Cortex-A57 Software Optimisation Notes

Document number:	ARM-EPM-040982 2.0
Date of Issue:	24th September 2013
Confidentiality:	Confidential

Contents

1 ABOUT THIS DOCUMENT

1.1 Change control

Change history for this document is maintained in Auspex.

1.1.1 Current status and keychanges in this revision

Version	Date	Change Summary
1.0	24th May 2013	Initial version for r0p0-00lac0 release
2.0	24th September 2013	Updated for r0p1-00lac0 release

1.2 References

This document refers to the following documents.

Ref	Doc No	Author(s)	Title
A	ARM DDI 0488A-7a	ARM	Cortex-A57 Technical Reference Manual

1.3 Terms and abbreviations

This document uses the following terms and abbreviations.

Term	Meaning
ALU	Arithmetic/Logical Unit
ASIMD	Advanced SIMD
Uop	Micro-Operation
VFP	Vector Floating Point

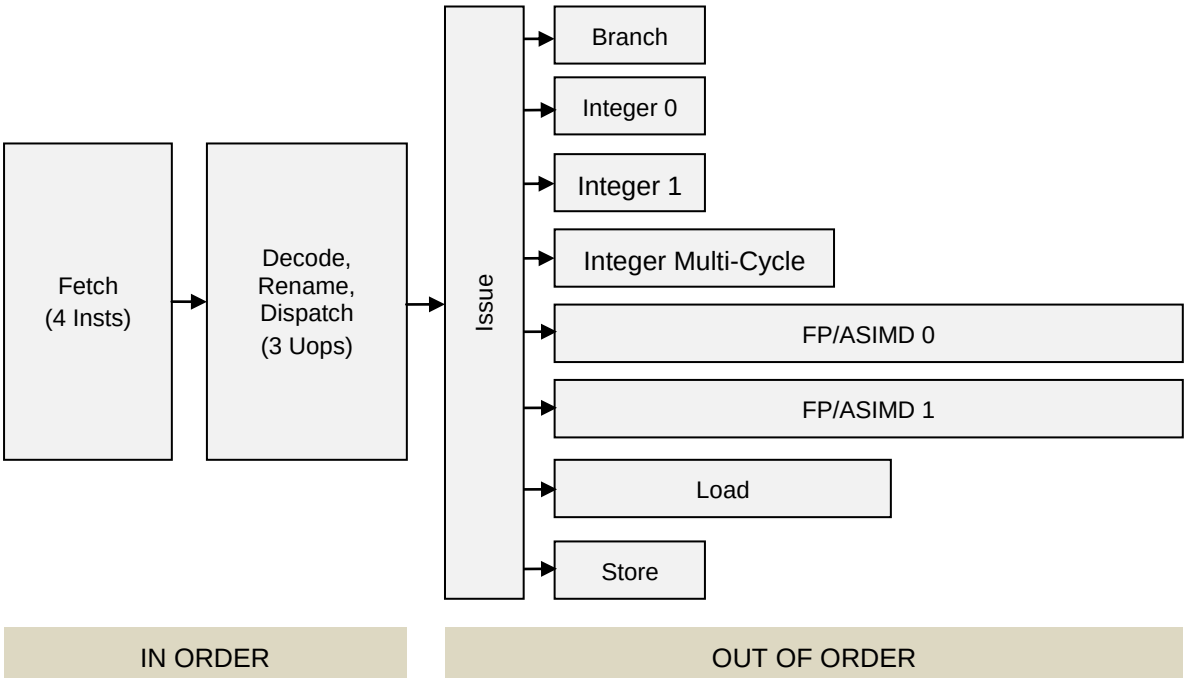
1.4 Document Scope

This document provides high-level information about the Cortex-A57 pipeline, instruction performance characteristics, and special performance considerations. This information is intended to aid people who are optimizing software and compilers for Cortex-A57. For a more complete description of the Cortex-A57 processor, please refer to the *Cortex-A57 Technical Reference Manual*.

2 INTRODUCTION

2.1 Pipeline Overview

The following diagram describes the Cortex-A57 instruction processing pipeline. Instructions are first fetched, up to four per cycle. Then they are decoded into internal micro-operations (uops). From there, the uops proceed through register renaming and dispatch stages, up to three per cycle. Once dispatched, uops are allowed to wait for their operands and issue out-of-order to one of eight execution pipelines. Each execution pipeline can accept and complete only one uop per cycle.



The execution pipelines support different types of operations. Uop types may be classified as follows.

- Type B – Branch uops
- Type I – Integer ALU
- Type L – Load and register transfer uops
- Type M – Integer shift-ALU, multiply, divide, and sum-of-absolute-differences uops
- Type S – Store and special memory uops
- Type V – ASIMD ALU, ASIMD misc, FP misc, FP add, FP multiply, crypto xor uops
- Type W – ASIMD integer multiply, crypto uops
- Type X – ASIMD shift uops, FP divide

The functionality of each execution pipeline is summarized below. Some pipelines have multiple internal sub-pipelines and support variable-latency operations.

- Branch Pipeline – Supports uop type B
- Integer Pipeline 0 – Supports uop type I
- Integer Pipeline 1 – Supports uop type I
- Multiply Pipeline – Supports uop type M
- FP/ASIMD Pipeline 0 – Supports uop type V and W
- FP/ASIMD Pipeline 1 – Supports uop type V and X
- Load Pipeline – Supports uop type L
- Store Pipeline – Supports uop type S

3 INSTRUCTION CHARACTERISTICS

3.1 Instruction Tables

This chapter describes high-level performance characteristics for most ARMv8 A32, T32 and A64 instructions. A series of tables summarize the effective execution latency, uop count, uop types, and special behaviors associated with each group of instructions. When multiple uops are generated, the execution latency corresponds to the total effective minimum latency, taking into account uop interdependences and pipeline requirements. Uop types corresponds to the execution pipelines described in chapter 2.

3.2 Branch Instructions

Instruction Group	AArch32 Instructions	Exec Latency	Uop Count	Uop Types	Notes
Branch, immed	B	1	1	B	
Branch, register	BX	1	1	B	
Branch and link, immed	BL, BLX	1	2	I B	
Branch and link, register	BLX	2	2	I B	
Compare and branch	CBZ, CBNZ	1	1	B	

Instruction Group	AArch64 Instructions	Exec Latency	Uop Count	Uop Types	Notes
Branch, immed	B	1	1	B	
Branch, register	BR, RET	1	1	B	
Branch and link, immed	BL	1	2	I B	
Branch and link, register	BLR	2	2	I B	
Compare and branch	CBZ, CBNZ, TBZ, TBNZ	1	1	B	

3.3 Arithmetic and Logical Instructions

Instruction Group	AArch32 Instructions	Exec Latency	Uop Count	Uop Types	Notes
ALU, basic	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	1	I	
ALU, shift by immed	(same as above)	2	1	M	
ALU, shift by register, unconditional	(same as above)	2	1	M	
ALU, shift by register, conditional	(same as above)	2	2	I I	
(ALU, branch forms)		+2	+1	+B	1

Instruction Group	AArch64 Instructions	Exec Latency	Uop Count	Uop Types	Notes
ALU, basic	ADD{S}, ADC{S}, AND{S}, BIC{S}, EON, EOR, ORN, ORR, SUB{S}, SBC{S}	1	1	I	
ALU, extend and/or shift	ADD{S}, AND{S}, BIC{S}, EON, EOR, ORN, ORR, SUB{S}	2	1	M	
Conditional compare	CCMN, CCMP	1	1	I	
Conditional select	CSEL, CSINC, CSINV, CSNEG	1	1	I	

NOTE 1 – Branch forms are possible when the instruction destination register is the PC. For those cases, an additional branch uop is required.

3.4 Move and Shift Instructions

Instruction Group	AArch32 Instructions	Exec Latency	Uop Count	Uop Types	Notes
Move, basic	MOV{S}, MOVW, MVN{S}	1	1	I	
Move, shift by immed, no setflags	ASR, LSL, LSR, ROR, RRX, MVN	1	1	I	
Move, shift by immed, setflags	ASRS, LSLS, LSRS, RORS, RRXS, MVNS	2	1	M	1
Move, shift by register, no setflags, unconditional	ASR, LSL, LSR, ROR, RRX, MVN	1	1	I	
Move, shift by register, no setflags, conditional	ASR, LSL, LSR, ROR, RRX, MVN	2	2	I I	
Move, shift by register, setflags, unconditional	ASRS, LSLS, LSRS, RORS, RRXS, MVNS	2	1	M	1
Move, shift by register, setflags, conditional	ASRS, LSLS, LSRS, RORS, RRXS, MVNS	2	2	I I	
Move, top	MOVT	2	1	2	1
(Move, branch forms)		+2	+1	+B	2

Instruction Group	AArch64 Instructions	Exec Latency	Uop Count	Uop Types	Notes
Address generation	ADR, ADRP	1	1	I	
Move immed	MOVN, MOVK, MOVZ	1	1	I	
Variable shift	ASRV, LSLV, LSRV, RORV	1	1	I	

NOTE 1 – Instructions that require both a shift and a setflags or combine operation are usually performed by a single “shift-alu” uop that stalls the integer pipeline for one extra cycle.

NOTE 2 – Branch forms are possible when the instruction destination register is the PC. For those cases, an additional branch uop is required.

