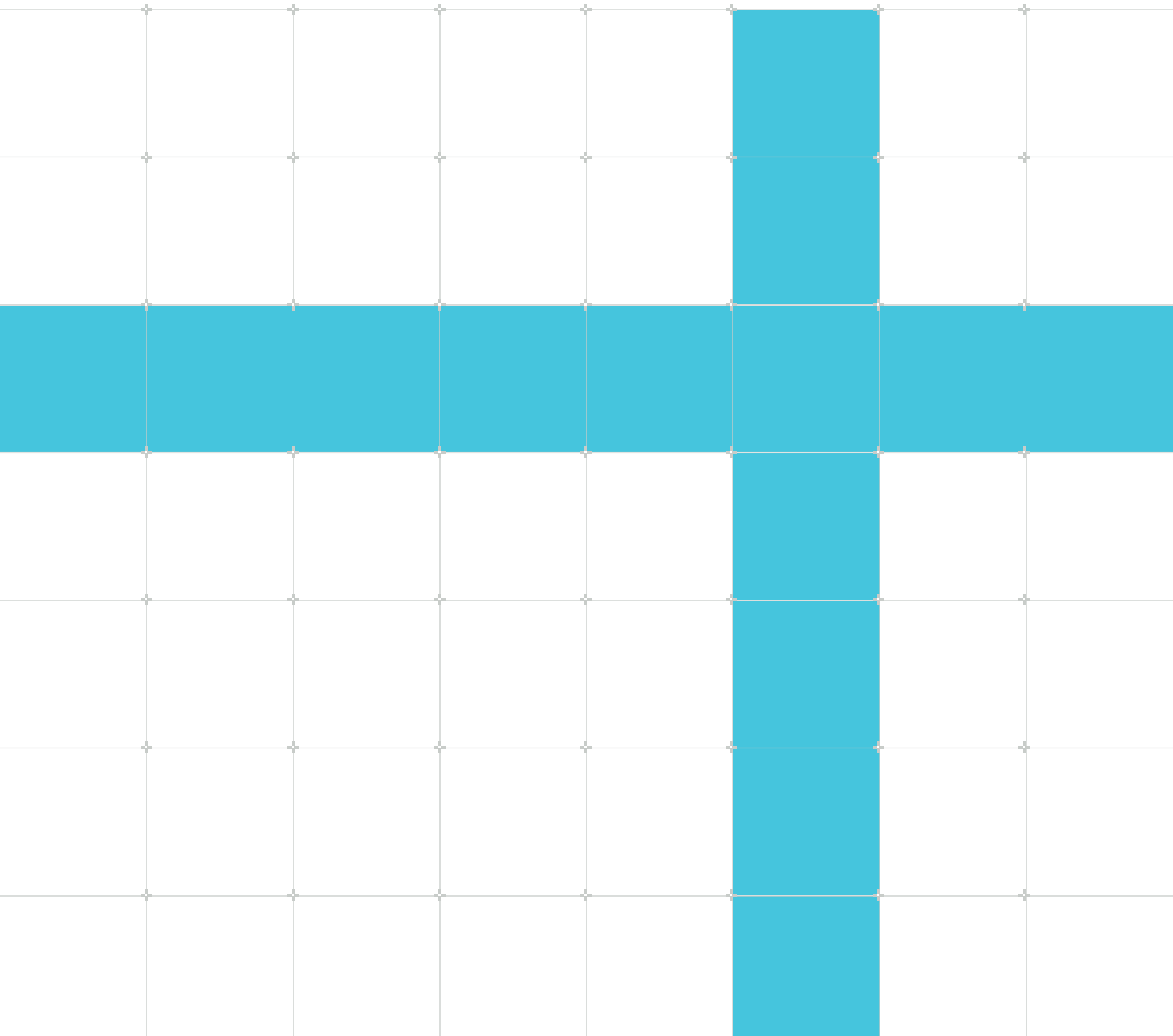
Arm® Cortex®-M55 Processor

Revision: r1p1

Software Optimization Guide

Issue 03
102692_0101_03_en



Arm® Cortex®-M55 Processor

Software Optimization Guide

Release Information

Document history

Issue	Date	Confidentiality	Change
0101-01	11 January 2022	Confidential	First release for r1p1
0101-02	7 June 2022	Confidential	Second release for r1p1
0101-03	23 December 2022	Non-Confidential	First non-confidential release for r1p1

Proprietary Notice

REGARDLESS OF THE THEORY OF LIABILITY, ARISING OUT OF ANY USE OF THIS DOCUMENT, EVEN IF ARM HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

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

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

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.

Typographic conventions

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

1.2 Other information

See the Arm® website for other relevant information.

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

1.3 Useful resources

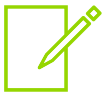
This document contains information that is specific to this product. See the following resources for other useful information.

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®-M55 Processor Devices Generic User Guide	101925	Non-Confidential
Arm® Cortex®-M55 Processor Technical Reference Manual	101051	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	SBN: 978-1-911531-23-4	Non-Confidential
Helium Programmer's Guide: Introduction to Helium	102102	Non-Confidential



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®-M55 processor.

2.1 Cortex®-M55 processor overview

The Cortex®-M55 processor is a fully synthesizable mid-range 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®-M55 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.

