



# A64FX®

## Microarchitecture Manual

日本語

---

Copyright© 2019 Fujitsu Limited, 4-1-1 Kamikodanaka, Nakahara-ku, Kawasaki, 211-8588, Japan. All rights reserved.

This product and related documentation are protected by copyright and distributed under licenses restricting their use, copying, distribution, and decompilation. No part of this product or related documentation may be reproduced in any form by any means without prior written authorization of Fujitsu Limited and its licensors, if any.

The product(s) described in this book may be protected by one or more U.S. patents, foreign patents, or pending applications.

#### TRADEMARKS

Fujitsu and the Fujitsu logo are trademarks of Fujitsu Limited.

This publication is provided "as is" without warranty of any kind, either express or implied, including, but not limited to, the implied warranties of merchantability, fitness for a particular purpose, or noninfringement.

This publication could include technical inaccuracies or typographical errors. Changes are periodically added to the information herein; these changes will be incorporated in new editions of the publication. Fujitsu Limited may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time.

---

# 改版履歴

---

変更日	版数	変更内容
2020/2/28	1.0	First Release.
2020/4/28	1.1	Correct typos.
2020/7/31	1.2	Update following chapters and sections: <ul style="list-style-type: none"><li>• 6.5.1. Merging predication</li><li>• 7.8.1. Multiple Structures 命令</li><li>• 7.8.2. Gather load / Scatter store 命令</li><li>• 9.6. Zero fill</li><li>• 16. 命令属性／レイテンシ一覧: Extra μOP およびレイテンシの表記方法について</li></ul>

# 目次

---

<b>1.はじめに</b>	<b>12</b>
1.1. A64FX プロセッサの概要	12
1.2. A64FX プロセッサの諸元	13
1.3. A64FX プロセッサのブロック	14
<b>2.アウト・オブ・オーダ・アーキテクチャ</b>	<b>15</b>
2.1. アーキテクチャ概要	15
2.2. マイクロ・オペレーション命令	16
2.3. オペレーション・フロー	17
2.4. アウト・オブ・オーダ・リソース	17
2.5. パイプライン・ステージ	19
2.6. 命令実行レイテンシ	22
2.7. オペランド・バイパス	22
2.8. リソースの割り当てと解放	25
2.9. レイテンシ切り替え	25
<b>3.命令フェッチ</b>	<b>27</b>
3.1. 命令フェッチ・ステージの概略	27
3.2. 分岐予測機構	28
3.2.1. Small Taken Chain Predictor (S-TCP)	28
3.2.2. Loop Prediction Table (LPT)	29
3.2.3. Branch Weight Table (BWT)	29
3.2.4. Branch Target Buffer (BTB)	30
3.2.5. Return Address Stack (RAS)	31
3.3. 分岐予測の組み合わせ	31
3.4. ショート・ループ検出	31
<b>4.命令デコードと命令コミット</b>	<b>32</b>
4.1. マイクロ・オペレーション命令	32
4.2. マルチ・オペレーション	32
4.3. MOVPRFX 命令の Pack 处理	33
4.4. 命令デコード	34
4.4.1. プリデコード	35
4.4.2. デコード	37
4.5. 命令コミット	38
4.5.1. No Exception Mode	39
4.6. パイプライン・フラッシュ	39
4.7. 特殊な命令制御	39
<b>5.命令ディスパッチ</b>	<b>41</b>
5.1. リザベーション・ステーション	41
5.2. 命令のディスパッチ属性	41
5.3. 命令の依存関係の検出	42

---

5.4. 命令ディスパッチ動作	43
<b>6. 命令実行</b>	<b>46</b>
6.1. 命令発行	46
6.2. 実行パイプライン	46
6.3. ブロッキング制御	48
6.4. 物理レジスタ・ファイル	48
6.5. 特殊な命令制御	49
6.5.1. Merging predication	49
6.5.2. レジスタ間転送	49
6.5.3. 非正規化数の演算	50
<b>7. メモリ・アクセス</b>	<b>51</b>
7.1. ロード／ストア・パイプラインの概要	51
7.2. ロード／ストアの基本処理	53
7.2.1. ロード命令	54
7.2.2. ストア命令	54
7.3. Fetch Port / Store Port	55
7.3.1. Virtual Fetch Port / Virtual Store Port	55
7.3.2. Fetch Port / Store Port の割り当て数	56
7.4. Write Buffer	56
7.5. ロード／ストアのアウト・オブ・オーダ実行	58
7.5.1. Store Fetch Bypass	58
7.5.2. アウト・オブ・オーダ実行の制約	60
7.6. パイプライン競合	61
7.7. キャッシュ・ライン・クロス	63
7.8. 特殊なロード／ストア命令の動作	63
7.8.1. Multiple Structures 命令	63
7.8.2. Gather load / Scatter store 命令	65
<b>8. アドレス変換機構</b>	<b>70</b>
8.1. Translation Lookaside Buffer (TLB)	70
8.2. Translation table cache	70
<b>9. キャッシュ・アーキテクチャ</b>	<b>71</b>
9.1. キャッシュ、メモリ階層の構造	71
9.2. キャッシュの基本構成	72
9.2.1. L1 キャッシュの構成	72
9.2.2. L2 キャッシュの構成	73
9.3. キャッシュ・コヒーレンス・プロトコル	74
9.4. Move-In / Move-Out	74
9.5. Move-In Bypass	75
9.6. Zero fill	76
<b>10. Memory Access Controller</b>	<b>78</b>
10.1. Memory Access Controller の構成	78
10.2. メモリ・アクセス性能	78
<b>11. データ・プリフェッч</b>	<b>79</b>
11.1. プリフェッチの動作概要	79
11.2. プリフェッチ・アクセスのタイプ	81

11.3.	プリフェッチ・アクセスの信頼度.....	81
11.4.	ソフトウェア・プリフェッチ.....	82
11.4.1.	プリフェッチ命令の分類.....	82
11.4.2.	ソフトウェア・プリフェッチの属性.....	83
11.5.	ハードウェア・プリフェッチ.....	83
11.5.1.	ハードウェア・プリフェッチのための資源.....	83
11.5.2.	Stream detect mode の動作.....	84
11.5.3.	Prefetch injection mode の動作.....	85
11.5.4.	ハードウェア・プリフェッチ・アシスト機能.....	86
11.5.5.	ハードウェア・プリフェッチのキャッシュ階層属性.....	87
11.6.	Prefetch injection mode の使用例.....	87
<b>12.</b>	<b>セクタ・キャッシング .....</b>	<b>89</b>
12.1.	セクタ・キャッシングの概要.....	89
12.2.	セクタ・キャッシングの動作.....	89
<b>13.</b>	<b>ハードウェア・バリア .....</b>	<b>91</b>
<b>14.</b>	<b>Performance Monitor Events.....</b>	<b>92</b>
14.1.	Instruction Mix.....	92
14.2.	FLOPS .....	94
14.3.	Hardware Resource Monitor Events .....	95
14.4.	Cycle Accounting .....	96
<b>15.</b>	<b>リソース一覧 .....</b>	<b>100</b>
<b>16.</b>	<b>命令属性／レイテンシー一覧 .....</b>	<b>102</b>
16.1.	ARMv8 Base instructions .....	103
16.2.	ARMv8 SIMD&FP instructions .....	114
16.3.	SVE instructions .....	126

# 図一覧

---

Figure 1-1	Main Functional Blocks on A64FX Processor Chip.....	14
Figure 2-1	Overall Illustration of Stages.....	15
Figure 2-2	Integer Operation Pipeline Stages .....	20
Figure 2-3	SIMD&FP and SVE Operation Pipeline Stages .....	20
Figure 2-4	Predicate Operation Pipeline Stages .....	20
Figure 2-5	Branch Pipeline Stages.....	20
Figure 2-6	Load/Store Pipeline Stages.....	21
Figure 2-7	Example of Conflict Between C Stages of Instructions with Different Latencies .....	25
Figure 2-8	Example of Latency Changing .....	26
Figure 3-1	Instruction Fetch Stage.....	27
Figure 3-2	Bubbles Due to Instructions Following Taken Branch Instruction .....	28
Figure 3-3	Chain Structure Consisting of Multiple Taken Branch Instructions .....	29
Figure 3-4	Histories of Conditional Branches and Weights .....	30
Figure 3-5	Prediction Equation for Conditional Branch Instruction Bo .....	30
Figure 3-6	Outline of Branch Target Buffer (BTB).....	30
Figure 4-1	Example of Efficient Packing with MOVPRFX .....	33
Figure 4-2	Example of Inefficient Packing Due to Instruction Order .....	34
Figure 4-3	Instruction Decode Stage.....	34
Figure 4-4	Restriction on Taken Branch Instruction When Splitting μOP Instructions .....	36
Figure 4-5	Restriction Related to Three or More μOP Splits Resulting from μOP Instruction Splitting .....	36
Figure 4-6	Restriction on μOP Instruction Splitting for Sequential Decode .....	37
Figure 4-7	CSE Structure .....	38
Figure 5-1	Example of Two Instructions That Have Dependency in Same Decode Window .....	43
Figure 5-2	Example of Two Instructions That Have Dependency Across Different Decode Windows .....	43
Figure 5-3	Allocation Table Selection Rule for Instructions with RSX Attribute .....	44
Figure 5-4	Allocation Table Selection Rule for Instructions with RSE or RSA Attribute .....	45
Figure 6-1	Outline of Execution Unit .....	47
Figure 6-2	Connection Relationship Between Physical Register Files and Execution Pipelines .....	48
Figure 6-3	Flow Time Chart of Transfer Instruction from General-Purpose Register to Floating-Point Register.....	49
Figure 6-4	Flow Time Chart of Transfer Instruction from Floating-Point Register to General-Purpose Register.....	50
Figure 7-1	Outline of Load/Store Unit .....	51
Figure 7-2	Relationship Between VFP/VSP and RFP/RSP .....	56
Figure 7-3	Store Data Write from SP to WB .....	57
Figure 7-4	Example of active/inactive in ST1B (Contiguous) .....	62
Figure 7-5	Illustration of Splitting LD3D (multiple structures) Instruction Flow .....	65
Figure 7-6	Requests of Gather Instruction .....	66
Figure 7-7	Requests of Scatter Instruction .....	67
Figure 7-8	Effective Address Generation for Gather Instruction.....	67
Figure 7-9	Summary of Elements for Gather Instruction .....	69
Figure 9-1	L2 Caches and Memory Levels .....	71
Figure 9-2	Connection Between L1 and L2 Caches .....	72
Figure 9-3	Basic Zero Fill Process .....	76
Figure 9-4	Zero Fill Process When L1D Cache Contains Data .....	77
Figure 11-1	Operation-Flows for Demand Access and Prefetch Access .....	80
Figure 11-2	Hardware Prefetch Behavior in Stream Detect Mode .....	84
Figure 11-3	Usage Example of Prefetch Injection Mode .....	88
Figure 12-1	L1D/L2 Sector Cache .....	89
Figure 12-2	Example of Sector Cache Capacity Adjustment (1).....	90
Figure 12-3	Example of Sector Cache Capacity Adjustment (2).....	90
Figure 13-1	Hardware Barrier Resources .....	91
Figure 13-2	Sample Code for Synchronization Processing.....	91

---

# 表一覽

---

Table 1-1	A64FX Processor Specifications.....	13
Table 1-2	Correspondence Between Processor Chip Block Markings and Functional Units .....	14
Table 2-1	Out-of-Order Resources.....	17
Table 2-2	Correspondence Between Pipeline Stage Symbols and Operations .....	19
Table 2-3	Execution Start and Completion Stages for Each Instruction in Each Pipeline.....	22
Table 2-4	Penalties for Operand Bypass Between μOP Instructions .....	23
Table 2-5	Penalties for Operand Bypass Between μOP Instructions (FTMAD Instruction) .....	24
Table 2-6	Out-of-Order Resource Allocation and Release Stages.....	25
Table 2-7	Instructions Whose Latency Changed, and Their Latencies.....	26
Table 3-1	Branch Predictors of Branch Prediction Mechanism .....	28
Table 3-2	Relationship Between Predictors Used for Branch Prediction and Prediction Result Adoption Rankings....	31
Table 4-1	Relation Between Instructions and Quantities of Allocated Resources .....	37
Table 4-2	FPCR Register When No Exception Mode Is Enabled .....	39
Table 5-1	Number of Entries and Connected Execution Pipelines of Each RS.....	41
Table 5-2	Attributes of Instructions and Operation-Flows .....	42
Table 5-3	Instructions That Require TOR .....	42
Table 5-4	Allocation Table for Instructions with RSX Attribute .....	44
Table 5-5	Allocation Table for Instructions with Either RSE or RSA Attribute .....	44
Table 6-1	Execution Pipelines.....	47
Table 7-1	Latencies of Load/Store Instructions.....	53
Table 7-2	Data Length and Merge Function Availability for Each Instruction Managed by WB Entry .....	58
Table 7-3	SFB Availability for Each Combination of Load and Store Instructions.....	59
Table 7-4	Specific Instructions of Each Group Shown in SFB Availability Table .....	60
Table 7-5	ST0 Flow Conditions .....	62
Table 7-6	Required Number of Flows for μOP Instructions Split from Architecture Instruction to Send to Load/Store .....	64
Table 7-7	Number of μOP Instructions and Number of Allocated FP/SP Entries for Each Gather/Scatter Instruction .....	66
Table 8-1	TLB Specifications .....	70
Table 8-2	Table Cache Specifications .....	70
Table 9-1	Bus Throughput .....	72
Table 9-2	L1 Cache Specifications.....	73
Table 9-3	L2 Cache Specifications.....	74
Table 9-4	Details of MESI Protocol .....	74
Table 9-5	Quantity of Queue Resources at Each Cache Level .....	75
Table 9-6	Instructions That Can Execute Move-In Bypass on L1D Cache .....	76
Table 10-1	Specifications of HBM2 Supported by A64FX .....	78
Table 10-2	Quantity of Scheduler Resources for HBM2.....	78
Table 10-3	A64FX Memory Access Performance .....	78
Table 11-1	Classifications and Mnemonics of Prefetch Instructions.....	82
Table 11-2	Correspondence Between Prefetch Instruction Options, Cache Levels, and States .....	83
Table 11-3	Correspondence Between pf_func[0] Bit and Software Prefetch Reliability .....	83
Table 11-4	Control Register Configuration Example .....	88
Table 14-1	Performance Events for Instruction Mix .....	92
Table 14-2	Formulas for Other (Instruction Mix) .....	94
Table 14-3	Performance Events for FLOPS .....	94
Table 14-4	Performance Events for Hardware Resource Monitoring .....	95
Table 14-5	Method to Calculate Hardware Performance Indicators at Program Execution .....	96
Table 14-6	Performance Events for Cycle Accounting .....	97
Table 14-7	Formulas for Other (Cycle Accounting).....	99
Table 15-1	Out-of-Order Resources .....	100
Table 15-2	Resources for Branch Misprediction Mechanism .....	101
Table 15-3	Resources for Memory Management Unit .....	101
Table 15-4	Resources for L1/L2 Cache.....	101
Table 16-1	Instruction Attributes/Latency (ARMv8) .....	103
Table 16-2	Instruction Attributes/Latency (ARMv8 SIMD&FP).....	114
Table 16-3	Instruction Attributes/Latency (SVE).....	126

---

# Preface

---

本書は、A64FX プロセッサのマイクロ・アーキテクチャの解説、およびソフトウェア・チューニングのための参考情報を提供することを目的としている。

本書の執筆においては以下の文書を参考にしている。これらの文書にて定義されている用語については、特に注釈なしに使用している。

- A64FX 論理仕様書（2020 年 6 月公開予定）
- ARM® Architecture Reference Manual (ARMv8, ARMv8.1, ARMv8.2, ARMv8.3)
- ARM® Architecture Reference Manual Supplement The Scalable Vector Extension

## Typographical and Notational Conventions

本書における表現規則を以下にまとめます。

### アセンブラー表記

アセンブラーのシンタックスは ARM® Architecture Reference Manual (以下、ARM Manual) に準拠し、すべて *Consolas* かつ小文字で記述する。

### 命令表記

命令の表記は基本的に ARM Manual に準拠し、Times New Roman、かつ大文字で記述する。ただし、命令属性／レイテンシ一覧表のみ Cambria で記述する。また、複数の命令をグループとして表現するために正規表現のような命令名の展開表現を導入する。

命令の展開表記は以下の通りとする。

*	アスタリスク	任意の文字列に展開される。長さ 0 の文字列を含む。
[ and ]	ブラケット	[ ] 内の文字列のうち、いずれか 1 文字に展開される。[ ]内の - (ハイフン) は文字範囲を表し、範囲内の 1 文字に展開される。
{ and }	カーリーブラケット	{ } 内の   (パイプ) で区切られた文字列のうち、いずれか 1 つに展開される。  の前後に文字列がないときは Null 文字列を表現する。

### 命令の区分 (class) 表記

命令名のみで区別できない命令がある。例えば、ADD 命令には Base Instruction に属する ADD (extended register)、ADD (immediate)、ADD (shifted register)、SIMD&FP に属する ADD (vector)、SVE に属する ADD (immediate)、ADD (vectors, predicated)、ADD (vectors, unpredicated) の 7 命令が存在する。これらの命令を表記する場合は ARM Manual に従って () で修飾する。また、Base Instructions に属する ADD 命令すべてを表す場合には ADD (base) のように区分名 (class) を用いて表記する。

### Variant 表記

ハードウェアの振る舞いは、命令の動作のほかにデータ型にも影響される。特に、 SIMD&FP、SVE 命令には同一の命令でありながらも、データ型によってハードウェアの動作やオペレーション数が大きく異なるものが存在する。そのため、必要に応じて Variant の修飾を付加する。Variant は以下に示すように命令表記の後方に - (ハイフン) を置き、続けて記述する。Variant の表記は命令の区分によって異なるが、原則としてデータ型を表すレジスタ表記としている。

例)

Base instruction	: ADD (immediate) - W
SIMD&FP instruction	: FADD (scalar) - [HS]
	FADD (vector) - {8B 16B}
SVE instruction	: ADD (immediate) - [SD]

## Terminology

### 命令 (Instruction)

ARMv8 Manual C6.2 章 「Alphabetical list of A64 base instructions」、および SVE Manual 5 章 「SVE Instruction Set」において定義されている個々の Instruction を指す。ARMv8 Manual と同様に命令の型 (form) の違いを () 表記にて区別する。命令の型の違いを区別しない場合は、上述の命令区分表記に従って () を省略する、または区分名表記を用いる。また、μOP 命令と明示的に区別する必要がある場合にはアーキテクチャ命令と表現する。

### μOP 命令 (μOP instruction)

プロセッサによりデコードされた命令の形式を指す。基本的にプロセッサのアウト・オブ・オーダ実行エンジンでは、すべての操作を μOP 命令として取り扱う。

### 整数命令 (Integer instruction)

ARMv8 Manual にて A64 Base Instruction として定義される命令を指す。主に整数値を扱うことから本書では整数命令と呼称する。

### SIMD&FP 命令 (SIMD&FP instruction)

ARMv8 Manual にて A64 Advanced SIMD and Floating-point Instruction として定義される命令を指す。

### SVE 命令 (SVE instruction)

SVE Manual にて定義される命令を指す。

### Variant

同一命令でレジスタ・サイズ、または要素あたりのデータ・サイズを複数指定できるとの表現である。Variant は命令の区分ごとに指示する意味が異なることに注意しなければならない。

整数命令には 32-bit variant と 64-bit variant があり、variant 表記ではそれぞれ W, X とする。

SIMD&FP 命令にはレジスタ・サイズそのものを表すものと、要素あたりのデータ・サイズを表すものがある。例えば、FADD (scalar) の Variant はレジスタ・サイズを示し、FADD (vector) の Variant は要素あたりのデータ・サイズを示している。

SVE 命令では演算の要素あたりのデータ・サイズと、メモリ・アクセス・サイズをそれぞれ独立して指定できる。そのため、演算のデータ・サイズ variant を esize、メモリ・アクセス・サイズを memsize として表現する。

整数命令と SIMD&FP 命令の esize と memsize の variant は基本的には一致しているが、どちらか一方に言及する場合には esize、memsize を明記する。

### Vector Length (VL)

SVE Manual にて定義されている Vector Length そのものを指す。本書では Vector Length は bit 数にて表現する。

### ベクトル・データ長

SIMD&FP 命令と SVE 命令における有効レジスタ・サイズ、または、メモリ・アクセスのサイズの総称である。

## **Element 数**

SIMD&FP 命令と SVE 命令におけるベクトル・データ長を esize、または memsize で除した数を指す。すなわち、ベクトル・データ内の要素数に相当する。

## **演算命令**

命令の入力オペランドと出力オペランドが同一のレジスタ・ファイルに閉じている命令を指す。算術演算、論理演算、ビット演算の命令などが該当する。広義では MOV 命令などの転送命令も含む。ただし、本書では異なるレジスタ・ファイル間における転送命令は演算命令として扱わない。

## **ロード／ストア命令**

メモリ空間からレジスタにデータを転送する命令をロード命令、レジスタからメモリ空間にデータを転送する命令をストア命令とする。

## **命令ディスパッチ**

デコーダからリザベーション・ステーションにオペレーション・フローを割り当てる動作である。

## **命令発行**

リザベーション・ステーションから実行パイプラインにオペレーション・フローを投入することである。

## **実行完了**

アーキテクチャ命令、μOP 命令、オペレーション・フローの実行が完了した状態を示す。投機状態の完了を含む。

## **命令コミット**

アーキテクチャ命令が実行完了し、かつ投機状態を確定しプロセッサのアーキテクチャ・ステイトを更新することである。本書では実行完了と明確に区別して表現する。

## **オペレーション・フロー、オペレーション・リクエスト**

μOP 命令を実行するためのパイプラインの動作そのものを指す。プロセッサはオペレーション・フローを組み合わせて μOP 命令を実行する。ハードウェア資源を消費する最小限の単位である。また、μOP 命令のように単にアーキテクチャ命令から派生した物を表すだけでなく、キャッシュ・アクセスやメモリ・アクセスの処理単位も表す。オペレーション・フローとオペレーション・リクエストの明確な区別はないが、フローの生成元と実行先に着目するときに、オペレーション・リクエストと表現する場合がある。なお、これらはフロー、リクエストに省略表記される場合がある。

# 1. はじめに

---

## 1.1. A64FX プロセッサの概要

A64FX プロセッサ (以下、A64FX と記述する) は High Performance Computing (HPC) 向けに設計され、ARMv8-A architecture profile、および Scalable Vector Extension for ARMv8-A に準拠したアウト・オブ・オーダ実行型スーパースカラ・プロセッサである。プロセッサは冗長コアを含む 52 個のプロセッサ・コア、HBM2 に対応したメモリ・コントローラ、Tofu-D インターコネクトのコントローラ、PCI-Express Gen3 対応ルート・コンプレックスを集積している。

A64FX は HPC 向けにいくつかの特徴的なアーキテクチャを採用している。

### Scalable Vector Extension

A64FX は ARM 命令セットアーキテクチャのベクトル拡張である Scalable Vector Extension (SVE) をサポートする。SVE は命令セットとして 2,048 bits までの Vector Length が定義されている一方で、ハードウェアに実装する Vector Length は 128 bits の倍数から選択できることを特徴とする。A64FX では 128 / 256 / 512 bits の Vector Length をサポートする。

### Core Memory Group

A64FX はその内に 13 個のプロセッサ・コア、独立した L2 キャッシュ、独立したメモリ・コントローラからなる Core Memory Group (CMG) と呼ばれるグループを持つ。プロセッサは 4 つの CMG を持ち、CMG 間は Non-Uniform Memory Access (NUMA) 構成である。物理メモリ空間はそれぞれ分割されており、キャッシュ・コヒーレンスはハードウェアによって暗黙に保証される。

### セクタ・キャッシュ

キャッシュを way 単位で仮想的に分割し、命令レベルで使用できる領域を指定できる機能である。プログラムはタグド・アドレスを使用することで領域を指定できる。L1 キャッシュは 4 セクタ、L2 キャッシュは 2 セクタを 2 グループ持っている。

### ハードウェア・バリア

ソフトウェアのプロセス、またはスレッド間の同期をハードウェアでサポートする機能である。この機能により、メモリ・アクセスを行わずに同期処理ができる。

### ハードウェア・プリフェッチ・アシスト

ハードウェア・プリフェッチの振る舞いをプログラムから制御できる機能である。プログラムはシステムレジスタとタグド・アドレスを用いてハードウェアのプリフェッチ機構に情報を与えることができる。

## High Bandwidth Memory

メインメモリに High Bandwidth Memory Gen2 (HBM2) を採用し、非常に高いメモリ帯域を提供している。

## 1.2. A64FX プロセッサの諸元

A64FX プロセッサの主な諸元を Table 1-1 に示す。

**Table 1-1 A64FX Processor Specifications**

	<b>Specification</b>
Number of processor cores	52 (13 cores / CMG)
Number of CMGs	4
L1I cache size	64 KiB / 4-way
L1D cache size	64 KiB / 4-way
L2 cache size	32 MiB / 16-way (8 MiB / CMG)
Cache-line size	256 bytes
Memory controller	4 (1 MAC / CMG)
Interconnect	Tofu-D
I/O	PCI-Express Gen3 16 Lanes
Instruction set architecture	ARMv8-A, ARMv8.1, ARMv8.2, ARMv8.3 <sup>(*)</sup> , SVE
SVE-implemented Vector Length	128 / 256 / 512 bits

(\*) ARMv8.3 supports only complex-number supported instructions.

### 1.3. A64FX プロセッサのブロック

A64FX プロセッサ・チップ上の主要な機能ブロックを Figure 1-1 に、各ブロックの表示と機能ユニットとの対応関係を Table 1-2 に示す。

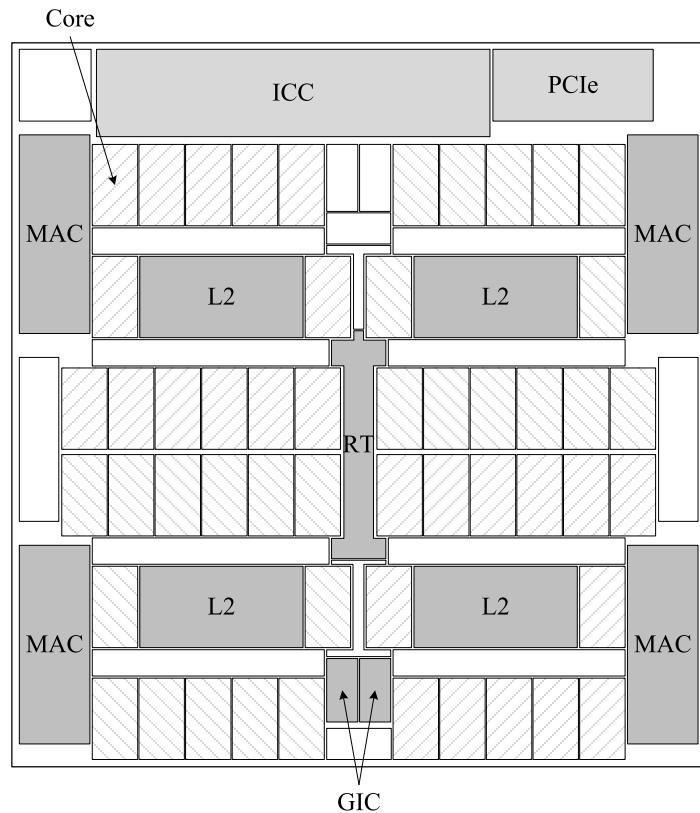


Figure 1-1 Main Functional Blocks on A64FX Processor Chip

Table 1-2 Correspondence Between Processor Chip Block Markings and Functional Units

Block Marking in Figure	Functional Unit
Core	Processor core
L2	L2 cache
ICC	Tofu-D interconnect controller
PCIe	PCI-Express Gen3 root complex
RT	Routing controller between CMGs
MAC	Memory controller
GIC	Interrupt controller

## 2. アウト・オブ・オーダ・アーキテクチャ

本章では A64FX プロセッサ・コアの基本的なアーキテクチャについて説明する。

### 2.1. アーキテクチャ概要

A64FX の基本的なパイプラインとステージの概要を Figure 2-1 に示す。A64FX は大きく 5 つの機能ステージにわたることができる。

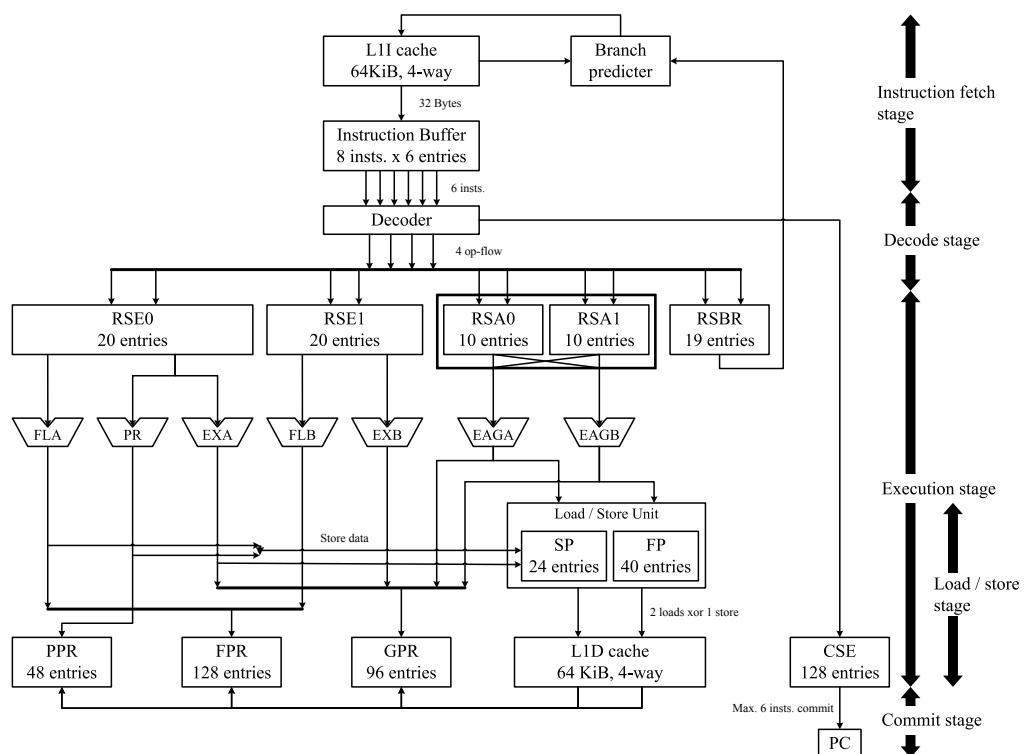


Figure 2-1 Overall Illustration of Stages

#### 命令フェッチ・ステージ (Instruction fetch stage)

L1I キャッシュ、L1-ITLB、L2-ITLB、分岐予測機構、命令フェッチの制御モジュールから構成される。命令フェッチ・ユニットは L1I キャッシュから最大 8 命令を同時にフェッチすることができる。分岐予測機構は 1 サイクルあたり最大 4 つの分岐命令について分岐方向を予測し、最大 1 つの Taken 分岐命令について分岐先を予測する。フェッチされた命令は Instruction Buffer (IBUFF) に一時的に保存される。

### デコード・ステージ (Decode stage)

IBUFF から 1 サイクルあたり通常で最大 4 命令、MOVPRFX 命令を含むときは最大 6 命令を、取得してデコードを行う。取得したアーキテクチャ命令は内部命令形式である μOP 命令にデコードされる。基本的に 1 つのアーキテクチャ命令は 1 つの μOP 命令に分解されるが、複雑なオペレーションを要するアーキテクチャ命令は複数の μOP 命令に分解される。一方で、MOVPRFX 命令は被修飾命令と Pack 処理され、あたかも 1 つのアーキテクチャ命令であるかのようにデコードされる。デコードされた μOP 命令はイン・オーダにオペレーション・フローとしてリザベーション・ステーションにディスパッチされる。

### 実行ステージ (Execution stage)

実行パイプラインのグループごとに 5 つのリザベーション・ステーション (RS) が実装されている。RS にディスパッチされたオペレーション・フローは RS によってスケジューリングされ、アウト・オブ・オーダで発行される。実行パイプラインは次のパイプラインから構成される。主に整数演算を行う整数演算系パイプライン (EXA / EXB)、SIMD&FP、SVE 命令の演算を行う浮動小数点演算系パイプライン (FLA / FLB)、Predicate 命令の演算を行う Predicate 演算系パイプライン (PR)、ロード／ストア命令のアドレス生成と一部の整数演算を行うアドレス演算系パイプライン (EAGA / EAGB)、分岐命令を実行する分岐実行パイプラインである。

### ロード／ストア・ステージ (Load / store stage)

L1D キャッシュ、L1-DTLB、L2-DTLB、2 本のロード／ストア・パイプラインで構成される。アドレス演算系パイプラインはロード／ストア・パイプラインに直接結合されている。ロード／ストア・パイプラインは同時に 2 つのロード・オペレーション・フロー、または 1 つのストア・オペレーション・フローを実行できる。

### コミット・ステージ (Commit stage)

実行結果の例外確認、分岐予測結果の確認など命令の完了判定を行う。実行が完了した μOP 命令はイン・オーダでコミットされる。アーキテクチャ命令と対になる全ての μOP 命令がコミットするとプロセッサのアーキテクチャ・ステイトを更新する。1 サイクル当たり最大 4 つの μOP 命令をコミット可能である。

## 2.2. マイクロ・オペレーション命令

A64FX ではアーキテクチャ命令は内部命令形式であるマイクロ・オペレーション命令 (μOP 命令) にデコードされる。μOP 命令はハードウェアの命令実行に適した命令単位である。複雑なアーキテクチャ命令は複数の μOP 命令に分解される。一方で、ハードウェアのオペレーションに最適になるように、いくつかのアーキテクチャ命令は 1 つの μOP 命令に結合される。アーキテクチャ命令の分解数は命令属性／レイテンシ一覧表に記載されている。

1 つのアーキテクチャ命令から 2 つ以上の μOP 命令に分解されるデコードには、標準的なデコードと、ディスパッチ、コミット、およびリソース割り当てに制限があるシーケンシャル・デコードの 2 種類がある。シーケンシャル・デコードでデコードされるか否かはデコード元のアーキテクチャ命令によって決まる。

## 2.3. オペレーション・フロー

オペレーション・フローは μOP 命令を実行するための回路動作そのものであり、実行エンジンのパイプライン処理の最小単位である。μOP 命令はオペレーション・フローの組み合わせによって実行される。つまり、実行ステージとロード／ストア・ステージの処理単位はすべてオペレーション・フローである。μOP 命令はデコーダからリザベーション・ステーションにディスパッチされるときにオペレーション・フローに変換される。また、オペレーション・フローは単に μOP 命令の派生物ではなく、ハードウェアが自発的に生成するものも含む。例えば、ハードウェア・プリフェッチャや、キャッシュ・ミスの処理のためのオペレーション・フローなどである。

## 2.4. アウト・オブ・オーダ・リソース

主なアウト・オブ・オーダ実行のためのリソースとその数を Table 2-1 に示す。

Table 2-1 Out-of-Order Resources

Resource	Quantity of Resource		
Commit Stack Entry (CSE)	128 entries		
Group ID (GID)	32 entries		
General-purpose physical register (GPR)	96 entries	Architecture register	32 entries
		Renaming register	64 entries
Floating-point physical register (FPR)	128 entries	Architecture register	32 entries
		Renaming register	96 entries
Predicate physical register (PPR)	48 entries	Architecture register	16 entries
		Renaming register	32 entries
Reservation Station for EAG (RSA)	10 entries x 2 (split)		
Reservation Station for EXE (RSE) (shared by Integer, SIMD&FP, SVE)	20 entries x 2 (split)		
Reservation Station for Branch (RSBR)	19 entries		
Temporary Operand Register (TOR)	3 entries		
Fetch Port (FP)	Virtual	160 entries	
	Real	40 entries	
Store Port (SP)	Virtual	192 entries	
	Real	24 entries	
Write Buffer (WB)	8 entries		

それぞれのリソースの主な機能は以下の通りである。

### **Commit Stack Entry (CSE)**

アウト・オブ・オーダで実行された命令をプログラム・オーダにリオーダするためのリソースである。アーキテクチャ命令は μOP 命令にデコードされて CSE のエントリに割り当てられる。

### **Group ID (GID)**

μOP 命令のディスパッチ・グループを管理する ID である。1 GID あたり最大 4 μOP 命令が割り当てられる。

### **General-purpose physical register (GPR)**

ARM Manual における汎用レジスタに割り当てられるアーキテクチャ・レジスタとリネーミング・レジスタの物理的実体である。

### **Floating-point physical register (FPR)**

ARM Manual における SIMD&FP レジスタと、SVE Manual におけるベクトルレジスタに割り当てられるアーキテクチャ・レジスタとリネーミング・レジスタの物理的実体である。

### **Predicate physical register (PPR)**

ARM Manual における Predicate レジスタに割り当てられるアーキテクチャ・レジスタとリネーミング・レジスタの物理的実体である。

### **Reservation Station for EAG (RSA)**

EAGA / EAGB パイプラインで実行されるオペレーション・フローを一時的に保存し、アウト・オブ・オーダで発行するためのスケジューラである。

### **Reservation Station for EXE (RSE)**

EXA / EXB / FLA / FLB / PR パイプラインで実行されるオペレーション・フローを一時的に保存し、アウト・オブ・オーダで発行するためのスケジューラである。

### **Reservation Station for BRanch (RSBR)**

分岐命令のオペレーション・フローを一時的に保存し、アウト・オブ・オーダで実行するためのスケジューラである。

### **Temporary Operand Register (TOR)**

命令フェッチステージからプログラム・カウンタ (PC) の値を実行ステージに転送するためのレジスタである。基本的には PC 相対の命令と Branch and Link 命令のみで使用される。

### **Fetch Port (FP)**

ロード／ストア命令の実行順序を管理するためのリソースである。A64FX では Virtual Fetch Port (VFP) と呼ばれる機能が新たに導入された。VFP に対して、本来の機能を持つ Fetch Port は Real Fetch Port (RFP) と呼ぶ。

### **Store Port (SP)**

ストア命令の実行順序を管理するためのリソースである。Fetch Port と同様に Virtual Store Port (VSP) と Real Store Port (RSP) がある。

### **Write Buffer (WB)**

コミット後のストアデータを L1D キャッシュ書き込みまで、一時的に保存するリソースである。

## 2.5. パイプライン・ステージ

アウト・オブ・オーダ実行におけるパイプライン・ステージについて記述する。パイプライン・ステージは実行パイプラインとそのパイプラインで実行される命令、演算、ロード／ストアの種別により異なる。

主なパイプラインステージのオペレーションを Figure 2-2 から Figure 2-6 に示す。なお、図中のステージ表記とオペレーションの対応関係は Table 2-2 の通りである。

Table 2-2 Correspondence Between Pipeline Stage Symbols and Operations

Stage Symbol	Operation
Common to all pipelines	
D, DT	Instruction decode
P, PT	Instruction scheduling
B*	Physical register read
C	Commit
W, W2	Architecture register update
Specific to operation pipelines	
Xn	Operation execution (The number of stages varies depending on the instruction.)
U, UT*	Operation result update
EXP	Exception judgment
Specific to branch execution pipeline	
BS	Scheduling
BR	Branch direction judgment
BC	Branch direction determination
Specific to load/store pipelines	
A	Effective address generation
T	Tag and TLB access
M, B, XT, XM, XB	Data access
R, RT*	Result out
W3 – W5	WB write

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
Integer	D	DT		P	PT		B1	B2	X1	X2	X3	X4	X5	X6	X7	X8	X9	U	UT	UT2		C	W	W2

Figure 2-2 Integer Operation Pipeline Stages

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	
SIMD&FP/ SVE	D	DT	P	PT	PT2	PT3	B1	B2	X1	X2	X3	X4	X5	X6	X7	X8	X9	U	UT	UT2		EXP	C	W	W2

Figure 2-3 SIMD&FP and SVE Operation Pipeline Stages

	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Predicate	D	DT	P	PT	PT2	B1	B2	X1	X2	X3	U	UT		
Predicate (update NZCV)	D	DT	P	PT	PT2	B1	B2	X1	X2	X3	U	UT		

Figure 2-4 Predicate Operation Pipeline Stages

	1	2	3	4	5	6	7	8	9	10	11	12	13
Unconditional branch	D	DT	BS	BC	C	W							
Conditional branch	D	DT	BS	BR	BC	C	W						
Unconditional indirect branch	D	DT	P BS	PT	B1	B2	X	U	UT	BR	BC	C	W
Compare & branch	D	DT	P BS	PT	B1	B2	X	U	UT	BR	BC	C	W

Figure 2-5 Branch Pipeline Stages

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22		
Integer load	D	DT	P	PT	B1	B2	X	A	T	M	B	R	RT	RT2	RT3	C	W	W2						
Integer store	D	DT	P	PT	B1	B2	X	A	T	M	B	R	RT	RT2	RT3	C	W	W2	W3	W4	W5			
SIMD&FP/SVE load (short)	D	DT	P	PT	B1	B2	X	A	T	M	B	R	RT	RT2	RT3	C	W	W2						
SIMD&FP/SVE load (long)	D	DT	P	PT	B1	B2	X	A	T	M	B	XT	XM	XB	R	RT	RT2	RT3	C	W	W2			
SIMD&FP/SVE store (short)	D	DT	P	PT	B1	B2	X	A	T	M	B	R	RT	RT2	RT3	C	W	W2	W3	W4	W5			
SIMD&FP/SVE store (long)	D	DT	P	PT	B1	B2	X	A	T	M	B	XT	XM	XB	R	RT	RT2	RT3	C	W	W2	W3	W4	W5
Predicate load	D	DT	P	PT	B1	B2	X	A	T	M	B	XT	XM	XB	R	RT	RT2	RT3	C	W	W2			
Predicate store	D	DT	P	PT	B1	B2	X	A	T	M	B	XT	XM	XB	R	RT	RT2	RT3	C	W	W2	W3	W4	W5

Figure 2-6 Load/Store Pipeline Stages

## 2.6. 命令実行レイテンシ

命令実行の基本レイテンシは、上記パイプライン・ステージの演算ステージの演算開始から演算完了、または、ロード／ストア・ステージのメモリ・アクセス開始からアクセス完了のステージ数で決定される。Table 2-3 に各パイプラインと各命令における実行開始と、完了のステージについてまとめた。また各命令のレイテンシは命令属性／レイテンシ一覧に記載されている。

Table 2-3 Execution Start and Completion Stages for Each Instruction in Each Pipeline

Pipeline	Instruction	Start Stage	Completion Stage
EXA / EXB		X	Xn
EAGA / EAGB	(Operation instruction only)		
FLA / FLB		X1	Xn
PR		X1	X3
Load / Store	Integer load	A	R
	SIMD&FP / SVE load	A	RT3
	Predicate load	A	RT

ストア命令は実行完了がコミット後になることから、一般的な実行レイテンシとは異なるためここでは省略する。

EXA / EXB、EAGA / EAGB、FLA / FLB の各パイプラインは、命令のオペレーションの内容によってステージ数が異なるため Xn と表現している。

## 2.7. オペランド・バイパス

オペランド・バイパスとは、後続命令のオペランドが先行命令の実行結果に依存するとき、先行命令で生成された値をレジスタを経由せずに後続命令に渡すことをいう。基本的には、前述の Table 2-3 にある完了ステージの直後に次の命令の開始ステージが接続されるようにバイパスの経路が実装されている。しかし、実行パイプラインや命令の組み合わせによってはペナルティなくバイパスできない場合がある。パイプラインや命令の種類に依存するオペランドの組み合わせで決まるペナルティ・サイクルを Table 2-4 に示す。表の縦軸はオペランドを生成する命令、横軸はそのオペランドを入力とする命令を表す。

**Table 2-4 Penalties for Operand Bypass Between μOP Instructions**

	<b>Consumer</b>	<b>EXA</b>	<b>EXB</b>	<b>EAGA / PIPE0</b>				<b>EAGB / PIPE1</b>				<b>FLA</b>	<b>FLB</b>	<b>PR</b>
Producer				SIMD&FP/SVE operation	-			Predicate operation	-					
				SIMD&FP/SVE operation	-			SIMD&FP/SVE operation	-					
EXA	Integer operation	0	1	1	1	1	1	1	1	1	1	-	-	-
	Integer operation (update NZCV)	0	1	-	-	-	-	-	-	-	-	7	7	6
EXB	Integer operation	1	0	1	1	1	1	1	-	1	1	1	1	-
	Integer operation (update NZCV)	1	0	-	-	-	-	-	-	-	-	-	7	7
EAGA / PIPE0	Integer operation	1	1	0	0	0	0	0	0	-	1	1	1	-
	Integer load	0	0	0	0	0	0	0	0	-	0	0	0	-
	Integer store	-	-	-	-	-	-	-	-	-	-	-	-	-
	SIMD&FP/SVE load (short)	-	-	-	-	-	-	-	-	-	-	-	0	0
	SIMD&FP/SVE load (long)	-	-	-	-	-	-	-	-	-	-	-	0	0
	SIMD&FP/SVE store (short)	-	-	-	-	-	-	-	-	-	-	-	-	-
	SIMD&FP/SVE store (long)	-	-	-	-	-	-	-	-	-	-	-	-	-
	Predicate load	-	-	-	-	-	0	0	-	-	-	0	0	3
EAGB / PIPE1	Predicate store	-	-	-	-	-	-	-	-	-	-	-	-	-
	Integer operation	1	1	1	1	1	1	1	1	-	0	0	0	-
	Integer load	0	0	0	0	0	0	0	0	-	0	0	0	-
	Integer store	-	-	-	-	-	-	-	-	-	-	-	-	-
	SIMD&FP/SVE load (short)	-	-	-	-	-	-	-	-	-	-	-	0	0
	SIMD&FP/SVE load (long)	-	-	-	-	-	-	-	-	-	-	-	0	0
	SIMD&FP/SVE store (short)	-	-	-	-	-	-	-	-	-	-	-	-	-
	SIMD&FP/SVE store (long)	-	-	-	-	-	-	-	-	-	-	-	-	-
PR	Predicate load	-	-	-	-	-	0	0	-	-	-	0	0	3
	Predicate store	-	-	-	-	-	-	-	-	-	-	-	-	-
PR	Predicate operation	-	-	-	-	-	1	1	-	-	-	1	0	-

	Consumer	EXA	EXB	EAGA / PIPE0								EAGB / PIPE1					FLA	FLB	PR	
	Predicate operation (update NZCV)	6	6	-	-	-	-	-	-	-	-	-	-	-	-	-	8	8	7	
FLA	SIMD&FP/SVE operation	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	0	0	-	
	SIMD&FP/SVE operation (update NZCV)	5	5	-	-	-	-	-	-	-	-	-	-	-	-	-	0	0	6	
	SVE CMP instruction (update PR)	-	-	-	-	-	2	2	-	-	-	-	-	2	2	-	-	1	1	2
	SVE CMP instruction (update NZCV)	9	9	-	-	-	-	-	-	-	-	-	-	-	-	-	11	11	10	
FLB	SIMD&FP/SVE operation	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	0	0	-	
	SIMD&FP operation (update NZCV)	5	5	-	-	-	-	-	-	-	-	-	-	-	-	-	0	0	6	

基本的にペナルティ・サイクルは命令の種類、および生成側と入力側のパイプラインの組み合わせで決まる。また、いくつかの命令では複数のオペランドで依存を持つことができる。例えば、整数演算では汎用レジスタと NZCV レジスタへ同時に output する命令がある。このとき、後続の命令が汎用レジスタのオペランドに依存するときと、NZCV レジスタのオペランドに依存するときではペナルティ・サイクルが異なる。SVE 命令は基本的に浮動小数点レジスタと Predicate レジスタを入力オペランドとする。このとき、どちらのオペランドが先行命令に依存するかによってペナルティ・サイクルが異なる。Table 2-4 の生成命令側にはオペランドの種類の組み合わせについても記述している。入力の命令側はそれに対応するオペランドが入力となる。

なお、FTMAD 命令については上記のルールに当てはまらず、オペランドを生成する命令によってペナルティ・サイクルが変化する。FTMAD 命令のペナルティ・サイクルを Table 2-5 に示す。

Table 2-5 Penalties for Operand Bypass Between μOP Instructions (FTMAD Instruction)

	FTMAD
SVE load (Long)	0
FTSMUL	0
Other floating-point operation instruction	1

## 2.8. リソースの割り当てと解放

アウト・オブ・オーダー・リソースは命令実行の度に割り当てられ、実行が終了すると解放される。それぞれのリソースの割り当てステージ、および解放ステージについて Table 2-6 に示す。