3.5 Divide and Multiply Instructions

Instruction Group	AArch32 Instructions	Exec Latency	Uop Count	Uop Types	Notes
Divide	SDIV, UDIV	19	1	M	1
Multiply	MUL, SMULBB, SMULBT, SMULTB, SMULTT, SMULWB, SMULWT, SMMUL{R}	3	1	M	
Multiply accumulate	MLA, MLS, SMLABB, SMLABT, SMLATB, SMLATT, SMLAWB, SMLAWT, SMLAD{X}, SMLSD{X}, SMMLA{R}, SMMLS{R}, SMUAD{X}, SMUSD{X}	3 (1)	1	M	2
Multiply accumulate long	SMLAL, SMLALBB, SMLALBT, SMLALTB, SMLALTT, SMLALD{X}, SMLSLD{X}, UMAAL, UMLAL	4 (1)	1	M	2, 3
Multiply long	SMULL, UMULL	4	1	M	3
(Multiply, setflags forms)		+1	+1	+I	4

Instruction Group	AArch64 Instructions	Exec Latency	Uop Count	Uop Types	Notes
Divide, W-form	SDIV, UDIV	19	1	M	1
Divide, X-form	SDIV, UDIV	35	1	M	1
Multiply accumulate, W-form	MADD, MSUB	3 (1)	1	M	2
Multiply accumulate, X-form	MADD, MSUB	5 (3)	1	M	2
Multiply accumulate long	SMADDL, SMSUBL, UMADDL, UMSUBL	3 (1)	1	M	2
Multiply high	SMULH, UMULH	6 [4]	1	M	5

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

NOTE 2 – Multiply-accumulate pipelines support late-forwarding of accumulate operands from similar uops, allowing a typical sequence of multiply-accumulate uops to issue one every N cycles (accumulate latency N shown in parentheses).

NOTE 3 – Long-form multiplies (which produce two result registers) stall the multiplier pipeline for one extra cycle.

NOTE 4 – Multiplies that set the condition flags require an additional integer uop.

NOTE 5 – Multiply high operations stall the multiplier pipeline for N cycles before any other type M uop can be issued to that pipeline, with N shown in brackets

3.6 Saturating and Parallel Arithmetic Instructions

Instruction Group	AArch32 Instructions	Exec Latency	Uop Count	Uop Types	Notes
Parallel arith, unconditional	SADD16, SADD8, SSUB16, SSUB8, UADD16, UADD8, USUB16, USUB8	2	1	M	
Parallel arith, conditional	SADD16, SADD8, SSUB16, SSUB8, UADD16, UADD8, USUB16, USUB8	2 (4)	4	M I I I	1
Parallel arith with exchange, unconditional	SASX, SSAX, UASX, USAX	3	2	I M	
Parallel arith with exchange, conditional	SASX, SSAX, UASX, USAX	3 (5)	5	I M I I I	1
Parallel halving arith	SHADD16, SHADD8, SHSUB16, SHSUB8, UHADD16, UHADD8, UHSUB16, UHSUB8	2	1	M	
Parallel halving arith with exchange	SHASX, SHSAX, UHASX, UHSAX	3	2	I M	
Parallel saturating arith	QADD16, QADD8, QSUB16, QSUB8, UQADD16, UQADD8, UQSUB16, UQSUB8	2	1	M	
Parallel saturating arith with exchange	QASX, QSAX, UQASX, UQSAX	3	2	I M	
Saturate	SSAT, SSAT16, USAT, USAT16	2	1	M	
Saturating arith	QADD, QSUB	2	1	M	
Saturating doubling arith	QDADD, QDSUB	3	2	I M	

NOTE 1 – Conditional GE-setting instructions require three extra uops and two additional cycles to conditionally update the GE field (GE latency shown in parentheses).

3.7 Miscellaneous Data-Processing Instructions

Instruction Group	AArch32 Instructions	Exec Latency	Uop Count	Uop Types	Notes
Bit field extract	SBFX, UBFX	1	1	I	
Bit field insert/clear	BFI, BFC	2	1	M	1

Instruction Group	AArch32 Instructions	Exec Latency	Uop Count	Uop Types	Notes
Count leading zeros	CLZ	1	1	I	
Pack halfword	PKH	2	1	M	
Reverse bits/bytes	RBIT, REV, REV16, REVSH	1	1	I	
Select bytes, unconditional	SEL	1	1	I	
Select bytes, conditional	SEL	2	2	I I	
Sign/zero extend, normal	SXTB, SXTH, UXTB, UXTH	1	1	I	
Sign/zero extend, parallel	SXTB16, UXTB16	2	1	M	
Sign/zero extend and add, normal	SXTAB, SXTAH,UXTAB, UXTAH	2	1	M	
Sign/zero extend and add, parallel	SXTAB16, UXTAB16	4	2	M M	
Sum of absolute differences	USAD8, USADA8	3	1	M	

Instruction Group	AArch64 Instructions	Exec Latency	Uop Count	Uop Types	Notes
Bitfield extract, one reg	EXTR	1	1	I	
Bitfield extract, two regs	EXTR	3	2	I M	
Bitfield move, basic	SBFM, UBFM	1	1	I	
Bitfield move, insert	BFM	2	1	M	
Count leading	CLS, CLZ	1	1	I	
Reverse bits/bytes	RBIT, REV, REV16, REV32	1	1	I	

3.8 Load Instructions

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

Instruction Group	AArch32 Instructions	Exec Latency	Uop Count	Uop Types	Notes
Load, immed offset	LDR{T}, LDRB{T}, LDRD, LDRH{T}, LDRSB{T}, LDRSH{T}	4	1	L	
Load, register offset, plus	LDR, LDRB, LDRD, LDRH, LDRSB, LDRSH	4	1	L	
Load, register offset, minus	LDR, LDRB, LDRD, LDRH, LDRSB, LDRSH	5	2	I L	
Load, scaled register offset, plus LSL2	LDR, LDRB	4	1	L	
Load, scaled register offset, other	LDR, LDRB, LDRH, LDRSB, LDRSH	5	2	I L	

Instruction Group	AArch32 Instructions	Exec Latency	Uop Count	Uop Types	Notes
Load, immed pre-indexed	LDR, LDRB, LDRD, LDRH, LDRSB, LDRSH	4 (1)	2	L I	1
Load, register pre-indexed	LDR, LDRB, LDRD, LDRH, LDRSB, LDRSH	4 (2)	3	I L I	1
Load, scaled register pre-indexed, plus LSL2	LDR, LDRB	4 (2)	3	I L I	1
Load, scaled register pre-indexed, other	LDR, LDRB	5 (2)	3	I L I	1
Load, immed post-indexed	LDR{T}, LDRB{T}, LDRD, LDRH{T}, LDRSB{T}, LDRSH{T}	4 (1)	2	L I	1
Load, register post-indexed	LDR{T}, LDRB{T}, LDRD, LDRH{T}, LDRSB{T}, LDRSH{T}	4 (2)	3	I L I	1
Load, scaled register post-indexed	LDR{T}	4 (2)	3	I L I	1
Preload, immed offset	PLD, PLDW	4	1	L	
Preload, register offset, plus	PLD, PLDW	4	1	L	
Preload, register offset, minus	PLD, PLDW	5	2	I L	
Preload, scaled register offset, plus LSL2	PLD, PLDW	4	1	L	
Preload, scaled register offset, other	PLD, PLDW	5	2	I L	
Load multiple, no writeback	LDMIA, LDMIB, LDMDA, LDMDB	3 + N	N	LxN	2
Load multiple, writeback	LDMIA, LDMIB, LDMDA, LDMDB, POP	3 + N	N + 1	LxN I	1, 2
(Load, branch forms)		+2	+1	+B	3

Instruction Group	AArch64 Instructions	Exec Latency	Uop Count	Uop Types	Notes
Load register, literal	LDR, LDRSW, PRFM	4	1	L	
Load register, unscaled immed	LDUR, LDURB, LDURH, LDURSB, LDURSH, LDURSW, PRFUM	4	1	L	
Load register, immed post-index	LDR, LDRB, LDRH, LDRSB, LDRSH, LDRSW	4 (1)	2	L I	1
Load register, immed pre-index	LDR, LDRB, LDRH, LDRSB, LDRSH, LDRSW	4 (1)	2	L I	1
Load register, immed unprivileged	LDTR, LDTRB, LDTRH, LDTRSB, LDTRSH, LDTRSW	4	1	L	

Instruction Group	AArch64 Instructions	Exec Latency	Uop Count	Uop Types	Notes
Load register, unsigned immed	LDR, LDRB, LDRH, LDRSB, LDRSH, LDRSW, PRFM	4	1	L	
Load register, register offset, basic	LDR, LDRB, LDRH, LDRSB, LDRSH, LDRSW, PRFM	4	1	L	
Load register, register offset, scale by 4/8	LDR, LDRSW, PRFM	4	1	L	
Load register, register offset, scale by 2	LDRH, LDRSH	5	2	I L	
Load register, register offset, extend	LDR, LDRB, LDRH, LDRSB, LDRSH, LDRSW, PRFM	4	1	L	
Load register, register offset, extend, scale by 4/8	LDR, LDRSW, PRFM	4	1	L	
Load register, register offset, extend, scale by 2	LDRH, LDRSH	5	2	I L	
Load pair, immed offset, normal	LDP	4	1	L	
Load pair, immed offset, signed words	LDPSW	5	3	I L L	
Load pair, immed post-index, normal	LDP	4 (1)	2	L I	1
Load pair, immed post-index, signed words	LDPSW	5 (1)	4	I L L I	1
Load pair, immed pre-index, normal	LDP	4 (1)	2	L I	1
Load pair, immed pre-index, signed words	LDPSW	5 (1)	4	I L L I	1

NOTE 1 – Address updates are typically completed in parallel with the load operation and with shorter latency (address update latency shown in parentheses).