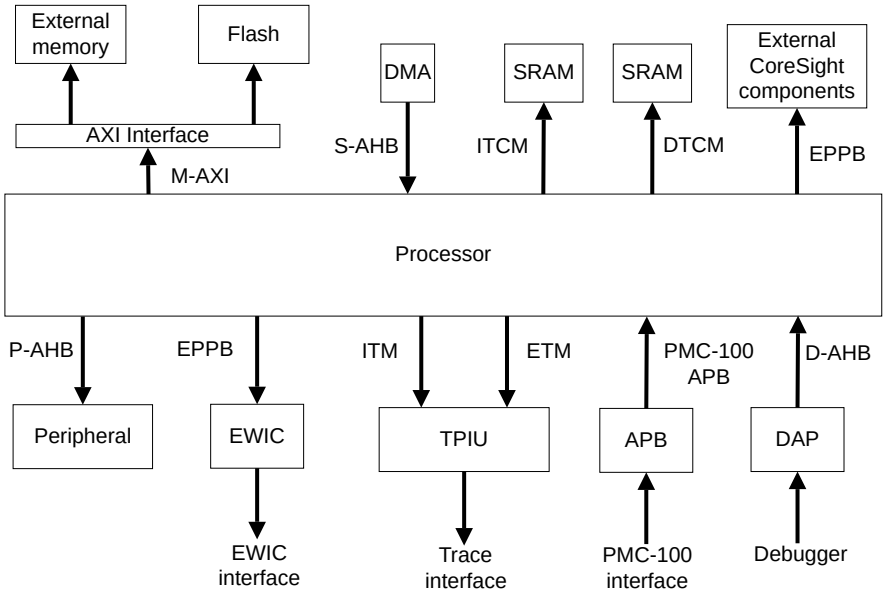
To support *Arm Custom Instructions* (ACI), the processor includes optional *Custom Datapath Extension* (CDE) modules, which are embedded inside the logic. These modules are used to execute user-defined instructions that work on general-purpose integer, floating point, and MVE registers.


Note

Where CDE is mentioned in this document, it is referring to the support of *Arm Custom Instructions* (ACI).

The following figure shows the Cortex®-M55 processor in a typical system.

Figure 2-1: Example processor system



Terms and abbreviations

The following table defines some important terms and abbreviations used in this document.

Table 2-1: Terms and definitions

Term	Expansion or Definition
MVE	M-profile Vector Extension It is also referred to as Arm Helium™ technology.
EPU	Extention Processing Unit It contains Vector Register File and performs scalar floating-point operations, and M-profile Vector Extension (MVE) operations. For more information, see *Arm® Cortex®-M55 Processor Technical Reference Manual* (101051)
DPU	Data Processing Unit It contains General Propose Register file and performs scalar integer instructions. For more information, see *Arm® Cortex®-M55 Processor Technical Reference Manual* (101051)
ERF	Extention Register File It is also known as Vector Register File. For more information, see *Arm®v8-M Architecture Reference Manual* (DDI 0553)

Term	Expansion or Definition
SRF	Scalar Register File It is also known as General Propose Register (GPR) file. For more information, see *Arm®v8-M Architecture Reference Manual* (DDI 0553)
Beat	MVE concept. The execution of ¼ of a vector operation. Because the vector length is 128 bits, one beat of a vector add instruction equates to computing 32 bits of result data. For more information, see *Arm®v8-M Architecture Reference Manual* (DDI 0553)
Tick	MVE concept. One architecture tick is an atomic unit of execution in an MVE implementation. Cortex®-M55 processor is a 2-beat per tick machine. That means each tick executes 2 beats of the MVE instruction. For more information, see *Arm®v8-M Architecture Reference Manual* (DDI 0553)
Scalar instructions	Instructions that do not read or write Vector Register File ERF, that is, they only read and write SRF.
MVE scalar instruction	MVE instructions that do not read or write MVE register bank ERF, that is, they only read and write SRF.

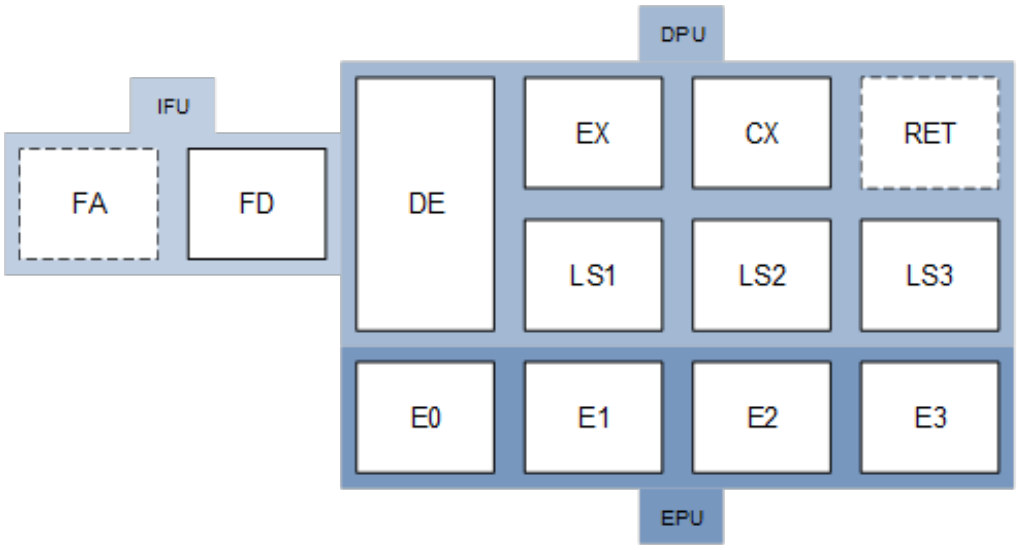
2.2 Pipeline overview

The Cortex®-M55 processor pipeline is 4-stages deep for integer instructions and 5-stages deep for *Floating Point* (FP) and *M-Profile Vector Extension* (MVE) instructions.

