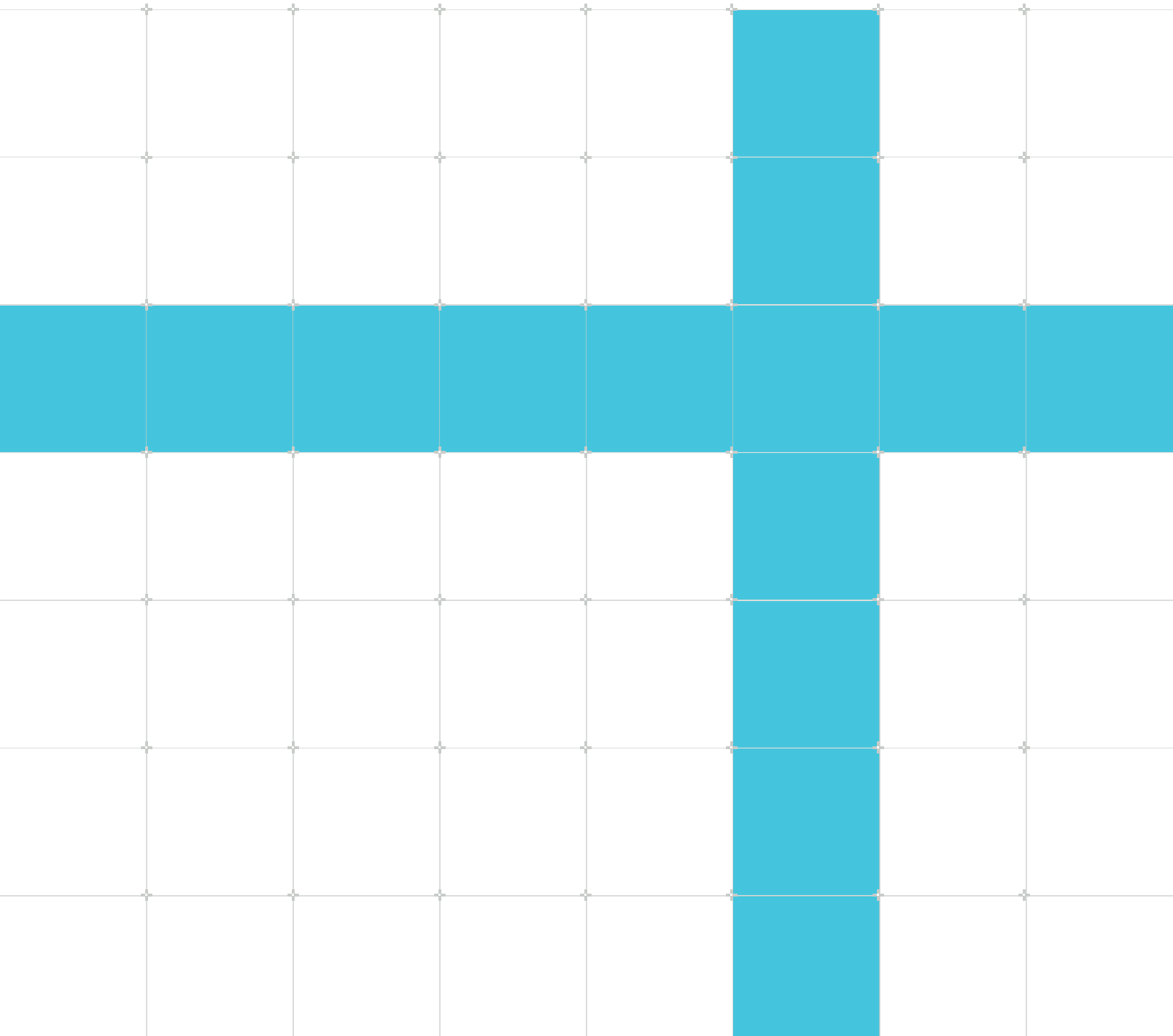
Arm® Cortex®-M85 Processor

Revision: r1p0

Software Optimization Guide

Issue 01
107950_0100_01_en



Arm® Cortex®-M85 Processor

Software Optimization Guide

Release Information

Document history

Issue	Date	Confidentiality	Change
0100-01	26 April 2023	Non-Confidential	First release for r1p0

Proprietary Notice

and regulations to assure that this document or any portion thereof is not exported, directly or indirectly, in violation of such export laws. Use of the word “partner” in reference to Arm’s customers is not intended to create or refer to any partnership relationship with any other company. Arm may make changes to this document at any time and without notice.

This document may be translated into other languages for convenience, and you agree that if there is any conflict between the English version of this document and any translation, the terms of the English version of the Agreement shall prevail.

Arm Limited. Company 02557590 registered in England.

110 Fulbourn Road, Cambridge, England CB1 9NJ.

(LES-PRE-20349|version 21.0)

Confidentiality Status

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

Unrestricted Access is an Arm internal classification.

Product Status

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

Feedback

Arm welcomes feedback on this product and its documentation. To provide feedback on the product, create a ticket on https://support.developer.arm.com.

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

Inclusive language commitment

Arm values inclusive communities. Arm recognizes that we and our industry have used language that can be offensive. Arm strives to lead the industry and create change.

This document includes language that can be offensive. We will replace this language in a future issue of this document.

To report offensive language in this document, email terms@arm.com.

Contents

1. Introduction....7
1.1 Conventions....7
1.2 Other information....8
1.3 Useful resources....8

2. Overview....10
2.1 Cortex®-M85 processor overview....10
2.2 Pipeline overview....11

3. Instruction latencies....13
3.1 Instruction tables....13
3.2 Branch instructions....14
3.3 Arithmetic and logical instructions....15
3.4 Move and shift instructions....22
3.5 Divide and multiply instructions....22
3.6 Load instructions....24
3.7 Store instructions....26
3.8 Miscellaneous instructions....28
3.9 FP data processing instructions....29
3.10 MVE integer vector instructions....31
3.11 MVE integer scalar instructions....37
3.12 MVE FP instructions....38
3.13 MVE miscellaneous instructions....40
3.14 MVE load instructions....41
3.15 MVE store instructions....42

4. General behaviors....43
4.1 Overlapping MVE execution....43
4.2 Data dependency special cases....43
4.3 Memory banking....44
4.4 Load and store address alignment....44
4.5 Branch instruction alignment....45
4.6 Data accesses to ITCM....45

4.7 Hardware data prefetch....45
4.8 Software prefetch instructions....45
4.9 Transient memory hint....45
4.10 Optimizing MVE execution....46

A. Revisions....50
A.1 Revisions....50

1. Introduction

1.1 Conventions

The following subsections describe conventions used in Arm documents.

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: developer.arm.com/glossary.

Convention	Use
italic	Citations.
bold	Terms in descriptive lists, where appropriate.
`monospace`	Text that you can enter at the keyboard, such as commands, file and program names, and source code.
`monospace underline`	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	Terms that have specific technical meanings as defined in the *Arm® Glossary*. For example, **IMPLEMENTATION DEFINED**, **IMPLEMENTATION SPECIFIC**, **UNKNOWN**, and **UNPREDICTABLE**.

Recommendations. Not following these recommendations might lead to system failure or damage.

Requirements for the system. Not following these requirements might result in system failure or damage.

Requirements for the system. Not following these requirements will result in system failure or damage.

An important piece of information that needs your attention.

A useful tip that might make it easier, better or faster to perform a task.

A reminder of something important that relates to the information you are reading.

1.2 Other information

See the Arm® website for other relevant information.

- Arm® Developer.
- Arm® Documentation.
- Technical Support.
- Arm® Glossary.

1.3 Useful resources

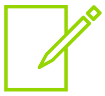
This document contains information that is specific to this product. See the following resources for other useful information.

Access to Arm documents depends on their confidentiality:

- Non-Confidential documents are available at developer.arm.com/documentation. Each document link in the following tables goes to the online version of the document.
- Confidential documents are available to licensees only through the product package.