NOTE 2 – For load multiple instructions, N is the number of required load uops. N=floor((num_regs+1)/2).

NOTE 3 – Branch forms are possible when the instruction destination register is the PC. For those cases, an additional branch uop is required.

3.9 Store Instructions

The following table describes performance characteristics for standard store instructions. Stores uops 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	Uop Count	Uop Types	Notes
Store, immed offset	STR{T}, STRB{T}, STRD, STRH{T}	1	1	S	
Store, register offset, plus	STR, STRB, STRD, STRH	1	1	S	
Store, register offset, minus	STR, STRB, STRD, STRH	3	2	I S	

Instruction Group	AArch32 Instructions	Exec Latency	Uop Count	Uop Types	Notes
Store, scaled register offset, plus LSL2	STR, STRB	1	1	S	
Store, scaled register offset, other	STR, STRB, STRD, STRH	3	2	I S	
Store, immed pre-indexed	STR, STRB, STRD, STRH	1 (1)	2	S I	1
Store, register pre-indexed, plus	STR, STRB, STRD, STRH	1 (1)	2	S I	1
Store, register pre-indexed, minus	STR, STRB, STRD, STRH	3 (2)	3	I S I	1
Store, scaled register pre-indexed, plus LSL2	STR, STRB	1 (2)	2	S M	1
Store, scaled register pre-indexed, other	STR, STRB	3 (2)	3	I S I	1
Store, immed post-indexed	STR{T}, STRB{T}, STRD, STRH{T}	1 (1)	2	S I	1
Store, register post-indexed	STR{T}, STRB{T}, STRD, STRH{T}	1 (2)	2	S M	1
Store, scaled register post-indexed	STR{T}	1 (2)	2	S M	1
Store multiple, no writeback	STMIA, STMIB, STMDA, STMDB	N	N	SxN	1, 2
Store multiple, writeback	STMIA, STMIB, STMDA, STMDB, PUSH	N	N + 1	SxN I	1, 2

Instruction Group	AArch64 Instructions	Exec Latency	Uop Count	Uop Types	Notes
Store register, unscaled immed	STUR, STURB, STURH	1	1	S	
Store register, immed post-index	STR, STRB, STRH	1 (1)	2	S I	1
Store register, immed pre-index	STR, STRB, STRH	1 (1)	2	S I	1
Store register, immed unprivileged	STTR, STTRB, STTRH	1	1	S	
Store register, unsigned immed	STR, STRB, STRH	1	1	S	
Store register, register offset, basic	STR, STRB, STRH	1	1	S	
Store register, register offset, scaled by 4/8	STR	1	1	S	
Store register, register offset, scaled by 2	STRH	3	2	I S	
Store register, register offset, extend	STR, STRB, STRH	1	1	S	
Store register, register offset, extend, scale by 4/8	STR	1	1	S	
Store register, register offset, extend, scale by 1	STRH	3	2	I S	
Store pair, immed offset, W-form	STP	1	1	S	
Store pair, immed offset, X-form	STP	2	2	S S	
Store pair, immed post-index, W-form	STP	1 (1)	2	S I	1

Instruction Group	AArch64 Instructions	Exec Latency	Uop Count	Uop Types	Notes
Store pair, immed post-index, X-form	STP	2 (1)	3	S S I	1
Store pair, immed pre-index, W-form	STP	1 (1)	2	S I	1
Store pair, immed pre-index, X-form	STP	2 (1)	3	S S I	1

NOTE 1 – Address updates are typically completed in parallel with the store operation and with shorter latency (address update latency shown in parentheses).

NOTE 2 – For store multiple instructions, N is the number of required store uops. N=floor((num_regs+1)/2).

3.10 FP Data Processing Instructions

Instruction Group	AArch32 Instructions	Exec Latency	Uop Count	Uop Types	Notes
FP absolute value	VABS	3	1	V	
FP arith	VADD, VSUB	5	1	V	
FP compare, unconditional	VCMP, VCMPE	3	1	V	
FP compare, conditional	VCMP, VCMPE	6	2	V V	
FP convert	VCVT{R}, VCVTB, VCVTT	5	1	V	
FP divide, S-form	VDIV	18	1	X	1
FP divide, D-form	VDIV	32	1	X	1
FP negate	VNEG	3	1	V	
FP multiply, FZ	VMUL, VNMUL	5	1	V	
FP multiply, no FZ	VMUL, VNMUL	6	1	V	
FP multiply accumulate, FZ	VFMA, VFMS, VFNMA, VFNMS, VMLA, VMLS, VNMLA, VNMLS	9 (4)	1	V	2
FP multiply accumulate, no FZ	VFMA, VFMS, VFNMA, VFNMS, VMLA, VMLS, VNMLA, VNMLS	10 (4)	1	V	2
FP square root, S-form	VSQRT	18	1	X	1
FP square root, D-form	VSQRT	32	1	X	1

Instruction Group	AArch64 Instructions	Exec Latency	Uop Count	Uop Types	Notes
FP absolute value	FABS	3	1	V	
FP arithmetic	FADD, FSUB	5	1	V	
FP compare	FCCMP{E}, FCMP{E}	3	1	V	
FP divide, S-form	FDIV	18	1	X	1

Instruction Group	AArch64 Instructions	Exec Latency	Uop Count	Uop Types	Notes
FP divide, D-form	FDIV	32	1	X	1
FP min/max	FMIN, FMINNM, FMAX, FMAXNM	5	1	V	
FP multiply, FZ	FMUL, FNMUL	5	1	V	
FP multiply, no FZ	FMUL, FNMUL	6	1	V	
FP multiply accumulate, FZ	FMADD, FMSUB, FNMADD, FNMSUB	9 (4)	1	V	2
FP multiply accumulate, no FZ	FMADD, FMSUB, FNMADD, FNMSUB	10 (4)	1	V	2
FP negate	FNEG	3	1	V	
FP round	FRINTA, FRINTI, FRINTM, FRINTN, FRINTP, FRINTX, FRINTZ	5	1	V	
FP select	FCSEL	3	1	V	
FP square root, S-form	FSQRT	18	1	X	1
FP square root, D-form	FSQRT	32	1	X	1

NOTE 1 – FP divide and square root operations are performed using an iterative algorithm and block subsequent similar operations to the same pipeline until complete.

NOTE 2 – FP multiply-accumulate pipelines support late-forwarding of accumulate operands from similar uops, allowing a typical sequence of multiply-accumulate uops to issue one every N cycles (accumulate latency N shown in parentheses).

3.11 FP Miscellaneous Instructions

Instruction Group	AArch32 Instructions	Exec Latency	Uop Count	Uop Types	Notes
FP move, immed	VMOV	3	1	V	
FP move, register	VMOV	3	1	V	
FP transfer, vfp to core reg	VMOV	5	1	L	
FP transfer, core reg to vfp	VMOV	5	1	L	

Instruction Group	AArch64 Instructions	Exec Latency	Uop Count	Uop Types	Notes
FP convert, from vec to vec reg	FCVT	5	1	V	
FP convert, from gen to vec reg	SCVTF, UCVTF	10	2	L V	
FP convert, from vec to gen reg	FCVTAS, VCVTAU, FCVTMS, FCVTMU, FCVTNS, FCVTNU, FCVTPS, FCVTPU, FCVTZS, FCVTZU	10	2	V L	

Instruction Group	AArch64 Instructions	Exec Latency	Uop Count	Uop Types	Notes
FP move, immed	FMOV	3	1	V	
FP move, register	FMOV	3	1	V	
FP transfer, from gen to vec reg	FMOV	5	1	L	
FP transfer, from vec to gen reg	FMOV	5	1	L	

3.12 FP Load Instructions

The latencies shown assume the memory access hits in the Level 1 Data Cache. Compared to standard loads, an extra cycle is required to forward results to FP/ASIMD pipelines.

Instruction Group	AArch32 Instructions	Exec Latency	Uop Count	Uop Types	Notes
FP load, register	VLDR	5	1	L	
FP load multiple, unconditional	VLDMIA, VLDMDB, VPOP	4 + N	N	LxN	1
FP load multiple, conditional	VLDMIA, VLDMDB, VPOP	4 + N	N	LxN	2
(FP load, writeback forms)		(1)	+1	+I	3

Instruction Group	AArch64 Instructions	Exec Latency	Uop Count	Uop Types	Notes
Load vector reg, literal	LDR	5	1	L	
Load vector reg, unscaled immed	LDUR	5	1	L	
Load vector reg, immed post-index	LDR	5 (1)	2	L I	3
Load vector reg, immed pre-index	LDR	5 (1)	2	L I	3
Load vector reg, unsigned immed	LDR	5	1	L	
Load vector reg, register offset, basic	LDR	5	1	L	
Load vector reg, register offset, scale, S/D-form	LDR	5	1	L	
Load vector reg, register offset, scale, H/Q-form	LDR	6	2	I L	
Load vector reg, register offset, extend	LDR	5	1	L	
Load vector reg, register offset, extend, scale, S/D-form	LDR	5	1	L	
Load vector reg, register offset, extend, scale, H/Q-form	LDR	6	2	I L	
Load vector pair, immed offset, S/D-form	LDP	5	1	L	
Load vector pair, immed offset, Q-form	LDP	6	2	L L	
Load vector pair, immed post-index, S/D-form	LDP	5 (1)	2	L I	3
Load vector pair, immed post-index, Q-form	LDP	6 (1)	3	L L I	3
Load vector pair, immed pre-index, S/D-form	LDP	5 (1)	2	L I	3
Load vector pair, immed pre-index, Q-form	LDP	6 (1)	3	L L I	3

NOTE 1 – For FP load multiple instructions, N is the number of required load uops. N=floor((num_regs+1)/2) for unconditional forms only.

