

Arm China Cortex®-M52 Processor

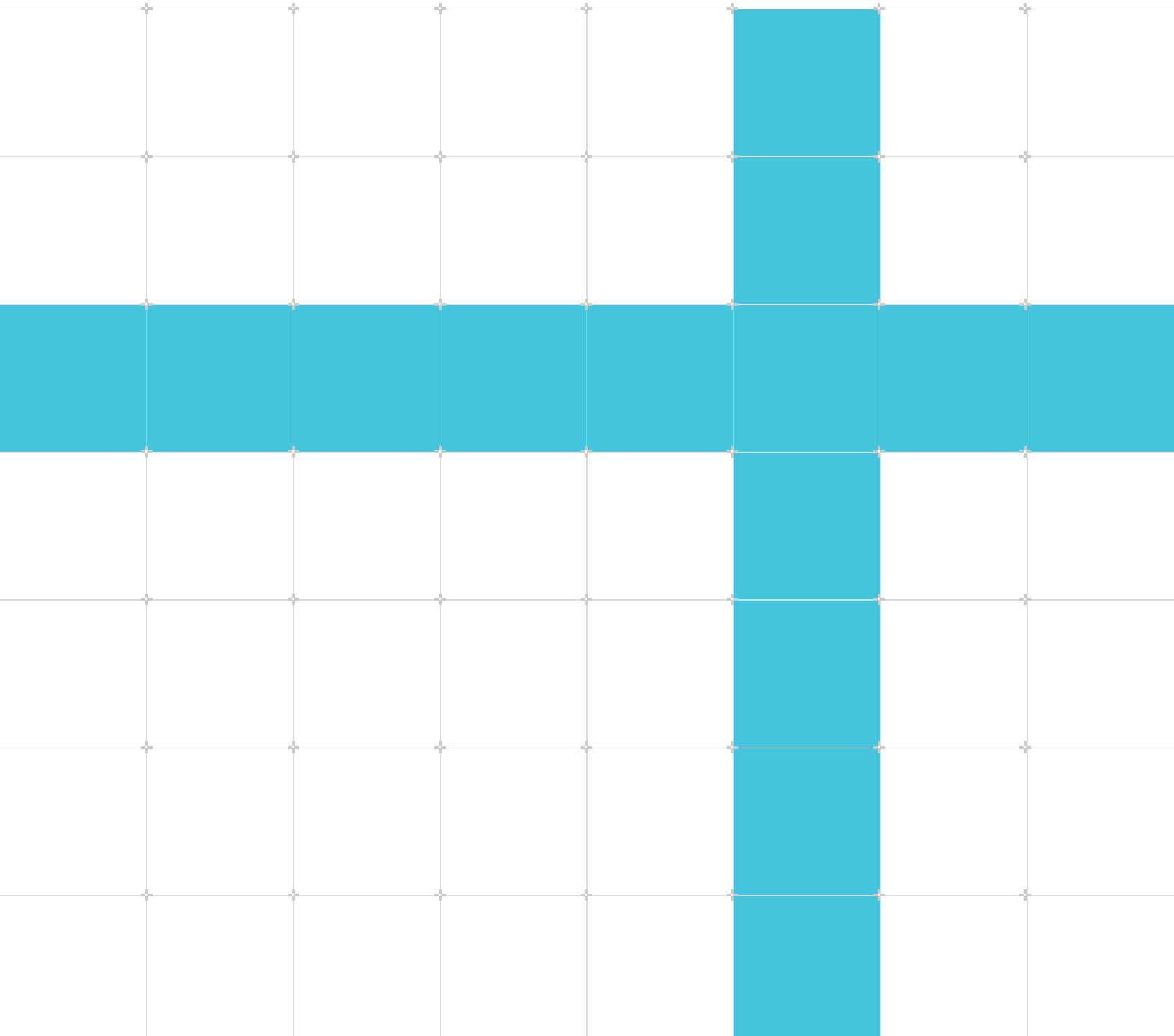
Revision: r0p3

Software Optimization Guide

Non-Confidential

Issue 03

Copyright © 2022–2024 Arm Technology (China) Co., Ltd. (or its affiliates) and Copyright © 2019–2021 Arm Limited (or its affiliates). All rights reserved. 107730_0003_03_en



Arm China Cortex®-M52 Processor

Software Optimization Guide

Copyright © 2022–2024 Arm Technology (China) Co., Ltd. (or its affiliates) and Copyright © 2019-2021 Arm Limited (or its affiliates). All rights reserved.

Release Information

Document history

Issue	Date	Confidentiality	Change
0003-03	10 May 2024	Non-Confidential	First release for r0p3
0002-02	30 September 2023	Non-Confidential	Second release for r0p2
0001-01	30 August 2022	Confidential	First release for r0p1

Proprietary Notice

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

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

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

This document may include technical inaccuracies or typographical errors.