Arm product resources	Document ID	Confidentiality
Arm® Cortex®-M85 Processor Devices Generic User Guide	101928	Non-Confidential
Arm® Cortex®-M85 Processor Technical Reference Manual	101924	Non-Confidential
Getting started with Armv8.1-M based processor: software development hints and tips	-	Non-Confidential

Arm architecture and specifications	Document ID	Confidentiality
Arm®v8-M Architecture Reference Manual	DDI 0553	Non-Confidential
Arm® Helium Technology M-Profile Vector Extension (MVE) for Arm Cortex-M Processors Reference Book	ISBN: 978-1-911531-23-4	Non-Confidential
Helium Programmer's Guide: Introduction to Helium	102102	Non-Confidential


Note

Arm tests its PDFs only in Adobe Acrobat and Acrobat Reader. Arm cannot guarantee the quality of its documents when used with any other PDF reader.

Adobe PDF reader products can be downloaded at http://www.adobe.com.

2. Overview

This document provides guidelines on generating optimal sequence of instructions while writing the assembly code for the Cortex®-M85 processor.

2.1 Cortex®-M85 processor overview

The Cortex®-M85 processor is a fully synthesizable high-performance microcontroller class processor that implements the Arm®v8.1-M Mainline architecture which includes support for the *M-profile Vector Extension* (MVE). The processor also supports previous Arm®v8-M architectural features.

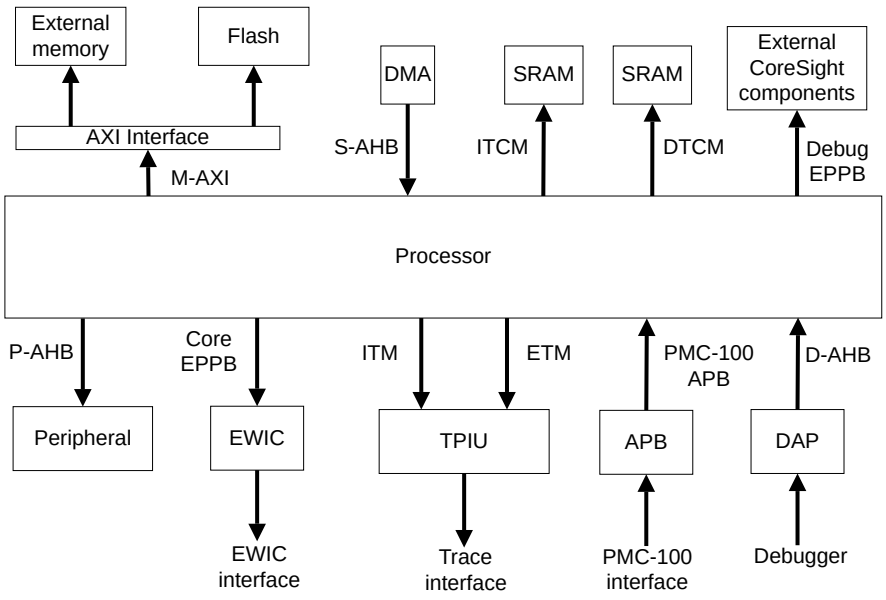
The design is focused on compute applications such as *Digital Signal Processing* (DSP) and machine learning. The Cortex®-M85 processor is energy efficient and achieves high compute performance across scalar and vector operations while maintaining low power consumption.

The processor can be configured to include *Dual-Core Lock-Step* (DCLS) functionality, which implements a redundant copy of most of the processor logic.

To support *Arm Custom Instructions* (ACIs), the processor includes optional *Custom Datapath Extension* (CDE) modules, which are embedded inside the logic. These modules are used to execute user-defined instructions that work on general-purpose integer, floating point, and MVE registers.

The following figure shows the Cortex®-M85 processor in a typical system.

Figure 2-1: Example processor system



Note

For additional information on the interface, see *Arm® Cortex®-M85 Processor Technical Reference Manual*.

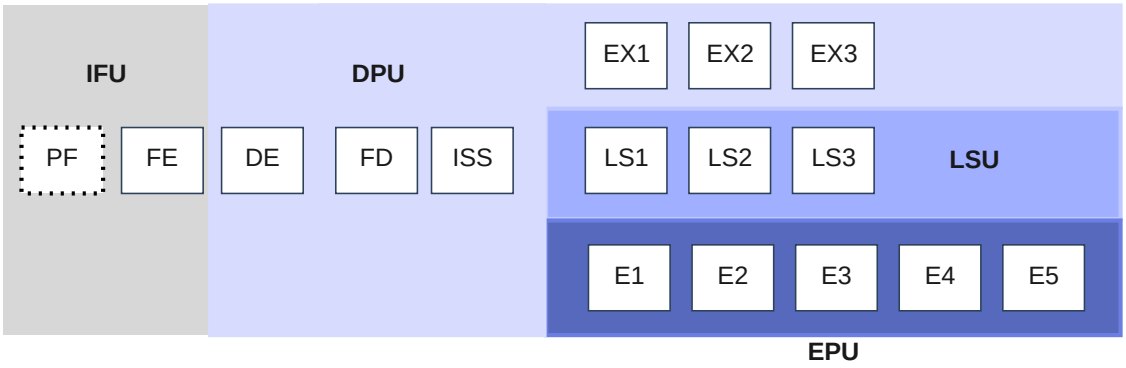
2.2 Pipeline overview

The Cortex®-M85 processor implements a 7-stage in-order integer pipeline and a 9-stage pipeline for *Floating Point* (FP) and *M-Profile Vector Extension* (MVE) instructions.

The following diagram describes the high-level Cortex®-M85 processor pipeline. The pipeline can be partitioned to four units:

- *Instruction Fetch Unit* (IFU)
- *Data Processing Unit* (DPU)
- *Load-Store Unit* (LSU)
- *Extension Processing Unit* (EPU)

Figure 2-2: Cortex®-M85 processor Core and EPU pipeline structure



The IFU fetches instructions from memory and provides them to the DPU to be issued to execution pipelines in the DPU, LSU, and EPU, which stall in unison as needed to maintain order. The pipeline stages are described in the following table.

Table 2-1: Pipeline stages

Stage	Description
PF	IFU prefetch address stage The PF stage presents instruction addresses to the ITCM and the I-cache. It fetches 64 bits per cycle, allowing sustained decode of two instructions per cycle later in the pipeline even for 32-bit instructions. This stage consists of branch prediction logic, target storage for previously predicted branches, and, loop and branch tracking register. Successful predictions remove the penalty associated with taken branches and allow instructions at predicted branch targets to follow immediately after branches. PF is shown with dashed lines because its inputs are unregistered. It receives branch resolution outcomes and branch targets for use as fetch addresses signaled from EX stages. In a nominal branch misprediction scenario detected in EX2 and provided directly into PF from EX3, the branch latency is 7 cycles. In practice, performance features throughout the pipeline often reduce the penalty by one or more cycles.
FE	IFU fetch data stage The IFU receives instruction fetch data from ITCM or I-cache in FE and buffers it for use by the DPU. Data is buffered in amounts of 64 bits unless fetching a branch target not aligned to a 64-bit boundary in memory.
DE	IFU or DPU delineate stage The DPU reads buffered fetch data in DE and marks out individual instructions for use in the FD stage.
FD	DPU predecode stage The FD stage uses processes individual instructions and prepares them for the ISS stage.
ISS	DPU issue stage ISS completes instruction decode, obtains available instruction operands from the integer register file in-flight in the execution and load/store pipelines below. Instructions which cannot be issued stall in ISS.
EX1-3	DPU integer execution stages The EX stages are populated with ALU, shifter, multiply/divide, and branch resolution resources which process issued integer instructions and branch instructions.
LS1-3	Load-store stages Address calculations in LS1 determine whether a memory access is made to the L1 data cache and M-AXI interface, TCM, the P-AHB interface, internal peripherals in the PPB memory region or the EPPB interfaces. The LSU can accommodate two load instructions, two store instructions, or one load and one store instruction per cycle, with a total of 64 bits of available load bandwidth and 64 bits of available store bandwidth per cycle. Word-sized load data returns from TCM or L1 data cache in two cycles, three for sub-word data. Data cache misses stall the pipeline.
E1-5	EPU execution stages The EPU implements a single-issue dual-beat MVE pipeline which can also accept a single FP instruction per cycle, whether half-, double-, or single-precision. Instruction execution datapath begins in the E3 stage, with E1 and E2 used for staging, decode, and obtaining operands from the vector register file. Some EPU instructions can require multicycle execution, leading to instruction latencies longer than the pipe length as described per instruction.