NOTE 2 – For conditional FP load multiple instructions, N is the number of required load uops. N = num_regs for conditional forms only.

NOTE 3 – Writeback forms of load instructions require an extra address update uop. This update is typically performed in parallel with or prior to the load uop (address update latency shown in parentheses).

3.13 FP Store Instructions

Stores uops 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	Uop Count	Uop Types	Notes
FP store, immed offset	VSTR	1	1	S	
FP store multiple, S-form	VSTMIA, VSTMDB, VPUSH	N	N	SxN	1
FP store multiple, D-form	VSTMIA, VSTMDB, VPUSH	N	N	SxN	1
(FP store, writeback forms)		(1)	+1	+I	2

Instruction Group	AArch64 Instructions	Exec Latency	Uop Count	Uop Types	Notes
Store vector reg, unscaled immed, B/H/S/D-form	STUR	1	1	S	
Store vector reg, unscaled immed, Q-form	STUR	2	2	S S	
Store vector reg, immed post-index, B/H/S/D-form	STR	1 (1)	2	S I	2
Store vector reg, immed post-index, Q-form	STR	2 (1)	3	S S I	2
Store vector reg, immed pre-index, B/H/S/D-form	STR	1 (1)	2	S I	2
Store vector reg, immed pre-index, Q-form	STR	2 (1)	4	I S S I	2
Store vector reg, unsigned immed, B/H/S/D-form	STR	1	1	S	
Store vector reg, unsigned immed, Q-form	STR	2	3	I S S	
Store vector reg, register offset, basic, B/H/S/D-form	STR	1	1	S	
Store vector reg, register offset, basic, Q-form	STR	2	3	I S S	
Store vector reg, register offset, scale, H-form	STR	3	2	I S	
Store vector reg, register offset, scale, S/D-form	STR	1	1	S	
Store vector reg, register offset, scale, Q-form	STR	3	4	I S I S	
Store vector reg, register offset, extend, B/H/S/D-form	STR	1	1	S	
Store vector reg, register offset, extend, Q-form	STR	3	3	M S S	
Store vector reg, register offset, extend, scale, H-form	STR	3	2	I S	

Instruction Group	AArch64 Instructions	Exec Latency	Uop Count	Uop Types	Notes
Store vector reg, register offset, extend, scale, S/D-form	STR	1	1	S	
Store vector reg, register offset, extend, scale, Q-form	STR	3	4	I S I S	
Store vector pair, immed offset, S-form	STP	1	1	S	
Store vector pair, immed offset, D-form	STP	2	2	S S	
Store vector pair, immed offset, Q-form	STP	4	5	I S S S S	
Store vector pair, immed post-index, S-form	STP	1 (1)	2	S I	2
Store vector pair, immed post-index, D-form	STP	2 (1)	3	S S I	2
Store vector pair, immed post-index, Q-form	STP	4 (1)	5	S S S S I	2
Store vector pair, immed pre-index, S-form	STP	1 (1)	2	S I	2
Store vector pair, immed pre-index, D-form	STP	2 (1)	3	S S I	2
Store vector pair, immed pre-index, Q-form	STP	4 (1)	6	I S S S S I	2

NOTE 1 – For FP store multiple instructions, N is the number of required store uops. For single-precision stores, N=floor((num_regs+1)/2). For double-precision stores, N=(num_regs).

NOTE 2 – Writeback forms of store instructions require an extra address update uop. This update is typically performed in parallel with or prior to the store uop (address update latency shown in parentheses).

3.14 ASIMD Integer Instructions

Instruction Group	AArch32 Instructions	Exec Latency	Uop Count	Uop Types	Notes
ASIMD absolute diff, D-form	VABD	3	1	V	
ASIMD absolute diff, Q-form	VABD	3	2	V V	
ASIMD absolute diff accum, D-form	VABA	4 (1)	1	X	2
ASIMD absolute diff accum, Q-form	VABA	5 (2)	2	X X	2
ASIMD absolute diff accum long	VABAL	4 (1)	1	X	2
ASIMD absolute diff long	VABDL	3	1	V	
ASIMD arith, basic	VADD, VADDL, VADDW, VNEG, VPADD, VPADDL, VSUB, VSUBL, VSUBW	3	1	V	
ASIMD arith, complex	VABS, VADDHN, VHADD, VHSUB, VQABS, VQADD, VQNEG, VQSUB, VRADDHN, VRHADD, VRSUBHN, VSUBHN	3	1	V	
ASIMD compare	VCEQ, VCGE, VCGT, VCLE, VTST	3	1	V	

Instruction Group	AArch32 Instructions	Exec Latency	Uop Count	Uop Types	Notes
ASIMD logical	VAND, VBIC, VMVN, VORR, VORN, VEOR	3	1	V	
ASIMD max/min	VMAX, VMIN, VPMAX, VPMIN	3	1	V	
ASIMD multiply, D-form	VMUL, VQDMULH, VQRDMULH	5	1	W	
ASIMD multiply, Q-form	VMUL, VQDMULH, VQRDMULH	6	2	W W	
ASIMD multiply accumulate, D-form	VMLA, VMLS	5 (1)	1	W	1
ASIMD multiply accumulate, Q-form	VMLA, VMLS	6 (2)	2	W W	1
ASIMD multiply accumulate long	VMLAL, VMLSL	5 (1)	1	W	1
ASIMD multiply accumulate saturating long	VQDMLAL, VQDMLSL	5 (1)	1	W	1
ASIMD multiply long	VMULL.S, VMULL.I, VMULL.P8, VQDMULL	5	1	W	
ASIMD pairwise add and accumulate	VPADAL	4 (1)	1	X	2
ASIMD shift accumulate	VSRA, VRSRA	4 (1)	1	X	2
ASIMD shift by immed, basic	VMOVL, VSHL, VSHLL, VSHR, VSHRN	3	1	X	
ASIMD shift by immed, complex	VQRSHRN, VQRSHRUN, VQSHL{U}, VQSHRN, VQSHRUN, VRSHR, VRSHRN	4	1	X	
ASIMD shift by register, basic, D-form	VSHL, VSLI, VSRI	3	1	X	
ASIMD shift by register, basic, Q-form	VSHL, VSLI, VSRI	4	2	X X	
ASIMD shift by register, complex, D-form	VQRSHL, VQSHL, VRSHL	4	1	X	
ASIMD shift by register, complex, Q-form	VQRSHL, VQSHL, VRSHL	5	2	X X	

Instruction Group	AArch64 Instructions	Exec Latency	Uop Count	Uop Types	Notes
ASIMD absolute diff, D-form	SABD, UABD	3	1	V	
ASIMD absolute diff, Q-form	SABD, UABD	3	1	V	
ASIMD absolute diff accum, D-form	SABA, UABA	4 (1)	1	X	2
ASIMD absolute diff accum, Q-form	SABA, UABA	5 (2)	2	X X	2
ASIMD absolute diff accum long	SABAL(2), UABAL(2)	4 (1)	1	X	2

Instruction Group	AArch64 Instructions	Exec Latency	Uop Count	Uop Types	Notes
ASIMD absolute diff long	SABDL, UABDL	3	1	V	
ASIMD arith, basic	ABS, ADD, ADDP, NEG, SADDL(2), SADDLP, SADDW(2), SHADD, SHSUB, SSUBL(2), SSUBW(2), SUB, UADDL(2), UADDLP, UADDW(2), UHADD, UHSUB, USUBW(2)	3	1	V	
ASIMD arith, complex	ADDHN(2), RADDHN(2), RSUBHN(2), SQABS, SQADD, SQNEG, SQSUB, SRHADD, SUBHN(2), SUQADD, UQADD, UQSUB, URHADD, USQADD	3	1	V	
ASIMD arith, reduce, 4H/4S	ADDV, SADDLV, UADDLV	4	1	X	
ASIMD arith, reduce, 8B/8H	ADDV, SADDLV, UADDLV	7	2	X V	
ASIMD arith, reduce, 16B	ADDV, SADDLV, UADDLV	8	2	X X	
ASIMD compare	CMEQ, CMGE, CMGT, CMHI, CMHS, CMLE, CMLT, CMTST	3	1	V	
ASIMD logical	AND, BIC, EOR, MOV, MVN, ORN, ORR	3	1	V	
ASIMD max/min, basic	SMAX, SMAXP, SMIN, SMINP, UMAX, UMAXP, UMIN, UMINP	3	1	V	
ASIMD max/min, reduce, 4H/4S	SMAXV, SMINV, UMAXV, UMINV	4	1	X	
ASIMD max/min, reduce, 8B/8H	SMAXV, SMINV, UMAXV, UMINV	7	2	X V	
ASIMD max/min, reduce, 16B	SMAXV, SMINV, UMAXV, UMINV	8	2	X X	
ASIMD multiply, D-form	MUL, PMUL, SQDMULH, SQRDMULH	5	1	W	
ASIMD multiply, Q-form	MUL, PMUL, SQDMULH, SQRDMULH	6	2	W W	
ASIMD multiply accumulate, D-form	MLA, MLS	5 (1)	1	W	
ASIMD multiply accumulate, Q-form	MLA, MLS	6 (2)	2	W W	
ASIMD multiply accumulate long	SMLAL(2), SMLSL(2), UMLAL(2), UMLSL(2)	5 (1)	1	W	1
ASIMD multiply accumulate saturating long	SQDMLAL(2), SQDMLSL(2)	5 (1)	1	W	1
ASIMD multiply long	SMULL(2), UMULL(2), SQDMULL(2)	5	1	W	
ASIMD multiply long	PMULL.1D, PMULL2.2D	3	1	W	3