The following diagram describes the high-level Cortex®-M55 processor pipeline. The pipeline can be partitioned to three parts:

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

Figure 2-2: Cortex®-M55 processor Core and EPU pipeline structure



Instructions are first fetched, then decoded, and then issued into one of three execution pipelines. The processor is fully in-order and therefore any stalls in the decode or execution stages will prevent all instructions from progressing. FA and RET stage are symbolic stages and do not have registers. These stages do not count as part of pipeline depth, and they are represented as dotted-lined blocks in Figure 2-2: Cortex-M55 processor Core and EPU pipeline structure on page 12.

Table 2-2: Cortex®-M55 processor Core and EPU pipeline structure

Stage	Description
FA	IFU Fetch Address stage The FA stage contains the logic required to present addresses to an instruction memory or TCM either from a branch or based on a sequential address from a previous fetch. Branches can be generated from the DE, EX, CX and RET stages of the Core pipeline depending on the operation forcing the change of PC. The FA stage detects loop end for Armv8.1-M low-overhead loop operation. The IFU always fetches 32-bits of data from memory which could consist of up to 1 32-bit Thumb instruction or 2 16-bit Thumb instructions.
FD	IFU Fetch Data stage The FD stage accepts data from an Instruction cache or TCM and either issues it to the main pipeline or stores it in an instruction queue. The FD stage can issue 1 32-bit Thumb instruction or up to 2 16-bit Thumb instructions, for dual-issue, to the Core Decode stage, which is described in 3. Instruction latencies on page 16.

DE	DPU DEcode stage The DE stage comprises the main decode logic together with register read for the main operands of most instructions and hazard logic for the remainder of the pipeline. The decoder can handle all scalar integer single and dual issue cases. Floating point and MVE operations are dispatched to the EPU E0 stage for further processing. Three register read ports can be used for scalar arithmetic, two for single issue, and the third for arithmetic dual-issue cases. The pipeline support forwarding of results from the EX stage and CX state into DE for arithmetic instructions. Load and store address operands are constructed from both scalar register reads and from the extended register port read in the E0 stage (for MVE instructions where the base address is taken from a vector, Qn). When MVE is included the DE stage supports two memory read operations for scatter or gather instructions. The stage also contains the sequencer required to handle multi-cycle operations associated with load and store multiple and double instructions as well as the separate sequencer required to carry out MVE scatter/gather operations to memory. The DE stage also carries out the PC change for conditional and unconditional indirect and function return (BX LR) branches.
EX	DPU Simple EXecute stage The EX stage handles most scalar arithmetic, logical, and bit-shift operations. EX also contains the first stage of the integer divider. This stage reads data from the register bank or CX stage for memory store operations and accumulate data for the scalar multiplier. The EX stage carried out further branch operations, BX Rm, and CB{N}Z. Branches which require results from the ALU calculated the PC in EX and pass the result to the FA stage in the next cycle. All operations which can complete their computation in EX terminate in the stage and write back the result directly to the register bank.
CX	DPU Complex eXecute stage This stage includes a second ALU which is used to handle all the SIMD and saturating instructions, along with a few complex instructions from the regular instruction set. The stage also includes the integer multiplier and second stage of the integer divider. Results from the CX stage are written back to the register bank using two dedicated 32-bit write ports, usually in parallel with writes from a following instruction in EX. Most integer arithmetic instructions only use one of the write ports, however both can be used to write 64 bits of data a limited set of scalar instructions, including Long Multiply and Long Multiply Accumulate instructions, register transfer from the EPU and external coprocessor interface when executing MRRC. The data phase of all load and store operations is synchronous to the CX stage of the pipeline. Scalar Load data prepared by the LSU LS2 stage is written back using the two register write-ports. Vector load data is passed through CX and sent to the EPU E2 stage for write-back to the Extended register file. Store data from the main register bank read in EX is combined with data from the EPU and registered into the RET stage. Branches based on load results from the LSU, including LDR PC, [x] and LDM/POP {...., PC} and TBB/TBH obtain new address and transfer to FA in the next cycle. **Note:** For load multiple operations which branch, the PC is loaded from the memory first.
RET	DPU Retire stage The RET stage is used to pass store data, which has been read from either the main register bank in EX or from the EPU extended register bank in E1 to a TCM or Data cache store buffers.

LS1	Load-store address stage Memory requests from the DPU are distributed to the appropriate structures and interfaces in the memory system including the Data cache and M-AXI interface, TCM, the P-AHB interface, internal peripherals in the PPB memory region and the EPPB interface. The interface selection is carried out in the LS1 stage – access to data cache and TCM RAM are carried out speculatively. Access to other interfaces like Device memory cannot be speculative and are carried out later when the instruction has been committed. This stage also converts unaligned access to constituent aligned access requests. During execution of these sub-requests, the original instruction is stalled in the EX stage of the DPU.
LS2	Load-store read data phase The LS2 stage corresponds to the data phase of the cache and TCM RAM for read accesses. Cache hit/miss result determines whether an M-AXI access is required. This LSU stalls the DPU in this stage until read data is available.
LS3	Load-store write transfer stage The LS3 stage is used for formatting and transferring store data from the DPU to the appropriate interfaces.
E0	EPU Decode and address transfer stage The E0 stage decodes MVE and floating-point instructions dispatched from the DE stage. It is also used for reading base address for MVE load and store operations from the Extended register file for computation of final address in the EX stage. This stage also: • Detects and handles hazards for instruction overlap for beat-wise MVE execution. • Handles multi-cycle operations (for example, double precision floating point arithmetic). **Note:** The EPU pipeline always operates in lock-step with the main DPU pipeline.
E1	EPU Operand register read stage The Extension register file is read for all arithmetic operands in the E1 stage. Results from E2 and E3 can be forwarded to following instructions of the same class, but is not supported between different classes that is, between floating point to fixed point instructions or vector to scalar floating-point instructions. Scalar operands are received from the DPU when required. Store data is also read in the E1 stage and registered into E2 where it is passed over to the DPU to be written to memory.