Table 2-6 Out-of-Order Resource Allocation and Release Stages

Resource		Allocation Stage	Release Stage	Supplemental Remarks
CSE		D	W	The resource is allocated and released in order.
General-purpose renaming register		D	W2	
Floating-point renaming register		D	W2	
Predicate renaming register		D	W2	
RSE	EXA / EXB pipeline	D	B1, B2, X, X+1	The release stage depends on the operand bypass timing. The resource is released out of order.
	FLA / FLB pipeline	D	PT2, PT3, PT3+1	The release stage depends on the operand bypass timing. The resource is released out of order.
RSA	Load/Store instruction	D	B2, A, A+1	The release stage depends on the operand bypass timing. The resource is released out of order.
	Integer operation instruction	D	B1, B2, X, X+1	The release stage depends on the operand bypass timing. The resource is released out of order.
Virtual FP		D	Same as for Real FP	The resource is allocated in order.
Virtual SP		D	Same as for Real SP	
Real FP	Integer load / store, SIMD&FP load / store, SVE load / store (excluding predicate)	B1	RT3	The resource is released in order and without waiting for commit.
	Predicate load / store		RT3	
Real SP	Integer store	B1	W5	The resource is released in order after commit.
	SIMD&FP / SVE store	PT2		

## 2.9. レイテンシ切り替え

同一のパイプラインを通過するオペレーションであっても、命令の種類によってレイテンシが異なる。このようなとき、Figure 2-7 に示すように複数の命令について、パイプラインへの投入タイミングが異なっていても後方のステージ (C, W) においてオペレーションが衝突してしまうことがある。このような状態はパイプラインとして成立し得ないため、A64FX では Figure 2-8 に示すように実行ステージにおけるレイテンシを切り替えることで衝突を回避している。

	1	2	3	4	5	6	7	8	9	10	11
Preceding instruction	X1	X2	X3	X4	X5	X6	X7	X8	X9	C	W
Following instruction						X1	X2	X3	X4	C	W

Figure 2-7 Example of Conflict Between C Stages of Instructions with Different Latencies

	<b>1</b>	<b>2</b>	<b>3</b>	<b>4</b>	<b>5</b>	<b>6</b>	<b>7</b>	<b>8</b>	<b>9</b>	<b>10</b>	<b>11</b>	<b>12</b>	<b>13</b>
Preceding instruction	X1	X2	X3	X4	X5	X6	X7	X8	X9	C	W		
Following instruction						X1	X2	X3	X4	$\frac{C}{X5}$	$\frac{W}{X6}$	C	W

**Figure 2-8 Example of Latency Changing**

レイテンシ切り替えが行われる命令の種類と、切り替え後のレイテンシの一覧を Table 2-7 に示す。

**Table 2-7 Instructions Whose Latency Changed, and Their Latencies**

	<b>Basic Latency</b>	<b>Latency After Change</b>
Instruction executed in floating-point operation pipeline	4	6 or 9
	6	9
	9	No change
Load instruction	5	8
	8	11
	9	No change
	11	No change

# 3. 命令フェッチ

## 3.1. 命令フェッチ・ステージの概略

命令フェッチ・ステージは、LII キャッシュから命令をフェッチしてデコード・ステージに命令を供給する。命令フェッチ・ステージには LII キャッシュ、L1-ITLB、分岐予測機構が含まれる。命令フェッチ・ステージの概略を Figure 3-1 に示す。IFEAG はプログラム・カウンタ (PC) を更新する加算器である。PC は分岐予測機構と L1-ITLB、LII キャッシュに送られる。分岐予測結果による PC、または IFEAG による PC のどちらかを基に L1-ITLB と LII キャッシュにアクセスし、命令を読み出す。命令の読み出しはアライメントされた 32 バイト単位で行われ、読み出された命令イメージのまま Instruction Buffer (IBUFF) に保存される。IBUFF は 6 エントリで構成され、1 エントリあたり 32 バイト (8 命令) を格納できる。

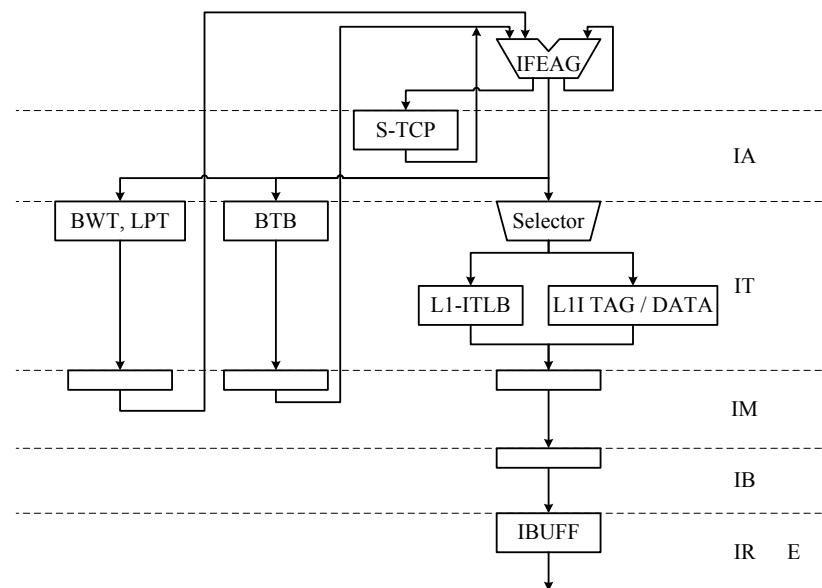


Figure 3-1 Instruction Fetch Stage

分岐予測機構は分岐命令の分岐方向と分岐先アドレスを予測する。フェッチされる命令列に“Taken”と予測される分岐命令が含まれる場合、次の命令フェッチ先は予測された分岐先アドレスである。基本的な分岐予測機構のアクセス・レイテンシは 3 サイクルである。このとき、Figure 3-2 に示すように Taken 分岐命令の後続命令のフェッチがキャンセルされるためにバブルが発生する。

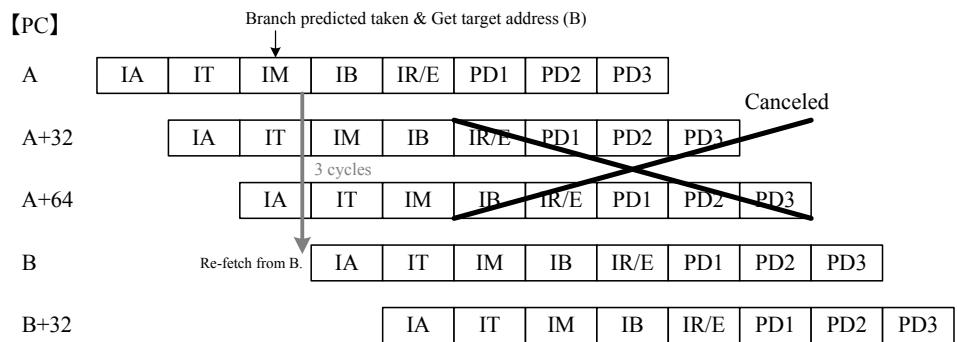


Figure 3-2 Bubbles Due to Instructions Following Taken Branch Instruction

## 3.2. 分岐予測機構

A64FX の分岐予測機構は Table 3-1 に示す分岐予測器から構成される。

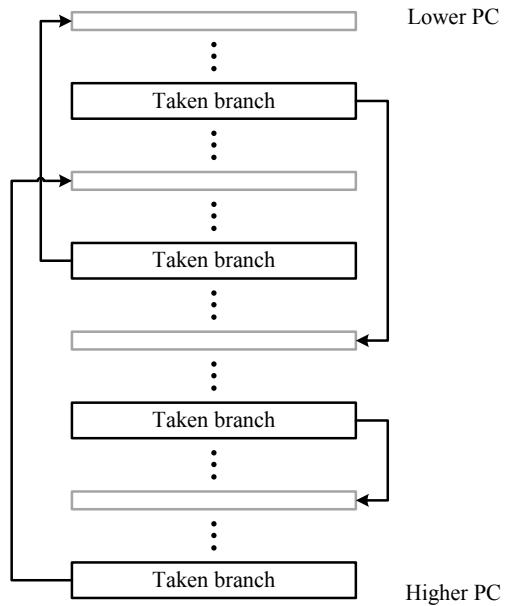
Table 3-1 Branch Predictors of Branch Prediction Mechanism

Role	Branch Predictor
Branch direction & Branch target address prediction	Small Taken Chain Predictor (S-TCP)
Branch direction prediction	Branch Weight Table (BWT) Loop Prediction Table (LPT) Return Address Stack (RAS)
Branch target address prediction	Branch Target Buffer (BTB)

このうち、BWT と BTB が主たる分岐予測機構であり、これらを組み合わせて分岐方向と分岐先アドレスを予測する。BWT は Global History Register (GHR) と組み合わせることで Piecewise linear 方式のアルゴリズムを用いて予測する。S-TCP は Taken 分岐命令の分岐先アドレスを記憶するための小容量、かつ短レイテンシなバッファであり、ループ構造を特定したときに予測をする。LPT はカウンタ方式のローカル分岐予測器で、ループ構造における継続判定を行う条件分岐命令のような、分岐方向が同方向である回数を予測する。RAS はサブルーチンからのリターンアドレスに関する予測器である。以降、各分岐予測器について詳細を記述する。

### 3.2.1. Small Taken Chain Predictor (S-TCP)

Small Taken Chain Predictor (S-TCP) は Taken 分岐命令によるプログラム実行パスのチェーンを検出し、予測をする機構である。Figure 3-3 に示すように、S-TCP は複数の Taken 分岐命令の実行パスからなるチェーン構造を検出する。このチェーンがループを形成し、実行の反復が一定回数を超えると検出した実行パスに従って命令フェッチを指示する。S-TCP は 4 つの Taken 分岐命令の分岐先アドレスを保存できる。S-TCP は Not taken 分岐命令の情報は保存しないため、チェーンに含まれる Not taken 分岐の命令数に制限はない。



**Figure 3-3 Chain Structure Consisting of Multiple Taken Branch Instructions**

S-TCP は予測ミスが発生しても直ちに検出した情報を削除しない。つまり、再度同一の実行パスの命令をフェッチすると S-TCP による予測を再開する。

また、S-TCP へは 1 サイクルでアクセスできることから命令フェッチ時のパイプライン・バブルは発生しない。

### 3.2.2. Loop Prediction Table (LPT)

Loop Prediction Table (LPT) はカウンタ方式のローカル履歴分岐方向予測器である。条件分岐命令の Taken または Not taken が連続した回数を記録し、その履歴を基に分岐方向を予測する。LPT は 8 エントリで構成され、分岐命令を 8 命令まで記録できる。

### 3.2.3. Branch Weight Table (BWT)

Branch Weight Table (BWT) は Global History Register (GHR) と組み合わせ、Piecewise linear アルゴリズムを用いて予測をする。BWT は Piecewise linear における重みテーブルであり、2,048 エントリ構成をとる。予測には Figure 3-4 に示すように、条件分岐の履歴、および過去の分岐命令実行時に算出した重みの履歴を使用する。2 つの履歴はいずれもグローバルな履歴である。条件分岐の履歴は Taken 時に “1”、Not taken 時に “-1” の 2 値をとる。重み (W) は符号付き整数である。Figure 3-4 に示すような条件分岐、および重みの履歴があるとき、条件分岐命令  $B_0$  は Figure 3-5 (Eq.1) の結果  $P$  として予測される。 $P$  が 0 以上のときは “Taken”、0 未満のときは “Not taken” と予測する。その後、分岐命令が実行されて分岐の結果が判明すると重みの履歴を更新する。重み履歴の更新は予測がミスしたときのみ行われる。例えば、分岐予測  $B_0$  の予測が “Taken” であったのに対して実行結果が “Not taken” であったときは Figure 3-5 (Eq.2) で示すような重みの更新式が使われる。

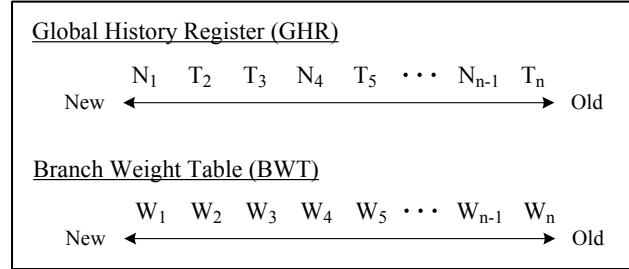


Figure 3-4 Histories of Conditional Branches and Weights

(Eq.1) Predicted threshold

$$P = W_0 + N_1 W_1 + T_2 W_2 + T_3 W_3 + N_4 W_4 + T_5 W_5 + \dots + N_{n-1} W_{n-1} + T_n W_n$$

(Eq.2) Update weights of the conditional branch instruction B<sub>0</sub>

$$\begin{aligned} [W_0 \ W_1 \ W_2 \ W_3 \ W_4 \ W_5 \ \dots \ W_{n-1} \ W_n] &= \\ [W_0 \ W_1 \ W_2 \ W_3 \ W_4 \ W_5 \ \dots \ W_{n-1} \ W_n] &+ \\ [T_c \ N_1 \ T_2 \ T_3 \ N_4 \ T_5 \ \dots \ N_{n-1} \ T_n] * N_0 \end{aligned}$$

\*  $T_c$  is constant.

Figure 3-5 Prediction Equation for Conditional Branch Instruction B<sub>0</sub>

なお、本マニュアルでは単純化のために予測結果を“Taken”、“Not taken”としているが、実装されている予測機構は Agree Prediction 方式である。

### 3.2.4. Branch Target Buffer (BTB)

Branch Target Buffer (BTB) は相対分岐命令と間接分岐命令の分岐先アドレスを記録するバッファである。BTB は 4-way、2,048 エントリの構成である。相対分岐命令については分岐先アドレスが静的に決まるため、過去 1 回分の分岐先履歴のみ保存する。間接分岐命令については分岐先アドレスが動的に変化する可能性があるため、複数の分岐先履歴を保存して予測をする Rehash と呼ばれる機構を採用している。Figure 3-6 に示すように、BTB のインデックスに対して分岐方向履歴と分岐先履歴を用いてハッシュを行う。これによって動的に変化する分岐先アドレスを予測できるようにしている。

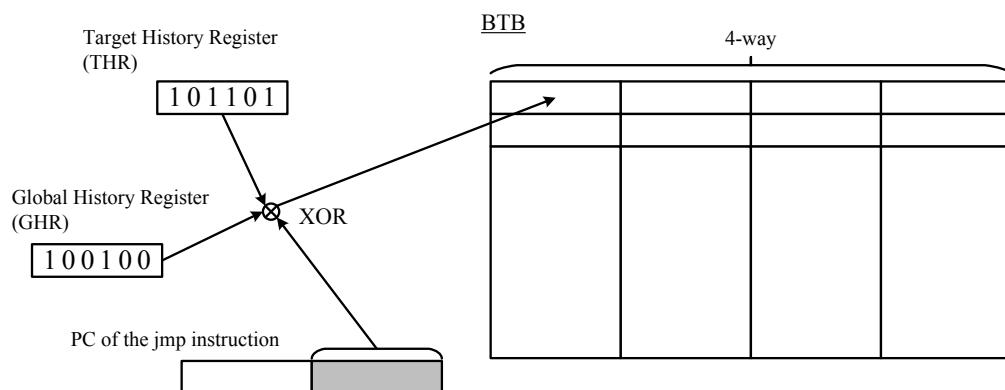


Figure 3-6 Outline of Branch Target Buffer (BTB)

### 3.2.5. Return Address Stack (RAS)

Return Address Stack (RAS) はサブルーチン・コール時にリターンアドレスを保存するスタックである。BL、BLR 命令実行時にリターンアドレスを記録し、RET 命令のフェッチ時に参照する。RAS は 8 エントリ構成をとる。RAS のアクセスサイクルは 2 サイクルであり、BTB より短い。

## 3.3. 分岐予測の組み合わせ

前述のように、A64FX では複数の予測器を組み合わせて分岐命令の予測を行う。分岐命令の種類と使用される予測器、さらに予測結果の採用順位を Table 3-2 に示す。

**Table 3-2 Relationship Between Predictors Used for Branch Prediction and Prediction Result Adoption Rankings**

Adoption Ranking	Conditional Branch Instruction	Indirect Branch Instruction, Unconditional Relative Branch Instruction
High	S-TCP	S-TCP
	LPT, BTB	RAS
Low	BWT, BTB	BTB

S-TCP はその動作の特徴から間接分岐命令、条件分岐命令に関係なく分岐先を予測する。さらに、アクセス・レイテンシも最も短いことから予測結果は最優先で採用される。LPT と BWT は分岐方向予測のみを行い、“Taken”と予測したときには BTB が予測した分岐先を使用する。LPT と BWT では LPT による予測結果が優先される。

間接分岐における RAS と BTB では RAS の結果が優先される。ただし、RAS はサブルーチンのリターン命令のみ予測するのに対して、BTB は全ての分岐命令について学習を行う。

## 3.4. ショート・ループ検出

A64FX ではショート・ループ検出と呼ばれる機構を実装している。この機構は IBUFF に保存されている命令列の中でループ構造を検出する機構である。ショート・ループを検出すると命令キャッシュからの読み出しを停止し、IBUFF から命令を供給できる。IBUFF からの命令供給時には Taken 分岐後続命令の読み出しひずれかはなく、パイプラインのバブルは発生しない。

ショート・ループの検出条件は以下の通りである。

- ループを構成する命令列全体が IBUFF に収まること。前述の通り、IBUFF には 48 命令を保存できるが、命令列のアライメント制約があることに注意が必要である。
- ループ内のすべての分岐命令の分岐方向が一定であること。ループ内には複数の分岐命令を含むことができる。

上記の条件を満たし、ループの反復回数がしきい値以上になると IBUFF からの命令供給を開始する。また、ショート・ループ内の分岐命令の方向が変化した場合は命令供給を終了し、命令キャッシュからの読み出しを再開する。

## 4. 命令デコードと命令コミット

---

### 4.1. マイクロ・オペレーション命令

A64FX では、アーキテクチャ命令はハードウェア固有の内部形式の μOP 命令としてデコードされる。μOP 命令は、リネーミング・レジスタ、CSE、FP、SP などの割り当ての基本単位となる。μOP 命令は 1 つのアーキテクチャ命令から複数に分解され生成されるものと、複数のアーキテクチャ命令を結合して 1 つの μOP 命令として生成されるものがある。μOP 命令の分解数は命令属性／レイテンシー一覧に記載している。アーキテクチャ命令の結合は 4.3 章にて説明している。

1 つのアーキテクチャ命令から 2 つ以上の μOP 命令への分解には、通常デコードとシーケンシャル・デコードの 2 種類がある。シーケンシャル・デコードとは、命令のディスパッチ、コミット、および GID の割り当てに制限があるデコードである。通常デコードは、同時に複数のアーキテクチャ命令をデコードし、複数の μOP 命令をディスパッチできる。それに対してシーケンシャル・デコードは、1 つのアーキテクチャ命令のみデコードし、μOP 命令を逐次的にディスパッチする。また、1 つの GID に 1 つの μOP 命令しか割り当てることができない。シーケンシャル・デコードになる命令は命令属性／レイテンシー一覧に記載されている。

### 4.2. マルチ・オペレーション

μOP 命令はオペレーション・フローにて実行される。必要なオペレーション・フロー数は μOP 命令の複雑さに依存する。例えば、単純な整数演算などは 1 フローで実行できるが、ADD (shifted register) 命令のような算術演算とシフトが組み合わさっている操作は、演算時に分割され複数のフローで実行される。また、NOP 命令などオペレーションが不要なものは、0 フロー（オペレーションが存在しない）となる。一方で、SVE の Gather load 命令などは複雑である。少數のアドレス生成・フローと多数のメモリ・アクセス・フローにて実行される。それぞれのパイプライン・ステージにて本質的な実行に必要なフロー数が異なることと、A64FX ではリソースの消費を抑制することをねらいとしてオペレーション・フローの分割はできるだけ下流のステージで行うアーキテクチャとなっている。オペレーション・フローの分割は以下のパイプラインステージにて行われる。

#### デコード・ステージ

μOP 命令をリザベーション・ステーションにディスパッチするときに分割される。主に命令実行のために機能が異なる実行パイプラインでの処理が必要な命令などが対象である。代表例としてストア命令がある。ストア命令は、RSA にディスパッチされる実効アドレスを計算するフローと、RSE にディスパッチされるデータ転送フローの 2 つに分割される。

#### 実行ステージ

リザベーション・ステーションから実行パイプラインへ発行されるときに分割される。複数の異なる演算を繰り返し行う命令が対象である。代表例として、ADD (shifted register) 命令があ

る。ADD 命令の μOP 命令はディスパッチ時は 1 フローに変換されるだけだが、リザベーション・ステーションから実行パイプラインへ発行されるときにシフト操作を行うフローと加算を行うフローの 2 つに分割される。

### ロード／ストア・ステージ

ロード／ストア・ステージにてロード／ストアを実行するときに分割される。Gather / Scatter 命令や Multiple structures 命令のような離散的なメモリ空間にアクセスする命令が対象である。

## 4.3. MOVPRFX 命令の Pack 処理

A64FX では MOVPRFX 命令は基本的に後続の被修飾命令と結合させ、あたかも被修飾命令が非デストラクティブ命令であるかのように振る舞うようにデコードされる。この結合処理を Pack 処理と呼ぶ。Pack 処理はプリデコードの第 1 ステージにて行われ、その後に μOP 命令分解を行う。すなわち、MOVPRFX 命令の有無はデコード時の μOP 命令数に影響を与えない。μOP 命令の分解数は被修飾命令の属性のみで決まる。

一方で、MOVPRFX 命令の Pack 処理には並列処理数に以下の制約がある。

- プリデコーダの第 1 ステージへの入力は、1 サイクルあたり最大 6 命令。
- プリデコーダの第 1 ステージでの Pack 処理は、1 サイクルあたり最大 3 組。
- プリデコーダの第 1 ステージからの出力は、Pack 処理後の命令数で最大 4 命令。

これらの制約から、プログラムにおけるアーキテクチャ命令の並びによってはプリデコードのスループットが低下することがある。一例を Figure 4-1、Figure 4-2 に示す。

```

loop:
    movprfx z0.d, p0/m, z1.d
    fmad    z1.d, p0/m, z2.d, z3.d
    add     z1.d, p0/m, z4.d
    movprfx z10.d, p0/m, z11.d
    fmad   z11.d, p0/m, z12.d, z13.d
    add    z11.d, p0/m, z14.d
    movprfx z5.d, p5/m, z6.d
    fmad   z6.d, p5/m, z7.d, z8.d
    sub    z6.d, p5/m, z9.d
    movprfx z15.d, p5/m, z16.d
    fmad   z16.d, p5/m, z17.d, z18.d
    sub    z16.d, p5/m, z19.d
    b.ne   loop

```

Figure 4-1 Example of Efficient Packing with MOVPRFX

```

loop:
    movprfx z0.d, p0/m, z1.d
    fmad     z1.d, p0/m, z2.d, z3.d
    movprfx z10.d, p0/m, z11.d
    fmad     z11.d, p0/m, z12.d, z13.d
    movprfx z5.d, p5/m, z6.d
    fmad     z6.d, p5/m, z7.d, z8.d
    movprfx z15.d, p5/m, z16.d
    fmad     z16.d, p5/m, z17.d, z18.d
    add      z1.d, p0/m, z4.d
    add      z11.d, p0/m, z14.d
    sub      z6.d, p5/m, z9.d
    sub      z16.d, p5/m, z19.d
    b.ne    loop

```

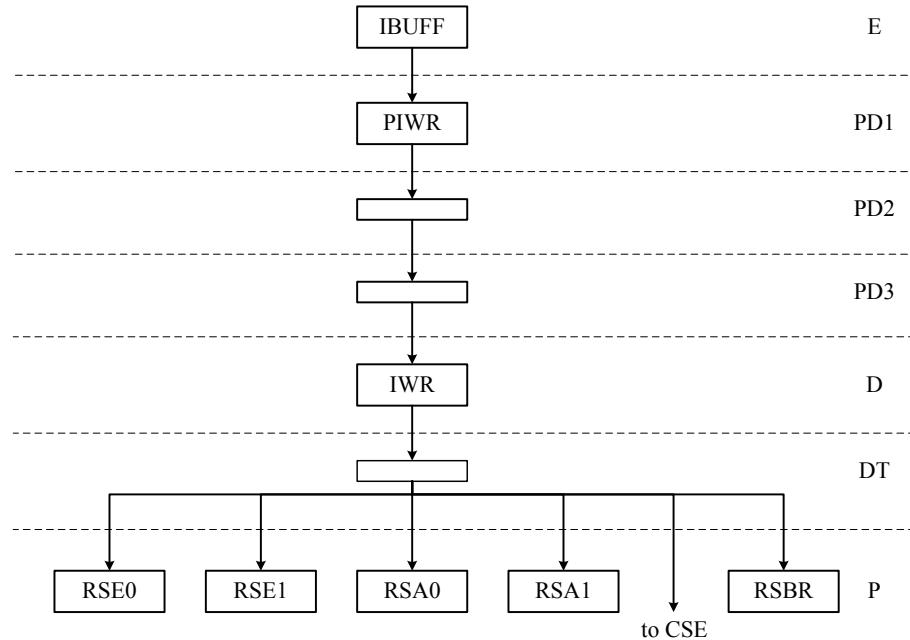
3 insts.  
decoded only  
 4 insts.  
decoded  
 2 insts.  
decoded only

**Figure 4-2 Example of Inefficient Packing Due to Instruction Order**

Figure 4-1、Figure 4-2 に示すように、同じ命令数をデコードするときであっても MOVPRFX 命令と被修飾命令の並びによってスループットが異なる。命令スケジューリングに自由度があるときは上記の制約を考慮することを推奨する。

## 4.4. 命令デコード

A64FX の命令デコード・ステージのパイプラインステージ概要を Figure 4-3 に示す。デコーダは IBUFF から命令を取得し、μOP 命令にデコードしてアウト・オブ・オーダ・リソースへの割り当てを行う。



**Figure 4-3 Instruction Decode Stage**

#### 4.4.1. プリデコード

プリデコードは PD1 ステージから PD3 ステージにて行われる。プリデコードでは主に MOVPRFX 命令の Pack 处理、μOP 命令への分解が行われる。PD1 ステージはアーキテクチャ命令で 6 命令幅の Pre-decode Instruction Windows Register (PIWR) を持ち、IBUFF から入力を受け付ける。PD2 ステージでは μOP 命令分解を行う。分解による命令数増加を吸収するために μOP 命令で 7 命令幅の命令レジスタを持つ。PD3 ステージは後段のデコーダへの出力であり、μOP 命令で 4 命令幅の命令レジスタを持つ。

まず、プリデコーダは PD1 ステージにて IBUFF から命令を取り出して PIWR に格納する。IBUFF からの読み出しは任意のアドレスからエントリをまたいで行うことができる。読み出された命令は IBUFF からクリアされる。なお、IBUFF からの読み出しには以下の制限がある。

- 4.3 章で説明した MOVPRFX 命令の Pack 处理における制約。Pack 处理は PD1 ステージで行われ、この制約を満たすように命令を読み出す。
- 分岐命令は一度に複数個読み出し可能であるが、“Taken” と予測された分岐命令は 1 命令のみ。
- Taken 予測された分岐命令の分岐先は同時に読み出せない。

PD1 ステージにて Pack 处理が行われた命令は、PD2 ステージから PD3 ステージにかけて μOP 命令分解が行われる。μOP 命令分解には以下の制約がある。

- Taken 分岐命令は PD3 において最後尾に配置される。
- 1 つのアーキテクチャ命令から 3 つ以上の μOP 命令に分解されるとき、その μOP 命令の組は PD3 において最前列から配置されなければならない。

シーケンシャル・デコードのときは、D ステージにて μOP 命令の展開が行われる。このとき、シーケンシャル・デコード対象の命令は最後尾に配置されなければならない制約があるため、PD2 ステージから PD3 ステージに送られる時点で、対象の命令が 1 命令かつ最後尾に配置されるように命令列が切られる。

それぞれの例を Figure 4-4 から Figure 4-6 に示す。

**PD2**

	V	Inst.	# of μOP
IWR0_H	0		
IWR1_H	0		
IWR2_H	0		
IWR3_H	1	A	1
IWR0_L	1	B	1
IWR1_L	1	TB	1
IWR2_L	1	C	1

**PD3**

	V	μOP
IWR0	0	
IWR1	0	
IWR2	0	
IWR3	0	

Next cycle

**PD2**

	V	Inst.	# of μOP
IWR0_H	0		
IWR1_H	0		
IWR2_H	1	C	1
IWR3_H	1	D	3
IWR0_L	1	E	1
IWR1_L	1	F	2
IWR2_L	1	G	2
IWR3_L	1	H	1

**PD3**

	V	μOP
IWR0	1	A
IWR1	1	B
IWR2	1	TB
IWR3	0	

Inst.C stays in PD2 even if PD3\_IWR3 is not full.

TB : Taken branch instruction.

Figure 4-4 Restriction on Taken Branch Instruction When Splitting μOP Instructions

**PD2**

	V	Inst.	# of μOP
IWR0_H	1	D	3
IWR1_H	1	E	1
IWR2_H	1	F	2
IWR3_H	1	G	2
IWR0_L	1	H	1
IWR1_L	1	I	1
IWR2_L	1	J	1
IWR3_L	1	K	3

**PD3**

	V	μOP
IWR0	1	C
IWR1	0	
IWR2	0	
IWR3	0	

Inst. D is not set to PD3 at this cycle even if PD3 is not full. Because Inst. D is being decoded to 3 μOPs.

Next cycle

**PD2**

	V	Inst.	# of μOP
IWR0_H	0		
IWR1_H	1	F	2
IWR2_H	1	G	2
IWR3_H	1	H	1
IWR0_L	1	I	1
IWR1_L	1	J	1
IWR2_L	1	K	3

**PD3**

	V	μOP
IWR0	1	D
IWR1	1	D
IWR2	1	D
IWR3	1	E

μOP Ds must be set from PD3\_IWR0.

Figure 4-5 Restriction Related to Three or More μOP Splits Resulting from μOP Instruction Splitting

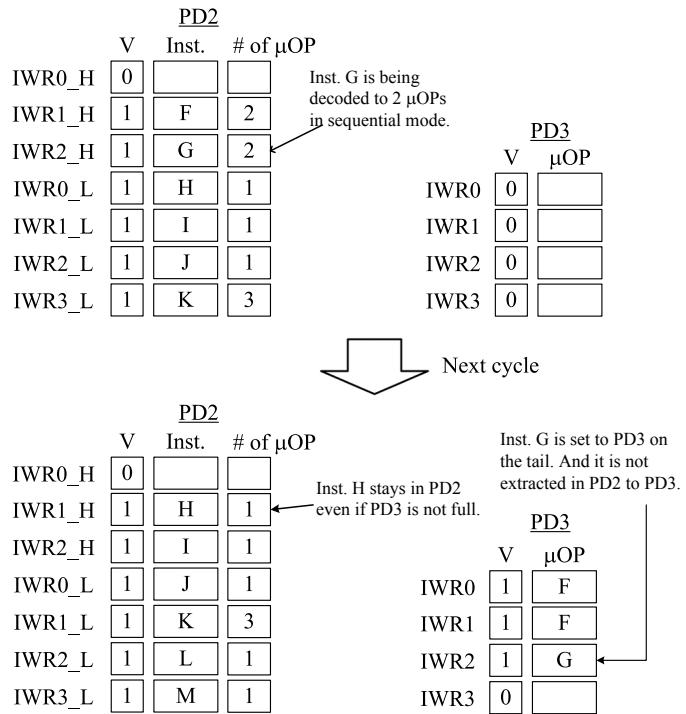


Figure 4-6 Restriction on μOP Instruction Splitting for Sequential Decode

#### 4.4.2. デコード

D ステージから DT ステージにてデコードが行われる。デコードにおいては、主にアウト・オブ・オーダ・リソースの割り当てと、後述するリザベーション・ステーションへのディスパッチが行われる。

デコーダは GID、CSE、物理レジスタの割り当て、および VFP、VSP の割り当てをする。デコーダには前段のプリデコーダから 1 サイクルあたり最大 4 つの μOP 命令が送り込まれる。この入力された μOP 命令の組をディスパッチ・グループと呼び、この単位でリソースが割り当てられる。主な命令とリソースの割り当て数との関係を Table 4-1 に示す。

Table 4-1 Relation Between Instructions and Quantities of Allocated Resources

Resource	Allocation Unit	Instruction Type
GID	Dispatch group	All instructions
CSE	μOP instruction	All instructions
Physical register	μOP instruction	Instructions that have destination registers
VFP/VSP	Processing unit for load/store unit	Load/Store instructions

Table 4-1 に示すように、GID はディスパッチ・グループ単位で割り当てられる。一方で、CSE の 4 エントリあたり GID は 1 つという制約がある。ディスパッチ・グループが 4 μOP 命令未満であった場合、CSE には未使用エントリがある状態で GID が割り当てられる。

FP、SP の割り当て数はロード／ストア・ユニットでの処理単位である。詳細は 7.3 章にて説明する。

アウト・オブ・オーダ・リソースの割り当てが完了すると、μOP 命令はリザベーション・ステーションへディスパッチされる。

## 4.5. 命令コミット

コミット・ステージでは投機的に実行された命令の確定が行われる。分岐命令の分岐パス検証や例外の有無検証などを行い、命令を完了させてよいと判定すればプログラム・オーダでコミットを行い、プロセッサのアーキテクチャ・ステートを確定する。Figure 4-7 に示すように、CSE は 4 エントリ単位でグループ化され、グループ単位で GID が割り当てられている。CSE への μOP 命令の割り当ては先述のように GID 単位で行われることから、CSE の解放、すなわち命令コミットも GID 単位で実施される。

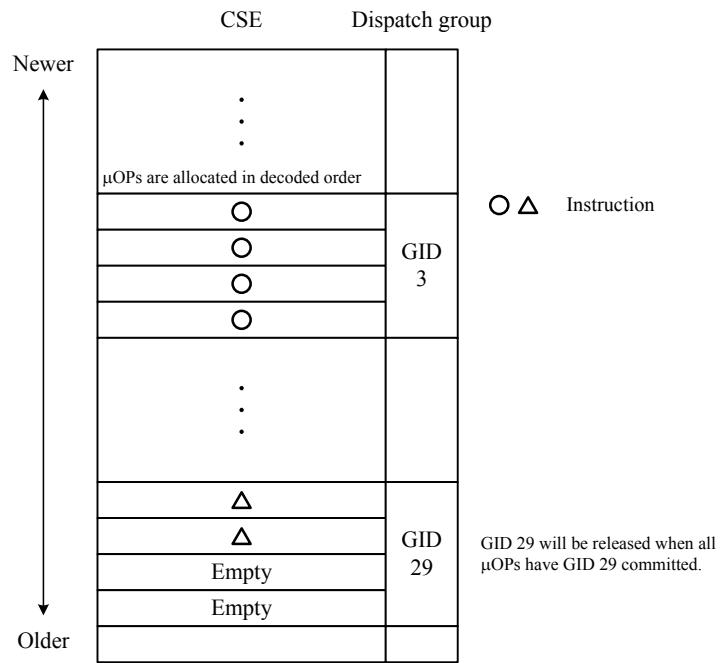


Figure 4-7 CSE Structure

命令コミットはプログラム・オーダでなされるため、CSE 内の最も古い命令がある CSE グループから行われる。命令コミットの動作は以下の通りである。

- 同時にコミットできる GID は 1 つ。すなわち、CSE グループをまたいでのコミットはできない。
- 同一 GID が割り当てられた複数個の μOP 命令は同時にコミットできる。
- コミットできる GID に属する中で最も古い μOP 命令は、同一 GID 内のより新しい μOP 命令の実行終了を待たずにコミットできる。
- 複数個の Taken 分岐命令は同時にコミットできない。
- 分岐命令が予測ミスしていたときは、その分岐命令までコミットし後続の命令を破棄する。
- 1 つのアーキテクチャ命令が複数個の μOP 命令に分解された場合、コミット自体は μOP 命令単位で可能である。ただし、PC は最後の μOP 命令がコミットされた時点で更新される。

#### 4.5.1. No Exception Mode

A64FX では、浮動小数点演算の例外の割り込みが通知されない設定のときに、コミットを可能な限り前倒しする機能を実装している。これによりパイプライン・ステージ全体を短くし、アウト・オブ・オーダのリソースの占有時間を短縮している。FPCR システムレジスタの設定が Table 4-2 のときに No Exception Mode となる。

Table 4-2 FPCR Register When No Exception Mode Is Enabled

FPCR Register Field	Field Value
FZ, FZ16	1
IDE, IXE, UFE, OFE, DZE, IOE	0

### 4.6. パイプライン・フラッシュ

コミット・ステージにおける命令完了判定の結果、命令の実行結果が正しくないと判断したときはパイプライン・フラッシュが発生する。パイプライン・フラッシュには以下の 2 種類がある。

#### 非同期フラッシュ

分岐予測ミス時に発生する。分岐予測ミスした場合、後続の命令は誤ったプログラム・パスの命令であることから命令コミットできない。したがって、後続命令は破棄される。分岐予測ミスは分岐命令が実行された時点、すなわち実行ステージで判明される。分岐予測ミスが判明すると、はじめにフロントエンドのフラッシュが行われ、正しいパスの命令フェッチが開始される。新しくフェッチされた命令はデコードされ、D ステージで待機する。次に、分岐ミスした命令とそれに先行する命令がすべてコミットしていることが確認されると、バックエンドのフラッシュが行われる。バックエンドのフラッシュが完了すると命令のディスパッチが再開される。

#### 同期フラッシュ

トラップ、例外発生、ロード／ストアの順序保証違反などによって発生する。これらの事象が発生すると後続命令は誤った実行結果になることから命令コミットできない。これらの事象は命令コミット時に判定されることから、命令コミットの時点でフロントエンドとバックエンドの両方のパイプラインがフラッシュされる。その後、プロセッサが正しい状態になれば命令のフェッチが再開される。同期フラッシュは、非同期フラッシュと異なってフロントエンドのフラッシュを事前に行えないため、命令フェッチとデコード処理を隠べできない。そのため、分岐予測ミスによる非同期フラッシュより命令の実行再開までに要するペナルティが大きい。

### 4.7. 特殊な命令制御

アーキテクチャ命令の中には、レジスタやメモリ以外のプロセッサのアーキテクチャ・ステイトを経由して依存が作られるものがある。これらの命令は先行命令が確実にコミットしている状態で実行されなければならない。また、その後続命令の実行は、先行命令が確実に完了している

ことが保証されていなければならない。これらの動作を保証するために、命令デコードと命令コミットには次の 2 つの特殊な命令制御がある。

### Pre-Sync

この制御の対象である命令は、直前の命令がコミットされるまでデコード・ステージに留められる。

### Post-Sync

この制御の対象である命令は、その命令がコミットされるまで後続命令をデコード・ステージに留める。

これらの命令制御は制御が必要なアーキテクチャ命令のみで行われる。制御対象の命令は命令属性／レイテンシ一覧に記載されている。

# 5. 命令ディスパッチ

---

デコードステージでは、アウト・オブ・オーダ・リソースへの割り当ての他にリザベーション・ステーション (RS) へのディスパッチ・スケジューリングを行う。A64FX には複数の RS があるが、各 RS に接続されている実行パイプラインの機能が異なるため、命令種と命令間の依存関係を考慮の上でスケジューリングされる。

## 5.1. リザベーション・ステーション

デコードされた  $\mu$ OP 命令はオペレーション・フローとしてリザベーション・ステーション (RS) にディスパッチされる。RS は実行可能、かつ実行パイプラインに割り当て可能な命令のうち最も古い命令から順にアウト・オブ・オーダで命令を発行する。A64FX の RS は 5 分割されており、それぞれに異なる実行パイプラインが接続されている。各 RS について各々のエントリ数、および接続される実行パイプラインを Table 5-1 に示す。

Table 5-1 Number of Entries and Connected Execution Pipelines of Each RS

RS	Number of Entries	Execution Pipeline
RSE0	20	EXA, FLA, PR
RSE1	20	EXB, FLB
RSA0	10	EAGA, EAGB
RSA1	10	
RSBR	19	BR

RSE0 / RSE1 は実行パイプラインの接続関係から互いの実行パイプラインに命令を発行できないが、RSA0 / RSA1 は互いの実行パイプラインに命令を発行できる。RS のエントリは CSE や FP / SP などのアウト・オブ・オーダ・リソースとは異なり、命令が発行されると解放される。割り当てと解放のステージについては 2.8 章の Table 2-6 に記述している。

各 RS は 2 つの書き込みポートと 2 つの発行ポートをそれぞれ持つ。そのため同一の RS にディスパッチできる命令は 2 命令に制限される。同様に同一の RS から発行できる命令は 2 命令までとなる。

## 5.2. 命令のディスパッチ属性

A64FX では RS と実行パイプラインの接続関係から、オペレーション・フローの種類によってディスパッチできる RS に制限がある。例えば、EXA パイプラインでしか実行できないオペレーションは RSE0 にしかディスパッチできない。このためオペレーション・フローのディスパッチ先はオペレーション・フローが実行可能なパイプラインと強い結びつきがある。本章でディスパッチ動作の説明のため、オペレーション・フローのディスパッチ可能な RS を属性として

Table 5-2 のように定義する。なお、アーキテクチャ命令とオペレーション・フローとその実行パイプラインの関連については命令属性／レイテンシー一覧に記載している。

**Table 5-2 Attributes of Instructions and Operation-Flows**

Attribute	Dispatch-Enabled RS	Destination Execution Pipeline
RSX	RSE0, RSE1, RSA0, RSA1	EXA, EXB, EAGA, EAGB
RSE	RSE0, RSE1	EXA, EXB, FLA, FLB
RSA	RSA0, RSA1	EAGA, EAGB
RSE0 only	RSE0	EXA, FLA, PR
RSE1 only	RSE1	EXB, FLB

また、ディスパッチに必要なリソースに RS のほかに Temporary Operand Register (TOR) が必要な命令がある。TOR はプログラム・カウンタから演算器にオペランドを中継するレジスタである。TOR が必要な命令を Table 5-3 に示す。

**Table 5-3 Instructions That Require TOR**

Attribute	Instruction
TOR	LDR{SW} (literal) ADR{P} BL{R} MRS

### 5.3. 命令の依存関係の検出

A64FX では 2.7 章で述べたように、異なる実行パイプライン間でオペランド・バイパスが行われる際にペナルティが生じる。特に整数演算命令は本来の演算レイテンシに対するペナルティの割合が大きいことから、命令間でオペランド依存があるときには可能な限り同じ実行パイプラインに発行することが望ましい。これを実現するため、A64FX はデコード時に命令間のオペランドの依存関係を検出している。デコーダは以下の条件がすべて成立したときに命令間に依存があると判断する。

- EXA, EXB, EAGA, EAGB パイプラインで実行される算術演算命令、論理演算命令、シフト命令である。
- 連続する前後 2 命令間のオペランドに依存関係がある、もしくは 2 命令ともに NZCV レジスタを使用している。
- 後続側の命令が RSX 属性、または RSE 属性である。

命令間の依存関係を検出すると、対象の命令は Dependence グループを形成する。依存関係の検出は常に前後 2 命令間で行われるため、Figure 5-1 に示すように連続する命令間で依存関係があるときに Dependence グループが形成される。また、Figure 5-2 に示すように、デコード・ウィンドウが異なる 2 命令間においてはデコード・ウィンドウのスロット 0 の命令と、1 つ前のデコード・ウィンドウのスロット 3 の命令に限って依存関係を検出して Dependence グループを形成できる。

Slot	Inst.	
0	A	Inst.B is depended on A.
1	B	
2	C	
3	D	Dependency group

Decoder can detect that Inst.B is depended on A.

Slot	Inst.	
0	A	
1	B	
2	C	Inst.C is depended on A.
3	D	

Decoder cannot detect that Inst.C is depended on A.

**Figure 5-1 Example of Two Instructions That Have Dependency in Same Decode Window**

Slot	Inst.	
0	A	
1	B	
2	C	
3	D	Inst.E is depended on D.
0	E	Dependency group

Decoder can detect that Inst.E is depended on D because D is slot 3.

Slot	Inst.	
0	A	
1	B	
2	C	
3	D	
0	E	Inst.E is depended on C.

Decoder cannot detect that Inst.E is depended on C because C is not slot 3.