Instruction Group	AArch64 Instructions	Exec Latency	Uop Count	Uop Types	Notes
ASIMD multiply long	PMULL.8B, PMULL2.16B	5	1	W	4
ASIMD pairwise add and accumulate	SADALP, UADALP	4 (1)	1	X	2
ASIMD shift accumulate	SRA, SRSRA, USRA, URSRA	4 (1)	1	X	2
ASIMD shift by immed, basic	SHL, SHLL(2), SHRN(2), SLI, SRI, SSHLL(2), SSHR, SXTL(2), USHLL(2), USHR, UXTL(2)	3	1	X	
ASIMD shift by immed, complex	RSHRN(2), SRSHR, SQSHL{U}, SQRSHRN(2), SQRSHRUN(2), SQSHRN(2), SQSHRUN(2), URSHR, UQSHL, UQRSHRN(2), UQSHRN(2)	4	1	X	
ASIMD shift by register, basic, D-form	SSHL, USHL	3	1	X	
ASIMD shift by register, basic, Q-form	SSHL, USHL	4	2	X X	
ASIMD shift by register, complex, D-form	SRSHL, SQRSHL, SQSHL, URSHL, UQRSHL, UQSHL	4	1	X	
ASIMD shift by register, complex, Q-form	SRSHL, SQRSHL, SQSHL, URSHL, UQRSHL, UQSHL	5	2	X X	

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

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

NOTE 3 – This category includes instructions of the form “PMULL Vd.1Q, Vn.1D, Vm.1D” and “PMULL2 Vd.1Q, Vn.2D, Vm.2D”

NOTE 4 – This category includes instructions of the form “PMULL Vd.8H, Vn.8B, Vm.8B” and “PMULL2 Vd.8H, Vn.16B, Vm.16B”

3.15 ASIMD Floating-Point Instructions

Instruction Group	AArch32 Instructions	Exec Latency	Uop Count	Uop Types	Notes
ASIMD FP absolute value	VABS	3	1	V	
ASIMD FP arith, D-form	VABD, VADD, VPADD, VSUB	5	1	V	
ASIMD FP arith, Q-form	VABD, VADD, VSUB	5	2	V V	
ASIMD FP compare, D-form	VACGE, VACGT, VACLE, VACLT, VCEQ, VCGE, VCGT, VCLE	5	1	V	
ASIMD FP compare, Q-form	VACGE, VACGT, VACLE, VACLT, VCEQ, VCGE, VCGT, VCLE	5	2	V V	

Instruction Group	AArch32 Instructions	Exec Latency	Uop Count	Uop Types	Notes
ASIMD FP convert, integer, D-form	VCVT	5	1	V	
ASIMD FP convert, integer, Q-form	VCVT	5	2	V V	
ASIMD FP convert, fixed, D-form	VCVT	5	1	V	
ASIMD FP convert, fixed, Q-form	VCVT	5	2	V V	
ASIMD FP convert, half-precision	VCVT	9	3	V V V	
ASIMD FP max/min, D-form	VMAX, VMIN, VPMAX, VPMIN	5	1	V	
ASIMD FP max/min, Q-form	VMAX, VMIN	5	2	V V	
ASIMD FP multiply, D-form, FZ	VMUL	5	1	V	
ASIMD FP multiply, D-form, no FZ	VMUL	6	1	V	
ASIMD FP multiply, Q-form, FZ	VMUL	5	2	V V	
ASIMD FP multiply, Q-form, no FZ	VMUL	6	2	V V	
ASIMD FP multiply accumulate, D-form, FZ	VMLA, VMLS, VFMA, VFMS	9 (4)	1	V	1
ASIMD FP multiply accumulate, D-form, no FZ	VMLA, VMLS, VFMA, VFMS	9 (4)	1	V	1
ASIMD FP multiply accumulate, Q-form, FZ	VMLA, VMLS, VFMA, VFMS	10 (4)	2	V V	1
ASIMD FP multiply accumulate, Q-form, no FZ	VMLA, VMLS, VFMA, VFMS	10 (4)	2	V V	1
ASIMD FP negate	VNEG	3	1	V	

Instruction Group	AArch64 Instructions	Exec Latency	Uop Count	Uop Types	Notes
ASIMD FP absolute value	FABS	3	1	V	
ASIMD FP arith, normal, D-form	FABD, FADD, FSUB	5	1	V	
ASIMD FP arith, normal, Q-form	FABD, FADD, FSUB	5	2	V V	
ASIMD FP arith, pairwise, D-form	FADDP	5	1	V	
ASIMD FP arith, pairwise, Q-form	FADDP	9	3	V V V	
ASIMD FP compare, D-form	FACGE, FACGT, FCMEQ, FCMGE, FCMGT, FCMLE, FCMLT	5	1	V	
ASIMD FP compare, Q-form	FACGE, FACGT, FCMEQ, FCMGE, FCMGT, FCMLE, FCMLT	5	2	V V	

Instruction Group	AArch64 Instructions	Exec Latency	Uop Count	Uop Types	Notes
ASIMD FP convert, long	FCVTL(2)	8	3	V V V	
ASIMD FP convert, narrow	FCVTN(2), FCVTXN(2)	8	3	V V V	
ASIMD FP convert, other, D-form	FCVTAS, VCVTAU, FCVTMS, FCVTMU, FCVTNS, FCVTNU, FCVTPS, FCVTPU, FCVTZS, FCVTZU, SCVT, UCVT	5	1	V	
ASIMD FP convert, other, Q-form	FCVTAS, VCVTAU, FCVTMS, FCVTMU, FCVTNS, FCVTNU, FCVTPS, FCVTPU, FCVTZS, FCVTZU, SCVT, UCVT	5	2	V V	
ASIMD FP divide, D-form, F32	FDIV	18	1	X	
ASIMD FP divide, Q-form, F32	FDIV	36	2	X X	
ASIMD FP divide, Q-form, F64	FDIV	64	2	X X	
ASIMD FP max/min, normal, D-form	FMAX, FMAXNM, FMIN, FMINNM	5	1	V	
ASIMD FP max/min, normal, Q-form	FMAX, FMAXNM, FMIN, FMINNM	5	2	V V	
ASIMD FP max/min, pairwise, D-form	FMAXP, FMAXNMP, FMINP, FMINNMP	5	1	V	
ASIMD FP max/min, pairwise, Q-form	FMAXP, FMAXNMP, FMINP, FMINNMP	9	3	V V V	
ASIMD FP max/min, reduce	FMAXV, FMAXNMV, FMINV, FMINNMV	10	3	V V V	
ASIMD FP multiply, D-form, FZ	FMUL, FMULX	5	1	V	
ASIMD FP multiply, D-form, no FZ	FMUL, FMULX	6	1	V	
ASIMD FP multiply, Q-form, FZ	FMUL, FMULX	5	2	V V	
ASIMD FP multiply, Q-form, no FZ	FMUL, FMULX	6	2	V V	
ASIMD FP multiply accumulate, D-form, FZ	FMLA, FMLS	9 (4)	1	V	1
ASIMD FP multiply accumulate, D-form, no FZ	FMLA, FMLS	9 (4)	1	V	1
ASIMD FP multiply accumulate, Q-form, FZ	FMLA, FMLS	10 (4)	2	V V	1
ASIMD FP multiply accumulate, Q-form, no FZ	FMLA, FMLS	10 (4)	2	V V	1
ASIMD FP negate	FNEG	3	1	V	
ASIMD FP round, D-form	FRINTA, FRINTI, FRINTM, FRINTN, FRINTP, FRINTX, FRINTZ	5	1	V	

Instruction Group	AArch64 Instructions	Exec Latency	Uop Count	Uop Types	Notes
ASIMD FP round, Q-form	FRINTA, FRINTI, FRINTM, FRINTN, FRINTP, FRINTX, FRINTZ	5	2	V V	