E2	EPU arithmetic and logic stage The E2 stage contains the structures used to carry out vector operations on all data types and scalar operations on all floating-point data type. The majority of arithmetic and bitwise logic operations (Scalar instruction or 2 beats of an MVE instruction) complete in a single cycle apart from: • Divide and square root • Operations on double-precision data types • Instructions which produce a scalar result across a vector • The chained variant of scalar floating point multiply-accumulate, VMLA.F{32,16}. Chained multiply-accumulate is carried out as a multiply operation followed by an add operation in serial in E2 with full rounding after each operation in E3. Double precision operations which require multiple iterations are recirculated until the full result is available. The E2 stage is also used to transfer extended registers from the EPU to the CX stage of the DPU. The VPR and flags in FPSCR.NZCV are updated in E2 stage. Load data is returned from the DPU CX stage in E2 and registered to the E3. The load data-path is separated into 8 byte-lanes support vector predications on write-back.
E3	EPU write-back stage The results for all EPU operations are written back to the extended register file in E3 including load data transferred from the memory system in E2 through the CX stage. All vector writes in E3 are forwarded back to E1. MVE operations involving scalar destinations write-back their results to the register file in the DPU in E3.

3. Instruction latencies

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

3.1 Instruction tables

A series of tables summarizes the effective execution latency and throughput, pipelines utilized, dual-issue ability, and special behaviors associated with each group of instructions. Cortex®-M55 processor supports limited dual-issue ability on 16-bit Thumb instructions.

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®-M55 processor microarchitecture.
- Cortex®-M55 processor has 2 slots to dual issue for certain 16-bit Thumb instructions. 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
- Cortex®-M55 processor is a 2 beat per tick machine, and it supports overlapping up to two beatwise MVE instructions at any time. That means, an MVE instruction can be issued after another MVE instruction without additional stall. The beatwise MVE instruction can be overlapped if they are using different utilized pipelines. Utilized pipelines correspond to the execution pipelines in EPU. There are:
  - *System Registers Pipe* (SY)
  - *Load/Store Pipe* (LS)
  - *Integer Vector Pipe* (I)
  - *Floating Point Pipe* (F)

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	Notes
Branch Future	BF (T1) BFCSEL (T2) BFL (T4) BFLX (T5) BFX (T3)	1	1	1
Branch Immediate	B (T3) B (T4)	1(2)	1(1/2)	2
Branch Immediate	BL (T1)	2	1/2	-
Low Overhead Loops	DLS (T2) DLSTP (T4) LCTP (T1)	1	1	-
Low Overhead Loops	LE (T1) LE (T2) LETP (T3)	3	1/3	-
Low Overhead Loops (While)	WLS (T1) WLSTP (T3)	1(3)	1(1/3)	3

Notes:

1 Acts as a NOP

2 If the branch immediate is a backwards branch, subsequent branches are predicted to be taken and the latency reduces to 0 as the branch is implied.

3 If the while loop is not executed, a branch occurs which results in a 3 cycle penalty in latency.

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(2)	2	11	1
Branch Immediate	BX (T1)	2	1	11	2
Branch Immediate	CBNZ, CBZ (T1)	3	1	00	-
Branch Register	BXNS (T1)	3	1	00	-

Instruction group	16-bit Thumb instructions	Execution latency	Execution throughput	Dual-issue	Notes
Branch Register	BLX, BLXNS (T1) BLXNS (T1)	3	1	01	-
Branch, register (with destination LR)	BX (T1)	3(2)	1	11	3

Notes:

1 If the branch immediate is a backwards branch, subsequent branches are predicted to be taken and the latency reduces to 0 as the branch is implied.

2 Branch Exchange instructions using the LR execute with a reduced latency because of a late-forwarding path implemented for the LR.

3 A conditional branch instruction can be dual-issued as the first instruction in a pair only if the first instruction is an unconditional immediate branch (B[T2]). If the branch immediate is a backwards branch, subsequent branches are predicted to be taken and the latency reduces to 0 as the branch is implied.

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	Notes
Add operations	ADC (immediate) (T1) ADR (T2) ADR (T3) CMN (immediate) (T1) CMP (immediate) (T2)	1	1	-
Add operations	ADD (SP plus register) (T3) SUB (SP plus register) (T3)	1(2)	1(1/2)	1
ALU SP operations	ADD SP (immediate) (T3) ADDW SP (immediate) (T4) SUB SP (immediate) (T3) SUBW SP (immediate) (T4)	1	1	-

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

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

Instruction group	32-bit Thumb instructions	Execution latency	Execution throughput	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) USAD8 (T1) USADA8 (T1) USAT16 (T1) USAX (T1) USUB16 (T1) USUB8 (T1)	2	1	-
Sign Extend Addition	SXTB (T2) SXTH (T2)	1	1	-
Sign Extend Addition	SXTAB (T1) SXTAB16 (T1) SXTAH (T1) SXTB16 (T1)	2	1	-

