1,#32]
      STP    X3,X4,[x0,#32]
      LDP    X3,X4,[x1,#48]
      STP    X3,X4,[x0,#48]
      LDP    X3,X4,[x1,#64]
```

```
STP    X3,X4,[x0,#64]
LDP    X3,X4,[x1,#80]
STP    X3,X4,[x0,#80]
ADD    X1,X1,#96
ADD    X0,X0,#96
BGT    Loop_start
```

4.3 Load/Store alignment

The Armv8.2-A architecture allows many types of load and store accesses to be arbitrarily aligned. Cortex-A65 executes unaligned load and store instructions with the same latency and bandwidth characteristics as naturally aligned equivalents unless the unaligned data crosses a 16-byte boundary. Those crossing 16-byte boundaries introduce performance penalties.

4.4 Hardware data prefetch

The Cortex-A65 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-A65 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 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.

4.5 Software prefetch performance

The Cortex-A65 core responds to PRFM hint instructions by generating corresponding prefetch transactions to memory. It is not advisable to use explicit software prefetching if the relevant access pattern falls within the capabilities of the hardware data prefetcher since prefetch instructions consume pipeline bandwidth.

4.6 Non-temporal and transient memory hints

The Cortex-A65 core makes use of non-temporal memory hints provided by some load, store, and software prefetch instructions, accounting for them in its cache allocation policies for corresponding memory regions. Memory pages marked as transient in the page tables have the

same effect. Software can use these mechanisms for the benefit of accesses to other memory regions more likely to be referenced again in the near future.

4.7 Branch instruction alignment

Adjacent pairs of branch instructions within the same aligned 64-bit region of memory can affect performance. It is preferable to avoid this arrangement if it can be done without affecting program density and performance otherwise. It is preferable for branch targets, including subroutine entry points, to be placed on aligned 64-bit boundaries to maximize instruction fetch efficiency.

4.8 Instruction fusion

Cortex-A65 can accelerate certain instruction pairs in an operation called fusion. Specific Aarch64 instruction pairs that can be fused are as follows:

1. ADRP + load (unsigned immediate)
2. ADRP + store (unsigned immediate)
3. MOVZ + MOVK
4. AESE + AESMC
5. AESD + AESIMC

These instruction pairs must be adjacent to each other in program code for fusion to take effect. In each case the destination register operand of the younger instruction must match the source operand of the older instruction.