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

3.16 ASIMD Miscellaneous Instructions

Instruction Group	AArch32 Instructions	Exec Latency	Uop Count	Uop Types	Notes
ASIMD bitwise insert, D-form	VBIF, VBIT, VBSL	3	1	V	
ASIMD bitwise insert, Q-form	VBIF, VBIT, VBSL	3	2	V V	
ASIMD count, D-form	VCLS, VCLZ, VCNT	3	1	V	
ASIMD count, Q-form	VCLS, VCLZ, VCNT	3	1	V V	
ASIMD duplicate, core reg, D-form	VDUP	8	2	L V	
ASIMD duplicate, core reg, Q-form	VDUP	8	3	L V V	
ASIMD duplicate, scalar	VDUP	3	1	V	
ASIMD extract	VEXT	3	1	V	
ASIMD move, immed	VMOV	3	1	V	
ASIMD move, register	VMOV	3	1	V	
ASIMD move, narrowing	VMOVN	3	1	V	
ASIMD move, saturating	VQMOVN, VQMOVUN	4	1	X	
ASIMD reciprocal estimate, D-form	VRECPE, VRSQRTE	5	1	V	
ASIMD reciprocal estimate, Q-form	VRECPE, VRSQRTE	5	2	V V	
ASIMD reciprocal step, D-form, FZ	VRECPS, VRSQRTS	9	1	V	
ASIMD reciprocal step, D-form, no FZ	VRECPS, VRSQRTS	10	1	V	
ASIMD reciprocal step, Q-form, FZ	VRECPS, VRSQRTS	9	2	V V	
ASIMD reciprocal step, Q-form, no FZ	VRECPS, VRSQRTS	10	2	V V	
ASIMD reverse	VREV16, VREV32, VREV64	3	1	V	
ASIMD swap, D-form	VSWP	3	1	V	
ASIMD swap, Q-form	VSWP	3	2	V V	
ASIMD table lookup, 1 reg	VTBL, VTBX	3	1	V	
ASIMD table lookup, 2 reg	VTBL, VTBX	3	1	V	
ASIMD table lookup, 3 reg	VTBL, VTBX	6	2	V V	
ASIMD table lookup, 4 reg	VTBL, VTBX	6	2	V V	
ASIMD transfer, scalar to core reg	VMOV	6	2	L I	
ASIMD transfer, core reg to scalar	VMOV	8	2	L V	
ASIMD transpose, D-form	VTRN	3	1	V	

Instruction Group	AArch32 Instructions	Exec Latency	Uop Count	Uop Types	Notes
ASIMD transpose, Q-form	VTRN	3	2	V V	
ASIMD unzip/zip, D-form	VUZP, VZIP	3	1	V	
ASIMD unzip/zip, Q-form	VUZP, VZIP	6	3	V V V	

Instruction Group	AArch64 Instructions	Exec Latency	Uop Count	Uop Types	Notes
ASIMD bit reverse	RBIT	3	1	V	
ASIMD bitwise insert, D-form	BIF, BIT, BSL	3	1	V	
ASIMD bitwise insert, Q-form	BIF, BIT, BSL	3	2	V V	
ASIMD count, D-form	CLS, CLZ, CNT	3	1	V	
ASIMD count, Q-form	CLS, CLZ, CNT	3	1	V V	
ASIMD duplicate, gen reg, D-form	DUP	8	2	L V	
ASIMD duplicate, gen reg, Q-form	DUP	8	2	L V	
ASIMD duplicate, element	DUP	3	1	V	
ASIMD extract	EXT	3	1	V	
ASIMD insert, element to element	INS	3	1	V	
ASIMD move, immed	MOVI	3	1	V	
ASIMD move, register	FMOV	3	1	V	
ASIMD move, narrowing	XTN	3	1	V	
ASIMD move, saturating	SQXTN(2), SQXTUN(2), UQXTN(2)	4	1	X	
ASIMD reciprocal estimate, D-form	FRECPE, FRECPX, FRSQRTE, URECPE, URSQRTE	5	1	V	
ASIMD reciprocal estimate, Q-form	FRECPE, FRECPX, FRSQRTE, URECPE, URSQRTE	5	2	V V	
ASIMD reciprocal step, D-form, FZ	FRECPS, FRSQRTS	9	1	V	
ASIMD reciprocal step, D-form, no FZ	FRECPS, FRSQRTS	10	1	V	
ASIMD reciprocal step, Q-form, FZ	FRECPS, FRSQRTS	9	2	V V	
ASIMD reciprocal step, Q-form, no FZ	FRECPS, FRSQRTS	10	2	V V	
ASIMD reverse	REV16, REV32, REV64	3	1	V	
ASIMD table lookup, D-form	TBL, TBX	3xN	N	VxN	
ASIMD table lookup, Q-form	TBL, TBX	3xN + 3	2xN + 1	2xVxN V	
ASIMD transfer, element to gen reg	SMOV, UMOV	6	2	L I	
ASIMD transfer, gen reg to element	INS	8	2	L V	
ASIMD transpose, D-form	TRN1, TRN2	3	1	V	
ASIMD transpose, Q-form	TRN1, TRN2	3	1	V	
ASIMD unzip/zip, D-form	UZP1, UZP2, ZIP1, ZIP2	3	1	V	
ASIMD unzip/zip, Q-form	UZP1, UZP2, ZIP1, ZIP2	6	3	V V V	

3.17 ASIMD Load Instructions

The latencies shown assume the memory access hits in the Level 1 Data Cache. Compared to standard loads, an extra cycle is required to forward results to FP/ASIMD pipelines.

Instruction Group	AArch32 Instructions	Exec Latency	Uop Count	Uop Types	Notes
ASIMD load, 1 element, multiple, 1 reg	VLD1	5	1	L	
ASIMD load, 1 element, multiple, 2 reg	VLD1	5	1	L	
ASIMD load, 1 element, multiple, 3 reg	VLD1	6	2	L L	
ASIMD load, 1 element, multiple, 4 reg	VLD1	6	2	L L	
ASIMD load, 1 element, one lane	VLD1	8	2	L V	
ASIMD load, 1 element, all lanes	VLD1	8	2	L V	
ASIMD load, 2 element, multiple, 2 reg	VLD2	8	2	L V	
ASIMD load, 2 element, multiple, 4 reg	VLD2	9	4	L L V V	
ASIMD load, 2 element, one lane, size 32	VLD2	8	2	L V	
ASIMD load, 2 element, one lane, size 8/16	VLD2	8	3	L V V	
ASIMD load, 2 element, all lanes	VLD2	8	2	L V	
ASIMD load, 3 element, multiple, 3 reg	VLD3	9	4	L L V V	
ASIMD load, 3 element, one lane, size 32	VLD3	8	3	L V V	
ASIMD load, 3 element, one lane, size 8/16	VLD3	9	4	L V V V	
ASIMD load, 3 element, all lanes	VLD3	8	3	L V V	
ASIMD load, 4 element, multiple, 4 reg	VLD4	9	4	L L V V	
ASIMD load, 4 element, one lane, size 32	VLD4	8	3	L V V	
ASIMD load, 4 element, one lane, size 8/16	VLD4	9	5	L V V V V	
ASIMD load, 4 element, all lanes	VLD4	8	3	L V V	
(ASIMD load, writeback form)			+1	+I	1

Instruction Group	AArch64 Instructions	Exec Latency	Uop Count	Uop Types	Notes
ASIMD load, 1 element, multiple, 1 reg, D-form	LD1	5	1	L	
ASIMD load, 1 element, multiple, 1 reg, Q-form	LD1	5	1	L	
ASIMD load, 1 element, multiple, 2 reg, D-form	LD1	5	1	L	
ASIMD load, 1 element, multiple, 2 reg, Q-form	LD1	6	2	L L	
ASIMD load, 1 element, multiple, 3 reg, D-form	LD1	6	2	L L	
ASIMD load, 1 element, multiple, 3 reg, Q-form	LD1	7	3	L L L	
ASIMD load, 1 element, multiple, 4 reg, D-form	LD1	6	2	L L	
ASIMD load, 1 element, multiple, 4 reg, Q-form	LD1	8	4	L L L L	
ASIMD load, 1 element, one lane, B/H/S	LD1	8	2	L V	
ASIMD load, 1 element, one lane, D	LD1	5	1	L	

Instruction Group	AArch64 Instructions	Exec Latency	Uop Count	Uop Types	Notes
ASIMD load, 1 element, all lanes, D-form, B/H/S	LD1R	8	2	L V	
ASIMD load, 1 element, all lanes, D-form, D	LD1R	5	1	L	
ASIMD load, 1 element, all lanes, Q-form	LD1R	8	2	L V	
ASIMD load, 2 element, multiple, D-form, B/H/S	LD2	8	2	L V	
ASIMD load, 2 element, multiple, Q-form, B/H/S	LD2	9	4	L L V V	
ASIMD load, 2 element, multiple, Q-form, D	LD2	6	2	L L	
ASIMD load, 2 element, one lane, B/H	LD2	8	3	L V V	
ASIMD load, 2 element, one lane, S	LD2	8	2	L V	
ASIMD load, 2 element, one lane, D	LD2	6	2	L L	
ASIMD load, 2 element, all lanes, D-form, B/H/S	LD2R	8	2	L V	
ASIMD load, 2 element, all lanes, D-form, D	LD2R	5	1	L	
ASIMD load, 2 element, all lanes, Q-form	LD2R	8	3	L V V	
ASIMD load, 3 element, multiple, D-form, B/H/S	LD3	9	4	L L V V	
ASIMD load, 3 element, multiple, D-form, D	LD3	8	4	L L L L	
ASIMD load, 3 element, multiple, Q-form, B/H/S	LD3	10	7	LL VV L VV	
ASIMD load, 3 element, one lane, Q-form, D	LD3	6	2	L L	
ASIMD load, 3 element, one lane, B/H	LD3	9	4	L V V V	
ASIMD load, 3 element, one lane, S	LD3	8	3	L V V	
ASIMD load, 3 element, all lanes, D-form, B/H/S	LD3R	8	3	L V V	
ASIMD load, 3 element, all lanes, D-form, D	LD3R	6	2	L L	
ASIMD load, 3 element, all lanes, Q-form, B/H/S	LD3R	9	4	L V V V	
ASIMD load, 3 element, all lanes, Q-form, D	LD3R	9	5	L L V V V	
ASIMD load, 4 element, multiple, D-form, B/H/S	LD4	9	4	L L V V	
ASIMD load, 4 element, multiple, Q-form, B/H/S	LD4	11	8	LL VV LL VV	
ASIMD load, 4 element, multiple, Q-form, D	LD4	8	4	L L L L	
ASIMD load, 4 element, one lane, B/H	LD4	9	5	L V V V V	
ASIMD load, 4 element, one lane, S	LD4	8	3	L V V	
ASIMD load, 4 element, one lane, D	LD4	6	2	L L	
ASIMD load, 4 element, all lanes, D-form, B/H/S	LD4R	8	3	L V V	
ASIMD load, 4 element, all lanes, D-form, D	LD4R	6	2	L L	
ASIMD load, 4 element, all lanes, Q-form, B/H/S	LD4R	9	5	L V V V V	
ASIMD load, 4 element, all lanes, Q-form, D	LD4R	9	6	L L V V V V	
(ASIMD load, writeback form)			+1	+I	1

NOTE 1 – Writeback forms of load instructions require an extra address update uop. This update is typically performed in parallel with or prior to the load uop (address update latency shown in parentheses).