Instruction group	32-bit Thumb instructions	Execution latency	Execution throughput	Notes
Signed Addition	SSAT (T1)	1	1	-
Signed Addition	SADD16 (T1) SADD8 (T1) SASX (T1) SHADD16 (T1) SHADD8 (T1) SHASX (T1) SHSAX (T1) SHSUB16 (T1) SHSUB8 (T1)	2	1	-
Subtract operations	RSB (immediate) (T2) SBC (immediate) (T1)	1	1	-
Test operations	TEQ (immediate) (T1) TST (immediate) (T1)	1	1	-
Test operations	TT, TTT, TTA, TTAT (T1)	2	1	-
Unsigned Addition	UADD16 (T1) UADD8 (T1) UASX (T1) UHADD16 (T1) UHADD8 (T1) UHASX (T1) UHSAX (T1) UHSUB16 (T1) UHSUB8 (T1)	2	1	-
Zero Extend Addition	UXTB (T2) UXTH (T2)	1	1	-
Zero Extend Addition	UXTAB (T1) UXTAB16 (T1) UXTAH (T1) UXTB16 (T1)	2	1	-

Notes:

1 If the shift amount is non-zero, then the latency is 2 and the throughput is 1. In addition, if the result is then written to the SP, the result is recycled in EX to perform the stack limit checks so the latency is 2 and the throughput is 1/2. Otherwise, the latency and throughput are 1.

2 If the shift amount is non-zero, then the latency is 2 and the throughput is 1. Otherwise, the latency and throughput are 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	ADD (register) (T2)	1	1	01(00)	1
Add operations	ADC (register) (T1) ADD (SP plus immediate) (T2) ADD (register) (T1) ADR (T1)	1	1	01	-
Add operations	ADD (SP plus immediate) (T2) ADD (SP plus register) (T2)	2	1/2	01	2
Add operations	ADD (SP plus immediate) (T1) ADD (immediate) (T1) ADD (immediate) (T2)	1	2	11	-
Add operations	ADD (SP plus immediate) (T1) ADD (SP plus register) (T1)	2	1/2	11	2
Basic ALU	CMN (register) (T1)	1	1	00	-
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	1	01	-
Basic ALU	CMP (immediate) (T1)	1	2	11	-

Instruction group	16-bit Thumb instructions	Execution latency	Execution throughput	Dual-issue	Notes
Sign Extend Addition	SXTB (T1) SXTH (T1)	1	2	11	-
Subtract operations	RSB (immediate) (T1) SBC (register) (T1) SUB (SP minus immediate) (T1) SUB (register) (T1)	1	1	01	-
Subtract operations	SUB (immediate) (T1) SUB (immediate) (T2)	1	2	11	-
Zero Extend Addition	UXTB (T1) UXTH (T1)	1	2	11	-

Notes:
1 Does not dual issue when Rd=PC or Rm=PC
2 When an ADD SP is performed, the result is recycled in EX to perform the stack limit checks. This will result in a bubble being created in the pipeline.

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	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	1	-

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	1	01	-
Basic Move operations	MOV (immediate) (T1)	1	2	11	-
Basic Move operations	MOV (T1)	1(4)	2(1/4)	11(00)	1

Notes:
1 MOV PC, Rm can only be single issued.

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	Notes
Divide	SDIV (T1) UDIV (T1)	2-12	1/11-1	1
Multiply	MUL (T2)	2	1	-

Instruction group	32-bit Thumb instructions	Execution latency	Execution throughput	Notes
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) SMMUL, SMMULR (T1) SMUAD, SMUADX (T1) SMULBB, SMULBT, SMULTB, SMULTT (T1) SMULL (T1) SMULWB, SMULWT (T1) SMUSD, SMUSDX (T1) SSAT16 (T1) SSAX (T1) SSUB16 (T1) SSUB8 (T1) UMAAL (T1) UMLAL (T1) UMULL (T1)	2	1	-

Notes:

1 Divides are performed using an iterative algorithm, and block any subsequent divide operations until complete. Early termination is possible, depending

Notes:

upon the data values. There are 2 main cases: (1) If it is divide-by-zero, the operation will have 2 cycle latency and 1 instruction per cycle throughput. (2) For other cases, let DIFF_SIGN be (Count_leading_sign_bit(Denominator) - Count_leading_sign_bit(Numerator)) where Count_leading_sign_bit counts leading zeros for UDIV. If DIFF_SIGN is less than zero, the operation will have 3 cycle latency and 1/2 instruction per cycle throughput. If DIFF_SIGN is equal or greater than 0, the operation will have latency of (4 + Round_up (DIFF_SIGN/4)) and throughput of (1/(3+ Round_up (DIFF_SIGN/4))).

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	01	-

3.6 Load instructions

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

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

Instruction group	32-bit Thumb instructions	Execution latency	Execution throughput	Notes
Basic Loads	LDA (T1) LDR (immediate) (T3) LDR (immediate) (T4) LDR (literal) (T2) LDR (register) (T2)	2	1	-
Exclusive operations	LDAEX (T1) LDAEXB (T1) LDAEXH (T1) LDREX (T1) LDREXB (T1) LDREXD (T1) LDREXH (T1)	2	1	-
Load Multiples	LDRD (immediate) (T1) LDRD (literal) (T1)	2	1	1

Instruction group	32-bit Thumb instructions	Execution latency	Execution throughput	Notes
Load Multiples	LDM, LDMIA, LDMFD (T2) LDMDB, LDMEA (T1)	N+1	1/N	1
Sub Word Loads	LDAB (T1) LDAH (T1) 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)	2	1	-