3. Instruction latencies

This chapter describes the high-level performance characteristics for most ARMv8.1M instructions.

3.1 Instruction tables

A series of tables summarize the effective execution latency and throughput, pipelines utilized, dual-issue ability, and special behaviors associated with each group of instructions. Most instructions support parallel execution on Cortex®-M85. Some specific issuing capabilities for scalar, floating point, and MVE instructions are detailed in the tables following this section.

In the tables that follow this section:

- Execution 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 or cycle) of the specified instruction group that can be achieved in the entirety of the Cortex®-M85 processor microarchitecture.
- Cortex®-M85 processor has 2 slots to dual issue. The Dual-issue field is interpreted as:
  - 01 dual-issuable from slot 0
  - 00 not dual-issuable
  - 11 dual-issuable from both slot 0 and slot 1

  The Cortex®-M85 issue logic allows the slot 0 execution resource to be claimed from either of the two incoming decode slots. In this way an instruction which can only execute in slot 0 can claim it regardless of which slot it occupies from decode to issue, but only one slot 0 instruction can issue in a cycle.
- The Cortex®-M85 processor is a 2 beat per tick machine for MVE, and it supports overlapping up to two beatwise MVE instructions at any time so that an MVE instruction can be issued after another MVE instruction without additional stall. MVE instructions fall into different “execution groups” which use common resources and cannot overlap with each other. Group membership is noted per instruction category, but in general terms, the groups are organized as follows:
  - *Group A:* mainly integer instructions along with floating-point add instructions
  - *Group B:* mainly floating point instructions along with integer multiplies
  - *Group C:* compare instructions
  - *Group SYS:* instructions that manipulate MVE vector predicates (only VCTP, VPNOT, and VPST)
  - *Group LD:* vector load instructions
  - *Group ST:* vector store instructions
  - Logical and MOV instructions can execute in either Group A or Group B

3.2 Branch instructions

The following tables summarize latency and throughput information for 32-bit and 16-bit Thumb Branch instructions.

Table 3-1: Latency and throughput information for 32-bit Thumb Branch instructions

Instruction group	32-bit Thumb instructions	Execution latency	Execution throughput	Dual-issue	Notes
Branch Future	BF (T1) BFCSEL (T2) BFL (T4) BFLX (T5) BFX (T3)	1	1	11	1
Branch Immediate	B (T3) B (T4)	1	1	11	-
Branch Immediate	BL (T1)	1	1	11	-
Low Overhead Loops	DLS (T2) DLSTP (T4) LCTP (T1)	1	1	11	-
Low Overhead Loops	LE (T1) LE (T2) LETP (T3)	1	1	11	-
Low Overhead Loops (While)	WLS (T1) WLSTP (T3)	1	1	11	-

Notes:

1 Acts as a NOP

Table 3-2: Latency and throughput information for 16-bit Thumb Branch instructions

Instruction group	16-bit Thumb instructions	Execution latency	Execution throughput	Dual-issue	Notes
Branch Immediate	B (T2)	1	1	11	-
Branch Immediate	BX (T1)	1	1	11	-
Branch Immediate	CBNZ, CBZ (T1)	1	1	11	-
Branch Register	BXNS (T1)	1	1	11	-
Branch Register	BLX, BLXNS (T1) BLXNS (T1)	1	1	11	-
Branch, register (with destination LR)	BX (T1)	1	1	11	-

3.3 Arithmetic and logical instructions

The following tables summarize latency and throughput information for 32-bit and 16-bit Thumb Arithmetic and Logical instructions.

Table 3-3: Latency and throughput information for 32-bit Thumb Arithmetic and Logical instructions

Instruction group	32-bit Thumb instructions	Execution latency	Execution throughput	Dual-issue	Notes
Add operations	ADC (immediate) (T1) ADR (T2) ADR (T3) CMN (immediate) (T1) CMP (immediate) (T2)	1	2	11	-
	ADD (SP plus register) (T3) SUB (SP plus register) (T3)	1	2	11	1
ALU SP operations	ADD SP (immediate) (T3) ADDW SP (immediate) (T4) SUB SP (immediate) (T3) SUBW SP (immediate) (T4)	1	2	11	-

Instruction group	32-bit Thumb instructions	Execution latency	Execution throughput	Dual-issue	Notes
ALU operations	ADC (register) (T2) ADD (register) (T3) AND (register) (T2) BIC (register) (T2) CMN (register) (T2) CMP (register) (T2) EOR (register) (T2) MVN (register) (T2) ORR (register) (T2) ORN (register) (T1) RSB (register) (T2) SBC (register) (T2) SUB (register) (T3) TEQ (register) (T2) TST (register) (T2)	1	2	11	1
ALU operations	ADD (immediate) (T3) ADDW (immediate) (T4) SUB (immediate) (T3) SUBW (immediate) (T4)	1	2	11	-

Instruction group	32-bit Thumb instructions	Execution latency	Execution throughput	Dual-issue	Notes
Basic ALU	AND (immediate) (T1) BIC (immediate) (T1) EOR (immediate) (T1) ORN (immediate) (T1) ORR (immediate) (T1) RBIT (T1) REV (T2) REV16 (T2) REVSH (T2) SBFX (T1) UBFX (T1)	1	2	11	-
	BFI (T1)	2	2	11	-
Basic ALU	CSEL (T1) CSINC (T1) CSINV (T1) CSNEG (T1)	1	1	11	-
Basic ALU	BFC (T1) CLZ (T1) SEL (T1)	1	1	01	-
	PKHBT, PKHTB (T1)	1	1	01	1
Basic Move operations	MVN (immediate) (T1)	1	2	11	-
Saturating Arithmetic	USAT (T1)	2	1	01	-

Instruction group	32-bit Thumb instructions	Execution latency	Execution throughput	Dual-issue	Notes
Saturating Arithmetic	QADD (T1) QADD16 (T1) QADD8 (T1) QASX (T1) QDADD (T1) QDSUB (T1) QSAX (T1) QSUB (T1) QSUB16 (T1) QSUB8 (T1) UQADD16 (T1) UQADD8 (T1) UQASX (T1) UQSAX (T1) UQSUB16 (T1) UQSUB8 (T1) USAT16 (T1)	2	1	01	-
Sum of Absolute Differences	USAD8 (T1)	2	1	11	-
Sum of Absolute Differences	USADA8 (T1)	2(1)	1	11	2
Sign Extend Addition	SXTB (T2) SXTH (T2) SXTB16 (T1)	1	1	01	-
	SXTAB (T1) SXTAB16 (T1) SXTAH (T1)	1	1	01	1
Signed Addition	SSAT (T1) SSAT16 (T1)	2	1	01	-

Instruction group	32-bit Thumb instructions	Execution latency	Execution throughput	Dual-issue	Notes
Signed Addition	SADD16 (T1) SADD8 (T1) SASX (T1) SHADD16 (T1) SHADD8 (T1) SHASX (T1) SHSAX (T1) SHSUB16 (T1) SHSUB8 (T1) SSAX (T1) SSUB16 (T1) SSUB8 (T1)	1	1	01	-
Subtract operations	RSB (immediate) (T2) SBC (immediate) (T1)	1	2	11	-
Test operations	TEQ (immediate) (T1) TST (immediate) (T1)	1	2	11	-
Test operations	TT, TTT, TTA, TTAT (T1)	2	2	11	-
Unsigned Addition	UADD16 (T1) UADD8 (T1) USUB16 (T1) USUB8 (T1) UASX (T1) USAX (T1) UHADD16 (T1) UHADD8 (T1) UHASX (T1) UHSAX (T1) UHSUB16 (T1) UHSUB8 (T1)	1	1	01	-

Instruction group	32-bit Thumb instructions	Execution latency	Execution throughput	Dual-issue	Notes
Zero Extend Addition	UXTB (T2) UXTH (T2) UXTB16 (T1)	1	1	01	-
	UXTAB (T1) UXTAB16 (T1) UXTAH (T1)	1	1	01	1

Notes:

1 The latency from the shifter source operand is 2, regardless of whether the shift immediate value is non-zero or not.

2 The latency from the addend source operand is 1.

Table 3-4: Latency and throughput information for 16-bit Thumb Arithmetic and Logical instructions

Instruction group	16-bit Thumb instructions	Execution latency	Execution throughput	Dual-issue	Notes
Add operations	ADC (register) (T1) ADD (SP plus immediate) (T2) ADD (register) (T1) ADD (register) (T2) ADR(T1)	1	2	11	-
	ADD (SP plus immediate) (T2) ADD (SP plus register) (T2)	1	2	11	-
	ADD (SP plus immediate) (T1) ADD (immediate) (T1) ADD (immediate) (T2)	1	2	11	-
	ADD (SP plus register) (T1)	1	2	11	-
Basic ALU	CMN (register) (T1)	1	2	11	-

Instruction group	16-bit Thumb instructions	Execution latency	Execution throughput	Dual-issue	Notes
Basic ALU	AND (register) (T1) BIC (register) (T1) CMP (register) (T1) CMP (register) (T2) EOR (register) (T1) ORR (register) (T1) REV (T1) REV16 (T1) REVSH (T1) TST (register) (T1)	1	2	11	-
Basic ALU	CMP (immediate) (T1)	1	2	11	-
Sign Extend Addition	SXTB (T1) SXTH (T1)	1	1	01	-
Subtract operations	RSB (immediate) (T1) SBC (register) (T1) SUB (SP minus immediate) (T1) SUB (register) (T1)	1	2	11	-
Subtract operations	SUB (immediate) (T1) SUB (immediate) (T2)	1	2	11	-
Zero Extend Addition	UXTB (T1) UXTH (T1)	1	1	01	-

3.4 Move and shift instructions

The following tables summarize latency and throughput information for 32-bit and 16-bit Thumb Move and Shift instructions.

Table 3-5: Latency and throughput information for 32-bit Thumb Move and Shift instructions

Instruction group	32-bit Thumb instructions	Execution latency	Execution throughput	Dual-issue	Notes
Basic Move operations	MOV (immediate) (T2) MOV (immediate) (T3) MOV (register) (T3) MOV (register) (T3) MOV, MOVS (register-shifted register) (T2) MOVT (T1)	1	2	11	-

Table 3-6: Latency and throughput information for 16-bit Thumb Move and Shift instructions

Instruction group	16-bit Thumb instructions	Execution latency	Execution throughput	Dual-issue	Notes
Basic Move operations	MOV (register) (T2) MOV, MOVS (register-shifted register) (T1) MVN (register) (T1)	1	2	11	-
Basic Move operations	MOV (immediate) (T1)	1	2	11	-
Basic Move operations	MOV (T1)	1	2	11	-

3.5 Divide and multiply instructions

The following table summarize latency information for T32 and T16 Divide and Multiply instructions.

Table 3-7: Latency and throughput information for 32-bit Thumb Divide and Multiply instructions

Instruction group	32-bit Thumb instructions	Execution latency	Execution throughput	Dual-issue	Notes
Divide	SDIV (T1)	6-21	1/19-1/4	-	1
Divide	UDIV (T1)	5-20	1/18-1/3	-	1

Instruction group	32-bit Thumb instructions	Execution latency	Execution throughput	Dual-issue	Notes
Multiply	MUL (T2) SMMUL, SMMULR (T1) SMUAD, SMUADX (T1) SMULBB, SMULBT, SMULTB, SMULTT (T1) SMULL (T1) SMULWB, SMULWT (T1) SMUSD, SMUSDX (T1) UMULL (T1)	2	1	11	-
Multiply Accumulate	MLA (T1) MLS (T1) SMLABB, SMLABT, SMLATB, SMLATT (T1) SMLAD, SMLADX (T1) SMLAL (T1) SMLALBB, SMLALBT, SMLALTB, SMLALTT (T1) SMLALD, SMLALDX (T1) SMLAWB, SMLAWT (T1) SMLSD, SMLSDX (T1) SMLSLD, SMLSLDX (T1) SMMLA, SMMLAR (T1) SMMLS, SMMLSR (T1) UMAAL (T1) UMLAL (T1)	2(1)	1	11	2

Notes:

1 Divides are performed using an iterative algorithm, and block any subsequent divide operations until complete. Latency is highest when divisors are much smaller than dividends, and lowest when divisor and dividend are similar in magnitude and in the case of divide by zero.

2 Latency to result is 1 from addend/minuend operand, and 2 from other operands.

Table 3-8: Latency and throughput information for 16-bit Thumb Divide and Multiply instructions

Instruction group	16-bit Thumb instructions	Execution latency	Execution throughput	Dual-issue	Notes
Multiply	MUL (T1)	2	1	11	-

3.6 Load instructions

The following tables summarize latency and throughput information for 32-bit and 16-bit Thumb Load instructions. Base register update latency is always 1.

Table 3-9: Latency and throughput information for 32-bit Thumb Load instructions

Instruction group	32-bit Thumb instructions	Execution latency	Execution throughput	Dual-issue	Notes
Basic loads	LDR (immediate) (T3) LDR (immediate) (T4) LDR (literal) (T2) LDR (register) (T2)	2	2	11	-
Exclusive operations	LDA (T1) LDAEX (T1) LDREX (T1)	2	1	00	-
Load multiples	LDRD (immediate) (T1) LDRD (literal) (T1)	2	1	11	1
Load multiples	LDM, LDMIA, LDMFD (T2) LDMDB, LDMEA (T1)	N+1	1/N	00	1

Instruction group	32-bit Thumb instructions	Execution latency	Execution throughput	Dual-issue	Notes
Sub word loads	LDRB (immediate) (T2) LDRB (immediate) (T3) LDRB (literal) (T1) LDRB (register) (T2) LDRBT (T1) LDRH (immediate) (T2) LDRH (immediate) (T3) LDRH (literal) (T1) LDRH (register) (T2) LDRHT (T1) LDRSB(immediate) (T1) LDRSB (immediate) (T2) LDRSB (literal) (T1) LDRSB (register) (T2) LDRSH (immediate) (T1) LDRSH (immediate) (T2) LDRSH (literal) (T1) LDRSH (register) (T2)	3	3	02	-
Sub word exclusives	LDAB(T1) LDAH (T1) LDAEXB(T1) LDAEXH (T1) LDREXB(T1) LDREXH(T1)	3	1	00	-

Notes:

1 Cortex®-M85 processor supports two 32-bit accesses per cycle. N=floor((num_regs+1)/2).

Table 3-10: Latency and throughput information for 16-bit Thumb Load instructions

Instruction group	16-bit Thumb instructions	Execution latency	Execution throughput	Dual-issue	Notes
Basic Loads	LDR (immediate) (T1) LDR (immediate) (T2) LDR (literal) (T1) LDR (register) (T1)	2	2	11	-
Load Multiples	LDM, LDMIA, LDMFD (T1) POP (multiple registers) (T3)	N+1	1/N	00	1
Sub Word Loads	LDRB (immediate) (T1) LDRB (register) (T1) LDRH (immediate) (T1) LDRH (register) (T1) LDRSB (register) (T1) LDRSH (register) (T1)	3	2	11	-

Notes:

1 Cortex®-M85 processor supports two 32-bit accesses per cycle. N=floor((num_regs+1)/2).

3.7 Store instructions

The following tables summarize latency and throughput information for 32-bit and 16-bit Thumb Store instructions. Base register update latency is always 1.