3.18 ASIMD Store Instructions

Stores uops 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	Uop Count	Uop Types	Notes
ASIMD store, 1 element, multiple, 1 reg	VST1	1	1	S	
ASIMD store, 1 element, multiple, 2 reg	VST1	2	2	S S	
ASIMD store, 1 element, multiple, 3 reg	VST1	3	3	S S S	
ASIMD store, 1 element, multiple, 4 reg	VST1	4	4	S S S S	
ASIMD store, 1 element, one lane	VST1	3	2	V S	
ASIMD store, 2 element, multiple, 2 reg	VST2	3	3	V S S	
ASIMD store, 2 element, multiple, 4 reg	VST2	4	6	VSS VSS	
ASIMD store, 2 element, one lane	VST2	3	2	V S	
ASIMD store, 3 element, multiple, 3 reg	VST3	3	5	VSS VS	
ASIMD store, 3 element, one lane, size 32	VST3	3	3	V S S	
ASIMD store, 3 element, one lane, size 8/16	VST3	3	2	V S	
ASIMD store, 4 element, multiple, 4 reg	VST4	4	6	VSS VSS	
ASIMD store, 4 element, one lane, size 32	VST4	3	3	V S S	
ASIMD store, 4 element, one lane, size 8/16	VST4	3	2	V S	
(ASIMD store, writeback form)			+1	+I	1

Instruction Group	AArch64 Instructions	Exec Latency	Uop Count	Uop Types	Notes
ASIMD store, 1 element, multiple, 1 reg, D-form	ST1	1	1	S	
ASIMD store, 1 element, multiple, 1 reg, Q-form	ST1	2	2	S S	
ASIMD store, 1 element, multiple, 2 reg, D-form	ST1	2	2	S S	
ASIMD store, 1 element, multiple, 2 reg, Q-form	ST1	4	4	S S S S	
ASIMD store, 1 element, multiple, 3 reg, D-form	ST1	3	3	S S S	
ASIMD store, 1 element, multiple, 3 reg, Q-form	ST1	6	6	S S S S S S	
ASIMD store, 1 element, multiple, 4 reg, D-form	ST1	4	4	S S S S	
ASIMD store, 1 element, multiple, 4 reg, Q-form	ST1	8	8	SSSS SSSS	
ASIMD store, 1 element, one lane, B/H/S	ST1	1	1	S	
ASIMD store, 1 element, one lane, D	ST1	3	2	V S	
ASIMD store, 2 element, multiple, D-form, B/H/S	ST2	3	3	V S S	
ASIMD store, 2 element, multiple, Q-form, B/H/S	ST2	4	6	V S S V S S	
ASIMD store, 2 element, multiple, Q-form, D	ST2	4	4	S S S S	
ASIMD store, 2 element, one lane, B/H/S	ST2	3	2	V S	
ASIMD store, 2 element, one lane, D	ST2	2	2	S S	
ASIMD store, 3 element, multiple, D-form, B/H/S	ST3	3	5	V S S V S	
ASIMD store, 3 element, multiple, Q-form, B/H/S	ST3	6	10	VSSVVSSVSS	
ASIMD store, 3 element, multiple, Q-form, D	ST3	6	6	S S S S S S	
ASIMD store, 3 element, one lane, B/H	ST3	3	2	V S	
ASIMD store, 3 element, one lane, S	ST3	3	3	V S S	

Instruction Group	AArch64 Instructions	Exec Latency	Uop Count	Uop Types	Notes
ASIMD store, 3 element, one lane, D	ST3	3	3	S S S	
ASIMD store, 4 element, multiple, D-form, B/H/S	ST4	4	6	VSS VSS	
ASIMD store, 4 element, multiple, Q-form, B/H/S	ST4	8	12	VSSVSSVSSVSS	
ASIMD store, 4 element, multiple, Q-form, D	ST4	8	8	SSSSSSSS	
ASIMD store, 4 element, one lane, B/H	ST4	3	2	V S	
ASIMD store, 4 element, one lane, S	ST4	3	3	V S S	
ASIMD store, 4 element, one lane, D	ST4	4	4	S S S S	
(ASIMD store, writeback form)			+1	+I	1

NOTE 1 – Writeback forms of store instructions require an extra address update uop. This update is typically performed in parallel with or prior to the store uop (address update latency shown in parentheses).

3.19 Cryptography Extensions

Instruction Group	AArch32 Instructions	Exec Latency	Uop Count	Uop Types	Notes
Crypto AES ops	AESD, AESE, AESIMC, AESMC	3	1	W	1
Crypto SHA1 xor ops	SHA1SU0	6	2	V V	
Crypto SHA1 fast ops	SHA1H, SHA1SU1	3	1	W	
Crypto SHA1 slow ops	SHA1C, SHA1M, SHA1P	6	2	W W	
Crypto SHA256 fast ops	SHA256SU0	3	1	W	
Crypto SHA256 slow ops	SHA256H, SHA256H2, SHA256SU1	6	2	W W	
Long polynomial multiply ops	VMULL.P64	3	1	W	
CRC checksum ops	CRC32, CRC32C	3	1	W	

Instruction Group	AArch64 Instructions	Exec Latency	Uop Count	Uop Types	Notes
Crypto AES ops	AESD, AESE, AESIMC, AESMC	3	1	W	1
Crypto SHA1 xor osp	SHA1SU0	6	2	V V	
Crypto SHA1 schedule acceleration ops	SHA1H, SHA1SU1	3	1	W	
Crypto SHA1 hash acceleration ops	SHA1C, SHA1M, SHA1P	6	2	W W	
Crypto SHA256 schedule acceleration op (1 uop)	SHA256SU0	3	1	W	
Crypto SHA256 schedule acceleration op (2 uops)	SHA256SU1	6	2	W W	
Crypto SHA256 hash acceleration ops	SHA256H, SHA256H2	6	2	W W	
CRC checksum ops	CRC32, CRC32C	3	1	W	

NOTE 1 – In Cortex-A57 r0p1 and later, AESE/AESMC and AESD/AESIMC instruction pairs can be issued together as a single W uop with an execution latency of 3 cycles as long as the two instructions are sequential in memory. See Section 4.10 for additional details.

4 SPECIAL CONSIDERATIONS

4.1 Conditional Execution

The ARMv8 architecture allows many types of A32 instructions to be conditionally executed based upon condition flags (N, Z, C, V). If the condition flags satisfy a condition specified in the instruction encoding, an instruction has its normal effect. If the flags do not satisfy this condition, the instruction acts as a NOP.

This leads to conditional register writes for most types of conditional instructions. To support these in an out-of-order processor, the conditional operation must be treated like the combination of a normal operation and a conditional select operation. For example, consider the following conditional ADD instruction.

```
ADDEQ R1, R2, R3              // R1 = EQ ? R2 + R3 : R1
```

Although handled by a single uop, that uop performs two simultaneous operations.

```
ADD    RX, R2, R3
SELEQ R1, RX, R1
```

One side-effect is that the resulting uop requires the old value of R1 as an operand. A second side-effect is that all subsequent consumers of R1 are dependent upon this operation, regardless of the state of the condition flags.

These effects should be taken into account when considering using conditional execution for long-latency operations. The overheads of conditional execution may begin to outweigh the benefits. Consider the following example.

```
MULEQ R1, R2, R3
MULNE R1, R2, R4
```

For this pair of instructions, the second multiply is dependent upon the result of the first multiply. The combined latency is six cycles, rather than the three cycles required for a single multiply. Of course, the penalty for a branch mispredict is greater still. So if the condition is easily predictable (by the branch predictor), conditional execution leads to a performance loss. But if the condition is not easily predictable, conditional execution may lead to a performance gain. In general, we recommend against using conditional forms of long-latency instructions.

4.2 Conditional ASIMD

Conditional execution is architecturally possible for ASIMD instructions in Thumb state using IT blocks. However, this type of encoding is considered abnormal and is not recommended for Cortex-A57. It will likely perform worse than the equivalent unconditional encodings.

Due to the number of operands required for most ASIMD instructions, it is not possible to generate uops with fused conditional select operations. Instead, ASIMD instructions are decoded assuming the condition check will pass. If the condition check fails at execute time, a pipeline flush occurs and the instruction is re-decoded as a NOP.

4.3 Register Forwarding Hazards

The Cortex-A57 processor employs register renaming in order to process instructions speculatively and out-of-order. Each uop's destination registers are assigned unique register tags. Similar register tags are retrieved from register rename tables for each uop's source registers. These tags allow register dependences to be tracked and speculative results to be forwarded between uops.