Notes:

1 Cortex-M55 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	1	01	-
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)	2	1	01	-

Notes:

1 Cortex-M55 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.

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

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

Instruction group	32-bit Thumb instructions	Execution latency	Execution throughput	Notes
Store Lock Release	STL (T1) STLB (T1) STLEX (T1) STLEXB (T1) STLEXH (T1) STLH (T1)	2	1	-
Store Multiple	STRD (immediate) (T1)	2	1	1
Store Multiple	STM, STMIA, STMEA (T2) STMDB, STMFD (T1)	N+1	1/N	1
Sub Word Stores	STRB (immediate) (T2) STRB (immediate) (T3) STRB (register) (T2) STRH (immediate) (T2) STRH (immediate) (T3) STRH (register) (T2)	2	1	-

Notes:

1 Cortex-M55 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	1	01	-
Store Multiple	PUSH (multiple registers) (T2) STM, STMIA, STMEA (T1)	N+1	1/N	00	1
Sub Word Stores	STRB (immediate) (T1) STRB (register) (T1) STRH (immediate) (T1) STRH (register) (T1)	2	1	01	-

Notes:

1 Cortex-M55 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	Notes
Hints	PLI (immediate, literal) (T1) PLI (immediate, literal) (T2) PLI (immediate, literal) (T3) PLI (register) (T1)	1	1	1
Hints	PLD (literal) (T1) PLD, PLDW (immediate) (T1) PLD, PLDW (immediate) (T2) PLD, PLDW (register) (T1)	1	1	-
No Operation	NOP (T2)	1	1	-
Register updates	CLRM (T1)	N+1	1/N	2

Notes:

1 Acts as a NOP.

2 CLRM supports clearing 2 registers per cycle. N=floor((num_regs+1)/2).

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)	1	2	11	-
Program Flow Control	IT (T1)	1	2	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	Notes
Continuous Vector Load	VLDR (T2) VLDR (T3)	2	1	-
Continuous Vector Load	VLDR (T1)	2	1	2
Continuous Vector Load	VLDM (T1) VLLDM (T1) VSCCLRM (T1)	(N/2)+1	1/((N/2)+1)	2
Divide (Double-precision)	VDIV (T1)	29	1/28	3
Divide (Half-precision)	VDIV (T1)	9	1/8	3
Divide (Single-precision)	VDIV (T1)	16	1/15	3
Divide (all-precision) with Input Zero/Infinite/ NaN or Invalid Operation	VDIV (T1)	6	1/5	3
Scalar Absolute	VABS (T2)	2	1	-
Scalar Arithmetic	VADD (T1) VSUB (T1)	2(15)	1(1/14)	1
Scalar Arithmetic	VMAXNM (T1)	2	1	-
Scalar Compare	VCMP (T1) VCMP (T2)	2	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 (T1) VRINTA, VRINTN, VRINTP, VRINTM (T1) VRINTR, VRINTZ (T1) VRINTX (T1)	2	1	-

Instruction group	Instructions	Execution latency	Execution throughput	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 a doubleword register) (T1) VMOV (between two general-purpose registers and two single-precision registers) (T1) VMOV (immediate) (T2) VMOV (register) (T1) VMOVX (T1)	2	1	-
Scalar Multiply	VMUL (T1) VNMUL (T2)	2(21)	1(1/20)	1
Scalar Multiply	VFMA (T1) VFNMA (T1)	2(24)	1(1/23)	1
Scalar Multiply	VMLA (T1) VNMLA (T1)	4(36)	1/3(1/35)	1
Scalar Negate	VNEG (T1)	2	1	-
Scalar Select	VSEL (T1)	2	1	-
Square Root (Double-precision)	VSQRT (T1)	29	1/28	3
Square Root (Half-precision)	VSQRT (T1)	9	1/8	3
Square Root (Single-precision)	VSQRT (T1)	16	1/15	3
Square Root (all-precision) with Input Zero/ Infinite/NaN or Invalid Operation	VSQRT (T1)	6	1/5	3
Store	VSTR (T2) VSTR (T3)	1	1	-
Store	VSTR (T1)	2	1	2
Store	VLSTM (T1) VSTM (T1)	(N/2)+1	1/((N/2)+1)	2

Notes:

1 Double-precision variants run as longer multiple-cycle instructions. The latency and throughput of these instructions are specified inside the parentheses.

Notes:

2 Cortex-M55 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 are performed using an iterative algorithm and block any subsequent divide operations until complete.

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	Utilized pipeline	Notes
MVE Absolute	VABAV (T1)	5	1/4	I & F	1
MVE Absolute	VABD (T1) VABS (T1) VQABS (T1)	1	1/2	I	-
MVE Arithmetic	VMAXV, VMAXAV (T1) VMAXV, VMAXAV (T2) VMINV, VMINAV (T1) VMINV, VMINAV (T2)	(64/esize) + 1	esize/64	I	2

Instruction group	Instructions	Execution latency	Execution throughput	Utilized pipeline	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	I	-

Instruction group	Instructions	Execution latency	Execution throughput	Utilized pipeline	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	1/2	I	-
MVE CLS/CLZ	VCLS (T1) VCLZ (T1)	1	1/2	I	-
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	F	-

Instruction group	Instructions	Execution latency	Execution throughput	Utilized pipeline	Notes
MVE Duplicate	VDDUP, VDWDUP (T1) VDDUP, VDWDUP (T2) VIDUP, VIWDUP (T1) VIDUP, VIWDUP (T2)	1	1/2	I	3
MVE Duplicate	VDUP (T1)	1	1/2	I	-
MVE MOV	VMOVL (T1) VMOVN (T1)	1	1/2	I	-