TO THE EXTENT NOT PROHIBITED BY LAW, IN NO EVENT WILL ARM CHINA BE LIABLE FOR ANY DAMAGES, INCLUDING WITHOUT LIMITATION ANY DIRECT, INDIRECT, SPECIAL, INCIDENTAL, PUNITIVE, OR CONSEQUENTIAL DAMAGES, HOWEVER CAUSED AND REGARDLESS OF THE THEORY OF LIABILITY, ARISING OUT OF ANY USE OF THIS

DOCUMENT, EVEN IF ARM CHINA 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 China's customers is not intended to create or refer to any partnership relationship with any other company. Arm China 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 China is a trading name of Arm Technology (China) Co., Ltd. The words marked with ® or ™ are registered trademarks or trademarks of Arm Limited (or its affiliates) in the People's Republic of China and/or elsewhere. All rights reserved. Visit <https://www.arm.com/company/policies/trademarks> and <https://www.armchina.com/usestandard> for full guidance on using Arm's trademarks. Other brands and names mentioned in this document may be the trademarks of their respective owners.

Copyright © 2022-2024 Arm Technology (China) Co., Ltd. (or its affiliates).

Copyright © 2019-2021 Arm Limited (or its affiliates). All rights reserved.

Arm Limited. Company 02557590 registered in England.

110 Fulbourn Road, Cambridge, England CB1 9NJ.

Arm Technology (China) Co., Ltd. registered in China.

Room 201, Building A, No. 1 First Qianwan Road, Qianhai Shengang Cooperation Zone, Shenzhen, the People's Republic of China.

(LES-PRE-20349 - Arm China)

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 China and the party that Arm China delivered this document to.

Unrestricted Access is an Arm China internal classification.

Product Status

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

Contents

1. Introduction.....	6
1.1 Conventions.....	6
1.2 Useful resources.....	6
2. The Cortex®-M52 processor.....	8
2.1 Cortex®-M52 processor overview.....	8
2.2 Pipeline overview.....	10
3. Instruction latencies.....	15
3.1 Instruction tables.....	15
3.2 Branch instructions.....	16
3.3 Arithmetic and Logical instructions.....	17
3.4 Move and Shift instructions.....	23
3.5 Divide and Multiply instructions.....	24
3.6 Load instructions.....	26
3.7 Store instructions.....	28
3.8 Miscellaneous instructions.....	30
3.9 FP Data Processing instructions.....	31
3.10 MVE Integer Vector instructions.....	33
3.11 MVE Integer Scalar instructions.....	39
3.12 MVE FP instructions.....	40
3.13 MVE Miscellaneous instructions.....	42
3.14 MVE Load instructions.....	43
3.15 MVE Store instructions.....	44
4. Additional information.....	45
4.1 MVE pipeline hazard.....	45
4.2 Hardware prefetcher.....	46
A. Revisions.....	47

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
<i>italic</i>	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 <u>underline</u>	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: <pre>MRC p15, 0, <Rd>, <CRn>, <CRm>, <Opcode_2></pre>
SMALL CAPITALS	Terms that have specific technical meanings as defined in the Arm® Glossary. For example, IMPLEMENTATION DEFINED , IMPLEMENTATION SPECIFIC , UNKNOWN , and UNPREDICTABLE .

1.2 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 China Cortex®-M52 Processor Devices Generic User Guide	107596	Non-Confidential
Arm China Cortex®-M52 Processor Technical Reference Manual	102776	Non-Confidential

Arm product resources	Document ID	Confidentiality
Arm China Cortex®-M52 Processor Integration and Implementation Manual	102775	Confidential
<i>Getting started with Armv8.1-M based processor: software development hints and tips</i>	-	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. The Cortex®-M52 processor

This document provides guidelines on generating optimal sequence of instructions while writing the assembly code for the Cortex®-M52 processor.

2.1 Cortex®-M52 processor overview

The Cortex®-M52 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®-M52 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.

