FUJITSU

SPARC64™ XIfx Extensions

Distribution : Public
Privilege Levels : Nonprivileged

第 14 版
2016/09/13

Fujitsu Limited

Fujitsu Limited
4-1-1 Kamikodanaka
Nakahara-ku, Kawasaki, 211-8588
Japan

目次

Preface

本仕様書は、SPARC64™ XIfx の論理仕様を定義する。

参考にする各仕様書は以下の通りである。

- SPARC64™ VIIIfx Extensions (日本語版)
  Ver 15, 26 Apr. 2010
  http://img.jp.fujitsu.com/downloads/jp/jhpc/sparc64viiifx-extensionsj.pdf
- SPARC64™ IXfx Extensions (日本語版)
  Ver 12, 2 Dec. 2013
  http://img.jp.fujitsu.com/downloads/jp/jhpc/sparc64ixfx-extensionsj.pdf
- SPARC® Joint Programming Specification (JPS1): Commonality
  Working Draft 1.0.4, 31 May 2002
  http://www.fujitsu.com/downloads/PRMPWR/JPS1-R1.0.4-Common-pub.pdf
  (本文では JPS1 と呼ぶ)
- UltraSPARC Architecture 2011

  Draft D0.9.6, 21 May 2014

  http://www.oracle.com/technetwork/server-storage/sun-sparc-enterprise/documentation/140521-ua2011-d096-p-ext-2306580.pdf

  (本文では UA2011 仕様と呼ぶ)

1. Document Overview

1.1. Navigating the SPARC64™ XIfx Extensions

SPARC64™ XIfx は、SPARCV9 に準拠し SPARC64™ VIIIfx 及び SPARC64™ IXfx と UA2011(Oracle SPARC Architecture 2011)をベースに High Performance Computing 向けに拡張を行ったプロセッサである。

本仕様書では SPARC64™ XIfx について UA2011 とは異なる実装を行っている個所及び、拡張を行った箇所について記載しており、共通の実装を行っている個所については記載しない。

必要に応じて UA2011 を参照されたい。

また、一部定義については SPARC64™ VIIIfx 及び SPARC64™ IXfx を参照しているため、そちらも参照されたい。

1.2. Fonts and Notations

1.2.1. フォント

- レジスタおよびレジスタのフィールドは Arial で REG, REG.field と表記する。ASI レジスタのフィールドが単独で出現する場合もこのフォントを使う。
- ASI 名は Courier で `ASI_NAME` と表記する。頭に ASI_をつける。
- 例外は Arial の斜体で *exception_name* と表記する。
- 命令は Courier で `INSTRUCTION` と表記する。大文字にする。
- CPU ステートは Courier で `CPU_state` と表記する。
- 将来の拡張用に予約されているレジスタフィールド等は Times Roman の斜体で *reserved* または— (Symbol フォントの BE_{16})と表記する。

1.2.2. 表記法

表記法は主として JPS1 に準拠する。

- 数字は 10 進数で 10 以外の基数の場合、1000_2 のように右下に基数を付す。
- 読みやすいよう、$1000\ 0000_{16}$ のように空白を挟むことがある。
- Verilog の表記を借用することがある。桁数'b0 または桁数'ha0 など。Verilog 表記のときは右下の基数はつけない。
- レジスタは R[数字], F[数字]。
- レジスタ表記について、どのような場合においても倍精度レジスタしか使用されないとき Fd[数字]表記を利用する。単精度レジスタしか使用されないとき Fs[数字]表記を利用する。XAR 拡張により使用するレジスタが単精度・倍精度で変更されるとき F[数字]表記を

それぞれ使用する。ただし、F[数字]表記については、XAR 拡張により使用するレジスタが変更される場合、Fd[数字]を使い明示することがある。

- シンボルおよび命令名称の選択は{}, *, n を使う。
  - {}は省略可能な文字列を表わす。ASI_PRIMARY{_LITTLE}は、ASI_PRIMARY または ASI_PRIMARY_LITTLE を表わす。
- {}に|が入っている場合、|で区切られた文字列の任意の一つを選択する。FMUL{s|d}は、FMULs または FMULd を表わす。省略可能な文字列も含めたい場合は FMUL{|s|d}と表記する。*および n は、置換可能なすべての文字列、数値を代替する。*DAE_**は、*DAE_invalid_asi*, *DAE_nc_page*, *DAE_nfo_page*, *DAE_privilege_violation*, および *DAE_side_effect_page* すべてを表わし、*spill_n_normal* は *spill_0_normal*, *spill_1_normal*, *spill_2_normal*, *spill_3_normal*, *spill_4_normal*, *spill_5_normal*, *spill_6_normal*, *spill_7_normal* を表わす。ビット列は<a:b>。
- ビット列の結合はコロン 2 つ“::”。

1.2.3. 仮想言語表記について

一部インストラクションの説明において、C 言語や, Veilog-HDL に似た仮想的な言語を使用する。

仮想言語表記で使用している制御構文や代入については以下のように扱う

- “IF”表記は C, Verilog の “if”
- “ELSEIF”表記は C, Verilog の “else if”
- “FOR”表記は C, Verilog の “for”
- “RETURN”表記は C の “return”
- “←”表記は C の “=”, Verilog の “<=”

ビット列を表すのにビットの範囲を表す“<31:0>”及び 1 ビットを表す “<15>”を使用する。

expr1 ? expr2 : expr3 表記が使用可能であり、C, Verilog と同様に expr1 の条件が TRUE(1)であるなら、出力として expr2 が選択される。また expr1 の条件が FALSE(0)であるなら、出力として expr3 が選択される。

各種演算子と”(“, ”)”は C, Verilog と等価として扱う。“{“ “}”表記は制御構文のブロックを表し C の “{“ “}” と等価であり, Verilog の “begin” “end”と対応する。

ASI レジスタは ASI_NAME.field として暗黙的に使用可能とする。

レジスタは R[レジスタ番号], F[レジスタ番号]で暗黙的に定義され、SIMD 演算に使用されるレジスタを示す場合、F[レジスタ番号][要素番号]で表記する。

メモリは MEM[アドレス, 幅]で定義し、ビックエンディアンでアドレスを先頭とした幅(バイト)が暗黙的に定義される。

命令内のフィールドを使用する際は IW.フィールド名で指定する。(ただし、IW.rd, IW.rs1, IW.rs2 については、暗黙的にそれぞれ rd, rs1, rs2 として使用)

一時変数は暗黙的に使用する。

値の表記は桁数’b0、桁数’ha0 または桁数’d95 などの Verilog 表記を用い、可読性を上げるため’ha_0 のようにアンダースコアで接続することがある。

サブルーチンを定義可能で、VOID は戻り値なし、BIT_N は N ビットデータが戻り値として扱う。

1.2.4. reserved の扱い

将来の拡張用に予約されていることを表わす *reserved* または—は不定値を意味する。両者の使い分けは、*reserved* は将来の拡張が予想されているもので、— は用途が未定なものを意図している。レジスタ等のフィールドで *reserved* が使われているときは、フィールド説明の表に予約してある理由を記述する。— は説明を記述しない。

1.2.5. アクセス属性

レジスタやレジスタフィールドのアクセス属性は記号で表記する。記号とその意味の対応は下表の通り。

表 1-1　アクセス属性

アクセス属性	対象	動作	
		読み出し	書き込み
	フィールド	不定値	無視される
R	レジスタ、フィールド	値が読み出される	無視される
RO	レジスタ、フィールド	値が読み出される	許可されていない
R0	フィールド	0 が読み出される	無視される
W	レジスタ、フィールド	不定値が読み出される	値が書き込まれる
WO	レジスタ、フィールド	許可されていない	値が書き込まれる
RW	レジスタ、フィールド	値が読み出される	値が書き込まれる
RW1C	フィールド	値が読み出される	1 を書き込むと 0 でクリアされる [i]
RWQF	フィールド	値が読み出される	レジスタに依存した条件が成立したとき、書き込み地がレジスタに保持される。他の場合、書き込みは無視される。
RWS	レジスタ	レジスタの値が読み込まれる。または常に 0 が読み込まれる	書き込み値は無視されるが、書き込みを契機に規定された side effect が発生する

1.2.6. Informational Notes

以下の注釈を適宜挿入する。

Compatibility Note　SPARC V8/V9, JPS1, SPARC64™ VIIIfx, SPARC64™ IXfx との互換性に関する説明。

Note　注意事項。

Programming Note　ソフトウェアの使用法を説明する。

[i] クリアされる範囲は個々に定義される。

2. Definitions

コアメモリグループ(CMG)：CPU チップ内に 2 グループあり、1 つのグループは 16 コアの演算コアと 1 コアのアシスタントコアの計 17 コア、L2 キャッシュ及びメモリで構成される。CMG の定義には、IO (ICC2, PCIe, SI バス)は含まれない。

アシスタントコア：OS や IO 処理などを担当するコア。コアメモリグループごとに 1 コアの構成となる。

演算コア：主に演算を担当するコア。コアメモリグループごとに 16 コアで構成される。

要素(Element)：要素と Element は同じ意味で利用される。SIMD 演算においては複数のデータを一括で処理するが、この処理する 64 ビット幅のデータの単位を 1 要素と呼ぶ。また、一括で処理する複数のデータを区別するため、1 つ目のデータ及び、その演算を行う演算器を要素 0 もしくは Element-0 と呼び、順番に 2 つ目のデータ及び、その演算を行う演算器を要素 1 もしくは Element-1 と呼ぶ。SPARC64™ XIfx では、要素 3/Element-3 まで存在する。

HPC-ACE2：High Performance Computing –Arithmetic Computational Extensions 2 の略で、SPARC64™ XIfx 独自に拡張された論理仕様の総称。SIMD 幅拡張、HPC 用途の命令、整数演算の SIMD 拡張などを含む。HPC-ACE の上位互換を有する。

VCPU(Virtual Processor)：システムや OS から 1 つの論理プロセッサ相当として識別され、独立したアーキテクチャステートを持つ。SPARC64™ XIfx では、1 つの物理 CPU コアあたり 1 VCPU。

上記以外の用語の定義は UA2011 及び SPARC64™ IXfx Extensions 参照。

3. Architectural Overview

特徴

- HPC-ACE / HPC-ACE2
- 4-wide SIMD
- サポートする VA は 64 ビット
- VA は 64 ビットフル実装、hold なし
- ノンキャッシャブル領域はローカル ROM の命令のみ実行可能
- NWINDOWS = 8

諸元

- 34 コア / チップ （(16 演算コア +1 アシスタントコア)）×2
- SMT 実装なし
- コアごとに L1 命令キャッシュ 64KB / 4way、L1 データキャッシュ 64KB / 4way、ラインサイズは 256 バイト
- 16 コア（17 コア）ごとに L2 キャッシュ 12MB / 24way（チップあたり 24MB)、ラインサイズは 256 バイト
- メイン TLB はセットアソシエティブ TLB のみ。命令 512 エントリ/ 4way、データ 512 エントリ/ 4way、ページサイズは 4 種類（8KB、512KB、4MB、32MB）

4. Data Formats

整数数、浮動小数点数については、UA2011 4.Data Formats 参照。

4.1. Floating-Point Data Formats

4.1.1. Floating Point, Dual Single Precision

SPARC64™ XIfx では倍精度レジスタを半分に区切って 2 つの単精度浮動小数点数として扱う命令を追加した。これらの命令を単精度倍幅(Dual Single Precision)演算命令と呼ぶ。この単精度倍幅演算命令は単精度浮動小数点演算を行う命令であるが、どのような場合においても倍精度レジスタを使用する。

これら演算に使用するフォーマットを単精度倍幅浮動小数点数と呼び、図 4-1 にフォーマットを示す。上位 32 ビットと下位 32 ビットを独立した 2 つの単精度浮動小数点数として扱う。各フィールドの詳細は UA2011 4.2.1 Floating Point, Single Precision 及び TABLE4-3 参照。

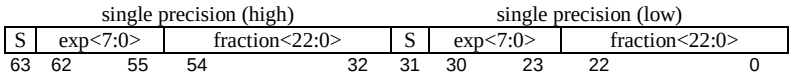


図 4-1　単精度倍幅浮動小数点数のフォーマット

5. Registers

5.1. reserved Register Fields

reserved フィールドの読み出しは基本的に不定値とする。ただし互換性の観点から、0 読み出しと明記する場合もある。

reserved フィールドへの書き込みは将来的な拡張に備えて 0 を書き込む必要がある。

5.2. 整数レジスタ R

5.2.1. 汎用整数レジスタ

R[0] - R[7]の 8 つのレジスタは、汎用整数レジスタ（グローバルレジスタ）である。SPARC64™ XIfx には 3 組（MAXPGL + 1）の汎用整数レジスタがあり、そのうち GL で指定した 1 組 8 つのレジスタを R[0] – R[7]として使用することができる。GL のどの値に対応するグローバルレジスタの組においても、R[0]は固定的に値 0 を持ち、読み出しには 0 が返り、書き込みは無視される。

汎用整数レジスタの仕様書上の表記は R[0] – R[7]または g[0] – g[7]で、GL 値ごとに区別する表記法は特に設けない。

5.2.2. HPC-ACE 拡張汎用整数レジスタ

SPARC64™ XIfx では SPARC64™ VIIIfx と同様に SPARC V9 から整数レジスタを 32 個拡張している。拡張した整数レジスタは仕様書上 R[32] - R[63]または xg[0] - xg[31]と表記される。

SPARC V9 の命令セットでは、レジスタ指定フィールドは 5 ビットが割り当てられている。しかし、HPC-ACE 拡張汎用整数レジスタはレジスタ番号が 32 から 63 までなので、既存の命令セットでは指定することができない。SPARC64™ XIfx では HPC-ACE 拡張レジスタを使うかどうかを XAR レジスタで指定する。詳細は 35 ページを参照。

HPC-ACE 拡張汎用整数レジスタは、整数レジスタを使う大部分の命令で使用できるが、一部の命令では使用できない。どの命令で使えるかは個々の命令定義を参照。

HPC-ACE 拡張汎用整数レジスタは、GL や CWP の影響を受けない。R[32] - R[63]は、GL や CWP の値によらず、常に同一のレジスタを指す。

5.2.3. Windowed R Registers

R[8] – R[31]は register window である。Register window については UA2011 5.2.2 を参照されたい。

SPARC64™ XIfx においては Register Window を構成するレジスタセットとして N_REG_WINDOWS = 8 を実装している。

5.2.4. Special R Registers

汎用整数レジスタのうち 2 つは、仕様により使い方が規定されている。

- R[0]レジスタの値は常にゼロである。R[0]への書き込みは無視される。
- CALL 命令は、その命令のアドレスを R[15]レジスタに書き込む。

また、LDTW, LDTWA, STTW, および STTWA 命令は連続する 2 つの汎用整数レジスタを使用する命令である。これらの命令ではレジスタフィールドの最低位ビットは使われないが、0 でなければならない。

LDTW, LDTWA 命令で R[0], R[1]の組が指定された場合、R[1]だけが更新される。STTW, STTWA 命令で R[0], R[1]の組が指定された場合、R[0]からは 0 が読み出されるので、低位 4 バイトには 0 が書かれ、高位 4 バイトに R[1]の内容が書かれる。

LDTW, LDTWA, STTW, および STTWA 命令で奇数番号のレジスタを指定すると、*illegal_instruction* 例外が発生する。

5.3. Floating-Point Registers

SPARC64™VIIIfx において、従来定義されていた Floating-Point Registers（UA2011 5.3 参照）に対してレジスタを追加し全部で 256 本の倍精度レジスタが使用可能となった。これらの HPC-ACE 拡張レジスタを使うかどうかを XAR レジスタで指定する(SPARC64™ VIIIfx 仕様 5.1.4 参照)。SPARC64™ XIfx ではさらに SIMD(Single Instruction Multiple-Data)演算に使用する 256 本の倍精度レジスタを追加する。

追加するレジスタは Basic Floating-Point Registers、Extended Floating-Point Registers と組み合わせ、4 本の倍精度レジスタを 1 組として SIMD 演算に使用する。SPARC64™ XIfx では SIMD を構成する演算それぞれを、要素もしくは Element と呼び、0 からの追番をつけて識別する。それぞれの要素に対応する倍精度レジスタをそれぞれ Element-N Floating Point Registers(0≦N≦3) と呼ぶ。

> **Compatibility Note** SPARC64™VIIIfx 及び SPARC64™IXfx において Basic Floating-Point Registers 及び Extended Floating-Point Registers と呼ばれていたレジスタは、それぞれ Element-0 Floating-Point Registers 及び Element-1 Floating-Point Registers にエイリアスされる。

SIMD 演算を行わない場合に拡張浮動小数点レジスタを使用する場合、f[2n](n=0-255)によりレジスタを指定する。

5.3.1. SIMD 演算時のレジスタビュー

浮動小数点レジスタを使う命令の大部分は、XAR.v = 1 かつ XAR.simd = 1 のときに SIMD 拡張命令となり、1 命令で XASR.simd_mode で指定される有効な要素で演算を実行する。

XASR.simd_mode = 0 のとき、有効な要素は Element-0, Element-1 の 2 要素(128bit 幅)となり 1 命令で 2 演算[ii]を実行する。このとき、HPC-ACE 互換としても振る舞う。SIMD 演算で使われるレジスタの組は、Element-0 Floating-Point Registers と Element-1 Floating-Point Registers であり、f[2n] (n=0-127[iii])を指定して使用する。このとき Element-1 に使われるレジスタは、Non-SIMD 時に f[2n+256](n=0-127)で指定できるものと同一である。SPARC64™ XIfx で追加されたレジスタは演算に使用できない。

[ii] 単精度倍幅演算命令では 4 演算

[iii] 一部命令（FM{ADD|SUB}{s|d}, FNM{ADD|SUB}{s|d}, FSHIFTORX, FPMADDX, FPMADDXHI）の一部オペランドには f[2n](n=0-255)が指定可能なものがある。詳細は 7 章参照。

XASR.simd_mode = 1 のとき、有効な要素は Element-0, 1, 2, 3 の 4 要素(256bit 幅)となり 1 命令で 4 演算[iv]を実行する。このとき、倍精度レジスタ 4 本を 1 つの組として f[2n](n=0-127[iii]) で指定する。

Programming Note XAR.v = 1, XAR.simd = 0 のとき、f[2n](n=0-127)のレジスタ(Element-0 Floating-Point Registers)もしくは、f[2n+256](n=0-127)のレジスタ(Element-1 Floating-Point Registers)に書き込みを行ったとき、f[2n]の Element-2 及び Element-3 Floating-Point Register の値は不定となる。
また、XAR.v = 1, XAR.simd = 1, XASR.simd_mode = 0 のとき、f[2n]/f[2n+256] レジスタが書き込まれた場合、Element-2 及び Element-3 Floating-Point Registers の f[2n]の値は不定となる。
書込みを行っていない Element-2 及び Element-3 Floating-Point Registers の f[2n]の値は保持される。
書込み先のレジスタとモードにより値が変更される範囲を表 5-1 にまとめる。