**Figure 5-2 Example of Two Instructions That Have Dependency Across Different Decode Windows**

## 5.4. 命令ディスパッチ動作

$\mu$ OP 命令は、CSE、リネーミングレジスタ、VFP、VSP の割り当てが完了すると、オペレーション・フローとして、RS にディスパッチされる。このときフローの分割が行われる命令もある。RS へのディスパッチは、ディスパッチ属性や、Dependence グループ、RS の使用エントリ数などを考慮して割り振りが決定される。デコーダは、上述した命令のディスパッチ属性ごとに RS 割り振りルールを持っており、まず、そのルールを使って基本となる RS の割り当て先を決定する。それぞれの割り振りルールについてまとめる。

### RSX 属性命令

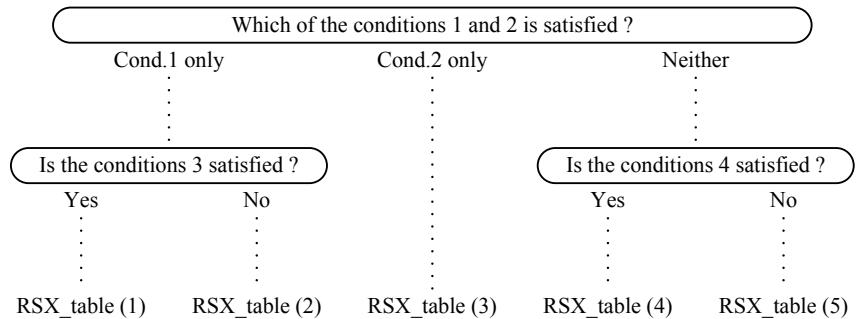
RSX 属性の命令は、RSE0/1、RSA0/1 のどの RS にもディスパッチできる。RS への振り分けは Table 5-4 に示すようにデコーダのスロットごとに割り当て先を決めるテーブルに基づいて行われる。このテーブルには、RS の使用数のバランスをとるために 5 つの割り振りパターンが用意されている。表中の RSEm/f は、RSE0/1 のうち空エントリ数が多いほうが RSEm、少ないほうが RSEf ということを意味している。RSA についても同様である。

**Table 5-4 Allocation Table for Instructions with RSX Attribute**

	Table 1	Table 2	Table 3	Table 4	Table 5
Slot 0	RSEm	RSEm	RSAm	RSEm	RSAm
Slot 1	RSEm	RSEf	RSAf	RSEf	RSAf
Slot 2	RSEm	RSEm	RSAm	RSAm	RSEm
Slot 3	RSEm	RSEf	RSAf	RSAf	RSEf

このテーブルは RS の空エントリを考慮して選択される。選択のルールは以下の条件と Figure 5-3 に示す条件の組み合わせで決まる。

- 条件 1 : RSA0 / 1 双方に空きが無く、RSE0 / 1 双方に空きがある。または、RSE0 / 1 の空きの合計数から RSA0 / 1 の空きの合計数を差し引いた数がしきい値以上。
- 条件 2 : RSE0 / 1 双方に空きが無く、RSA0 / 1 双方に空きがある。または、RSA0 / 1 の空きの合計数から RSE0 / 1 の空きの合計数を差し引いた数がしきい値以上。
- 条件 3 : RSE0 / 1 の空き数の差分がしきい値以上。
- 条件 4 : RSE0 / 1 のどちら一方が、RSBR を除く RS の中で最も空き数が多い。

**Figure 5-3 Allocation Table Selection Rule for Instructions with RSX Attribute**

### RSE 属性命令と RSA 属性命令

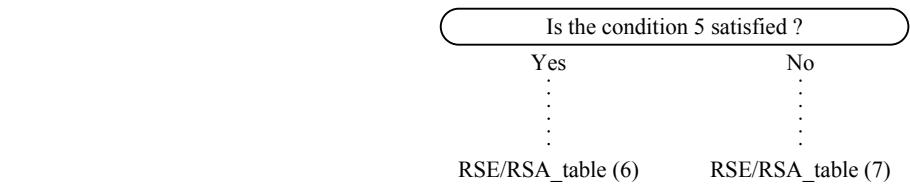
これらの属性の命令は、それぞれ RSE と RSA にしかディスパッチできない命令である。そのため、それぞれの RS の 0 番か 1 番に割り当て先を決定すればよい。Table 5-5 に割り振りテーブルを示す。

**Table 5-5 Allocation Table for Instructions with Either RSE or RSA Attribute**

	Table 6	Table 7
Slot 0	RS{E A}m	RS{E A}0
Slot 1		RS{E A}1
Slot 2		RS{E A}0
Slot 3		RS{E A}1

割り振りテーブルの選択は条件 5 と Figure 5-4 の組み合わせで選択される。

- 条件 5 : RSEm / RSAm に空きがあり、かつ RSEf / RSAf に空きがない。



**Figure 5-4 Allocation Table Selection Rule for Instructions with RSE or RSA Attribute**

### RSE0 only 属性命令と RSE1 only 属性命令

これらの属性の命令は割り当て可能な RS が 1 つしかないため、割り振りテーブルはなく、属性が示す RS にそのまま割り当てられる。

### Dependence グループの命令

Dependence グループを作る命令は、グループの先頭の命令のみ命令属性に基づいた割り当てが行われる。後続命令は先頭命令と同じ RS に暗黙に割り当てが決定される。

デコード・ステージの各スロットの命令は、個別に上記の割り振りルールに基づいて割り当て RS が決定される。そのため、デコード・ステージ全体では、3 命令以上の命令が同一の RS に割り当てられることがある。各 RS は 2 つの書き込みポートしか持っていないため、同一サイクルに 2 命令までしかディスパッチできない。このときは初めの 2 命令のみディスパッチを行い。残りの命令はディスパッチされない。改めて、次のサイクルでディスパッチされることになる。

また、TOR を使用する命令は、TOR に空エントリがなければならず、かつ、同一サイクルに 1 命令しかディスパッチできないという制約がある。

# 6. 命令実行

---

リザベーション・ステーション (RS) にディスパッチされたオペレーション・フローは、アウト・オブ・オーダにスケジューリングされ実行パイプラインに発行される。発行されたフローは実行パイプラインに実装されている演算器で実行される。命令実行とは広義ではロード／ストア命令のメモリ・アクセスまで含むが、本章では演算命令の演算を行うパイプライン、及び、ロード／ストア命令の実効アドレス演算パイプラインのアドレス演算ステージ部分を実行パイプラインとして取り扱う。

## 6.1. 命令発行

オペレーション・フローはソース・オペランドが利用可能になるまで RS にて待機する。ソース・オペランドが利用可能になったフローは実行可能状態に遷移する。RS は実行可能なフローの中からより古く、かつ、発行先のパイプラインが使用可能なものを選択し、発行する。これは RSA でスケジューリングされるロード／ストアのオペレーション・フローも同様である。オペレーション・フローの発行は基本的に RSE0、RSE1、RSA が独立に制御されるが、複数の実行パイプラインが連携してフローを実行する必要があるときは同期制御を行う。また、RSA0、RSA1 はそれぞれ、EAGA/EAGB パイプラインに投入できるため、常に互いに同期して制御している。

また、実行ステージでのオペレーション・フロー分割の対象のフローはこの時に分割される。すなわち、RS の 1 エントリから 2 つ以上のフローが発行される動作となる。

## 6.2. 実行パイプライン

実行パイプラインと主な演算器の組み合わせを Table 6-1 に示す。実行パイプラインは大きく 5 系統に分けられ、それぞれのパイプラインには演算器群である演算ユニットが配置されている。

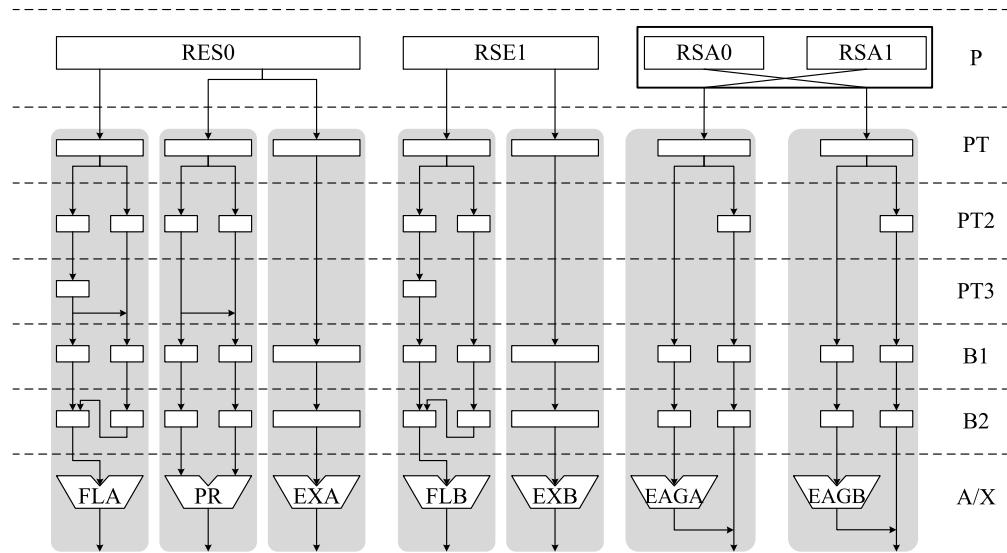
**Table 6-1 Execution Pipelines**

Pipeline Group	Pipeline	Function
Integer operation pipelines	EXA	Arithmetic & logic, shift, multiplication
	EXB	Arithmetic & logic, shift, division
Address calculation pipelines	EAGA	Address calculation, arithmetic & logic
	EAGB	
Floating-point operation pipelines	FLA	Integer arithmetic & logic, shift, floating-point arithmetic & multiply-add, floating-point division, crypto calculation, vector address calculation
	FLB	Integer arithmetic & logic, shift, floating-point arithmetic & multiply-add
Predicate operation pipeline	PR	Predicate manipulation
Branch pipeline	BR	

整数演算系パイプラインは主に整数命令の演算を担当する。アドレス演算系パイプラインには実効アドレス生成のための演算器と、部分的な機能を持った整数 ALU (Arithmetic Logic Unit) が実装されており、整数演算命令の一部を実行できる。浮動小数点演算系は SIMD & FP と SVE の演算命令を実行できる。

Figure 6-1 に示すように、各演算パイプラインはそれぞれリザベーション・ステーション (RS) に割り当てられている。基本的に各パイプラインは独立して動作するが、EXA パイプラインと PR パイプラインは RSE0 の発行ポートを共有することから同時に命令を投入できないという制約がある。

アドレス演算系のパイプラインは、ロード／ストア・パイプラインに接続されていて、ロード／ストア命令の実効アドレスの演算結果は、直接ロード／ストア・パイプラインに送られる。



**Figure 6-1 Outline of Execution Unit**

## 6.3. ブロックキング制御

いくつかのオペレーション・フローにはパイプライン処理できないものがある。このための制御をブロックキング呼ぶ。ブロックキング制御には、パイプライン・ブロックキングと演算ブロックキングがある。パイプライン・ブロックキングはあるパイプラインにおいて演算を実行しているとき、それが終了するまで後続の命令発行を受け付けることができない。演算ブロックキングは演算ブロックキングの属性を持つ命令を実行しているとき、それが終了するまで同じ演算ブロックキング属性の命令発行を受け付けない。パイプライン・ブロックキング中にはそのパイプラインにいかなる後続命令も発行できないのに対し、演算ブロックキング中に発行できない後続命令は演算ブロックキング型の命令のみという点で異なる。例えば、演算ブロックキング型である SDIV 命令は実行に 9~42 サイクルを要する。このとき、実行完了まで演算ブロックキング型の命令は発行できないが、他の命令は発行できる。命令ごとのブロックキング属性は命令属性／レイテンシ一覧表に記載している。

## 6.4. 物理レジスタ・ファイル

A64FX の物理レジスタ・ファイルは、アーキテクチャ・レジスタの種類ごとに分けられて実装されている。各物理レジスタ・ファイルの読み出しポート、書き込みポートと実行パイプラインの接続関係を Figure 6-2 に示す。

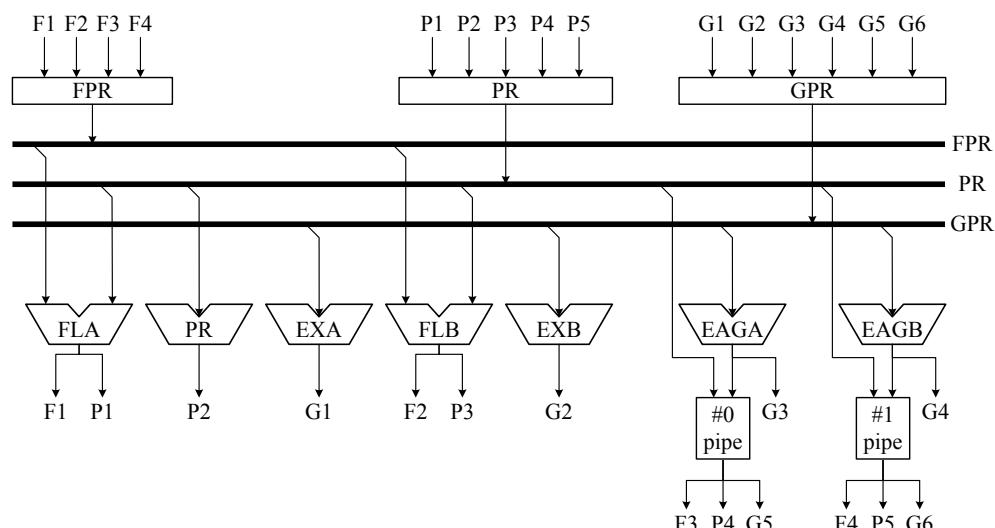


Figure 6-2 Connection Relationship Between Physical Register Files and Execution Pipelines

## 6.5. 特殊な命令制御

### 6.5.1. Merging predication

A64FX では、4.3 章の説明のように MOVPRFX 命令に修飾されたデストラクティブ命令は Pack 处理され、非デストラクティブ命令としてデコードされる。基本的に、Pack 处理された命令の μOP 命令分解数は被修飾命令単体の分解数となる。しかし、この修飾で Merging predication が指定されているときに限り、ベクトル・レジスタの要素を結合するための μOP 命令が 1 つ追加される。なお、MOVPRFX 命令に修飾されていないときは Merging predication であっても μOP 命令は追加されない。

### 6.5.2. レジスタ間転送

汎用レジスタ、浮動小数点レジスタ、Predicate レジスタを横断的に使用する命令は、それぞれのレジスタの値を転送するために特殊な制御が必要である。これは、各物理レジスタの読み出しポートと書き込みポートがすべての実行パイプライン系統に接続していないためである。そのため、複数のパイプライン系統を同期させて転送を処理している。レジスタの組み合わせと転送方向により制御が異なるため、例を用いて説明する。

#### 汎用レジスタから浮動小数点レジスタへの転送

FMOV (general) 命令、SCVTF 命令などにおいて、ソース・レジスタに汎用レジスタ、ディスティネーション・レジスタに SIMD&FP レジスタを指定したときの動作である。これらの命令は、汎用物理レジスタからソース・オペランドを読み出す EXA で実行されるオペレーション・フロー (EXA フロー) と、浮動小数点物理レジスタにオペランドを書き込む FLA で実行されるオペレーション・フロー (FLA フロー) に分割されて実行される。フロー実行のタイムチャートを Figure 6-3 に示す。

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
EXA flow	D	DT	P	PT	B1	B2	X												
FLA flow					P	PT	PT2	PT3	B1	B2	X1	X2	X3	X4	X5	X6	U C	UT W	W2

Figure 6-3 Flow Time Chart of Transfer Instruction from General-Purpose Register to Floating-Point Register

#### 浮動小数点レジスタから汎用レジスタへの転送

FMOV (general) 命令、FCVTZ\* 命令などにおいて、ソース・レジスタに SIMD&FP レジスタ、ディスティネーション・レジスタに汎用レジスタを指定したときの動作である。これらの命令は、ロード／ストア・パイプラインを使ってオペランドが転送される。FLA にて実行される浮動小数点物理レジスタからソース・オペランドを読み出すオペレーション・フロー (FLA フロー) と、ロード／ストア・パイプラインにて実行される汎用物理レジスタにオペランドを書き込む LD フローの 2 つに分割されて実行される。LD フローは 0 / 1 番のどちらのパイプラインでも実行できる。フロー実行のタイムチャートを Figure 6-4 に示す。なお、FLA フローと LD フローは実際は非同期であるため、LD フローはさらに後方のタイミングで実行される可能性がある。

	<b>1</b>	<b>2</b>	<b>3</b>	<b>4</b>	<b>5</b>	<b>6</b>	<b>7</b>	<b>8</b>	<b>9</b>	<b>10</b>	<b>11</b>	<b>12</b>	<b>13</b>	<b>14</b>	<b>15</b>	<b>16</b>	<b>17</b>	<b>18</b>	<b>19</b>	<b>20</b>	<b>21</b>	<b>22</b>	<b>23</b>
FLA flow	D	DT	P	PT	PT2	PT3	B1	B2	X	U	UT	UT2	UT3	UT4									
1 <sup>st</sup> LD flow			P	PT	B1	B2	A	T	M	B	R	RT											
2 <sup>nd</sup> LD flow														A	T	M	B	R	RT	RT2	C	W	W2

**Figure 6-4 Flow Time Chart of Transfer Instruction from Floating-Point Register to General-Purpose Register**

### 6.5.3. 非正規化数の演算

A64FX では浮動小数点表現の非正規化数の演算を特殊なプロセッサ・モードを用いてハードウェアで実行している。このため、非正規化数の演算におけるレイテンシ、スループットは正規化数の演算のそれとは全く異なる。一例として倍精度の FADD (scalar) 命令のレイテンシは約 90 サイクルである。このモードでは各演算処理が完全にイン・オーダかつブロッキング化されるため、演算のオペレーション・フロー数に比例して実行時間が増加する。

# 7. メモリ・アクセス

メモリ・アクセスはロード／ストア・パイプラインにて処理される。リザベーション・ステーションから発行されたロード／ストアのオペレーション・フローは、実効アドレスを計算されてロード／ストア・パイプラインに投入される。ロード／ストア・パイプラインは仮想アドレス変換を行ってL1Dキャッシュへアクセスし、ロードのオペレーション・フローならばデータを読み出し、ストアのオペレーション・フローならばデータを書き込む。また、ロード／ストア・パイプラインはキャッシング時の処理も取り扱う。

## 7.1. ロード／ストア・パイプラインの概要

ロード／ストア・パイプラインの主な構成モジュール、およびパイプライン・ステージを

Figure 7-1 に示す。

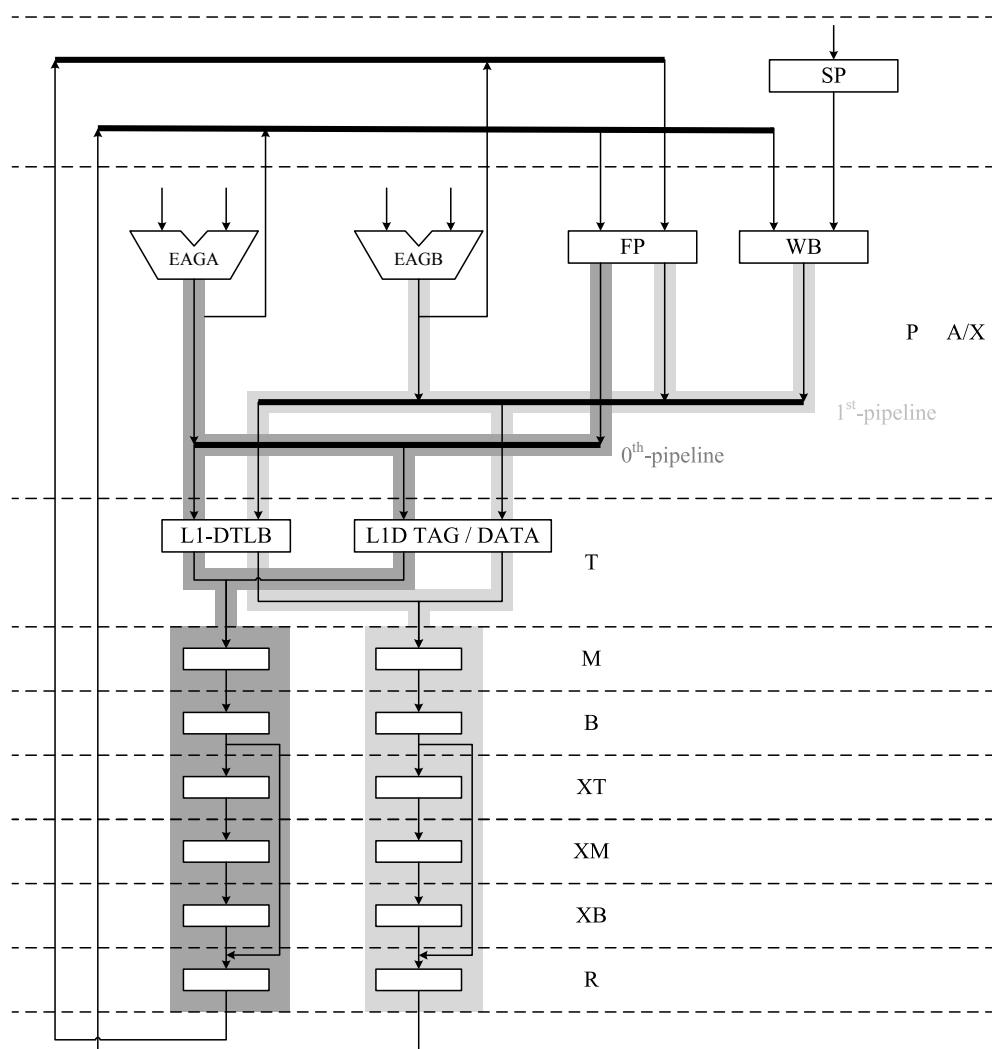


Figure 7-1 Outline of Load/Store Unit

図中の各モジュールの役割を以下にまとめる。

#### Fetch port (FP)

ロード／ストア命令の実行順序を管理するためのキューである。命令の順序のみを管理する Virtual Fetch Port (VFP) とロード／ストアのアクセス順序も管理する Real Fetch Port (RFP) が実装されている。VFP には、デコード時にロード／ストア命令がインオーダで割り当てられる。RFP には、ロード／ストアのオペレーション・フローが発行されるときに、アウト・オブ・オーダで割り当てられる。VFP, RFP ともに命令の実行が完了した時点で解放可能となり、イン・オーダで解放される。

#### Store port (SP)

ストア命令の実行順序とストア・データを管理するためのキューである。命令の順序のみを管理する Virtual Store Port (VSP) とロード／ストアのアクセス順序も管理する Real Store Port (RSP) が実装されている。VSP には、デコード時にストア命令がインオーダで割り当てられる。RSP には、ストアのオペレーション・フローが発行されるときに、アウト・オブ・オーダで割り当てられる。VSP, RSP ともにストア命令がコミットし、ストア・データが Write buffer に書き込まれると解放される。

#### Write buffer (WB)

SP にあるストア・データを L1D キャッシュに書き込む前に一旦保持するバッファである。ストア命令のコミット動作と L1D キャッシュへの書き込み動作を分離するために設けられている。

#### L1-DTLB

データ・アクセスに必要なアドレス変換情報を保持する 1 次 TLB である。

#### L2-DTLB

データ・アクセスに必要なアドレス変換情報を保持する 2 次 TLB である。パイプライン上には現れないが、ロード／ストア・ユニット内に実装されている。

#### L1D キャッシュ

L1D キャッシュのデータとタグ情報を保持する RAM である。

#### EAGA, EAGB

ロード／ストア命令の実効アドレスを生成する演算器である。演算ユニットに区分されるが、パイプラインステージ上では P ステージに位置する。

ロード／ストア・ユニットにおいては 2 本（0 番、1 番）のパイプラインが実装されている。2 本のパイプラインは基本的には同様の機能を有しているが、L1D キャッシュへの書き込みリクエスト処理については 1 番パイプラインのみが処理できる。

ロード／ストアのパイプラインはショートとロングの 2 つの動作モードを持っている。これらの動作モードは、基本的には命令によって決まっている。各命令の動作モードと、各ロード命令の load-to-use レイテンシを Table 7-1 に示す。ただし、2.9 章のレイテンシ切り替えで述べたように、オペレーション・フローが後方ステージで衝突する場合にはレイテンシ切り替えが発生する。

**Table 7-1 Latencies of Load/Store Instructions**

<b>Instruction</b>	<b>Functional Mode</b>	<b>Load-to-Use Latency</b>	<b>After Latency Change</b>
Integer load instruction	Short	5 cycles	8 cycles
Integer store instruction	Short	-	-
SIMD&FP load instruction	Short	8 cycles	11 cycles
SIMD&FP store instruction	Short	-	-
SVE load instruction, and part of SIMD&FP instructions	Long	11 cycles	-
SVE store instruction	Long	-	-
Predicate load instruction	Long	9 cycles	-
Predicate store instruction	Long	-	-

## 7.2. ロード／ストアの基本処理

ロード／ストア・パイプラインは、主にロード／ストア命令のメモリ・アクセスと L1D キャッシュのデータ・フィル／ライトバックを処理する。これらは以下の基本的なオペレーション・フローで処理される。

### LD フロー

ロード命令のメモリ・アクセスを行うフローである。仮想アドレス変換と L1D キャッシュへのアクセスのすべての操作を行う。

### ST0 フロー

ストア命令の仮想アドレス変換と、L1D キャッシュのタグアクセスを行うフローである。L1D キャッシュへのデータの書き込みは行わない。タグにアクセスすることで、L1D キャッシュのヒットチェックを行う。

### ST2 フロー

ストアデータを L1D キャッシュに書き込むフローである。1 番パイプラインでのみ実行できる。

### MI フロー

データ・フィルのための Move-In 操作を行うフローである。1 キャッシュラインの Move-In に 4 フローを必要とする。0/1 番の両パイプラインが同期して 2 フローずつ実行する。

### MO フロー

ライトバックのための Move-Out 操作を行うフローである。基本的に 1 キャッシュラインの Move-Out に 4 フローを必要とする。ただし、ライトバック対象のキャッシュラインが clean であるときは 2 フローになる。0/1 番 の両パイプラインが同期して 2 フローずつ、または 1 フローずつ実行する。

### 7.2.1. ロード命令

ロード命令の基本動作について説明する。

1. ロード命令が μOP 命令にデコードされて VFP のエントリに割り当てられる。ロード μOP 命令は 1 つの LD フローとして RSA にディスパッチされる。
2. LD フローが実行可能になると RSA から発行される。EAGA / EAGB にて実効アドレスが計算され、0 / 1 番パイプラインに投入される。
3. EAGA / EAGB から投入された LD フローは、調停スケジューラにて選択されるとペナルティなく 0 / 1 番パイプラインに投入される。同時に RFP のエントリに実効アドレスを書き込む。0 / 1 番パイプラインには EAGA / EAGB だけでなく RFP と WB も接続されており、それらのオペレーション・フローも実行する。どのオペレーション・フローを実行するかを調停スケジューラが決定する。
4. “3.”において、EAGA / EAGB から投入された LD フローがスケジューラに選択されなかったときは 0 / 1 番パイプラインに投入されない。LD フローは RFP に実行可能状態で待機する。
5. LD フローは 0 / 1 番パイプラインにて L1-DTLB でアドレス変換を行いつつ、L1D キャッシュのタグとデータを読み出す。L1-DTLB と L1D キャッシュにヒットすれば、所定のパイプライン・ステージに沿ってデータを読み出し、レジスタにデータを書き込む。LD フローの実行は完了する。
6. “5.”にて L1-DTLB にヒットしないときは、L2-DTLB と Translation Table を段階的に検索し、仮想アドレス変換情報を取得する。L1D キャッシュミスのときは下位のキャッシュ階層からデータを取得しデータ・フィルを行う。このとき、MI フローと MO フローが実行される。
7. L1-DTLB、または L1D キャッシュ・ミスが解決すると、再度 RFP から LD フローが 0 / 1 番パイプラインに投入されて実行される。
8. LD フローが実行完了するとロード μOP 命令はコミット可能となる。VFP、RFP のエントリはコミットを待たず、先行のロードが完了していれば解放される。

LD フローの処理は L1-DTLB と L1D キャッシュにヒットする限り、基本的に 1 フローで完了する。しかし、キャッシュ・ミスが発生するとパイプラインへの再投入が必要になるため、LD フローは複数回実行される。また、後述の Multiple Structures 命令や Gather / Scatter 命令はアドレスパターンにより、キャッシュにヒットするときでも複数のフローに分割され実行される。

### 7.2.2. ストア命令

ストア命令の基本動作を説明する。

1. ストア命令は μOP 命令にデコードされ、VFP と VSP の各エントリに割り当てられる。ストア μOP 命令は ST0 フローとデータ転送フローに分けられてディスパッチされる。ST0 フローは RSA にディスパッチされる。データ転送フローはストア命令特有のフローで、ストアデータをレジスタから RSP に転送するためのものある。データ転送フローは RSE0 にディスパッチされる。
2. データ転送フローは実行可能になると RSE0 から発行される。このデータ転送フローの実行は ST0 フローとは同期であり、どちらが先に実行されるかは決まっていない。

3. ST0 フローが実行可能になると RSA から発行される。EAGA / EAGB にて実効アドレスが計算され、0 / 1 番パイプラインに投入される。
4. EAGA / EAGB から投入された ST0 フローは、調停スケジューラにて選択されると 0 / 1 番パイプラインに投入される。同時に RFP と RSP のエントリに実効アドレスを書き込む。
5. “4”において ST0 フローが調停スケジューラに選択されなかったときは、0 / 1 番パイプラインに投入されずに ST0 フローは RFP に実行可能状態で待機する。
6. ST0 フローは 0 / 1 番パイプラインにて L1-DTLB でアドレス変換を行いつつ、L1D キャッシュのタグ読み出しのみを行う。L1-DTLB にヒットすれば、アドレス変換後の物理アドレスを SP に保存する。L1D キャッシュにヒットしないときは、下位のキャッシュ階層にデータ要求のリクエストを起動する。どちらの場合でも ST0 フローの実行は完了する。
7. “6”において L1-DTLB にヒットしないときは、L2-DTLB と Translation Table を段階的に検索し、仮想アドレス変換情報を取得する。取得後に ST0 フローを再投入する。
8. “6”において起動されたデータ要求のリクエストは、ST0 フローの実行とは独立にキャッシュ・ミスの処理を行う。このとき、MI フローと MO フローが実行されてデータ・ファイルが行われる。
9. ST0 フローとデータ転送フローの実行が完了すると、ストア μOP 命令はコミット可能となる。ストア μOP 命令がコミットされると、RSP からストアの物理アドレスとデータが WB に移される。
10. WB から ST2 フローが起動されて 1 番パイプラインに投入される。L1D キャッシュにヒットするとデータ書き込みを行い ST2 フローの実行が完了する。
11. “10”において L1D キャッシュ・ミスが発生したときは、再度キャッシュ・ミスの処理を行う。キャッシュ・ミスが解決されると ST2 フローを再投入してデータ書き込みを行う。

ストア命令は ST0 フローではデータ書き込みが行われず、命令のコミット後に ST2 フローで書き込みを行う。さらに、ST0 フローと ST2 フローのそれぞれで L1D キャッシュミスが発生する可能性があることに注意が必要である。

## 7.3. Fetch Port / Store Port

### 7.3.1. Virtual Fetch Port / Virtual Store Port

A64FX では本来の機能を持つ Fetch Port / Store Port に加えて、Virtual Fetch Port (VFP) / Virtual Store Port (VSP) が実装されている。VFP / VSP はロード／ストア命令のプログラム・オーダのみを管理する。Fetch Port / Store Port はそれに加え、ロード／ストアのアドレスを保持してメモリ・アクセスの実行順序も管理する。本書では Virtual Fetch Port / Virtual Store Port と区別するため、本来の Fetch Port / Store Port を Real Fetch Port (RFP) / Real Store Port (RSP) と記述する。VFP / VSP と RFP / RSP の関係を Figure 7-2 に示す。RFP / RSP には VFP / VSP の一部のウィンドウがマップされている。RFP / RSP へのマップは、ロード／ストアのオペレーション・フローがリザベーション・ステーションから発行されるときに行われる。RFP / RSP のエントリ数以上はマップできないため、その時は命令発行が制限される。VFP / VSP はデコード時に割り当てられるのに対し、RFP / RSP は命令実行時にマッピングされていなければよいため、命令のデコード時に必要

な RFP / RSP を削減できる。これによって RFP / RSP のエントリ不足に起因するデコード時のストールを回避している。

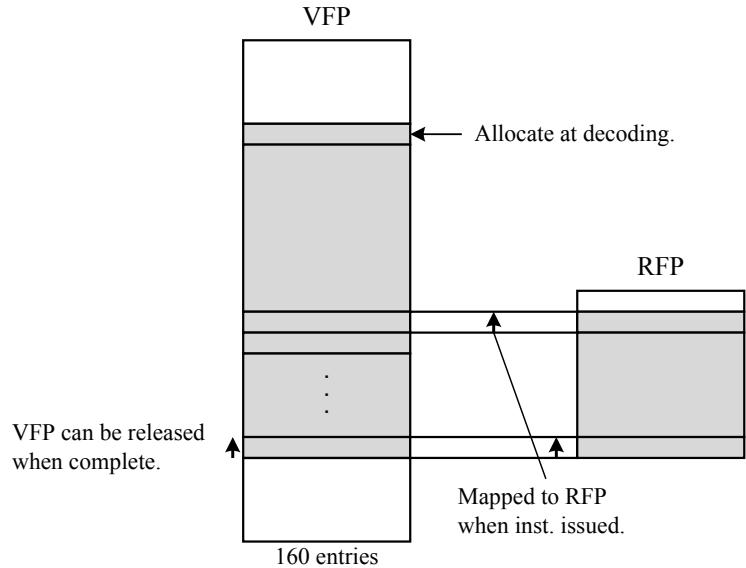


Figure 7-2 Relationship Between VFP/VSP and RFP/RSP

### 7.3.2. Fetch Port / Store Port の割り当て数

FP / SP の割り当て数は命令種によって決まっている。基本的にロード命令は 1 μOP 命令に対して FP の 1 エントリが割り当てられる。ストア命令は 1 μOP 命令に対して FP と SP のそれぞれ 1 エントリが割り当てられる。一方で、SVE の Gather / Scatter 命令及び SVE の LD[234][BH] / ST[234][BH] 命令は 1 μOP 命令に対して複数の FP / SP のエントリが割り当てられる。各命令の FP/SP の割り当て数は命令属性／レイテンシ一覧表に記載している。

## 7.4. Write Buffer

A64FX ではコミットしたストア命令のデータは WB に移され、その後スケジューラによって任意のタイミングで L1D キャッシュに書き込まれる。これは、命令コミットとキャッシュへの書き込みという 2 つの操作を分離することで、コミットのストールを削減するねらいがある。WB は基本的に 1 エントリあたり 64 バイトであり、全 8 エントリから構成される。L1D キャッシュへ書き込む直前のデータを保持することから、SP とは異なり、アドレスのアライメントに関する制約がある。WB に保持されるデータは、そのサイズに応じて適切なアドレス・アライメントをとる必要がある。アライメント制約に起因して、WB にデータを書き込む際にデータ・マージ、またはデータ・スプリットと呼ばれる操作が発生する場合がある。

データ・マージ、データ・スプリットの一例を Figure 7-3 に示す。WB が 64 バイト長の書き込みデータを保持するとき、アライメントは 64 バイトである必要がある。ストア・データのアドレスが 64 バイト・アライメントでない場合には、データは 64 バイト境界でスプリットされてから WB に書き込まれる。反対に、SP からストア・データを書き込む時点で先行するエントリに空きがある場合には、そのエントリに対して部分的にデータを書き込んだ上でマージができる。マージは複数回行えるため、データ長が小さい命令は L1D キャッシュへの書き込み回

数が相対的に少なくなることがある。なお、WB マージはメモリ・オーダリングの制約内で行うため、上記条件を満たしても行われない場合もある。

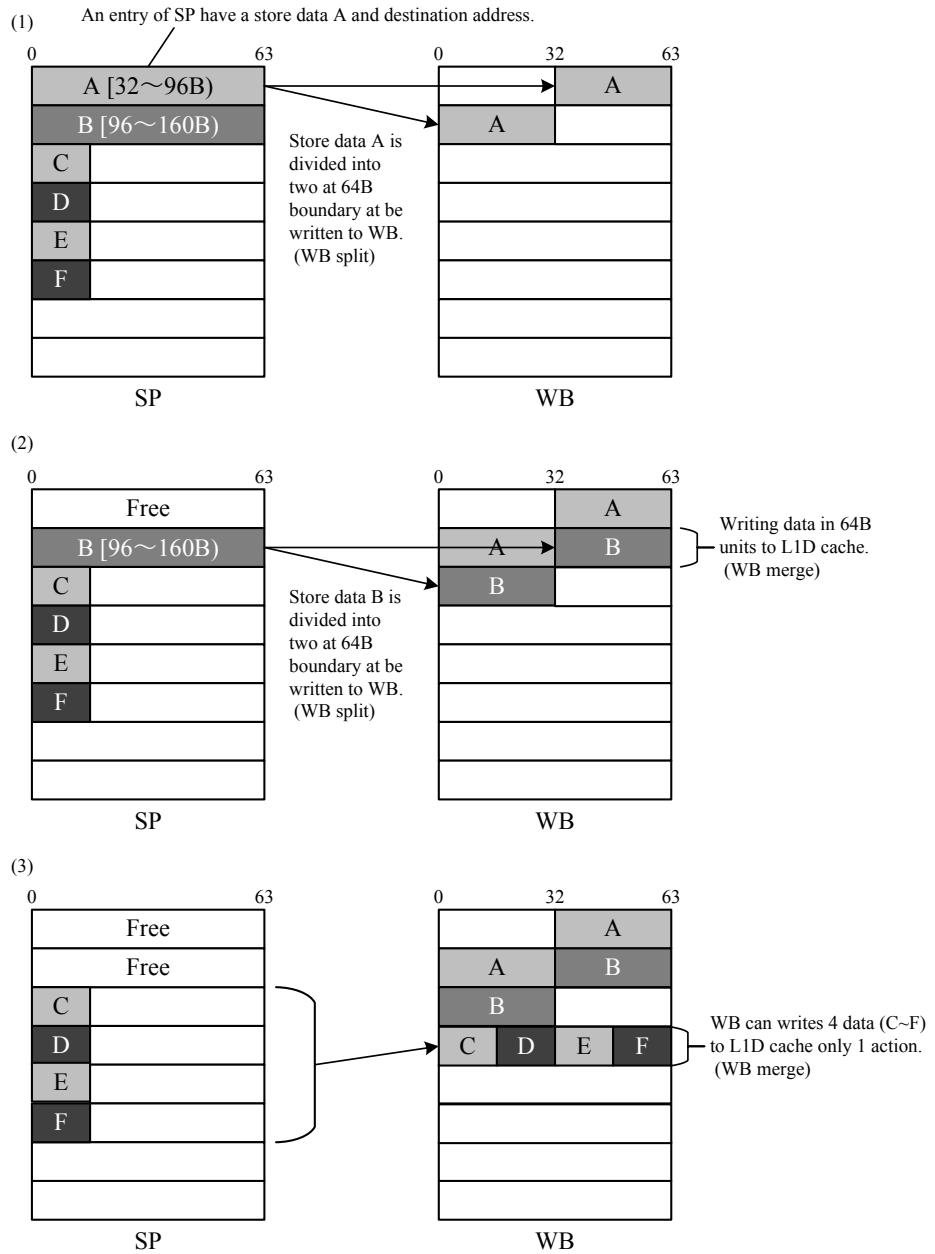


Figure 7-3 Store Data Write from SP to WB

WB のエントリが管理するデータ長とマージ機能の動作は命令によって異なる。関係を Table 7-2 に示す。

**Table 7-2 Data Length and Merge Function Availability for Each Instruction Managed by WB Entry**

	Instruction	Data Length	Merging
Integer instruction	STLR*, STNP, STP, STR*, STTR*, STUR*	16 bytes	✓
SIMD&FP instruction	STNP, STP, STR, STUR	16 bytes	✓
	ST[1234] (single structure)	16 bytes	✓
	ST1 (multiple structures) – {1D 2D 2S 4H 8B}	16 bytes	✓
SVE instruction	ST1 (multiple structures) – {4S 8H 16B}	64 bytes	✓
	ST[234] (multiple structures)	64 bytes	✓
	ST1[BHWD] (contiguous)	64 bytes	✓
	ST1[BHWD] (scatter)	64 bytes	✓
	ST[234][BH]	64 bytes	✓
	ST[234][WD]	128 bytes	
	STR (vector), STR (predicate)	64 bytes	

## 7.5. ロード／ストアのアウト・オブ・オーダ実行

A64FX ではロード／ストア命令を基本的にはアウト・オブ・オーダで実行している。一方で、メモリ依存のあるストア命令とロード命令は依存関係を検出した上で正しい順序で実行しなければならない。メモリ依存のあるストア命令とロード命令は基本的に以下の 2 つの動作で順序保証をしている。

### Store Fetch Interlock (SFI)

先行するストア命令の L1D キャッシュ書き込み、すなわち ST2 フローの完了まで後続のロード命令の実行を待たせる動作である。先行ストア命令の物理アドレスが確定しているときは、後続のロード命令の LD フローは 0/1 番パイプラインに投入される。先行ストアのアドレスとの一致を検出すると実行をキャンセルし、LD フローは RFP でストアの完了まで待機する。

### パイプライン・フラッシュ

先行ストア命令の物理アドレスが未確定のときの動作である。先行ストア命令の物理アドレスが未確定の状態では後続のロード命令とのメモリ依存を検出できないため、後続のロード命令の LD フローは投機的に実行される。その後、先行ストア命令の ST0 フローの実行時に実行済みのロード命令のアドレスと比較する。これにより、メモリ依存が検出されるとストア命令のコミット時にパイプライン・フラッシュを行い、ロード命令は再実行される。

### 7.5.1. Store Fetch Bypass

Store Fetch Interlock (SFI) は上述のように、ストア命令が完全に完了するまで、後続のロード命令の実行を止めてしまうため、性能への影響が大きい。このペナルティを緩和するために Store Fetch Bypass (SFB) が実装されている。SFB は先行ストア命令データを、後続のロード命令

が SP または WB から読み出す機能である。これにより、ロード命令はストア命令の完了を待たずに実行することができる。ただし、SFB はハードウェアの実装制約からすべてのロード命令とストア命令の組み合わせで実行できるわけではない。SFB が可能な組み合わせを Table 7-3 と Table 7-4 に示す。また、前述のように WB はアドレスのアライメントに制約があり、かつ 2 エントリをまたいでのバイパス・データの読み出しができないことから、SFB を実行させるには、バイパス対象のデータが 1 エントリに収まっている必要がある。

**Table 7-3 SFB Availability for Each Combination of Load and Store Instructions**

Load Instruction \ Store Instruction	LD (1)	LD (2-1)	LD (2-2)	LD (4-1)	LD (4-2)	LD (8-1)	LD (8-2)	LD (16)	LD (32)	LD (64)
ST (1)	✓									
ST (2-1)	✓	✓								
ST (2-2)			✓							
ST (4-1)	✓	✓		✓						
ST (4-2)					✓					
ST (8-1)	✓	✓		✓		✓				
ST (8-2)							✓			
ST (16)								✓		
ST (32)									✓	
ST (64)										✓

**Table 7-4 Specific Instructions of Each Group Shown in SFB Availability Table**

Group Name	Instruction	Group Name	Instruction
LD (1)	Length = 1 byte LDR*B (general) LDTR*B (general) LDR (SIMD&FP) – B LDUR (SIMD&FP) – B	ST (1)	Length = 1 byte STRB (general) STR (SIMD&FP) – B STUR (SIMD&FP) – B
LD (2-1)	Length = 2 bytes LDR*H (general) LDTR*H (general) LDR (SIMD&FP) – H LDUR (SIMD&FP) – H	ST (2-1)	Length = 2 bytes STRH (general) STR (SIMD&FP) – H STUR (SIMD&FP) – H
LD (2-2)	Length = 2 bytes LDR (predicate) : VL = 128-bit	ST (2-2)	Length = 2 bytes STR (predicate) : VL = 128-bit
LD (4-1)	Length = 4 bytes LDR (general) – W LDTR (general) – W LDR (SIMD&FP) – S LDUR (SIMD&FP) – S	ST (4-1)	Length = 4 bytes STR (general) – W STR (SIMD&FP) – S STUR (SIMD&FP) – S
LD (4-2)	Length = 4 bytes LDR (predicate) : VL = 256-bit	ST (4-2)	Length = 4 bytes STR (predicate) : VL = 256-bit
LD (8-1)	Length = 8 bytes LDR (general) – X LDTR (general) – X LDR (SIMD&FP) – D LDUR (SIMD&FP) – D LD1 (multiple structure, 1 register) - {8B 4H 2S 1D}	ST (8-1)	Length = 8 bytes STR (general) – X STR (SIMD&FP) – D STUR (SIMD&FP) – D ST1 (multiple structure, 1 register) - {8B 4H 2S 1D}
LD (8-2)	Length = 8 bytes LDR (predicate) : VL = 512-bit	ST (8-2)	Length = 8 bytes STR (predicate) : VL = 512-bit
LD (16)	Length = 16 bytes LDR (vector) : VL = 128-bit	ST (16)	Length = 16 bytes STR (vector) : VL = 128-bit
LD (32)	Length = 32 bytes LDR (vector) : VL = 256-bit	ST (32)	Length = 32 bytes STR (vector) : VL = 256-bit
LD (64)	Length = 64 bytes LDR (vector) : VL = 512-bit	ST (64)	Length = 64 bytes STR (vector) : VL = 512-bit

### 7.5.2. アウト・オブ・オーダ実行の制約

ロード／ストア命令はロード／ストアの対象アドレスが異なっていれば、基本的にはアウト・オブ・オーダで実行しても構わない。しかしながら、ハードウェアの実装制約からアドレス一致の検出は理想的にはなっていない。そのため、いくつかの条件において疑似的にメモリ依存を検出してしまい、アウト・オブ・オーダ実行の制限が発生する。以下にその条件をまとめると。

- Predicate 修飾のロード／ストア命令の `inactive` 要素における制約

メモリ依存の検出はロード／ストア命令のすべての要素が `active` と扱われる。そのため本来 `inactive` な要素のアドレス間で疑似メモリ依存が発生する。例外的に、ストア命令の要素がすべて `inactive` の時にはストア操作そのものが省略されるため、疑似メモリ依存は発生しない。

- Gather load 命令における制約  
Gather load 命令の 2 ペアの要素のオペレーション・フローが分割されずに実行されるときは、その 2 ペアが含まれるキャッシュ・ラインの全体がメモリ依存の検出単位となる。本来の 2 要素のアクセス以外の範囲で疑似メモリ依存が発生する。
- memsize が 4 バイト未満のロード／ストア命令における制約  
SIMD&FP の Vector ロード／ストア命令と SVE のロード／ストア命令では、メモリ依存の検出は 4 バイト境界単位で行われる。アクセスの先頭アドレスと終端アドレスがそれぞれ、4 バイト境界になるように拡張される。そのため、その拡張された部分で疑似メモリ依存が発生する。
- Multiple Structures 命令における制約  
7.8.1 章での説明の通り、Multiple Structures 命令のメモリ・アクセスはレジスタ単位で行われる。これらのメモリ・アクセスの空間は memsize と Element 数、レジスタ数の積として取り扱われる。さらに、メモリ依存の検出がキャッシュ・ラインとなる。アクセス空間に含まれるキャッシュ・ラインがすべて検出対象となる。そのため、本来のアクセス範囲以外で疑似メモリ依存が発生する。また、SVE の LD[234] / ST[234]においては、常に Vector Length が 512-bit であるとして扱われる。
- 4KiB 境界をまたぐロード／ストア命令における制約  
少なくとも memsize の単位でアライメントされているロード／ストア命令が 4KiB 境界をまたいでアクセスするときは、メモリ依存が完全な物理アドレスで検出されない。そのため物理アドレスが異なるときでも、疑似メモリ依存が発生する場合がある。
- インフライト中のロード／ストア命令における制約  
ロード／ストア・パイプラインをインフライト中のロード／ストア命令間のメモリ依存は完全な物理アドレスで行われない。そのため物理アドレスが異なるときでも、疑似メモリ依存が発生する場合がある。この制約はオペレーション・フローが FP, WB に待機しているときには発生しない。
- ストア命令の LID キャッシュ・ミスにおける制約  
ストア命令が LID キャッシュ・ミスをすると、その後続のロード命令はメモリ依存の検出が完全な物理アドレスでは行われない。そのため物理アドレスが異なるときでも、疑似メモリ依存が発生する場合がある。この制約はストア命令のキャッシュ・ミスが解決すると解消される。

## 7.6. パイプライン競合

ロード／ストア・パイプラインの 0 番パイプラインと 1 番パイプラインは機能に差がある。そのため、それぞれのパイプラインで任意のオペレーション・フローは実行できず、実行可能なパイプラインに制約があるフローがある。また、フローの組み合わせによっては 0 番パイプラインと 1 番パイプラインに同時に投入できないものがある。各オペレーション・フローのパイプラインの使用条件は以下の通りである。

- MI フロー、MO フロー  
これらのフローは必ず 2 フローをペアとして 0 番パイプラインと 1 番パイプラインにて同時に実行される。すなわち、他のフローと一緒に実行されることはない。

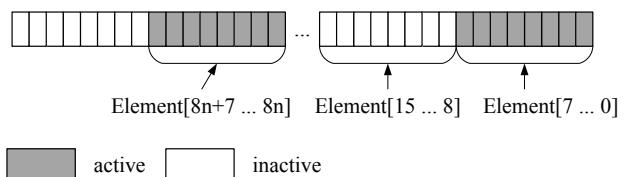
- LD フロー  
0番パイプラインと1番パイプラインの両方で実行できる。ただし、ST2 フローとは同時に実行できない。
- ST0 フロー  
0番パイプラインと1番パイプラインの両方で実行できる。一部のストア命令の ST0 フローは ST2 フローと一緒に実行できる。
- ST2 フロー  
1番パイプラインのみでしか実行できない。このとき、1番パイプラインは一部のストア命令の ST0 フローしか同時に実行できない。

ST2 フローと一緒に実行できるオペレーション・フローは ST0 フローのみである。ただし、実装制約などからすべてのストア命令の ST0 フローが実行できるわけではない。ST2 フローと一緒に実行可能なストア命令とその条件を Table 7-5 に示す。

**Table 7-5 ST0 Flow Conditions**

	Instruction	Condition
Integer instruction	ST{T U R} – X STP{N P} – X	
SIMD&FP instruction	ST{U R} – [DQ] ST{N P} – [DQ] ST1 (single structure) – D ST1 (multiple structures) ST[234] (single structure) – D	Addresses are at least 8-byte aligned.
SVE instruction	ST[1234]D (contiguous) STR (vector) STR(predicate) : VL = 512-bit	
	ST1B (contiguous)	Addresses are at least 8-byte aligned and all of Element[8n] – Element[8n+7] are either active or inactive. Figure 7-4 shows an example.
	ST1H (contiguous)	Addresses are at least 8-byte aligned and all of Element[4n] – Element[4n+3] are either active or inactive.
	ST1W (contiguous)	Addresses are at least 8-byte aligned and both Element[2n] and Element[2n+1] are either active or inactive.
	ST1[BHW] (scatter)	When Element[n] is inactive, the ST0 flow corresponding to this element can be executed simultaneously.
	ST1D (scatter)	When the address in Element[n] is at least 8-byte aligned or its element is inactive, the ST0 flow corresponding to this element can be executed simultaneously.