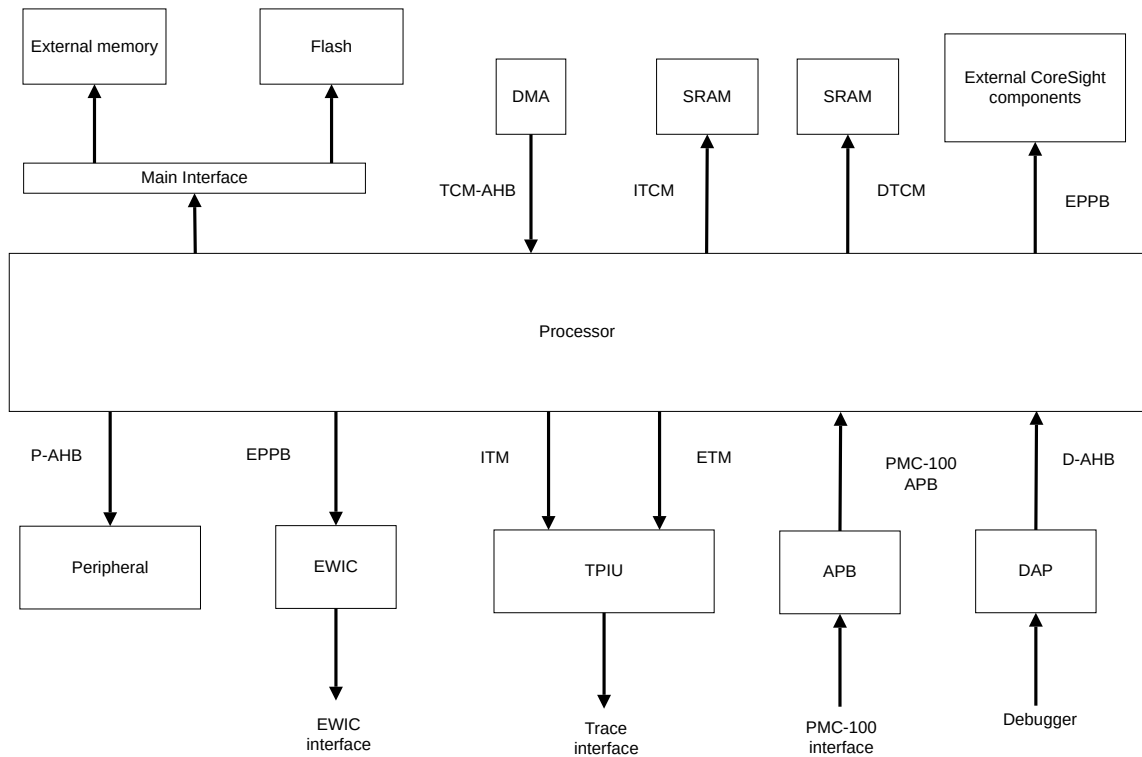
To support *Arm Custom Instructions* (ACI), the processor includes optional *Custom Datapath Extension* (CDE) modules, which are embedded inside the logic. These modules are used to execute user-defined instructions that work on general-purpose integer, floating point, and MVE registers.



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

The following figure shows the Cortex®-M52 processor in a typical system.

Figure 2-1: Example system integration



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	Extended Processing Unit It contains Vector Register File and performs scalar floating-point operations, and M-profile Vector Extension (MVE) operations. For more information, see <i>Arm China Cortex®-M52 Processor Technical Reference Manual</i> (102776)
DPU	Data Processing Unit It contains General Purpose Register file and performs scalar integer instructions. For more information, see <i>Arm China Cortex®-M52 Processor Technical Reference Manual</i> (102776)

Term	Expansion or Definition
ERF	<p>Extended Register File</p> <p>It is also known as Vector Register File.</p> <p>For more information, see <i>Arm®v8-M Architecture Reference Manual</i> (DDI 0553)</p>
SRF	<p>Scalar Register File</p> <p>It is also known as General Purpose Register (GPR) file.</p> <p>For more information, see <i>Arm®v8-M Architecture Reference Manual</i> (DDI 0553)</p>
Beat	<p>MVE concept. The execution of $\frac{1}{4}$ 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.</p> <p>For more information, see <i>Arm®v8-M Architecture Reference Manual</i> (DDI 0553)</p>
Tick	<p>MVE concept. One architecture tick is an atomic unit of execution in an MVE implementation. Cortex®-M52 processor is a 1-beat per tick machine. That means each tick executes 1 beat of the MVE instruction.</p> <p>For more information, see <i>Arm®v8-M Architecture Reference Manual</i> (DDI 0553)</p>
Scalar instructions	Instructions that do not read or write vector register bank 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.

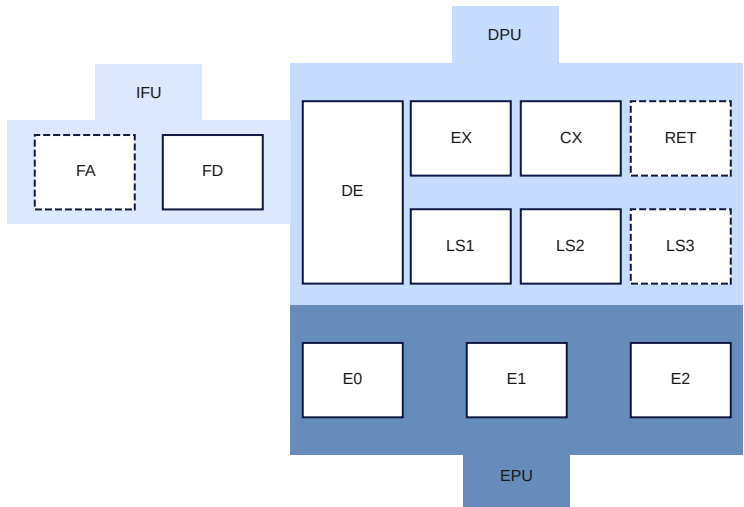
2.2 Pipeline overview

The Cortex®-M52 processor pipeline is 4-stages deep for integer instructions and 4-stages deep for *Floating Point* (FP) and *M-Profile Vector Extension* (MVE) instructions.

The following diagram describes the high-level Cortex®-M52 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®-M52 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 [Cortex-M52 processor Core and EPU pipeline structure](#).