表 5-1 書込み先 Floating-Point Registers 及びモードとその影響範囲

書込み先	モード	Element-0	Element-1	Element-2,3
f[n](n = 0-31) 単精度	Non-SIMD (XAR.v = 0)	書込み値	変化せず	不定
f[2n](n = 0-31) 倍精度	Non-SIMD (XAR.v = 0)	書込み値	変化せず	不定
f[2n](n = 0-127)	Non-SIMD (XAR.v = 1)	書込み値	変化せず	不定
f[2n](n = 128-255)	Non-SIMD (XAR.v = 1)	変化せず	書込み値	不定
f[2n](n = 0-127)	2-wide SIMD	書込み値	書込み値	不定
f[2n](n = 0-127)	4-wide SIMD	書込み値	書込み値	書込み値

SIMD 演算を実施しないかつ、HPC-ACE レジスタ拡張を行うとき（XAR.v = 1 かつ XAR.simd = 0）、Element-0 Floating-Point Registers, Element-1 Floating-Point Registers は f[2n](n = 0-255)で個別にアクセスできるが、SPARC64™ XIfx で追加する Element-2 Floating-Point Registers と Element-3 Floating-Point Registers は単一の倍精度浮動小数点レジスタとしてアクセスできない。

以下に各モード・XAR 拡張時の Floating-Point Registers の論理ビューを示す。網掛けされたレジスタは、そのモード・XAR 拡張時には使用できないことを示している。

表 5-2 SPARC V9[v] Floating-Point Registers 論理ビュー

HPC-ACE	Basic Floating-Point Registers		Extend Floating-Point Registers		
HPC-ACE2	Element-0 Floating Point Registers		Element-1 Floating Point Registers	Element-2 Floating Point Registers	Element-3 Floating Point Registers
倍精度 [vi]	63 32	31 0	63 0	63 0	63 0
f[0]	f[0]	f[1]			
f[2]	f[2]	f[3]			
f[4]	f[4]	f[5]			
:	:	:			
f[30]	f[30]	f[31]			
	f[32]				
	f[34]				
	:				
	f[62]				

[iv] 単精度倍幅演算命令では 8 演算
[v] XAR.v = 0 のとき
[vi] V9 レジスタで、単精度と倍精度のレジスタが重複している領域のみ記載

表 5-3　XAR 拡張レジスタ使用、Non-SIMD[vii]時 Floating-Point Registers 論理ビュー

HPC-ACE	Basic Floating-Point Registers	Extend Floating-Point Registers		
HPC-ACE2	Element-0 Floating Point Registers	Element-1 Floating Point Registers	Element-2 Floating Point Registers	Element-3 Floating Point Registers
	63 0	63 0	63 0	63 0
	f[0]	f[256]		
	f[2]	f[258]		
	f[4]	f[260]		
	:	:		
	f[30]	f[286]		
	f[32]	f[288]		
	f[34]	f[290]		
	:	:		
	f[62]	f[318]		
	:	:		
	f[252]	f[508]		
	f[254]	f[510]		

表 5-4　HPC-ACE 互換 2-wide SIMD[viii]時 Floating-Point Registers 論理ビュー

HPC-ACE	Basic Floating-Point Registers	Extend Floating-Point Registers		
HPC-ACE2	Element-0 Floating Point Registers	Element-1 Floating Point Registers	Element-2 Floating Point Registers	Element-3 Floating Point Registers
	63 0	63 0	63 0	63 0
	f[0]			
	f[2]			
	f[4]			
	:			
	f[30]			
	f[32]			
	f[34]			
	:			
	f[62]			
	:			
	f[252]			
	f[254]			

表 5-5　HPC-ACE2 4-wide SIMD[ix]時 Floating-Point Registers 論理ビュー

HPC-ACE	Basic Floating-Point Registers	Extend Floating-Point Registers		
HPC-ACE2	Element-0 Floating Point Registers	Element-1 Floating Point Registers	Element-2 Floating Point Registers	Element-3 Floating Point Registers
	63 0	63 0	63 0	63 0
	f[0]			
	f[2]			
	f[4]			
	:			
	f[30]			
	f[32]			
	f[34]			
	:			
	f[62]			
	:			
	f[252]			
	f[254]			

[vii] XAR.v = 1 かつ XAR.simd = 0 のとき（XASR.simd_mode は 0 の場合も 1 の場合も同様である）
[viii] XAR.v = 1, XAR.simd = 1 かつ XASR.simd_mode = 0 のとき
[ix] XAR.v = 1, XAR.simd = 1 かつ XASR.simd_mode = 1 のとき

5.3.2. 倍精度レジスタの単精度利用

SPARC64™ VIIIfx より倍精度レジスタを単精度浮動小数点演算でも使用可能となった。SPARC64™ XIfx では、追加された単精度倍幅演算命令を除きこの仕様を引き継ぐ。HPC-ACE の仕様と同様に SPARC V9 で定義された倍精度レジスタ、SPARC64 VIIIfx で追加されたものに加え、SPARC64™ XIfx で追加されたレジスタも単精度浮動小数点演算に使うことができる。単精度浮動小数点演算命令で倍精度レジスタを指定するには、その命令の実行時点で XAR.v = 1 であればよい。したがって、単精度で SIMD 演算を行うときは倍精度レジスタを使うことになる。

倍精度レジスタを単精度浮動小数点演算に使用した際のデータの流れを図 5-1 に示し、使用時の注意点を記載する。

- 偶数番号レジスタのみが使用可能である
- レジスタの値は bit<63:32>の上位 4 バイトを単精度値と解釈し、bit<31:0>の下位 4 バイトは無視される
- 演算結果やロード結果はレジスタの上位 4 バイトに書き込まれ、下位 4 バイトには 0 が書き込まれる

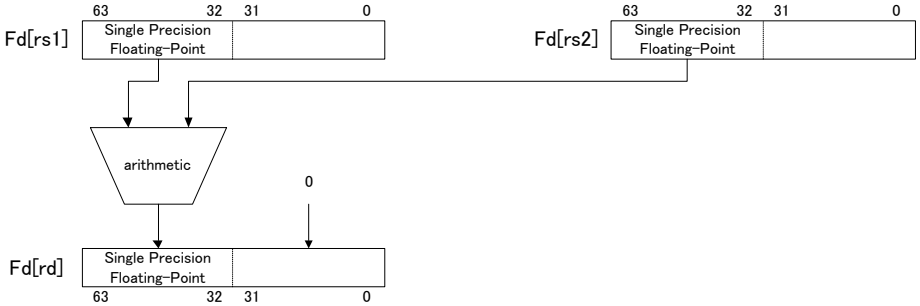


図 5-1 倍精度レジスタの単精度利用

5.3.3. 倍精度レジスタの単精度倍幅利用

SPARC64™ XIfx では倍精度レジスタを半分に区切って 2 つの単精度浮動小数点データとして扱う命令を追加した。これらの命令を単精度倍幅演算命令と呼ぶ。この単精度倍幅演算命令は単精度浮動小数点演算を行う命令であるが、どのような場合においても倍精度レジスタを使用する。

単精度倍幅演算命令以外の通常の単精度浮動小数点演算命令は、5.3.2 に記載した通り、SPARC64™ VIIIfx, SPARC64™ IXfx と同様に動作する。単精度倍幅演算命令実行時のデータの流れを図 5-2 に示し、使用時の注意点を記載する。

- 偶数番号レジスタのみが使用可能である
- レジスタの値は bit<63:32>の上位 4 バイトと bit<31:0>の下位 4 バイトをそれぞれ別の 2 つの単精度値と解釈する
- 上位 4 バイトデータ同士の演算結果をレジスタの上位 4 バイトに書き込み、下位 4 バイトデータ同士の演算結果をレジスタの下位 4 バイトに書き込む
- 上位 4 バイトデータと下位 4 バイトデータを演算することはない
- 倍精度レジスタへの単精度倍幅演算データのロードストア命令は 4 バイトアラインでアクセス可能な、LDDFDS, STDFDS 命令の使用を推奨する
- それぞれの演算で別の例外を検出する可能性があるが、優先順位に基づき適切に処理を行う（場合によっては、両方の例外要因の OR が記録される）

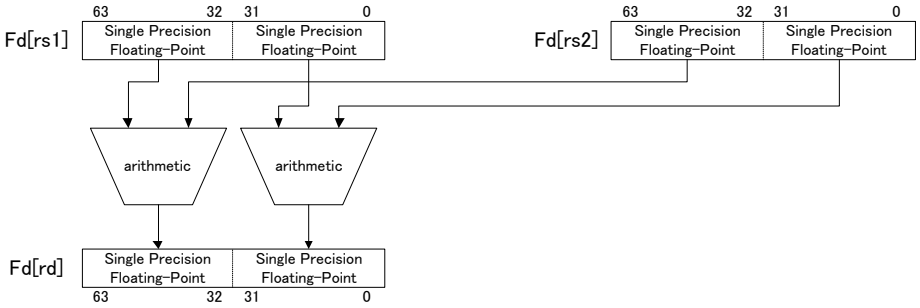


図 5-2　倍精度レジスタの単精度倍幅利用

5.3.4. Floating-Point Registers Number Encoding

単精度、倍精度、および 4 倍精度レジスタは、浮動小数点演算命令の 5 ビットのレジスタフィールドでは、それぞれ異なる方法でエンコードされる。HPC-ACE 拡張浮動小数点レジスタを使う場合はさらに異なる方法が使われる。表 5-6 は、SPARC V9 で定義された浮動小数点レジスタのエンコード法を示す。ここで、レジスタフィールドの 5 ビットを b<4>…b<0>で表わしている。b<4>が最上位ビット、b<0>が最下位ビットである。

表 5-6　浮動小数点レジスタのエンコード (SPARC V9)

精度	6 ビットのレジスタ番号						命令で指定する 5 ビットのレジスタ番号				
単精度	0	b<4>	b<3>	b<2>	b<1>	b<0>	b<4>	b<3>	b<2>	b<1>	b<0>
倍精度	b<5>	b<4>	b<3>	b<2>	b<1>	0	b<4>	b<3>	b<2>	b<1>	b<5>
4 倍精度	b<5>	b<4>	b<3>	b<2>	0	0	b<4>	b<3>	b<2>	0	b<5>

HPC-ACE 拡張浮動小数点レジスタを指示する場合は XAR レジスタを使用する。XAR レジスタには命令後の rs1, rs2, rs3, および rd フィールドに対応する拡張フィールド urs1, urs2, urs3, および urd フィールドがある。拡張フィールドは各々3 ビット幅で、表 5-6 のデコード後の 6 ビットの上位に結合して 9 ビットのレジスタ番号となる。HPC-ACE 拡張浮動小数点レジスタを使う命令は、すべて倍精度レジスタを使用するので、最下位ビットは常に 0 となり、Fd[0] – Fd[510]の 256 本のレジスタが使用される。

SIMD 演算を行う場合、Fd[0] – Fd[254]の倍精度レジスタを複数本束ねた 128 組のレジスタが使用可能である。

Compatibility Note　SPARC64™ VIIIfx, SPARC64™ IXfx において、一部の SIMD 演算では、Fd[256]-Fd[510]を指定可能なものがあった。これらの命令の論理仕様上の定義を変更する。仕様の記述上の表記は変更されるが命令の動作は変更されない。表記が変更されることで影響を受けるのは、`FM{ADD|SUB}{s|d}, FNM{ADD|SUB}{s|d}`命令である。変更内容については、7 章を参照。

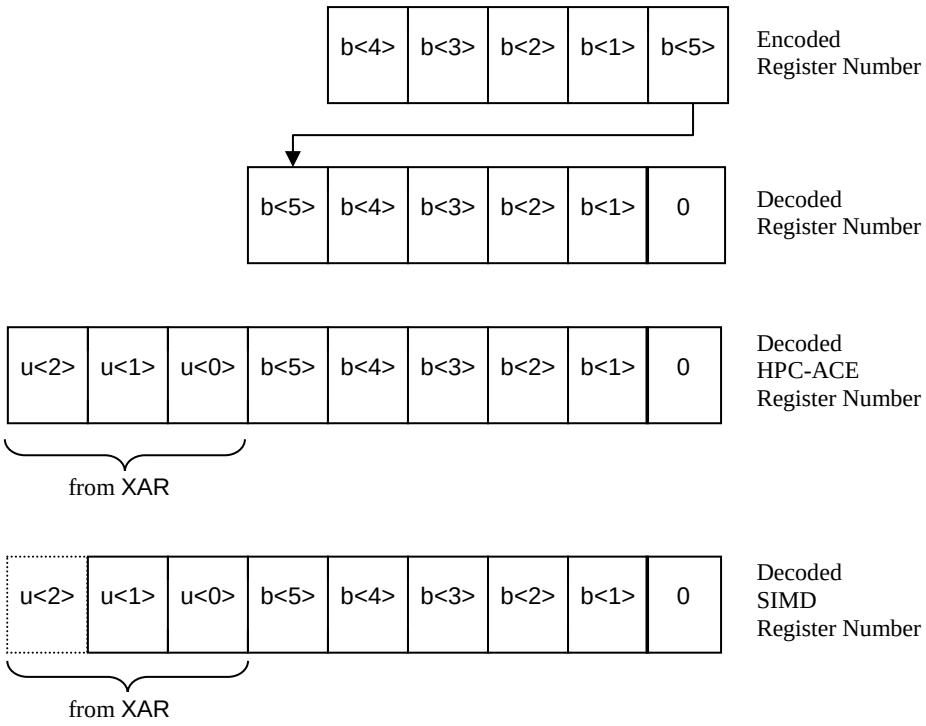
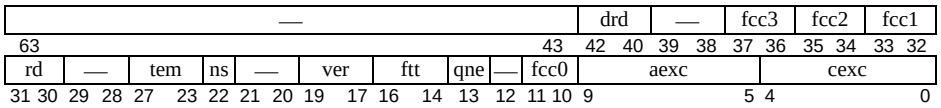


図 5-3　HPC-ACE 拡張浮動小数点レジスタのエンコード

5.4. Floating-Point State Register (FSR)



ビット	フィールド	アクセス	説明
42:40	drd	R0	常に 0。
37:36	fcc3	RW	fcc0 参照。
35:34	fcc2	RW	fcc0 参照。
33:32	fcc1	RW	fcc0 参照。
31:30	rd	RW	浮動小数点演算の丸め方法を指定する。詳細は 24 ページの説明を参照。
27:23	tem	RW	IEEE 754 で定義された例外を検出したとき、トラップを発生させるかどうかを制御する。詳細は 27 ページの説明を参照。
22	ns	RW	IEEE 754 に準拠しない演算結果を生成することを許すかどうかを指定する。詳細は 24 ページの説明を参照
19:17	ver	RO	浮動小数点演算ユニットのバージョンを表示する。
16:14	ftt	RO	浮動小数点演算例外によるトラップが発生した際、その詳細情報を表示する。詳細は 25 ページの説明を参照。
13	qne	R0	常に 0。

11:10	fcc0	RW	浮動小数点の比較結果を表示する。24 ページの説明を参照。
9:5	aexc	RW	IEEE 754 で定義された例外を検出したが、トラップがマスクされている場合、発生した例外を記録・蓄積する。詳細は 27 ページの説明を参照。
4:0	cexc	RW	一番最近実行された浮動小数点演算で IEEE 754 で定義された例外を検出したとき、その要因を表示する。トラップがマスクされているかどうかに関わらず更新される。詳細は 27 ページの説明を参照。

fcc*n*

SPARC64™ XIfx は浮動小数点数の比較結果を保持するフィールドを 4 つ持っている。これらのフィールドは、以下の浮動小数点比較命令で更新される。

- FCMP 命令および FCMPE 命令
- FLCMP 命令および FPCMP{64X|U64X}命令

また、これらのフィールドは、FSR レジスタをメモリの値で更新する以下の命令でも更新される。ただし、LDFSR 命令で更新できるのは fcc0 のみである。

- LDFSR および LDXFSR 命令
- LDXEFSR 命令

fcc*n* は同じ精度の浮動小数点数の大小関係を 2 ビットの値で表わす。表 5-7 にその関係を示す。fcc*n* = 3 は数値の大小関係が疑問符になっているが、これは大小関係がないことを意味する。F[rs1], F[rs2]のいずれかが NaN の場合にこの結果になる。

表 5-7　fcc*n*

値	浮動小数点数の大小関係
0	F[rs1] = F[rs2]
1	F[rs1] < F[rs2]
2	F[rs1] > F[rs2]
3	F[rs1] ? F[rs2] (unordered)

rd

rd は浮動小数点演算における丸め方法を指示するフィールドである。丸め方法は表 7-15 に示す 4 種類である。GSR.im = 1 のときは、rd ではなく GSR.irnd で指定した丸め方法が使用される。詳細は General Status Register (GSR) (ASR 19)(34 ページ) 参照。

表 5-8　浮動小数点演算の丸め方法

rd	丸め方法
0	近いほう、等しいときは偶数 (Nearest, Ties to Even)
1	0 方向 (Round toward 0)
2	+∞方向 (Round toward +∞)
3	-∞方向 (Round toward -∞)

ns

ns は浮動小数点演算を IEEE 754 仕様に準拠させるかどうかを指定するフィールドである。XASR.fed = 0 かつ ns = 0 のときは、すべての演算結果および例外生成は IEEE 754 仕様に準拠

する。XASR.fed = 1 もしくは ns = 1 のときは、入力や出力が非正規化数になるケースにおいて、トラップを発生させる代わりに非正規化数を同符号の 0 で置き換える。詳細は SPARC64™ XIIIfx Extensions 参照。

Compatibility Note XASR.fed は SPARC64™ XIfx で追加された。XASR.fed = 1 のとき、ns にどのような値が設定されても、ns = 1 相当の動作となる。

XASR.fed = 0 のときの ftt

SPARC64™ XIfx では浮動小数点演算例外無効モードが追加された。このモードが設定されないとき、すなわち、XASR.fed = 0 のとき、SPARC64™ VIIIfx, SPARC64™ IXfx と同様に浮動小数点演算例外が発生する。

浮動小数点演算例外によるトラップが発生した際、ftt にトラップの詳細情報が表示される。ftt の情報は、ST{X}FSR 命令かまたはトラップを発生させない浮動小数点演算を実行すると 0 でクリアされる。

このフィールドは読み出しのみ可能である。LD{X}FSR 命令では更新されない。

浮動小数点演算のトラップを処理するソフトウェアは、ST{X}FSR 命令を使ってトラップの種類を特定する。ST{X}FSR 命令は、エラーを起こさず終了した場合、ftt を 0 でクリアする。ST{X}FSR 命令がストアでエラーを起こした場合は ftt の値は変更されない。

Programming Note トラップを起こさない浮動小数点演算（“fmovs %f0, %f0”など）を実行することで ftt をクリアすることができる。ftt は次の浮動小数点演算が実行されるまではゼロのままである。

ftt には *fp_exception_other* または *fp_exception_ieee_754* 例外の原因となった理由の一つが表示される。複数の例外要因が同時に発生することもありうるが、そのとき ftt には一番優先度の高い要因が表示される。表 5-9 に、ftt の値の意味と優先度の一覧を示す。

Compatibility Note ftt = 4 および 5 は SPARC V9 および JPS1 で定義されていたが SPARC64™ VIIIfx 及び SPARC64™ IXfx では使われておらず、SPARC64™ XIfx でも使われない。ftt = 3 (*unimplemented_FPop*) は SPARC V9 および JPS1 で定義され、実際に発生していたが、SPARC64™ XIfx では起こらない。