The ARM architecture allows FP instructions to read and write 32-bit S-registers. Each S-register corresponds to one half (upper or lower) of an overlayed 64-bit D-register. Register forwarding hazards may occur when one uop reads a D-register operand that has recently been written as one or more S-register results. Consider the following abnormal scenario.

```
VMOV  S0,R0
VMOV  S1,R1
VADD  D2, D1, D0
```

The first two instructions write S0 and S1, which correspond to the bottom and top halves of D0. The third instruction then requires D0 as its second operand. Multiple register tags would need to be associated with the second operand in order to independently track and forward each half of the 64-bit D0 value. But the hardware supports only one register tag per source operand. If this scenario occurs, it creates a "single-to-double" register forwarding hazard. It results in the consuming uop stalling at the dispatch stage until all previous uops retire.

Instructions with 128-bit Q-register operands are decoded into uops that read and write the equivalent D-register operands. So an instruction that reads a Q-register will be susceptible to the same hazards against S-register-producing instructions as if it had read the corresponding pair of D-registers.

The processor avoids hazards for certain cases, by decoding D-register operands into two separate S-register operands. The following rules clarify when a hazard can occur.

- The producer writes an S-register (not a D[x] scalar)
- The consumer reads an overlapping D-register (not a D[x] scalar, not an implicit operand due to conditional execution)
- The consumer is a FP/ASIMD uop (not a store uop)

To avoid unnecessary hazards, be sure to use D[x] scalar writes when populating registers prior to ASIMD operations. For example, either of the following instruction forms would safely prevent a subsequent hazard.

```
VLD1.32     Dd[x], [address]
VMOV.32     Dd[x], Rt
```

The Performance Monitor Unit (PMU) in Cortex-A57 may be used to determine when register forwarding hazards are actually occuring. The implementation defined PMU event number 0x12C (DISP_SWDW_STALL) has been assigned to count the number of cycles spent stalling due to these hazards.

4.4 Load/Store Throughput

The Cortex-A57 processor includes separate load and store pipelines, which allow it to execute one load uop and one store uop every cycle. Each load uop may read up to sixteen bytes and two registers. Each store uop may write up to eight bytes and two registers. Loads may be issued out of order with respect to other loads or stores, and stores may be issued out-of-order with respect to other loads.

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 discrete, non-writeback forms of load and store instructions (such as LDRD and STRD) and alternate them so that one load and one store operation may be performed each cycle. Avoid load/store multiple instruction encodings (such as LDM and STM), which lead to separated bursts of load and store uops.

The following example shows a recommended instruction sequence for a long memory copy in AArch32 state:

```
Loop_start:
      SUBS  r2,r2,#64
      LDRD  r3,r4,[r1,#0]
      STRD  r3,r4,[r0,#0]
      LDRD  r3,r4,[r1,#8]
      STRD  r3,r4,[r0,#8]
      LDRD  r3,r4,[r1,#16]
      STRD  r3,r4,[r0,#16]
      LDRD  r3,r4,[r1,#24]
      STRD  r3,r4,[r0,#24]
      LDRD  r3,r4,[r1,#32]
      STRD  r3,r4,[r0,#32]
      LDRD  r3,r4,[r1,#40]
      STRD  r3,r4,[r0,#40]
      LDRD  r3,r4,[r1,#48]
      STRD  r3,r4,[r0,#48]
      LDRD  r3,r4,[r1,#56]
      STRD  r3,r4,[r0,#56]
      ADD   r1,r1,#64
      ADD   r0,r0,#64
      BGT   Loop_start
```

A recommended copy routine for AArch64 would look similar to the sequence above, but would use LDP/STP instructions.

4.5 Load/Store Alignment

The ARM architecture allows many types of load and store accesses to be arbitrarily aligned. The Cortex-A57 processor handles most unaligned accesses without performance penalties. However, there are cases which require extra cache or memory accesses. These are summarized below.

- Load operations that cross a cache line boundary (64 bytes) require two load issue cycles instead of one
- Store operations that cross a cache line boundary (64 bytes) require two store issue cycles instead of one
- Store operations that cross a quadword boundary (16 bytes) require two store commit cycles instead of one

For best-case performance, one should avoid unaligned accesses where possible.

4.6 Branch Alignment

Branch instruction and branch target instruction alignment can affect performance. For best-case performance, consider the following guidelines.

- Try not to pack more than two taken branches into the same quadword (16 bytes) of instruction memory. The branch predictor can track and predict up to two branches per quadword. This applies for both ARM and Thumb states.
- Consider aligning subroutine entry points and branch targets to quadword boundaries. This will ensure that the subsequent fetch can retrieve four (or a full quadword's worth of) instructions. Of course, code density trade-offs should also be considered.

4.7 Setting Condition Flags

The ARM instruction set includes instruction forms that set the condition flags. In addition to compares, many types of data processing operations may do this as a side-effect. Excessive use of *setflags* forms may result in performance degradation.

Condition flag results are assigned a temporary rename register to hold the speculative result until it may be committed to the architected state. Up to 24 speculative condition flag results can be tracked at one time. Once that limit is reached, subsequent instructions that set condition flags must stall at the rename stage until prior instructions have retired.

When using the Thumb instruction set, special attention should be given to the use of 16-bit instruction forms. Many of those (moves, adds, shifts, etc) automatically set the condition flags. For best performance, consider using the 32-bit encodings instead. The 32-bit encodings include forms that do not set the condition flags.

4.8 Floating-Point Multiplies

As described in chapter 2 of this document, the Cortex-A57 processor implements two floating point execution pipelines. Normally, uops are steered to one pipeline or the other based upon a load-balancing hardware mechanism. However, a different steering policy applies for FP/ASIMD floating-point multiply and multiply-accumulate uops. To facilitate accumulator forwarding, all such uops writing an even-numbered D-register are steered to FP/ASIMD pipeline 0 and those writing an odd-numbered D-register are steered to FP/ASIMD pipeline 1. Quadword forms of these instructions generate two uops, one writing the lower half of the quadword register (an even D-register) and the other writing the upper half (an odd D-register). This steering policy works well for quadword ASIMD sequences of dependent multiply-accumulate operations, but can artificially limit the execution bandwidth for other sequences.

For best-case performance, try to use a balanced mix of odd and even D-registers when performing a critical sequence of independent, non-quadword FP/ASIMD floating-point multiply or multiply-accumulate operations.

4.9 Special Register Access

The Cortex-A57 processor performs register renaming for general purpose registers to enable speculative and out-of-order instruction execution. But most special-purpose registers are not renamed. Instructions that read or write non-renamed registers are subjected to one or more of the following additional execution constraints.

- Non-Speculative Execution – Instructions may only execute non-speculatively.
- In-Order Execution – Instructions must execute in-order with respect to other similar instructions or in some cases all instructions.
- Flush Side-Effects – Instructions trigger a flush side-effect after executing for synchronization.

The table below summarizes various special instructions and the associated execution constraints or side-effects.

Instructions	Forms	Non-Speculative	In-Order	Flush Side-Effect	Notes
ISB		Yes	Yes	Yes	
CPS		Yes	Yes	Yes	
SETEND		Yes	Yes	Yes	
MRS (read)	APSR, CPSR	Yes	Yes	No	
MRS (read)	SPSR	No	Yes	No	
MSR (write)	ASPR_nzcvq, CPSR_f	No	No	No	2, 3
MSR (write)	APSR, CPSR other	Yes	Yes	Yes	
MSR (write)	SPSR	Yes	Yes	No	
VMRS (read)	FPSCR to APSR_nzcv	No	No	No	2
VMRS (read)	Other	Yes	Yes	No	
VMSR (write)		Yes	Yes	Yes	
VMSR (write)	FPSCR, changing only NZCV	Yes	Yes	No	5
MRC (read)		Some	Yes	No	2, 4
MCR (write)		Yes	Yes	Some	4

NOTE 1 – Conditional forms of the above instructions for which the condition is not satisfied will not access special registers or trigger flush side-effects.

NOTE 2 – Conditional forms of the above instructions are always executed non-speculatively and in-order to properly resolve the condition.

NOTE 3 – MSR instructions that write APSR_nzcvq generate a separate uop to write the Q bit. That uop executes non-speculatively and in-order. But the main uop, which writes the NZCV bits, executes as shown in the table above.

NOTE 4 – A subset of MCR instructions must be executed non-speculatively for various reasons. A subset of MRC instructions trigger flush side-effects for synchronization. Those subsets are not documented here.

NOTE 5 – Special hardware recognizes an FPSCR write with a data value that only affects the NZCV flags, avoiding the flush side-effect typical with other VMSR writes.

4.10 AES Encryption/Decryption

Cortex-A57 can issue one AESE/AESMC/AESD/AESIMC instruction every cycle (fully pipelined) with an execution latency of three cycles (see Section 3.19). This means encryption or decryption for at least three data chunks should be interleaved for maximum performance:

```
AESE  data0, key0
AESMC data0, data0
AESE  data1, key0
AESMC data1, data1
AESE  data2, key0
AESMC data2, data2
AESE  data0, key1
AESMC data0, data0
...
```

In Cortex-A57 r0p1 and later revisions, AESE/AESMC and AESD/AESIMC instruction pairs can be issued together as a single uop as long as the two instructions appear sequentially in the executed instruction stream. Therefore it is important to ensure that these instructions come in pairs in AES encryption/decryption loops, as shown in the code segment above.