Instruction group	Instructions	Execution latency	Execution throughput	Utilized pipeline	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) VMULL (polynomial) (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	F	-
MVE Negate	VNEG (T1) VQNEG (T1)	1	1/2	I	-
MVE Select	VPSEL (T1)	1	1/2	I	-

Instruction group	Instructions	Execution latency	Execution throughput	Utilized pipeline	Notes
MVE Shift	VBRSR (T1)	1	1/2	I	-
	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 (T1)				
	VSHL (T2)				
	VSHL (T3)				
	VSHLC (T1)				
	VSHLL (T1)				
	VSHLL (T2)				
	VSHR (T1)				
	VSHRN (T1)				
	VSLI (T1)				
	VSRI (T1)				

Instruction group	Instructions	Execution latency	Execution throughput	Utilized pipeline	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)	3(2)	1/2	F	3

Notes:

1 The first cycle is executed in the I pipe. The second cycle is executed in the F pipe. Therefore, for fully overlapped execution, the older instruction should use the non-I pipe, and the younger instruction should use the non-F pipe.

2 Refer to 4.1.2 SRF write port hazard E3 on page 47. esize is the element size of the instruction variant. It could be 8, 16, or 32, depending on the variant this instruction supports.

3 Refer to 4.1.2 SRF write port hazard E3 on page 47.

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	Utilized pipeline	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	I/F	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	F	2

Instruction group	Instructions	Execution latency	Execution throughput	Utilized pipeline	Notes
Scalar Shift	ASRL (immediate) (T1) ASRL (register) (T1) LSLL (immediate) (T1) LSLL (register) (T1) LSRL (immediate) (T1) SQRSHR (register) (T1) SQRSHRL (register) (T1) SQSHL (immediate) (T1) SQSHLL (immediate) (T1) SRSHR (immediate) (T1) SRSHRL (immediate) (T1) UQRSHL (register) (T1) UQRSHLL (register) (T1) UQSHL (immediate) (T1) UQSHLL (immediate) (T1) URSHR (immediate) (T1) URSHRL (immediate) (T1)	2(5)	1(1/4)	I	3

Notes:

1 Issue to the I pipe, if available; otherwise, issue to the F pipe.

2 Refer to 4.1.1 SRF write port hazard E2 on page 46.

3 When CPACR.CP10 != 0b00, the instruction is executed in the EPU with shorter latency and larger throughput. When CPACR.CP10 == 0b00, the instruction is executed in the DPU with longer latency and smaller throughput with the value in the parentheses.

3.12 MVE FP instructions

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

Table 3-18: Latency and throughput information for MVE FP instructions

Instruction group	Instructions	Execution latency	Execution throughput	Utilized pipeline	Notes
MVE Absolute	VABD (floating-point) (T1) VABS (floating-point) (T1)	2	1/2	F	-
MVE Arithmetic	VMAXNMV, VMAXNMAV (floating-point) (T1) VMAXNMV, VMAXNMAV (floating-point) (T2) VMINNMV, VMINNMAV (floating-point) (T1) VMINNMV, VMINNMAV (floating-point) (T2)	(64/esize) + 1	esize/64	F	1
MVE Arithmetic	VADD (floating-point) (T1) VADD (floating-point) (T2) VCADD (floating-point) (T1) VMAXNM, VMAXNMA (floating-point) (T1) VMAXNM, VMAXNMA (floating-point) (T2) VMINNM, VMINNMA (floating-point) (T1) VMINNM, VMINNMA (floating-point) (T2) VSUB (floating-point) (T1) VSUB (floating-point) (T2)	2	1/2	F	-
MVE Compare	VPT (floating-point) (T1) VPT (floating-point) (T2)	2	1/2	F	2
MVE Compare	VCMP (floating-point) (T1) VCMP (floating-point) (T2)	2	1/2	F	-
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	F	-

Instruction group	Instructions	Execution latency	Execution throughput	Utilized pipeline	Notes
MVE Multiply	VCMLA (floating-point) (T1) VCMUL (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) VMUL (floating-point) (T1) VMUL (floating-point) (T2)	2	1/2	F	-
MVE Negate	VNEG (floating-point) (T1)	2	1/2	F	-

Notes:

1 Refer to 4.1.1 SRF write port hazard E2 on page 46. esize is the element size of the instruction variant. It could be 8, 16, or 32, depending on the variant this instruction supports.

2 Cannot overlap with the following I pipe instructions, because there is no forwarding path from F to I.

3.13 MVE Miscellaneous instructions

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

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

Instruction group	Instructions	Execution latency	Execution throughput	Utilized pipeline	Notes
System	VCTP (T1) VLDR (System Register) (T1) VMRS (T1) VMSR (T1) VPNOT (T1) VPST (T1) VSTR (System Register) (T1)	1	1	SY	1

Notes:

1 Instructions with utilized pipeline SY can overlap with themselves.

3.14 MVE Load instructions

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

Table 3-20: Latency and throughput information for MVE Load instructions

Instruction group	Instructions	Execution latency	Execution throughput	Utilized pipeline	Notes
Continuous Vector Load	VLDRB, VLDRH, VLDRW (T1) VLDRB, VLDRH, VLDRW (T2) VLDRB, VLDRH, VLDRW (T5) VLDRB, VLDRH, VLDRW (T6) VLDRB, VLDRH, VLDRW (T7)	1	1/2	LS	-
Continuous Vector Load	VLD2 (T1) VLD4 (T1)	2	1/2	LS	-
Gather Load	VLDRB, VLDRH, VLDRW, VLDRD (vector) (T1)	(64/esize) + 1	esize/64	LS	1
Gather Load	VLDRB, VLDRH, VLDRW, VLDRD (vector) (T2) VLDRB, VLDRH, VLDRW, VLDRD (vector) (T3) VLDRB, VLDRH, VLDRW, VLDRD (vector) (T4)	(64/esize) + 1	esize/64	LS	2
Gather Load	VLDRB, VLDRH, VLDRW, VLDRD (vector) (T5)	1(2)	1/2	I & LS	3
Gather Load	VLDRB, VLDRH, VLDRW, VLDRD (vector) (T6)	2	1/4	I & LS	3

Notes:

1 Gather loads cannot read forwarded data, so the latency of the preceding instruction will have an additional latency of 4. esize is the element size of the instruction variant. It could be 8, 16, or 32, depending on the variant this instruction supports.

2 esize is the element size of the instruction variant. It could be 8, 16, or 32, depending on the variant this instruction supports.

3 The first cycle is executed in the LS pipe. The second cycle is executed in the I pipe. Therefore for fully overlapped execution, the older instruction should use the non-LS pipe, and the younger instruction should use the non-I pipe.

3.15 MVE Store instructions

The following tables summarize latency and throughput information for MVE Store instructions.

Table 3-21: Latency and throughput information for MVE Store instructions

Instruction group	Instructions	Execution latency	Execution throughput	Utilized pipeline	Notes
Scatter Store	VSTRB, VSTRH, VSTRW, VSTRD (vector) (T1) VSTRB, VSTRH, VSTRW, VSTRD (vector) (T2) VSTRB, VSTRH, VSTRW, VSTRD (vector) (T3) VSTRB, VSTRH, VSTRW, VSTRD (vector) (T4)	1	esize/64	LS	1
Scatter Store	VSTRB, VSTRH, VSTRW, VSTRD (vector) (T5)	2	1/2	I & LS	2
Scatter Store	VSTRB, VSTRH, VSTRW, VSTRD (vector) (T6)	2	1/4	I & LS	2
Store	VSTRB, VSTRH, VSTRW (T1) VSTRB, VSTRH, VSTRW (T2)	1	1/2	LS	1
Store	VSTRB, VSTRH, VSTRW (T5) VSTRB, VSTRH, VSTRW (T6) VSTRB, VSTRH, VSTRW (T7)	1	1/2	LS	2
Store	VST2 (T1) VST4 (T1)	1	1/2	LS	-

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. If the data to store depends on a preceding instruction, the preceding instruction latency is not affected.

2 If the data to store depends on a preceding instruction, the preceding instruction latency is not affected.

4. General behaviors

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

4.1 MVE pipeline hazard

MVE vector instructions are issued as 2 micro-ops. Each micro-op operates on 64 bits of data and is also known as a tick. Overlapping means tick1 of an MVE instruction can execute in parallel to a tick0 of the succeeding MVE instruction.

For MVE instructions, the decision of whether to overlap is made in the E0 stage. The decoded instruction is checked against the current micro-ops in its pipeline and the control determines whether this instruction can be overlapped based on resource or data availability. Therefore, if any hazards occur, an E0 stall will prevent any overlapping.

In Cortex®-M55 processor, a newer MVE instruction can overlap with the older MVE instruction if the older tick1 does not use the same pipe as the newer tick0. Utilized pipeline can be referred to in the instruction latency tables.

There are sets of scalar instructions which are allowed to overlap with tick1 of the preceding vector instruction. These are the following:

- Immediate branches (B, BL and also CB[N]Z)
- Low-overhead-loop instructions
- Branch Future (these are just NOP)
- Integer arithmetic, except DIV, CSEL (all), MVE scalar shifts, and PC modifying
- Load or store, except LSMs and PC modifying

The following micro-architectural limitations need to be considered, which can affect scalar and vector overlap:

- Arithmetic instructions cannot overlap with vector load or stores with base writeback to a scalar register.
- Any instruction which checks the stack limit does not overlap.
- A scalar cannot overlap with a vector instruction marked with an implicit LE, that is, the last instruction in a low-overhead-loop.
- If there is dependency between the scalar and vector instruction, then it is unlikely to overlap.

4.1.1 SRF write port hazard E2

Instructions that write to the SRF in E2 cannot overlap with other instructions that write to the SRF in E2. All instructions referred from latency tables to this section write to the SRF in E2.

4.1.2 SRF write port hazard E3

Instructions that write to the SRF in E3 cannot overlap with other instructions that write to the SRF in E3. All instructions referred from latency tables to this section write to the SRF in E3.

4.1.3 Memory access instructions

This section provides a high-level detail of the Memory access.

TCM banks

The TCM in Cortex®-M55 processor is 4 banks of 32-bit word per bank. If the load addresses in a vector load instruction are in the same bank, for example, 0x2001000 and 0x20010010, there will be a bank conflict, which means they cannot be loaded in the same cycle. Addresses are in the same bank if their bits [3:2] are the same. In such cases, there will be a 1 cycle penalty.

Appendix A Revisions

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

A.1 Revisions

The following tables show changes in the book issues.

Table A-1: Issue 0101-01

Change	Location
First release	-

Table A-2: Differences between issue 0101-01 and issue 0101-02

Change	Location
Updated the Pipeline description	2.2 Pipeline overview on page 11

Table A-3: Differences between issue 0101-02 and issue 0101-03

Change	Location
First non-confidential release	-