表 5-9 FSR.ftt の値と優先度

浮動小数点演算実行中に検出した例外要因	優先度 (1 が高い)	FSR.ftt の値	発生するトラップ
浮動小数点レジスタの指定が命令定義に違反している	20	6 (*invalid_fp_register*)	*fp_exception_other*
浮動小数点演算が未完了	30	2 (*unfinished_FPop*)	*fp_exception_other*
IEEE 754 仕様で定義された例外要因を検出した	40	1 (*IEEE_754_exception*)	*fp_exception_ieee_754*
reserved	—	3, 4, 5, 7	—
なにも検出しなかった	—	0	—

IEEE_754_exception と *unfinished_FPop* は通常の命令実行の結果起こることがたびたびあるので、システムソフトウェアによって復旧可能でなければならない。

浮動小数点演算のトラップが発生した際、ユーザソフトウェアが観測できる事象は以下の通りである。

- FSR.aexc の値は変化しない。
- *fp_exception_ieee_754* トラップが起きた場合、
  - そのトラップを発生させたのが Non-SIMD 演算かつ単精度倍幅演算命令でないなら、FSR.cexc のどれかひとつのフィールドだけに 1 がセットされている。
  - そのトラップを発生させたのが 2-wide SIMD 演算かつ単精度倍幅演算命令でない、もしくは Non-SIMD 演算かつ単精度倍幅演算命令なら、FSR.cexc の 1 つまたは 2 つのフィールドに 1 がセットされている。
  - そのトラップを発生させたのが 4-wide SIMD 演算(単精度倍幅演算も含む)、もしくは 2-wide SIMD 演算かつ単精度倍幅演算命令なら、FSR.cexc の 1〜4 つのフィールドに 1 がセットされている。

    Comment　4-wide SIMD 演算かつ単精度倍幅演算命令の場合に 1〜5 つのフィールドに 1 がセットされないのは、フィールドのうちひとつが 0 除算のフィールドであることに起因する。現在単精度倍幅演算には除算命令や逆数近似命令がなく、このフィールドはセットされない。
- *fp_exception_ieee_754* 以外のトラップでは FSR.cexc は変化しない。
- 入力レジスタ、出力レジスタは変化しない。
- FSR.fcc*n* の値は変化しない。

ここまでの ftt に関する説明は、IEEE 例外を処理するユーザトラップハンドラから観測できる結果について記述してきた。ユーザトラップハンドラに制御が移されるのは、浮動小数点演算により直接 *fp_exception_ieee_754* 例外が発生したときや、*unfinished_FPop* がソフトウェアによって処理され間接的に例外条件を検出したときなどがある。どちらの場合でも、FSR.cexc にはトラップを起こした要因が表示される。

fp_exception_other 例外が *unfinished_FPop* で起こり、その後のソフトウェアによる処理で IEEE 例外を生じなかった場合、ソフトウェアは FSR.cexc, FSR.aexc, FSR.fcc*n* および出力レジスタを適切に設定しなくてはならない。

- ftt = 1 (*IEEE_754_exception*) — IEEE 754-1985 仕様に準拠した浮動小数点演算例外が起きたとき、この値がセットされる。IEEE 754 例外の種類（オーバーフロー、不正確、など）が FSR.cexc にセットされる。FSR.aexc, FSR.fcc*n* および出力レジスタは変化しない。
- ftt = 2 (*unfinished_FPop*) — 浮動小数点演算器が演算を中断したときや、IEEE 754-1985 仕様に準拠した浮動小数点演算例外が起きたとき、この値がセットされる。例外がおきた場合、FSR.cexc は更新されない。
- ftt = 6 (*invalid_fp_register*) — 浮動小数点演算命令に指示された入出力レジスタのどれかが、レジスタアラインメント条件に違反している場合に、この値がセットされる。4 倍精度命令で 4 の倍数でない浮動小数点レジスタを指定した場合がこれに該当する。

  SPARC64™ XIfx では CPU がこの値をセットすることはない。

SPARC64™ XIfx では上記以外の値は *reserved* である。

XASR.fed = 1 のときの ftt

XASR.fed = 1 のとき、*fp_exception_ieee_754* 及び、*fp_exception_other (unfinished_FPop)*の浮動小数点演算例外は発生しない。また、*fp_exception_other(invalid_fp_register)*トラップは発生しないため、実質、すべての浮動小数点例外が発生しない。

このとき、FSR.tem 及び、FSR.ns に設定された値はすべて無視され、どのような影響も及ぼさない。

Programming Note　SIAM 命令を実行した場合（GSR.im = 1 を設定した場合)、FSR.ns = 0 相当の動作をする。XASR.fed = 1 の場合は、SIAM 命令での設定も無視され、FSR.ns = 1 相当の動作となる。

FSR.tem = 0_0000 及び、FSR.ns = 1 を設定した場合も、*fp_exception_ieee_754* 及び、*fp_exception_other (unfinished_FPop)*のトラップは発生しないが、以下の点で動作が異なる。

- FSR.aexc は更新されない
- FSR.cexc は FSR が更新される命令が実行された場合 0 クリアされる

予期せぬ浮動小数点例外が発生した場合、演算結果以外にそれを外部から観測する術はない。

> **Programming Note**　XASR.fed = 1 のとき、すべての浮動小数点例外を無視する代わり、演算の高速化が期待できる。

tem, aexc, cexc

tem

nvm	ofm	ufm	dzm	nxm
27	26	25	24	23

aexc

nva	ofa	ufa	dza	nxa
9	8	7	6	5

cexc

nvc	ofc	ufc	dzc	nxc
4	3	2	1	0

これら 3 つのフィールドは、IEEE 754 で定義された 5 つの例外について、例外発生の状況を表示したり、例外によるトラップ発生を制御するためのフィールドである。どのフィールドとも 5 ビットからなり、各ビットがそれぞれ IEEE 754 で定義された例外に対応している。ビットの並びかたは 3 フィールドとも同じである。表 5-11 にフィールドの各ビットの意味を示す。

- tem は、浮動小数点演算で起きた IEEE 754 例外により、トラップを発生させるかどうかを制御するために使われる。1 をセットすると、対応する例外によるトラップが発生し、0 をセットするとトラップ発生は抑止される。
- aexc は、浮動小数点演算で起きた IEEE 754 例外の情報を蓄積（accumulate）するためのフィールドである。浮動小数点演算において IEEE 754 で定義された例外が起きたが、その例外でのトラップ発生が抑止されている場合、起こった例外が記録される。
- cexc は、一番最近実行した浮動小数点演算で起きた IEEE 754 例外の情報を表示するためのフィールドである。発生した例外に対応するフィールドには 1 がセットされ、発生しなかった例外に対応するフィールドには 0 がセットされる。

> **Programming Note**　浮動小数点演算によりトラップが発生し、ソフトウェアがエミュレーションした場合、元のソフトウェアに戻る前に cexc を正しく設定する必要がある。

浮動小数点演算により発生する事象と、ftt, aexc, cexc それぞれのフィールドがどのように更新されるかの関係を表 5-10 にまとめた。

表 5-10　浮動小数点演算例外と FSR 更新

事象	ftt	cexc	aexc
例外が発生せず実行終了	0	0 クリアされる	変更されない
IEEE 754 で定義された例外が発生したが、トラップがマスクされていた	0	例外に対応するビットに 1 がセットされる。 検出された例外すべてに対応するビットがセットされる。	更新後の cexc と aexc を **or** した結果で更新される。
IEEE 754 で定義された例外が発生し、トラップも発生した	1	例外に対応するビットに 1 がセットされる。 Non-SIMD 演算では、優先順位によりどれか 1 つのフィールドに 1 がセットされる。 SIMD 演算では、2 つのフィールドに 1 がセットされることがある。	変更されない
fp_exception_other 例外が発生し、トラップした	2[x]	変更されない	変更されない

表 5-11　aexc, cexc の各フィールド

フィールド	例外の意味	仕様書上の表記 トラップが	
		発生する場合	発生しない場合
nva, nvc	無効な演算を行った。 0.0 ÷ 0.0, ∞ – ∞などの演算を行った場合に 1 がセットされる。	NV	nv
ofa, ofc	オーバーフローが起きた。 演算結果を、指数部が無限精度であると仮定して丸めた後の数値が、出力形式の最大値を超えている場合に 1 がセットされる。	OF	of
ufa, ufc	アンダーフローが起きた。 演算結果を丸めた後の数値が、出力形式の最小の正規化数より小さい場合に 1 がセットされる。 アンダーフローは、丸め前の値が正確にゼロになる場合には起こらない。 FSR.tem.ufm = 0 のとき、結果がゼロでない小さな値で、正確さが損なわれた場合にアンダーフローが起きる。 FSR.tem.ufm = 1 のとき、結果がゼロでない小さな値の場合にアンダーフローが起きる。	UF	uf
dza, dzc	ゼロで除算を行った。 ゼロおよび NaN 以外の数をゼロで除算使用とした場合に 1 がセットされる。	DZ	dz
nxa, nxc	不正確な演算結果が出力された。 演算結果を丸めた後の値が、無限精度を仮定したときの演算結果と異なる場合に 1 がセットされる。	NX	nx

IEEE 754 で定義された 5 つの例外のうち、不正確な演算（nx）は、オーバーフロー（of）およびアンダーフロー（uf）と同時に発生することがある。一方 cexc は、非 SIMD 演算の場合は、トラップが発生する場合はどれか 1 つ、発生しない場合は全例外の情報が表示される。表 5-12 に非 SIMD 演算における of, uf, nx 例外発生と、トラップマスク、および cexc に表示される情報の関係をまとめた。

[x] CPU ハードウェアが 6 (*invalid_fp_register*) を設定することはない。

表 5-12　非 SIMD 演算における例外とトラップマスク、および FSR.cexc の関係

条件						結果			
浮動小数点演算で検出した例外			FSR.tem 設定			トラップ発生するか?	FSR.cexc の値		
of	uf	nx	ofm	ufm	nxm		ofc	ufc	nxc
—	—	—	x	x	x	発生しない	0	0	0
—	—	✓	x	x	0	発生しない	0	0	1
—	✓[xi]	✓[xi]	x	0	0	発生しない	0	1	1
✓[xii]	—	✓[xii]	0	x	0	発生しない	1	0	1
—	—	✓	x	x	1	発生する	0	0	1
—	✓[xi]	✓[xi]	x	0	1	発生する	0	0	1
—	✓	—	x	1	x	発生する	0	1	0
—	✓	✓	x	1	x	発生する	0	1	0
✓[xii]	—	✓[xii]	1	x	x	発生する	1	0	0
✓[xii]	—	✓[xii]	0	x	1	発生する	0	0	1

HPC-ACE 互換 (2-wide SIMD)演算における cexc, aexc 更新

SIMD 演算では、トラップが発生する場合でも cexc に 2 つの要因が表示される場合がある。

SIMD 演算では Element-0, Element-1 両方の演算が同時に行われる。入力データは Element-0, Element-1 で異なるので、片方だけで例外が検出されることも、両方で検出されることもある。

Element-0, Element-1 どちらか一方で例外が検出された場合の動作は、非 SIMD 演算の場合と同じである。両方で例外が検出された場合、aexc, cexc の更新、トラップ発生、および ftt 更新は以下のようになる。なお、以下では説明のために、Element-0 側で起きた例外に関する情報を表示する element_0.cexc、Element-1 側で起きた例外に関する情報を表示する element_1.cexc という仮想的なレジスタを使用する。

- Element-0, Element-1 両方で IEEE 754 例外が検出された場合
  - 両方ともトラップが抑止されている場合

    cexc には element_0.cexc と element_1.cexc の論理和が表示され、aexc には element_0.cexc と element_1.cexc の論理和が反映される。

    cexc ← element_0.cexc | element_1.cexc

    aexc ← aexc | element_0.cexc | element_1.cexc
  - Element-0, Element-1 のどちらかのトラップが抑止されていない場合

    cexc には element_0.cexc と element_1.cexc の論理和が表示される。aexc は更新されない。

    cexc ← element_0.cexc | element_1.cexc
  - Element-0, Element-1 両方ともトラップ可能な場合

    cexc には element_0.cexc と element_1.cexc の論理和が表示される。aexc は更新されない。

    cexc ← element_0.cexc | element_1.cexc
- Element-0, Element-1 のどちらか一方で IEEE 754 例外が、他方で *fp_exception_other* 例外が検出された場合

  fp_exception_other 例外によるトラップが発生する。aexc, cexc は更新されない。

[xi] FRCPA{s|d}を除きアンダーフローによるトラップがマスクされている場合（FSR.tem.ufm = 0)、uf は必ず nx を伴う。FRCPA{s|d}はアンダーフローによるトラップがマスクされている場合も（FSR.tem.ufm = 0)、nx を伴わない。

[xii] of は必ず nx を伴う。

Programming Note *fp_exception_other* 例外によるトラップが発生した場合、同時に IEEE 754 例外がおきたかどうかを判断する方法はないので、ソフトウェアのエミュレーションルーチンは IEEE に準拠した例外の検出と、関連するレジスタの更新を行う必要がある。

- Element-0, Element-1 両方で *fp_exception_other* 例外が検出された場合

  fp_exception_other 例外によるトラップが発生する。aexc, cexc は更新されない。

HPC-ACE2 (4-wide SIMD)演算における cexc, aexc 更新

HPC-ACE2 4-wide SIMD 演算では、トラップが発生する場合でも cexc に 4 つの要因が表示される場合がある。

SIMD 演算では element-0 から element-3 までの 4 演算が同時に行われる。入力データは要素ごとで異なるので、1 要素だけで例外が検出される場合も、全要素で例外が検出される場合もある。

演算の要素内のどこか一要素で例外が検出された場合の動作は、非 SIMD 演算の場合と同じである。複数の要素で例外が検出された場合、aexc, cexc の更新、トラップ発生、および ftt 更新は以下のようになる。なお、以下では説明のために、element-0 で起きた例外に関する情報を表示する element_0.cexc、element-1 で起きた例外に関する情報を表示する element_1.cexc、同様に、element_2.cexc、element_3.cexc、という仮想的なレジスタを使用する。

- すべての要素で IEEE 754 例外が検出された場合
  - すべてのトラップが抑止されている場合

    cexc には全要素の論理和が表示され、aexc にも全要素の論理和が反映される。

    cexc ← element_0.cexc | element_1.cexc |
    element_2.cexc | element_3.cexc

    aexc ← aexc | element_0.cexc | element_1.cexc |
    element_2.cexc | element_3.cexc
  - すべての要素のなかでどれかひとつ以上のトラップが抑止されていない場合

    cexc には全要素の論理和が表示される。aexc は更新されない。

    cexc ← element_0.cexc | element_1.cexc |
    element_2.cexc | element_3.cexc
  - すべての要素がトラップ可能な場合

    cexc には全要素の論理和が表示される。aexc は更新されない。

    cexc ← element_0.cexc | element_1.cexc |
    element_2.cexc | element_3.cexc
- どこかの 1 要素以上で IEEE 754 例外が、ほかの 1 要素以上で *fp_exception_other* 例外が検出された場合

  fp_exception_other 例外によるトラップが発生する。aexc, cexc は更新されない。
- 全要素で *fp_exception_other* 例外が検出された場合

  fp_exception_other 例外によるトラップが発生する。aexc, cexc は更新されない。

XASR.fed = 1 のときの tem, aexc, cexc

XASR.fed = 1 のとき、tem の値は無視され、すべての浮動小数点演算例外を検出しない。このとき、tem の値を変更した場合も動作にはまったく影響を及ぼさない。

このとき、ユーザが観測できる状況は以下の通りである。

- aexc は更新されない
- cexc は FSR が更新される命令が実行された場合 0 クリアされる

FSR Conformance

SPARC V9 では FSR.tem, FSR.aexc, FSR.cexc のハードウェアでの実装方法を二種類許容している（どちらも IEEE Std 754-1985 に準拠している)。SPARC64™ XIfx は(1)、すなわちこれら 3 つを全部実装している。

5.5. Ancillary State Registers

SPARC64™ XIfx で定義されている Ancillary State Registers を表 5-13 にまとめる。チェックがついているものはそのモードで使用可能であることを示している。条件付きで使用可能なものは、注釈を付記している。また、使用不可能なものは、各モードで WRASR, RDASR 命令で検出される例外を記載している。

表 5-13　SPARC64™ XIfx で実装される Ancillary State Register 一覧

ASR number	ASR Name	Access	user	Note
0	Y	RW	✓	使用は推奨されない
2	CCR	RW	✓	
3	ASI	RW	✓	
4	TICK	R	✓1)	
5	PC	R	✓	
6	FPRS	RW	✓	
15	opcode only			
16	PCR	RW	✓2), 3)	
17	PIC	RW	✓2)	
19	GSR	RW	✓4)	
24	STICK	R	✓5)	
		W	*illegal_instruction*	
29	XAR	W	✓	
30	XASR	RW	✓	

1) TICK.npt = 0 のとき、*privileged_action*
2) PCR.priv = 1 のとき、*privileged_action*
3) PCR.priv = 0 のとき、PCR.priv = 1 に設定しようとした場合 *privileged_action*
4) PSTATE.pef = 0 または FPRS.fef = 0 のとき、*fp_disabled*
5) STICK.npt = 0 のとき、*privileged_action*

Compatibility Note　UA2011 で定義されている CFR(ASR 26)及び PAUSE(ASR 27)は SPARC64™ XIfx において実装されない

5.5.1. 32-bit Multiply/Divide Register (Y) (ASR 0)

UA2011 仕様 5.5.1 参照。

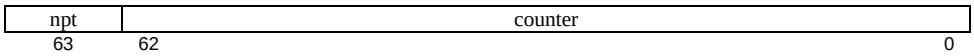
5.5.2. Integer Condition Codes Register (CCR) (ASR 2)

UA2011 仕様 5.5.2 参照。

5.5.3. Address Space Identifier (ASI) Register (ASR 3)

UA2011 仕様 5.5.3 参照。

5.5.4. Tick (TICK) Register (ASR 4)



TICK は、CPU のクロックを計測する counter と、非特権モードのアクセスを制御する npt からなる。SPARC64™ XIfx は counter を 63 ビット幅で実装している（SPARC V9 Impl. Dep. #105b)。

非特権モードでは、npt の値により読み出せるかどうかが決まる。npt = 0 ならば読み出し可能で、npt = 1 なら *privileged_action* 例外が検出される。

Compatibility Note UA2011 では npt は存在していない。SPARC64™ XIfx では非特権モードにおける動作が異なる点に注意が必要である。

TICK への書き込み後読み出しをすると、書き込んだ値より大きな値が読み出される。書き込みの値と読み出しの値の差は、書き込みと読み出しそれぞれの実行時間の差を正確には表してはいない。ソフトウェアは、2 回の読み出しを行いその差分を取ることで正確な時間とすることができる。

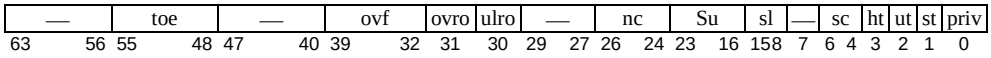
5.5.5. Program Counters (PC, NPC) (ASR 5)