Z register image with size .B



**Figure 7-4 Example of active/inactive in ST1B (Contiguous)**

## 7.7. キャッシュ・ライン・クロス

ロード命令の中には、少なくとも memsize と同じアドレス・アライメントが保証されているときであってもメモリ・アクセスの範囲が 2 つのキャッシュ・ラインにまたがるときがある。そのようなメモリ・アクセスをキャッシュ・ライン・クロス（ライン・クロス）と呼ぶ。

ライン・クロスにおいては、両方のキャッシュ・ラインがキャッシュ・ヒットする限りペナルティは発生しない。このとき、LD フローは 1 フローで完了する。一方で、キャッシュ・ミスが発生したときは、キャッシュ・ミスとなったキャッシュ・ラインが片方のみの場合であっても、その LD フロー全体がキャッシュ・ミスとして扱われる。また、両方のキャッシュ・ラインでキャッシュ・ミスが発生したときは、2 つのキャッシュ・ミスの処理を同時に起動できる。

なお、ライン・クロスはロード命令のみに発生する。ストア命令の場合、WB にて書き込みのアドレスがアライメントされるためである。

## 7.8. 特殊なロード／ストア命令の動作

### 7.8.1. Multiple Structures 命令

SIMD&FP の LD[234] (multiple structures) / ST[234] (multiple structures)、および SVE の LD[234][BHWD] / ST[234][BHWD] 命令の動作について説明する。なお、命令の仕様上の表記としては LD1/ST1 (multiple structure) も Multiple structures 命令に属するが、命令の動作の違いから本節での説明の対象にならない。

#### デコード・ステージ分解と実行ステージ分解

Multiple structures 命令は 1 命令で複数のデスティネーション・レジスタを持つ命令である。これらの命令はデコード・ステージにて複数の μOP 命令にデコードされる。基本的にはデスティネーション・レジスタ数分の μOP 命令に分解される。アドレッシング・モードによっては、さらにアドレス生成のための補助的な μOP 命令が追加されることもある。この補助 μOP 命令はアドレス演算のみを行い、メモリ・アクセスは行わないためロード／ストア・ユニットには送られない。

ロード／ストア・ユニットに送られる μOP 命令はメモリ・アクセスのための μOP 命令のみである。基本的にはデスティネーション・レジスタ数と同一であるが、LD[234][BH] / ST[234][BH] 命令に限っては実行ステージにおいてさらに 4 分割される。各命令のロード／ストア・ユニットに送られるのに必要なフロー数を Table 7-6 に示す。このフロー数は実質 FP / SP の割り当て数に等しい。それぞれの分解数は命令属性／レイテンシ一覧表に記載している。

**Table 7-6 Required Number of Flows for  $\mu$ OP Instructions Split from Architecture Instruction to Send to Load/Store**

Architecture Instruction	Required Number of Flows
LD2 (multiple structures) LD2[WD] ST2 (multiple structures) ST2[WD]	2
LD3 (multiple structures) LD3[WD] ST3 (multiple structures) ST3[WD]	3
LD4 (multiple structures) LD4[WD] ST4 (multiple structures) ST4[WD]	4
LD2[BH]	8
LD3[BH]	12
LD4[BH]	16

### ロード／ストア・ステージ分解

LD[234][WD] / ST[234][WD] 命令では、ロード／ストア・ユニットに送られたメモリ・アクセスのフローは、アクセス・アドレスのパターンに応じてさらに分解される。LD[234][WD] / ST[234][WD] 命令のメモリ・アクセスはデスティネーション・レジスタ単位で行われる。つまり、1 フローあたりのメモリ・アクセスの空間はレジスタのベクトル・データ長より広く、さらに L1D キャッシュの読み出し幅より広くなることがある。L1D キャッシュの読み出し幅は 128 バイト、かつ 128 バイト・アライメントである。128 バイト境界をまたぐフローは逐次に分割され実行される。このとき、それぞれのフローは先行フローの完了を待たなければならず、かつ、先行フローの完了から後続フローのパイプライン投入まで少なくとも 5 サイクルのペナルティが発生する。

一例として LD3D (multiple structures) 命令のフロー分割イメージを Figure 7-5 に示す。

ld3d {z3.d, z4.d, z5.d}, p3/z, [x10, #0 mul v1]

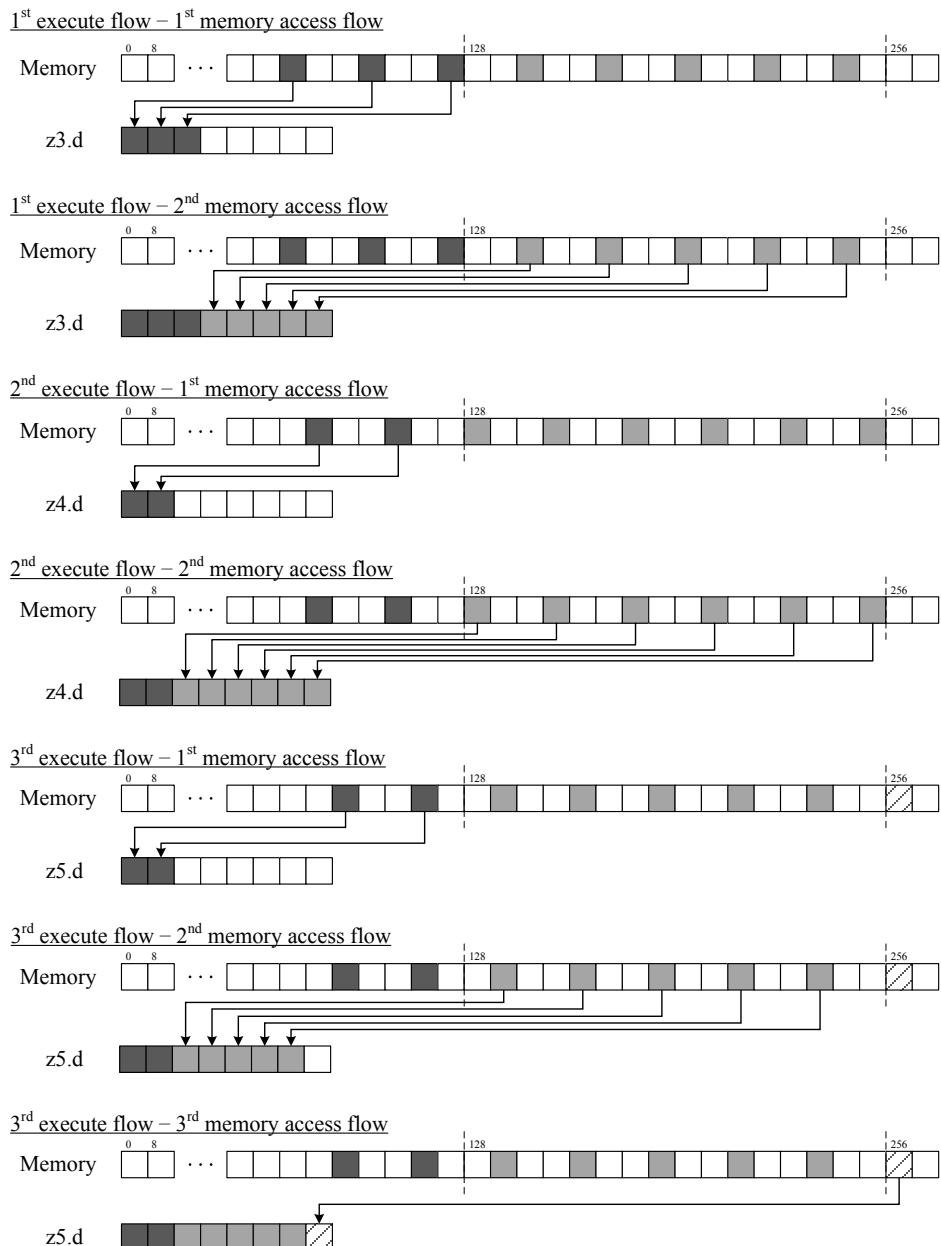


Figure 7-5 Illustration of Splitting LD3D (multiple structures) Instruction Flow

## 7.8.2. Gather load / Scatter store 命令

SVE の Gather load 命令 (以降、Gather 命令) と Scatter store 命令 (以降、Scatter 命令) は 1 命令で複数の離散したアドレスに対してメモリ・アクセスする命令である。実効アドレスのソース・オペランドの一方がベクトル・レジスタ上にあることから、実効アドレスは浮動小数点演算系パイプラインで計算される。そのため、整数のロード／ストア命令や SVE の Contiguous ロード／ストア命令とは異なるハードウェア動作となる。

## デコード

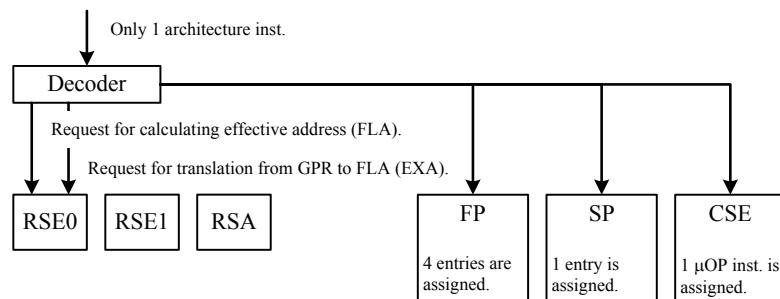
Gather / Scatter 命令は通常のロード／ストア命令と異なり、1 μOP 命令に対して複数の FP / SP のエントリが割り当てられる。これは Element ごとにメモリ・アクセスが独立していることから、ロード／ストア・ユニットの処理も Element ごとに独立するためである。Gather 命令においては、ロード命令にも関わらず SP のエントリが割り当てられることに注意が必要である。このように特殊な FP / SP の割り当てをすることから、デコードは同一サイクルに Gather / Scatter 命令のいずれか 1 命令のみしかデコードできないという制約がある。その他の命令もデコードできない点で、シーケンシャル・デコードより強い制約となる。Gather / Scatter 命令に属する各命令の μOP 数と FP / SP の割り当て数との関係を Table 7-7 に示す。

**Table 7-7 Number of μOP Instructions and Number of Allocated FP/SP Entries for Each Gather/Scatter Instruction**

	Number of μOP Instructions	FP	SP
LD1[BHW] (Gather) - S	1	8	1
LD1[BHWD] (Gather) - D	1	4	1
ST1[BHW] (Scatter) - S	8	16	16
ST1[BHWD] (Scatter) - D	4	8	8

Gather 命令においては、1 アーキテクチャ命令は 1 μOP 命令にデコードされるが、FP / SP は Figure 7-6 に示すように複数のエントリが割り当てられる。Gather μOP 命令は、実効アドレスを計算する FLA パイプラインに接続する RSE0 にディスパッチされる。アドレッシングが “scalar plus vector” のときは、汎用レジスタからのベースアドレス・オペランド転送リクエストも RSE0 にディスパッチされる。Gather 命令におけるリクエストのイメージを Figure 7-6 に示す。

ld1d z1.d, p3/z, [x1, z2.d]



**Figure 7-6 Requests of Gather Instruction**

Scatter 命令においては、1 アーキテクチャ命令は Element 数に応じて 4 μOP 命令、ないしは 8 μOP 命令にデコードされる。FP / SP は Element 数と同数のエントリが割り当てられる。Scatter 命令は Gather 命令とは異なり、複数の実効アドレス演算のためのリクエストが RSE0 にディスパッチされる。さらに、ベクトル・レジスタからストアデータを SP に転送するリクエストも RSE0 にリクエストされる。Scatter 命令のリクエストのイメージを Figure 7-7 に示す。

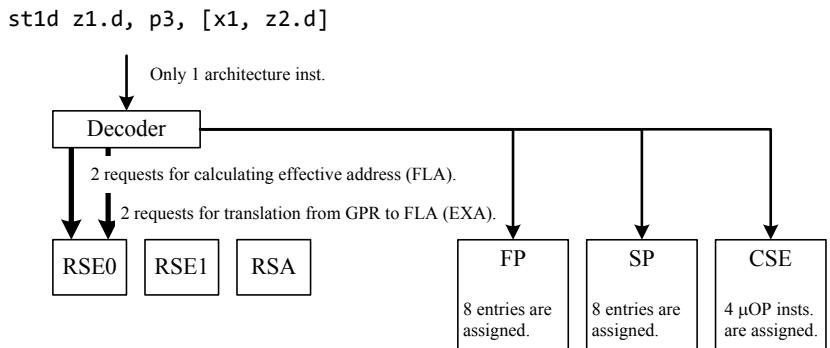


Figure 7-7 Requests of Scatter Instruction

なお、アドレッシングが “vector plus immediate” のときは即値がデコーダから直接 FLA に渡されるため、整数レジスタからベース・オペランドを転送するリクエストは省略される。

## 実効アドレス演算

Gather / Scatter 命令の実効アドレス演算の概要を Figure 7-8 に示す。実効アドレスの生成は通常のロード／ストア命令のそれとは異なり、ベクトル演算器で計算される。ただし、実効アドレスを計算できるのは FLA パイプラインのみである。アドレッシングが “scalar plus vector” のときは整数レジスタから FLA 演算器にベース・オペランドを転送する必要がある。この転送は整数レジスタから EXA パイプラインを経由して行われる。FLA 演算器で計算された実効アドレスは FP と SP に分けられて一時保存される。このとき、FP の書き込みポートにおける競合のために、他の命令の EAGA / EAGB パイプラインへの発行は制限される。

Gather 命令では実効アドレスを FP と SP に分けて保存する。それに対して Scatter 命令では、各 Element の実効アドレスは個別に FP / SP のエントリに保存される。また、Scatter 命令ではベクトル・レジスタからストアデータを転送する必要があり、このリクエストも FLA パイプラインを使って処理される。

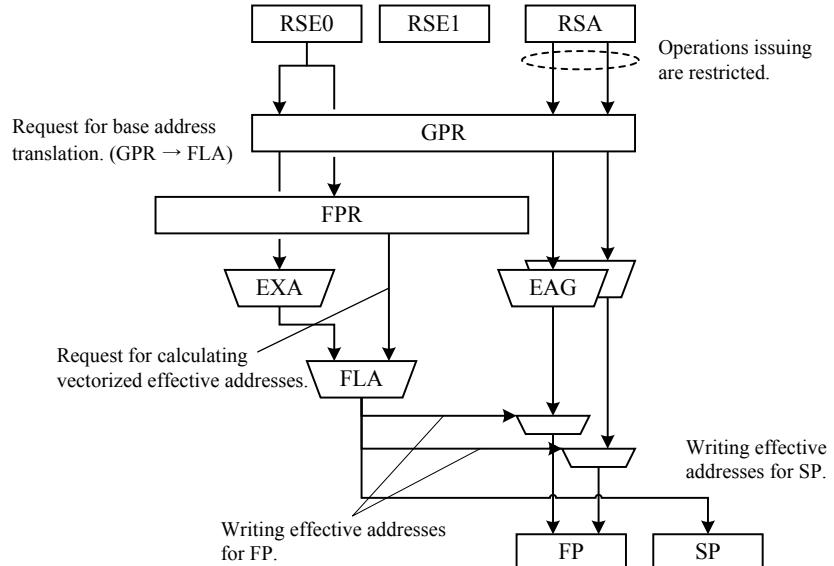


Figure 7-8 Effective Address Generation for Gather Instruction

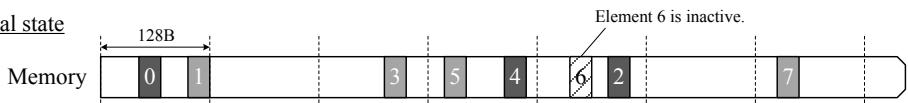
## メモリ・アクセス

Gather / Scatter 命令は各 Element が個別のメモリ・アドレスを指すため、基本的に各 Element は個別のメモリ・アクセスとして扱われる。特に Scatter 命令はオーダリングに制約があるため、各 Element は完全に個別のストアのフローとして分解され処理される。つまり、8 Elements 時には 8 つの個別ストアに、16 Elements 時には 16 個の個別ストアとして処理される。個々のフローや操作は通常ストア命令の基本処理と同様である。

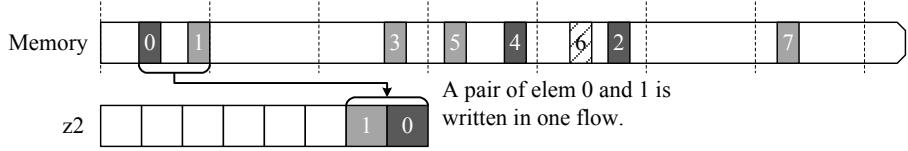
一方で、Gather 命令はフロー数削減の観点から分解は 2 段階で行われる。まず、ベクトルの Element において先頭から 2 ペアずつに分割される。さらに 2 ペアはアクセスするアドレス空間が同一の 128 バイト境界の 128 バイト空間に収まっているときに分割される。反対に同一空間に収まっている場合は分割されない。Gather 命令のメモリ・アクセスの遷移を Figure 7-9 に示す。フローの分割が行われた場合は、Multiple Structures 命令と同様に後続フローの実行は先行フローの完了を待たなければならず、かつ、後続フローの実行までに少なくとも 5 サイクルのペナルティが発生する。

`ld1d z2.d, p3/z, [x0, z1.d]`

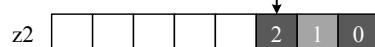
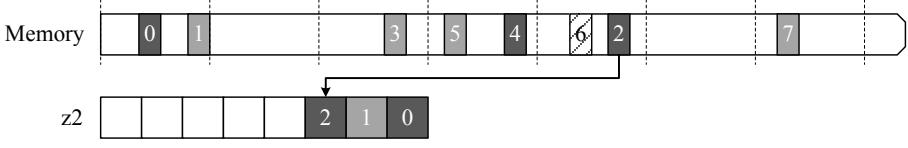
Initial state



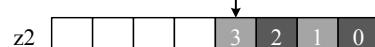
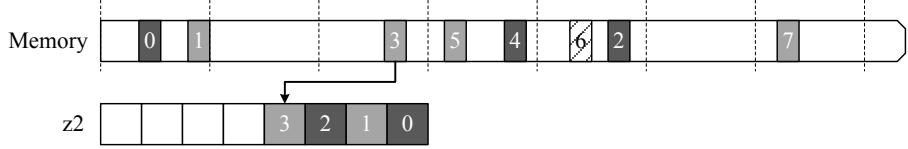
1<sup>st</sup> flow



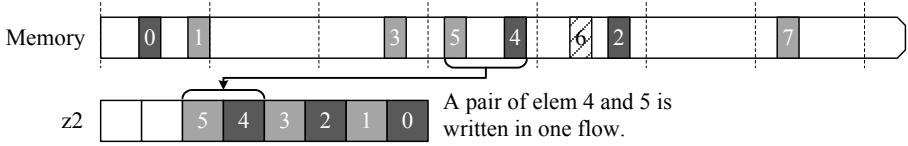
2<sup>nd</sup> flow



3<sup>rd</sup> flow



4<sup>th</sup> flow



5<sup>th</sup> flow

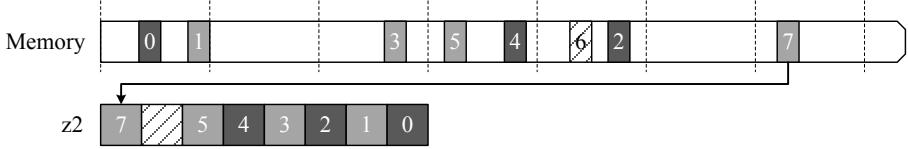


Figure 7-9 Summary of Elements for Gather Instruction

また、これらの2ペアの両方のElementがInactiveの場合はメモリ・アクセスのフローそのものが削除される。A64FXでは、Gather命令の2ペアずつの実行の機能をCombined Gather機能と称している。

# 8. アドレス変換機構

## 8.1. Translation Lookaside Buffer (TLB)

A64FX の TLB 構成を Table 8-1 に示す。TLB は命令用とデータ用が実装されており、それぞれが L1-TLB と L2-TLB の 2 階層になっており。L1-TLB はフル・アソシアティブ構造で、入れ替えアルゴリズムには FIFO 方式を採用する。L2-TLB は 4-way のセット・アソシアティブ構造で、入れ替えアルゴリズムには LRU 方式を採用する。

また A64FX の TLB は Contiguous bit をサポートする。Contiguous bit がセットされている Page は 1 エントリで変換情報を保存している。

Table 8-1 TLB Specifications

		For Instruction	For Data
L1	Association method	Full associative	Full associative
	Number of entries	16 entries	16 entries
	Replacement algorithm	FIFO	FIFO
L2	Association method	4-way set associative	4-way set associative
	Number of entries	1,024 entries	1,024 entries
	Replacement algorithm	LRU	LRU

## 8.2. Translation table cache

Translation table は多段のツリー構造をとり、Block / Page descriptor を得るために複数回のメモリ・アクセスを要する。このメモリ・アクセス時間を見短縮することを目的として Table descriptor を一時的に保存する Translation table cache が実装されている。Translation table cache は処理時間の短縮を目的とする点で TLB と類似するが、TLB が Table walk そのものの発生を抑止することを目的としているのに対して、Translation table cache は Table walk 中のメモリ・アクセスによって発生するレイテンシを削減することを目的としているという違いがある。

Translation table cache は Table 8-2 に示すようにフル・アソシアティブ構造をとるバッファである。エントリ数は 16 で、各エントリにて Table descriptor を保持する。なお、Table cache に保存される Table descriptor は 2 ステージ変換のうち Stage-1 のものだけであり、Stage-2 のものは保存されない。

Table 8-2 Table Cache Specifications

Translation table cache	Association method	Full associative
	Number of entries	16 entries
	Replacement algorithm	LRU

# 9. キャッシュ・アーキテクチャ

A64FX はオンチップで 2 階層構成のキャッシュを持つ。L1 キャッシュはプロセッサ・コア単位で実装されており、命令用とデータ用の 2 種類がある。L2 キャッシュは CMG 単位で実装されており、命令とデータで共有する。各キャッシュ間のデータはハードウェアによってコヒーレンスが保証される。

## 9.1. キャッシュ、メモリ階層の構造

Figure 9-1 に示すように、L2 キャッシュとメモリ階層は 4 つの CMG から構成される。CMG 間は ccNUMA(cache coherent NUMA) アーキテクチャを採用している。メモリは CMG 内の L2 キャッシュとのみ接続され、CMG ごとに物理アドレス空間は分割されている。L2 キャッシュからの Read / Write リクエストは、MAC (Memory Access Controller) を介してメモリに送られる。L2 キャッシュと MAC との間には Move In Buffer (MIB) と呼ばれるバッファがあり、MAC へのインフライトのリクエストを管理する。

Figure 9-1 に示すように、CMG 間は L2 キャッシュ階層にてリング構造で接続される。L2 キャッシュ間はハードウェアによりコヒーレンスが保証される。L2 キャッシュ同士は方向の違う 2 重のバスで接続されている。

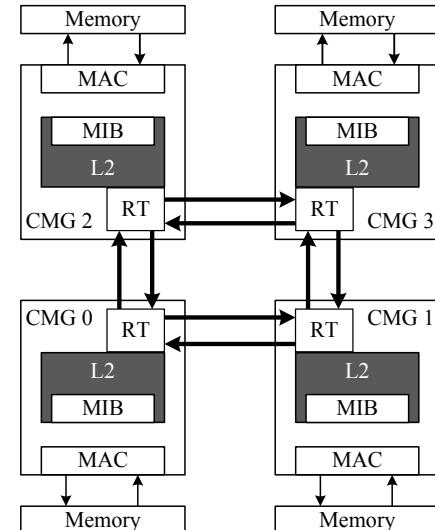


Figure 9-1 L2 Caches and Memory Levels

Figure 9-2 に示すように、LII / L1D キャッシュはプロセッサ・コアごとに実装され、CMG 内の L2 キャッシュと一対一で接続される。CMG 内の LII / L1D キャッシュは同じ CMG の L2 キャッシュを共有し、LII / L1D キャッシュのデータは L2 キャッシュに包含される。L1D キャッシュと L2 キャッシュは上下方向が独立したバスで接続されている。L1D キャッシュと L2 キャッシュの間には MIB に加えて MOB (Move Out Buffer) がある。L1D キャッシュは Move-In のリクエストと Move-Out のリクエストを非同期に管理する構造をとっており、別々のキューが実装され

ている。Table 9-1 に主なバスの帯域をまとめる。バスの接続先により実装単位が異なることに注意が必要である。

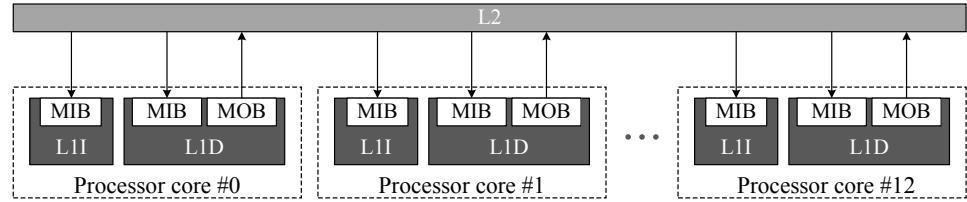


Figure 9-2 Connection Between L1 and L2 Caches

Table 9-1 Bus Throughput

	Direction	Bus Throughput
L1D	L2 to L1D	64 bytes / cycle (per Core)
	L1D to L2	32 bytes / cycle (per Core)
L2	L2 to L1D	512 bytes / cycle (per CMG)
	L1D to L2	256 bytes / cycle (per CMG)
L2	L2 to L2	64 bytes / cycle (per Ring)
L2	Memory to L2	128 bytes / cycle (per CMG)
	L2 to Memory	64 bytes / cycle (per CMG)

## 9.2. キャッシュの基本構成

### 9.2.1. L1 キャッシュの構成

L1 キャッシュの主な構成を Table 9-2 に示す。L1D キャッシュは命令の種類によってアクセス・レイテンシが 5 サイクルから 11 サイクルまで変化する。L1D キャッシュは同時に 2 つのロード、または 1 つのストアを受け付ける。

**Table 9-2 L1 Cache Specifications**

		<b>For Instruction</b>	<b>For Data</b>
L1 cache	Association method	4-way set associative	4-way set associative
	Capacity	64 KiB	64 KiB
	Hit latency (load-to-use)	4 cycles	5 cycles(integer)
			8 cycles (SIMD&FP / SVE in short mode)
			11 cycles (SIMD&FP / SVE in long mode)
	Line size	256 bytes	256 bytes
	Write method	---	Writeback
	Index tag	Virtual index and physical tag (VIPT)	Virtual index and physical tag (VIPT)
Index formula		index_A = (A mod 16,384) / 256	index_A = (A mod 16,384) / 256
Protocol		SI state	MESI state

ページ・サイズの選択によっては L1 キャッシュでシノニム問題が発生する。L1 キャッシュは容量 64 KiB の 4-way セット・アソシティブ方式であることから、そのインデックスは 16 KiB 空間をとる。このとき、ページとして 4 KiB page を選択しているとアドレスの bits[13:12]においてシノニムが起こりうる。A64FX はハードウェアによってシノニムを回避するようになっている。

### 9.2.2. L2 キャッシュの構成

L2 キャッシュの構成を Table 9-3 に示す。L2 キャッシュはプロセッサ・コアとキャッシュとの位置関係によってアクセス・レイテンシが異なるため、キャッシュ・ヒット時のレイテンシは 37~47 サイクルとなる。L2 キャッシュはプロセス間のインデックス競合を緩和するためにインデックスをハッシュしている。また、2 バンク構成になっており、物理アドレスのうち bit[8]でインターリーブしている。

**Table 9-3 L2 Cache Specifications**

		<b>For instruction and data (by shared)</b>
L2 cache (shared by instruction & data)	Association method	16-way set associative
	Capacity	8 MiB
	Hit latency (load-to-use)	37 to 47 cycles
	Line size	256 bytes
	Write method	Writeback
	Index and tag	Physical index and physical tag (PIPT)
	Index formula	$\text{index } <10:0>$ $= ( (\text{PA} <36:34> \text{xor PA} <32:30> \text{xor PA} <31:29> \text{xor PA} <27:25> \text{xor PA} <23:21>) << 8 ) \text{xor PA} <18:8>$
	Protocol	MESI state

### 9.3. キャッシュ・コヒーレンス・プロトコル

A64FX では各キャッシュ間のコヒーレンスはハードウェアによって保証される。一貫性のプロトコルには一般的な MESI プロトコルを採用している。MESI プロトコルの各ステートと、そのステートになりうる主な要因を Table 9-4 に示す。

**Table 9-4 Details of MESI Protocol**

<b>Condition</b>		<b>State</b>	<b>Possible Cause of State</b>
M	Modified	Data has been modified from main memory values (Dirty). Other caches at the same level do not have the data.	Data filling due to a store demand request. Stored in a cache line in the E/S state.
E	Exclusive	Data matches main memory values (Clean). Other caches at the same level do not have the data.	Data filling due to a load demand request while other caches do not have the data. Data filling due to prefetch access with a predefined type attribute while other caches do not have the data.
S	Shared	Data matches main memory values (Clean). Other caches at the same level also have the data.	Load demand request in the E state, or data fill request due to prefetch access with the Read attribute.
I	Invalid	A cache line is invalid.	Other caches request data when the data in the E/M state. Data writeback.

### 9.4. Move-In / Move-Out

あるキャッシュ階層へのデータ・ファイル／ライトバックは、キャッシュに対する Move-In, Move-Out と呼ばれる操作で行われる。Move-In, Move-Out 操作の定義を以下にまとめます。

#### Move-In

そのキャッシュ階層に対してデータ書き込み、タグの書き込み、ステートの状態を更新し、データのコヒーレンスとメモリ・オーダリングの整合性を確定する操作である。

### Move-Out

そのキャッシュ階層において、データの読み出し、タグのステート無効化を行い、データのコピーレンスとメモリ・オーダリングの整合性を確定する操作である。

データ・フィル／ライトバックは Move-In / Move-Out 操作を組み合わせて行われる。あるキャッシュ階層における基本的なキャッシュ・ミスの処理は以下の通りである。

1. 上位階層からリクエストを受け取り、自キャッシュ階層に該当アドレスのデータがあるか否かをチェックする。
2. キャッシュ・ミスが確定すると、受けたリクエストを MIB に登録し、さらに下位のキャッシュ、メモリ階層に伝達する。
3. 自キャッシュ階層にデータ・フィルするための空きラインが無い場合、古いデータを Move-Out する。
4. 下位のキャッシュ、メモリ階層からデータの応答があると自キャッシュ階層に Move-In する。
5. Move-In したデータを読み出し、リクエスト要求元である上位階層にデータを応答する。

L1 キャッシュ、L2 キャッシュともに複数のキャッシュ・ミスの処理をインフライトできる。インフライト中の Move-In / Move-Out を管理するための資源が、それぞれ Move-In Buffer (MIB) と Move-Out Buffer (MOB) である。各キャッシュ階層の資源数を Table 9-5 に示す。

Table 9-5 Quantity of Queue Resources at Each Cache Level

	Queue Type	Number of Entries
L1I cache	MIB	3 entries / core
	MOB	---
L1D cache	MIB	12 entries / core
	MOB	4 entries / core
L2 cache	MIB	256 entries / CMG
	Store Lock Register	244 entries / CMG

L2 キャッシュにおいては Move-Out データは直接 MAC へ送出されるため、MOB は存在しない。一方で、読み書きの整合性を保証するために Move-Out 操作中であることを示す Store Lock Register が実装されている。このレジスタのエントリ数がライトバックにおけるインフライト数の上限を決定している。

## 9.5. Move-In Bypass

前節で述べたように、キャッシュ・ミスした対象データを上位階層のキャッシュに応答できるのは Move-In が完了した後である。しかし、Move-In 完了までデータ応答を待つと、その分だけメモリ・アクセスに時間を要する。これを短縮するために、A64FX では Move-In の完了を待たずに上位のキャッシュ階層にデータを応答する Move-In Bypass 機能を実装している。

Move-In Bypass はメモリ・アクセス時間を短縮できるが、全てのキャッシュ・ミスのリクエストに対して行われるわけではない。特に L1D キャッシュについてはハードウェアの実装制約か

ら Move-In Bypass が可能な命令が限定される。L1D キャッシュにおいて Move-In Bypass ができる命令を Table 9-6 に示す。

Table 9-6 Instructions That Can Execute Move-In Bypass on L1D Cache

Instruction Classification	Instruction
Integer instruction	LDR{B H SB SH SW }
	LDTR{B H SH SW }
	LDA{P X R{B H }, LDLAR{B H }}
SIMD&FP instruction	LDR - {B H S D}
	LDUR - {B H S D}
	LD1 (single structure) - {B H S D}
	LD1R - {B H S D}

## 9.6. Zero fill

ARMv8 命令セットにはキャッシュ・メンテナンス命令として DC ZVA 命令が定義されている。この命令は、命令で指示した仮想アドレスを含むブロックに対してゼロデータを書き込む。A64FX では、DC ZVA 命令は L2 キャッシュ階層に対してゼロデータ書き込みを行う。DCZID\_EL0 システム・レジスタが示すブロックサイズはキャッシュ・ライン・サイズと同じである。Figure 9-3 に示すように、プロセッサ・コアで DC ZVA 命令が実行されるとロード／ストア・ユニットを経由して L2 キャッシュにゼロデータ書き込みのリクエストが送られる。L2 キャッシュは DC ZVA リクエストを受け取ると、指定のアドレスに対応するキャッシュ・ラインを確保してゼロデータ書き込みをする。

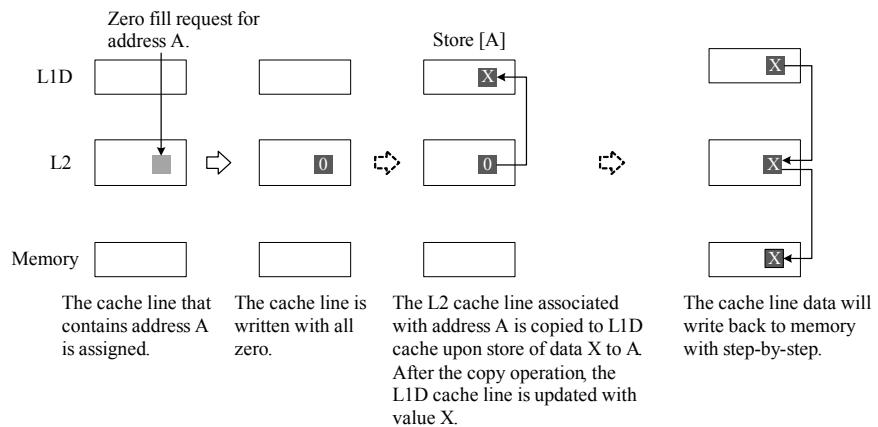


Figure 9-3 Basic Zero Fill Process

L2 キャッシュに指定アドレスのデータが存在しない場合であってもキャッシュ・ミスとはならず、メモリからのデータ・フィル操作は行われない。また、L1D キャッシュに指定されたアドレスのデータが存在するときは Figure 9-4 に示すように L1D キャッシュのデータが L2 キャッシュにライトバックされた後にゼロデータ書き込みが行われる。

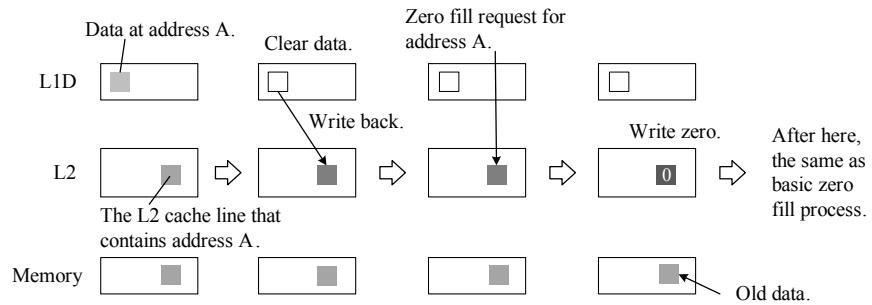


Figure 9-4 Zero Fill Process When L1D Cache Contains Data

このように、DC ZVA 命令は一般的なストア命令とは異なり、メモリから L2 キャッシュへのデータ・フィル操作を省略する。これによってメモリ空間への書き込み時におけるメモリ帯域の消費を抑え、実効メモリ帯域を改善することができる。

# 10. Memory Access Controller

---

## 10.1. Memory Access Controller の構成

Memory Access Controller (MAC) は、メイン・メモリの読み書きを行うユニットである。メイン・メモリには第 2 世代の High Bandwidth Memory (HBM2) を採用している。MAC は各 CMG に 1 つずつ実装され、それぞれが HBM2 の 1 チップと P2P で接続される。

A64FX の MAC がサポートする HBM2 の諸元を Table 10-1 に示す。

Table 10-1 Specifications of HBM2 Supported by A64FX

	Specification
Memory standard	HBM Gen2
Memory capacity	8 GiB (8Gib x8 stacks) / 1MAC
Data rate	2 Gbps

MAC は HBM2 の規格に準拠しつつ最大限のスループットを得るために、アクセス順序を制御するためのスケジューラを持つ。スケジューラのリソース数を Table 10-2 に示す。

Table 10-2 Quantity of Scheduler Resources for HBM2

	Quantity of Resources
Scheduler queue size	244 entries / MAC

## 10.2. メモリ・アクセス性能

A64FX の基本的なメモリ・アクセス性能を Table 10-3 に示す。なお、性能値は CMG あたりの性能である。

Table 10-3 A64FX Memory Access Performance

		Memory Access Performance
Local memory latency (load-to-use)	Shortest core	131 ns (@ CPU 2GHz)
	Longest core	140 ns (@ CPU 2GHz)
Read throughput	Peak	256 GB/s (per MAC) (@ CPU 2GHz)
Write throughput	Peak	128 GB/s (per MAC) (@ CPU 2GHz)

# 11. データ・プリフェッチ

---

プリフェッチとは、近い将来に利用が予測されるデータを事前にキャッシュに読み込んでおくことで性能向上を図る動作である。プリフェッチには、専用命令で明示的にデータをキャッシュに読み出すソフトウェア・プリフェッチと、ハードウェアがアドレスを自動的に予測して読み出しをするハードウェア・プリフェッチの2種類がある。A64FXは両方のプリフェッチ方式に対応し、さらにプリフェッチの動作を細かく制御するためのハードウェア・プリフェッチ・アシスト機能を実装している。

## 11.1. プリフェッチの動作概要

本節ではプリフェッチのハードウェア動作について説明する。まず、説明を容易にするために以下の用語について定義する。

### ソフトウェア・プリフェッチ

アーキテクチャ命令による明示的なプリフェッチ指示である。

### ハードウェア・プリフェッチ

ハードウェアのアドレス予測機構に基づいたプリフェッチ指示である。

### デマンド・アクセス

ロード／ストア命令など、レジスタとメモリ空間との間でデータの移動があるメモリ・アクセスの概念である。

### プリフェッチ・アクセス

プリフェッチ命令およびハードウェア・プリフェッチによって生成されるメモリ空間へのメモリ・アクセスの概念である。プリフェッチ・アクセスにはどのキャッシュ階層にデータ・フィルするかという情報、アクセスのタイプ、アドレスの信頼度を示す情報が含まれている。

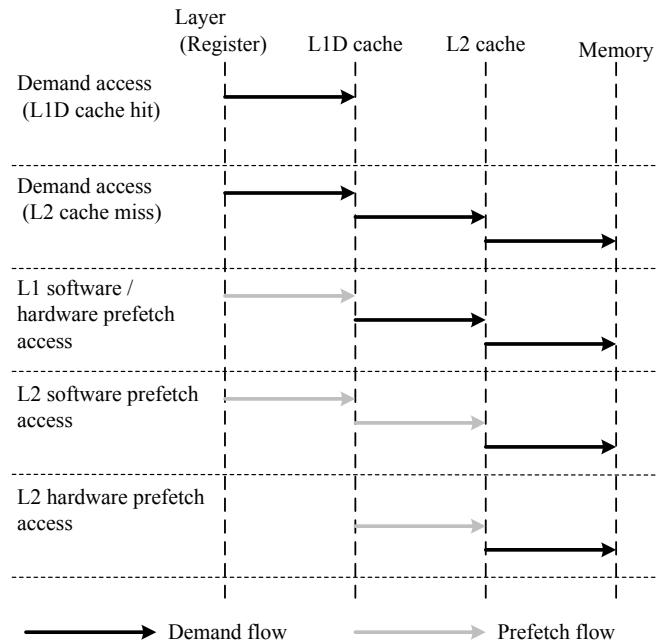
### デマンド・フロー

オペレーション・フローのうち、レジスタやキャッシュ、メモリの各階層間でデータのやり取りを伴うものを指す。例えば、ロード命令の LD フローはロード／ストア・パイプラインのデマンド・フローそのものである。LD フローはレジスタにデータを書き込むからである。同様に、L2 キャッシュのパイプラインのオペレーション・フローのうち、L1 キャッシュにデータを返答する必要のあるものがデマンド・フローである。

### プリフェッチ・フロー

デマンド・フローに対して、レジスタやキャッシュ、メモリの各階層間でデータのやり取りが不要なものを指す。例えば、プリフェッチ命令のオペレーション・フローはレジスタにデータを書き込む必要がないため、ロード／ストア・パイプラインにとってはプリフェッチ・フローとなる。同様に、L2 キャッシュのパイプラインにとって、L1 キャッシュにデータを返答する必要のないものがプリフェッチ・フローである。

デマンドおよびプリフェッチ・アクセスと、それによって生成されるオペレーション・フローの関係を Figure 11-1 に示す。



**Figure 11-1 Operation-Flows for Demand Access and Prefetch Access**

ロード／ストア命令のアクセスはすべてデマンド・アクセスであり、それらの処理のためのオペレーション・フローはすべてデマンド・フローである。ロード／ストア命令は、最終的にレジスタにデータを読み込む、または、レジスタからデータを書き込む必要あり、L2 キャッシュや MAC はデータを返答する必要があるからである。

一方、プリフェッチ・アクセスの処理のためのオペレーション・フローは各キャッシュ階層にて種類が分かれる。例えば、ソフトウェア・プリフェッチによる L1D キャッシュへのプリフェッチ・アクセスはロード／ストア・パイプラインにとって、プリフェッチ・フローであるが、L2 キャッシュのパイプラインにとってデマンド・フローとなる。L1D キャッシュにデータを返答する必要があるためである。

また、ソフトウェア・プリフェッチとハードウェア・プリフェッチではロード／ストア・パイプラインにおける必要なオペレーション・フローが異なる点に注意が必要である。ソフト・ウェア・プリフェッチは命令であるため、そのオペレーション・フローはロード命令の LD フローのように実行パイプラインやロード／ストア・パイプラインを通過する。つまり、それらのパイプラインのリソースを消費する。一方、ハードウェア・プリフェッチによるプリフェッチ・アクセスは実行パイプラインとロード／ストア・パイプラインに対するオペレーション・フローが不要である。L2 キャッシュのパイプラインに直接オペレーション・フローを投入するからである。ただし、データ・ファイル／ライトバックための、MI フロー、MO フローは必要になることに注意が必要である。

プリフェッチにはデマンド・アクセスとの間に距離 (Distance) と呼ばれる概念がある。プリフェッチの目的はデマンド・アクセスのレイテンシを隠蔽することであるが、これを達成するためにはデマンド・アクセスと同じアドレスに対して時間軸方向に先行してプリフェッチをする必要がある。例えば、N サイクルのデマンド・アクセスのレイテンシを隠蔽するためにはデマンド・アクセスが生成される N サイクル以上前にプリフェッチをする必要がある。この時間差がプリフェッチの距離である。特に、メモリ・アクセスが連続アクセスの場合には距離をアドレス空間方向に置き換えることができる。例えば、デマンド・アクセスが A のときにアドレス

A+P へのプリフェッчをする。そして、デマンド・アクセスが A+P に到達したときにアドレス A+P へのデマンド・アクセスがキャッシュ・ヒットすればデマンド・アクセスのレイテンシ N を隠ぺいできたといえる。このとき、アドレスの差 P が距離である。本ガイドではアドレス空間方向の距離をプリフェッч距離として扱う。

## 11.2. プリフェッч・アクセスのタイプ

プリフェッч・アクセスに付加される情報のひとつに「タイプ」がある。タイプとはプリフェッчされたデータがロードのためであるか、ストアのためであるかを示す情報であり、"Read" と "Write" の 2 種類がある。ハードウェアはタイプ情報をを利用してデータ・フィル時のキャッシュ・ステートを決める。

## 11.3. プリフェッч・アクセスの信頼度

プリフェッч・アクセスに付加される情報のひとつに「信頼度」がある。信頼度とは生成されたプリフェッч・リクエストを処理する優先度を決定するための指標であり、"Strong" と "Weak" の 2 種類がある。A64FX では、ソフトウェア・プリフェッчはタグド・アドレスとシステム・レジスタを組み合わせて、ハードウェア・プリフェッчはシステム・レジスタを用いてプリフェッч・アクセスごとに信頼度を個別設定できる。設定された信頼度はプリフェッч・フローにも適用される。

### Strong 属性のプリフェッч・フロー

ハードウェアはプリフェッч・アクセスを可能な限り正しく完了させようとする。例えば、生成されたプリフェッч・フローをメモリ階層まで伝達するための資源が不足している場合には資源に空きが生じるまで待機する。このとき、後続のロード／ストア命令に影響がある場合でもフローは削除されずにプリフェッч・アクセスは最後まで実行される。ただし、ソフトウェア・プリフェッчにおいては、TLB ミス、または Page fault が発生した時点でプリフェッч・リクエストは削除される。このとき、プリフェッч・アクセスの元となったプリフェッч命令は NOP 命令として扱われる。ハードウェア・プリフェッчにおいては、TLB ミスした時点でプリフェッч・アクセスが停止し、PFQ もクリアされる。Strong 属性のプリフェッч・アクセスはロード／ストア・パイプラインをひっ迫させる場合もあるので、プリフェッчしたデータが確実に使われる場合にのみ使うことを推奨する。なお、プリフェッч命令の動作については A64FX 論理仕様書を参照のこと。

### Weak 属性のプリフェッч・フロー

ハードウェアは資源に余裕があればプリフェッч・アクセスを正しく完了させるが、資源に余裕がない場合は生成されたプリフェッч・リクエストは削除される。基本的にデマンド・フローや Strong 属性のプリフェッч・フローの処理が優先される。

## 11.4. ソフトウェア・プリフェッチ

ソフトウェア・プリフェッチは、プリフェッチ命令によって明示的にプリフェッチ・アクセスをする。プリフェッチ命令はオペランド部でプリフェッチ・アクセスに必要なプリフェッチ・アドレス、データ・フィル先のキャッシュ階層、キャッシュ・ステートを制御できる。A64FXは独自のHPC向け拡張としてタグド・アドレスを用いてハードウェアの振る舞いを制御できるHPCタグド・アドレス・オーバーライド機能を実装している。

### 11.4.1. プリフェッチ命令の分類

プリフェッチ命令には大きく分けてARMv8プリフェッチ命令、SVE Contiguousプリフェッチ命令、SVE Gatherプリフェッチ命令の3種類がある。各々の定義と特徴を以下に示す。

#### ARMv8 プリフェッチ命令

オペランドで指定したプリフェッチ・アドレスに対してプリフェッチ・アクセスをさせる命令である。ハードウェアは指定されたアドレスを含むキャッシュ・ライン単位でメモリからデータ・フィルする。なお、ARMv8プリフェッチ命令のデータサイズはバイト型と解釈されるため、キャッシュ・ライン境界をまたぐ現象は発生しない。

#### SVE Contiguous プリフェッチ命令

オペランドで指定したアドレスを先頭に、そこからSVEベクトル・データ長を加算したアドレスまでの範囲に対してプリフェッチ・アクセスをさせる命令である。ハードウェアは先頭から終端までの範囲のアドレスを含む、キャッシュ・ライン単位でメモリからデータ・フィルする。プリフェッチの先頭と終端アドレスが別のキャッシュ・ラインに属する場合には両方のメモリブロックをフィルする。

#### SVE Gather プリフェッチ命令

離散アクセス命令(Gather / Scatter)と同様のアドレッシング・モードをサポートする命令である。一命令で複数のアドレスに対してプリフェッチ・アクセスを行うことができる。個々のアドレスに対する基本的な動作はARMv8プリフェッチ命令と同様である。

各プリフェッチ命令の分類とニーモニックの対応をTable 11-1に示す。

Table 11-1 Classifications and Mnemonics of Prefetch Instructions

Classification	Mnemonic	Description
ARMv8 prefetch instruction	PRFM (immediate)	Prefetch instructions that support consecutive load/store (without consideration of line cross)
	PRFM (literal)	
	PRFM (register)	
	PRFM (unscaled offset)	
SVE contiguous prefetch instruction	PRF[BHWD] (scalar plus immediate)	Prefetch instructions that support consecutive load/store (with consideration of line cross)
	PRF[BHWD] (scalar plus scalar)	
SVE gather prefetch instruction	PRF[BHWD] (scalar plus vector)	Prefetch instructions that support discrete access instructions (gather/scatter)
	PRF[BHWD] (vector plus immediate)	

### 11.4.2. ソフトウェア・プリフェッチの属性

プリフェッチ命令では、第1オペランドにてプリフェッチ・オプションを指定することができます。オプションにはType、Target、Policyの3項目があり、TypeとTargetの組み合わせによってTable 11-2に示すようにデータ・ファイル先のキャッシング階層とキャッシング・ステートを制御できる。A64FXではPolicyはハードウェア制御に使用していないため、ハードウェアの動作に影響しない。

Table 11-2 Correspondence Between Prefetch Instruction Options, Cache Levels, and States

		Target		
		L1	L2	L3
Type	PLI	NOP	NOP	NOP
	PLD	L1D / S or E	L2 / S or E	NOP
	PST	L1D / E	L2 / E	NOP

また、プリフェッチ命令のタグド・アドレス部のうち pf\_func[0] ビットを利用することでソフトウェア・プリフェッチの信頼度を制御できる。ビット・フィールドと信頼度の関係をTable 11-3に示す。ビットフィールドについてはA64FX論理仕様書を参照のこと。

Table 11-3 Correspondence Between pf\_func[0] Bit and Software Prefetch Reliability

pf_func[0]	Software Prefetch Reliability
0	Strong
1	Weak

## 11.5. ハードウェア・プリフェッチ

A64FXはハードウェアによって近い将来アクセスするであろうアドレスを予測してプリフェッチ・アクセスをする機能を持つ。これら機能を総称してハードウェア・プリフェッチと呼ぶ。A64FXのハードウェア・プリフェッチは連続アクセス・ストリームに対してアドレス予測ができる。ハードウェア・プリフェッチには”Stream detect mode”と”Prefetch injection mode”的2種類のモードがある。

### 11.5.1. ハードウェア・プリフェッチのための資源

ハードウェアはPFQ(Pre-Fetch Queue)と呼ばれるアドレス予測およびプリフェッチ・アクセスを行うための資源を持つ。PFQは各プロセッサ・コア内部にあり、プロセッサ・コアあたり16エントリを有する。PFQには予測アドレス、プリフェッチ距離、予測のためのアドレスオフセットが保存される。