Table 3-11: Latency and throughput information for 32-bit Thumb Store instructions

Instruction group	32-bit Thumb instructions	Execution latency	Execution throughput	Dual-issue	Notes
Basic stores	STR (immediate) (T3) STR (immediate) (T4) STR (register) (T2)	-	2	11	-
Exclusive operations	STREX (T1) STREXB (T1) STREXH (T1)	-	1	00	-

Instruction group	32-bit Thumb instructions	Execution latency	Execution throughput	Dual-issue	Notes
Store lock release	STL (T1) STLB (T1) STLEX (T1) STLEXB (T1) STLEXH (T1) STLH (T1)	-	1/5	00	-
Store multiple	STRD (immediate) (T1)	-	1	11	1
Store multiple	STM, STMIA, STMEA (T2) STMDB, STMFD (T1)	-	1/N	00	1
Sub word stores	STRB (immediate) (T2) STRB (immediate) (T3) STRB (register) (T2) STRH (immediate) (T2) STRH (immediate) (T3) STRH (register) (T2)	-	2	11	-

Notes:

1 Cortex®-M85 processor supports two 32-bit accesses per cycle. N=floor((num_regs+1)/2).

Table 3-12: Latency and throughput information for 16-bit Thumb Store instructions

Instruction group	16-bit Thumb instructions	Execution latency	Execution throughput	Dual-issue	Notes
Basic stores	STR (immediate) (T1) STR (immediate) (T2) STR (register) (T1)	-	2	11	-
Store multiple	PUSH (multiple registers) (T2) STM, STMIA, STMEA (T1)	-	1/N	00	1
Sub word stores	STRB (immediate) (T1) STRB (register) (T1) STRH (immediate) (T1) STRH (register) (T1)	-	2	11	-

Notes:

1 Cortex®-M85 processor supports two 32-bit accesses per cycle. N=floor((num_regs+1)/2).

3.8 Miscellaneous instructions

The following tables summarize latency and throughput information for 32-bit and 16-bit Thumb Miscellaneous instructions.

Table 3-13: Latency and throughput information for 32-bit Thumb Miscellaneous instructions

Instruction group	32-bit Thumb instructions	Execution latency	Execution throughput	Dual-issue	Notes
Hints	PLI (immediate, literal) (T1) PLI (immediate, literal) (T2) PLI (immediate, literal) (T3) PLI (register) (T1)	-	1	11	1
Hints	PLD (literal) (T1) PLD, PLDW (immediate) (T1) PLD, PLDW (immediate) (T2) PLD, PLDW (register) (T1)	-	1	11	-
No Operation	NOP (T2)	-	2	11	-
Register updates	CLRM (T1)	4	1/4	00	-

Notes:

1 Acts as a NOP.

Table 3-14: Latency and throughput information for 16-bit Thumb Miscellaneous instructions

Instruction group	16-bit Thumb instructions	Execution latency	Execution throughput	Dual-issue	Notes
No Operation	NOP (T1)	-	3	11	-
Program Flow Control	IT (T1)	1	1	11	-

3.9 FP data processing instructions

The following table summarizes latency and throughput information for FP Data Processing Instructions.

Table 3-15: Latency and throughput information for FP Data Processing instructions

Instruction group	Instructions	Execution latency	Execution throughput	Dual-issue	Notes
Continuous vector load	VLDR (T2) VLDR (T3)	2	2	11	-
Continuous vector load	VLDR (T1)	2	1	01	2
Continuous vector load	VLDM (T1) VLLDM (T1)	N+1	1/N	11(01)	1,2
Divide (double-precision)	VDIV (T1)	32	1/31	01	3
Divide (half-precision)	VDIV (T1)	6	1/5	11	3
Divide (single-precision)	VDIV (T1)	18	1/17	11	3
Divide(all-precision) with input zero/infinite/ NaN or invalid operation	VDIV (T1)	6	1/5	11(01)	1,3
Scalar absolute	VABS (T2)	1	1	11(01)	1
Scalar arithmetic (single and half-precision)	VADD (T1) VSUB (T1)	2	1	01	-
Scalar arithmetic (double-precision)	VADD (T1) VSUB (T1)	3-8	1/2-1/7	01	1
Scalar arithmetic	VMAXNM (T1)	1	1	11(01)	1
Scalar compare	VCMP (T1) VCMP (T2)	1	1	11(01)	1
Scalar convert	VCVT (between double-precision and single-precision) (T1) VCVT (between floating-point and fixed-point) (T1) VCVT (floating-point to integer) (T1) VCVTA, VCVTN, VCVTP, VCVTM (T1) VCVTB, VCVTT (T1) VRINTA, VRINTN, VRINTP, VRINTM (T1) VRINTR VRINTZ (T1) VRINTX (T1)	2	1	11(01)	1

Instruction group	Instructions	Execution latency	Execution throughput	Dual-issue	Notes
Scalar MOV	VINS (T1) VMOV (between general-purpose register and half-precision register) (T1) VMOV (between general-purpose register and single-precision register) (T1) VMOV (between two general-purpose registers and adoubleword register) (T1) VMOV (between two general-purpose registers andtwo single-precision registers) (T1) VMOV (immediate) (T2) VMOV (register) (T1) VMOVX (T1)	1	1	11(01)	4
Scalar multiply (single and half-precision)	VMUL (T1) VNMUL (T2)	3	1	01	1
Scalar multiply (double-precision)	VMUL (T1) VNMUL (T2)	5-10	1/3-1/8	01	1
Scalar multiply (single and half-precision)	VFMA, VFMS (T1) VFNMA, VFNMS (T1)	4(2)	1	01	1
Scalar multiply (double-precision)	VFMA, VFMS (T1) VFNMA, VFNMS (T1)	10-15	1/7-1/12	01	1
Scalar multiply (single and half-precision)	VMLA, VMLS(T1) VNMLA, VNMLS(T1)	5(2)	1	01	1
Scalar multiply (double-precision)	VMLA, VMLS(T1) VNMLA, VNMLS(T1)	11-16	1/7-1/12	01	1
Scalar negate	VNEG (T1)	1	1	11(01)	1
Scalar select	VSEL (T1)	1	1	11(01)	1
Square root (double-precision)	VSQRT (T1)	31	1/30	01	3
Square root (half-precision)	VSQRT (T1)	6	1/5	11	3
Square root (single-precision)	VSQRT (T1)	17	1/16	11	3
Square root (all-precision) with input zero/ infinite/NaN or invalid operation	VSQRT (T1)	6	1/5	11(01)	1,3
Store	VSTR (T2) VSTR (T3)	-	2	11	1
Store	VSTR (T1)	-	1	01	2

Instruction group	Instructions	Execution latency	Execution throughput	Dual-issue	Notes
Store	VLSTM (T1) VSTM (T1)	-	1/N	11(01)	2

Notes:

1 The latency from the addend source operand is 2.

2 Cortex®-M85 processor supports two 32-bit accesses per cycle. For single-precision store multiple instructions, N=floor((num_regs+1)/2). For double-precision store multiple instructions, N=(num_regs).

3 Divides and square roots are performed using an iterative algorithm and block any subsequent divide and square root operations until complete.

4 For encoding VMOV (between two general-purpose registers and two single-precision registers), latency is 4 when S registers are written and the two S register sources are not part of the same D register.

3.10 MVE integer vector instructions

The following table summarizes latency and throughput information for MVE Integer Vector instructions.

Table 3-16: Latency and throughput information for MVE Integer Vector instructions

Instruction group	Instructions	Execution latency	Execution throughput	Execution group	Notes
MVE absolute	VABAV (T1)	2	1/2	A	-
	VABD (T1) VABS (T1) VQABS (T1)	1	1/2	A	-
MVE arithmetic	VMAXV, VMAXAV (T1) VMAXV, VMAXAV (T2) VMINV, VMINAV (T1) VMINV, VMINAV (T2)	2	1/2	A	-

Instruction group	Instructions	Execution latency	Execution throughput	Execution group	Notes
MVE arithmetic	VADC (T1) VADD (vector) (T1) VADD (vector) (T2) VCADD (T1) VHADD (T1) VHADD (T2) VHCADD (T1) VHSUB (T1) VHSUB (T2) VMAX, VMAXA (T1) VMAX, VMAXA (T2) VMIN, VMINA (T1) VMIN, VMINA (T2) VQADD (T1) VQADD (T2) VQSUB (T1) VQSUB (T2) VRHADD (T1) VSBC (T1) VSUB (T1) VSUB (T2)	1	1/2	A	-

Instruction group	Instructions	Execution latency	Execution throughput	Execution group	Notes
MVE bitwise	VAND (T1) VBIC (immediate) (T1) VBIC (register) (T1) VEOR (T1) VMOV (immediate) (T1) VMVN (immediate) (T1) VMVN (register) (T1) VORN (T1) VORR (T1) VORR (immediate) (T1) VREV16 (T1) VREV32 (T1) VREV64 (T1)	1	1	A or B	-
MVE CLS/CLZ	VCLS (T1) VCLZ (T1)	1	1/2	A	-
MVE compare	VCMP (T1) VCMP (T2) VCMP (T3) VCMP (T4) VCMP (T5) VCMP (T6) VPT (T1) VPT (T2) VPT (T3) VPT (T4) VPT (T5) VPT (T6)	1	1/2	C	-