Table 2-2: Cortex®-M52 processor Core and EPU pipeline structure

Stage	Description
FA	<p>IFU Fetch Address stage</p> <p>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.</p> <p>The FA stage detects loop end for Armv8.1-M low-overhead loop operation.</p> <p>The IFU always fetches 32-bit of data from memory which could consist of up to 1 32-bit Thumb instruction or 2 16-bit Thumb instructions.</p>
FD	<p>IFU Fetch Data stage</p> <p>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.</p> <p>The instruction queue allows the processor to decouple the operation of the Core pipeline from instruction fetches, allowing execution to continue when the fetch stage is stalled, for example due to a cache miss.</p> <p>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 Instruction latencies.</p>

DE	<p>DPU Decode stage</p> <p>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.</p> <p>The decoder can handle all scalar integer single and dual issue cases. Floating point and MVE operations are dispatched to the EPU EO stage for further processing.</p> <p>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.</p> <p>Load and store address operands are constructed from both scalar register reads and from the extended register port read in the EO 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.</p> <p>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.</p> <p>The DE stage also carries out the PC change for conditional and unconditional indirect and function return (BX LR) branches. Forcing the PC change early minimizes the branch latency in common cases improving the performance of the processor.</p>
EX	<p>DPU Simple EXecute stage</p> <p>The EX stage handles most scalar arithmetic, logical, and bit-shift operations. EX also contains the first stage of the integer divider. And the EX stage also handles all the SIMD and saturating instructions.</p> <p>This stage reads data from the register bank or CX stage for memory store operations and accumulate data for the scalar multiplier.</p> <p>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.</p> <p>All operations which can complete their computation in EX terminate in the stage can forward to the following instructions which have dependence, or propagate to CX stage to write back the result to register bank.</p>
CX	<p>DPU Complex eXecute stage</p> <p>This stage includes a second ALU which is used to handle a few complex instructions from the regular instruction set. The stage also includes the integer multiplier and second stage of the integer divider.</p> <p>Results from the CX stage are written back to the register bank using two dedicated 32-bit write ports. 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.</p> <p>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 CX stage and then send to LSU in CX.</p> <p>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.</p> <p>Note: To improve performance for load multiple operations which branch, the PC is loaded from memory first so instructions at the target address can be fetched while the remaining registers are loaded.</p>

RET	<p>DPU Retire stage</p> <p>ECC errors received from the LSU in CX are combined with uncorrectable errors received in the RET stage. If the error is correctable the instruction is re-fetched and re-executed by forcing a branch in the IFU.</p>
LS1	<p>Load-store address stage</p> <p>The LSU is responsible for distributing memory requests from the DPU to the appropriate structures and interfaces in the memory system including the Data cache (or unified cache) and M-AXI (or M-AHB) 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 and Data cache (or unified cache) and TCM RAM is enabled to minimize the latency to these interfaces critical for processor performance. Access to other interfaces are selected later in the pipeline when the instruction has been committed and cannot be interrupted. These accesses are less critical for performance and typically use Device memory and cannot be speculative.</p> <p>Unaligned load/store requests are split out in DPU, and DPU send aligned requests to LSU, but the read data is combined and aligned before being returned to the DPU in the correct format for writing to the register file. Store data is taken from the DPU and broken down as required to write out.</p>
LS2	<p>Load-store read data phase</p> <p>The LS2 stage corresponds to the data phase of the cache and TCM RAM for read accesses. The Data cache (or unified cache) Tag comparison is carried out and hit information is used to determine whether or not an M-AXI access is required. The LSU will stall the DPU in this stage until read data is available from the M-AXI or TCM.</p> <p>The address for all store requests and load requests to P-AHB, internal PPB peripherals, and EPPB are sent out in LS2. Store data is relayed from the DPU in the LS2 stag.</p> <p>Read data is collected from the appropriate interface units and processed according the instruction and processor state. This can involve sign/zero extension and byte 'swizzling' for Endianness before being sent back to the DPU.</p> <p>If ECC error detection is included in the processor the data returned from the RAM is checked against the ECC code in LS2. Any errors detected signaled to the DPU in the CX stage, however for timing reasons the determination of uncorrectable errors cannot be completed until the next cycle. This information is returned in the RET stage of the DPU pipeline.</p>
LS3	<p>Uncorrectable ECC Error Report</p> <p>The LS3 stage is only used to report uncorrectable ECC error related information to DPU.</p>
E0	<p>EPU Decode and address transfer and EPU Operand register read stage</p> <p>The E0 stage contains the MVE and floating-point decoder based on instructions dispatched from the DE stage of the pipeline. The stage is also used to read the base address for MVE load and store operations which use the Extended register file. The register data is sent back to the DPU in the same cycle and used to compute the final address in the EX stage.</p> <p>The stage also contains the control and hazard logic used to handle instruction overlap for beat-wise MVE execution based on resource availability in the pipeline and a state-machine to generate micro-operations for instructions which require multiple issue cycles to execute – particularly double precision floating point arithmetic where results are built by recirculating through the EPU data path.</p> <p>Note: The EPU pipeline always operates in lock-step with the main DPU pipeline.</p> <p>The Extended register file is read for all arithmetic operands in the E0 stage. Results from E1 and E2 can be forwarded to following instructions of the same class to avoid or reduce RAW hazards. Forwarding is not supported between classes. For example, floating point to fixed point instructions or vector to scalar floating-point instructions to reduce control complexity as registers are unlikely to be used for different data-types simultaneously.</p> <p>Scalar operands are received from the DPU and combined in the operand data path when required. Store data is also read in the E0 stage, and registered into E1 where it is passed over to the DPU to be written to memory.</p>