予測アドレスは、将来デマンド・アクセスすると思われるアドレスである。ハードウェアが事前に用意した予測アドレスと実際のデマンド・アクセスのアドレスが一致したとき、PFQはそのアドレスにプリフェッチ距離を加算したアドレスに対してプリフェッチ・アクセスをする。その

後、さらにアドレス・オフセットを加算して予測アドレスを更新する。この動作を繰り返すことでプリフェッチ・アクセスが継続する。

### 11.5.2. Stream detect mode の動作

ハードウェア・プリフェッチの2つのモードのうち、"Stream detect mode" の動作について説明する。Stream detect mode では、連続アクセスストリームを自動で検出してプリフェッチ・アクセスをする。Stream detect mode であるときのハードウェア・プリフェッチ動作イメージを Figure 11-2 に示す。

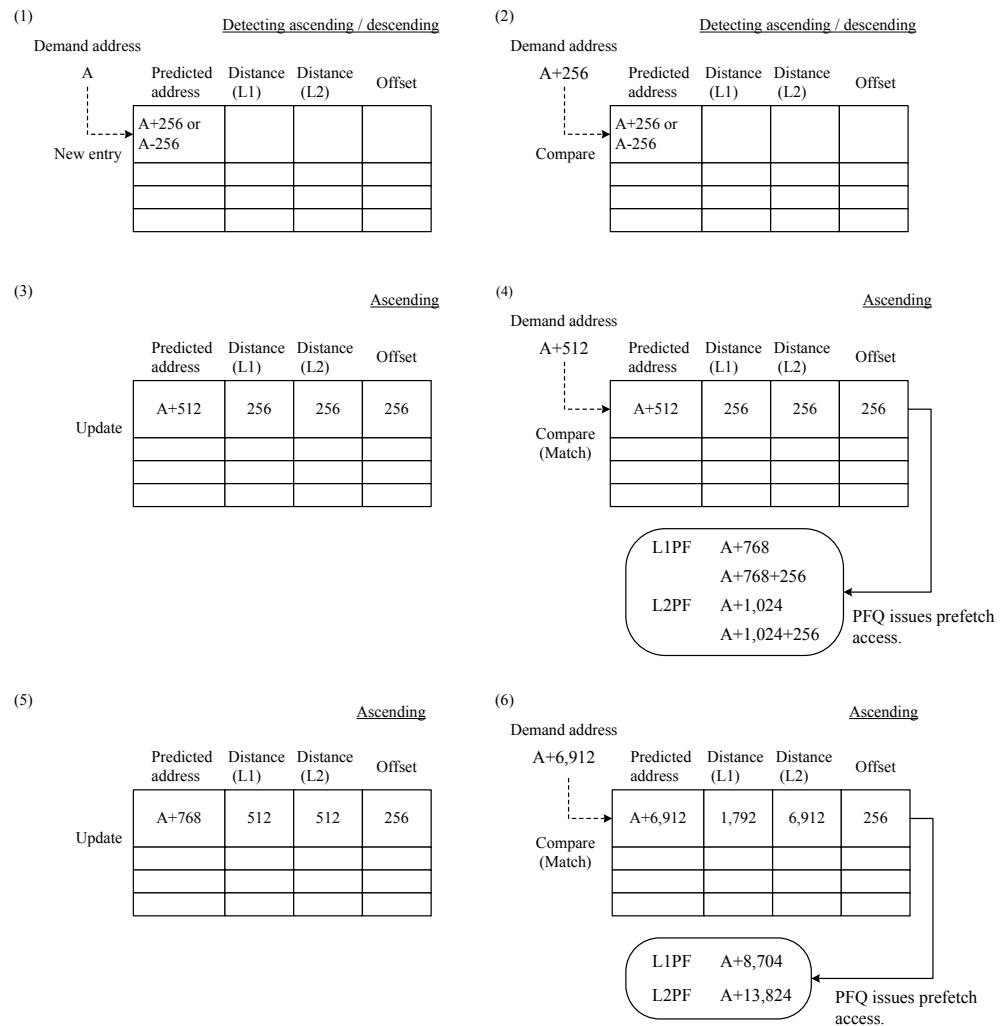


Figure 11-2 Hardware Prefetch Behavior in Stream Detect Mode

#### ストリームの検出と PFQ への登録

PFQ はデマンド・アクセスにおける L1D キャッシュ・ミスを監視する。キャッシュ・ミスが起こると、デマンド・アドレスを基に予測アドレスを生成して PFQ に登録することでデマンド・アクセスのストリームが昇順か降順かを判定しようとする (Figure 11-2 (1))。引き続きデマンド・アクセスが行われると、デマンド・アドレスと先程 PFQ に登録した予測アドレスとを比較してストリーム方向を決定する (Figure 11-2 (2))。ストリーム方向が決定すると PFQ の予測ア

ドレスを更新し、新たにプリフェッチ距離、アドレス・オフセットも登録してデマンド・アクセスを追従し始める。プリフェッチ距離、アドレス・オフセットは昇順の時はプラス値、降順のときはマイナス値として登録される。最初に登録されるプリフェッチ距離およびオフセットの絶対値はそれぞれ 256 バイトである (Figure 11-2 (3))。

### プリフェッチ・アクセスの生成

PFQ は予測アドレスと一致するデマンド・アクセスを検出するとプリフェッチ・アクセスを発行する (Figure 11-2 (4))。プリフェッチ・アクセスを指示すると PFQ は予測アドレスにオフセットを加算して予測アドレスを更新する (Figure 11-2 (5))。デマンド・アクセスの追従を始めた時から一定期間は PFQ は 2 キャッシュ・ライン分の L1、L2 プリフェッチ・アクセスを指示する。また、プリフェッチ・アクセスをする度にプリフェッチ距離を 256 バイトずつ加算することでプリフェッチ距離を伸長する。

### プリフェッチ・アクセスの定常生成

同一の PFQ によるプリフェッチ・アクセス指示を繰り返すと、やがてプリフェッチ距離は最大値に到達する。プリフェッチ距離が最大値に到達すると PFQ はプリフェッチ・アクセスの指示を 1 キャッシュ・ライン分に切り替える。同時にプリフェッチ距離の伸長も停止する (Figure 11-2 (6))。以後、予測アドレスと一致するデマンド・アクセスが続く限りはこの状態を継続する。

なお、A64FX の Stream detect mode ではデマンド・アクセスとプリフェッチ・アクセスのアドレスはキャッシュ・ライン単位に丸められる。すなわち、アクセス・アドレスの下位 7 ビットは無視される。これにより、完全に連続なストリーム・アクセスでなくとも、キャッシュ・ライン単位で連続であればプリフェッチ・アクセスを発行できる。

## 11.5.3. Prefetch injection mode の動作

ハードウェア・プリフェッチの 2 つのモードのもう一方が "Prefetch injection mode" である。Prefetch injection mode はプリフェッチ制御用レジスタを使ってロード／ストア命令のアクセスの特徴を設定し、それを用いてハードウェア・プリフェッチを行う。Prefetch injection mode はさらに PFQ\_ALLOCATE モードと PFQ\_NOALLOCATE モードにわかれる。

### PFQ\_ALLOCATE モード

本モードの動作は基本的に Stream detect mode と同様である。デマンド・アクセスの LID キャッシュ・ミスを監視し、キャッシュ・ミスが発生すると PFQ に予測アドレス、プリフェッチ距離、オフセットを登録する。ただし、予測アドレス、プリフェッチ距離、オフセットがシステム・レジスタの設定値から算出されるという点で Stream detect mode と異なる。PFQ\_ALLOCATE モード下におけるハードウェア・プリフェッチは以下の 2 ステップで行われる。

## 1. ストリームの検出

対象とするストリームのデマンド・アクセスが L1D キャッシュ・ミスすると、そのデマンド・アクセスのアドレスにオフセットを加算した値を予測アドレスとする。プリフェッヂ距離とオフセットはシステム・レジスタに設定された値が使われる。

## 2. プリフェッヂ・アクセスの生成

予測アドレスがデマンド・アクセスのアドレスと一致すると、そのアドレスにプリフェッヂ距離を加算したアドレスに対してプリフェッヂ・アクセスを指示する。さらに、予測アドレスにオフセットを加算して更新する。ただし、プリフェッヂ距離は初期値が保持され伸長しない。

## PFQ\_NOALLOCATE モード

本モードは PFQ でアクセスの L1D キャッシュ・ミスの監視をせず、対象とするストリームのデマンド・アクセスが発生した時点でプリフェッヂ・アクセスを指示する。プリフェッヂ・アクセスの対象はデマンド・アクセスのアドレスにプリフェッヂ距離を加算したアドレスである。本モードでは、無条件でプリフェッヂ・アクセスを生成するため、デマンド・アクセスがキャッシュ・ヒットするときでもプリフェッヂ・アクセスが発生する。一方、PFQ による監視を行わないため、PFQ を消費しない利点がある。

### 11.5.4. ハードウェア・プリフェッヂ・アシスト機能

A64FX はハードウェア・プリフェッヂの利便性を高めるために、ハードウェア・プリフェッヂの動作をコントロールするインターフェースとしてハードウェア・プリフェッヂ・アシスト機能を持つ。

#### Stream detect mode の場合

Stream detect mode ではタグド・アドレスとシステム・レジスタを用いて以下のようにプリフェッヂ動作を制御できる。

- プリフェッヂの無効化

タグド・アドレスを用いて命令ごとに PFQ の監視対象とするか否かを設定できる。PFQ の監視対象から外れた命令からはプリフェッヂ・アクセスは生成されない。

- プリフェッヂ・アクセスのキャッシュ階層属性の指定

タグド・アドレスを用いて指定したキャッシュ階層属性を持つプリフェッヂ・アクセスを指示できる。キャッシュ階層属性は、L1D キャッシュ、L2 キャッシュ、両方の 3 つから選択できる。

- プリフェッヂ・アクセスの信頼度属性の指定

システム・レジスタを経由することで PFQ が生成するプリフェッヂ・リクエストの信頼度属性を指定できる。信頼度属性は、Strong 属性、Weak 属性の 2 つから選択できる。

- PFQ の最大プリフェッヂ距離の指定

システム・レジスタを経由することで PFQ のプリフェッヂ距離の最大値を指定できる。

## Prefetch injection mode の場合

Prefetch injection mode では Stream detect mode の機能に加えて以下のようにプリフェッチ動作を制御できる。

- ストリーム番号の付加  
タグド・アドレスを用いて命令ごとにストリーム番号を付加できる。基本的にストリーム単位で Stream detect mode 機能を指示できる。
- ストリームごとのプリフェッチ距離の指定  
システム・レジスタを使ってストリーム番号ごとにプリフェッチ距離を指定できる。
- ストリームごとのオフセットの指定  
システム・レジスタを使ってストリーム番号ごとにオフセットを指定できる。

### 11.5.5. ハードウェア・プリフェッチのキャッシュ階層属性

プリフェッチ・アクセスに付加される情報のひとつにキャッシュ階層の属性がある。キャッシュ階層属性とは、プリフェッチ先がどのキャッシュ階層かを示すものであり、プログラムの高速化のためには適切な階層属性を持ったプリフェッチ・アクセスをする必要がある。不要なプリフェッチ・アクセスをすることはハードウェア・リソースの浪費につながる。

ハードウェア・プリフェッチの場合、最適なキャッシュ階層属性はハードウェアが自動で決定する。基本的に PFQ は L1D キャッシュ属性を持つプリフェッチ・アクセスと、L2 キャッシュ属性を持つプリフェッチ・アクセスの両方を指示する。このとき、L2 キャッシュ属性を持ったプリフェッチ・アクセスが L2 キャッシュにおいてキャッシュ・ヒットを繰り返すと、PFQ は L2 キャッシュ属性を持つプリフェッチ・アクセスを停止する。ただし、L1D キャッシュ属性を持ったプリフェッチ・アクセスが L2 キャッシュ・ミスを起こすと、PFQ は L2 キャッシュ属性を持つプリフェッチ・アクセスを再開させる。

## 11.6. Prefetch injection mode の使用例

Prefetch injection mode の特徴のひとつがオフセットをソフトウェアで指定できることである。この機能を使うことで、キャッシュ・ライン・サイズを超えるストライド・アクセスに対してプリフェッチ・アクセスを生成することができる。Figure 11-3 にプログラム例を、Table 11-4 に制御レジスタの構成例を示す。タグド・アドレスと制御レジスタの仕様は A64FX 論理仕様書に記述されている。

Sample code

```

int i;
double y[N], x[N*64];
assert (N > 0);
for (i = 0; i < N; i++) {
    y[i] = x[i*64];      /* accesses the array x with stride of 512 bytes width. */
}

```

Sample assembly code

```

mov      x0,#N
adr      x1,y          // sets the address of array y.
adr      x2,x          // sets the address of array x.
orr      x2,x2,#(8<<60) // merges base address and tagged
                           // address which assigns the stream to
                           // control#0 register.

loop:
ldr      d0,[x2]
str      d0,[x1]
add      x2,x2,#512
add      x1,x1,#8
subs   x0,x0,#1
b.gt    x0,loop

```

Figure 11-3 Usage Example of Prefetch Injection Mode

Table 11-4 Control Register Configuration Example

System Register	Bit Field	Set Value	Description
IMP_PF_INJECTION_CTRL0_EL0	V	1	Enables the control register.
	L1W	0	Sets the L1 prefetch attribute to Strong.
	L2W	0	Sets the L2 prefetch attribute to Strong.
	A	1	Sets PFQ_ALLOCATE mode.
	T	0	Sets the prefetch attribute to PLD.
	SWW	0	This is an instruction for software prefetch. It does not matter whether the value is 0 or 1 in this example.
	PFQ_OFFSET	512	Sets the same value as the stride width.
IMP_PF_INJECTION_DISTANCE0_EL0	L1PF_DISTANCE	1,024	Sets the L1 prefetch distance.
	L2PF_DISTANCE	10,240	Sets the L2 prefetch distance.

なお、プリフェッチ・アクセスの信頼度、PFQ\_ALLOCATE / PFQ\_NOALLOCATE モード、およびプリフェッチ距離はプログラムの特性に合わせて適切に決定する必要がある。

# 12. セクタ・キャッシュ

## 12.1. セクタ・キャッシュの概要

セクタ・キャッシュは、キャッシュの領域を区分けし、命令単位、またはプロセッサ・コア単位で使用する区域を選択できる機能である。ソフトウェアがキャッシュの使用をより細粒度に制御する方法を提供することを目的としている。A64FXでは、区分けされた各々の領域をセクタと呼ぶ。セクタはシステム・レジスタを介して、キャッシュのway単位で任意の容量に区分けできる。本機能はL1DキャッシュとL2キャッシュに実装されている。Figure 12-1に示すように、L1Dキャッシュには命令単位で指示できるセクタが4領域ある。L2キャッシュにも4領域のセクタがあるが、プロセッサ・コア単位で指示できるセクタが2領域、命令単位で指示できるセクタが2領域という階層構造をとる。L1Dキャッシュの4セクタは、L2キャッシュにおいては同一のセクタグループ内の2セクタにマップされる。また、セクタの領域はそれぞれのCMGに閉じている。命令単位のセクタの指示にはタグド・アドレスを用いる。プロセッサ・コア単位の指定にはシステム・レジスタを用いる。タグド・アドレス、およびシステム・レジスタの仕様はA64FX論理仕様書を参照のこと。

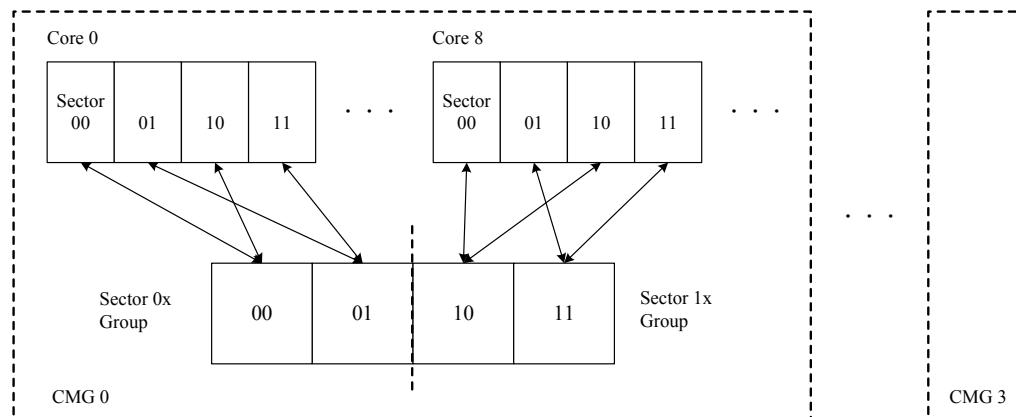
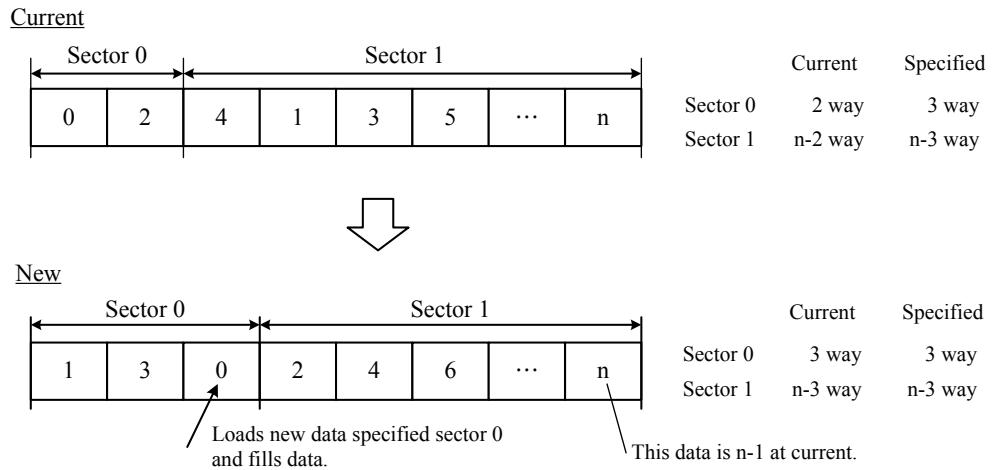


Figure 12-1 L1D/L2 Sector Cache

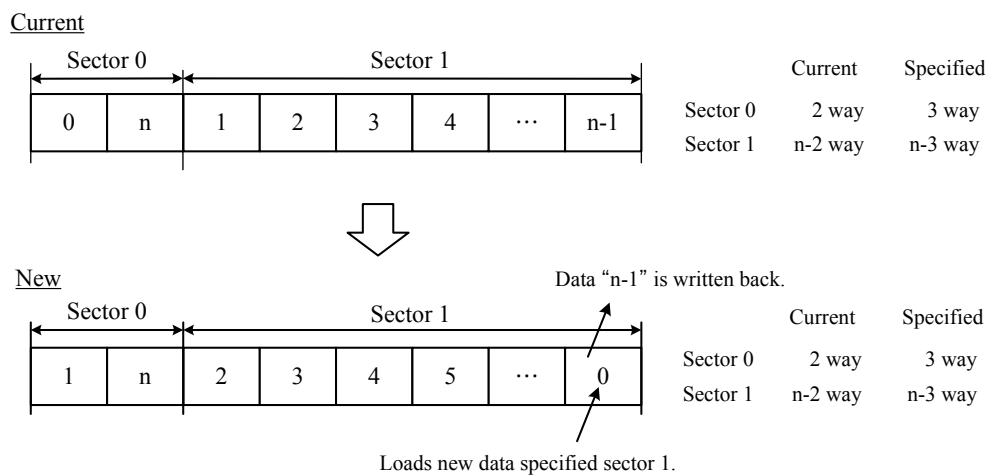
## 12.2. セクタ・キャッシュの動作

セクタ・キャッシュのセクタの容量は、各セクタに割り当てられる最大way数にて指示される。また、セクタ容量はプログラム実行中に動的に変更できる。ハードウェアはセクタ容量が変更されると、それぞれのキャッシュ・ラインへデータ・フィルする機会に容量調整を行い、徐々に指定されたセクタ容量に近づける制御を行う。Figure 12-2、Figure 12-3に例を示す。Figure 12-2は要求セクタ容量に対してセクタ0側の使用量が少ない例である。セクタ0の指定があるロード命令が実行されると、セクタ1側のデータがライトバックされることで容量が調整される。対して、Figure 12-3はセクタ1側の使用量が多いときにセクタ1の指定があるロード命令が実行

される例である。この場合、セクタ 0 の使用量が減少しないようにセクタ 1 側のデータがライトバックされる。



**Figure 12-2 Example of Sector Cache Capacity Adjustment (1)**



**Figure 12-3 Example of Sector Cache Capacity Adjustment (2)**

# 13. ハードウェア・バリア

ハードウェア・バリアはソフトウェアのプロセス、またはスレッド間の同期をハードウェアでサポートする機能である。Figure 13-1 に示すように、各 CMG 内に専用のシステム・レジスタがあり、それを介して同期処理を行う。システム・レジスタは L2 キャッシュ内に実装されているため、同期処理のためのレジスタ・アクセスは L2 キャッシュ・ヒットと同程度の応答時間になる。また、システム・レジスタそのものがバリア変数となってハードウェアがレジスタ操作のアトミック性を保証するため、変数の操作のための排他処理が不要である。これらの特徴によってプログラムの簡素化、同期処理の高速化を狙っている。

なお、ハードウェア・バリアのリソースは CMG 単位に実装されているため、CMG 間を横断した同期処理はサポートしていない。Figure 13-2 に同期処理のためのサンプルコードを示す。ハードウェア・バリアの詳細な仕様は A64FX 論理仕様書を参照のこと。

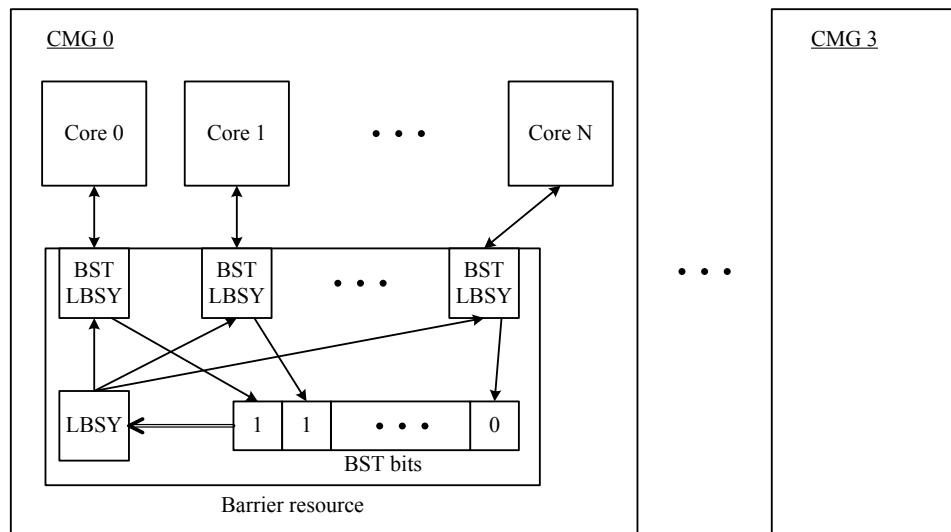


Figure 13-1 Hardware Barrier Resources

```
#define BARRIER_LBSY_SYNC_W1_EL0 S3_3_C15_C15_1
#define BARRIER_BST_SYNC_W1_EL0 S3_3_C15_C15_1

mrs_s x2, BARRIER_LBSY_SYNC_W1_EL0           // Load LBSY to x2
eor x1, x2, #0x1
msr_s BARRIER_BST_SYNC_W1_EL0, x1            // Write ~LBSY to BST
sevl                                         // EVENT register clear
wfe                                          
loop:
    wfe                                         // Sleep
    mrs_s x2, BARRIER_LBSY_SYNC_W1_EL0
    cmp x2, x1                                     // Compare x2 (= LBSY) and x1 (= ~LBSY)
    b.ne loop                                     // Reload LBSY to x2 if LBSY != ~LBSY
```

Figure 13-2 Sample Code for Synchronization Processing

# 14. Performance Monitor Events

プロセッサには、プログラムの動的なふるまいを観測するために Performance Monitoring Unit (PMU) が実装されている。A64FX は ARMv8 Manual、および SVE Manual に定義されている Event の他に、独自の Event を実装している。これらの Event は各 Event を直接的な意味で利用するだけでなく、複数の Event を組み合わせることでソフトウェアの性能解析を補助する指標を算出できるように設計されている。本章ではこれらの指標の作成方法について説明する。なお、各 Event の詳細仕様は A64FX PMU events 仕様書に記載されている。

## 14.1. Instruction Mix

プログラム実行時の動的な命令頻度分布を算出するための Event を Table 14-1 にまとめる。Instruction Mix のためのすべての Event は、アーキテクチャ命令のコミットを数え上げる実装になっている。これらの Event が数える命令群には包含関係があり、これらを組み合わせることで Event が定義されてない命令群の頻度を算出できる。Event の包含関係は Table 14-1 の Event 名の列においてインデントの深さで表されている。また、Other と表記されている行は実際の Event ではなく、いくつかの Event を組み合わせて算出される項目である。算出式は Table 14-2 に示すとおりである。なお、SVE\_MATH\_SPEC event は INST\_SPEC event に含まれないことに注意が必要である。

Table 14-1 Performance Events for Instruction Mix

Instruction Mix のための Event	Event のカウント対象
INST_SPEC	全命令
FP_SPEC	浮動小数点演算に分類されるすべての命令
FP_FMA_SPEC	FMA 系の演算命令
FP_RECPE_SPEC	逆数近似命令と逆数平方根近似命令
Others (Basic FP operations)	一般的な浮動小数点演算命令
FP_CVT_SPEC	浮動小数点精度変換命令(汎用レジスタの値との変換も含む)
FP_MV_SPEC	浮動小数点レジスタを使用する転送命令(汎用レジスタも含む)
ASE_SVE_INT_SPEC	浮動小数点レジスタを使用する整数演算命令
PRD_SPEC	Predicate レジスタを使用する演算命令
LD_SPEC	すべてのロード命令
BASE_LD_REG_SPEC	汎用レジスタへのロード命令
ASE_SVE_LD_SPEC	浮動小数点レジスタへのロード命令
FP_LD_SPEC	浮動小数点レジスタへのスカラ型のロード命令
Others (All vector loads)	浮動小数点レジスタへのベクトル型のロード命令
SVE_LDR_REG_SPEC	SVE のすべての LDR 命令

Instruction Mix のための Event		Event のカウント対象
	SVE_LDR_PREG_SPEC	SVE LDR(predicate)命令
	Others (LDR vector)	SVE LDR(vector)命令
	BC_LD_SPEC	浮動小数点レジスタへの Replicate と Broadcast ロード命令 (LD1R 命令)
	ASE_SVE_LD_MULTI_SPEC	浮動小数点レジスタへの Multiple structure ロード命令 (LD[234]*命令)
	SVE_LD_GATHER_SPEC	SVE Gather load 命令
	SVE_LDFF_SPEC	SVE first-fault と non-fault ロード命令
	Others (Basic vector loads)	浮動小数点レジスタへの一般的なベクトル型のロード命令
ST_SPEC		すべてのストア命令
	BASE_ST_REG_SPEC	汎用レジスタからのストア命令
	ASE_SVE_ST_SPEC	浮動小数点レジスタからのストア命令
	FP_ST_SPEC	浮動小数点レジスタからのスカラ型のストア命令
	Others (All vector stores)	浮動小数点レジスタからのベクトル型のストア命令
	SVE_STR_REG_SPEC	SVE のすべての STR 命令
	SVE_STR_PREG_SPEC	SVE STR(predicate)命令
	Others (STR vector)	SVE STR(vector)命令
	ASE_SVE_ST_MULTI_SPEC	浮動小数点レジスタからの Multiple structure ストア命令 (ST[234]*命令)
	SVE_ST_SCATTER_SPEC	SVE Scatter store 命令
	Others (Basic vector stores)	浮動小数点レジスタからの一般的なベクトル型のストア命令
PRF_SPEC		すべてのプリフェッチ命令
	SVE_PRF_GATHER_SPEC	SVE gather prefetch 命令
	SVE_PRF_CONTIG_SPEC	SVE contiguous prefetch 命令
	Others (Prefetch in base inst.)	Base instruction プリフェッチ
DCZVA_SPEC		DC ZVA 命令
BR_PRED		すべての分岐命令
CRYPTO_SPEC		すべての暗号命令
SVE_MOVPRFX_SPEC		SVE MOVPRFX 命令
Others (Base insts. excluding load/store)		ロード／ストア命令を除く、Base instruction に属する命令
包含関係がない Event		
SVE_MATH_SPEC		SVE 数学関数補助命令

Table 14-2 Formulas for Other (Instruction Mix)

項目	算出式
Basic FP operations	FP_SPEC – (FP_FMA_SPEC – FP_RECPE_SPEC)
All vector loads	ASE_SVE_LD_SPEC – FP_LD_SPEC
LDR vector	SVE_LDR_REG_SPEC – SVE_LDR_PREG_SPEC
Basic vector loads	ASE_SVE_LD_SPEC – (FP_LD_SPEC + SVE_LDR_REG_SPEC + BC_LD_SPEC + ASE_SVE_LD_MULTI_SPEC + SVE_LD_GATHER_SPEC + SVE_LDFF_SPEC)
All vector stores	ASE_SVE_ST_SPEC – FP_TD_SPEC
STR vector	SVE_STR_REG_SPEC – SVE_STR_PREG_SPEC
Basic vector stores	ASE_SVE_ST_SPEC – (FP_ST_SPEC + SVE_STR_REG_SPEC + ASE_SVE_ST_MULTI_SPEC + SVE_ST_SCATTER_SPEC)
Prefetch in base instruction	PRF_SPEC – (SVE_PRF_GATHER_SPEC + SVE_PRF_CONTIG_SPEC)
Base insts. excluding load/store	INST_SPEC – (FP_SPEC + FP_CVT_SPEC + FP_MV_PSEC + ASE_SVE_INT_SPEC + PRD_SPEC + LD_SPEC + ST_SPEC + DCZVA_SPEC + BR_PRED + CRYPTO_SPEC + SVE_MOVPRFX_SPEC)

## 14.2. FLOPS

Floating operations per second (FLOPS) を算出するための Event を Table 14-3 にまとめる。これらの Event はコミットされた命令の浮動小数点演算数を数え上げる。Event のうち SVE 命令に関するものは 128 bits 単位の演算数しか数えないことから、プログラム実行時の Vector Length の影響を受けない。したがって、正しい演算数は実行時の Vector Length を考慮して算出しなければならない。また、FMA 系の演算は要素あたり 2 演算として数える。

PMU は PE 単位の資源であることから、Event で測定される演算数は PE 単位である。したがって、並列実行時の総演算数を算出するときはジョブモデルなどの考慮が必要である。また、FLOPS は単位時間あたりの演算数であるため、プログラム実行中のプロセッサ動作周波数とプログラムの実行時間の外部パラメータを別途必要とする。実行時間を高精度で求めるには、同時に取得した CPU\_CYCLES event を使用することを推奨する。

Table 14-3 Performance Events for FLOPS

Performance Event	Event の説明
FP_SCALE_OPS_SPEC	SVE 命令の各命令の要素数を考慮した 128 bits あたりの演算数
FP_FIXED_OPS_SPEC	SIMD&FP 命令の要素数を考慮した演算数
FP_HP_SCALE_OPS_SPEC	FP_SCALE_OPS_SPEC のうち、半精度演算のみの演算数
FP_HP_FIXED_OPS_SPEC	FP_FIXED_OPS_SPEC のうち、半精度演算のみの演算数
FP_SP_SCALE_OPS_SPEC	FP_SCALE_OPS_SPEC のうち、单精度演算のみの演算数
FP_SP_FIXED_OPS_SPEC	FP_FIXED_OPS_SPEC のうち、单精度演算のみの演算数
FP_DP_SCALE_OPS_SPEC	FP_SCALE_OPS_SPEC のうち、倍精度演算のみの演算数
FP_DP_FIXED_OPS_SPEC	FP_FIXED_OPS_SPEC のうち、倍精度演算のみの演算数

## 14.3. Hardware Resource Monitor Events

プロセッサの基本的なリソースの振る舞いを観測するための Event を Table 14-4 にまとめる。これらの Event は、キャッシュ・ミスや分岐予測ミスなどプログラム実行時の動的なハードウェアの動作を数え上げる。

Table 14-4 Performance Events for Hardware Resource Monitoring

Performance Event	Event の説明
BR_MIS_PRED	分岐予測ミスによるパイプライン・フラッシュの回数
L1I_CACHE_REFILL	L1I キャッシュ・ミスの回数
L1D_CACHE_REFILL	L1D キャッシュ・ミスの回数
L1D_CACHE_REFILL_DM	デマンド・アクセスに起因する L1D キャッシュ・ミスの回数
L1D_CACHE_REFILL_PRF	プリフェッチ・アクセスに起因する L1D キャッシュ・ミス回数
L1D_CACHE_REFILL_HWPRF	ハードウェア・プリフェッチ・アクセスに起因する L1D キャッシュ・ミス回数
L1D_CACHE_WB	L1D キャッシュからのライトバックの回数
L1_MISS_WAIT	L1D キャッシュ・ミス処理のサイクル当たりのインフライト数の積算値 (= 1 サイクル毎に L1D キャッシュの MIB 使用数を積算した値)
L2D_CACHE_REFILL	L2 キャッシュ・ミスの回数
L2D_CACHE_REFILL_DM	デマンド・フローに起因する L2 キャッシュ・ミスの回数
L2D_CACHE_REFILL_PRF	プリフェッチ・フローに起因する L2 キャッシュ・ミスの回数
L2D_CACHE_REFILL_HWPRF	プリフェッチ・フローの内、ハードウェア・プリフェッチに起因する L2 キャッシュ・ミスの回数
L2D_CACHE_WB	L2 キャッシュからのライトバック・回数
L2_MISS_WAIT	L2 キャッシュ・ミス処理のサイクル当たりのインフライト数の積算値 (= 1 サイクル毎に L2 キャッシュの MIB 使用数を積算した値)
L1I_TLB_REFILL	L1-ITLB ミスの回数
L1D_TLB_REFILL	L1-DTLB ミスの回数
L2I_TLB_REFILL	L2-ITLB ミスの回数
L2D_TLB_REFILL	L2-DTLB ミスの回数
EFFECTIVE_INST_SPEC	MOVPRFX 命令を除く、コミットしたアーキテクチャ命令数
BR_PRED	コミットした分岐命令数
CPU_CYCLES	PE のサイクル数

Table 14-4 の Event を利用することで、プログラム実行時のハードウェアの性能指標を算出することができる。Table 14-5 にまとめる

**Table 14-5 Method to Calculate Hardware Performance Indicators at Program Execution**

指標	算出式
Cycles per Instruction (CPI)	CPU_CYCLES / EFFECTIVE_INST_SPEC
分岐予測ミス率	BR_MIS_PRED / EFFECTIVE_INST_SPEC
L1I キャッシュ・ミス率	L1I_CACHE_REFILL / EFFECTIVE_INST_SPEC
L1D キャッシュ・ミス率	L1D_CACHE_REFILL / EFFECTIVE_INST_SPEC
デマンド・アクセスに起因する L1D キャッシュ・ミス率	L1D_CACHE_REFILL_DM / EFFECTIVE_INST_SPEC
プリフェッч・アクセスに起因する L1D キャッシュ・ミス率	L1D_CACHE_REFILL_PRF / EFFECTIVE_INST_SPEC
ハードウェア・プリフェッチが生成したプリフェッチ・アクセスに起因する L1D キャッシュ・ミス率	L1D_CACHE_REFILL_HWPRF / EFFECTIVE_INST_SPEC
ソフトウェア・プリフェッチ・アクセスに起因する L1D キャッシュ・ミス率	(L1D_CACHE_REFILL_PRF - L1D_CACHE_REFILL_HWPRF) / EFFECTIVE_INST_SPEC
L2 キャッシュ・ミス率	L2D_CACHE_REFILL / EFFECTIVE_INST_SPEC
デマンド・フローに起因する L2 キャッシュ・ミス率	L2D_CACHE_REFILL_DM / EFFECTIVE_INST_SPEC
プリフェッチ・フローに起因する L2 キャッシュ・ミス率	L2D_CACHE_REFILL_PRF / EFFECTIVE_INST_SPEC
ハードウェア・プリフェッチが生成したプリフェッチ・フローに起因する L2 キャッシュ・ミス率	L2D_CACHE_REFILL_HWPRF / EFFECTIVE_INST_SPEC
ソフトウェア・プリフェッチが生成したプリフェッチ・フローに起因する L2 キャッシュ・ミス率	(L2D_CACHE_REFILL_PRF - L2D_CACHE_REFILL_HWPRF) / EFFECTIVE_INST_SPEC
L1D キャッシュ・ミス処理の平均レイテンシ	L1_MISS_WAIT / L1D_CACHE_REFILL
L2 キャッシュ・ミス処理の平均レイテンシ	L2_MISS_WAIT / L2D_CACHE_REFILL
L1D キャッシュ・ミス処理の平均アウトスタンディング数	L1_MISS_WAIT / CPU_CYCLES
L2 キャッシュ・ミス処理の平均アウトスタンディング数	L2_MISS_WAIT / CPU_CYCLES
L1-ITLB ミス率	L1I_TLB_REFILL / EFFECTIVE_INST_SPEC
L1-DTLB ミス率	L1D_TLB_REFILL / EFFECTIVE_INST_SPEC
L2-ITLB ミス率	L2I_TLB_REFILL / EFFECTIVE_INST_SPEC
L2-DTLB ミス率	L2D_TLB_REFILL / EFFECTIVE_INST_SPEC
L1D キャッシュと L2 キャッシュ間の双方向実効 bandwidth	(L1D_CACHE_REFILL + L1D_CACHE_WB) * 256 * Processor frequency / CPU_CYCLES
L2 キャッシュとメモリ間の双方向実効 bandwidth	(L2D_CACHE_REFILL + L2D_CACHE_WB) * 256 * Processor frequency / CPU_CYCLES

## 14.4. Cycle Accounting

プロセッサの性能指標の一つの Cycle Per Instruction (CPI) は、プロセッサが 1 命令を実行するのに費やした平均的な CPU サイクルである。この CPI は、命令を実行するための様々なオペレーション・フローの処理時間、例えば演算やメモリ・アクセスの時間、が積み重なったものと考

えることができる。CPI をこれらの個別の処理時間の積み上げとして表すことを Cycle Accounting と呼ぶ。A64FX は Cycle Accounting のための Event を実装している。Table 14-6 にまとめる。これらの Event も Instruction Mix で使われる Event と同様に計測する対象に包含関係がある。”Other”は Event を組み合わせて算出できる項目であり、算出式を Table 14-7 にまとめた。

Table 14-6 Performance Events for Cycle Accounting

Cycle Accounting のための Event	Event のカウント対象
CPU_CYCLES	CPU クロック・サイクル
0INST_COMMIT	命令コミット数が “0” であるサイクル
LD_COMP_WAIT	CSE の最も古い命令がメモリ・アクセス完了待ちでコミットできないサイクル
LD_COMP_WAIT_EX	LD_COMP_WAIT のうち、Base inst. に属する命令が要因であるサイクル
LD_COMP_WAIT_L2_MISS	LD_COMP_WAIT のうち、L2 キャッシュ・ミス中のサイクル
LD_COMP_WAIT_L2_MISS_EX	LD_COMP_WAIT_L2_MISS のうち、Base inst. に属する命令が要因であるサイクル
Other (ld_comp_wait_l2_miss_fn)	LD_COMP_WAIT_L2_MISS のうち、SIMD&FP と SVE 命令に属する命令が要因であるサイクル
LD_COMP_WAIT_L1_MISS	LD_CIMP_WAIT のうち、L1D キャッシュ・ミスかつ、L2 キャッシュ・ヒット中のサイクル (厳密には、L2 キャッシュ・ミス時の L2 キャッシュ・ミスが確定するまでのサイクルも含まれる)
LD_COMP_WAIT_L1_MISS_EX	LD_COMP_WAIT_L1_MISS のうち、Base inst. に属する命令が要因であるサイクル
Other (ld_comp_wait_l1_miss_fn)	LD_COMP_WAIT_L1_MISS のうち、SIMD&FP と SVE 命令に属する命令が要因であるサイクル
LD_COMP_WAIT_PFP_BUSY	LD_COMP_WAIT のうち、L2 キャッシュのプリフェッチ処理リソース不足のため、メモリ・メモリアクセス命令がコミットできないサイクル (プロフェッヂ・フローが処理できず、フローの生成元の命令がコミットできない事象を表す)
LD_COMP_WAIT_PFP_BUSY_EX	LD_COMP_WAIT_PFP_BUSY のうち、Base inst. に属する命令が要因であるサイクル
LD_COMP_WAIT_PFP_BUSY_SWPF	LD_COMP_WAIT_PFP_BUSY のうち、ソフトウェア・プリフェッヂ命令が要因であるサイクル
Other (ld_comp_wait_pfp_busy_fn)	LD_COMP_WAIT_PFP_BUSY のうち、SIMD&FP と SVE 命令が要因であるサイクル
Other (ld_comp_wait_l1_hit)	LD_COMP_WAIT のうち、L1D キャッシュ・ヒット中のサイクル (厳密には、L1D キャッシュ・ミス時の L1D キャッシュ・ミスが確定するまでのサイクルも含まれる)

Cycle Accounting のための Event	Event のカウント対象
Other (ld_comp_wait_ll_hit_ex)	ld_comp_wait_ll_hit のうち、Base inst. に属する命令が要因であるサイクル
Other (ld_comp_wait_ll_hit_fl)	ld_comp_wait_ll_hit のうち、SIMD&FP と SVE 命令に属する命令が要因であるサイクル
EU_COMP_WAIT	CSE の最も古い命令が演算完了待ちでコミットできないサイクル
FL_COMP_WAIT	EU_COMP_WAIT のうち、SIMD&FP と SVE 命令に属する命令が要因であるサイクル
Other (ex_comp_wait)	EU_COMP_WAIT のうち、Base inst. に属する命令が要因であるサイクル
BR_COMP_WAIT	CSE の最も古い命令が分岐命令で、分岐方向の確定待ちでコミットできないサイクル
ROB_EMPTY	CSE が空のために命令がコミットできないサイクル（命令がデコードステージ以降に存在していない状態）
ROB_EMPTY_STQ_BUSY	CSE が空、かつ Virtual SP がフルの状態のために命令がコミットできないサイクル (Virtual SP がフルのためデコードが止まっている状態)
WFE_WFI_CYCLE	WFE 命令、または WFI により PE の動作が停止しているサイクル (並列プログラムでは同期待ちの時間として現れる)
Other (rob_empty_not_stq_busy)	その他の要因で CSE が空のために命令がコミットできないサイクル (主に、命令フェッチ待ちの時間として現れる)
UOP_ONLY_COMMIT	$\mu$ OP 命令のみがコミットされたサイクル (2 $\mu$ OP 命令以上にデコードされたアーキテクチャ命令においては、最後の $\mu$ OP 命令のコミットがアーキテクチャ命令としてのコミットを意味するため、 $\mu$ OP 命令のみのコミットという状態がある)
SINGLE_MOVPRFX_COMMIT	Pack されなかった MOVPRFX 命令のみがコミットしたサイクル
Other (0inst_commit_other)	その他の要因で命令がコミットできなったサイクル
1INST_COMMIT	命令コミット数が "1" であるサイクル
2INST_COMMIT	命令コミット数が "2" であるサイクル
3INST_COMMIT	命令コミット数が "3" であるサイクル
4INST_COMMIT	命令コミット数が "4" であるサイクル
包含関係がない Event	
LD_COMP_WAIT_EX	LD_COMP_WAIT のうち、Base inst. に属する命令が要因であるサイクル

**Table 14-7 Formulas for Other (Cycle Accounting)**

項目	算出式
ld_comp_wait_l2_miss_f1	LD_COMP_WAIT_L2_MISS - LD_COMP_WAIT_L2_MISS_EX
ld_comp_wait_l1_miss_f1	LD_COMP_WAIT_L1_MISS - LD_COMP_WAIT_L1_MISS_EX
ld_comp_wait_pfp_busy_f1	LD_COMP_WAIT_PFP_BUSY -(LD_COMP_WAIT_PFP_BUSY_EX + LD_COMP_WAIT_PFP_BUSY_SWPF)
ld_comp_wait_l1_hit	LD_COMP_WAIT -(LD_COMP_WAIT_L2_MISS + LD_COMP_WAIT_L1_MISS + LD_COMP_WAIT_PFP_BUSY)
ld_comp_wait_l1_hit_ex	LD_COMP_WAIT_EX -(LD_COMP_WAIT_L2_MISS_EX + LD_COMP_WAIT_L1_MISS_EX + LD_COMP_WAIT_PFP_BUSY_EX)
ld_comp_wait_l1_hit_f1	ld_comp_wait_l1_hit - ld_comp_wait_l1_hit_ex
ex_comp_wait	EU_COMP_WAIT - FL_COMP_WAIT
rob_empty_not_stq_busy	ROB_EMPTY - (ROB_EMPTY_STQ_BUSY + WFE_WFI_CYCLE)
0inst_commit_other	0INST_COMMIT -(UOP_ONLY_COMMIT + SINGLE_MOVPRFX_COMMIT + LD_COMP_WAIT + EU_COMP_WAIT + BR_COMP_WAIT + ROB_EMPTY)

# 15. リソース一覧

---

この章では A64FX のハードウェア・リソースを一覧にまとめる。

**Table 15-1 Out-of-Order Resources**

Resource	Quantity of Resource		
Commit stack entry (CSE)	128 entries		
Group ID (GID)	32 entries		
General-purpose physical register (GPR)	96 entries	Architecture register	32 entries
		Renaming register	64 entries
Floating-point physical register (FPR)	128 entries	Architecture register	32 entries
		Renaming register	96 entries
Predicate physical register (PPR)	48 entries	Architecture register	16 entries
		Renaming register	32 entries
Reservation station for EAG (RSA)	10 entries x 2 (split)		
Reservation station for EXE (RSE) (shared by Integer, SIMD&FP, SVE)	20 entries x 2 (split)		
Reservation station for branch (RSBR)	19 entries		
Temporary operand register (TOR)	3 entries		
Fetch port (FP)	Virtual	160 entries	
	Real	40 entries	
Store port (SP)	Virtual	192 entries	
	Real	24 entries	
Write buffer (WB)	8 entries		

**Table 15-2 Resources for Branch Misprediction Mechanism**

Resource	Quantity of Resource
Instruction Buffer (IBUFF)	6 entries
Small Taken Chain Predictor (S-TCP)	4 entries
Loop Prediction Table (LPT)	8 entries
Branch Weight Table (BWT)	2,048 entries
Branch Target Buffer (BTB)	2,048 entries (4-way set associative)
Return Address Stack (RAS)	8 entries

**Table 15-3 Resources for Memory Management Unit**

Resource	Quantity of Resource
L1-ITLB	16 entries (full associative)
L1-DTLB	16 entries (full associative)
L2-ITLB	1,024 entries (4-way set associative)
L2-DTLB	1,024 entries (4-way set associative)
Translation Table Cache	16 entries (full associative)

**Table 15-4 Resources for L1/L2 Cache**

Resource	Quantity of Resource
L1I cache	64 KiB (4-way set associative)
L1D cache	64 KiB (4-way set associative)
L2 unified cache	8 MiB (16-way set associative)
L1I MIB	3 entries/core
L1D MIB	12 entries/core
L1D MOB	4 entries/core
L2 MIB	256 entries/CMG
L2 Store lock register	244 entries/CMG

## 16. 命令属性／レイテンシ一覧

A64FX がサポートする全命令のレイテンシ一覧を ARMv8 (Table 16-1)、ARMv8 SIMD&FP (Table 16-2)、SVE (Table 16-3) に分けてそれぞれ示す。