Instruction group	Instructions	Execution latency	Execution throughput	Execution group	Notes
MVE duplicate	VDDUP, VDWDUP (T1) VDDUP, VDWDUP (T2) VIDUP, VIWDUP (T1) VIDUP, VIWDUP (T2)	1	1/2	A	-
	VDUP (T1)	1	1/2	A	-
MVE MOV	VMOVL (T1) VMOVN (T1)	1	1/2	A	-

Instruction group	Instructions	Execution latency	Execution throughput	Execution group	Notes
MVE multiply	VMLA (vector by scalar plus vector) (T1) VMLAS (vector by vector plus scalar) (T1) VMUL (T1) VMUL (T2) VMULH,VRMULH (T1) VMULH, VRMULH (T2) VMULL (integer) (T1) VQDMLADH, VQRDMLADH (T1) VQDMLADH, VQRDMLADH (T2) VQDMLAH, VQRDMLAH (vector by scalar plus vector) (T1) VQDMLAH, VQRDMLAH (vector by scalar plus vector) (T2) VQDMLASH, VQRDMLASH (vector by vector plus scalar) (T1) VQDMLASH, VQRDMLASH (vector by vector plus scalar) (T2) VQDMLSDH, VQRDMLSDH (T1) VQDMLSDH, VQRDMLSDH (T2) VQDMULH, VQRDMULH (T1) VQDMULH, VQRDMULH (T2) VQDMULH, VQRDMULH (T3) VQDMULH, VQRDMULH (T4) VQDMULL (T1) VQDMULL (T2)	2	1/2	B	-
MVE multiply	VMULL (polynomial) (T1)	1	1/2	A	
MVE negate	VNEG (T1) VQNEG (T1)	1	1/2	A	-
MVE select	VPSEL (T1)	1	1/2	A	-

Instruction group	Instructions	Execution latency	Execution throughput	Execution group	Notes
MVEshift	VBRSR (T1) VSHL (T1) VSHLL (T1)	1	1/2	A	-
	VQMOVN (T1) VQMOVUN (T1) VQRSHL (T1) VQRSHL (T2) VQRSHRN (T1) VQRSHRUN (T1) VQSHL, VQSHLU (T1) VQSHL, VQSHLU (T2) VQSHL, VQSHLU (T3) VQSHL, VQSHLU (T4) VQSHRN (T1) VQSHRUN (T1) VRSHL (T1) VRSHL (T2) VRSHR (T1) VRSHRN (T1) VSHL (T2) VSHL (T3) VSHLC (T1) VSHLL (T2) VSHR (T1) VSHRN (T1) VSLI (T1) VSRI (T1)	2	1/2	A	-

Instruction group	Instructions	Execution latency	Execution throughput	Execution group	Notes
MVE arithmetic to scalar	VADDLV (T1) VADDV (T1) VMLADAV (T1) VMLADAV (T2) VMLALDAV (T1) VMLSDAV (T1) VMLSDAV (T2) VMLSLDAV (T1) VRMLALDAVH (T1) VRMLSLDAVH (T1)	2	1/2	B	-

3.11 MVE integer scalar instructions

The following table summarize and throughput information for MVE Integer Scalar instructions.

Table 3-17: Latency and throughput information for MVE Integer Scalar instructions

Instruction group	Instructions	Execution latency	Execution throughput	Execution group	Notes
Scalar MOV	VMOV (general-purpose register to vector lane) (T1) VMOV (two general-purpose registers to two 32 bit vector lanes) (T1)	1	1	A or B	1
Scalar MOV	VMOV (two 32 bit vector lanes to two general-purpose registers) (T1) VMOV (vector lane to general-purpose register) (T1)	1	1/2	B	-

Notes:

1 Latency from the shift amount operand is 2. Latency from the shifted value operands is 1, shown in parentheses.

Table 3-18: Latency and throughput information for MVE Integer Scalar shift instructions

Instruction group	Instructions	Execution latency	Execution throughput	Dual-issue	Notes
Scalar shift	ASRL (immediate) (T1) LSLL (immediate) (T1) LSRL (immediate) (T1) SQSHL (immediate) (T1) SQSHLL (immediate) (T1) SRSHR (immediate) (T1) SRSHRL (immediate) (T1) UQSHL (immediate) (T1) UQSHLL (immediate) (T1) URSHR (immediate) (T1) URSHRL (immediate) (T1)	1	1	11	
	ASRL (register) (T1) LSLL (register) (T1) SQRSHR (register) (T1) SQRSHRL (register) (T1) UQRSHL (register) (T1) UQRSHLL (register) (T1)	2(1)	1	11	1

Notes:

1 Latency from the shift amount operand is 2. Latency from the shifted value operands is 1, shown in parentheses.

3.12 MVE FP instructions

The following table summarizes latency and throughput information for MVE FP instructions.

Table 3-19: Latency and throughput information for MVE FP instructions

Instruction group	Instructions	Execution latency	Execution throughput	Execution group	Notes
MVE absolute	VABD (floating-point) (T1)	2	1/2	A	-
	VABS (floating-point) (T1)	1	1/2	A	-

Instruction group	Instructions	Execution latency	Execution throughput	Execution group	Notes
MVE arithmetic	VMAXNMV, VMAXNMAV (floating-point) (T1) VMAXNMV, VMAXNMAV (floating-point) (T2) VMINNMV, VMINNMAV (floating-point) (T1) VMINNMV, VMINNMAV (floating-point) (T2)	2	1/2	B	-
MVE arithmetic	VADD (floating-point) (T1) VADD (floating-point) (T2) VCADD (floating-point) (T1) VSUB (floating-point) (T1) VSUB (floating-point)(T2)	2	1/2	A	-
MVE arithmetic	VMAXNM, VMAXNMA (floating-point) (T1) VMAXNM,VMAXNMA (floating-point) (T2) VMINNM, VMINNMA (floating-point) (T1) VMINNM, VMINNMA (floating-point) (T2)	1	1/2	B	-
MVE compare	VPT (floating-point) (T1) VPT (floating-point) (T2)	1	1/2	C	-
MVE compare	VCMP (floating-point) (T1) VCMP (floating-point) (T2)	1	1/2	C	-
MVE convert	VCVT (between floating-point and fixed-point) (T1) VCVT (between floating-point and integer) (T1) VCVT (between single and half-precision floating-point)(T1) VCVT(from floating-point to integer) (T1) VRINT (floating-point) (T1)	2	1/2	B	-
MVE multiply	VCMUL (floating-point) (T1) VMUL (floating-point) (T1) VMUL (floating-point) (T2)	3	1/2	B	-

Instruction group	Instructions	Execution latency	Execution throughput	Execution group	Notes
MVE multiply	VCMLA (floating-point) (T1) VFMA (vector by scalar plus vector, floating-point) (T1) VFMA, VFMS (floating-point) (T1) VFMA,VFMS (floating-point) (T2) VFMAS (vector by vector plus scalar, floating-point) (T1)	4(2)	1/2	A and B	2
MVE negate	VNEG (floating-point) (T1)	1	1/2	B	-

Notes:

1 These instructions appear to be in group B to older adjacent instructions and group A to earlier adjacent instructions.

2 The latency from the addend source operand is 2.

3.13 MVE miscellaneous instructions

The following table summarize latency and throughput information for MVE Miscellaneous instructions.

Table 3-20: Latency and throughput information for Miscellaneous instructions

Instruction group	Instructions	Execution latency	Execution throughput	Execution group	Notes
System	VCTP (T1) VPNOT (T1) VPST (T1)	1	1	SYS	-

Table 3-21: Latency and throughput information for Miscellaneous scalar instructions

Instruction group	Instructions	Execution latency	Execution throughput	Dual-issue	Notes
System	VMRS (T1)	1	1	-	-
	VMSR (T1)	1(4)	1(1/4)	-	1

Notes:

1 Values in parentheses apply when the destination register is the FPSCR.

3.14 MVE load instructions

The following table summarizes latency and throughput information for MVE Load instructions.