UA2011 仕様 5.5.5 参照。

5.5.6. Floating-Point Registers State (FPRS) Register (ASR 6)

UA2011 仕様 5.5.6 参照。

5.5.7. Performance Control Register (PCR) (ASR 16)



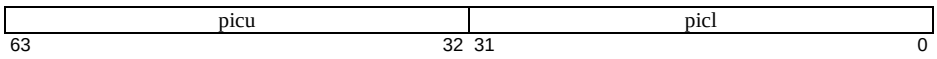
ビット	フィールド	アクセス	説明
55:48	toe	RW	イベントによる例外発生を制御する。書き込みで設定を更新し、読み出しでは現在の設定が返される。toe<i>が 1 のとき、ovf<i>に対応するカウンタがオーバーフローすると、ovf<i> = 1 となり、*pic_overflow* 例外が通知される。toe<i>が 0 のとき、ovf<i>に対応するカウンタがオーバーフローすると、ovf<i> = 1 と

			なるが、*pic_overflow* 例外は通知されない。ovf<i> = 1 のとき toe<i>を 0 から 1 に変更しても *pic_overflow* 例外は通知されない。
39:32	ovf	RW	カウンタのオーバーフロー情報。ovf のビットが 1 であれば対応するカウンタがオーバーフローしたことを示す。ovf<2n>は sc で選択するカウンタペア n の下位カウンタ picl、ovf<2n+1>はカウンタペア n の上位カウンタ picu に対応する。ovf のビットに 0 を書き込むことでオーバーフロー状態をクリアすることができる。ソフトウェアが ovf のビットに 1 を書き込んでも *pic_overflow* 例外は通知されない。
31	ovro	RW	オーバーフロー情報の変更禁止。書き込みの際、書き込もうとするデータの ovro が 0 だと、データの ovf で ovf が更新される。データの ovro が 1 だと、データの ovf は無視され、ovf は更新されない。読み出しには 0 が返る。ovro はオーバーフロー情報を変更することなく PCR を更新するためのものである。ovf はハードウェアが常に最新状態を保つので、次の PCR 読み出し時にはその時点でのオーバーフロー情報が返される。
30	ulro	RW	su, sl の変更禁止。書き込みの際、書き込もうとするデータの ulro が 0 だと、データの su, sl で su, sl が更新される。データの ulro が 1 だと、データの su, sl は無視され、su, sl は更新されない。読み出しには 0 が返る。
26:24	nc	RO	カウンタペアの数。SPARC64™ XIfx では 3(カウンタペア数 4)が読み出される。書き込みは無視される。
23:16	su	RW	PIC.picu で計測するイベントを指定する。書き込みで設定を更新し、読み出しでは現在の設定が返される。
15:8	sl	RW	PIC.picl で計測するイベントを指定する。書き込みで設定を更新し、読み出しでは現在の設定が返される。
6:4	sc	RW	PIC にマッピングするカウンタペアの指定。書き込みでは PIC のマッピングを更新し、読み出しでは現在のマッピング情報が返される。
3	ht	RO (priv) RO (user)	ht = 1 のとき、ハイパーバイザモードで発生したイベントを計測する。ht = 0 のときは計測しない。 読み出しのみ可能である。
2	ut	RW	ut = 1 のとき、非特権モードで発生したイベントを計測する。ut = 0 のときは計測しない。
1	st	RW	st = 1 のとき、特権モードで発生したイベントを計測する。st = 0 のときは計測しない。
0	priv	RW	priv = 1 のとき、非特権モードで PCR, PIC を読み書きすると *privileged_action* 例外が通知される。 priv = 0 のとき、非特権モードで PCR, PIC を読み書きできる。ただし、非特権モードで priv を 1 に変更しようとすると *privileged_action* 例外が通知される。

PCR は 4 組実装されている Performance counter の制御を行うレジスタである。Performance counter のカウンタは PIC(ASR 17)から読み出すことが可能である。PCR, PIC の詳細については、11.Performance Instrumentation(ページ 327) 及び各ビットフィールドの説明を参照のこと。

Compatibility Note su, sl は JPS1 の 6 ビットに対し、SPARC64™ VIIIfx / SPARC64™ IXfx で 7 ビット、SPARC64™ XIfx で 8 ビットに拡張した。SPARC64™ VIIIfx / SPARC64™ IXfx と SPARC64™ XIfx ではレジスタのビット割り当てが異なっていることに注意が必要である。

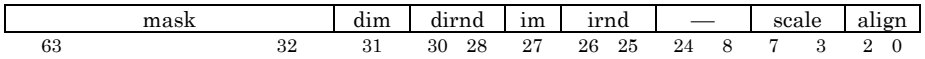
5.5.8. Performance Instrumentation Counter (PIC) Register (ASR 17)



ビット	フィールド	アクセス	説明
63:32	picu	RW	32 ビットカウンタ。PCR.sc で選択されたカウンタの組の upper 側が表示される。
31:0	picl	RW	32 ビットカウンタ。PCR.sc で選択されたカウンタの組の lower 側が表示される。

PIC は PCR と組み合わせて使用することで、PCR で設定された Performance counter の値が表示される。

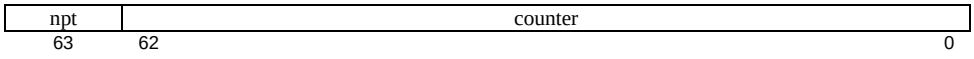
5.5.9. General Status Register (GSR) (ASR 19)



ビット	フィールド	アクセス	説明
63:32	mask	RW	UA2011 参照。
31	dim	R	常に 0。
30:28	dirnd	R	常に 0。
27	Im	RW	UA2011 参照。
26:25	Irnd	RW	UA2011 参照。
7:3	scale	RW	UA2011 参照。
2:0	align	RW	UA2011 参照。

GSR.dim, GSR.drd は常に 0 である。それ以外のフィールドについては UA2011 5.5.9 参照。

5.5.10. System Tick (STICK) Register (ASR 24)



STICK は、CPU 外部から一定間隔で送られてくる信号を計測する counter と、非特権モードのアクセスを制御する npt からなる。外部信号のクロックはこの仕様書では定義されない。SPARC64™ XIfx は counter を 63 ビット幅で実装している。

非特権モードでは、npt の値により読み出せるかどうかが決まる。npt = 0 ならば読み出し可能で、npt = 1 なら *privileged_action* 例外が検出される。表 5-14 に STICK 読み出し、書き込みと例外の関係をまとめた。

表 5-14　STICK の読み出し、書き込みと例外

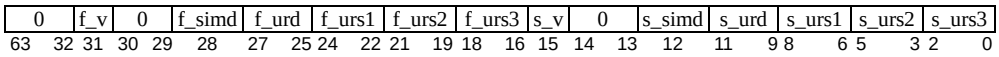
	RDSTICK	WRSTICK
user	STICK.npt = 0 のとき OK それ以外は *privileged_action*	*illegal_instruction* (TICK と異なる)

Compatibility Note　JPS1 では、非特権モードで書き込もうとしたときには *privileged_opcode* 例外を検出していた。

Compatibility Note　UA2011 では npt は存在していない。SPARC64™ XIfx では非特権モードにおける動作が異なる点に注意が必要である。

Compatibility Note　UA2011 では counter の下位 m ビットに ALL0 が返ることが許容されているが、SPARC64™ XIfx では全ビット有効である。

5.5.11.　Extended Arithmetic Register (XAR) (ASR 29)



ビット	フィールド	アクセス	説明
31	f_v	RW	f_ で始まるフィールドの内容が有効かどうかを示す。f_v = 1 なら、1 番目に実行される命令に f_ が適用される。1 番目の命令が完了すると、f_ フィールドはすべて 0 クリアされる。
28	f_simd	RW	f_simd = 1 なら、1 番目に実行される命令は SIMD 命令として実行される。f_simd = 0 なら Non-SIMD で実行される。SIMD 命令の実行モードは XASR.simd_mode により制御される。
27:25	f_urd	RW	1 番目の命令の rd の拡張フィールド。
24:22	f_urs1	RW	1 番目の命令の rs1 の拡張フィールド。
21:19	f_urs2	RW	1 番目の命令の rs2 の拡張フィールド。
18:16	f_urs3	RW	1 番目の命令の rs3 の拡張フィールド。
15	s_v	RW	s_ で始まるフィールドの内容が有効かどうかを示す。s_v = 1 なら、2 番目に実行される命令に s_ が適用される。2 番目の命令が完了すると、s_ フィールドはすべて 0 クリアされる。
12	s_simd	RW	s_simd = 1 なら、2 番目に実行される命令は SIMD 命令として実行される。s_simd = 0 なら Non-SIMD で実行される。SIMD 命令の実行モードは XASR.simd_mode により制御される。
11:9	s_urd	RW	2 番目の命令の rd の拡張フィールド。
8:6	s_urs1	RW	2 番目の命令の rs1 の拡張フィールド。
5:3	s_urs2	RW	2 番目の命令の rs2 の拡張フィールド。
2:0	s_urs3	RW	2 番目の命令の rs3 の拡張フィールド。

XAR は命令フィールドを拡張するためのレジスタである。命令のレジスタ指定フィールド（rs1, rs2, rs3, rd）の上位 3 ビットと、SIMD 演算するかどうかを指定する。

XAR は 2 命令分のレジスタ指示フィールドを持つ。1 命令目、2 命令目用の各フィールドには V (valid) ビットがあり、その他のフィールドは v = 1 のときに有効である。レジスタフィールドには整数/浮動小数点数の区別はなく、任意のレジスタと結合する。

トラップ時には、XAR の内容は TXAR[TL]にセーブされ、XAR の全フィールドに 0 が設定される。セーブされるのは、実行中の命令の実行直前の XAR の値である。

Note Tcc 命令の条件が成立してトラップした時も、Tcc 実行直前の XAR の内容がセーブされる。

Compatibility Note SIMD 命令として実行される場合、XASR.simd_mode により SIMD 命令の実行モードが選択される。XASR.simd_mode = 0 として実行した場合、SPARC64™ VIIIfx, SPARC64™ IXfx と互換の動作となる。

本仕様書における XAR の記述について

上で示した XAR のフィールド名のほかに、以下の別名表記も用いる。

表 5-15 XAR 別名表記

別名	フィールド名	用途
XAR.f_dis_hw_pf	XAR.f_urs3<1>	ハードウェアプリフェッチ抑止指定
XAR.s_dis_hw_pf	XAR.s_urs3<1>	ハードウェアプリフェッチ抑止指定
XAR.f_sector	XAR.f_urs3<0>	セクタキャッシュ番号指定
XAR.s_sector	XAR.s_urs3<0>	セクタキャッシュ番号指定
XAR.f_negate_mul	XAR.f_urd<2>	SIMD FMA 用
XAR.s_negate_mul	XAR.s_urd<2>	SIMD FMA 用
XAR.f_rs1_copy	XAR.f_urs3<2>	SIMD FMA 用
XAR.s_rs1_copy	XAR.s_urs3<2>	SIMD FMA 用
XAR.f_ldst_type	XAR.f_urs2<2:1>	FPR への整数ロード指定
XAR.s_ldst_type	XAR.s_urs2<2:1>	FPR への整数ロード指定
XAR.f_ldst_SIMD_op	XAR.f_urd<2>, XAR.f_urs1<2:1>	SIMD LD/ST のオペレーション指定
XAR.s_ldst_SIMD_op	XAR.s_urd<2>, XAR.s_urs1<2:1>	SIMD LD/ST のオペレーション指定
XAR.f_right_justify	XAR.f_urs1<0>	Convert 命令用
XAR.s_right_justify	XAR.s_urs1<0>	Convert 命令用
XAR.f_unpartition	XAR.f_urs3<0>	SIMD Compare 用
XAR.s_unpartition	XAR.s_urs3<0>	SIMD Compare 用
XAR.f_sector_h	XAR.f_urs3<2>	セクタキャッシュ番号指定
XAR.s_sector_h	XAR.s_urs3<2>	セクタキャッシュ番号指定

XAR 動作

XAR を参照する命令と参照しない命令がある。ここでは XAR を参照する命令のことを、"XAR 対象命令" と呼ぶ。どの命令が XAR 対象命令かは、表 7-3 を参照。

- XAR 対象でない命令は、その命令の実行時に XAR.v = 1 だと *illegal_action* 例外が発生する。
- XAR 拡張が必須の命令は、その命令実行時に XAR.v = 0 だと *illegal_action* 例外を発生する。
- XAR 対象命令の動作は、
  - XAR.v = 1 のとき、XAR.urs1, XAR.urs2, XAR.urs3, XAR.urd がそれぞれ命令フィールドの rs1, rs2, rs3, rd と結合する。

    整数レジスタでは、XAR のフィールドを上位 3 ビット、命令フィールドを下位 5 ビットとする計 8 ビットで指定されるレジスタを使用する。ただし、上位 2 ビットは命令によってはレジスタ指定以外の用途に使用される。

    浮動小数点レジスタでは、XAR のフィールドを上位 3 ビットとし、命令フィールドの 5 ビットを倍精度レジスタのエンコーディング方式にしたがってデコードした 6 ビットを下位とする計 9 ビットで指定されるレジスタを使用する。浮動小数点レジスタのエンコーディングの詳細は Floating-Point Registers Number Encoding 5.3.4 を参照。SIMD 拡張された場合上位 1 ビットは一部命令ではレジスタの指定に使用される。また、一部命令以外ではレジスタ指定以外の用途に使用される。ほとんどの命令では *reserved* として扱う。
  - XAR.f_v = 1 なら XAR.f_urs1, XAR.f_urs2, XAR.f_urs3, XAR.f_urd が使われる。
  - XAR.f_v = 0 かつ XAR.s_v = 1 なら XAR.s_urs1, XAR.s_urs2, XAR.s_urs3, XAR.s_urd が使われる。
- XAR の値は 1 回のみ有効。XAR を参照した命令が完了すると、XAR の関連フィールドにはすべて 0 が設定される。
- XAR 対象命令であっても、以下のどれかにあたる場合 *illegal_action* 例外が発生する。
  - XAR のフィールドに特殊機能が割り当てられていない場合かつ、以下が成り立つ場合
    - xg[32]以上の整数レジスタを指定した場合。
    - rs1 を使わない命令に対して、urs1 ≠ 0 を指定した場合。rs2, rs3, rd についても同様である。
    - rs2 フィールドを simm13 や fcn などの即値として使う命令で、urs2 ≠ 0 の場合も *illegal_action* 例外が発生する。
  - XAR のフィールドに特殊なモードビットが割り当てられている場合は各命令での発生条件参照
  - `FDIV{S|D}`, `FSQRT{S|D}`, `FEPERMD`, `FESUMMD`, `FECPD` で rd に f[256]以上を指定した場合。
  - SIMD 拡張されない命令（整数演算も含む）に対して XAR.simd = 1 を指示した場合。
  - XAR.simd = 1 で f[256]以上を指定した場合。

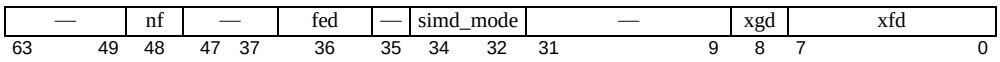
    ただし `F{N}MADD{s|d}`, `F{N}MSUB{s|d}`, `FSHIFTORX`, `FPMADDX`, `FPMADDXHI` など一部の命令は、f[256]以上を指定することができる。f[256]以上を指定したときの動作は個々の命令仕様を参照。

後続 1 命令のレジスタ番号を指示する場合、1st を使っても 2nd を使ってもよい。

> **Programming Note**　`WRXAR` 命令を使えば XAR.f_v, XAR.s_v どちらでも任意に 1 を設定できる。`SXAR1` 命令では XAR.f_v = 1 になる。

XAR.f_v = 0 のときは、f_simd, f_urs1, f_urs2, f_urs3, f_urd の各フィールドに 0 でない値があっても無視される。命令実行後、各フィールドの値がどうなるかは未定義である。XAR.s_v = 0 のときは、s_simd, s_urs1, s_urs2, s_urs3, s_urd の各フィールドに 0 でない値があっても無視される。命令実行後各フィールドの値がどうなるかは未定義である。

5.5.12. Extended Arithmetic Register Status Register (XASR) (ASR 30)



ビット	フィールド	アクセス	説明
48	nf	RW	Non-Faulting モード XASR.nf は SPARC64™ XIfx 独自のフィールドである
36	fed	RW	Floating-Point Exception Disable すべての浮動小数点例外を検出しない SPARC64™ XIfx 独自のフィールドである
34:32	simd_mode	RW	SIMD width 選択 000_2 HPC-ACE 互換 001_2 HPC-ACE2 4-wide SIMD モード 上記以外 *reserved* XASR.simd_mode は SPARC64™ XIfx 独自のフィールドである
8	xgd	RW	拡張整数レジスタを更新したことを示すビット。
7:0	xfd	RW	浮動小数点レジスタを更新したことを示すビット。

XASR<63:49>, XASR<47:37>, XASR<35>, XASR<31:9>は今後の拡張のため、*reserved* に変更された。XASR に値を書き込む場合、これらのフィールドは 0 を書き込む必要がある。

nf

nf は Non-Faulting モードを設定するモードビットであり、WRXASR 命令もしくは、snf 命令で変更される。

本ビットフィールドは TL = 0 のときのみ有効であり、TL > 0 のときは無効である。無効となったとき、どのような値が設定されても nf = 0 と同様に振舞う。

nf はすべてのメモリ読み出し（ロード）の動作を変更する。nf = 1 のとき、すべてのロード命令は指定された ASI に追加して NO_FAULT 属性を持つかのように振舞う。nf = 1 を指定した結果、ASI に存在しないようなアクセスとなる場合も許容される。すでにロードの ASI に NO_FAULT 属性をもつものが指定されている場合、nf により動作は変更されない。

nf = 0 のときには、メモリ読み出し（ロード）の動作は変更されず、命令で指定された暗黙の ASI もしくは明示的に指定された ASI でアクセスを行う。

詳細や対象となる命令については 325 ページ参照。

> **Programming Note** XASR.nf は TL > 0 で無効化される。特権モードかつ TL = 0 で NO_FAULT 属性をもつロード命令が許容されない場合、特権モードかつ TL > 0 で事前に XASR.nf を保存した後、XASR.nf = 0 を設定する必要があるかもしれない。そのような場合、非特権モードに移行する前に、保存した XASR.nf が復元されている必要がある。

fed

fed は浮動小数点演算例外をマスクするモードを設定するフィールドである。WRXASR 命令で変更される。

fed = 1 を指定したとき、すべての浮動小数点演算例外はマスクされ検出されない。このため、対象となる例外によるトラップも発生しない。FSR.tem, FSR.ns に設定された値によらず、

FSR.aexc が更新されず FSR.cexc, FSR.ftt は 0 クリアされるため、浮動小数点例外が発生したことを観測することは困難である。詳細は、FSR の項目(26, 30 ページ)参照。また、FSHIFTORX, FEPERMD 命令で発生する *illegal_instruction* トラップも抑止される。以下に、fed = 1 のときに、動作が変更されるトラップと動作をまとめる。

対象のトラップ	XASR.fed = 0	XASR.fed = 1
fp_exception_ieee	FSR.tem の設定に従う	トラップは抑止される FSR が更新される命令を実行した場合 • FSR.cexc, FSR.ftt は 0 クリアされる • FSR.aexc は更新されない
fp_exception_other (*unfinished_FPop*)	FSR.ns の設定に従う	トラップは抑止される
illegal_instruction (FSHIFTORX)	Fd[rs3]の値により *illegal_instruction* が発生する	トラップは抑止される Fd[rd]に格納される値は不定となる
illegal_instruction (FEPERMD)	Fd[rs2]の値により *illegal_instruction* が発生する	トラップは抑止される Fd[rd]に格納される値は不定となる

fed = 1 を設定したときの演算結果は、FSHIFTORX, FEPERMD 命令を除いて fed = 0, FSR.tem = 0_0000_2かつ FSR.ns = 1 を設定したときと同等である。