各表の列項目について説明する。

- Instruction, Alias  
命令のリストである。Alias 命令は元になる命令のサブセットとして表記している。
- Control option  
同一命令において、異なるハードウェアの動作となるときの条件を示している。基本的にアセンブラ・シンタックスで条件を表現している。  
Variant を区別するときは、ディステイネーション・オペランドのレジスタ・サイズで表記している。  
ディステイネーション・オペランドで区別できないときは、ソース・オペランドで表記している。
- VL  
Control option に VL が影響するときに区別を表記している。空欄であるときはハードウェアの動作は VL の影響を受けないことを意味している。
- Number of μOP  
デコード時に分解される μOP 命令の数である。μOP 命令については 4.1 章を参照のこと。
- Sequential decode  
シーケンシャル・デコードの対象命令であることを示す。シーケンシャル・デコードについては 4.1 章を参照のこと。
- Pre-sync, Post-sync  
Pre-sync および Post-sync 制御の対象命令であることを示す。Sync 制御については 4.7 章を参照のこと。
- Pack  
MOVPRFX 命令にて修飾されたときに Pack 处理が可能な命令を示す。Pack 处理については 4.3 章を参照のこと。
- Extra μOP  
MOVPRFX 命令による修飾、かつ Merging predication のときに μOP 命令が追加される命令であることを示す。Merging predication については 6.5.1 章を参照のこと。
- Blocking  
命令実行時にブロッキング制御される命令であることを示す。  
パイプライン・ブロッキングされる命令は”P”，演算ブロッキングされる命令は”E”と表記している。ブロッキング制御については 6.3 章を参照のこと。
- Latency  
命令の実行レイテンシを表す。基本的には μOP 命令単位で表記している。  
ロード命令においては、L1D キャッシュ・ヒット時のレイテンシとしている。  
また、ロード／ストア・ステージにおけるオペレーション・フロー分割はアクセスの性質、例えばアドレス・アライメントやデータ長などに依存するため、本一覧では処理に必須の最初のフローのみを計上している。  
表記ルールは以下の通りである。
  - μOP 命令間の分離記号は “/” である。
  - μOP 命令間に依存関係があるときは、先行する依存対象 μOP 命令の相対位置を [ ] で囲んだ左上付き表記で表す。  
例えば、”1 / 2 / [1,2]4 / 2” という表記があれば、3 番目の μOP 命令の入力は一つ前と二つ前の μOP 命令の出力であることを示している。
  - () 表記はグループ化を表す。さらに、()xN 表記はグループの展開表記であり、() 内を N 回展開すること意味する。  
例えば、”(1 / 2) x 3” は “1 / 2 / 1 / 2 / 1 / 2” に、”1 / (2 / [1]4) x 2” は “1 / 2 / [1]4 / 2 / [1]4” に展開される。
  - 複雑な依存関係がある命令は、位置表記もグループ化する。  
例えば、”1 / [1]2 / [2]2 / [3]2” は、先頭のレイテンシ 1 の μOP 命令に後続のすべての μOP 命令が依存している。  
このとき、1 / [1,2,3](2) x 3 とグループ化して展開表記とする。

$\mu$ OP 命令には複数オペレーション・フローに分解されるものがある。複数オペレーション・フローが組み合わさったときの表記ルールは以下の通りである。マルチ・オペレーションについては 4.2 章を参照のこと。

また、オペレーション・フロー間の依存関係は前後フローのみのため、オペレーション・フローにおける依存関係の位置表示はない。

- デコード・ステージにて分解されたオペレーション・フローに依存関係がなく、複数のパイプラインで独立して実行されるときは、”区切りで並べて表記する。
- デコード・ステージにて分解されたオペレーション・フローに依存関係がありながらも、複数のパイプラインで実行されるときは、”区切りで連結して表記する。
- 実行ステージにて分解されたオペレーション・フローに依存関係があり、オペレーション・フローが逐次に実行されるときは、”+”区切りで連結して表記する。
- Pipe( ) 関数表記は Gather load / Scatter store, Multiple structures load / store のための特別な表記である。

Pipe(L, N) はレイテンシ L のフローを、N 回連続して毎サイクル発行することを表している。フロー間には依存がないためパイプライン実行される。

#### ● Pipeline

オペレーション・フローを実行するパイプラインを表す。実行パイプラインについては 6.2 章を参照のこと。

表記ルールはレイテンシ表記を基本とし、以下のルールを追加する。

- オペレーション・フローを実行可能なパイプラインが複数あるときは、\*(ワイルドカード) と |(論理和) を使って表記する。  
例えば、EX\* | EAG\* は EX(整数演算系) パイプラインと EAG(アドレス演算系) のいずれのパイプラインでも実行できることを表している。  
また、(EXA + EXA) | (EXB + EXB) は、オペレーション・フローの 1 番目と 2 番目に依存があり、両方とも EXA パイプラインで実行、または両方とも EXB パイプラインで実行することを表している。
- いくつかのオペレーション・フローの組み合わせでは、バイパス・ペナルティが静的に加えられる。  
このようなときは、バイパスが発生するパイプライン間に ”+NULL+” を挿入し、かつレイテンシ表記も同じ位置にバイパス・ペナルティのレイテンシが表記されている。  
例えば、レイテンシ表記が ”1+3+6” でパイプライン表記が ”EXA + NULL + FLA” であるときは、中央の 3 サイクルがバイパス・ペナルティである。
- Pipe( ) 関数表記はレイテンシ表記と同様に、Gather load / Scatter store, Multiple structures load / store のための特別な表記である。

Pipe(P, N) はパイプライン P に N 回フローが発行されることを表している。

なお、Gather load / Scatter store は一つのオペレーション・フローが EAGA と EAGB の両方を使用するため、使用するパイプラインは ”EAGA & EAGB” と表記している。

#### ● Number of FP

ロード／ストア命令に割り当てられる Fetch Port 数である。Fetch Port については 7.3 章を参照のこと。

#### ● Number of SP

ロード／ストア命令に割り当てられる Store Port 数である。Fetch Port については 7.3 章を参照のこと。

#### ● FLOPS

命令の Performance Event で数え上げられる Element あたりの浮動小数点演算数である。

本項目が空欄であるときは ”0” FLOPS として扱われる。Performance Event での FLOPS 算出については 14.2 章を参照のこと。

## 16.1. ARMv8 Base instructions

Table 16-1 Instruction Attributes/Latency (ARMv8)

Instruction	Alias	Control option	# of $\mu$ OP	Seq. decode	Pre-sync	Post-sync	Blocking	Latency	Pipeline	# of FP	# of SP
ADC			1					1	EX*		
ADCS			1					1	EX*		
ADD (extended register)		<amount> = 0 && ( If sf = 0 Then <extend> = {LSL UXTW UXTX SXTW SXTX} Else <extend> = {UXTX SXTX} )	1					1	EX*   EAG*		
			1				P	1+1	(EXA + EXA)   (EXB + EXB)		
ADD (immediate)	MOV (to/from SP)		1					1	EX*   EAG*		
			1					1	EX*   EAG*		
ADD (shifted register)		<amount> = 0	1					1	EX*   EAG*		

Instruction	Alias	Control option	# of µOP	Seq. decode	Pre-sync	Post-sync	Blocking	Latency	Pipeline	# of FP	# of SP
		<amount> = [1-4] && <shift>=LSL	1				P	1+1	(EXA + EXA)   (EXB + EXB)		
			1				P	2+1	(EXA + EXA)   (EXB + EXB)		
ADDS (extended register)	CMN (extended register)	<amount> = 0	1					1	EX*		
			1				P	1+1	(EXA + EXA)   (EXB + EXB)		
		<amount>=0 && (If sf = 0 Then <extend>= {LSL UXTW UXTX SXTW SXTX} Else <extend> = {UXTX SXTX})	1					1	EX*		
			1				P	1+1	(EXA + EXA)   (EXB + EXB)		
ADDS (immediate)	CMN (immediate)		1					1	EX*		
			1					1	EX*		
ADDS (shifted register)	CMN (shifted register)	<amount> = 0	1					1	EX*		
		<amount> = [1-4] && <shift> = LSL	1				P	1+1	(EXA + EXA)   (EXB + EXB)		
			1				P	2+1	(EXA + EXA)   (EXB + EXB)		
		<amount> = 0	1					1	EX*		
		<amount> = [1-4] && <shift> = LSL	1				P	1+1	(EXA + EXA)   (EXB + EXB)		
			1				P	2+1	(EXA + EXA)   (EXB + EXB)		
ADR			1					1	EAGB		
ADRP			1					1	EAGB		
AND (immediate)			1					1	EX*   EAG*		
AND (shifted register)		<amount> = 0	1					1	EX*   EAG*		
			1				P	2+1	(EXA + EXA)   (EXB + EXB)		
ANDS (immediate)	TST (immediate)		1					1	EX*		
			1					1	EX*		
ANDS (shifted register)	TST (shifted register)	<amount> = 0	1					1	EX*		
			1				P	2+1	(EXA + EXA)   (EXB + EXB)		
		<amount> = 0	1					1	EX*		
			1				P	2+1	(EXA + EXA)   (EXB + EXB)		
ASRV	ASR (register)		1					2	EX*		
B.cond			1					NA	BR		
B			1					NA	BR		
BFM	BFI		4	✓				2 / [1]1 / 1 / [1,2]1	EX* / EX* / EX* / EX*		
	BFXIL		4	✓				2 / [1]1 / 1 / [1,2]1	EX* / EX* / EX* / EX*		
			4	✓				2 / [1]1 / 1 / [1,2]1	EX* / EX* / EX* / EX*		
BIC (shifted register)		<amount> = 0	1					1	EX*   EAG*		
			1				P	2+1	(EXA + EXA)   (EXB + EXB)		
BICS (shifted register)		<amount> = 0	1					1	EX*		
			1				P	2+1	(EXA + EXA)   (EXB + EXB)		
BL			1					1	EAGB, BR		

Instruction	Alias	Control option	# of μOP	Seq. decode	Pre-sync	Post-sync	Blocking	Latency	Pipeline	# of FP	# of SP
BLR			1					1, NA, NA	EAGB, EXA, BR		
BR			1					1, NA	EXA, BR		
BRK			2	✓	✓	✓		NA / NA	/		
CAS{[A AL L]}			3	✓				1 / 5;1 / [2]1	EAG* / EAGA; EXA / EXA	1	1
CAS{[A AL L}B			3	✓				1 / 5;1 / [2]1	EAG* / EAGA; EXA / EXA	1	1
CAS{[A AL L}H			3	✓				1 / 5;1 / [2]1	EAG* / EAGA; EXA / EXA	1	1
CASP{[A AL L]}			7	✓				1 / 5;[1]1 / 1 / [2]1 / 5;[4]1 / 1 / [2]1	EAG* / EAGA; EXA / EXA / EAG* / EAGA; EXA / EXA / EAG*	2	2
CBNZ			1					1	EX*		
CBZ			1					1	EX*		
CCMN (immediate)			1				P	1+1	(EXA + EXA)   (EXB + EXB)		
CCMN (register)			1				P	1+1	(EXA + EXA)   (EXB + EXB)		
CCMP (immediate)			1				P	1+1	(EXA + EXA)   (EXB + EXB)		
CCMP (register)			1				P	1+1	(EXA + EXA)   (EXB + EXB)		
CLREX			2	✓				NA / NA	/ EAGA	1	
CLS			1					2	EX*		
CLZ			1					2	EX*		
CRC32B			1				E	10	EXB		
CRC32H			1				E	10	EXB		
CRC32W			1				E	12	EXB		
CRC32X			1				E	20	EXB		
CRC32CB			1				E	10	EXB		
CRC32CH			1				E	10	EXB		
CRC32CW			1				E	12	EXB		
CRC32CX			1				E	20	EXB		
CSEL			1					1	EX*		
CSINC	CINC		1					1	EX*		
	CSET		1					1	EX*		
			1					1	EX*		
CSINV	CINV		1					1	EX*		
	CSETM		1					1	EX*		
			1					1	EX*		
CSNEG	CNEG		1					1	EX*		
			1					1	EX*		
DCPS1			2	✓	✓	✓		NA / NA	/		
DCPS2			2	✓	✓	✓		NA / NA	/		
DCPS3			2	✓	✓	✓		NA / NA	/		
DMB			2	✓				NA / NA	/ EAGA	1	
DRPS			2	✓	✓	✓		NA / NA	/		
DSB			2	✓				NA / NA	/ EAGA	1	

Instruction	Alias	Control option	# of µOP	Seq. decode	Pre-sync	Post-sync	Blocking	Latency	Pipeline	# of FP	# of SP
EON (shifted register)		<amount> = 0	1					1	EX*   EAG*		
			1				P	2+1	(EXA + EXA)   (EXB + EXB)		
EOR (immediate)			1					1	EX*   EAG*		
EOR (shifted register)		<amount> = 0	1					1	EX*   EAG*		
			1				P	2+1	(EXA + EXA)   (EXB + EXB)		
ERET			2	✓	✓	✓		NA / NA	/		
EXTR	ROR (immediate)		1					2	EX*		
			3	✓				2 / 2 / [1,2]1	EX* / EX* / EX*		
HINT	NOP		1					NA			
	YIELD		6	✓	✓	✓		NA / NA / NA / NA / NA / NA			
	WFE		2	✓	✓	✓		NA / NA	/		
	WFI		2	✓	✓	✓		NA / NA	/		
	SEV		2	✓	✓	✓		NA / NA	/		
	SEVL		2	✓	✓	✓		NA / NA	/		
HLT			2	✓	✓	✓		NA / NA	/		
HVC			2	✓	✓	✓		NA / NA	/		
ISB			2	✓		✓		NA / NA	/ EAGA	1	
LDADD{ A AL L}			4	✓				1 / [1]5 / [1]1 / [1]NA	EAG* / EAGA / EXA / EXA	1	1
LDADD{ A AL L}B			4	✓				1 / [1]5 / [1]1 / [1]NA	EAG* / EAGA / EXA / EXA	1	1
LDADD{ A AL L}H			4	✓				1 / [1]5 / [1]1 / [1]NA	EAG* / EAGA / EXA / EXA	1	1
LDAR			1					5	EAGA	1	
LDARB			1					5	EAGA	1	
LDARH			1					5	EAGA	1	
LDAXP			3	✓				1 / [1]5 / [2]5	EAG* / EAGA / EAGA	3	
LDAXR			1					5	EAGA	1	
LDAXB			1					5	EAGA	1	
LDAXRH			1					5	EAGA	1	
LDCLR{ A AL L}			4	✓				1 / [1]5 / [1]1 / [1]NA	EAG* / EAGA / EXA / EXA	1	1
LDCLR{ A AL L}B			4	✓				1 / [1]5 / [1]1 / [1]NA	EAG* / EAGA / EXA / EXA	1	1
LDCLR{ A AL L}H			4	✓				1 / [1]5 / [1]1 / [1]NA	EAG* / EAGA / EXA / EXA	1	1
LDEOR{ A AL L}			4	✓				1 / [1]5 / [1]1 / [1]NA	EAG* / EAGA / EXA / EXA	1	1
LDEOR{ A AL L}B			4	✓				1 / [1]5 / [1]1 / [1]NA	EAG* / EAGA / EXA / EXA	1	1
LDEOR{ A AL L}H			4	✓				1 / [1]5 / [1]1 / [1]NA	EAG* / EAGA / EXA / EXA	1	1
LDLAR			1					5	EAGA	1	
LDLARB			1					5	EAGA	1	
LDLARH			1					5	EAGA	1	
LDNP			2					5 / 5	EAG* / EAG*	2	
LDP		Post-index	3					5 / 5 / 1	EAG* / EAG* / EX*   EAG*	2	
		Pre-index	3					5 / 5 / 1	EAG* / EAG* / EX*   EAG*	2	

Instruction	Alias	Control option	# of μOP	Seq. decode	Pre-sync	Post-sync	Blocking	Latency	Pipeline	# of FP	# of SP
		Signed offset	2					5 / 5	EAG*/EAG*	2	
LDPSW		Post-index	3					5 / 5 / 1	EAG* / EAG* / EX*  EAG*	2	
		Pre-index	3					5 / 5 / 1	EAG* / EAG* / EX*  EAG*	2	
		Signed offset	2					5 / 5	EAG*/EAG*	2	
LDR (immediate)		Post-index	2					5 / 1	EAG* / EX*  EAG*	1	
		Pre-index	2					5 / 1	EAG* / EX*  EAG*	1	
		Unsigned offset	1					5	EAG*	1	
LDR (literal)			1					5	EAGB	1	
LDR (register)			1					5	EAG*	1	
LDRB (immediate)		Post-index	2					5 / 1	EAG* / EX*  EAG*	1	
		Pre-index	2					5 / 1	EAG* / EX*  EAG*	1	
		Unsigned offset	1					5	EAG*	1	
LDRB (register)			1					5	EAG*	1	
LDRH (immediate)		Post-index	2					5 / 1	EAG* / EX*  EAG*	1	
		Pre-index	2					5 / 1	EAG* / EX*  EAG*	1	
		Unsigned offset	1					5	EAG*	1	
LDRH (register)			1					5	EAG*	1	
LDRSB (immediate)		Post-index	2					5 / 1	EAG* / EX*  EAG*	1	
		Pre-index	2					5 / 1	EAG* / EX*  EAG*	1	
		Unsigned offset	1					5	EAG*	1	
LDRSB (register)			1					5	EAG*	1	
LDRSH (immediate)		Post-index	2					5 / 1	EAG* / EX*  EAG*	1	
		Pre-index	2					5 / 1	EAG* / EX*  EAG*	1	
		Unsigned offset	1					5	EAG*	1	
LDRSH (register)			1					5	EAG*	1	
LDRSW (immediate)		Post-index	2					5 / 1	EAG* / EX*  EAG*	1	
		Pre-index	2					5 / 1	EAG* / EX*  EAG*	1	
		Unsigned offset	1					5	EAG*	1	
LDRSW (literal)			1					5	EAGB	1	
LDRSW (register)			1					5	EAG*	1	
LDSET{ A AL L}			4	✓				1 / [1]5 / [1]1 / [1]NA	EAG* / EAGA / EXA / EXA	1	1
LDSET{ A AL L}B			4	✓				1 / [1]5 / [1]1 / [1]NA	EAG* / EAGA / EXA / EXA	1	1
LDSET{ A AL L}H			4	✓				1 / [1]5 / [1]1 / [1]NA	EAG* / EAGA / EXA / EXA	1	1
LDSMAX{ A AL L}			4	✓		// P /	1 / [1]5 / 1+[1]1 / [1]NA	EAG* / EAGA / EXA+EXA / EXA	1	1	
LDSMAX{ A AL L}			4	✓		// P /	1 / [1]5 / 1+[1]1 / [1]NA	EAG* / EAGA / EXA+EXA / EXA	1	1	
LDSMAX{ A AL L}			4	✓		// P /	1 / [1]5 / 1+[1]1 / [1]NA	EAG* / EAGA / EXA+EXA / EXA	1	1	
LDSMIN{ A AL L}			4	✓		// P /	1 / [1]5 / 1+[1]1 / [1]NA	EAG* / EAGA / EXA+EXA / EXA	1	1	
LDSMIN{ A AL L}B			4	✓		// P /	1 / [1]5 / 1+[1]1 / [1]NA	EAG* / EAGA / EXA+EXA / EXA	1	1	
LDSMIN{ A AL L}H			4	✓		// P /	1 / [1]5 / 1+[1]1 / [1]NA	EAG* / EAGA / EXA+EXA / EXA	1	1	

Instruction	Alias	Control option	# of μOP	Seq. decode	Pre-sync	Post-sync	Blocking	Latency	Pipeline	# of FP	# of SP
LDTR			1					5	EAG*	1	
LDTRB			1					5	EAG*	1	
LDTRH			1					5	EAG*	1	
LDTRSB			1					5	EAG*	1	
LDTRSH			1					5	EAG*	1	
LDTRSW			1					5	EAG*	1	
LDUMAX{ A AL L}			4	✓			// P /	1 / [1]5 / 1+[1]1 / [1]NA	EAG* / EAGA / EXA+EXA / EXA	1	1
LDUMAX{ A AL L}B			4	✓			// P /	1 / [1]5 / 1+[1]1 / [1]NA	EAG* / EAGA / EXA+EXA / EXA	1	1
LDUMAX{ A AL L}H			4	✓			// P /	1 / [1]5 / 1+[1]1 / [1]NA	EAG* / EAGA / EXA+EXA / EXA	1	1
LDUMIN{ A AL L}			4	✓			// P /	1 / [1]5 / 1+[1]1 / [1]NA	EAG* / EAGA / EXA+EXA / EXA	1	1
LDUMIN{ A AL L}B			4	✓			// P /	1 / [1]5 / 1+[1]1 / [1]NA	EAG* / EAGA / EXA+EXA / EXA	1	1
LDUMIN{ A AL L}H			4	✓			// P /	1 / [1]5 / 1+[1]1 / [1]NA	EAG* / EAGA / EXA+EXA / EXA	1	1
LDUR			1					5	EAG*	1	
LDURB			1					5	EAG*	1	
LDURH			1					5	EAG*	1	
LDURSB			1					5	EAG*	1	
LDURSH			1					5	EAG*	1	
LDURSW			1					5	EAG*	1	
LDXP			3	✓				1 / [1]5 / [2]5	EAG* / EAGA / EAGA	3	
LDXR			1					5	EAGA	1	
LDXRB			1					5	EAGA	1	
LDXRH			1					5	EAGA	1	
LSLV	LSL (register)		1					2	EX*		
LSRV	LSR (register)		1					2	EX*		
MADD	MUL		1					5	EXA		
			2					5 / [1]	EXA / EXA		
MOVK			1					1	EX*   EAG*		
MOVN	MOV (inverted wide immediate)		1					1	EX*   EAG*		
			1					1	EX*   EAG*		
MOVZ	MOV (wide immediate)		1					1	EX*   EAG*		
			1					1	EX*   EAG*		
MRS (注 1)			2	✓	✓						
MSR (immediate) (注 1)			2	✓		✓					
MSR (register) (注 1)			2	✓		✓					
MSUB	MNEG		2					5 / [1]	EXA / EXA		
			2					5 / [1]	EXA / EXA		
ORN (shifted register)	MVN	<amount> = 0	1					1	EX*   EAG*		
			1				P	2+1	(EXA + EXA)   (EXB + EXB)		

Instruction	Alias	Control option	# of μOP	Seq. decode	Pre-sync	Post-sync	Blocking	Latency	Pipeline	# of FP	# of SP
		<amount> = 0	1					1	EX*   EAG*		
			1				P	2+1	(EXA + EXA)   (EXB + EXB)		
ORR (immediate)	MOV (bitmask immediate)		1					1	EX*   EAG*		
			1					1	EX*   EAG*		
ORR (shifted register)	MOV (register)	<amount> = 0	1					1	EX*   EAG*		
			1				P	2+1	EX* + EX*		
		<amount> = 0	1					1	EX*   EAG*		
			1				P	2+1	(EXA + EXA)   (EXB + EXB)		
PRFM (immediate)			1					NA	EAG*	1	
PRFM (literal)			1					NA	EAGB	1	
PRFM (register)			1					NA	EAG*	1	
PRFM (unscaled offset)			1					NA	EAG*	1	
RBIT			1					1	EX*   EAG*		
RET			1					1	EXA		
REV	REV64		1					1	EX*   EAG*		
REV16			1					1	EX*   EAG*		
REV32			1					1	EX*   EAG*		
RORV	ROR (register)		1					2	EX*		
SBC	NGC		1					1	EX*		
			1					1	EX*		
SBCS	NGCS		1					1	EX*		
			1					1	EX*		
SVC			2	✓	✓	✓		NA / NA	/		
SBFM	ASR (immediate)	<shift> = 0	1					1	EX*		
			1					2	EX*		
	SBFIZ		1				P	2+1	(EXA + EXA)   (EXB + EXB)		
	SBFX		1				P	2+1	(EXA + EXA)   (EXB + EXB)		
	SXTB		1					1	EX*		
	SXTH		1					1	EX*		
	SXTW		1					1	EX*		
			1				P	2+1	(EXA + EXA)   (EXB + EXB)		
			1								
SDIV		sf = 0	1				E	n (9-26)	EXB		
		sf = 1	1				E	n (9-42)	EXB		
SMADDL	SMULL		1					5	EXA		
			2					5 / [1]1	EXA / EXA		
SMC			2	✓	✓	✓		NA / NA	/		
SMSUBL	SMNEGL		2					5 / [1]1	EXA / EXA		
			2					5 / [1]1	EXA / EXA		

Instruction	Alias	Control option	# of μOP	Seq. decode	Pre-sync	Post-sync	Blocking	Latency	Pipeline	# of FP	# of SP
SMULH			1					5	EXA		
STADD{ L}			4	✓				1 / [1]5 / [1]1 / [1]NA	EAG* / EAGA / EXA / EXA	1	1
STADD{ L}B			4	✓				1 / [1]5 / [1]1 / [1]NA	EAG* / EAGA / EXA / EXA	1	1
STADD{ L}H			4	✓				1 / [1]5 / [1]1 / [1]NA	EAG* / EAGA / EXA / EXA	1	1
STCLR{ L}			4	✓				1 / [1]5 / [1]1 / [1]NA	EAG* / EAGA / EXA / EXA	1	1
STCLR{ L}B			4	✓				1 / [1]5 / [1]1 / [1]NA	EAG* / EAGA / EXA / EXA	1	1
STCLR{ L}H			4	✓				1 / [1]5 / [1]1 / [1]NA	EAG* / EAGA / EXA / EXA	1	1
STEOR{ L}			4	✓				1 / [1]5 / [1]1 / [1]NA	EAG* / EAGA / EXA / EXA	1	1
STEOR{ L}B			4	✓				1 / [1]5 / [1]1 / [1]NA	EAG* / EAGA / EXA / EXA	1	1
STEOR{ L}H			4	✓				1 / [1]5 / [1]1 / [1]NA	EAG* / EAGA / EXA / EXA	1	1
STLLR			1					NA, NA	EAG*, EXA	1	1
STLLRB			1					NA, NA	EAG*, EXA	1	1
STLLRH			1					NA, NA	EAG*, EXA	1	1
STLR			1					NA, NA	EAG*, EXA	1	1
STLRB			1					NA, NA	EAG*, EXA	1	1
STLRH			1					NA, NA	EAG*, EXA	1	1
STLXP			7	✓				1 / 8;[1]1 / 1 / [2]1 / 8;[4]1 / 1 / [2]1	EAG* / EAGA; EXA / EXA / EAG* / EAGA; EXA / EXA / EAG*	2	2
STLXR			3	✓				1 / 8;1 / [2]NA	EAG* / EAGA; EXA / EXA	1	1
STLXRB			3	✓				1 / 8;1 / [2]NA	EAG* / EAGA; EXA / EXA	1	1
STLXRH			3	✓				1 / 8;1 / [2]NA	EAG* / EAGA; EXA / EXA	1	1
STNP			2					NA, NA / NA, NA	EXA, EAG* / EXA, EAG*	2	2
STP		Post-index	3					NA, NA / NA, NA / 1	EAG*, EXA / EAG*, EXA / EX*  EAG*	2	2
		Pre-index	3					NA, NA / NA, NA / 1	EAG*, EXA / EAG*, EXA / EX*  EAG*	2	2
		Signed offset	2					NA, NA / NA, NA	EAG*, EXA / EAG*, EXA	2	2
STR (immediate)		Post-index	2					NA, NA / 1	EAG*, EXA / EX*  EAG*	1	1
		Pre-index	2					NA, NA / 1	EAG*, EXA / EX*  EAG*	1	1
		Unsigned offset	1					NA, NA	EAG*, EXA	1	1
STR (register)			1					NA, NA	EAG*, EXA	1	1
STRB (immediate)		Post-index	2					NA, NA / 1	EAG*, EXA / EX*  EAG*	1	1
		Pre-index	2					NA, NA / 1	EAG*, EXA / EX*  EAG*	1	1
		Unsigned offset	1					NA, NA	EAG*, EXA	1	1
STRB (register)			1					NA, NA	EAG*, EXA	1	1
STRH (immediate)		Post-index	2					NA, NA / 1	EAG*, EXA / EX*  EAG*	1	1
		Pre-index	2					NA, NA / 1	EAG*, EXA / EX*  EAG*	1	1
		Unsigned offset	1					NA, NA	EAG*, EXA	1	1
STRH (register)			1					NA, NA	EAG*, EXA	1	1
STSET{ L}			4	✓				1 / [1]5 / [1]1 / [1]NA	EAG* / EAGA / EXA / EXA	1	1
STSET{ L}B			4	✓				1 / [1]5 / [1]1 / [1]NA	EAG* / EAGA / EXA / EXA	1	1

Instruction	Alias	Control option	# of μOP	Seq. decode	Pre-sync	Post-sync	Blocking	Latency	Pipeline	# of FP	# of SP
STSET{ L}H			4	✓				1 / <sup>[1]</sup> 5 / <sup>[1]</sup> 1 / <sup>[1]</sup> NA	EAG* / EAGA / EXA / EXA	1	1
STS MAX{ L}			4	✓				1 / <sup>[1]</sup> 5 / 1+ <sup>[1]</sup> 1 / <sup>[1]</sup> NA	EAG* / EAGA / EXA+EXA / EXA	1	1
STS MAX{ L}B			4	✓				1 / <sup>[1]</sup> 5 / 1+ <sup>[1]</sup> 1 / <sup>[1]</sup> NA	EAG* / EAGA / EXA+EXA / EXA	1	1
STS MAX{ L}H			4	✓				1 / <sup>[1]</sup> 5 / 1+ <sup>[1]</sup> 1 / <sup>[1]</sup> NA	EAG* / EAGA / EXA+EXA / EXA	1	1
STS MIN{ L}			4	✓				1 / <sup>[1]</sup> 5 / 1+ <sup>[1]</sup> 1 / <sup>[1]</sup> NA	EAG* / EAGA / EXA+EXA / EXA	1	1
STS MIN{ L}B			4	✓				1 / <sup>[1]</sup> 5 / 1+ <sup>[1]</sup> 1 / <sup>[1]</sup> NA	EAG* / EAGA / EXA+EXA / EXA	1	1
STS MIN{ L}H			4	✓				1 / <sup>[1]</sup> 5 / 1+ <sup>[1]</sup> 1 / <sup>[1]</sup> NA	EAG* / EAGA / EXA+EXA / EXA	1	1
STTR			1					NA, NA	EAG*, EXA	1	1
STTRB			1					NA, NA	EAG*, EXA	1	1
STTRH			1					NA, NA	EAG*, EXA	1	1
STU MAX{ L}			4	✓				1 / <sup>[1]</sup> 5 / 1+ <sup>[1]</sup> 1 / <sup>[1]</sup> NA	EAG* / EAGA / EXA+EXA / EXA	1	1
STU MAX{ L}B			4	✓				1 / <sup>[1]</sup> 5 / 1+ <sup>[1]</sup> 1 / <sup>[1]</sup> NA	EAG* / EAGA / EXA+EXA / EXA	1	1
STU MAX{ L}H			4	✓				1 / <sup>[1]</sup> 5 / 1+ <sup>[1]</sup> 1 / <sup>[1]</sup> NA	EAG* / EAGA / EXA+EXA / EXA	1	1
STU MIN{ L}			4	✓				1 / <sup>[1]</sup> 5 / 1+ <sup>[1]</sup> 1 / <sup>[1]</sup> NA	EAG* / EAGA / EXA+EXA / EXA	1	1
STU MIN{ L}B			4	✓				1 / <sup>[1]</sup> 5 / 1+ <sup>[1]</sup> 1 / <sup>[1]</sup> NA	EAG* / EAGA / EXA+EXA / EXA	1	1
STU MIN{ L}H			4	✓				1 / <sup>[1]</sup> 5 / 1+ <sup>[1]</sup> 1 / <sup>[1]</sup> NA	EAG* / EAGA / EXA+EXA / EXA	1	1
STUR			1					NA, NA	EAG*, EXA	1	1
STURB			1					NA, NA	EAG*, EXA	1	1
STURH			1					NA, NA	EAG*, EXA	1	1
STXP			7	✓				1 / 8; <sup>[1]</sup> 1 / 1 / <sup>[2]</sup> 1 / 8; <sup>[4]</sup> 1 / 1 / <sup>[2]</sup> 1	EAG* / EAGA; EXA / EXA / EAG* / EAGA; EXA / EXA / EAG*	2	2
STXR			3	✓				1 / 8;1 / <sup>[2]</sup> NA	EAG* / EAGA; EXA / EXA	1	1
STXRB			3	✓				1 / 8;1 / <sup>[2]</sup> NA	EAG* / EAGA; EXA / EXA	1	1
STXRH			3	✓				1 / 8;1 / <sup>[2]</sup> NA	EAG* / EAGA; EXA / EXA	1	1
SUB (extended register)		<amount> = 0 && ( If sf = 0 Then <extend> = {LSL UXTW UXTX SXTW SXTX} Else <extend> = {UXTX SXTX} )	1					1	EX*   EAG*		
			1				P	1+1	(EXA + EXA)   (EXB + EXB)		
SUB (immediate)			1					1	EX*   EAG*		
SUB (shifted register)	NEG (shifted register)	<amount> = [1-4] && <shift> = LSL	1				P	1+1	(EXA + EXA)   (EXB + EXB)		
		<amount> == 0	1					1	EX*   EAG*		
			1				P	2+1	(EXA + EXA)   (EXB + EXB)		
		<amount> = 0	1					1	EX*   EAG*		
		<amount> = [1-4] && <shift> = LSL	1				P	1+1	(EXA + EXA)   (EXB + EXB)		
			1				P	2+1	(EXA + EXA)   (EXB + EXB)		
SUBS (extended register)	CMP (extended register)	<amount> = 0	1					1	EX*		
			1				P	1+1	(EXA + EXA)   (EXB + EXB)		

Instruction	Alias	Control option	# of μOP	Seq. decode	Pre-sync	Post-sync	Blocking	Latency	Pipeline	# of FP	# of SP
		<amount> = 0 && (If sf = 0 Then <extend> = {LSL UXTW UXTX SXTW SXTX} Else <extend> = {UXTX SXTX})	1					1	EX*		
			1				P	1+1	(EXA + EXA)   (EXB + EXB)		
SUBS (immediate)	CMP (immediate)		1					1	EX*		
			1					1	EX*		
SUBS (shifted register)	CMP (shifted register)	<amount> = 0	1					1	EX*		
		<amount> = [1-4] && <shift> = LSL	1				P	1+1	(EXA + EXA)   (EXB + EXB)		
			1				P	2+1	(EXA + EXA)   (EXB + EXB)		
	NEGS	<amount> = 0	1					1	EX*		
		<amount> = [1-4] && <shift> = LSL	1				P	1+1	(EXA + EXA)   (EXB + EXB)		
			1				P	2+1	(EXA + EXA)   (EXB + EXB)		
		<amount> = 0	1					1	EX*		
		<amount> = [1-4] && <shift> = LSL	1				P	1+1	(EXA + EXA)   (EXB + EXB)		
			1				P	2+1	(EXA + EXA)   (EXB + EXB)		
SWP{A AL L}			1					NA, NA	EAGA, EXA	1	1
SWP{A AL L}B			1					NA, NA	EAGA, EXA	1	1
SWP{A AL L}H			1					NA, NA	EAGA, EXA	1	1
SYS	AT		1					NA, NA	EAGA, EXA	1	1
	DC		1					NA, NA	EAGA, EXA	1	1
	IC		1					NA, NA	EAGA, EXA	1	1
	TLBI		1					NA, NA	EAGA, EXA	1	1
			1					NA, NA	EAGA, EXA	1	1
SYSL			2	✓				NA / NA	/		
TBNZ			1					1	EX*		
TBZ			1					1	EX*		
UBFM	LSL (immediate)	<shift> = [1-4]	1					1	EX*		
			1					2	EX*		
	LSR (immediate)	<shift> = 0	1					1	EX*		
			1					2	EX*		
	UBFIZ		1				P	2+1	(EXA + EXA)   (EXB + EXB)		
	UBFX		1				P	2+1	(EXA + EXA)   (EXB + EXB)		
	UXTB		1					1	EX*		
	UXTH		1					1	EX*		
		If sf = 1 Then immr == '000000' && imms == '011111'	1					1	EX*		
UDIV		sf = 0	1				E	n (9-25)	EXB		
		sf = 1	1				E	n (9-41)	EXB		

Instruction	Alias	Control option	# of μOP	Seq. decode	Pre-sync	Post-sync	Blocking	Latency	Pipeline	# of FP	# of SP
UMADDL	UMULL		1					5	EXA		
			2					5 / [1]1	EXA / EXA		
UMSUBL	UMNEGL		2					5 / [1]1	EXA / EXA		
			2					5 / [1]1	EXA / EXA		
UMULH			1					5	EXA		

(注 1) MRS / MSR 命令はアクセスするシステム・レジスタにより制御が異なる。

## 16.2. ARMv8 SIMD&FP instructions

Table 16-2 Instruction Attributes/Latency (ARMv8 SIMD&FP)

Instruction	Alias	Control option	# of μOP	Seq. decode	Pre-sync	Post-sync	Blocking	Latency	Pipeline	# of FP	# of SP	FLOPS
ABS			1					4	FL*			
ADD (vector)			1					4	FL*			
ADDHN, ADDHN2			2	✓				4 / [1]6	FL* / FLB			
ADDP (scalar)			2	✓				6 / [1]4	FLA / FL*			
ADDP (vector)			3	✓				6 / 6 / [1,2]4	FLA / FLA / FL*			
ADDV			6	✓				4 / [1]4 / [1]6 / [1,2]4 / [1]4 / [1]4	FL* / FL* / FLA / FL* / FL* / FL*			
AESD			1				E	8	FLA			
AESE			1				E	8	FLA			
AESIMC			1				E	8	FLA			
AESMC			1				E	8	FLA			
AND (vector)			1					4	FL*			
BIC (vector, immediate)			1					4	FLA			
BIC (vector, register)			1					4	FL*			
BIF			1					1+4	FL* + FL*			
BIT			1					1+4	FL* + FL*			
BSL			1					1+4	FL* + FL*			
CLS (vector)			1					4	FLA			
CLZ (vector)			1					4	FLA			
CMEQ (register)			1					4	FL*			
CMEQ (zero)			1					4	FL*			
CMGE (register)			1					4	FL*			
CMGE (zero)			1					4	FL*			
CMGT (register)			1					4	FL*			
CMGT (zero)			1					4	FL*			
CMHI (register)			1					4	FL*			
CMHS (register)			1					4	FL*			
CMLE (zero)			1					4	FL*			
CMLT (zero)			1					4	FL*			
CMTST			1					4	FL*			
CNT			1					4	FLB			
DUP (element)	MOV (scalar)		1					6	FLA			
DUP (general)			1					1+3+6	EXA + NULL + FLA			
EOR (vector)			1					4	FL*			
EXT			1					6	FLA			
FABD			1					9	FL*			1
FABS (scalar)			1					4	FL*			

Instruction	Alias	Control option	# of μOP	Seq. decode	Pre-sync	Post-sync	Blocking	Latency	Pipeline	# of FP	# of SP	FLOPS
FABS (vector)			1					4	FL*			
FACGE			1					4	FL*			
FACGT			1					4	FL*			
FADD (scalar)			1					9	FL*			1
FADD (vector)			1					9	FL*			1
FADDP (scalar)			2	✓				6 / <sup>[1]</sup> 9	FLA / FL*			1
FADDP (vector)			3	✓				6 / 6 / <sup>[1,2]</sup> 9	FLA / FLA / FL*			1
FCADD			2					6 / <sup>[1]</sup> 9	FLA / FLB			1
FCCMP			1					4	FL*			
FCCMPE			1					4	FL*			
FCMEQ (register)			1					4	FL*			
FCMEQ (zero)			1					4	FL*			
FCMGE (register)			1					4	FL*			
FCMGE (zero)			1					4	FL*			
FCMGT (register)			1					4	FL*			
FCMGT (zero)			1					4	FL*			
FCMLA			3					6 / 6 / <sup>[1,2]</sup> 9	FLA / FLA / FL*			2
FCMLA (by element)			3					6 / 6 / <sup>[1,2]</sup> 9	FLA / FLA / FL*			2
FCMLE (zero)			1					4	FL*			
FCMLT (zero)			1					4	FL*			
FCMP			1					4	FL*			
FCMPE			1					4	FL*			
FCSEL			1					4	FL*			
FCVT			1					9	FL*			
FCVTAS (scalar)			1					9+1; 15	FLA + NULL ; EAG*	1	1	
FCVTAS (vector)			1					9	FL*			
FCVTAU (scalar)			1					9+1; 15	FLA + NULL ; EAG*	1	1	
FCVTAU (vector)			1					9	FL*			
FCVTL, FCVTL2		<Ta> = 4S <Ta> = 2D	2					6 / <sup>[1]</sup> 9	FLB / FL*			
			1					6	FLB			
FCVTMS (scalar)			1					9+1; 15	FLA + NULL ; EAG*	1	1	
FCVTMS (vector)			1					9	FL*			
FCVTMU (scalar)			1					9+1; 15	FLA + NULL ; EAG*	1	1	
FCVTMU (vector)			1					9	FL*			
FCVTN, FCVTN2			2					9 / <sup>[1]</sup> 6	FL* / FLA			
FCVTNS (scalar)			1					9+1; 15	FLA + NULL ; EAG*	1	1	
FCVTNS (vector)			1					9	FL*			
FCVTNU (scalar)			1					9+1; 15	FLA + NULL ; EAG*	1	1	
FCVTNU (vector)			1					9	FL*			

Instruction	Alias	Control option	# of µOP	Seq. decode	Pre-sync	Post-sync	Blocking	Latency	Pipeline	# of FP	# of SP	FLOPS
FCVTPS (scalar)			1					9+1; 15	FLA + NULL ; EAG*	1	1	
FCVTPS (vector)			1					9	FL*			
FCVTPU (scalar)			1					9+1; 15	FLA + NULL ; EAG*	1	1	
FCVTPU (vector)			1					9	FL*			
FCVTXN, FCVTXN2		Scalar	1					9	FL*			
		Vector	2					9 / [1]6	FL* / FLA			
FCVTZS (scalar, fixed-point)			1					9+1; 15	FLA + NULL ; EAG*	1	1	
FCVTZS (scalar, integer)			1					9+1; 15	FLA + NULL ; EAG*	1	1	
FCVTZS (vector, fixed-point)			1					9	FL*			
FCVTZS (vector, integer)			1					9	FL*			
FCVTZU (scalar, fixed-point)			1					9+1; 15	FLA + NULL ; EAG*	1	1	
FCVTZU (scalar, integer)			1					9+1; 15	FLA + NULL ; EAG*	1	1	
FCVTZU (vector, fixed-point)			1					9	FL*			
FCVTZU (vector, integer)			1					9	FL*			
FDIV (scalar)	<R> = H <R> = S <R> = D		1				E	38	FLA			1
FDIV (vector)		<T> = {4H 8H}	1				E	38	FLA			
FDIV (vector)		<T> = {2S 4S}	1				E	29	FLA			
FDIV (vector)		<T> = 2D	1				E	43	FLA			
FMADD			1					9	FL*			2
FMAX (scalar)			1					4	FL*			
FMAX (vector)			1					4	FL*			
FMAXNM (scalar)			1					4	FL*			
FMAXNM (vector)			1					4	FL*			
FMAXNMP (scalar)			2	✓				6 / [1]4	FLA / FL*			
FMAXNMP (vector)			3	✓				6 / 6 / [1,2]4	FLA / FLA / FL*			
FMAXNMV	<T> = {4H 8H} <T> = 4S		7	✓				4 / ([1]6 / [1,2]4) x 3	FL* / (FLA / FL*) x 3			
			5	✓				4 / ([1]6 / [1,2]4) x 2	FL* / (FLA / FL*) x 2			
FMAXP (scalar)			2	✓				6 / [1]4	FLA / FL*			
FMAXP (vector)			3	✓				6 / 6 / [1,2]4	FLA / FLA / FL*			
FMAXV	<T> = {4H 8H} <T> = 4S		7	✓				4 / ([1]6 / [1,2]4) x 3	FL* / (FLA / FL*) x 3			
			5	✓				4 / ([1]6 / [1,2]4) x 2	FL* / (FLA / FL*) x 2			
FMIN (scalar)			1					4	FL*			
FMIN (vector)			1					4	FL*			
FMINNM (scalar)			1					4	FL*			
FMINNM (vector)			1					4	FL*			
FMINNMP (scalar)			2	✓				6 / [1]4	FLA / FL*			
FMINNMP (vector)			3	✓				6 / 6 / [1,2]4	FLA / FLA / FL*			

Instruction	Alias	Control option	# of µOP	Seq. decode	Pre-sync	Post-sync	Blocking	Latency	Pipeline	# of FP	# of SP	FLOPS
FMINNMV		<T> = {4H 8H}	7	✓				4 / ([1]6 / [1,2]4) x 3	FL* / (FLA / FL*) x 3			
		<T> = 4S	5	✓				4 / ([1]6 / [1,2]4) x 2	FL* / (FLA / FL*) x 2			
FMINP (scalar)			2	✓				6 / [1]4	FLA / FL*			
FMINP (vector)			3	✓				6 / 6 / [1,2]4	FLA / FLA / FL*			
FMINV		<T>= {4H 8H}	7	✓				4 / ([1]6 / [1,2]4) x 3	FL* / (FLA / FL*) x 3			
		<T> = 4S	5	✓				4 / ([1]6 / [1,2]4) x 2	FL* / (FLA / FL*) x 2			
FMLA (by element)			2	✓				6 / [1]9	FLA / FL*			2
FMLA (vector)			1					9	FL*			2
FMLS (by element)			2	✓				6 / [1]9	FLA / FL*			2
FMLS (vector)			1					9	FL*			2
FMOV (vector, immediate)			1					4	FLA			
FMOV (register)			1					4	FL*			
FMOV (general)		{Wn Xn} to {Hd Sd Dd Vd}	1					1+3+6	EXA + NULL + FLA			
		{Hn Sn Dn} to {Wd Xd}	1					1 ; 13	FLA ; EAG*	1	1	
		Vn.D[1] to Xd	1					6+1 ; 18	FLA + NULL ; EAG*	1	1	
FMOV (scalar, immediate)			1					4	FLA			
FMSUB			1					9	FL*			2
FMUL (by element)			2	✓				6 / [1]9	FLA / FL*			1
FMUL (scalar)			1					9	FL*			1
FMUL (vector)			1					9	FL*			1
FMULX (by element)			2	✓				6 / [1]9	FLA / FL*			1
FMULX			1					9	FL*			1
FNEG (scalar)			1					4	FL*			
FNEG (vector)			1					4	FL*			
FNMADD			1					9	FL*			2
FNMSUB			1					9	FL*			2
FNMUL			1					9	FL*			1
FRECPE			1					4	FL*			
FRECPSCS			1					9	FLA			1
FRECPX			1					4	FL*			
FRINTA (scalar)			1					9	FL*			
FRINTA (vector)			1					9	FL*			
FRINTI (scalar)			1					9	FL*			
FRINTI (vector)			1					9	FL*			
FRINTM (scalar)			1					9	FL*			
FRINTM (vector)			1					9	FL*			
FRINTN (scalar)			1					9	FL*			
FRINTN (vector)			1					9	FL*			
FRINTP (scalar)			1					9	FL*			

Instruction	Alias	Control option	# of μOP	Seq. decode	Pre-sync	Post-sync	Blocking	Latency	Pipeline	# of FP	# of SP	FLOPS
FRINTP (vector)			1					9	FL*			
FRINTX (scalar)			1					9	FL*			
FRINTX (vector)			1					9	FL*			
FRINTZ (scalar)			1					9	FL*			
FRINTZ (vector)			1					9	FL*			
FRSQRTE			1					4	FL*			
FRSQRTS			1					9	FLA			1
FSQRT (scalar)		<R> = H	1				E	38	FLA			1
		<R> = S	1				E	29	FLA			
		<R> = D	1				E	43	FLA			
FSQRT (vector)		<T> = {4H 8H}	1				E	38	FLA			1
		<T> = {2S 4S}	1				E	29	FLA			
		<T> = 2D	1				E	43	FLA			
FSUB (scalar)			1					9	FL*			1
FSUB (vector)			1					9	FL*			1
INS (element)	MOV (element)		1					6	FLA			
INS (general)	MOV (from general)		1					1+3+6	EXA + NULL + FLA			
LD1 (multiple structures)		No offset 1 register <T> = {8B 4H 2S 2D 1D}	1					8	EAG*			1
		No offset 1 register <T> = {16B 8H 4S}	1					11	EAG*			1
		No offset 2 registers <T> = {8B 4H 2S 2D 1D}	2					8 / 8	EAG* / EAG*			2
		No offset 2 registers <T> = {16B 8H 4S}	2					11 / 11	EAG* / EAG*			2
		No offset 3 registers <T> = {8B 4H 2S 2D 1D}	3					8 / 8 / 8	EAG* / EAG* / EAG*			3
		No offset 3 registers <T> = {16B 8H 4S}	3					11 / 11 / 11	EAG* / EAG* / EAG*			3
		No offset 4 registers <T> = {8B 4H 2S 2D 1D}	4					8 / 8 / 8 / 8	EAG* / EAG* / EAG* / EAG*			4
		No offset 4 registers <T> = {16B 8H 4S}	4					11 / 11 / 11 / 11	EAG* / EAG* / EAG* / EAG*			4
		Post-index 1 register <T> = {8B 4H 2S 2D 1D}	2					8 / 1	EAG* / EAG*			1
		Post-index 1 register <T> = {16B 8H 4S}	2					11 / 1	EAG* / EAG*			1
		Post-index 2 registers <T> = {8B 4H 2S 2D 1D}	3					8 / 8 / 1	EAG* / EAG* / EAG*			2
		Post-index 2 registers <T> = {16B 8H 4S}	3					11 / 11 / 1	EAG* / EAG* / EAG*			2
		Post-index 3 registers <T> = {8B 4H 2S 2D 1D}	4					8 / 8 / 8 / 1	EAG* / EAG* / EAG* / EAG*			3
		Post-index 3 registers <T> = {16B 8H 4S}	4					11 / 11 / 11 / 1	EAG* / EAG* / EAG* / EAG*			3
		Post-index 4 registers <T> = {8B 4H 2S 2D 1D}	5					8 / 8 / 8 / 8 / 1	EAG* / EAG* / EAG* / EAG* / EAG*			4
		Post-index 4 registers <T> = {16B 8H 4S}	5					11 / 11 / 11 / 11 / 1	EAG* / EAG* / EAG* / EAG* / EAG*			4
LD1 (single structure)		No offset	2	✓				8 / 6	EAG* / FLA			1
		Post-index	3	✓				8 / 6 / 1	EAG* / FLA / EAG*			1
LD1R		No offset	1					8	EAG*			1
		Post-index	2					8 / 1	EAG* / EAG*			1
LD2 (multiple structures)		No offset	2					11 / 11	EAG* / EAG*			2
		Post-index	3					11 / 11 / 1	EAG* / EAG* / EAG*			2