Table 3-22: Latency and throughput information for MVE Load instructions

Instruction group	Instructions	Execution latency	Execution throughput	Execution group	Notes
Continuous vector load	VLDRB,VLDRH, VLDRW (T1) VLDRB, VLDRH, VLDRW (T2) VLDRB, VLDRH, VLDRW (T5) VLDRB, VLDRH, VLDRW (T6) VLDRB,VLDRH, VLDRW (T7)	2	1/2	LD	-
Continuous vector load	VLD2(T1) VLD4(T1)	4	1/2	LD	-
Gather load	VLDRB, VLDRH, VLDRW, VLDRD (vector) (T1) VLDRB,VLDRH, VLDRW, VLDRD (vector) (T2) VLDRB,VLDRH, VLDRW, VLDRD (vector) (T3)	(64/esize)+ 1	esize/64	LD	1
Gather load	VLDRB,VLDRH, VLDRW, VLDRD (vector) (T4)	3	1/2	LD	-
Gather load	VLDRB, VLDRH, VLDRW, VLDRD (vector) (T5)	3	1/2	LD	-
Gather load	VLDRB, VLDRH, VLDRW, VLDRD (vector) (T6)	3	1/2	LD	-

Notes:

1 *esize* is the element size of the instruction variant. It could be 8, 16, or 32, depending on the variant this instruction supports. Base register update latency is 7.

3.15 MVE store instructions

The following tables summarize latency and throughput information for MVE Store instructions. Unless specified, base register update latency is 1.

Table 3-23: Latency and throughput information for MVE Store instructions

Instruction group	Instructions	Execution latency	Execution throughput	Execution group	Notes
Scatter Store	VSTRB, VSTRH, VSTRW, VSTRD (vector) (T1) VSTRB, VSTRH, VSTRW, VSTRD (vector) (T2) VSTRB, VSTRH, VSTRW, VSTRD (vector) (T3)	-	esize/64	ST	1
Scatter Store	VSTRB, VSTRH, VSTRW, VSTRD (vector) (T4)	-	1/2	ST	-
Scatter Store	VSTRB, VSTRH, VSTRW, VSTRD (vector) (T5)	-	1/2	ST	-
Scatter Store	VSTRB, VSTRH, VSTRW, VSTRD (vector) (T6)	-	1/2	ST	-
Store	VSTRB, VSTRH, VSTRW (T1) VSTRB, VSTRH, VSTRW (T2)	-	1/2	ST	-
Store	VSTRB, VSTRH, VSTRW (T5) VSTRB, VSTRH, VSTRW (T6) VSTRB, VSTRH, VSTRW (T7)	-	1/2	ST	-
Store	VST2 (T1) VST4(T1)	-	1/2	ST	-

Notes:

1 *esize* is the element size of the instruction variant. It could be 8, 16, or 32, depending on the variant this instruction supports. Base register update latency is 7.

4. General behaviors

This chapter describes the general behaviors of MVE ARMv8.1M instructions.

4.1 Overlapping MVE execution

As a two beat per tick implementation of the ARMv8.1-M MVE architecture, the Cortex®-M85 processor issues MVE vector instructions over two cycles known as ticks, called tick0 and tick1.

Each cycle of MVE execution in the EPU or LSU operates on 64 bits of data. Overlapping means tick1 of an MVE instruction can execute in parallel to a tick0 of the succeeding MVE instruction.

The ability to overlap instructions is assessed in the ISS stage of the DPU based on instruction types and data dependencies. A newer MVE instruction can overlap with the older MVE instruction if the older tick1 does not require the same issue group resource as the newer tick0. Issue groups can be seen per instruction in Table 3-16: Latency and throughput information for MVE Integer Vector instructions on page 31. By overlapping execution, 64 bits of vector datapath can be made to provide the performance of 128 bits on common code sequences and the typical tabulated MVE instruction execution throughputs of ½ can add to a combined vector instruction IPC of 1.

By interacting with scalar instructions, MVE instructions act as a slot 0 only instruction in tick0 and do not occupy the issue stage in tick 1. Scalar floating-point instructions share the execution resources of the EPU pipeline with MVE instructions and can encounter structural hazards when issuing in adjacent cycles to MVE instructions. Structural hazards between scalar integer instructions and MVE instructions can also occur in cases in which MVE instructions modify scalar registers.

4.2 Data dependency special cases

Some interaction between instructions is not fully described by the individual instruction parameters tabulated in 3. Instruction latencies on page 13.

As described in Table 2-1: Pipeline stages on page 12, different instruction types start execution at different times in the Cortex®-M85 pipeline. Specifically, scalar floating-point and MVE integer and floating-point instructions start execution in the E3 stage, two cycles after integer instructions and load/store instructions. As a result:

- Floating-point and MVE vector instructions perceive shorter latencies in integer and load/store instructions. For example, a dependent vector instruction issued immediately after a vector load instruction with a latency of two, can issue with no penalty, and a dependent MVE vector instruction immediately following a vector load with latency four, will stall for two cycles.
- Integer instructions perceive correspondingly longer latencies from floating-point or MVE vector instructions which produce scalar integer register values.

Store instructions can accept their data source arguments several cycles after issuing. As a result:

- Integer store instructions have the latency of integer instructions producing store data as two cycles faster than tabulated. For example, a scalar store with store data dependent on a two-cycle load can issue immediately after the load with no stalls.
- Vector store instructions other than interleaving stores have the latency of vector instructions producing store data as three cycles faster than tabulated. For example, a vector stores immediately following a vector or scalar VFMA instruction stall for one cycle.
- Interleaving store instructions have the latency of vector instructions producing store data as tabulated.

4.3 Memory banking

Performance of memory accesses to DTCM and L1 data cache is influenced by memory banking.

Each has four banks of 32-bit width with address bits [3:2] determining which bank is accessed. Two load instructions issued in the same cycle to the same memory bank will result in a conflict, causing the second access to stall for a cycle while the first proceeds. Ideally, scalar loads and MVE gather load accesses to the same memory bank should not be adjacent in code sequences so that no bank conflict occurs. ITCM is implemented as a single bank of memory, allowing only one access to be served at a time.

Buffering of store data decouples pipeline store instruction flow from banking considerations.

Bank conflicts in the DTCM can also occur between CPU and DMA accesses. With four DTCM banks and at most two consumed by CPU and by DMA on a sustained basis, the DTCM has sufficient bandwidth to serve requests from both with some loss to be expected due to conflicts.

4.4 Load and store address alignment

Load and store instruction performance is affected by the alignment of the memory addresses accessed.

The maximum bandwidth of the 64-bit load and store datapaths and best latency to memory can be achieved with pairs of word and subword loads and stores which do not cross 32-bit boundaries, or multi-word loads and stores which have 32-bit aligned starting addresses. Unaligned accesses can introduce stall cycles.

4.5 Branch instruction alignment

Placement of multiple branch instructions within the same aligned 64-bit region of memory can affect performance negatively by burdening branch prediction logic.

It is preferable to avoid this arrangement if it can be done without affecting program density and performance. Else, it is preferable for branch targets, including subroutine entry points, to be placed on aligned 64-bit boundaries to maximize the use of instruction fetch bandwidth.

4.6 Data accesses to ITCM

Data accesses to ITCM in the code region is supported, but DTCM is a better choice for data in performance-sensitive TCM-bound applications. This is because ITCM serves instruction fetches and can only manage one access per cycle compared with the dedicated 4-way banked DTCM.

4.7 Hardware data prefetch

When configured with an L1 data cache, the Cortex®-M85 processor includes a hardware data prefetcher which anticipates future data accesses when it detects cache misses in a streaming data access pattern.

The prefetcher considers constant strides forward or backward within 2 cache lines of observed miss addresses. Hardware prefetch activity does not cross aligned 8KB boundaries in memory.

4.8 Software prefetch instructions

The PLD and PLDW hint instructions generate data cache line fills with their respective read and write allocation behaviors.

It is not advisable to use explicit software prefetching if the relevant access pattern falls within the capabilities of the hardware data prefetcher as prefetch instructions consume pipeline bandwidth. The PLI instruction behaves as a NOP.