E1	<p>EPU arithmetic and logic stage</p> <p>The E1 stage contains the structures used to carry out vector operations all data types and scalar operations on all floating point data type, including a combined multiply accumulate unit and a dedicated divide and square root unit. The majority of arithmetic and bitwise logic operations (scalar Instruction or 1 beats of an MVE instruction) complete in a single cycle apart from:</p> <ul style="list-style-type: none"> • 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} <p>Chained multiply-accumulate is carried out as a multiply operation followed by an add operation in serial in E1 with full rounding after each operation in E2.</p> <p>Double precision operations, where required partial results are calculated in E1 and recirculated into the E0 until the full E1 double precision result is available.</p> <p>The E1 stage is also used to transfer extended registers from the EPU to the CX stage of the DPU.</p>
E2	<p>EPU write-back stage</p> <p>The results from floating point operations are normalized and rounded in the E2 stage.</p> <p>The vector result for all EPU operations are written back to the extended register file in E2 including load data transferred in from the memory system in E2 via the CX stage. All vector writes in E2 are forwarded back to E0.</p> <p>MVE operations with scalar result also write-back to the register file in the DPU in E2.</p> <p>The VPR and flags in FPSCR.NZCV are updated in E2 stage.</p> <p>MVE and floating-point instruction which leave the E2 stage of the EPU pipeline are committed and can no longer be interrupted.</p> <p>Load data is returned from the DPU CX stage in E2. The load data-path is separate into 4 byte-lanes supporting vector predication on write-back.</p>

3. Instruction latencies

This chapter describes the high-level performance characteristics for most ARMv8.1-M 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®-M52 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®-M52 processor microarchitecture.
- Cortex®-M52 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®-M52 processor is a 1 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 with additional 1-cycle a 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)*
 - *Vector and Floating Point Pipe (VF)*

Cortex®-M52 processor supports overlapping MVE vector instructions which use different execution pipelines.

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

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	B (T1)	1(2)	1	11	3
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)	3	1	01	-
	BLXNS (T1)				
Branch, register (with destination LR)	BX (T1)	3(2)	1	11	2

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

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

Instruction group	32-bit Thumb instructions	Execution latency	Execution throughput	Notes
Basic ALU	AND (immediate) (T1)	1	1	-
	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)			
Basic ALU	ORN (register) (T1)	1(2)	1(1/2)	2
	ORR (register) (T2)			
Basic ALU	PKHBT, PKHTB (T1)	1	1	-
	SEL (T1)			
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)	1	1	-
	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)			
	USAX (T1)			
	USUB16 (T1)			
	USUB8 (T1)			
	USAD8 (T1)	2	1	-
	USADA8 (T1)			
Sign Extend Addition	SXTB (T2)	1	1	-
	SXTH (T2)			
Sign Extend Addition	SXTAB (T1)	1	1	-
	SXTAB16 (T1)			
	SXTAH (T1)			
	SXTB16 (T1)			
Signed Addition	SSAT (T1)	1	1	-

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

Notes:

- 1 If the shift type is not LSL, or if the shift type is LSL but the shift amount is greater than 4, then the latency is 2 and the throughput is 1. In addition, if the result is 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 type is not LSL, or if the shift type is LSL but the shift amount is greater than 4, 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	-

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

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)	1	1	-
	MOV (immediate) (T3)			
	MOV (register) (T3)			
	MOV (register) (T3)			
	MOV, MOVS (register-shifted register) (T2)			
	MOVT (T1)			

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)	1	1	01	-
	MOV, MOVS (register-shifted register) (T1)				
	MVN (register) (T1)				
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)	2-20	1/19-1	1
	UDIV (T1)			
Multiply	MUL (T2)	2	1	-

Instruction group	32-bit Thumb instructions	Execution latency	Execution throughput	Notes
Multiply Accumulate	MLA (T1)	2	1	-
	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)			
	SMLS LD, SMLS LDX (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)			

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/2)) and throughput of (1/(3+ Round_up (DIFF_SIGN/2))).

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)	2	1	-
	LDR (immediate) (T3)			
	LDR (immediate) (T4)			
	LDR (literal) (T2)			
	LDR (register) (T2)			
Exclusive operations	LDAEX (T1)	2	1	-
	LDAEXB (T1)			
	LDAEXH (T1)			
	LDREX (T1)			
	LDREXB (T1)			
	LDREXD (T1)			
	LDREXH (T1)			
Load Multiples	LDRD (immediate) (T1)	3	1/2	1
	LDRD (literal) (T1)			

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®-M52 processor supports 32-bit accesses per cycle. N=num_regs.