Programming Note fed = 1 を設定することで、演算完了を高速化することが期待される。例外検出の必要がない場合は fed = 1 で実行することを推奨する。

simd_mode

XAR.v = 1, XAR.simd=1 で実行される命令は SIMD 拡張されるが、simd_mode はその命令で有効な要素（実行される演算幅）を設定する。WRXASR 命令もしくは、ssm 命令で変更される。

XASR.simd_mode = 0 のとき、HPC-ACE 互換モードとして動作する。このとき、有効な要素は Element-0,1 の 2 要素(128bit)となる。

XASR.simd_mode = 1 のとき、HPC-ACE2 4-wide SIMD モードとして動作する。このとき、有効な要素は Element-0,1,2,3 の 4 要素(256bit)となる。

XASR.simd_mode > 1 の値は将来の拡張に向けて予約されており、XASR.simd_mode > 1 の値を指定した場合 ssm 命令、WRXASR 命令では *illegal_instruction* 例外を検出する。

xfd, xgd

コンテキストスイッチ時に、レジスタを保存する必要があるかどうかを判断するために使用する。拡張レジスタを更新すると、対応するビットが 1 になる。

- V9 定義の整数レジスタの更新を示すフラグはない。
- xg[0] – xg[31]のいずれかのレジスタを更新すると、XASR.xgd に 1 がセットされる。
- 浮動小数点レジスタを更新すると、XASR.xfd の対応するビットに 1 がセットされる。ビット対応は以下の通り。

xfd のビット	対応する浮動小数点レジスタ
0	Element-0 F[0] – F[62]
1	Element-0 F[64] – F[126]
2	Element-0 F[128] – F[190]
3	Element-0 F[192] – F[254]
4	Element-1, Element-2, Element-3 F[0] – F[62] (F[256] – F[318])
5	Element-1, Element-2, Element-3 F[64] – F[126] (F[320] – F[382])
6	Element-1, Element-2, Element-3 F[128] – F[190] (F[384] – F[446])
7	Element-1, Element-2, Element-3 F[192] – F[254] (F[448] – F[510])

Programming Note　XASR.xfd<0>は V9 FPR を更新したときにセットされる。このとき、FPRS も同時に更新される。例えば F[15]を更新した場合は、FPRS.dl = 1 と XASR.xfd<0> = 1 がセットされる。

MOVr, MOVcc, FMOVr, FMOVcc 命令で条件が不成立のとき、対応する XASR のビットが 1 になるかどうかは実装に依存する。

Compatibility Note　HPC-ACE では xfd<4:7>は Extended Floating-Point Registers が更新されたことを示すフラグであった。HPC-ACE2 ではこれを拡張し、Element-1, Element-2, Element-3 Floating-Point Registers が更新されたことを示すフラグとして扱う。

Comment　HPC-ACE 互換モードであっても、HPC-ACE2 モードであっても、xfd<4:7>のビットが立っているなら、Element-1, Element-2, Element-3 すべての該当 Floating-Point Registers を保存しなければならない。

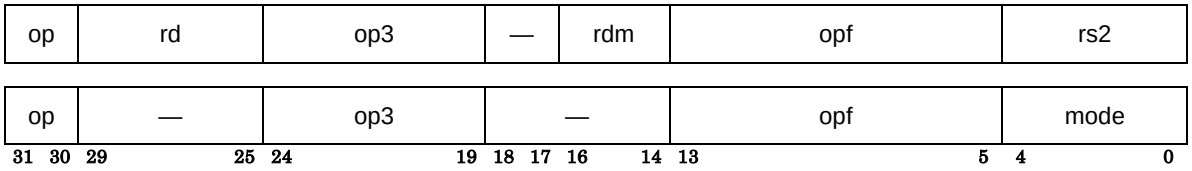
6. Instruction Set Overview

6.1. Instruction Formats and Fields

SPARC64™ XIfx では JPS1 仕様 6.2 に定義フォーマットに加えて、下記の独自フォーマットをサポートした。詳細は各命令の詳細は命令定義を参照のこと。

JPS1 仕様 6.2 に記載されている Format 3 に加えて、以下のフォーマットをサポートしている

Format 3 (op = 2) : RDM, SSM and SNF instructions



ビット	フィールド	説明
16:14	rdm	FRD{s|d}で丸めモードを指示するために使われる。
4:0	mode	浮動小数点演算の種類を指示するために使われるほか、Impdep2 に定義された命令を選択するために使われる。

Format 5 は SPARC64™ VIIIfx で追加された。

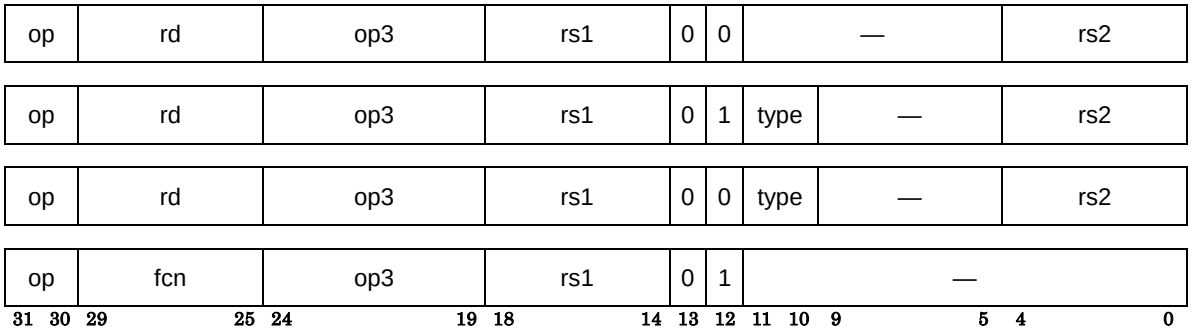
Format 5 (op = 2, op3 =37_{16}) : FMADD, FPMADDX,FSELMOV, FTRIMADD and FSHIFTORX

ビット	フィールド	説明
13:9	rs3	浮動小数点演算の 3 番目の入力浮動小数点レジスタを示す。
8:7	var	浮動小数点演算の種類を指示するために使われるほか、Impdep2 に定義された命令を選択するために使われる。
6:5	size	浮動小数点演算のサイズを指示するために使われるほか、Impdep2 に定義された命令を選択するために使われる。
13:9	index	FTRIMADDd で係数テーブルを指示するために使われる。

Format 6 は SPARC64™ XIfx で追加された HPC-ACE2 拡張フォーマットである。XAR レジスタと組み合わせて使用することで、拡張命令として振舞う。詳細は各命令の詳細は命令定義を参照のこと。

Programming Note この拡張に伴いこれまで、*illegal_instruction* 例外を検出していたプログラムで *illegal_instruction* 例外に代わり、*illegal_action* 例外を検出することがある。

Format 6 (op = 3) : HPC-ACE2 拡張フォーマット



ビット	フィールド	説明
12	id	XAR.v = 1 かつ i = 0 かつ id = 1 の場合、浮動小数点ロードストア及びプリフェッチ命令は Indirect 命令となる
11:10	type	XAR.v = 1 かつ、以下の条件に当てはまる場合、型修飾子の指定フィールドとして機能する • 4 バイトの Store Floating-Point Register on Register Condition 命令かつ i = 0 • 4 バイトの Indirect 命令

6.2. ニーモニックについて

HPC-ACE 及び HPC-ACE2 の拡張表記についてまとめる。

6.2.1. HPC-ACE 拡張のサフィックス

HPC-ACE 拡張は、ニーモニックの後にコンマを置き、その後ろに英数字一文字で指定する。以下に HPC-ACE 拡張命令で使用できるサフィックスを示す。

表 6-1　HPC-ACE 拡張 サフィックス

XAR 表記	サフィックス	備考
XAR.simd	s	
XAR.dis_hw_pf	d	
XAR.sector	1	0 でセクタ 0 指定（デフォルト）
XAR.negate_mul	n	
XAR.rs1_copy	c	

サフィックスは大文字・小文字の区別はなく、複数のサフィックスを指定する場合は任意の順序で指定可能とする。

詳細は、SPARC64™ VIIIfx-Extension G.4 を参照。

6.2.2. HPC-ACE2 拡張のサフィックス

HPC-ACE2 ではセクタキャッシュ機能が拡張された。これにより HPC-ACE 拡張サフィックスに加えて新たに、“2”及び “3” の 2 種のサフィックスが指定可能となった。サフィックス“0” 及び “1”は HPC-ACE 拡張サフィックスと互換である。また、追加されたサフィックスは HPC-ACE 拡張のサフィックスと混在して使用することが可能である。ただし、セクタキャッシュ機能のサフィックス “0”, “1”, “2”, “3”は最大 1 つしか指定できない。

表 6-2　HPC-ACE2 拡張サフィックス

XAR 表記	サフィックス	備考
XAR.sector_h=0, XAR_sector=0	0	HPC-ACE 拡張サフィックスと互換 セクタ 0 指定（デフォルト）
XAR.sector_h=0, XAR_sector=1	1	HPC-ACE 拡張サフィックスと互換 セクタ 1 指定
XAR.sector_h=1, XAR_sector=0	2	セクタ 2 指定
XAR.sector_h=1, XAR_sector=1	3	セクタ 3 指定

6.2.3. HPC-ACE2 拡張表記について

HPC-ACE2 ではメモリアクセスを拡張しており、拡張についてのニーモニック表記の考え方を説明する。7 章の説明ではこれら拡張表記について、使用可能なすべての場合を個別の命令として説明する。

なお、SPARC64™ XIfx でインダイレクトロード命令(LDFID, LDDFID)、インダイレクトストア命令(STFID, STDFID)、マスク付きインダイレクトストア命令(STFRID, STDFRID)が追加されたが、説明では断りなく使用している。

HPC-ACE2 で導入する修飾子を HPC-ACE2 拡張修飾子と呼ぶ。HPC-ACE2 拡張修飾子には、型修飾子とオペレーション修飾子があり、被修飾命令においてそれぞれの修飾子を最大 1 種類ずつ使用可能である。被修飾命令によっては、型修飾子とオペレーション修飾子のどちらかしか使用できない場合もある。

型修飾子とオペレーション修飾子は大文字・小文字の区別はなく、型修飾子とオペレーション修飾子の両方を指定する場合は命令に続けてオペレーション修飾子、型修飾子の順番に記述する。HPC-ACE2 拡張修飾子を使用する際も、HPC-ACE 拡張サフィックスを同時に指定可能である。その際は、[命令][HPC-ACE2 拡張修飾子][HPC-ACE 拡張サフィックス] という順で使用すること。HPC-ACE2 拡張修飾子は HPC-ACE 拡張サフィックスの前に記載する必要がある。

例えば、stride 幅 3 の符号付き整数の sector1 を使った SIMD 浮動小数点ロードは以下のようにあらわす。

```
ldstsw,s1 [%r1+48]@3, %f16
```

マスク付インダイレクトストアの符号無し整数の SIMD 浮動小数点ストアは以下のようにあらわす。

```
stfriduw,s %f220, %f192, [%f12]
```

オペレーション修飾子

オペレーション修飾子とは、浮動小数点ロード命令及び浮動小数点ストア命令が SIMD 拡張されたとき、各要素がアクセスするアドレスの生成方法を選択する。動作モードが 2-wide SIMD であっても 4-wide SIMD であっても指定可能である。

オペレーション修飾子が指定されるかどうかに関わらず、要素 0 のアクセスするアドレスは変更されない。変更されるのは要素 1～3 がアクセスするアドレスである。

オペレーション修飾子を使用しない場合、暗黙の修飾子となり各要素は要素 0 がアクセスするアドレスから連続するアドレスにアクセスする。これは、SPARC64™ VIIIfx, SPARC64™ IXfx, などの動作と同一である。

ブロードキャストオペレーションはロード命令のみに使用できる。すべての要素は要素 0 がアクセスするアドレスにアクセスし、データをロードする。

ストライドオペレーションはロード・ストア命令で使用でき、要素ごとに一定アドレス間隔にアクセスする。要素 0 がアクセスするアドレスから strc で指示される間隔により要素 1～3 がアクセスするアドレスが決定される。この修飾子が指定されたとき、アセンブリ言語表記では、即値形式でアドレス指定の後ろに strc を指定する必要がある。strc には 2～7 が指定可能である。

表 6-3 にオペレーション修飾子の一覧を示す。

表 6-3　HPC-ACE2 オペレーション修飾子

表記/値	修飾子	strc	備考
XAR.ldst_SIMD_op = 000_2	—	—	暗黙の修飾子（連続のオペレーション）
XAR.ldst_SIMD_op = 001_2	bc	—	Broadcast オペレーション
XAR.ldst_SIMD_op = 010_2	st	2	Stride 幅 2 のオペレーション
XAR.ldst_SIMD_op = 011_2	st	3	Stride 幅 3 のオペレーション
XAR.ldst_SIMD_op = 100_2	st	4	Stride 幅 4 のオペレーション
XAR.ldst_SIMD_op = 101_2	st	5	Stride 幅 5 のオペレーション
XAR.ldst_SIMD_op = 110_2	st	6	Stride 幅 6 のオペレーション
XAR.ldst_SIMD_op = 111_2	st	7	Stride 幅 7 のオペレーション

SPARC64™ XIfx で追加されたインダイレクトロード命令、インダイレクトストア命令、マスク付きインダイレクトストア命令はオペレーション修飾子の修飾対象ではない。さらに、ASI 指定ができるロード命令、ストア命令についても修飾対象とはならない。表 6-4 にオペレーション修飾子で修飾可能な命令を示す。表に記載されていない命令にはオペレーション修飾子を付与することはできない。

表 6-4　HPC-ACE2 オペレーション修飾子の被修飾対象

命令	st (strc=2-7)	bc
LDF	✓	✓
LDDF	✓	✓
STF	✓	
STDF	✓	
STFR	✓	
STDFR	✓	

暗黙の修飾子以外のオペレーション修飾子を使用する場合、SXAR1 もしくは SXAR2 による前置が必要である。また、オペレーション修飾子は SIMD 拡張が必要となるため、オペレーション修飾子が指定された場合、暗黙的に HPC-ACE 拡張のサフィックス “s”が指定される。明示的に HPC-ACE 拡張のサフィックス “s”を指定した場合、サフィックス“s”が指定されない場合との違いはない。

Programming Note　オペレーション修飾子を使用する場合、プログラムの可読性の観点からサフィックス“s”を明示的に指定することを推奨する。

型修飾子

型修飾子とは、単精度浮動小数点ロード命令が倍精度レジスタにデータをロードする場合、4 バイトのデータをどういった形式で倍精度レジスタに格納するかを選択する。同様に、単精度浮動小数点ストア命令が倍精度レジスタからデータをストアする場合、倍精度レジスタのどの 4 バイトデータをストアするか選択する。

倍精度浮動小数点ロード命令、倍精度浮動小数点ストア命令は型修飾子の対象とはならない。さらに、ASI 指定ができるロード命令、ストア命令についても型修飾を使用することはできない。

型修飾子を使用しない場合、暗黙の修飾子が使用される。このとき、SXAR によりレジスタ拡張される場合、倍精度レジスタの単精度利用(ページ 21)の動作となる。これは、SPARC64™ VIIIfx, SPARC64™ IXfx などの動作と同一である。

表 6-5 に型修飾子の一覧を示す。

表 6-5　HPC-ACE2 型修飾子

表記/値 [xiii]	修飾子	備考
XAR.ldst_type = 00_2 (IW.type = 00_2)	—[xiv]	暗黙の修飾子(上位 32 ビット)
XAR.ldst_type = 01_2 (IW.type = 01_2)	uw	符号無し整数(下位 32 ビット)
XAR.ldst_type = 10_2 (IW.type = 10_2)	ib	要素内ブロードキャスト
XAR.ldst_type = 11_2 (IW.type = 11_2)	sw	符号付き整数

[xiii] 命令により、XAR.ldst_type で指定する場合と、IW.type を使う場合がある
[xiv] 暗黙の型修飾子であり、型修飾子がない場合この型修飾子となる

型修飾子は 4 種類あり、単精度浮動小数点ロード命令ではすべて使用できる。ただし、SPARC64™ XIfx で追加されたインダイレクトロード命令では単精度倍幅の修飾子は使用できない。ストア命令では暗黙の修飾子及び符号無し整数の 2 種類が使用できる。表 6-6 に型修飾子で修飾可能な命令を示す。表に記載されていない命令に型修飾子を付与することはできない。

暗黙の修飾子以外の型修飾子を使用する場合、SXAR1 もしくは SXAR2 による前置が必要である。

Note 一部命令において、インストラクションワードの定義上、型修飾子を SXAR1, SXAR2 前置なしにインストラクションワード上(IW.type)で表現可能なものがある。これらの命令においては、SXAR1, SXAR2 前置なしに型修飾子フィールドに暗黙の修飾子(IW.type=0)以外を使用した場合、*illegal_action* 例外を検出し正常に動作しない。

表 6-6 HPC-ACE2 型修飾子の被修飾命令

命令	uw	ib	sw
LDF	✓	✓	✓
STF	✓		
STFR	✓		
LDFID	✓		✓
STFID	✓		
STFRID	✓		

7. Instructions

この章では、SPARC64™ XIfx で定義されている命令について記述する。

Compatibility Note JPS1 と UA2011 で仕様が異なる命令が数多くあるが、SPARC64™ XIfx の仕様は UA2011 に準拠する。

SPARC64™ XIfx の命令仕様には、以下の記述が含まれている。

- 命令表。そのオペコードに固有のフィールド値と、HPC-ACE/HPC-ACE2 拡張機能に対応しているかどうかの情報、およびアセンブリ言語での表記法を含む。
- 命令フォーマットの図。図中、ダッシュ(—)で表示されているフィールドは将来の拡張用に予約されており（*reserved*)、SPARC64™ XIfx 用のプログラムでは 0 が入っているべきである。*reserved* の扱いは、Section 1.2.4 “reserved の扱い“ (13 ページ）を参照。
- 命令の詳細、制限事項などの説明。

  Note 例外発生条件は説明文ではなく、各命令説明の最後に表として載せ、簡潔に表わせない条件がある場合のみ本文で説明する。
- 浮動小数点演算命令については、入力オペランドと演算結果および IEEE 754 で定義された演算例外（OF, UF, DZ, NX, NV）の関係を表わした表。
- その命令を実行したときに起こりうる例外の表。例外は優先順位の高いものから順に記載している。

  ただし、以下の例外は記載していない。

  - 全ての命令で起こり得る *IAE_**。
  - 実装されていない命令で起こる *illegal_instruction*。

  なお、*illegal_action* 例外の発生条件の記述では、XAR のフィールド名には f_, s_をつけず表記する。

命令仕様には、タイミングに関する情報は記載していない。

SPARC64™ XIfx の全命令の一覧である表 7-3 及びこの章と第 17 章ではオペコードの右肩に文字列が付されている命令がある。その文字列の意味は下表の通りである。