Instruction	Alias	Control option	# of µOP	Seq. decode	Pre-sync	Post-sync	Blocking	Latency	Pipeline	# of FP	# of SP	FLOPS
LD2 (single structure)		No offset	4	✓				(8 / 6) x 2	(EAG* / FLA) x 2	2		
		Post-index	5	✓				(8 / 6) x 2 / 1	(EAG* / FLA) x 2 / EAG*	2		
LD2R		No offset	2					8 / 8	EAG* / EAG*	2		
		Post-index	3					8 / 8 / 1	EAG* / EAG* / EAG*	2		
LD3 (multiple structures)		No offset	3					11 / 11 / 11	EAG* / EAG* / EAG*	3		
		Post-index	4					11 / 11 / 11 / 1	EAG* / EAG* / EAG* / EAG*	3		
LD3 (single structure)		No offset	6	✓				(8 / 6) x 3	(EAG* / FLA) x 3	3		
		Post-index	7	✓				(8 / 6) x 3 / 1	(EAG* / FLA) x 3 / EAG*	3		
LD3R		No offset	3					8 / 8 / 8	EAG* / EAG* / EAG*	3		
		Post-index	4					8 / 8 / 8 / 1	EAG* / EAG* / EAG* / EAG*	3		
LD4 (multiple structures)		No offset	4					11 / 11 / 11 / 11	EAG* / EAG* / EAG* / EAG*	4		
		Post-index	5					11 / 11 / 11 / 11 / 1	EAG* / EAG* / EAG* / EAG* / EAG*	4		
LD4 (single structure)		No offset	8	✓				(8 / 6) x 4	(EAG* / FLA) x 4	4		
		Post-index	9	✓				(8 / 6) x 4 / 1	(EAG* / FLA) x 4 / EAG*	4		
LD4R		No offset	4					8 / 8 / 8 / 8	EAG* / EAG* / EAG* / EAG*	4		
		Post-index	5					8 / 8 / 8 / 8 / 1	EAG* / EAG* / EAG* / EAG* / EAG*	4		
LDNP (SIMD&FP)			2					8 / 8	EAG* / EAG*	2		
LDP (SIMD&FP)		Post-index	3					8 / 8 / 1	EAG* / EAG* / EX*  EAG*	2		
		Pre-index	3					8 / 8 / 1	EAG* / EAG* / EX*  EAG*	2		
		Signed offset	2					8 / 8	EAG* / EAG*	2		
LDR (immediate, SIMD&FP)		Post-index	2					8 / 1	EAG* / EX*  EAG*	1		
		Pre-index	2					8 / 1	EAG* / EX*  EAG*	1		
		Unsigned offset	1					8	EAG*	1		
LDR (literal, SIMD&FP)			1					8	EAGB	1		
LDR (register, SIMD&FP)			1					8	EAG*	1		
LDUR (SIMD&FP)			1					8	EAG*	1		
MLA (by element)			2	✓				6 / [1]9	FLA / FL*			
MLA (vector)			1					9	FL*			
MLS (by element)			2	✓				6 / [1]9	FLA / FL*			
MLS (vector)			1					9	FL*			
MOVI			1					4	FLA			
MUL (by element)			2	✓				6 / [1]9	FLA / FL*			
MUL (vector)			1					9	FL*			
MVNI			1					4	FLA			
NEG (vector)			1					4	FL*			
NOT	MVN		1					4	FL*			
ORN (vector)			1					4	FL*			
ORR (vector, immediate)			1					4	FLA			
ORR (vector, register)	MOV (vector)		1					4	FL*			

Instruction	Alias	Control option	# of μOP	Seq. decode	Pre-sync	Post-sync	Blocking	Latency	Pipeline	# of FP	# of SP	FLOPS
			1					4	FL*			
PMUL			1				E	8	FLA			
PMULL, PMULL2			1				E	8	FLA			
RADDHN, RADDHN2			3	✓				4 / <sup>[1]</sup> 4 / <sup>[1]</sup> 6	FL* / FL* / FLB			
RBIT (vector)			1					4	FL*			
REV16 (vector)			1					4	FL*			
REV32 (vector)			1					4	FL*			
REV64			1					4	FL*			
RSHRN, RSHRN2			3	✓				4 / <sup>[1]</sup> 4 / <sup>[1]</sup> 6	FL* / FL* / FLB			
RSUBHN, RSUBHN2			3	✓				4 / <sup>[1]</sup> 4 / <sup>[1]</sup> 6	FL* / FLA / FLB			
SABA			2	✓				4 / <sup>[1]</sup> 4	FL* / FL*			
SABAL, SABAL2			4	✓				6 / 6 / <sup>[1,2]</sup> 4 / <sup>[1]</sup> 4	FLB / FLB / FL* / FL*			
SABD			1					4	FL*			
SABDL, SABDL2			3	✓				6 / 6 / <sup>[1,2]</sup> 4	FLB / FLB / FL*			
SADALP			3	✓				6 / <sup>[1]</sup> 4 / <sup>[1]</sup> 4	FLB / FL* / FL*			
SADDL, SADDL2			3					6 / 6 / <sup>[1,2]</sup> 4	FLB / FLB / FL*			
SADDLP			2	✓				6 / <sup>[1]</sup> 4	FLB / FL*			
SADDLV			6	✓				4 / <sup>[1]</sup> 4 / <sup>[1]</sup> 6 / <sup>[1,2]</sup> 4 / <sup>[1]</sup> 4 / <sup>[1]</sup> 4	FL* / FL* / FLA / FL* / FL* / FL*			
SADDW, SADDW2			2	✓				6 / <sup>[1]</sup> 4	FLB / FL*			
SCVTF (scalar, fixed-point)			1					1+3+9	EXA + NULL + FLA			
SCVTF (scalar, integer)			1					1+3+9	EXA + NULL + FLA			
SCVTF (vector, fixed-point)			1					9	FL*			
SCVTF (vector, integer)			1					9	FL*			
SHA1C			1				E	1+11	FLA + FLA			
SHA1H			1				E	8	FLA			
SHA1M			1				E	1+11	FLA + FLA			
SHA1P			1				E	1+11	FLA + FLA			
SHA1SU0			1				E	1+8	FLA + FLA			
SHA1SU1			1				E	8	FLA			
SHA256H2			1				E	1+11	FLA + FLA			
SHA256H			1				E	1+11	FLA + FLA			
SHA256SU0			1				E	8	FLA			
SHA256SU1			1				E	1+8	FLA + FLA			
SHADD			1					4	FL*			
SHL			1					4	FL*			
SHLL, SHLL2			2					6 / <sup>[1]</sup> 4	FLB / FL*			
SHRN, SHRN2			2	✓				4 / <sup>[1]</sup> 6	FL* / FLB			
SHSUB			1					4	FL*			

Instruction	Alias	Control option	# of µOP	Seq. decode	Pre-sync	Post-sync	Blocking	Latency	Pipeline	# of FP	# of SP	FLOPS
SLI			3	✓				4 / 4 / [1:2]4	FL* / FLA / FL*			
SMAX			1					4	FL*			
SMAXP			3	✓				6 / 6 / [1:2]4	FLA / FLA / FL*			
SMAXV			6	✓				4 / [1]6 / [1:2]4 / [1]4 / [1]4 / [1]4	FL* / FLA / FL* / FL* / FL* / FL*			
SMIN			1					4	FL*			
SMINP			3	✓				6 / 6 / [1:2]4	FLA / FLA / FL*			
SMINV			6	✓				4 / [1]6 / [1:2]4 / [1]4 / [1]4 / [1]4	FL* / FLA / FL* / FL* / FL* / FL*			
SMLAL, SMLAL2 (by element)		<Ta> = 4S	4	✓			// E /	6 / 6 / [1:2]8 / [1]4	FLB / FLA / FLA / FL*			
		<Ta> = 2D	3	✓				6 / 6 / [1:2]9	FLB / FLA / FL*			
SMLAL, SMLAL2 (vector)		<Ta> = {8H 4S}	4	✓			// E /	6 / 6 / [1:2]8 / [1]4	FLB / FLB / FLA / FL*			
		<Ta> = 2D	3	✓				6 / 6 / [1:2]9	FLB / FLB / FL*			
SMLS, SMLS2 (by element)		<Ta> = 4S	4	✓			// E /	6 / 6 / [1:2]8 / [1]4	FLB / FLA / FLA / FL*			
		<Ta> = 2D	3	✓				6 / 6 / [1:2]9	FLB / FLA / FL*			
SMLS, SMLS2 (vector)		<Ta> = {8H 4S}	4	✓			// E /	6 / 6 / [1:2]8 / [1]4	FLB / FLB / FLA / FL*			
		<Ta> = 2D	3	✓				6 / 6 / [1:2]9	FLB / FLB / FL*			
SMOV			1					6 + 1 + 18	FLA + NULL + EAG*	1	1	
SMULL, SMULL2 (by element)		<Ta> = 4S	3	✓			// E	6 / 6 / [1:2]8	FLB / FLA / FLA			
		<Ta> = 2D	3	✓				6 / 6 / [1:2]9	FLB / FLA / FL*			
SMULL, SMULL2 (vector)		<Ta> = {8H 4S}	3	✓			// E	6 / 6 / [1:2]8	FLB / FLB / FLA			
		<Ta> = 2D	3	✓				6 / 6 / [1:2]9	FLB / FLB / FL*			
SQABS			1					4	FL*			
SQADD			1					4	FL*			
SQDMLAL, SQDMLAL2 (by element)		Scalar <Va> = S	4	✓			// E /	6 / 6 / [1:2]8 / [1]4	FLB / FLA / FLA / FL*			
		Scalar <Va> = D	3	✓				6 / [1]9 / [1]4	FLA / FL* / FL*			
		Vector <Ta> = 4S	4	✓			// E /	6 / 6 / [1:2]8 / [1]4	FLB / FLA / FLA / FL*			
		Vector <Ta> = 2D	4	✓				6 / 6 / [1:2]9 / [1]4	FLB / FLA / FL* / FL*			
SQDMLAL, SQDMLAL2 (vector)		Scalar <Va> = S	4	✓			// E /	6 / 6 / [1:2]8 / [1]4	FLB / FLB / FLA / FLA			
		Scalar <Va> = D	2	✓				9 / [1]4	FL* / FL*			
		Vector <Ta> = 4S	4	✓			// E /	6 / 6 / [1:2]8 / [1]4	FLB / FLB / FLA / FLA			
		Vector <Ta> = 2D	4	✓				6 / 6 / [1:2]9 / [1]4	FLB / FLB / FL* / FL*			
SQDMLSL, SQDMLSL2 (by element)		Scalar <Va> = S	4	✓			// E /	6 / 6 / [1:2]8 / [1]4	FLB / FLA / FLA / FL*			
		Scalar <Va> = D	3	✓				6 / [1]9 / [1]4	FLA / FL* / FL*			
		Vector <Ta> = 4S	4	✓			// E /	6 / 6 / [1:2]8 / [1]4	FLB / FLA / FLA / FL*			
		Vector <Ta> = 2D	4	✓				6 / 6 / [1:2]9 / [1]4	FLB / FLA / FL* / FL*			
SQDMLSL, SQDMLSL2 (vector)		Scalar <Va> = S	4	✓			// E /	6 / 6 / [1:2]8 / [1]4	FLB / FLB / FLA / FLA			
		Scalar <Va> = D	2	✓				9 / [1]4	FL* / FL*			
		Vector <Ta> = 4S	4	✓			// E /	6 / 6 / [1:2]8 / [1]4	FLB / FLB / FLA / FLA			
		Vector <Ta> = 2D	4	✓				6 / 6 / [1:2]9 / [1]4	FLB / FLB / FL* / FL*			

Instruction	Alias	Control option	# of µOP	Seq. decode	Pre-sync	Post-sync	Blocking	Latency	Pipeline	# of FP	# of SP	FLOPS
SQDMULH (by element)		Scalar <V> = H, Vector <T> = {4H 8H}	2	✓			/ E	6 / <sup>[1]</sup> 8	FLA / FLA			
		Scalar <V> = S, Vector <T> = {2S 4S}	2	✓				6 / <sup>[1]</sup> 9	FLA / FL*			
SQDMULH (vector)		Scalar <V> = H, Vector <T> = {4H 8H}	1				E	8	FLA			
		Scalar <V> = S, Vector <T> = {2S 4S}	1					9	FL*			
SQDMULL, SQDMULL2 (by element)		Scalar <Va> = S	3	✓			// E	6 / 6 / <sup>[1,2]</sup> 8	FLB / FLA / FLA			
		Scalar <Va> = D	2	✓				6 / <sup>[1]</sup> 9	FLA / FL*			
		Vector <Ta> = 4S	3	✓			// E	6 / 6 / <sup>[1,2]</sup> 8	FLB / FLA / FLA			
		Vector <Ta> = 2D	3	✓				6 / 6 / <sup>[1,2]</sup> 9	FLB / FLA / FL*			
SQDMULL, SQDMULL2 (vector)		Scalar <Va> = S	3	✓			// E	6 / 6 / <sup>[1,2]</sup> 8	FLB / FLB / FLA			
		Scalar <Va> = D	1					9	FL*			
		Vector <Ta> = 4S	3	✓			// E	6 / 6 / <sup>[1,2]</sup> 8	FLB / FLB / FLA			
		Vector <Ta> = 2D	3	✓				6 / 6 / <sup>[1,2]</sup> 9	FLB / FLB / FL*			
SQNEG			1					4	FL*			
SQRDMLAH (by element)		Scalar <V> = H, Vector <T> = {4H 8H}	2	✓			/ E	6 / 1+ <sup>[1]</sup> 8	FLA / FLA + FLA			
		Scalar <V> = S, Vector <T> = {2S 4S}	2	✓				6 / <sup>[1]</sup> 9	FLA / FL*			
SQRDMLAH (vector)		Scalar <V> = H, Vector <T> = {4H 8H}	1				E	1+8	FLA + FLA			
		Scalar <V> = S, Vector <T> = {2S 4S}	1					9	FL*			
SQRDMLSH (by element)		Scalar <V> = H, Vector <T> = {4H 8H}	2	✓			/ E	6 / 1+ <sup>[1]</sup> 8	FLA / FLA + FLA			
		Scalar <V> = S, Vector <T> = {2S 4S}	2	✓				6 / <sup>[1]</sup> 9	FLA / FL*			
SQRDMLSH (vector)		Scalar <V> = H, Vector <T> = {4H 8H}	1				E	1+8	FLA + FLA			
		Scalar <V> = S, Vector <T> = {2S 4S}	1					9	FL*			
SQRDMULH (by element)		Scalar <V> = H, Vector <T> = {4H 8H}	2	✓			/ E	6 / <sup>[1]</sup> 8	FLA / FLA			
		Scalar <V> = S, Vector <T> = {2S 4S}	2	✓				6 / <sup>[1]</sup> 9	FLA / FL*			
SQRDMULH (vector)		Scalar <V> = H, Vector <T> = {4H 8H}	1				E	8	FLA			
		Scalar <V> = S, Vector <T> = {2S 4S}	1					9	FL*			
SQRSHL			2	✓				6 / <sup>[1]</sup> 4	FLB / FL*			
SQRSHRN, SQRSHRN2			3	✓				4 / <sup>[1]</sup> 4 / <sup>[1]</sup> 6	FL* / FL* / FLB			
SQRSHRUN, SQRSHRUN2			3	✓				4 / <sup>[1]</sup> 4 / <sup>[1]</sup> 6	FL* / FL* / FLB			
SQSHL (immediate)			1					6	FLB			
SQSHL (register)			1					6	FLB			
SQSHLU			1					6	FLB			
SQSHRN, SQSHRN2			2	✓				4 / <sup>[1]</sup> 6	FL* / FLB			
SQSHRUN, SQSHRUN2			2	✓				4 / <sup>[1]</sup> 6	FL* / FLB			
SQSUB			1					4	FL*			
SQXTN, SQXTN2			1					6	FLB			
SQXTUN, SQXTUN2			1					6	FLB			
SRHADD			1					4	FL*			
SRI			3	✓				4 / 4 / <sup>[1,2]</sup> 4	FL* / FLA / FL*			

Instruction	Alias	Control option	# of μOP	Seq. decode	Pre-sync	Post-sync	Blocking	Latency	Pipeline	# of FP	# of SP	FLOPS
SRSHL			2	✓				6 / <sup>[1]</sup> 4	FLB / FL*			
SRSHR			2	✓				4 / <sup>[1]</sup> 4	FL* / FL*			
SRSRA			3	✓				4 / <sup>[1]</sup> 4 / <sup>[1]</sup> 4	FL* / FL* / FL*			
SSHLL			1					6	FLB			
SSHLL, SSHLL2	SXTL, SXTL2		2					6 / <sup>[1]</sup> 4	FLB / FL*			
			2					6 / <sup>[1]</sup> 4	FLB / FL*			
SSHR			1					4	FL*			
SSRA			2	✓				4 / <sup>[1]</sup> 4	FL* / FL*			
SSUBL, SSUBL2			3					6 / 6 / <sup>[1,2]</sup> 4	FLB / FLB / FL*			
SSUBW, SSUBW2			2	✓				6 / <sup>[1]</sup> 4	FLB / FL*			
ST1 (multiple structures)		No offset 1 register	1					NA, NA	EAG*, FLA	1	1	
		No offset 2 registers	2					(NA, NA) x 2	(EAG*, FLA) x 2	2	2	
		No offset 3 registers	3					(NA, NA) x 3	(EAG*, FLA) x 3	3	3	
		No offset 4 registers	4					(NA, NA) x 4	(EAG*, FLA) x 4	4	4	
		Post-index 1 register	2					NA, NA / 1	EAG*, FLA / EAG*	1	1	
		Post-index 2 registers	3					(NA, NA) x 2 / 1	(EAG*, FLA) x 2 / EAG*	2	2	
		Post-index 3 registers	4					(NA, NA) x 3 / 1	(EAG*, FLA) x 3 / EAG*	3	3	
		Post-index 4 registers	5					(NA, NA) x 4 / 1	(EAG*, FLA) x 4 / EAG*	4	4	
ST1 (single structure)		No offset	1					NA, NA	EAG*, FLA	1	1	
		Post-index	2					NA, NA / 1	EAG*, FLA / EAG*	1	1	
ST2 (multiple structures)		No offset	2					(NA, NA) x 2	(EAG*, FLA) x 2	2	2	
		Post-index	3					(NA, NA) x 2 / 1	(EAG*, FLA) x 2 / EAG*	2	2	
ST2 (single structure)		No offset	2					(NA, NA) x 2	(EAG*, FLA) x 2	2	2	
		Post-index	3					(NA, NA) x 2 / 1	(EAG*, FLA) x 2 / EAG*	2	2	
ST3 (multiple structures)		No offset	3					(NA, NA) x 3	(EAG*, FLA) x 3	3	3	
		Post-index	4					(NA, NA) x 3 / 1	(EAG*, FLA) x 3 / EAG*	3	3	
ST3 (single structure)		No offset	3					(NA, NA) x 3	(EAG*, FLA) x 3	3	3	
		Post-index	4					(NA, NA) x 3 / 1	(EAG*, FLA) x 3 / EAG*	3	3	
ST4 (multiple structures)		No offset	4					(NA, NA) x 4	(EAG*, FLA) x 4	4	4	
		Post-index	5					(NA, NA) x 4 / 1	(EAG*, FLA) x 4 / EAG*	4	4	
ST4 (single structure)		No offset	4					(NA, NA) x 4	(EAG*, FLA) x 4	4	4	
		Post-index	5					(NA, NA) x 4 / 1	(EAG*, FLA) x 4 / EAG*	4	4	
STNP (SIMD&FP)			2					(NA, NA) x 2	(EAG*, FLA) x 2	2	2	
STP (SIMD&FP)		Post-index	3					(NA, NA) x 2 / 1	(EAG*, FLA) x 2 / EX*  EAG*	2	2	
		Pre-index	3					(NA, NA) x 2 / 1	(EAG*, FLA) x 2 / EX*  EAG*	2	2	
		Signed offset	2					(NA, NA) x 2	(EAG*, FLA) x 2	2	2	
STR (immediate, SIMD&FP)		Post-index	2					NA, NA / 1	EAG*, FLA / EX*  EAG*	1	1	
		Pre-index	2					NA, NA / 1	EAG*, FLA / EX*  EAG*	1	1	
		Unsigned offset	1					NA, NA	EAG*, FLA	1	1	

Instruction	Alias	Control option	# of µOP	Seq. decode	Pre-sync	Post-sync	Blocking	Latency	Pipeline	# of FP	# of SP	FLOPS
STR (register, SIMD&FP)			1					NA, NA	EAG*, FLA	1	1	
STUR (SIMD&FP)			1					NA, NA	EAG*, FLA	1	1	
SUB (vector)			1					4	FL*			
SUBHN, SUBHN2			2	✓				4 / <sup>[1]</sup> 6	FL* / FLB			
SUQADD			1					4	FL*			
TBL		Single register table	1					6	FLB			
		Tow register table	3	✓				6 / 6 / <sup>[1,2]</sup> 4	FLB / FLB / FL*			
		Three register table	5	✓				6 / (6 / <sup>[1,2]</sup> 4) x 2	FLB / (FLB / FL*) x 2			
		Four register table	7	✓				6 / (6 / <sup>[1,2]</sup> 4) x 3	FLB / (FLB / FL*) x 3			
TBX		Single register table	3	✓				6 / 6 / <sup>[1,2]</sup> 4	FLB / FLB / FL*			
		Tow register table	5	✓				6 / (6 / <sup>[1,2]</sup> 4) x 2	FLB / (FLB / FL*) x 2			
		Three register table	7	✓				6 / (6 / <sup>[1,2]</sup> 4) x 3	FLB / (FLB / FL*) x 3			
		Four register table	9	✓				6 / (6 / <sup>[1,2]</sup> 4) x 4	FLB / (FLB / FL*) x 4			
TRN1			1					6	FLA			
TRN2			1					6	FLA			
UABA			2	✓				4 / <sup>[1]</sup> 4	FL* / FL*			
UABAL, UABAL2			4	✓				6 / 6 / <sup>[1,2]</sup> 4 / <sup>[1]</sup> 4	FLB / FLB / FL* / FL*			
UABD			1					4	FL*			
UABDL, UABDL2			3	✓				6 / 6 / <sup>[1,2]</sup> 4	FLB / FLB / FL*			
UADALP		<Ta> = {4H 8H 2S 4S}	1					6	FLB			
		<Ta> = {1D 2D}	3	✓				6 / <sup>[1]</sup> 4 / <sup>[1]</sup> 4	FLB / FL* / FL*			
UADDL, UADDL2			3					6 / 6 / <sup>[1,2]</sup> 4	FLB / FLB / FL*			
UADDLP			2	✓				6 / <sup>[1]</sup> 4	FLB / FL*			
UADDLV			6	✓				4 / <sup>[1]</sup> 4 / <sup>[1]</sup> 6 / <sup>[1,2]</sup> 4 / <sup>[1]</sup> 4 / <sup>[1]</sup> 4	FL* / FL* / FLA / FL* / FL* / FL*			
UADDW, UADDW2			2	✓				6 / <sup>[1]</sup> 4	FLB / FL*			
UCVTF (scalar, fixed-point)			1					1+3+9	EXA + NULL + FLA			
UCVTF (scalar, integer)			1					1+3+9	EXA + NULL + FLA			
UCVTF (vector, fixed-point)			1					9	FL*			
UCVTF (vector, integer)			1					9	FL*			
UHADD			1					4	FL*			
UHSUB			1					4	FL*			
UMAX			1					4	FL*			
UMAXP			3	✓				6 / 6 / <sup>[1,2]</sup> 4	FLA / FLA / FL*			
UMAXV			6	✓				4 / <sup>[1]</sup> 6 / <sup>[1]</sup> 4 / <sup>[1,2]</sup> 4 / <sup>[1]</sup> 4 / <sup>[1]</sup> 4	FL* / FLA / FL* / FL* / FL* / FL*			
UMIN			1					4	FL*			
UMINP			3	✓				6 / 6 / <sup>[1,2]</sup> 4	FLA / FLA / FL*			
UMINV			6	✓				4 / <sup>[1]</sup> 6 / <sup>[1]</sup> 4 / <sup>[1,2]</sup> 4 / <sup>[1]</sup> 4 / <sup>[1]</sup> 4	FL* / FLA / FL* / FL* / FL* / FL*			
UMLAL, UMLAL2 (by element)		<Ta> = 4S	4	✓			// E /	6 / 6 / <sup>[1,2]</sup> 8 / <sup>[1]</sup> 4	FLB / FLA / FLA / FL*			
		<Ta> = 2D	3	✓				6 / 6 / <sup>[1,2]</sup> 9	FLB / FLA / FL*			

Instruction	Alias	Control option	# of μOP	Seq. decode	Pre-sync	Post-sync	Blocking	Latency	Pipeline	# of FP	# of SP	FLOPS
UMLAL, UMLAL2 (vector)		<Ta> = {8H 4S}	4	✓			// E /	6 / 6 / [1:2]8 / [1]4	FLB / FLB / FLA / FL*			
		<Ta> = 2D	3	✓				6 / 6 / [1:2]9	FLB / FLB / FL*			
UMLSL, UMLSL2 (by element)		<Ta> = 4S	4	✓			// E /	6 / 6 / [1:2]8 / [1]4	FLB / FLA / FLA / FL*			
		<Ta> = 2D	3	✓				6 / 6 / [1:2]9	FLB / FLA / FL*			
UMLSL, UMLSL2 (vector)		<Ta> = {8H 4S}	4	✓			// E /	6 / 6 / [1:2]8 / [1]4	FLB / FLB / FLA / FL*			
		<Ta> = 2D	3	✓				6 / 6 / [1:2]9	FLB / FLB / FL*			
UMOV	MOV (to general)		1					6+1+18	FLA + NULL + EAG*	1	1	
			1					6+1+18	FLA + NULL + EAG*	1	1	
UMULL, UMULL2 (by element)		<Ta> = 4S	3	✓			// E	6 / 6 / [1:2]8	FLB / FLA / FLA			
		<Ta> = 2D	3	✓				6 / 6 / [1:2]9	FLB / FLA / FL*			
UMULL, UMULL2 (vector)		<Ta> = {8H 4S}	3	✓			// E	6 / 6 / [1:2]8	FLB / FLB / FLA			
		<Ta> = 2D	3	✓				6 / 6 / [1:2]9	FLB / FLB / FL*			
UQADD			1					4	FL*			
UQRSHL			2	✓				6 / [1]4	FLB / FL*			
UQRSHRN, UQRSHRN2			3	✓				4 / [1]4 / [1]6	FL* / FL* / FLB			
UQSHL (immediate)			1					6	FLB			
UQSHL (register)			1					6	FLB			
UQSHRN, UQSHRN2			2	✓				4 / [1]6	FL* / FLB			
UQSUB			1					4	FL*			
UQXTN, UQXTN2			1					6	FLB			
URECPE			1					4	FL*			
URHADD			1					4	FL*			
URSHL			2	✓				6 / [1]4	FLB / FL*			
URSHR			2	✓				4 / [1]4	FL* / FL*			
URSRQE			1					4	FL*			
URSRA			3	✓				4 / [1]4 / [1]4	FL* / FL* / FL*			
USHL			1					6	FLB			
USHLL, USHLL2	UXTL, UXTL2		2					6 / [1]4	FLB / FL*			
			2					6 / [1]4	FLB / FL*			
USHR			1					4	FL*			
USQADD			1					4	FL*			
USRA			2	✓				4 / [1]4	FL* / FL*			
USUBL, USUBL2			3					6 / 6 / [1:2]4	FLB / FLB / FL*			
USUBW, USUBW2			2	✓				6 / [1]4	FLB / FL*			
UZP1			1					6	FLA			
UZP2			1					6	FLA			
XTN, XTN2			1					6	FLB			
ZIP1			1					6	FLA			
ZIP2			1					6	FLA			

### 16.3. SVE instructions

Table 16-3 Instruction Attributes/Latency (SVE)

Instruction	Alias	Control option	VL	# of μOP	Seq. decode	Pre-sync	Post-sync	Pack	Extra μOP	Blocking	Latency	Pipeline	# of FP	# of SP	FLOPS
ABS				1				✓			4	FL*			
ADD (immediate)				1				✓			4	FL*			
ADD (vectors, predicated)				1				✓	✓		4	FL*			
ADD (vectors, unpredicated)				1							4	FL*			
ADDPL				1							1	EX*			
ADDVL				1							1	EX*			
ADR		Packed offsets		1							1+4	FLA + FLA			
				1							4	FLA			
AND (immediate)	BIC (immediate)			1				✓			4	FLA			
AND (predicates)	MOV (predicate, predicated, zeroing)			1							3	PRX			
				1							3	PRX			
AND (vectors, predicated)				1				✓	✓		4	FL*			
AND (vectors, unpredicated)				1							4	FL*			
ANDS	MOVS (predicated)			1							3	PRX			
				1							3	PRX			
ANDV		<V> = B		10	✓						4 / ([1]6 / [1,2]4) x 3 / [1]4 / [1]4 / [1]4	FL* / (FLA / FL*) x 3 / FL* / FL* / FL*			
		<V> = H		9	✓						4 / ([1]6 / [1,2]4) x 3 / [1]4 / [1]4	FL* / (FLA / FL*) x 3 / FL* / FL*			
		<V> = S		8	✓						4 / ([1]6 / [1,2]4) x 3 / [1]4	FL* / (FLA / FL*) x 3 / FL*			
		<V> = D		128	3	✓					4 / [1]6 / [1,2]4	FL* / FLA / FL*			
				256	5	✓					4 / ([1]6 / [1,2]4) x 2	FL* / (FLA / FL*) x 2			
				512	7	✓					4 / ([1]6 / [1,2]4) x 3	FL* / (FLA / FL*) x 3			
ASR (immediate, predicated)				1				✓	✓		4	FL*			
ASR (immediate, unpredicated)				1							4	FL*			
ASR (vectors)				1				✓	✓		4	FL*			
ASR (wide elements, predicated)				1				✓	✓		4	FL*			
ASR (wide elements, unpredicated)				1							4	FL*			
ASRD				2	✓			✓	✓		4 / [1]4	FLA / FL*			
ASRR				1				✓	✓		4	FL*			
BIC (predicates)				1							3	PRX			
BIC (vectors, predicated)				1				✓	✓		4	FL*			
BIC (vectors, unpredicated)				1							4	FL*			
BICS				1							3	PRX			
BRKA				1							3	PRX			
BRKAS				1							3	PRX			
BRKB				1							3	PRX			

Instruction	Alias	Control option	VL	# of μOP	Seq. decode	Pre-sync	Post-sync	Pack	Extra μOP	Blocking	Latency	Pipeline	# of FP	# of SP	FLOPS
BRKBS				1							3	PRX			
BRKN				1							3	PRX			
BRKNS				1							3	PRX			
BRKPA				1							3	PRX			
BRKPAS				1							3	PRX			
BRKPB				1							3	PRX			
BRKPBS				1							3	PRX			
CLASTA (scalar)				1							1+3+6+1+18	EXA + NULL + EAG* + NULL + FLA	1	1	
CLASTA (SIMD&FP scalar)				1							6	FLA			
CLASTA (vectors)				1				✓			6	FLA			
CLASTB (scalar)				1							1+3+6+1+18	EXA + NULL + EAG* + NULL + FLA	1	1	
CLASTB (SIMD&FP scalar)				1							6	FLA			
CLASTB (vectors)				1				✓			6	FLA			
CLS				1				✓			4	FLA			
CLZ				1				✓			4	FLA			
CMPEQ (immediate)				1							4	PRX, FLA			
CMPEQ (vectors)				1							4	PRX, FLA			
CMPEQ (wide elements)				1							4	PRX, FLA			
CMPGE (immediate)				1							4	PRX, FLA			
CMPGE (vectors)	CMPLT (vectors)			1							4	PRX, FLA			
CMPGE (wide elements)				1							4	PRX, FLA			
CMPGT (immediate)				1							4	PRX, FLA			
CMPGT (vectors)	CMPLE (vectors)			1							4	PRX, FLA			
CMPGT (wide elements)				1							4	PRX, FLA			
CMPHI (immediate)				1							4	PRX, FLA			
CMPHI (vectors)	CMPLS (vectors)			1							4	PRX, FLA			
CMPHI (wide elements)				1							4	PRX, FLA			
CMPHS (immediate)				1							4	PRX, FLA			
CMPHS (vectors)	CMPL0 (vectors)			1							4	PRX, FLA			
CMPHS (wide elements)				1							4	PRX, FLA			
CMPLE (immediate)				1							4	PRX, FLA			
CMPLE (wide elements)				1							4	PRX, FLA			
CMPL0 (immediate)				1							4	PRX, FLA			
CMPL0 (wide elements)				1							4	PRX, FLA			
CMPLS (immediate)				1							4	PRX, FLA			
CMPLS (wide elements)				1							4	PRX, FLA			
CMPLT (immediate)				1							4	PRX, FLA			
CMPLT (wide elements)				1							4	PRX, FLA			
CMPNE (immediate)				1							4	PRX, FLA			

Instruction	Alias	Control option	VL	# of μOP	Seq. decode	Pre-sync	Post-sync	Pack	Extra μOP	Blocking	Latency	Pipeline	# of FP	# of SP	FLOPS
CMPNE (vectors)				1							4	PRX, FLA			
CMPNE (wide elements)				1							4	PRX, FLA			
CNOT				1				✓			4	FL*			
CNT				1				✓			4	FLB			
CNTB				1							1	EX*			
CNTD				1							1	EX*			
CNTH				1							1	EX*			
CNTP				1							3+2+1	PRX + NULL + EXA			
CNTW				1							1	EX*			
COMPACT				1							6	FLA			
CPY (immediate)	FMOV (zero, predicated)			1				✓			4	FLA			
	MOV (immediate, predicated)			1				✓			4	FLA			
CPY (scalar)	MOV (scalar, predicated)			1				✓			1+3+4	EXA + NULL + FLA			
CPY (SIMD&FP scalar)	MOV (SIMD&FP scalar, predicated)			1				✓			6	FLA			
CTERMEQ				1						E	1+1	EX* + EX*			
CTERMNE				1						E	1+1	EX* + EX*			
DEC B				1							1	EX*			
DEC D (scalar)				1							1	EX*			
DEC D (vector)				1				✓			4	FL*			
DEC H (scalar)				1							1	EX*			
DEC H (vector)				1				✓			4	FL*			
DEC P (scalar)				2							3+2+1 / [1]1	PRX+NULL+EXA / EXB			
DEC P (vector)				1				✓			3+5+4	PRX + NULL + FLA			
DEC W (scalar)				1							1	EX*			
DEC W (vector)				1				✓			4	FL*			
DUP (immediate)	FMOV (zero, unpredicated)			1							4	FLA			
	MOV (immediate, unpredicated)			1							4	FLA			
DUP (indexed)	MOV (SIMD&FP scalar, unpredicated)			1							6	FLA			
				1							6	FLA			
DUP (scalar)	MOV (scalar, unpredicated)			1							1+3+4	EXA + NULL + FLA			
DUPM	MOV (bitmask immediate)			1							4	FLA			
				1							4	FLA			
EOR (immediate)	EON			1				✓			4	FLA			
EOR (predicates)	NOT (predicate)			1							3	PRX			
				1							3	PRX			
EOR (vectors, predicated)				1				✓	✓		4	FL*			
EOR (vectors, unpredicated)				1							4	FL*			

Instruction	Alias	Control option	VL	# of μOP	Seq. decode	Pre-sync	Post-sync	Pack	Extra μOP	Blocking	Latency	Pipeline	# of FP	# of SP	FLOPS
EORS	NOTS			1							3	PRX			
				1							3	PRX			
EORV		<V> = B		10	✓						4 / ([1]6 / [1,2]4) x 3 / [1]4 / [1]4 / [1]4	FL* / (FLA / FL*) x 3 / FL* / FL* / FL*			
		<V> = H		9	✓						4 / ([1]6 / [1,2]4) x 3 / [1]4 / [1]4	FL* / (FLA / FL*) x 3 / FL* / FL*			
		<V> = S		8	✓						4 / ([1]6 / [1,2]4) x 3 / [1]4	FL* / (FLA / FL*) x 3 / FL*			
		<V> = D	128	3	✓						4 / [1]6 / [1,2]4	FL* / FLA / FL*			
			256	5	✓						4 / ([1]6 / [1,2]4) x 2	FL* / (FLA / FL*) x 2			
			512	7	✓						4 / ([1]6 / [1,2]4) x 3	FL* / (FLA / FL*) x 3			
EXT				1				✓			6	FLA			
FABD				1				✓			9	FL*			1
FABS				1				✓			4	FL*			
FACGE	FACLT			1							4	FLA			
FACGT	FACLE			1							4	FLA			
FADD (immediate)				1				✓			9	FLA			1
FADD (vectors, predicated)				1				✓			9	FL*			1
FADD (vectors, unpredicated)				1							9	FL*			1
FADDA		<V> = H	128	15	✓						9 / 6 / ([1,2]9 / [1]6) x 6 / [1,2]9	FL* / FLA / (FL* / FLA) x 6 / FL*			1
			256	31	✓						9 / 6 / ([1,2]9 / [1]6) x 14 / [1,2]9	FL* / FLA / (FL* / FLA) x 14 / FL*			
			512	63	✓						9 / 6 / ([1,2]9 / [1]6) x 30 / [1,2]9	FL* / FLA / (FL* / FLA) x 30 / FL*			
		<V> = S	128	7	✓						9 / 6 / ([1,2]9 / [1]6) x 2 / [1,2]9	FL* / FLA / (FL* / FLA) x 2 / FL*			
			256	15	✓						9 / 6 / ([1,2]9 / [1]6) x 6 / [1,2]9	FL* / FLA / (FL* / FLA) x 6 / FL*			
			512	31	✓						9 / 6 / ([1,2]9 / [1]6) x 14 / [1,2]9	FL* / FLA / (FL* / FLA) x 14 / FL*			
		<V> = D	128	3	✓						9 / 6 / [1,2]9	FL* / FLA / FL*			
			256	7	✓						9 / 6 / ([1,2]9 / [1]6) x 2 / [1,2]9	FL* / FLA / (FL* / FLA) x 2 / FL*			
			512	15	✓						9 / 6 / ([1,2]9 / [1]6) x 6 / [1,2]9	FL* / FLA / (FL* / FLA) x 6 / FL*			
FADDV		<V> = H	128	7	✓						4 / 6 / ([1,2]9 / [1]6) x 2 / [1,2]9	FL* / FLA / (FL* / FLA) x 2 / FL*			1
			256	9	✓						4 / 6 / ([1,2]9 / [1]6) x 3 / [1,2]9	FL* / FLA / (FL* / FLA) x 3 / FL*			
			512	11	✓						4 / 6 / ([1,2]9 / [1]6) x 4 / [1,2]9	FL* / FLA / (FL* / FLA) x 4 / FL*			
		<V> = S	128	5	✓						4 / 6 / [1,2]9 / [1]6 / [1,2]9	FL* / FLA / FL* / FLA / FL*			
			256	7	✓						4 / 6 / ([1,2]9 / [1]6) x 2 / [1,2]9	FL* / FLA / (FL* / FLA) x 2 / FL*			
			512	9	✓						4 / 6 / ([1,2]9 / [1]6) x 3 / [1,2]9	FL* / FLA / (FL* / FLA) x 3 / FL*			
		<V> = D	128	3	✓						4 / 6 / [1,2]9	FL* / FLA / FL*			
			256	5	✓						4 / 6 / [1,2]9 / [1]6 / [1,2]9	FL* / FLA / FL* / FLA / FL*			
			512	7	✓						4 / 6 / ([1,2]9 / [1]6) x 2 / [1,2]9	FL* / FLA / (FL* / FLA) x 2 / FL*			
FCADD				2							6 / [1]9	FLA / FLB			1
FCMEQ (vectors)				1							4	FLA			
FCMEQ (zero)				1							4	FLA			
FCMGE (vectors)	FCMLT (vectors)			1							4	FLA			

Instruction	Alias	Control option	VL	# of μOP	Seq. decode	Pre-sync	Post-sync	Pack	Extra μOP	Blocking	Latency	Pipeline	# of FP	# of SP	FLOPS	
FCMGE (zero)				1							4	FLA				
FCMGT (vectors)	FCMLE (vectors)			1							4	FLA				
FCMGT (zero)				1							4	FLA				
FCMLA (indexed)				3							6 / 6 / [1:2]9	FLA / FLA / FL*			2	
FCMLA (vectors)				3							6 / 6 / [1:2]9	FLA / FLA / FL*			2	
FCMLE (zero)				1							4	FLA				
FCMLT (zero)				1							4	FLA				
FCMNE (vectors)				1							4	FLA				
FCMNE (zero)				1							4	FLA				
FCMUO				1							4	FLA				
FCPY	FMOV (immediate, predicated)			1					✓		4	FLA				
FCVT				1					✓		9	FL*				
FCVTZS				1					✓		9	FL*				
FCVTZU				1					✓		9	FL*				
FDIV		<T> = H	128	1					✓	✓	E	38	FLA			1
			256	1					✓	✓	E	70	FLA			
			512	1					✓	✓	E	134	FLA			
		<T> = S	128	1					✓	✓	E	29	FLA			
			256	1					✓	✓	E	52	FLA			
			512	1					✓	✓	E	98	FLA			
		<T> = D	128	1					✓	✓	E	43	FLA			
			256	1					✓	✓	E	80	FLA			
			512	1					✓	✓	E	154	FLA			
FDIVR		<T> = H	128	1					✓	✓	E	38	FLA			1
			256	1					✓	✓	E	70	FLA			
			512	1					✓	✓	E	134	FLA			
		<T> = S	128	1					✓	✓	E	29	FLA			
			256	1					✓	✓	E	52	FLA			
			512	1					✓	✓	E	98	FLA			
		<T> = D	128	1					✓	✓	E	43	FLA			
			256	1					✓	✓	E	80	FLA			
			512	1					✓	✓	E	154	FLA			
FDUP	FMOV (immediate, unpredicated)			1							4	FLA				
FEXPA				1							4	FL*				
FMAD				1					✓	✓		9	FL*			2
FMAX (immediate)				1					✓	✓		4	FLA			
FMAX (vectors)				1					✓	✓		4	FL*			
FMAXNM (immediate)				1					✓	✓		4	FLA			

Instruction	Alias	Control option	VL	# of μOP	Seq. decode	Pre-sync	Post-sync	Pack	Extra μOP	Blocking	Latency	Pipeline	# of FP	# of SP	FLOPS
FMAXNM (vectors)				1				✓	✓		4	FL*			
FMAXNMV		<V> = H	128	7	✓						4 / ([1]6 / [1,2]4) x 3	FL* / (FLA / FL*) x 3			
			256	9	✓						4 / ([1]6 / [1,2]4) x 4	FL* / (FLA / FL*) x 4			
			512	11	✓						4 / ([1]6 / [1,2]4) x 5	FL* / (FLA / FL*) x 5			
		<V> = S	128	5	✓						4 / ([1]6 / [1,2]4) x 2	FL* / (FLA / FL*) x 2			
			256	7	✓						4 / ([1]6 / [1,2]4) x 3	FL* / (FLA / FL*) x 3			
			512	9	✓						4 / ([1]6 / [1,2]4) x 4	FL* / (FLA / FL*) x 4			
		<V> = D	128	3	✓						4 / [1]6 / [1,2]4	FL* / FLA / FL*			
			256	5	✓						4 / ([1]6 / [1,2]4) x 2	FL* / (FLA / FL*) x 2			
			512	7	✓						4 / ([1]6 / [1,2]4) x 3	FL* / (FLA / FL*) x 3			
FMAXV		<V> = H	128	7	✓						4 / ([1]6 / [1,2]4) x 3	FL* / (FLA / FL*) x 3			
			256	9	✓						4 / ([1]6 / [1,2]4) x 4	FL* / (FLA / FL*) x 4			
			512	11	✓						4 / ([1]6 / [1,2]4) x 5	FL* / (FLA / FL*) x 5			
		<V> = S	128	5	✓						4 / ([1]6 / [1,2]4) x 2	FL* / (FLA / FL*) x 2			
			256	7	✓						4 / ([1]6 / [1,2]4) x 3	FL* / (FLA / FL*) x 3			
			512	9	✓						4 / ([1]6 / [1,2]4) x 4	FL* / (FLA / FL*) x 4			
		<V> = D	128	3	✓						4 / [1]6 / [1,2]4	FL* / FLA / FL*			
			256	5	✓						4 / ([1]6 / [1,2]4) x 2	FL* / (FLA / FL*) x 2			
			512	7	✓						4 / ([1]6 / [1,2]4) x 3	FL* / (FLA / FL*) x 3			
FMIN (immediate)				1				✓	✓		4	FLA			
FMIN (vectors)				1				✓	✓		4	FL*			
FMINNM (immediate)				1				✓	✓		4	FLA			
FMINNM (vectors)				1				✓	✓		4	FL*			
FMINNMV		<V> = H	128	7	✓						4 / ([1]6 / [1,2]4) x 3	FL* / (FLA / FL*) x 3			
			256	9	✓						4 / ([1]6 / [1,2]4) x 4	FL* / (FLA / FL*) x 4			
			512	11	✓						4 / ([1]6 / [1,2]4) x 5	FL* / (FLA / FL*) x 5			
		<V> = S	128	5	✓						4 / ([1]6 / [1,2]4) x 2	FL* / (FLA / FL*) x 2			
			256	7	✓						4 / ([1]6 / [1,2]4) x 3	FL* / (FLA / FL*) x 3			
			512	9	✓						4 / ([1]6 / [1,2]4) x 4	FL* / (FLA / FL*) x 4			
		<V> = D	128	3	✓						4 / [1]6 / [1,2]4	FL* / FLA / FL*			
			256	5	✓						4 / ([1]6 / [1,2]4) x 2	FL* / (FLA / FL*) x 2			
			512	7	✓						4 / ([1]6 / [1,2]4) x 3	FL* / (FLA / FL*) x 3			
FMINV		<V> = H	128	7	✓						4 / ([1]6 / [1,2]4) x 3	FL* / (FLA / FL*) x 3			
			256	9	✓						4 / ([1]6 / [1,2]4) x 4	FL* / (FLA / FL*) x 4			
			512	11	✓						4 / ([1]6 / [1,2]4) x 5	FL* / (FLA / FL*) x 5			
		<V> = S	128	5	✓						4 / ([1]6 / [1,2]4) x 2	FL* / (FLA / FL*) x 2			
			256	7	✓						4 / ([1]6 / [1,2]4) x 3	FL* / (FLA / FL*) x 3			
			512	9	✓						4 / ([1]6 / [1,2]4) x 4	FL* / (FLA / FL*) x 4			

Instruction	Alias	Control option	VL	# of μOP	Seq. decode	Pre-sync	Post-sync	Pack	Extra μOP	Blocking	Latency	Pipeline	# of FP	# of SP	FLOPS
<V> = D			128	3	✓						4 / [1]6 / [1,2]4	FL* / FLA / FL*			
			256	5	✓						4 / ([1]6 / [1,2]4) x 2	FL* / (FLA / FL*) x 2			
			512	7	✓						4 / ([1]6 / [1,2]4) x 3	FL* / (FLA / FL*) x 3			
FMLA				1				✓	✓		9	FL*			2
FMLA (indexed)				2							6 / [1]9	FLA / FLB			2
FMLS				1				✓	✓		9	FL*			2
FMLS (indexed)				2							6 / [1]9	FLA / FLB			2
FMSB				1				✓	✓		9	FL*			2
FMUL (immediate)				1				✓			9	FLA			1
FMUL (indexed)				2							6 / [1]9	FLA / FLB			1
FMUL (vectors, predicated)				1				✓			9	FL*			1
FMUL (vectors, unpredicated)				1							9	FL*			1
FMULX				1				✓			9	FL*			1
FNEG				1				✓			4	FL*			
FNMAD				1				✓	✓		9	FL*			2
FNMLA				1				✓	✓		9	FL*			2
FNMLS				1				✓	✓		9	FL*			2
FNMSB				1				✓	✓		9	FL*			2
FRECPE				1							4	FL*			
FRECPS				1							9	FLA			1
FRECPX				1				✓			4	FL*			
FRINTA				1				✓			9	FL*			
FRINTI				1				✓			9	FL*			
FRINTM				1				✓			9	FL*			
FRINTN				1				✓			9	FL*			
FRINTP				1				✓			9	FL*			
FRINTX				1				✓			9	FL*			
FRINTZ				1				✓			9	FL*			
FRSQRTE				1							4	FL*			
FRSQRTS				1							9	FLA			1
FSCALE				1				✓			9	FL*			1
FSQRT		<T> = H	128	1				✓		E	38	FLA			1
			256	1				✓		E	70	FLA			
			512	1				✓		E	134	FLA			
		<T> = S	128	1				✓		E	29	FLA			
			256	1				✓		E	52	FLA			
			512	1				✓		E	98	FLA			
		<T> = D	128	1				✓		E	43	FLA			
			256	1				✓		E	80	FLA			