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)	2	1	01	-
	LDR (immediate) (T2)				
	LDR (literal) (T1)				
	LDR (register) (T1)				
Load Multiples	LDM, LDMIA, LDMFD (T1)	N+1	1/N	00	1
	POP (multiple registers) (T3)				
Sub Word Loads	LDRB (immediate) (T1)	2	1	01	-
	LDRB (register) (T1)				
	LDRH (immediate) (T1)				
	LDRH (register) (T1)				
	LDRSB (register) (T1)				
	LDRSH (register) (T1)				

Notes:

1 Cortex®-M52 processor supports 32-bit accesses per cycle. N=num_regs.

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)	2	1	-
	STR (immediate) (T4)			
	STR (register) (T2)			
Exclusive operations	STREX (T1)	2	1	-
	STREXB (T1)			
	STREXH (T1)			

Instruction group	32-bit Thumb instructions	Execution latency	Execution throughput	Notes
Store Lock Release	STL (T1)	2	1	-
	STLB (T1)			
	STLEX (T1)			
	STLEXB (T1)			
	STLEXH (T1)			
	STLH (T1)			
Store Multiple	STRD (immediate) (T1)	3	1/2	1
Store Multiple	STM, STMIA, STMEA (T2)	N+1	1/N	1
	STMDB, STMFD (T1)			
Sub Word Stores	STRB (immediate) (T2)	2	1	-
	STRB (immediate) (T3)			
	STRB (register) (T2)			
	STRH (immediate) (T2)			
	STRH (immediate) (T3)			
	STRH (register) (T2)			

Notes:

1 Cortex®-M52 supports 32-bit accesses per cycle. N=num_regs.

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

Notes:

1 Cortex®-M52 processor supports 32-bit accesses per cycle. N=num_regs.

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)	1	1	1
	PLI (immediate, literal) (T2)			
	PLI (immediate, literal) (T3)			
	PLI (register) (T1)			
Hints	PLD (literal) (T1)	1	1	-
	PLD, PLDW (immediate) (T1)			
	PLD, PLDW (immediate) (T2)			
	PLD, PLDW (register) (T1)			
No Operation	NOP (T2)	1	1	-
Register updates	CLRM (T1)	N+1	1/N	2
PACBTI	AUT	2	1	-
	AUTG	2	1	-
	BXAUT	4	1/4	-
	PAC	2	1	-
	PACBTI	2	1	-
	PACG	2	1	-
	BTI	1	1	-

Notes:

- 1 Acts as a NOP.
- 2 CLRM supports clearing 1 registers per cycle. N=num_regs.

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
Scalar FP Load	VLDR (T2)	2	1	-
	VLDR (T3)			
Scalar FP Load	VLDR (T1)	3	1/2	2
Scalar FP Load	VLDM (T1/T2)	N+1	1/N	2
	VLLDM (T1/T2)			
	VSCCLRM (T1/T2)			
Divide (Double-precision)	VDIV (T1)	32	1/31	3
Divide (Half-precision)	VDIV (T1)	11	1/10	3
Divide (Single-precision)	VDIV (T1)	17	1/16	3
Divide (all-precision) with Input Zero/Infinite/NaN or Invalid Operation	VDIV (T1)	5	1/4	3
Scalar Absolute	VABS (T2)	2	1	-
Scalar Arith	VADD (T1)	2(15)	1(1/14)	1
	VSUB (T1)			
Scalar Arith	VMAXNM (T1)	2	1	-
Scalar Compare	VCMP (T1)	2	1	-
	VCMP (T2)			
Scalar Convert	VCVT (between double-precision and single-precision) (T1)	2	1	-
	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)			

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) VMOVB (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)	32	1/31	3
Square Root (Half-precision)	VSQRT (T1)	11	1/10	3
Square Root (Single-precision)	VSQRT (T1)	17	1/16	3
Square Root (all-precision) with Input Zero/Infinite/NaN or Invalid Operation	VSQRT (T1)	5	1/4	3
Scalar FP Store	VSTR (T2) VSTR (T3)	1	1	-
Scalar FP Store	VSTR (T1)	2	1/2	2
Scalar FP Store	VLSTM (T1/T2) VSTM (T1/T2)	N+1	1/N	2

Notes:

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

Notes:

- 2** Cortex®-M52 processor supports one 32-bit accesses per cycle. For single-precision store multiple instructions, $N = \text{num_regs}$. For double-precision store multiple instructions, $N = (\text{num_regs}) \times 2$.
- 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)	9	1/8	VF	1
MVE Absolute	VABD (T1)	2	1/4	VF	-
	VABS (T1)				
	VQABS (T1)				
MVE Arith	VMAXV, VMINV(T1, esize==32b)	5	1/4	VF	-
	VMAXV, VMINV(T1, esize==16b)	9	1/8		
	VMAXV, VMINV(T1, esize==8b)	13	1/12		
	VMAXAV, VMINAV(T2, esize==32b)	9	1/8		
	VMAXAV, VMINAV(T2, esize==16b)	13	1/12		
	VMAXAV, VMINAV(T2, esize==8b)	17	1/16		