表 7-1　オペコードの右肩の文字の意味

文字	意味
D	使用を推奨しない命令
N	非互換命令
P_{ASI}	ASI のビット 7 が 0 だと特権動作
P_{ASR}	ASR 番号によっては特権動作
P_{NPT}	PSTATE.priv = 0 かつ {S}TICK.npt = 1 のとき特権動作
P_{PIC}	PCR.priv = 1 のとき特権動作
P_{PCR}	PCR.priv = 1 のとき特権アクセス

命令の詳細説明では、演算に使うデータについて表 7-2 に示す表記を使う。

表 7-2　レジスタ表記の意味（rs2, rs3, rd も同様）

表記	意味	
	XAR.v = 0	XAR.v = 1
R[rs1]	命令語の rs1 で指示される整数レジスタの内容	命令語の rs1 で指示される整数レジスタの内容
Fs[rs1]	命令語の rs1 で指示される単精度浮動小数点レジスタの内容	XAR.urs1 および命令語の rs1 で指示される単精度浮動小数点レジスタの内容
Fd[rs1]	命令語の rs1 で指示される倍精度浮動小数点レジスタの内容	XAR.urs1 および命令語の rs1 で指示される倍精度浮動小数点レジスタの内容
F[rs1]	命令語の rs1 で指示される浮動小数点レジスタの内容 (単精度・倍精度・4 倍精度の区別が不要な場合の表記)	XAR.urs1 および命令語の rs1 で指示される浮動小数点レジスタの内容 (単精度・倍精度・4 倍精度の区別が不要な場合の表記)

本論理仕様において、記載されている命令を表 7-3 にまとめる。各列については、

- XAR.v = 0 に✓がある命令は HPC-ACE(HPC-ACE2)拡張なしで使用可能なことを示す
- XAR.v = 1 Regs.に✓がある命令は拡張レジスタが使用可能な命令を示す
- XAR.v = 1 Regs.に☆がある命令は非 SIMD 動作において、rd レジスタに Element-0 側の FPR のみを使用可能な命令を示す
- XAR.v = 1 2-wide SIMD に✓がある命令は 2 要素(128bit)幅 SIMD 拡張可能な命令を示す
- XAR.v = 1 4-wide SIMD に✓がある命令は 4 要素(256bit 幅)SIMD 拡張可能な命令を示す

であることを示す。

基本的に SIMD 拡張可能な命令は 2/4-wide SIMD ともに使用できるが、FECSLD のみ 4-wide SIMD でしか使用できない。

表 7-3　SPARC64™ XIfx の命令セット一覧

命令	XAR.v=0	XAR. v = 1 Regs.	XAR.v = 1 2-wide SIMD	XAR.v = 1 4-wide SIMD	ページ
ADD (ADDcc)	✓	✓			174
ADDC (ADDCcc)	✓	✓			174
ALIGNADDRESS{_LITTLE}	✓				175
AND (ANDcc)	✓	✓			236
ANDN (ANDNcc)	✓	✓			236
ARRAY{8|16|32}	✓				176
Bicc[D]	✓				182
BMASK	✓				177
BPcc	✓				178
BPr	✓				180
BSHUFFLE	✓				177
CALL	✓				184
CASA[PASI], CASXA[PASI]	✓	✓			185
EDGE{8|16|32}{L}cc	✓				187
EDGE{8|16|32}{L}N	✓				188
F{ADD|SUB}ds	✓	✓	✓	✓	70
F{i|x}TO{s|d}	✓	✓	✓	✓	76
F{i|x}TOq	✓	✓			76
F{s|d}TO{i|x}	✓	✓	✓	✓	79
F{s|d}TO{s|d}	✓	✓	✓	✓	189
F{s|d}TOq	✓	✓			189
F{s|d}TOw		✓	✓	✓	79
F{S|Z}EXTW	✓	✓	✓	✓	69
FABS{s|d}	✓	✓	✓	✓	191
FABSq	✓	✓			191
FADD{s|d}	✓	✓	✓	✓	192
FADDq	✓	✓			192
FALIGNDATA	✓				194
FAND{S}	✓	✓	✓	✓	226
FANDNOT{1|2}{S}	✓	✓	✓	✓	226
FBfcc[D]	✓				195
FBPfcc	✓				197
FCMP{E}{s|d|q}	✓	✓			199
FCMP{LE|LT|GE|GT|EQ|NE}{E}{s|d}	✓	✓	✓	✓	201

命令	XAR.v=0	XAR. v = 1 Regs.	XAR.v = 1 2-wide SIMD	XAR.v = 1 4-wide SIMD	ページ
FCMP{LE|NE|GT|EQ}{16|32}	✓				203
FDIV{s|d|q}	✓	☆			204
FdMULq	✓	✓			218
FECPD	✓	☆	✓	✓	151
FECSLD				✓	147
FEPERMD	✓	☆	✓	✓	144
FESUMMD	✓	☆	✓	✓	149
FEXPAd	✓	✓	✓	✓	205
FEXPAND	✓				207
FLCMP{s|d}	✓	✓			303
FLUSH	✓	✓			171
FLUSHW	✓				208
FMADD{1|2}ds	✓	✓	✓	✓	73
FMADD{s|d}	✓	✓	✓	✓	66
FPMAX{64x|u64x|32x|u32x}	✓	✓	✓	✓	60
FMAX{s|d}	✓	✓	✓	✓	209
FPMIN{64x|u64x|32x|u32x}	✓	✓	✓	✓	60
FMIN{s|d}	✓	✓	✓	✓	209
FMOV{s|d|q}cc	✓				212
FMOV{s|d|q}r	✓				215
FMOV{s|d}	✓	✓	✓	✓	211
FMOVq	✓	✓			211
FMSUB{1|2}ds	✓	✓	✓	✓	73
FMSUB{s|d}	✓	✓	✓	✓	66
FMUL{s|d}	✓	✓	✓	✓	218
FMUL8{SU|UL}x16	✓				217
FMUL8x16	✓				217
FMUL8x16{AU|AL}	✓				217
FMULD8{SU|UL}x16	✓				217
FMULds	✓	✓	✓	✓	72
FMULq	✓	✓			218
FNADD{s|d}	✓	✓	✓	✓	305
FNAND{S}	✓	✓	✓	✓	226
FNEG{s|d}	✓	✓	✓	✓	219
FNEGq	✓	✓			219
FNMADD{1|2}ds	✓	✓	✓	✓	73

命令	XAR.v=0	XAR. v = 1 Regs.	XAR.v = 1 2-wide SIMD	XAR.v = 1 4-wide SIMD	ページ
FNMADD{s|d}	✓	✓	✓	✓	66
FNMSUB{1|2}ds	✓	✓	✓	✓	73
FNMSUB{s|d}	✓	✓	✓	✓	66
FNMUL{s|d}	✓	✓	✓	✓	307
FNOR{S}	✓	✓	✓	✓	226
FNOT{1|2}{S}	✓	✓	✓	✓	226
FNsMULd	✓	✓	✓	✓	307
FONE{S}	✓	✓	✓	✓	226
FOR{S}	✓	✓	✓	✓	226
FORNOT{1|2}{S}	✓	✓	✓	✓	226
FPACK{16|32|FIX}	✓				220
FPADD{16|32}{S}	✓	✓	✓	✓	221
FPADD64	✓	✓	✓	✓	62
FPCMP{64|U64}X	✓	✓			312
FPCMP{LE|GT}{64x|32x|16x|8x}	✓	✓	✓	✓	56
FPCMP{LE|GT}{w|h|b}		✓	✓	✓	56
FPCMPU{EQ|LE|NE|GT}{64x|32x|16x|8x}	✓	✓	✓	✓	56
FPCMPU{EQ|LE|NE|GT}{w|h|b}		✓	✓	✓	56
FPCMPU{LE|NE|GT|EQ}8	✓				301
FPMADDX{|HI}	✓	✓	✓	✓	166
FPMERGE	✓				223
FPMUL64	✓	✓	✓	✓	64
FPSELMOV{8|16|32}X	✓	✓	✓	✓	310
FPSLL64X	✓	✓	✓	✓	65
FPSRA64X	✓	✓	✓	✓	65
FPSRL64X	✓	✓	✓	✓	65
FPSUB{16|32}{S}	✓	✓	✓	✓	224
FPSUB64	✓	✓	✓	✓	63
FqTO{i|x}	✓	✓			79
FqTO{s|d}	✓	✓			189
FRCPA{s|d}	✓	✓	✓	✓	168
FRD{s|d}	✓	✓	✓	✓	164
FRSQRTA{s|d}	✓	✓	✓	✓	168
FSELMOV{s|d}	✓	✓	✓	✓	228
FSHIFTORX	✓	✓	✓	✓	160
FsMULd	✓	✓	✓	✓	218

命令	XAR.v=0	XAR. v = 1 Regs.	XAR.v = 1 2-wide SIMD	XAR.v = 1 4-wide SIMD	ページ
FSQRT{s|d|q}	✓	☆			229
FSRC{1|2}{S}	✓	✓	✓	✓	226
FSUB{s|d}	✓	✓	✓	✓	192
FSUBq	✓	✓			192
FTRIMADDd	✓	✓	✓	✓	230
FTRISMULd	✓	✓	✓	✓	230
FTRISSELd	✓	✓	✓	✓	230
FwTO{s|d}		✓	✓	✓	76
FXNOR{S}	✓	✓	✓	✓	226
FXOR{S}	✓	✓	✓	✓	226
FZERO{S}	✓	✓	✓	✓	226
ILLTRAP	✓				235
JMPL	✓				238
LD{S|U}{B|H|W}, LDX	✓	✓			239
LD{S|U}{B|H|W}A$^{P_{ASI}}$, LDXA$^{P_{ASI}}$	✓	✓			241
LDBLOCKF	✓	✓			243
LDDF	✓	✓	✓	✓	82
LDDFBC			✓	✓	92
LDDFDS		✓	✓	✓	82
LDDFID		✓	✓	✓	102
LDDFST			✓	✓	97
LDF	✓	✓	✓	✓	82
LDF{UW|SW|IB}		✓	✓	✓	82
LDFA$^{P_{ASI}}$, LDDFA$^{P_{ASI}}$, LDQFA$^{P_{ASI}}$	✓	✓	✓	✓	89
LDFBC{|UW|SW|IB}			✓	✓	92
LDFID{|UW|SW}		✓	✓	✓	102
LDFSR^{D}	✓	✓			256
LDFST{|UW|SW|IB}			✓	✓	97
LDQF	✓	✓			82
LDSHORTF	✓				245
LDSTUB	✓	✓			247
LDSTUBA$^{P_{ASI}}$	✓	✓			248
LDTWA$^{D, P_{ASI}}$	✓	✓			252
LDTW^{D}	✓	✓			250
LDTXA^{N}	✓	✓			254
LDXEFSR	✓	✓			309

命令	XAR.v=0	XAR. v = 1 Regs.	XAR.v = 1 2-wide SIMD	XAR.v = 1 4-wide SIMD	ページ
LDXFSR	✓	✓			256
LZD	✓	✓			300
MEMBAR	✓				258
MOVcc	✓	✓			260
MOVr	✓	✓			263
MULScc[D]	✓	✓			264
MULX	✓	✓			266
NOP	✓	✓			267
OR (Orcc)	✓	✓			236
ORN (ORNcc)	✓	✓			236
PADD32	✓	✓			268
PDIST	✓				269
POPC	✓	✓			270
PREFETCH, PREFETCHA[PASI]	✓	✓			140
PREFETCHID	✓	✓	✓	✓	143
RDASI	✓	✓			271
RDasr[PASR]	✓	✓			271
RDCCR	✓	✓			271
RDFPRS	✓	✓			271
RDGSR	✓	✓			271
RDPC	✓	✓			271
RDPCR[PPCR]	✓	✓			271
RDPIC[PPIC]	✓	✓			271
RDSTICK[PNPT]	✓	✓			271
RDTICK[PNPT]	✓	✓			271
RDXASR	✓	✓			271
RDY[D]	✓	✓			271
RESTORE	✓	✓			274
RETURN	✓				273
ROLX	✓	✓			278
SAVE	✓	✓			274
SDIV[D] (SDIVcc[D])	✓	✓			275
SDIVX	✓	✓			266
SETHI	✓	✓			276
SIAM	✓				277
SLEEP	✓				173

命令	XAR.v=0	XAR. v = 1 Regs.	XAR.v = 1 2-wide SIMD	XAR.v = 1 4-wide SIMD	ページ
SLL, SLLX	✓	✓			278
SMUL[D] (SMULcc[D])	✓	✓			280
SNF	✓				153
SRA, SRAX	✓	✓			278
SRL, SRLX	✓	✓			278
SSM	✓				153
ST{B|H|W}, STX	✓	✓			282
ST{B|H|W}A[PASI], STXA[PASI]	✓	✓			283
ST{D}FR	✓	✓	✓	✓	116
STBAR[D]	✓				281
STBI[N]	✓	✓			284
STBLOCKF	✓	✓			285
STDF	✓	✓	✓	✓	107
STDFDS		✓	✓	✓	107
STDFID		✓	✓	✓	131
STDFRID		✓	✓	✓	135
STDFRST			✓	✓	127
STDFST			✓	✓	123
STF	✓	✓	✓	✓	107
STFA[PASI], STDFA[PASI], STQFA[PASI]	✓	✓	✓	✓	113
STFID{|UW}		✓	✓	✓	131
STFRID{|UW}		✓	✓	✓	135
STFRST{|UW}			✓	✓	127
STFRUW		✓	✓	✓	116
STFSR[D], STXFSR	✓	✓			291
STFST{|UW}			✓	✓	123
STFUW		✓	✓	✓	107
STPARTIALF	✓				287
STQF	✓	✓			107
STSHORTF	✓				288
STTWA[D, PASI]	✓	✓			290
STTW[D]	✓	✓			289
SUB (SUBcc)	✓	✓			292
SUBC (SUBCcc)	✓	✓			292
SWAP[D], SWAPA[D, PASI]	✓	✓			293
SXAR{1|2}	✓				294

命令	XAR.v=0	XAR. v = 1 Regs.	XAR.v = 1 2-wide SIMD	XAR.v = 1 4-wide SIMD	ページ
TADDcc (TADDccTV^{D})	✓	✓			295
Tcc	✓				296
TSUBcc (TSUBccTV^{D})	✓	✓			295
UDIV^{D} (UDIVcc^{D})	✓	✓			298
UDIVX	✓	✓			266
UMUL^{D} (UMULcc^{D})	✓	✓			299
WRASI	✓	✓			154
WRASR^{PASR}	✓	✓			154
WRCCR	✓	✓			154
WRFPRS	✓	✓			154
WRGSR	✓	✓			154
WRPCR^{PPCR}	✓	✓			154
WRPIC^{PPIC}	✓	✓			154
WRXAR	✓	✓			154
WRXASR	✓	✓			154
WRY^{D}	✓	✓			154
XFILL	✓	✓			156
XFILL256	✓	✓			156
XNOR (XNORcc)	✓	✓			236
XOR (XORcc)	✓	✓			236

各命令のオペコード表にある HPC-ACE2 拡張の列は、その命令で SPARC64™ XIfx のどの拡張機能が有効かを示している。

- **Regs.**

  XAR 対象であることを示す。拡張された整数および浮動小数点レジスタが使用できるほか、メモリアクセス命令ではセクタキャッシュ指定もできる。

  SIMD 拡張が可能な命令の場合、XAR 対象であるか否かではなく、Non-SIMD での命令の振る舞いや実行の可否を示している。

  - ✓ が記入されている場合 XAR 対象であることを示す。
  - ☆ が記入されている場合 XAR 対象であるが rd レジスタに指定できるのは Element-0 側の FPR のみであることを示している。
  - ※ が記入されている場合、Non-SIMD 演算で XAR 必須であることを示す。この場合 XAR 拡張なしに使用することはできない。

- **SIMD**

  SIMD 拡張が可能であることを示す。

  - ✓ が記入されている場合 2/4-wide SIMD 演算可能なことを示す。
  - 4 が記入されている場合、4-wide SIMD 演算のみ可能なことを示す。

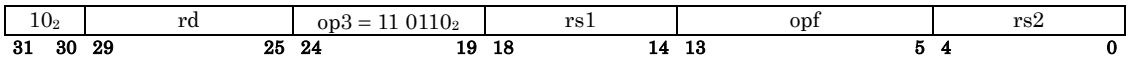
この 2 つの欄のどちらにもマークがついていない命令は XAR 非対象命令である。また、Regs. にマークが記入されておらず、SIMD に✓もしくは 4 が記入されている場合は SIMD 拡張でのみ実行可能であることを示している。

7.1. SIMD Compare

命令	opf	urs3 <0>	操作	HPC-ACE2		アセンブリ言語表記	
				Regs	SIMD		
FPCMPUEQ64x	0 1101 1111_2	0_2	Fd[rs1] = Fd[rs2] 64bit 整数比較	✓	✓	fpcmpueq64x	$freg_{rs1}$, $freg_{rs2}$, $freg_{rd}$
FPCMPLE64x	0 1101 0100_2	0_2	Fd[rs1] ≤ Fd[rs2] 64bit符号付き整数比較	✓	✓	fpcmple64x	$freg_{rs1}$, $freg_{rs2}$, $freg_{rd}$
FPCMPUNE64x	0 1101 0111_2	0_2	Fd[rs1] ≠ Fd[rs2] 64bit 整数比較	✓	✓	fpcmpune64x	$freg_{rs1}$, $freg_{rs2}$, $freg_{rd}$
FPCMPGT64x	0 1101 1100_2	0_2	Fd[rs1] > Fd[rs2] 64bit符号付き整数比較	✓	✓	fpcmpgt64x	$freg_{rs1}$, $freg_{rs2}$, $freg_{rd}$
FPCMPULE64x	0 1101 0101_2	0_2	Fd[rs1] ≤ Fd[rs2] 64bit符号無し整数比較	✓	✓	fpcmpule64x	$freg_{rs1}$, $freg_{rs2}$, $freg_{rd}$
FPCMPUGT64x	0 1101 1101_2	0_2	Fd[rs1] > Fd[rs2] 64bit符号無し整数比較	✓	✓	fpcmpugt64x	$freg_{rs1}$, $freg_{rs2}$, $freg_{rd}$
FPCMPUEQ32x	0 1100 1111_2	0_2	2 つの 32bit 符号無し整数比較 Fd[rs1] = Fd[rs2]	✓	✓	fpcmpueq32x	$freg_{rs1}$, $freg_{rs2}$, $freg_{rd}$
FPCMPLE32x	0 1100 0100_2	0_2	2 つの 32bit 符号付き整数比較 Fd[rs1] ≤ Fd[rs2]	✓	✓	fpcmple32x	$freg_{rs1}$, $freg_{rs2}$, $freg_{rd}$
FPCMPUNE32x	0 1100 0111_2	0_2	2 つの 32bit 符号無し整数比較 Fd[rs1] ≠ Fd[rs2]	✓	✓	fpcmpune32x	$freg_{rs1}$, $freg_{rs2}$, $freg_{rd}$
FPCMPGT32x	0 1100 1100_2	0_2	2 つの 32bit 符号付き整数比較 Fd[rs1] > Fd[rs2]	✓	✓	fpcmpgt32x	$freg_{rs1}$, $freg_{rs2}$, $freg_{rd}$
FPCMPULE32x	0 1100 0101_2	0_2	2 つの 32bit 符号無し整数比較 Fd[rs1] ≤ Fd[rs2]	✓	✓	fpcmpule32x	$freg_{rs1}$, $freg_{rs2}$, $freg_{rd}$
FPCMPUGT32x	0 1100 1101_2	0_2	2 つの 32bit 符号無し整数比較 Fd[rs1] > Fd[rs2]	✓	✓	fpcmpugt32x	$freg_{rs1}$, $freg_{rs2}$, $freg_{rd}$
FPCMPUEQw	0 1100 1111_2	1_2	Fd[rs1] = Fd[rs2] 32bit 整数比較	※	✓	fpcmpueqw	$freg_{rs1}$, $freg_{rs2}$, $freg_{rd}$
FPCMPLEw	0 1100 0100_2	1_2	Fd[rs1] ≤ Fd[rs2] 32bit符号付き整数比較	※	✓	fpcmplew	$freg_{rs1}$, $freg_{rs2}$, $freg_{rd}$
FPCMPUNEw	0 1100 0111_2	1_2	Fd[rs1] ≠ Fd[rs2] 32bit 整数比較	※	✓	fpcmpunew	$freg_{rs1}$, $freg_{rs2}$, $freg_{rd}$
FPCMPGTw	0 1100 1100_2	1_2	Fd[rs1] > Fd[rs2] 32bit符号付き整数比較	※	✓	fpcmpgtw	$freg_{rs1}$, $freg_{rs2}$, $freg_{rd}$
FPCMPULEw	0 1100 0101_2	1_2	Fd[rs1] ≤ Fd[rs2] 32bit符号無し整数比較	※	✓	fpcmpulew	$freg_{rs1}$, $freg_{rs2}$, $freg_{rd}$