4.9 Transient memory hint

The transient memory attribute is implemented and serves as a hint to the data side memory system to prefer retention of non-transient memory over transient in replacement of clean L1 data cache lines.

4.10 Optimizing MVE execution

Various aspects of coding for MVE performance on the Cortex®-M85 processor are considered here.

- Arrangement of code sequences to allow frequent instruction overlap is a significant factor in performance. It could be advantageous to unroll loops to provide more flexibility in instruction placement to optimize overlap even though there is no loop overhead cost in MVE loops using LE/LETP.
- While vector load/store instructions occupy their own separate execution groups and nominally overlap with all MVE instructions, some coding outcomes are difficult to anticipate due to hardware complexity. The following rules can be employed to achieve optimized performance in most scenarios. To keep the examples simple, MVE load/store instructions are referred to as LDST instructions and MVE integer and floating-point instructions are referred to as SIMD instructions in the following:
  - Alternating sequences of LDST instructions always fully overlap.

```
Fully overlapping:

vldrw.u32 q1,[r1],#16
vstrw.32 q0,[r2]
vldrw.u32 q3,[r1],#16
vstrw.32 q2,[r2]
```

```
Sub-optimal:

vldrw.u32 q1,[r1],#16
vldrw.u32 q3,[r1],#16 // Cannot overlap with prior instruction
vstrw.32 q0,[r2]
vstrw.32 q2,[r2] // Cannot overlap with prior instruction
```

  - SIMD instructions can overlap unless they belong to the same execution group.

```
Fully overlapping:

vadd.i32 q0,q1,q3
vmul.i32 q2,q1,q3
vadd.i32 q4,q1,q3
vmul.i32 q6,q1,q3
```

```
Sub-optimal:

vmul.i32 q0,q1,q3
vmul.i32 q2,q1,q3 // Cannot overlap with prior instruction
vadd.i32 q4,q1,q3
vadd.i32 q6,q1,q3 // Cannot overlap with prior instruction
```

  - LDST instructions can overlap with each other.

```
Fully overlapping:

vldrw.u32 q1,[r1],#16
vstrw.32 q0,[r2],#16
vldrw.u32 q3,[r1],#16
vstrw.32 q2,[r2],#16
```

```
Sub-optimal:
```

```
vldrw.u32 q1,[r1],#16
vldrw.u32 q3,[r1],#16 // Cannot overlap with prior instruction
vstrw.32 q0,[r2],#16
vstrw.32 q2,[r2],#16 // Cannot overlap with prior instruction
```

- LE and LETP instructions have no issue cost and do not influence LDST and SIMD instruction overlap regardless of placement.

```
Fully overlapping:

loop:
vadd.i32 q0,q1,q3
vmul.i32 q2,q1,q3
vadd.i32 q4,q1,q3
vmul.i32 q6,q1,q3
le lr,loop
```

```
Sub-optimal:

loop:
vadd.i32 q0,q1,q3
vmul.i32 q2,q1,q3
vadd.i32 q4,q1,q3
vmul.i32 q6,q1,q3
vadd.i32 q0,q1,q3 // Cannot overlap with subsequent vadd instruction
le lr,loop
```

- The first LDST instruction in a sequence of LDST instructions will always overlap with the last SIMD instruction in a sequence of SIMD instructions.
- A sequence of fully overlapping LDST instructions following a pair of overlapping SIMD instruction overlaps with the following SIMD instruction if the sequence has an even number of LDST instructions.

```
loop:
  vadd.i32 q0,q1,q3
  vmul.i32 q2,q1,q3
  vldrw.u32 q1,[r1],#16 // Overlaps with prior
  vstrw.32 q0,[r2],#16
  vmul.i32 q6,q1,q3 // Overlaps with prior
  le lr,loop
```

```
loop:
  vadd.i32 q0,q1,q3
  vmul.i32 q2,q1,q3
  vldrw.u32 q1,[r1],#16 // Overlaps with prior
  vstrw.32 q0,[r2],#16
  vldrw.u32 q4,[r1],#16
  vstrw.32 q2,[r2],#16
  vadd.i32 q0,q1,q3 // Overlaps with prior
  le lr,loop
```

```
loop:
  vadd.i32 q0,q1,q3
  vmul.i32 q2,q1,q3
  vldrw.u32 q1,[r1],#16 // Overlaps with prior
  vstrw.32 q0,[r2],#16
  vldrw.u32 q4,[r1],#16
  vadd.i32 q0,q1,q3 // Cannot overlap with prior
  le lr,loop
```

- A sequence of fully overlapping LDST instructions following a pair of non-overlapping SIMD instruction will overlap with the following SIMD instruction if the sequence has an odd number of LDST instructions.

```
loop:
  vadd.i32 q0,q1,q3
  vadd.i32 q2,q1,q3
  vldrw.u32 q1,[r1],#16
  vstrw.32 q0,[r2],#16
  vmul.i32 q6,q1,q3 // Cannot overlap with prior
  le lr,loop
```

```
loop:
  vadd.i32 q0,q1,q3
  vadd.i32 q2,q1,q3
  vldrw.u32 q1,[r1],#16
  vstrw.32 q0,[r2],#16
  vldrw.u32 q4,[r1],#16
  vstrw.32 q2,[r2],#16
  vadd.i32 q0,q1,q3 // Cannot overlap with prior
  le lr,loop
```

```
loop:
  vadd.i32 q0,q1,q3
  vadd.i32 q2,q1,q3
  vldrw.u32 q1,[r1],#16
  vstrw.32 q0,[r2],#16
  vldrw.u32 q4,[r1],#16
  vadd.i32 q0,q1,q3 // Overlaps with prior
  le lr,loop
```

- When sequences of LDST instructions cannot be made to overlap (adjacent loads or stores occur in the sequence), the sequence will overlap with a following SIMD instruction, only if it contains an even number of instructions from the point of non-overlap within the LDST sequence, or a single instruction.

```
loop:
  vadd.i32 q0,q1,q3
  vadd.i32 q2,q1,q3
  vldrw.u32 q1,[r1],#16
  vldrw.u32 q4,[r1],#16
  vmul.i32 q6,q1,q3 // Overlaps with prior
  le lr,loop
```

```
loop:
  vadd.i32 q0,q1,q3
  vmul.i32 q2,q1,q3
  vldrw.u32 q1,[r1],#16
  vldrw.u32 q4,[r1],#16
  vstrw.32 q0,[r2],#16
  vldrw.u32 q4,[r1],#16
  vadd.i32 q0,q1,q3 // Can't overlap with prior
  le lr,loop
```

```
loop:
  vadd.i32 q0,q1,q3
  vadd.i32 q2,q1,q3
  vstrw.32 q0,[r2],#16
  vstrw.32 q2,[r2],#16
  vldrw.u32 q4,[r1],#16
```

```
    vadd.i32 q0,q1,q3 // Overlaps with prior
    le lr,loop
```

- With scalar load and store bandwidth well matched with MVE loads and stores, trade-offs can be considered in employing MVE scatter loads and stores vs. multiple scalar operations to/from vector registers. Factors to consider include the relative number of free scalar and vector registers available to hold addresses and the ease of arranging access patterns to avoid bank conflicts.
- Scatter-gather operations are unable to forward base register updates to subsequent instructions, potentially leading to additional stalls. In some code sequences, it could be advantageous to use immediate offset values to achieve the intended addressing where possible, and potentially to use a separate vector add instruction to update the base register if it overlaps well with surrounding code.
- Vector floating-point multiply accumulate instructions (VCMLA, VMLA, VFMA, VFMS) impose an additional structural hazard cycle on EPU issue between their multiply and add operations. In some cases performance can be optimized by breaking up the operation into the corresponding separate vector floating-point multiply and add instructions. This offers the potential of distributing the operation between other instructions which can overlap, for example when vector load and store instructions are plentiful in the code sequence. Note that for those instructions which fuse the multiply and addition instructions, the separate rounding operation occurring within the separate instructions must be considered.

Appendix A Revisions

This appendix gives the technical changes between released issues of this book.

A.1 Revisions

The following tables show any significant technical changes between released issues of this book.

Table A-1: Issue 0101-01

Change	Location
First release	-