Instruction group	Instructions	Execution latency	Execution throughput	Utilized pipeline	Notes
MVE Arith	VADC (T1)	2	1/4	VF	-
	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)				

Instruction group	Instructions	Execution latency	Execution throughput	Utilized pipeline	Notes
MVE Bitwise	VAND (T1)	2	1/4	VF	-
	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)				
MVE CLS/CLZ	VCLS (T1)	2	1/4	VF	-
	VCLZ (T1)				
MVE Compare	VCMP (T1)	2	1/4	VF	-
	VCMP (T2)				
	VCMP (T3)				
	VCMP (T4)				
	VCMP (T5)				
	VCMP (T6)				
	VPT (T1)				
	VPT (T2)				
	VPT (T3)				
	VPT (T4)				
	VPT (T5)				
	VPT (T6)				

Instruction group	Instructions	Execution latency	Execution throughput	Utilized pipeline	Notes
MVE Duplicate	VDDUP, VDWDUP (T1)	2	1/4	VF	-
	VDDUP, VDWDUP (T2)				
	VIDUP, VIWDUP (T1)				
	VIDUP, VIWDUP (T2)				
MVE Duplicate	VDUP (T1)	2	1/4	VF	-
MVE MOV	VMOVL (T1)	2	1/4	VF	-
	VMOVN (T1)				

Instruction group	Instructions	Execution latency	Execution throughput	Utilized pipeline	Notes
MVE Multiply	VMLA (vector by scalar plus vector) (T1)	2	1/4	VF	-
	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)				
MVE Negate	VNEG (T1)	2	1/4	VF	-
	VQNEG (T1)				
MVE Select	VPSEL (T1)	2	1/4	VF	-

Instruction group	Instructions	Execution latency	Execution throughput	Utilized pipeline	Notes
MVE Shift	VBR SR (T1)	2	1/4	VF	-
	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 arith to scalar	VADDLV (T1)	2	1/4	VF	-
	VADDV (T1)				
	VMLADAV (T1)				
	VMLADAV (T2)				
	VMLALDAV (T1)				
	VMLSDAV (T1)				
	VMLSDAV (T2)				
	VMLSLDAV (T1)				
	VRMLALDAVH (T1)				
	VRMLSLDAVH (T1)				

Notes:

- The instruction is executed beat-by-beat as the multicycle MVE instruction.

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)	2	1	VF	-
	VMOV (two general-purpose registers to two 32 bit vector lanes) (T1)				
Scalar MOV	VMOV (two 32 bit vector lanes to two general-purpose registers) (T1)	2	1	VF	-
	VMOV (vector lane to general-purpose register) (T1)				

Instruction group	Instructions	Execution latency	Execution throughput	Utilized pipeline	Notes
Scalar Shift	ASRL (immediate) (T1)	5	1/4	DPU	-
	ASRL (register) (T1)				
	LSLL (immediate) (T1)				
	LSLL (register) (T1)				
	LSRL (immediate) (T1)				
	SQRSHR (register) (T1)				
	SQRSHRL (register) (T1)				
	SQSHL (immediate) (T1)				
	SQSHLL (immediate) (T1)				
	SRRSHR (immediate) (T1)				
	SRRSHRL (immediate) (T1)				
	UQRSHL (register) (T1)				
	UQRSHLL (register) (T1)				
	UQSHL (immediate) (T1)				
	UQSHLL (immediate) (T1)				
	URSHR (immediate) (T1)				
	URSHRL (immediate) (T1)				

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)	2	1/4	VF	-
	VABS (floating-point) (T1)				

Instruction group	Instructions	Execution latency	Execution throughput	Utilized pipeline	Notes
MVE Arith	VMAXNMV, VMINNMV (floating-point) (T1, SP)	5	1/4	VF	1
	VMAXNMAV, VMINNMAV (floating-point) (T2, SP)	5	1/4		
	VMAXNMV, VMINNMV (floating-point) (T1, HP)	9	1/8		
	VMAXNMAV, VMINNMAV (floating-point) (T2, HP)	9	1/8		
MVE Arith	VADD (floating-point) (T1)	2	1/4	VF	-
	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)				
MVE Compare	VPT (floating-point) (T1)	2	1/4	VF	-
	VPT (floating-point) (T2)				
MVE Compare	VCMP (floating-point) (T1)	2	1/4	VF	-
	VCMP (floating-point) (T2)				
MVE Convert	VCVT (between floating-point and fixed-point) (T1)	2	1/4	VF	-
	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)				

Instruction group	Instructions	Execution latency	Execution throughput	Utilized pipeline	Notes
MVE Multiply	VCMLA (floating-point) (T1)	2	1/4	VF	-
	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)				
MVE Negate	VNEG (floating-point) (T1)	2	1/4	VF	-

Notes:

- 1 The instruction is executed beat-by-beat as the multicycle MVE instruction.