命令	opf	urs3 <0>	操作	HPC-ACE2		アセンブリ言語表記	
				Regs	SIMD		
FPCMPUGTw	0 1100 1101_2	1_2	Fd[rs1] > Fd[rs2] 32bit符号無し整数比較	※	✓	fpcmpugtw	$freg_{rs1}$, $freg_{rs2}$, $freg_{rd}$
FPCMPUEQ16x	0 1100 1011_2	0_2	4つの16bit符号無し整数比較 Fd[rs1] = Fd[rs2]	✓	✓	fpcmpueq16x	$freg_{rs1}$, $freg_{rs2}$, $freg_{rd}$
FPCMPLE16x	0 1100 0000_2	0_2	4つの16bit符号付き整数比較 Fd[rs1] ≤ Fd[rs2]	✓	✓	fpcmple16x	$freg_{rs1}$, $freg_{rs2}$, $freg_{rd}$
FPCMPUNE16x	0 1100 0011_2	0_2	4つの16bit符号無し整数比較 Fd[rs1] ≠ Fd[rs2]	✓	✓	fpcmpune16x	$freg_{rs1}$, $freg_{rs2}$, $freg_{rd}$
FPCMPGT16x	0 1100 1000_2	0_2	4つの16bit符号付き整数比較 Fd[rs1] > Fd[rs2]	✓	✓	fpcmpgt16x	$freg_{rs1}$, $freg_{rs2}$, $freg_{rd}$
FPCMPULE16x	0 1100 0001_2	0_2	4つの16bit符号無し整数比較 Fd[rs1] ≤ Fd[rs2]	✓	✓	fpcmpule16x	$freg_{rs1}$, $freg_{rs2}$, $freg_{rd}$
FPCMPUGT16x	0 1100 1001_2	0_2	4つの16bit符号無し整数比較 Fd[rs1] > Fd[rs2]	✓	✓	fpcmpugt16x	$freg_{rs1}$, $freg_{rs2}$, $freg_{rd}$
FPCMPUEQh	0 1100 1011_2	1_2	Fd[rs1] = Fd[rs2] 16bit 整数比較	※	✓	fpcmpueqh	$freg_{rs1}$, $freg_{rs2}$, $freg_{rd}$
FPCMPLEh	0 1100 0000_2	1_2	Fd[rs1] ≤ Fd[rs2] 16bit符号付き整数比較	※	✓	fpcmpleh	$freg_{rs1}$, $freg_{rs2}$, $freg_{rd}$
FPCMPUNEh	0 1100 0011_2	1_2	Fd[rs1] ≠ Fd[rs2] 16bit 整数比較	※	✓	fpcmpuneh	$freg_{rs1}$, $freg_{rs2}$, $freg_{rd}$
FPCMPGTh	0 1100 1000_2	1_2	Fd[rs1] > Fd[rs2] 16bit符号付き整数比較	※	✓	fpcmpgth	$freg_{rs1}$, $freg_{rs2}$, $freg_{rd}$
FPCMPULEh	0 1100 0001_2	1_2	Fd[rs1] ≤ Fd[rs2] 16bit符号無し整数比較	※	✓	fpcmpuleh	$freg_{rs1}$, $freg_{rs2}$, $freg_{rd}$
FPCMPUGTh	0 1100 1001_2	1_2	Fd[rs1] > Fd[rs2] 16bit符号無し整数比較	※	✓	fpcmpugth	$freg_{rs1}$, $freg_{rs2}$, $freg_{rd}$
FPCMPUEQ8x	0 1101 1011_2	0_2	8つの8bit符号無し整数比較 Fd[rs1] = Fd[rs2]	✓	✓	fpcmpueq8x	$freg_{rs1}$, $freg_{rs2}$, $freg_{rd}$
FPCMPLE8x	0 1101 0000_2	0_2	8つの8bit符号付き整数比較 Fd[rs1] ≤ Fd[rs2]	✓	✓	fpcmple8x	$freg_{rs1}$, $freg_{rs2}$, $freg_{rd}$
FPCMPUNE8x	0 1101 0011_2	0_2	8つの8bit符号無し整数比較 Fd[rs1] ≠ Fd[rs2]	✓	✓	fpcmpune8x	$freg_{rs1}$, $freg_{rs2}$, $freg_{rd}$
FPCMPGT8x	0 1101 1000_2	0_2	8つの8bit符号付き整数比較 Fd[rs1] > Fd[rs2]	✓	✓	fpcmpgt8x	$freg_{rs1}$, $freg_{rs2}$, $freg_{rd}$
FPCMPULE8x	0 1101 0001_2	0_2	8つの8bit符号無し整数比較 Fd[rs1] ≤ Fd[rs2]	✓	✓	fpcmpule8x	$freg_{rs1}$, $freg_{rs2}$, $freg_{rd}$
FPCMPUGT8x	0 1101 1001_2	0_2	8つの8bit符号無し整数比較 Fd[rs1] > Fd[rs2]	✓	✓	fpcmpugt8x	$freg_{rs1}$, $freg_{rs2}$, $freg_{rd}$

命令	opf	urs3 <0>	操作	HPC-ACE2		アセンブリ言語表記	
				Regs	SIMD		
FPCMPUEQb	0 1101 1011_2	1_2	Fd[rs1] = Fd[rs2] 8bit 整数比較	※	✓	fpcmpueqb	*freg*$_{rs1}$, *freg*$_{rs2}$, *freg*$_{rd}$
FPCMPLEb	0 1101 0000_2	1_2	Fd[rs1] ≤ Fd[rs2] 8bit 符号付き整数比較	※	✓	fpcmpleb	*freg*$_{rs1}$, *freg*$_{rs2}$, *freg*$_{rd}$
FPCMPUNEb	0 1101 0011_2	1_2	Fd[rs1] ≠ Fd[rs2] 8bit 整数比較	※	✓	fpcmpuneb	*freg*$_{rs1}$, *freg*$_{rs2}$, *freg*$_{rd}$
FPCMPGTb	0 1101 1000_2	1_2	Fd[rs1] > Fd[rs2] 8bit 符号付き整数比較	※	✓	fpcmpgtb	*freg*$_{rs1}$, *freg*$_{rs2}$, *freg*$_{rd}$
FPCMPULEb	0 1101 0001_2	1_2	Fd[rs1] ≤ Fd[rs2] 8bit 符号無し整数比較	※	✓	fpcmpuleb	*freg*$_{rs1}$, *freg*$_{rs2}$, *freg*$_{rd}$
FPCMPUGTb	0 1101 1001_2	1_2	Fd[rs1] > Fd[rs2] 8bit 符号無し整数比較	※	✓	fpcmpugtb	*freg*$_{rs1}$, *freg*$_{rs2}$, *freg*$_{rd}$



動作説明

FPCMP{LE|GT}64x は、Fd[rs1]と Fd[rs2]を 64bit 符号付き整数として比較し、条件が成立していれば Fd[rd]<63>=1_2、不成立なら Fd[rd]<63>=0_2 を格納し、Fd[rd]<62:0>を 0 クリアする。

FPCMPU{EQ|LE|NE|GT}64x は、Fd[rs1]と Fd[rs2]を 64bit 符号無し整数として比較し、条件が成立していれば Fd[rd]<63>=1_2、不成立なら Fd[rd]<63>=0_2 を格納し、Fd[rd]<62:0>を 0 クリアする。

FPCMP{LE|GT}32x は、Fd[rs1]<63:32>と Fd[rs2]<63:32>、Fd[rs1]<31:0>と Fd[rs2]<31:0>をそれぞれ 32bit 符号付き整数として比較し、条件が成立していればそれぞれ Fd[rd]<63>、Fd[rd]<62>に 1_2、不成立なら 0_2 を格納し、Fd[rd]<61:0>を 0 クリアする。
FPCMPU{EQ|LE|NE|GT}32x は、Fd[rs1]<63:32>と Fd[rs2]<63:32>、Fd[rs1]<31:0>と Fd[rs2]<31:0>をそれぞれ 32bit 符号無し整数として比較し、条件が成立していればそれぞれ Fd[rd]<63>、Fd[rd]<62>に 1_2、不成立なら 0_2 を格納し、Fd[rd]<61:0>を 0 クリアする。

FPCMP{LE|GT}16x は、Fd[rs1]＜63:48>と Fd[rs2]<63:48>、Fd[rs1]＜47:32>と Fd[rs2]<47:32>、…、Fd[rs1]<15:0>と Fd[rs2]<15:0>をそれぞれ 16bit 符号付き整数として比較し、条件が成立していればそれぞれ Fd[rd]<63>、Fd[rd]<62>、…、Fd[rd]<60>に 1_2、不成立なら 0_2 を格納し、Fd[rd]<59:0>を 0 クリアする。

FPCMPU{EQ|LE|NE|GT}16x は、Fd[rs1]＜63:48>と Fd[rs2]<63:48>、Fd[rs1]＜47:32>と Fd[rs2]<47:32>、…、Fd[rs1]<15:0>と Fd[rs2]<15:0>をそれぞれ 16bit 符号無し整数として比較し、条件が成立していればそれぞれ Fd[rd]<63>、Fd[rd]<62>、…、Fd[rd]<60>に 1_2、不成立なら 0_2 を格納し、Fd[rd]<59:0>を 0 クリアする。

FPCMP{LE|GT}8x は、Fd[rs1]＜63:56>と Fd[rs2]<63:56>、Fd[rs1]＜55:48>と Fd[rs2]<55:48>、…、Fd[rs1]<7:0>と Fd[rs2]<7:0>をそれぞれ 8bit 符号付き整数として比較し、条件が成立していればそれぞれ Fd[rd]<63>=1_2、Fd[rd]<62>、…、Fd[rd]<56>に 1_2、不成立なら 0_2 を格納し、Fd[rd]<55:0>を 0 クリアする。

FPCMPU{EQ|LE|NE|GT}8x は、Fd[rs1]＜63:56>と Fd[rs2]<63:56>、Fd[rs1]＜55:48>と Fd[rs2]<55:48>、…、Fd[rs1]<7:0>と Fd[rs2]<7:0>をそれぞれ 8bit 符号無し整数として比較し、条件が成立していればそれぞれ Fd[rd]<63>、Fd[rd]<62>、…、Fd[rd]<56>に 1_2、不成立なら 0_2 を格納し、Fd[rd]<55:0>を 0 クリアする。

FPCMP{LE|GT}w は、Fd[rs1]<31:0>と Fd[rs2]<31:0>を 32bit 符号付き整数として比較し、条件が成立していれば Fd[rd]<63>=1_2、不成立なら Fd[rd]<63>=0_2を格納し、Fd[rd]<62:0>を 0 クリアする。

FPCMPU{EQ|LE|NE|GT}w は、Fd[rs1]<31:0>と Fd[rs2]<31:0>を 32bit 符号無し整数として比較し、条件が成立していれば Fd[rd]<63>=1_2、不成立なら Fd[rd]<63>=0_2を格納し、Fd[rd]<62:0>を 0 クリアする。

FPCMP{LE|GT}h は、Fd[rs1]<15:0>と Fd[rs2]<15:0>を 16bit 符号付き整数として比較し、条件が成立していれば Fd[rd]<63>=1_2、不成立なら Fd[rd]<63>=0_2を格納し、Fd[rd]<62:0>を 0 クリアする。

FPCMPU{EQ|LE|NE|GT}h は、Fd[rs1]<15:0>と Fd[rs2]<15:0>を 16bit 符号無し整数として比較し、条件が成立していれば Fd[rd]<63>=1_2、不成立なら Fd[rd]<63>=0_2を格納し、Fd[rd]<62:0>を 0 クリアする。

FPCMP{LE|GT}b は、Fd[rs1]<7:0>と Fd[rs2]<7::0>を 8bit 符号付き整数として比較し、条件が成立していれば Fd[rd]<63>=1_2、不成立なら Fd[rd]<63>=0_2を格納し、Fd[rd]<62:0>を 0 クリアする。

FPCMPU{EQ|LE|NE|GT}b は、Fd[rs1]<7:0>と Fd[rs2]<7:0>を 8bit 符号無し整数として比較し、条件が成立していれば Fd[rd]<63>=1_2、不成立なら Fd[rd]<63>=0_2を格納し、Fd[rd]<62:0>を 0 クリアする。

FPCMP{LE|GT}{w|h|b}及び FPCMPU{EQ|LE|NE|GT}{w|h|b}命令は FSR のどのフィールドも更新しない。また、FPCMP{LE|GT}{64x|32x|16x|8x}及び FPCMPU{EQ|LE|NE|GT}{64x|32x|16x|8x}命令は FSR のどのフィールドも更新しない。

Programming Note FPCMP{LE|GT}{w|h|b}及び FPCMPU{EQ|LE|NE|GT}{w|h|b}（XAR.urs3<0> = 1 で定義された命令)を使用するためには、SXAR の前置が必要である。XAR 拡張を行わない場合は XAR.urs3<0> = 0 として動作する

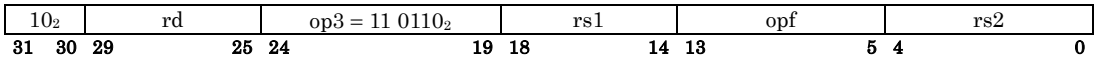
SIMD 動作

有効な要素においては上記の動作を行い、有効でない要素の Fd[rd]には不定値が格納される。

例外	対象命令	検出条件
fp_disabled	全て	PSTATE.pef = 0 または FPRS.fef = 0
illegal_action	FPCMP{LE|GT}64x, FPCMPU{EQ|LE|NE|GT}64x	XAR.v = 1 かつ、下記のいずれかが成立している場合 • XAR.urs3 ≠ 0 • XAR.simd = 1 かつ XAR.urs1<2> ≠ 0 • XAR.simd = 1 かつ XAR.urs2<2> ≠ 0 • XAR.simd = 1 かつ XAR.urd<2> ≠ 0
	FPCMP{LE|GT}{w|h|b}, FPCMPU{EQ|LE|NE|GT}{w|h|b}, FPCMP{LE|GT}{32x|16x|8x}, FPCMPU{EQ|LE|NE|GT}{32x|16x|8x}	XAR.v = 1 かつ、下記のいずれかが成立している場合 • XAR.urs3<2:1> ≠ 00_2 • XAR.simd = 1 かつ XAR.urs1<2> ≠ 0 • XAR.simd = 1 かつ XAR.urs2<2> ≠ 0 • XAR.simd = 1 かつ XAR.urd<2> ≠ 0

7.2. Integer Minimum and Maximum

命令	Opf	操作	HPC-ACE2		アセンブリ言語表記
			Regs	SIMD	
FPMAX64x	0 1110 1100_2	符号付き 64bit 整数最大値	✓	✓	fpmax64x *freg*$_{rs1}$, *freg*$_{rs2}$, *freg*$_{rd}$
FPMAXu64x	0 1110 1101_2	符号無し 64bit 整数最大値	✓	✓	fpmaxu64x *freg*$_{rs1}$, *freg*$_{rs2}$, *freg*$_{rd}$
FPMIN64x	0 1110 1110_2	符号付き 64bit 整数最小値	✓	✓	fpmin64x *freg*$_{rs1}$, *freg*$_{rs2}$, *freg*$_{rd}$
FPMINu64x	0 1110 1111_2	符号無し 64bit 整数最小値	✓	✓	fpminu64x *freg*$_{rs1}$, *freg*$_{rs2}$, *freg*$_{rd}$
FPMAX32x	0 1110 0100_2	符号付き 32bit 整数最大値	✓	✓	fpmax32x *freg*$_{rs1}$, *freg*$_{rs2}$, *freg*$_{rd}$
FPMAXu32x	0 1110 0101_2	符号無し 32bit 整数最大値	✓	✓	fpmaxu32x *freg*$_{rs1}$, *freg*$_{rs2}$, *freg*$_{rd}$
FPMIN32x	0 1110 0110_2	符号付き 32bit 整数最小値	✓	✓	fpmin32x *freg*$_{rs1}$, *freg*$_{rs2}$, *freg*$_{rd}$
FPMINu32x	0 1110 0111_2	符号無し 32bit 整数最小値	✓	✓	fpminu32x *freg*$_{rs1}$, *freg*$_{rs2}$, *freg*$_{rd}$



動作説明

FPMAX64x は、Fd[rs1]と Fd[rs2]を符号付き 64bit 整数として比較し、Fd[rs1] > Fd[rs2]なら Fd[rs1]を、そうでなければ Fd[rs2]を、Fd[rd]に格納する。

FPMAXu64x は、Fd[rs1]と Fd[rs2]を符号無し 64bit 整数として比較し、Fd[rs1] > Fd[rs2]なら Fd[rs1]を、そうでなければ Fd[rs2]を、Fd[rd]に格納する。

FPMIN64x は、Fd[rs1]と Fd[rs2]を符号付き 64bit 整数として比較し、Fd[rs1] < Fd[rs2]なら Fd[rs1]を、そうでなければ Fd[rs2]を、Fd[rd]に格納する。

FPMINu64x は、Fd[rs1]と Fd[rs2]を符号無し 64bit 整数として比較し、Fd[rs1] < Fd[rs2]なら Fd[rs1]を、そうでなければ Fd[rs2]を、Fd[rd]に格納する。

FPMAX32x は、Fd[rs1]<63:32>と Fd[rs2]<63:32>を符号付き 32bit 整数として比較し、Fd[rs1]<63:32> > Fd[rs2]<63:32>なら Fd[rs1]<63:32>を、そうでなければ Fd[rs2]<63:32>を、Fd[rd]<63:32>に格納する。同時に、Fd[rs1]<31:0>と Fd[rs2]<31:0>を符号付き 32bit 整数として比較し、Fd[rs1]<31:0> > Fd[rs2]<31:0>なら Fd[rs1]<31:0>を、そうでなければ Fd[rs2]<31:0>を、Fd[rd]<31:0>に格納する。