Instruction	Alias	Control option	VL	# of μOP	Seq. decode	Pre-sync	Post-sync	Pack	Extra μOP	Blocking	Latency	Pipeline	# of FP	# of SP	FLOPS
			512	1				✓		E	154	FLA			
FSUB (immediate)				1				✓			9	FLA			1
FSUB (vectors, predicated)				1				✓			9	FL*			1
FSUB (vectors, unpredicated)				1							9	FL*			1
FSUBR (immediate)				1				✓			9	FLA			1
FSUBR (vectors)				1				✓			9	FL*			1
FTMAD				1				✓			9	FL*			2
FTSMUL				1							9	FL*			1
FTSSEL				1							4	FL*			
INCB				1							1	EX*			
INCD (scalar)				1							1	EX*			
INCD (vector)				1				✓			4	FL*			
INCH (scalar)				1							1	EX*			
INCH (vector)				1				✓			4	FL*			
INCP (scalar)				2							3+2+1 / [1]1	PRX+NULL+EXA / EXB			
INCP (vector)				1				✓			3+5+4	PRX + NULL + FLA			
INCW (scalar)				1							1	EX*			
INCW (vector)				1				✓			4	FL*			
INDEX (immediate, scalar)	<T> = {B H}		2								1+3+4 / 1+3+[1]9	EXA+NULL+FLA / EXA+NULL+FLA			
INDEX (immediates)		<T> = {S D}	1								1+3+9	EXA + NULL + FLA			
INDEX (scalar, immediate)	<T> = {B H}		2								4 / [1]9	FLA / FLA			
INDEX (scalars)		<T> = {S D}	1								9	FLA			
INSR (scalar)				1				✓			1+3+6	EXA + NULL + FLA			
INSR (SIMD&FP scalar)				1				✓			6	FLA			
LASTA (scalar)				1							6+1+18	FLA + NULL + EAG*	1	1	
LASTA (SIMD&FP scalar)				1							6	FLA			
LASTB (scalar)				1							6+1+18	FLA + NULL + EAG*	1	1	
LASTB (SIMD&FP scalar)				1							6	FLA			
LD1B (scalar plus immediate)				1							11	EAG*	1		
LD1B (scalar plus scalar)				1							11	EAG*	1		
LD1B (scalar plus vector)	32-bit unscaled offset		1								1+3+1+4+Pipe(11, 8)	EXA + NULL + FLA + FLA + Pipe((EAGA & EAGB), 8)	8	1	
LD1B (vector plus immediate)		32-bit element	1								1+3+4+Pipe(11, 4)	EXA + NULL + FLA + Pipe((EAGA & EAGB), 4)	4	1	
LD1D (scalar plus immediate)			1								11	EAG*	1		

Instruction	Alias	Control option	VL	# of μOP	Seq. decode	Pre-sync	Post-sync	Pack	Extra μOP	Blocking	Latency	Pipeline	# of FP	# of SP	FLOPS
LD1D (scalar plus scalar)				1							11	EAG*	1		
LD1D (scalar plus vector)				1							1+3+4+Pipe(11, 4)	EXA + NULL + FLA + Pipe((EAGA & EAGB), 4)	4	1	
LD1D (vector plus immediate)				1							4+Pipe(11, 4)	FLA + Pipe((EAGA & EAGB), 4)	4	1	
LD1H (scalar plus immediate)				1							11	EAG*	1		
LD1H (scalar plus scalar)				1							11	EAG*	1		
LD1H (scalar plus vector)		32-bit scaled offset, 32-bit unscaled offset		1							1+3+1+4+Pipe(11, 8)	EXA + NULL + FLA + FLA + Pipe((EAGA & EAGB), 8)	8	1	
				1							1+3+4+Pipe(11, 4)	EXA + NULL + FLA + Pipe((EAGA & EAGB), 4)	4	1	
LD1H (vector plus immediate)		32-bit element		1							1+4+Pipe(11, 8)	FLA + FLA + Pipe((EAGA & EAGB), 8)	8	1	
				1							4+Pipe(11, 4)	FLA + Pipe((EAGA & EAGB), 4)	4	1	
LD1RB				1							11	EAG*	1		
LD1RD				1							11	EAG*	1		
LD1RH				1							11	EAG*	1		
LD1RQB (scalar plus immediate)				1							11	EAG*	1		
LD1RQB (scalar plus scalar)				1							11	EAG*	1		
LD1RQD (scalar plus immediate)				1							11	EAG*	1		
LD1RQD (scalar plus scalar)				1							11	EAG*	1		
LD1RQH (scalar plus immediate)				1							11	EAG*	1		
LD1RQH (scalar plus scalar)				1							11	EAG*	1		
LD1RQW (scalar plus immediate)				1							11	EAG*	1		
LD1RQW (scalar plus scalar)				1							11	EAG*	1		
LD1RSB				1							11	EAG*	1		
LD1RSH				1							11	EAG*	1		
LD1RSW				1							11	EAG*	1		
LD1RW				1							11	EAG*	1		
LD1SB (scalar plus immediate)				1							11	EAG*	1		
LD1SB (scalar plus scalar)				1							11	EAG*	1		
LD1SB (scalar plus vector)		32-bit unscaled offset		1							1+3+1+4+Pipe(11, 8)	EXA + NULL + FLA + FLA + Pipe((EAGA & EAGB), 8)	8	1	
				1							1+3+4+Pipe(11, 4)	EXA + NULL + FLA + Pipe((EAGA & EAGB), 4)	4	1	
LD1SB (vector plus immediate)		32-bit element		1							1+4+Pipe(11, 8)	FLA + FLA + Pipe((EAGA & EAGB), 8)	8	1	
				1							4+Pipe(11, 4)	FLA + Pipe((EAGA & EAGB), 4)	4	1	
LD1SH (scalar plus immediate)				1							11	EAG*	1		
LD1SH (scalar plus scalar)				1							11	EAG*	1		
LD1SH (scalar plus vector)		32-bit scaled offset, 32-bit unscaled offset		1							1+3+1+4+Pipe(11, 8)	EXA + NULL + FLA + FLA + Pipe((EAGA & EAGB), 8)	8	1	
				1							1+3+4+Pipe(11, 4)	EXA + NULL + FLA + Pipe((EAGA & EAGB), 4)	4	1	
LD1SH (vector plus immediate)		32-bit element		1							1+4+Pipe(11, 8)	FLA + FLA + Pipe((EAGA & EAGB), 8)	8	1	
				1							4+Pipe(11, 4)	FLA + Pipe((EAGA & EAGB), 4)	4	1	
LD1SW (scalar plus immediate)				1							11	EAG*	1		
LD1SW (scalar plus scalar)				1							11	EAG*	1		

Instruction	Alias	Control option	VL	# of μOP	Seq. decode	Pre-sync	Post-sync	Pack	Extra μOP	Blocking	Latency	Pipeline	# of FP	# of SP	FLOPS
LD1SW (scalar plus vector)				1							1+3+4+Pipe(11, 4)	EXA + NULL + FLA + Pipe((EAGA & EAGB), 4)	4	1	
LD1SW (vector plus immediate)				1							4+Pipe(11, 4)	FLA + Pipe((EAGA & EAGB), 4)	4	1	
LD1W (scalar plus immediate)				1							11	EAG*	1		
LD1W (scalar plus scalar)				1							11	EAG*	1		
LD1W (scalar plus vector)		32-bit scaled offset, 32-bit unscaled offset		1							1+3+1+4+Pipe(11, 8)	EXA + NULL + FLA + FLA + Pipe((EAGA & EAGB), 8)	8	1	
				1							1+3+4+Pipe(11, 4)	EXA + NULL + FLA + Pipe((EAGA & EAGB), 4)	4	1	
LD1W (vector plus immediate)		32-bit element		1							1+4+Pipe(11, 8)	FLA + FLA + Pipe((EAGA & EAGB), 8)	8	1	
				1							4+Pipe(11, 4)	FLA + Pipe((EAGA & EAGB), 4)	1	1	
LD2B (scalar plus immediate)				3							1 / [1/2]((Pipe(11, 4)) x 2	EAG* / (Pipe(EAGA, 4)   Pipe(EAGB, 4)) x 2	8		
LD2B (scalar plus scalar)				3							1 / [1/2]((Pipe(11, 4)) x 2	EAG* / (Pipe(EAGA, 4)   Pipe(EAGB, 4)) x 2	8		
LD2D (scalar plus immediate)				2							11 / 11	EAG* / EAG*	2		
LD2D (scalar plus scalar)				3							1 / [1/2](11) x 2	EAG* / (EAG*) x 2	2		
LD2H (scalar plus immediate)				3							1 / [1/2]((Pipe(11, 4)) x 2	EAG* / (Pipe(EAGA, 4)   Pipe(EAGB, 4)) x 2	8		
LD2H (scalar plus scalar)				3							1 / [1/2]((Pipe(11, 4)) x 2	EAG* / (Pipe(EAGA, 4)   Pipe(EAGB, 4)) x 2	8		
LD2W (scalar plus immediate)				2							11 / 11	EAG* / EAG*	2		
LD2W (scalar plus scalar)				3							1 / [1/2](11) x 2	EAG* / (EAG*) x 2	2		
LD3B (scalar plus immediate)				4							1 / [1/2/3]((Pipe(11, 4)) x 3	EAG* / (Pipe(EAGA, 4)   Pipe(EAGB, 4)) x 3	12		
LD3B (scalar plus scalar)				4							1 / [1/2/3]((Pipe(11, 4)) x 3	EAG* / (Pipe(EAGA, 4)   Pipe(EAGB, 4)) x 3	12		
LD3D (scalar plus immediate)				3							11 / 11 / 11	EAG* / EAG* / EAG*	3		
LD3D (scalar plus scalar)				4							1 / [1/2/3](11) x 3	EAG* / (EAG*) x 3	3		
LD3H (scalar plus immediate)				4							1 / [1/2/3]((Pipe(11, 4)) x 3	EAG* / (Pipe(EAGA, 4)   Pipe(EAGB, 4)) x 3	12		
LD3H (scalar plus scalar)				4							1 / [1/2/3]((Pipe(11, 4)) x 3	EAG* / (Pipe(EAGA, 4)   Pipe(EAGB, 4)) x 3	12		
LD3W (scalar plus immediate)				3							11 / 11 / 11	EAG* / EAG* / EAG*	3		
LD3W (scalar plus scalar)				4							1 / [1/2/3](11) x 3	EAG* / (EAG*) x 3	3		
LD4B (scalar plus immediate)				5							1 / [1/2/3/4]((Pipe(11, 4)) x 4	EAG* / (Pipe(EAGA, 4)   Pipe(EAGB, 4)) x 4	16		
LD4B (scalar plus scalar)				5							1 / [1/2/3/4]((Pipe(11, 4)) x 4	EAG* / (Pipe(EAGA, 4)   Pipe(EAGB, 4)) x 4	16		
LD4D (scalar plus immediate)				4							11 / 11 / 11 / 11	EAG* / EAG* / EAG* / EAG*	4		
LD4D (scalar plus scalar)				5							1 / [1/2/3/4](11) x 4	EAG* / (EAG*) x 4	4		
LD4H (scalar plus immediate)				5							1 / [1/2/3/4]((Pipe(11, 4)) x 4	EAG* / (Pipe(EAGA, 4)   Pipe(EAGB, 4)) x 4	16		
LD4H (scalar plus scalar)				5							1 / [1/2/3/4]((Pipe(11, 4)) x 4	EAG* / (Pipe(EAGA, 4)   Pipe(EAGB, 4)) x 4	16		
LD4W (scalar plus immediate)				4							11 / 11 / 11 / 11	EAG* / EAG* / EAG* / EAG*	4		
LD4W (scalar plus scalar)				5							1 / [1/2/3/4](11) x 4	EAG* / (EAG*) x 4	4		
LDFF1B (scalar plus scalar)				1							11	EAG*	1		
LDFF1B (scalar plus vector)		32-bit unscaled offset		1							1+3+1+4+Pipe(11, 8)	EXA + NULL + FLA + FLA + Pipe((EAGA & EAGB), 8)	8	1	
				1							1+3+4+Pipe(11, 4)	EXA + NULL + FLA + Pipe((EAGA & EAGB), 4)	4	1	
LDFF1B (vector plus immediate)		32-bit element		1							1+4+Pipe(11, 8)	FLA + FLA + Pipe((EAGA & EAGB), 8)	8	1	
				1							4+Pipe(11, 4)	FLA + Pipe((EAGA & EAGB), 4)	4	1	
LDFF1D (scalar plus scalar)				1							11	EAG*	1		

Instruction	Alias	Control option	VL	# of μOP	Seq. decode	Pre-sync	Post-sync	Pack	Extra μOP	Blocking	Latency	Pipeline	# of FP	# of SP	FLOPS
LDFF1D (scalar plus vector)				1							1+3+4+Pipe(11, 4)	EXA + NULL + FLA + Pipe((EAGA & EAGB), 4)	4	1	
LDFF1D (vector plus immediate)				1							4+Pipe(11, 4)	FLA + Pipe((EAGA & EAGB), 4)	4	1	
LDFF1H (scalar plus scalar)				1							11	EAG*	1		
LDFF1H (scalar plus vector)		32-bit scaled offset, 32-bit unscaled offset		1							1+3+1+4+Pipe(11, 8)	EXA + NULL + FLA + FLA + Pipe((EAGA & EAGB), 8)	8	1	
				1							1+3+4+Pipe(11, 4)	EXA + NULL + FLA + Pipe((EAGA & EAGB), 4)	4	1	
LDFF1H (vector plus immediate)		32-bit element		1							1+4+Pipe(11, 8)	FLA + FLA + Pipe((EAGA & EAGB), 8)	8	1	
				1							4+Pipe(11, 4)	FLA + Pipe((EAGA & EAGB), 4)	4	1	
LDFF1SB (scalar plus scalar)				1							11	EAG*	1		
LDFF1SB (scalar plus vector)		32-bit unscaled offset		1							1+3+1+4+Pipe(11, 8)	EXA + NULL + FLA + FLA + Pipe((EAGA & EAGB), 8)	8	1	
				1							1+3+4+Pipe(11, 4)	EXA + NULL + FLA + Pipe((EAGA & EAGB), 4)	4	1	
LDFF1SB (vector plus immediate)		32-bit element		1							1+1+4+Pipe(11, 8)	FLA + FLA + Pipe((EAGA & EAGB), 8)	8	1	
				1							4+Pipe(11, 4)	FLA + Pipe((EAGA & EAGB), 4)	4	1	
LDFF1SH (scalar plus scalar)				1							11	EAG*	1		
LDFF1SH (scalar plus vector)		32-bit scaled offset, 32-bit unscaled offset		1							1+3+1+4+Pipe(11, 8)	EXA + NULL + FLA + FLA + Pipe((EAGA & EAGB), 8)	8	1	
				1							1+3+4+Pipe(11, 4)	EXA + NULL + FLA + Pipe((EAGA & EAGB), 4)	4	1	
LDFF1SH (vector plus immediate)		32-bit element		1							1+4+Pipe(11, 8)	FLA + FLA + Pipe((EAGA & EAGB), 8)	8	1	
				1							4+Pipe(11, 4)	FLA + Pipe((EAGA & EAGB), 4)	4	1	
LDFF1SW (scalar plus scalar)				1							11	EAG*	1		
LDFF1SW (scalar plus vector)				1							1+3+4+Pipe(11, 4)	EXA + NULL + FLA + Pipe((EAGA & EAGB), 4)	4	1	
LDFF1SW (vector plus immediate)				1							4+Pipe(11, 4)	FLA + Pipe((EAGA & EAGB), 4)	4	1	
LDFF1W (scalar plus scalar)				1							11	EAG*	1		
LDFF1W (scalar plus vector)		32-bit scaled offset, 32-bit unscaled offset		1							1+3+1+4+Pipe(11, 8)	EXA + NULL + FLA + FLA + Pipe((EAGA & EAGB), 8)	8	1	
				1							1+3+4+Pipe(11, 4)	EXA + NULL + FLA + Pipe((EAGA & EAGB), 4)	4	1	
LDFF1W (vector plus immediate)		32-bit element		1							1+4+Pipe(11, 8)	FLA + FLA + Pipe((EAGA & EAGB), 8)	8	1	
				1							4+Pipe(11, 4)	FLA + Pipe((EAGA & EAGB), 4)	4	1	
LDNF1B				1							11	EAG*	1		
LDNF1D				1							11	EAG*	1		
LDNF1H				1							11	EAG*	1		
LDNF1SB				1							11	EAG*	1		
LDNF1SH				1							11	EAG*	1		
LDNF1SW				1							11	EAG*	1		
LDNF1W				1							11	EAG*	1		
LDNT1B (scalar plus immediate)				1							11	EAG*	1		
LDNT1B (scalar plus scalar)				1							11	EAG*	1		
LDNT1D (scalar plus immediate)				1							11	EAG*	1		
LDNT1D (scalar plus scalar)				1							11	EAG*	1		
LDNT1H (scalar plus immediate)				1							11	EAG*	1		

Instruction	Alias	Control option	VL	# of μOP	Seq. decode	Pre-sync	Post-sync	Pack	Extra μOP	Blocking	Latency	Pipeline	# of FP	# of SP	FLOPS
LDNT1H (scalar plus scalar)				1							11	EAG*	1		
LDNT1W (scalar plus immediate)				1							11	EAG*	1		
LDNT1W (scalar plus scalar)				1							11	EAG*	1		
LDR (predicate)				1							11	EAGA	1		
LDR (vector)				1							11	EAGA	1		
LSL (immediate, predicated)				1					✓	✓	4	FL*			
LSL (immediate, unpredicated)				1							4	FL*			
LSL (vectors)				1					✓	✓	4	FL*			
LSL (wide elements, predicated)				1					✓	✓	4	FL*			
LSL (wide elements, unpredicated)				1							4	FL*			
LSLR				1					✓	✓	4	FL*			
LSR (immediate, predicated)				1					✓	✓	4	FL*			
LSR (immediate, unpredicated)				1							4	FL*			
LSR (vectors)				1					✓	✓	4	FL*			
LSR (wide elements, predicated)				1					✓	✓	4	FL*			
LSR (wide elements, unpredicated)				1							4	FL*			
LSRR				1					✓	✓	4	FL*			
MAD				1					✓	✓	9	FL*			
MLA				1					✓	✓	9	FL*			
MLS				1					✓	✓	9	FL*			
MOVPRFX (predicated)				1							4	FL*			
MOVPRFX (unpredicated)				1							4	FL*			
MSB				1					✓	✓	9	FL*			
MUL (immediate)				1					✓		9	FLA			
MUL (vectors)				1					✓		9	FL*			
NAND				1							3	PRX			
NANDS				1							3	PRX			
NEG				1					✓		4	FL*			
NOR				1							3	PRX			
NORS				1							3	PRX			
NOT (vector)				1					✓		4	FL*			
ORN (predicates)				1							3	PRX			
ORNS				1							3	PRX			
ORR (immediate)	ORN (immediate)			1					✓		4	FLA			
ORR (predicates)	MOV (predicate, unpredicated)			1							3	PRX			
				1							3	PRX			
ORR (vectors, predicated)				1					✓	✓	4	FL*			
ORR (vectors, unpredicated)	MOV (vector, unpredicated)			1							4	FL*			

Instruction	Alias	Control option	VL	# of μOP	Seq. decode	Pre-sync	Post-sync	Pack	Extra μOP	Blocking	Latency	Pipeline	# of FP	# of SP	FLOPS
				1							4	FL*			
ORRS	MOVS (unpredicated)			1							3	PRX			
				1							3	PRX			
ORV		<V> = B		10	✓						4 / ([1]6 / [1,2]4) x 3 / [1]4 / [1]4 / [1]4	FL* / (FLA / FL*) x 3 / FL* / FL* / FL*			
		<V> = H		9	✓						4 / ([1]6 / [1,2]4) x 3 / [1]4 / [1]4	FL* / (FLA / FL*) x 3 / FL* / FL*			
		<V> = S		8	✓						4 / ([1]6 / [1,2]4) x 3 / [1]4	FL* / (FLA / FL*) x 3 / FL*			
		<V> = D	128	3	✓						4 / [1]6 / [1,2]4	FL* / FLA / FL*			
			256	5	✓						4 / ([1]6 / [1,2]4) x 2	FL* / (FLA / FL*) x 2			
			512	7	✓						4 / ([1]6 / [1,2]4) x 3	FL* / (FLA / FL*) x 3			
PFALSE				1							3	PRX			
PFIRST				1							3	PRX			
PNEXT				1							3	PRX			
PRFB (scalar plus immediate)				1							NA	EAG*		1	
PRFB (scalar plus scalar)				1							NA	EAG*		1	
PRFB (scalar plus vector)		32-bit scaled offset		1							1+3+1+4+Pipe(NA, 8)	EXA + NULL + FLA + FLA + Pipe((EAGA & EAGB), 8)	8	1	
				1							1+3+4+Pipe(NA, 4)	EXA + NULL + FLA + Pipe((EAGA & EAGB), 4)	4	1	
PRFB (vector plus immediate)		32-bit element		1							1+4+Pipe(NA, 8)	FLA + FLA + Pipe((EAGA & EAGB), 8)	8	1	
				1							4+Pipe(NA, 4)	FLA + Pipe((EAGA & EAGB), 4)	4	1	
PRFD (scalar plus immediate)				1							NA	EAG*		1	
PRFD (scalar plus scalar)				1							NA	EAG*		1	
PRFD (scalar plus vector)		32-bit scaled offset		1							1+3+1+4+Pipe(NA, 8)	EXA + NULL + FLA + FLA + Pipe((EAGA & EAGB), 8)	8	1	
				1							1+3+4+Pipe(NA, 4)	EXA + NULL + FLA + Pipe((EAGA & EAGB), 4)	4	1	
PRFD (vector plus immediate)		32-bit element		1							1+4+Pipe(NA, 8)	FLA + FLA + Pipe((EAGA & EAGB), 8)	8	1	
				1							4+Pipe(NA, 4)	FLA + Pipe((EAGA & EAGB), 4)	4	1	
PRFH (scalar plus immediate)				1							NA	EAG*		1	
PRFH (scalar plus scalar)				1							NA	EAG*		1	
PRFH (scalar plus vector)		32-bit scaled offset		1							1+3+1+4+Pipe(NA, 8)	EXA + NULL + FLA + FLA + Pipe((EAGA & EAGB), 8)	8	1	
				1							1+3+4+Pipe(NA, 4)	EXA + NULL + FLA + Pipe((EAGA & EAGB), 4)	4	1	
PRFH (vector plus immediate)		32-bit element		1							1+4+Pipe(NA, 8)	FLA + FLA + Pipe((EAGA & EAGB), 8)	8	1	
				1							4+Pipe(NA, 4)	FLA + Pipe((EAGA & EAGB), 4)	4	1	
PRFW (scalar plus immediate)				1							NA	EAG*		1	
PRFW (scalar plus scalar)				1							NA	EAG*		1	
PRFW (scalar plus vector)		32-bit scaled offset		1							1+3+1+4+Pipe(NA, 8)	EXA + NULL + FLA + FLA + Pipe((EAGA & EAGB), 8)	8	1	
				1							1+3+4+Pipe(NA, 4)	EXA + NULL + FLA + Pipe((EAGA & EAGB), 4)	4	1	
PRFW (vector plus immediate)		32-bit element		1							1+4+Pipe(NA, 8)	FLA + FLA + Pipe((EAGA & EAGB), 8)	8	1	
				1							4+Pipe(NA, 4)	FLA + Pipe((EAGA & EAGB), 4)	4	1	
PTEST				1							3	PRX			
PTRUE				1							3	PRX			
PTRUES				1							3	PRX			

Instruction	Alias	Control option	VL	# of μOP	Seq. decode	Pre-sync	Post-sync	Pack	Extra μOP	Blocking	Latency	Pipeline	# of FP	# of SP	FLOPS
PUNPKHI				1							3	PRX			
PUNPKLO				1							3	PRX			
RBIT				1				✓			4	FL*			
RDFFR (predicated)				1							3	PRX			
RDFFR (unpredicated)				1							3	PRX			
RDFFRS				1							3	PRX			
RDVL				1							1	EX*			
REV (predicate)				1							3	PRX			
REV (vector)				1							6	FLA			
REVB				1				✓			4	FL*			
REVH				1				✓			4	FL*			
REVV				1				✓			4	FL*			
SABD				1				✓	✓		4	FL*			
SADDV		<T> = B		11	✓						4 / [1]4 / ([1,2]4 / [1]6) x 3 / [1,2]4 / [1]4 / [1]4	FL* / FL* / (FL* / FLA) x 3 / FL* / FL* / FL*			
		<T> = H		10	✓						4 / [1]4 / ([1,2]4 / [1]6) x 3 / [1,2]4 / [1]4	FL* / FL* / (FL* / FLA) x 3 / FL* / FL*			
		<T> = S	128	5	✓						4 / [1]4 / [1,2]4 / [1]6 / [1,2]4	FL* / FL* / FL* / FLA / FL*			
			256	7	✓						4 / [1]4 / ([1,2]4 / [1]6) x 2 / [1,2]4	FL* / FL* / (FL* / FLA) x 2 / FL*			
			512	9	✓						4 / [1]4 / ([1,2]4 / [1]6) x 3 / [1,2]4	FL* / FL* / (FL* / FLA) x 3 / FL*			
SCVTF				1				✓			9	FL*			
SDIV		<T> = S	128	1				✓	✓	E	33	FLA			
			256	1				✓	✓	E	60	FLA			
			512	1				✓	✓	E	114	FLA			
		<T> = D	128	1				✓	✓	E	49	FLA			
			256	1				✓	✓	E	92	FLA			
			512	1				✓	✓	E	178	FLA			
SDIVR		<T> = S	128	1				✓	✓	E	33	FLA			
			256	1				✓	✓	E	60	FLA			
			512	1				✓	✓	E	114	FLA			
		<T> = D	128	1				✓	✓	E	49	FLA			
			256	1				✓	✓	E	92	FLA			
			512	1				✓	✓	E	178	FLA			
SDOT (indexed)				2							6 / [1]9	FLA / FLB			
SDOT (vectors)				1							9	FL*			
SEL (predicates)	MOV (predicate, predicated, merging)			1							3	PRX			
				1							3	PRX			
SEL (vectors)	MOV (vector, predicated)			1							4	FL*			
				1							4	FL*			
SETFFR				1							NA				
SMAX (immediate)				1				✓			4	FLA			

Instruction	Alias	Control option	VL	# of μOP	Seq. decode	Pre-sync	Post-sync	Pack	Extra μOP	Blocking	Latency	Pipeline	# of FP	# of SP	FLOPS
SMAX (vectors)				1				✓	✓		4	FL*			
SMAXV		<V> = B		10	✓						4 / ([1]6 / [1,2]4) x 3 / [1]4 / [1]4 / [1]4	FL* / (FLA / FL*) x 3 / FL* / FL* / FL*			
		<V> = H		9	✓						4 / ([1]6 / [1,2]4) x 3 / [1]4 / [1]4	FL* / (FLA / FL*) x 3 / FL* / FL*			
		<V> = S		8	✓						4 / ([1]6 / [1,2]4) x 3 / [1]4	FL* / (FLA / FL*) x 3 / FL*			
		<V> = D	128	3	✓						4 / [1]6 / [1,2]4	FL* / FLA / FL*			
			256	5	✓						4 / ([1]6 / [1,2]4) x 2	FL* / (FLA / FL*) x 2			
			512	7	✓						4 / ([1]6 / [1,2]4) x 3	FL* / (FLA / FL*) x 3			
SMIN (immediate)				1				✓			4	FLA			
SMIN (vectors)				1				✓	✓		4	FL*			
SMINV		<V> = B		10	✓						4 / ([1]6 / [1,2]4) x 3 / [1]4 / [1]4 / [1]4	FL* / (FLA / FL*) x 3 / FL* / FL* / FL*			
		<V> = H		9	✓						4 / ([1]6 / [1,2]4) x 3 / [1]4 / [1]4	FL* / (FLA / FL*) x 3 / FL* / FL*			
		<V> = S		8	✓						4 / ([1]6 / [1,2]4) x 3 / [1]4	FL* / (FLA / FL*) x 3 / FL*			
		<V> = D	128	3	✓						4 / [1]6 / [1,2]4	FL* / FLA / FL*			
			256	5	✓						4 / ([1]6 / [1,2]4) x 2	FL* / (FLA / FL*) x 2			
			512	7	✓						4 / ([1]6 / [1,2]4) x 3	FL* / (FLA / FL*) x 3			
SMULH				1				✓			9	FL*			
SPLICE				1				✓			6	FLA			
SQADD (immediate)				1				✓			4	FL*			
SQADD (vectors)				1							4	FL*			
SQDECB				1						P	1+1	(EXA + EXA)   (EXB + EXB)			
SQDEC D (scalar)				1						P	1+1	(EXA + EXA)   (EXB + EXB)			
SQDEC D (vector)				1				✓			4	FL*			
SQDEC H (scalar)				1						P	1+1	(EXA + EXA)   (EXB + EXB)			
SQDEC H (vector)				1				✓			4	FL*			
SQDEC P (scalar)				2						/ P	3+2+1 / 1+[1]1	PRX + NULL + EXA / EXB + EXB			
SQDEC P (vector)				1				✓			3+5+4	PRX + NULL + FLA			
SQDEC W (scalar)				1						P	1+1	(EXA + EXA)   (EXB + EXB)			
SQDEC W (vector)				1				✓			4	FL*			
SQINCB				1						P	1+1	(EXA + EXA)   (EXB + EXB)			
SQINCD (scalar)				1						P	1+1	(EXA + EXA)   (EXB + EXB)			
SQINCD (vector)				1				✓			4	FL*			
SQINCH (scalar)				1						P	1+1	(EXA + EXA)   (EXB + EXB)			
SQINCH (vector)				1				✓			4	FL*			
SQINCP (scalar)				2						/ P	3+2+1 / 1+[1]1	PRX + NULL + EXA / EXB + EXB			
SQINCP (vector)				1				✓			3+5+4	PRX + NULL + FLA			
SQINCW (scalar)				1						P	1+1	(EXA + EXA)   (EXB + EXB)			
SQINCW (vector)				1				✓			4	FL*			
SQSUB (immediate)				1				✓			4	FL*			
SQSUB (vectors)				1							4	FL*			

Instruction	Alias	Control option	VL	# of μOP	Seq. decode	Pre-sync	Post-sync	Pack	Extra μOP	Blocking	Latency	Pipeline	# of FP	# of SP	FLOPS
ST1B (scalar plus immediate)				1							NA, NA	EAG*, FLA	1	1	
ST1B (scalar plus scalar)				1							NA, NA	EAG*, FLA	1	1	
ST1B (scalar plus vector)		32-bit unscaled offset	8								(1+3+4+Pipe(NA, 2) / 1+3+NA) x 4	(EXA + NULL + FLA + Pipe((EAGA & EAGB), 2) / EXA + NULL + FLA) x 4	16	16	
				4							(1+3+4+Pipe(NA, 2) / 1+3+NA) x 2	(EXA + NULL + FLA + Pipe((EAGA & EAGB), 2) / EXA + NULL + FLA) x 2			
ST1B (vector plus immediate)		32-bit element	8								(4+Pipe(NA, 2) / NA) x 4	(FLA + FLA + Pipe((EAGA & EAGB), 2) / FLA) x 4	16	16	
				4							(4+Pipe(NA, 2) / NA) x 2	(FLA + Pipe((EAGA & EAGB), 2) / FLA) x 2			
ST1D (scalar plus immediate)				1							NA, NA	EAG*, FLA	1	1	
ST1D (scalar plus scalar)				1							NA, NA	EAG*, FLA	1	1	
ST1D (scalar plus vector)				4							(1+3+4+Pipe(NA, 2) / 1+3+NA) x 2	(EXA + NULL + FLA + Pipe((EAGA & EAGB), 2) / EXA + NULL + FLA) x 2	8	8	
ST1D (vector plus immediate)				4							(4+Pipe(NA, 2) / NA) x 2	(FLA + Pipe((EAGA & EAGB), 2) / FLA) x 2	8	8	
ST1H (scalar plus immediate)				1							NA, NA	EAG*, FLA	1	1	
ST1H (scalar plus scalar)				1							NA, NA	EAG*, FLA	1	1	
ST1H (scalar plus vector)		32-bit scaled offset, 32-bit unscaled offset	8								(1+3+4+Pipe(NA, 2) / 1+3+NA) x 4	(EXA + NULL + FLA + Pipe((EAGA & EAGB), 2) / EXA + NULL + FLA) x 4	16	16	
				4							(1+3+4+Pipe(NA, 2) / 1+3+NA) x 2	(EXA + NULL + FLA + Pipe((EAGA & EAGB), 2) / EXA + NULL + FLA) x 2			
ST1H (vector plus immediate)		32-bit element	8								(4+Pipe(NA, 2) / NA) x 4	(FLA + FLA + Pipe((EAGA & EAGB), 2) / FLA) x 4	16	16	
				4							(4+Pipe(NA, 2) / NA) x 2	(FLA + Pipe((EAGA & EAGB), 2) / FLA) x 2			
ST1W (scalar plus immediate)				1							NA, NA	EAG*, FLA	1	1	
ST1W (scalar plus scalar)				1							NA, NA	EAG*, FLA	1	1	
ST1W (scalar plus vector)		32-bit scaled offset, 32-bit unscaled offset	8								(1+3+4+Pipe(NA, 2) / 1+3+NA) x 4	(EXA + NULL + FLA + Pipe((EAGA & EAGB), 2) / EXA + NULL + FLA) x 4	16	16	
				4							(1+3+4+Pipe(NA, 2) / 1+3+NA) x 2	(EXA + NULL + FLA + Pipe((EAGA & EAGB), 2) / EXA + NULL + FLA) x 2			
ST1W (vector plus immediate)		32-bit element	8								(4+Pipe(NA, 2) / NA) x 4	(FLA + Pipe((EAGA & EAGB), 2) / FLA) x 4	16	16	
				4							(4+Pipe(NA, 2) / NA) x 2	(FLA + Pipe((EAGA & EAGB), 2) / FLA) x 2			
ST2B (scalar plus immediate)				3							1 / [1/2](Pipe(NA, 4), Pipe(NA, 4)) x 2	EAG* / ((Pipe(EAGA, 4)   Pipe(EAGB, 4)), Pipe(FLA, 4)) x 2	8	8	
ST2B (scalar plus scalar)				3							1 / [1/2](Pipe(NA, 4), Pipe(NA, 4)) x 2	EAG* / ( (Pipe(EAGA, 4)   Pipe(EAGB, 4)), Pipe(FLA, 4) ) x 2	8	8	
ST2D (scalar plus immediate)				2							NA, NA / NA, NA	EAG*, FLA / EAG*, FLA	2	2	
ST2D (scalar plus scalar)				3							1 / [1/2](NA, NA) x 2	EAG* / (EAG*, FLA) x 2	2	2	
ST2H (scalar plus immediate)				3							1 / [1/2](Pipe(NA, 4), Pipe(NA, 4)) x 2	EAG* / ( (Pipe(EAGA, 4)   Pipe(EAGB, 4)), Pipe(FLA, 4) ) x 2	8	8	
ST2H (scalar plus scalar)				3							1 / [1/2](Pipe(NA, 4), Pipe(NA, 4)) x 2	EAG* / ( (Pipe(EAGA, 4)   Pipe(EAGB, 4)), Pipe(FLA, 4) ) x 2	8	8	
ST2W (scalar plus immediate)				2							NA, NA / NA, NA	EAG*, FLA / EAG*, FLA	2	2	
ST2W (scalar plus scalar)				3							1 / [1/2](NA, NA) x 2	EAG* / (EAG*, FLA) x 2	2	2	
ST3B (scalar plus immediate)				4							1 / [1/2/3](Pipe(NA, 4), Pipe(NA, 4)) x 3	EAG* / ( (Pipe(EAGA, 4)   Pipe(EAGB, 4)), Pipe(FLA, 4) ) x 3	12	12	
ST3B (scalar plus scalar)				4							1 / [1/2/3](Pipe(NA, 4), Pipe(NA, 4)) x 3	EAG* / ( (Pipe(EAGA, 4)   Pipe(EAGB, 4)), Pipe(FLA, 4) ) x 3	12	12	
ST3D (scalar plus immediate)				3							NA, NA / NA, NA / NA, NA	(EAG*, FLA) x 3	3	3	
ST3D (scalar plus scalar)				4							1 / [1/2/3](NA, NA) x 3	EAG* / (EAG*, FLA) x 3	3	3	
ST3H (scalar plus immediate)				4							1 / [1/2/3](Pipe(NA, 4), Pipe(NA, 4)) x 3	EAG* / ( (Pipe(EAGA, 4)   Pipe(EAGB, 4)), Pipe(FLA, 4) ) x 3	12	12	
ST3H (scalar plus scalar)				4							1 / [1/2/3](Pipe(NA, 4), Pipe(NA, 4)) x 3	EAG* / ( (Pipe(EAGA, 4)   Pipe(EAGB, 4)), Pipe(FLA, 4) ) x 3	12	12	

Instruction	Alias	Control option	VL	# of μOP	Seq. decode	Pre-sync	Post-sync	Pack	Extra μOP	Blocking	Latency	Pipeline	# of FP	# of SP	FLOPS
ST3W (scalar plus immediate)				3							NA,NA / NA,NA / NA,NA	EAG*, FLA / EAG*, FLA / EAG*, FLA	3	3	
ST3W (scalar plus scalar)				4							1 / [1/2/3](NA,NA) x 3	EAG* / (EAG*, FLA) x 3	3	3	
ST4B (scalar plus immediate)				5							1 / [1/2/3/4](Pipe(NA, 4), Pipe(NA, 4)) x 4	EAG* / ( (Pipe(EAGA, 4)   Pipe(EAGB, 4)), Pipe(FLA, 4) ) x 4	16	16	
ST4B (scalar plus scalar)				5							1 / [1/2/3/4]( Pipe(NA, 4), Pipe(NA, 4) ) x 4	EAG* / ( (Pipe(EAGA, 4)   Pipe(EAGB, 4)), Pipe(FLA, 4) ) x 4	16	16	
ST4D (scalar plus immediate)				4							NA,NA / NA,NA / NA,NA / NA,NA	EAG*, FLA / EAG*, FLA / EAG*, FLA / EAG*, FLA	4	4	
ST4D (scalar plus scalar)				5							1 / [1/2/3/4](NA,NA) x 4	EAG* / (EAG*, FLA) x 4	4	4	
ST4H (scalar plus immediate)				5							1 / [1/2/3/4](Pipe(NA, 4), Pipe(NA, 4)) x 4	EAG* / ( (Pipe(EAGA, 4)   Pipe(EAGB, 4)), Pipe(FLA, 4) ) x 4	16	16	
ST4H (scalar plus scalar)				5							1 / [1/2/3/4]( Pipe(NA, 4), Pipe(NA, 4) ) x 4	EAG* / ( (Pipe(EAGA, 4)   Pipe(EAGB, 4)), Pipe(FLA, 4) ) x 4	16	16	
ST4W (scalar plus immediate)				4							NA,NA / NA,NA / NA,NA / NA,NA	EAG*, FLA / EAG*, FLA / EAG*, FLA / EAG*, FLA	4	4	
ST4W (scalar plus scalar)				5							1 / [1/2/3/4](NA,NA) x 4	EAG* / (EAG*, FLA) x 4	4	4	
STNT1B (scalar plus immediate)				1							NA, NA	EAG*, FLA	1	1	
STNT1B (scalar plus scalar)				1							NA, NA	EAG*, FLA	1	1	
STNT1D (scalar plus immediate)				1							NA, NA	EAG*, FLA	1	1	
STNT1D (scalar plus scalar)				1							NA, NA	EAG*, FLA	1	1	
STNT1H (scalar plus immediate)				1							NA, NA	EAG*, FLA	1	1	
STNT1H (scalar plus scalar)				1							NA, NA	EAG*, FLA	1	1	
STNT1W (scalar plus immediate)				1							NA, NA	EAG*, FLA	1	1	
STNT1W (scalar plus scalar)				1							NA, NA	EAG*, FLA	1	1	
STR (predicate)				1							NA, NA	EAGA, PRX	1	1	
STR (vector)				1							NA, NA	EAGA, FLA	1	1	
SUB (immediate)				1				✓			4	FL*			
SUB (vectors, predicated)				1				✓	✓		4	FL*			
SUB (vectors, unpredicated)				1							4	FL*			
SUBR (immediate)				1				✓			4	FLA			
SUBR (vectors)				1				✓	✓		4	FL*			
SUNPKHI				1							6	FLA			
SUNPKLO				1							6	FLA			
SXTB				1				✓			4	FL*			
SXTH				1				✓			4	FL*			
SXTW				1				✓			4	FL*			
TBL				1							6	FLA			
TRN1 (predicates)				1							3	PRX			
TRN1 (vectors)				1							6	FLA			
TRN2 (predicates)				1							3	PRX			
TRN2 (vectors)				1							6	FLA			
UABD				1				✓	✓		4	FL*			
UADDV	<T> = B		11	✓							4 / [1]4 / ([1,2]4 / [1]6) x 3 / [1,2]4 / [1]4 / [1]4	FL* / FL* / (FL* / FLA) x 3 / FL* / FL* / FL*			
	<T> = H		10	✓							4 / [1]4 / ([1,2]4 / [1]6) x 3 / [1,2]4 / [1]4	FL* / FL* / (FL* / FLA) x 3 / FL* / FL*			
	<T> = {S D}		128	5	✓						4 / [1]4 / [1,2]4 / [1]6 / [1,2]4	FL* / FL* / FL* / FLA / FL*			

Instruction	Alias	Control option	VL	# of μOP	Seq. decode	Pre-sync	Post-sync	Pack	Extra μOP	Blocking	Latency	Pipeline	# of FP	# of SP	FLOPS
			256	7	✓						4 / $[^{11}4 / ([^{1,2}4 / [^{11}6] \times 2 / [^{1,2}4]$	FL* / FL* / (FL* / FLA) x 2 / FL*			
			512	9	✓						4 / $[^{11}4 / ([^{1,2}4 / [^{11}6] \times 3 / [^{1,2}4]$	FL* / FL* / (FL* / FLA) x 3 / FL*			
UCVTF				1				✓			9	FL*			
UDIV		<T> = S	128	1				✓	✓	E	33	FLA			
			256	1				✓	✓	E	60	FLA			
			512	1				✓	✓	E	114	FLA			
		<T> = D	128	1				✓	✓	E	49	FLA			
			256	1				✓	✓	E	92	FLA			
			512	1				✓	✓	E	178	FLA			
UDIVR		<T> = S	128	1				✓	✓	E	33	FLA			
			256	1				✓	✓	E	60	FLA			
			512	1				✓	✓	E	114	FLA			
		<T> = D	128	1				✓	✓	E	49	FLA			
			256	1				✓	✓	E	92	FLA			
			512	1				✓	✓	E	178	FLA			
UDOT (indexed)				2							6 / $[^{11}9$	FLA / FLB			
UDOT (vectors)				1							9	FL*			
UMAX (immediate)				1				✓			4	FLA			
UMAX (vectors)				1				✓	✓		4	FL*			
UMAXV		<V> = B	10	✓							4 / $[^{11}6 / [^{1,2}4] \times 3 / [^{11}4 / [^{11}4 / [^{11}4$	FL* / (FLA / FL*) x 3 / FL* / FL* / FL*			
			9	✓							4 / $[^{11}6 / [^{1,2}4] \times 3 / [^{11}4 / [^{11}4$	FL* / (FLA / FL*) x 3 / FL* / FL*			
			8	✓							4 / $[^{11}6 / [^{1,2}4] \times 3 / [^{11}4$	FL* / (FLA / FL*) x 3 / FL*			
		<V> = D	128	3	✓						4 / $[^{11}6 / [^{1,2}4$	FL* / FLA / FL*			
			256	5	✓						4 / $[^{11}6 / [^{1,2}4] \times 2$	FL* / (FLA / FL*) x 2			
			512	7	✓						4 / $[^{11}6 / [^{1,2}4] \times 3$	FL* / (FLA / FL*) x 3			
UMIN (immediate)				1				✓			4	FLA			
UMIN (vectors)				1				✓	✓		4	FL*			
UMINV		<V> = B	10	✓							4 / $[^{11}6 / [^{1,2}4] \times 3 / [^{11}4 / [^{11}4 / [^{11}4$	FL* / (FLA / FL*) x 3 / FL* / FL* / FL*			
			9	✓							4 / $[^{11}6 / [^{1,2}4] \times 3 / [^{11}4 / [^{11}4$	FL* / (FLA / FL*) x 3 / FL* / FL*			
			8	✓							4 / $[^{11}6 / [^{1,2}4] \times 3 / [^{11}4$	FL* / (FLA / FL*) x 3 / FL*			
		<V> = D	128	3	✓						4 / $[^{11}6 / [^{1,2}4$	FL* / FLA / FL*			
			256	5	✓						4 / $[^{11}6 / [^{1,2}4] \times 2$	FL* / (FLA / FL*) x 2			
			512	7	✓						4 / $[^{11}6 / [^{1,2}4] \times 3$	FL* / (FLA / FL*) x 3			
UMULH				1				✓			9	FL*			
UQADD (immediate)				1				✓			4	FL*			
UQADD (vectors)				1				✓			4	FL*			
UQDECB				1						P	1+1	(EXA + EXA)   (EXB + EXB)			
UQDECD (scalar)				1						P	1+1	(EXA + EXA)   (EXB + EXB)			

Instruction	Alias	Control option	VL	# of μOP	Seq. decode	Pre-sync	Post-sync	Pack	Extra μOP	Blocking	Latency	Pipeline	# of FP	# of SP	FLOPS
UQDECD (vector)				1				✓			4	FL*			
UQDECH (scalar)				1						P	1+1	(EXA + EXA)   (EXB + EXB)			
UQDECH (vector)				1				✓			4	FL*			
UQDEC P (scalar)				2						/ P	3+2+1 / 1+[1]1	PRX + NULL + EXA / EXB + EXB			
UQDEC P (vector)				1					✓		3+5+4	PRX + NULL + FLA			
UQDECW (scalar)				1						P	1+1	(EXA + EXA)   (EXB + EXB)			
UQDECW (vector)				1				✓			4	FL*			
UQINCB				1						P	1+1	(EXA + EXA)   (EXB + EXB)			
UQINCD (scalar)				1						P	1+1	(EXA + EXA)   (EXB + EXB)			
UQINCD (vector)				1				✓			4	FL*			
UQINCH (scalar)				1						P	1+1	EX* + EX*			
UQINCH (vector)				1				✓			4	FL*			
UQINCP (scalar)				2						/ P	3+2+1 / 1+[1]1	PRX + NULL + EXA / EXB + EXB			
UQINCP (vector)				1				✓			3+5+4	PRX + NULL + FLA			
UQINCW (scalar)				1						P	1+1	(EXA + EXA)   (EXB + EXB)			
UQINCW (vector)				1				✓			4	FL*			
UQSUB (immediate)				1				✓			4	FL*			
UQSUB (vectors)				1							4	FL*			
UUNPKHI				1							6	FLA			
UUNPKLO				1							6	FLA			
UXTB				1				✓			4	FL*			
UXTH				1				✓			4	FL*			
UXTW				1				✓			4	FL*			
UZP1 (predicates)				1							3	PRX			
UZP1 (vectors)				1							6	FLA			
UZP2 (predicates)				1							3	PRX			
UZP2 (vectors)				1							6	FLA			
WHILELE				1							1+3	EXA + PRX			
WHILELO				1							1+3	EXA + PRX			
WHILELS				1							1+3	EXA + PRX			
WHILELT				1							1+3	EXA + PRX			
WRFFR				2	✓	✓	✓				NA / 3	/ PRX			
ZIP1 (predicates)				1							3	PRX			
ZIP1 (vectors)				1							6	FLA			
ZIP2 (predicates)				1							3	PRX			
ZIP2 (vectors)				1							6	FLA			