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
System	VLDR (System Register) (T1)	1	1	SY
	VMRS (T1)			
	VMSR (T1)			
	VSTR (System Register) (T1)			
System	VCTP (T1)	2	1/4	SY
	VPNOT (T1)			
	VPST (T1)			

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 (T1)	2	1/4	LS	1
	VLDRH (T2)				
	VLDRB (T5)				
	VLDRH (T6)				
	VLDRW (T7)				
Deinterleaving Vector Load	VLD2 (T1)	2	1/4	LS	-
	VLD4 (T1)				
Gather Vector Load	VLDRB (T1, 8b)	17	1/16	LS & VF	1
	VLDRB (T1, 16b)	9	1/8		
	VLDRB (T1, 32b)	2	1/4		
	VLDRH (T2, 16b)	9	1/8		
	VLDRH (T2, 32b)	2	1/4		
	VLDRW (T3)	2	1/4		
	VLDRD (T4)	2	1/4		
	VLDRW (T5, non WB)	2	1/4		
	VLDRW (T5, WB)	9	1/8		
	VLDRD (T6, non WB)	2	1/4		
	VLDRD (T6, WB)	9	1/8		

Notes:

- The instruction is executed beat-by-beat as the multicycle MVE instruction.
The T5/T6 WB operation executed in the VF 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
Continuous Vector Load Store	VSTRB, VSTRH, VSTRW (T1)	2	1/4	LS	1
	VSTRB, VSTRH, VSTRW (T2)				
	VSTRB, VSTRH, VSTRW (T5)				
	VSTRB, VSTRH, VSTRW (T6)				
	VSTRB, VSTRH, VSTRW (T7)				
Interleaving Vector Store	VST2 (T1)	2	1/4	LS	-
	VST4 (T1)				
Scatter Vector Store	VSTRB (T1, 8b)	17	1/16	LS & VF	1
	VSTRB (T1, 16b)	9	1/8		
	VSTRB (T1, 32b)	2	1/4		
	VSTRH (T2, 16b)	9	1/8		
	VSTRH (T2, 32b)	2	1/4		
	VSTRW (T3)	2	1/4		
	VSTRD (T4)	2	1/4		
	VSTRW (T5, non WB)	2	1/4		
	VSTRW (T5, WB)	9	1/8		
	VSTRD (T6, non WB)	2	1/4		
	VSTRD (T6, WB)	9	1/8		

Notes:

- The instruction is executed beat-by-beat as the multicycle MVE instruction.
The T5/T6 WB operation executed in the VF pipe.

4. Additional information

This chapter describes some general behaviors related to the micro-architecture for the Cortex®-M52 processor.

4.1 MVE pipeline hazard

MVE vector instructions are issued as 4 micro-ops. Each micro-op operates on 32 bits of data and is also known as a tick. Overlapping means tick2 or tick3 of an MVE instruction can execute in parallel to a tick0 or tick1 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®-M52 processor, a newer MVE instruction can overlap with the older MVE instruction if the older tick2/3 does not use the same pipe as the newer tick0/1. Utilized pipeline can be referred to in the instruction latency tables.

There are sets of scalar instructions which are allowed to overlap with tick3 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

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

- 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.
- Any scalar load or store instructions cannot overlap with vector instructions.
- If there is dependency between the scalar and vector instruction, then it is unlikely to overlap.

4.2 Hardware prefetcher

The Cortex®-M52 processor supports a hardware data prefetcher which monitors the address of line-fills for patterns which indicate a stream of data is being accessed by the software.

The prefetcher uses the pattern information to predict where future line-fills may happen and attempts to fetch the data from the system into the Data cache before they are needed. This feature can significantly improve the overall performance by hiding load latency from the instructions executing on the processor.

A configurable parameter `PREFETCH` is provided to determine whether the hardware prefetcher is included or not, which is only applicable when D-CACHE is present and the main interface is configured as M-AXI. This feature is only applicable if the main interface is configured as AXI. If the hardware prefetcher is included, software can control its behavior through *Prefetcher Control Register* (PFCR).

For details of the `PREFETCH` parameter, refer to *Processor-level configuration options summary* in *Arm China Cortex®-M52 Processor Integration and Implementation Manual*.

For details of the PFCR register, refer to *PFCR, Prefetcher Control Register* in *Arm China Cortex®-M52 Processor Technical Reference Manual*.

Appendix A Revisions

Changes between released issues of this manual are summarized in tables.

Table A-1: Issue 0001-01

Change	Location
First release.	-

Table A-2: Differences between issue 0001-01 and 0002-02

Change	Location
Second release.	-
Change the product name from Mizar to Cortex®-M52.	-

Table A-3: Differences between issue 0002-02 and 0003-03

Change	Location
Release for r0p3.	-
In the <i>Additional information</i> chapter, added a new topic <i>Hardware prefetcher</i> .	4.2 Hardware prefetcher on page 45