FPMAXu32x は、Fd[rs1]<63:32>と Fd[rs2]<63:32>を符号無し 32bit 整数として比較し、Fd[rs1]<63:32> > Fd[rs2]<63:32>なら Fd[rs1]<63:32>を、そうでなければ Fd[rs2]<63:32>を、Fd[rd]<63:32>に格納する。同時に、Fd[rs1]<31:0>と Fd[rs2]<31:0>を符号無し 32bit 整数として比較し、Fd[rs1]<31:0> > Fd[rs2]<31:0>なら Fd[rs1]<31:0>を、そうでなければ Fd[rs2]<31:0>を、Fd[rd]<31:0>に格納する。

FPMIN32x は、Fd[rs1]<63:32>と Fd[rs2]<63:32>を符号付き 32bit 整数として比較し、Fd[rs1]<63:32> < Fd[rs2]<63:32>なら Fd[rs1]<63:32>を、そうでなければ Fd[rs2]<63:32>を、Fd[rd]<63:32>に格納する。同時に、Fd[rs1]<31:0>と Fd[rs2]<31:0>を符号付き 32bit 整数として比較し、Fd[rs1]<31:0> < Fd[rs2]<31:0>なら Fd[rs1]<31:0>を、そうでなければ Fd[rs2]<31:0>を、Fd[rd]<31:0>に格納する。

FPMINu32x は、Fd[rs1]<63:32>と Fd[rs2]<63:32>を符号無し 32bit 整数として比較し、Fd[rs1]<63:32> < Fd[rs2]<63:32>なら Fd[rs1]<63:32>を、そうでなければ Fd[rs2]<63:32>を、Fd[rd]<63:32>に格納する。同時に、Fd[rs1]<31:0>と Fd[rs2]<31:0>を符号無し 32bit 整数として比較し、Fd[rs1]<31:0> < Fd[rs2]<31:0>なら Fd[rs1]<31:0>を、そうでなければ Fd[rs2]<31:0>を、Fd[rd]<31:0>に格納する。

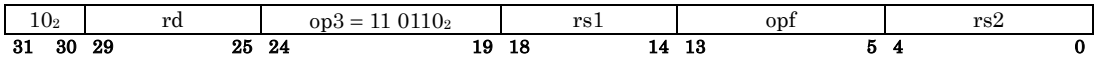
FPMAX{64x|u64x|32x|u32x}及び FMIN{64x|u64x|32x|u32x}命令は FSR のどのフィールドも更新しない。

SIMD 動作　有効な要素においては上記の動作を行い、有効ではない要素の Fd[rd]には不定値が格納される。

例外	対象命令	検出条件
fp_disabled	全て	PSTATE.pef = 0 または FPRS.fef = 0
illegal_action	全て	XAR.v = 1 かつ、下記のいずれかが成立している場合 • XAR.urs3 ≠ 0 • XAR.simd = 1 かつ XAR.urs1<2> ≠ 0 • XAR.simd = 1 かつ XAR.urs2<2> ≠ 0 • XAR.simd = 1 かつ XAR.urd<2> ≠ 0

7.3. Fixed-point Partitioned Add (64-bit)

命令	opf	操作	HPC-ACE2		アセンブリ言語表記	
			Regs	SIMD		
FPADD64	$0\ 0100\ 0010_2$	符号無し 8 バイト整数の加算	✓	✓	fpadd64	$freg_{rs1}$,$freg_{rs2}$,$freg_{rd}$



動作説明　浮動小数点レジスタに格納された符号無し 8 バイト整数の加算を行う。

Fd[rs1]の符号無し 8 バイト整数と Fd[rs2]の符号無し 8 バイト整数を加算し、結果の下位 8 バイトを Fd[rd]に格納する。

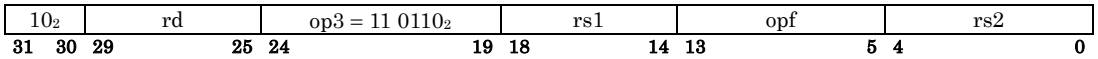
FPADD64 は FSR のどのフィールドも更新しない。

SIMD 動作　有効な要素においては上記の動作を行い、有効でない要素の Fd[rd]は不定値が格納される。

例外	対象命令	検出条件
fp_disabled	全て	PSTATE.pef = 0 または FPRS.fef = 0
illegal_action	全て	XAR.v = 1 かつ、下記のいずれかが成立している場合 • XAR.urs3 ≠ 0 • XAR.simd = 1 かつ XAR.urs1<2> ≠ 0 • XAR.simd = 1 かつ XAR.urs2<2> ≠ 0 • XAR.simd = 1 かつ XAR.urd<2> ≠ 0

7.4. Fixed-point Partitioned Subtract (64-bit)

命令	opf	操作	HPC-ACE2		アセンブリ言語表記	
			Regs	SIMD		
FPSUB64	$0\ 0100\ 0110_2$	符号無し 8 バイト整数の減算	✓	✓	fpsub64	$freg_{rs1}$,$freg_{rs2}$,$freg_{rd}$



動作説明　浮動小数点レジスタに格納された符号無し 8 バイト整数の減算を行う。

Fd[rs1]の符号無し 8 バイト整数と Fd[rs2]の符号無し 8 バイト整数を減算し、結果の下位 8 バイトを Fd[rd]に格納する。

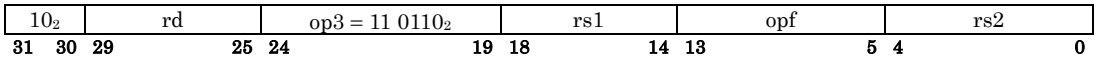
FPSUB64 は FSR のどのフィールドも更新しない。

SIMD 動作　有効な要素においては上記の動作を行い、有効でない要素の Fd[rd]は不定値が格納される。

例外	対象命令	検出条件
fp_disabled	全て	PSTATE.pef = 0 または FPRS.fef = 0
illegal_action	全て	XAR.v = 1 かつ、下記のいずれかが成立している場合 • XAR.urs3 ≠ 0 • XAR.simd = 1 かつ XAR.urs1<2> ≠ 0 • XAR.simd = 1 かつ XAR.urs2<2> ≠ 0 • XAR.simd = 1 かつ XAR.urd<2> ≠ 0

7.5. Fixed-point Partitioned Multiply (64-bit)

命令	opf	操作	HPC-ACE2		アセンブリ言語表記
			Regs	SIMD	
FPMUL64	$0\ 0100\ 1110_2$	符号無し 8 バイト整数の乗算	✓	✓	fpmul64 *freg*$_{rs1}$,*freg*$_{rs2}$,*freg*$_{rd}$



動作説明　浮動小数点レジスタに格納された符号無し 8 バイト整数の乗算を行う。

Fd[rs1]の符号無し 8 バイト整数と Fd[rs2]の符号無し 8 バイト整数を乗算し、結果の下位 8 バイトを Fd[rd]に格納する。

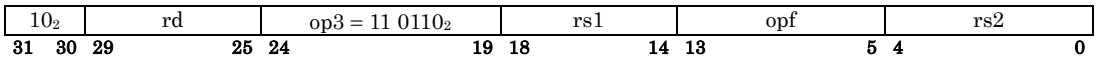
FPMUL64 は FSR のどのフィールドも更新しない。

SIMD 動作　有効な要素においては上記の動作を行い、有効でない要素の Fd[rd]は不定値が格納される。

例外	対象命令	検出条件
fp_disabled	全て	PSTATE.pef = 0 または FPRS.fef = 0
illegal_action	全て	XAR.v = 1 かつ、下記のいずれかが成立している場合 • XAR.urs3 ≠ 0 • XAR.simd = 1 かつ XAR.urs1<2> ≠ 0 • XAR.simd = 1 かつ XAR.urs2<2> ≠ 0 • XAR.simd = 1 かつ XAR.urd<2> ≠ 0

7.6. 64-bit Integer Shift on Floating-Point Register

命令	opf	操作	HPC-ACE2		アセンブリ言語表記
			Regs	SIMD	
FPSLL64x	1 0000 0110_2	倍精度浮動小数点レジスタ左論理シフト	✓	✓	`fpsll64x` *freg*$_{rs1}$, *freg*$_{rs2}$, *freg*$_{rd}$
FPSRL64x	1 0000 0111_2	倍精度浮動小数点レジスタ右論理シフト	✓	✓	`fpsrl64x` *freg*$_{rs1}$, *freg*$_{rs2}$, *freg*$_{rd}$
FPSRA64x	1 0000 1111_2	倍精度浮動小数点レジスタ右算術シフト	✓	✓	`fpsra64x` *freg*$_{rs1}$, *freg*$_{rs2}$, *freg*$_{rd}$



動作説明

これらの命令は、Fd[rs1]のデータを右または左にシフトし、結果を Fd[rd]に格納する。シフト量は Fd[rs2]<5:0>で指定する。

FPSLL64x 命令は、Fd[rs1]の全 64 ビットを左（上位方向）にシフトし、結果を Fd[rd]に格納する。シフトにより空いた右側（下位側）のビットには 0 が入る。

FPSRL64x 命令は、Fd[rs1]の全 64 ビットを右（下位方向）にシフトし、結果を Fd[rd]に格納する。シフトにより空いた左側（上位側）のビットには 0 が入る。

FPSRA64x 命令は、Fd[rs1]の全 64 ビットを右（下位方向）にシフトし、結果を Fd[rd]に格納する。シフトにより空いた左側（上位側）のビットには、Fd[rs1]<63>の値がコピーされる。

FPSLL64x, FPSRL64x 及び FPSRA64x 命令は FSR のどのフィールドも更新しない。

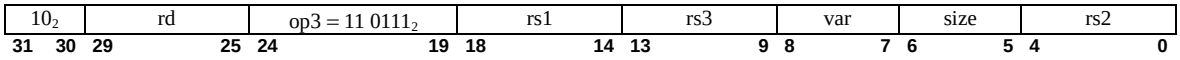
SIMD 動作

有効な要素においては上記の動作を行い、有効ではない要素の Fd[rd]には不定値が格納される。シフト量は各要素の Fd[rs2]<5:0>で指定される。

例外	対象命令	検出条件
fp_disabled	全て	PSTATE.pef = 0 または FPRS.fef = 0
illegal_action	全て	XAR.v = 1 かつ、下記のいずれかが成立している場合 • XAR.urs3 ≠ 0 • XAR.simd = 1 かつ XAR.urs1<2> ≠ 0 • XAR.simd = 1 かつ XAR.urs2<2> ≠ 0 • XAR.simd = 1 かつ XAR.urd<2> ≠ 0

7.7. Floating-Point Multiply-Add/Subtract

命令	var	size	操作	HPC-ACE2		アセンブリ言語表記	
				Regs	SIMD		
FMADDs	00_2	01_2	単精度の乗算と加算	✓	✓	fmadds	$freg_{rs1},freg_{rs2},freg_{rs3},freg_{rd}$
FMADDd	00_2	10_2	倍精度の乗算と加算	✓	✓	fmaddd	$freg_{rs1},freg_{rs2},freg_{rs3},freg_{rd}$
FMSUBs	01_2	01_2	単精度の乗算と減算	✓	✓	fmsubs	$freg_{rs1},freg_{rs2},freg_{rs3},freg_{rd}$
FMSUBd	01_2	10_2	倍精度の乗算と減算	✓	✓	fmsubd	$freg_{rs1},freg_{rs2},freg_{rs3},freg_{rd}$
FNMSUBs	10_2	01_2	単精度の乗算と減算後符号反転	✓	✓	fnmsubs	$freg_{rs1},freg_{rs2},freg_{rs3},freg_{rd}$
FNMSUBd	10_2	10_2	倍精度の乗算と減算後符号反転	✓	✓	fnmsubd	$freg_{rs1},freg_{rs2},freg_{rs3},freg_{rd}$
FNMADDs	11_2	01_2	単精度の乗算と加算後符号反転	✓	✓	fnmadds	$freg_{rs1},freg_{rs2},freg_{rs3},freg_{rd}$
FNMADDd	11_2	10_2	倍精度の乗算と加算後符号反転	✓	✓	fnmaddd	$freg_{rs1},freg_{rs2},freg_{rs3},freg_{rd}$



処理	演算
乗算と加算	F[rd] ← F[rs1] × F[rs2] + F[rs3]
乗算と減算	F[rd] ← F[rs1] × F[rs2] − F[rs3]
乗算と減算後符号反転	F[rd] ← −(F[rs1] × F[rs2] − F[rs3])
乗算と加算後符号反転	F[rd] ← −(F[rs1] × F[rs2] + F[rs3])

Non-SIMD 動作 FMADD{s|d}は、F[rs1]と F[rs2]を乗じた値に F[rs3]を加算し、結果を F[rd]に格納する。

FMSUB{s|d}は、F[rs1]と F[rs2]を乗じた値から F[rs3]を減算し、結果を F[rd]に格納する。

FNMADD{s|d}は、F[rs1]と F[rs2]を乗じた値に F[rs3]を加算し、その符号を反転させた結果を F[rd]に格納する。

FNMSUB{s|d}は、F[rs1]と F[rs2]を乗じた値から F[rs3]を減算し、その符号を反転させた結果を F[rd]に格納する。

浮動小数点積和演算命令は一つの連続した(fused)命令として処理される。つまり、乗算の結果は内部的に丸められることなく無限の精度を持つとして扱われ、FM{ADD|SUB}{s|d}は加減算の後に、FNM{ADD|SUB}{s|d}は符号反転の後に丸め処理が行われる。したがって、丸め処理による誤差が発生するのは一回だけである。

表 7-4 に、SPARC64™ XIfx が浮動小数点積和演算でトラップをどのように処理するかをまとめた。乗算部分での無効処理例外（NV）を検出するか、FSR.ns = 1 で乗算の入力が非正規化数のとき、命令の実行は中止されトラップが発生する。このとき FSR.cexc には例外の情報が表示され、FSR.aexc は更新されない。加減算の部分は乗算部でトラップする無効処理例外（NV）が生じなかったときのみ実行される。

加減算部分でトラップを生じる IEEE 754 例外条件が発生したときは、トラップを生じる例外条件のみが FSR.cexc に表示され、FSR.aexc は更新されない。トラップを生じる IEEE 754 例外条件が発生していない場合、トラップを生じない例外条件の情報が FSR.cexc に表示され、FSR.aexc にはそれまでの FSR.aexc と FSR.cexc の論理和が表示される。*unfinished_FPop* 例外を通知する境界条件は、乗算部分は rs1, rs2 について FMUL と同じ、加減算部分は乗算結果と rs3 について FADD と同じである。

表 7-4　浮動小数点積和演算命令の IEEE 754 例外発生条件

FMUL FADD	IEEE 754 例外(NV, NX のみ) —	例外なし IEEE 754 例外	例外なし 例外なし
cexc	FMUL の例外検出条件と同じ	FADD の例外検出条件と同じ	FADD の例外検出条件(例外なし)と同じ
aexc	更新されない	更新されない	cexc(上記)と aexc の論理和

cexc に表示される値は各種条件に依存する。詳細を表 7-5 にまとめた。この表で uf, of, inv, nx はそれぞれ IEEE 754 で定義された例外（uf: アンダーフロー, of: オーバーフロー, inv: 無効演算, nx: 不正確な演算）でトラップしない場合を意味する。

表 7-5　トラップしない場合の cexc の値（FSR.NS = 0 の時）

		FADD			
		None	nx	of nx	inv
FMUL	none	None	nx	of nx	inv
	inv	inv	—	—	inv

表 7-6　トラップしない場合の cexc の値（FSR.NS = 1 の時）

		FADD				
		None	nx	of nx	uf nx	inv
FMUL	none	None	nx	of nx	uf nx	inv
	inv	inv	—	—	—	inv
	nx	nx	nx	of nx	uf nx	inv nx

表中の “—” のケースは起こりえない

SIMD 動作

SPARC64™ XIfx の SIMD 拡張では、各 Element の演算は独立して行われることになっている。命令で指定するのはレジスタ Fd[0] ~ Fd[254]なので、rs1, rs2, rs3, rd の最上位ビットは必ず 0 になる。これに対し、FMADD の SIMD 拡張では、制限つきではあるが使用する Element の LSB を反転させることが可能となっている。

> **Note**　ここに書かれているのは XAR.simd = 1 のときのことである。XAR.simd = 0 のときは rs1, rs2, rs3, rd に Element-0, Element-1 のすべての浮動小数点レジスタが使える。

FMADD の SIMD 拡張では、rs1, rs2 にレジスタ Fd[2n] (n = 0 ~ 255) を指定することができる。Fd[2n](n = 128 ~ 255)のレジスタを指定すると、実際に使用するレジスタは Fd[2n-128]であるが、使用するレジスタの Element 番号 LSB を反転させる。たとえば、Fd[2n] (n = 128 ~ 255) を使用すると、Element-0 側の演算では、Element-1 のレジスタを、Element-1 側の演算では、Element-0 のレジスタをそれぞれ使用する。同様に Element-2 側の演算では、Element-3 のレジスタを、Element-3 側の演算では Element-2 のレジスタを使用する。

> **Compatibility Note**　表記を変更しているが、2-wide SIMD 時の動作は SPARC64™ VIIIfx, SPARC64™ IXfx と同等である

これに対し rs3 と rd は他の SIMD 拡張と同じで、Fd[2n] (n = 0 ~ 127)という制限のままである。したがって、urs3<2>と urd<2>はレジスタ指示には使われない。他の SIMD 拡張ではこれらのビットは 0 である必要があるが、FMADD ではこのビットを、演算を変化させるために使用する。urs3<2> = 1 のとき、Element 番号が Odd 側の演算の rs1 には Element 番号が Even 側と

同じものが使われる。urd<2> = 1 のとき、Element 番号が Odd 側演算の積演算の符号を反転させる。

FMADD の SIMD 演算での XAR.urs1, XAR.urs2, XAR.urs3, XAR.urd の意味をまとめると以下のようになる。

- XAR.urs1<2>　　rs1 に使うレジスタの Element 番号 LSB を反転
- XAR.urs2<2>　　rs2 に使うレジスタの Element 番号 LSB を反転
- XAR.urs3<2>　　rs1 に使うレジスタの Odd 側 Element 番号のみ LSB を反転
- XAR.urd<2>　　Odd 側 Element 番号の積演算の符号を反転

表 7-7　SIMD 演算時の操作

命令	Even Element 演算側	Odd Element 演算側
fmadd	$frd_b \leftarrow frs1 \times frs2 + frs3_b$	$frd_e \leftarrow (-1)^n \times (c\,?\,frs1 : frs1_i) \times frs2_i + frs3_e$
fmsub	$frd_b \leftarrow frs1 \times frs2 - frs3_b$	$frd_e \leftarrow (-1)^n \times (c\,?\,frs1 : frs1_i) \times frs2_i - frs3_e$
fnmsub	$frd_b \leftarrow -(frs1 \times frs2 - frs3_b)$	$frd_e \leftarrow -((-1)^n \times (c\,?\,frs1 : frs1_i) \times frs2_i - frs3_e)$
fnmadd	$frd_b \leftarrow -(frs1 \times frs2 + frs3_b)$	$frd_e \leftarrow -((-1)^n \times (c\,?\,frs1 : frs1_i) \times frs2_i + frs3_e)$

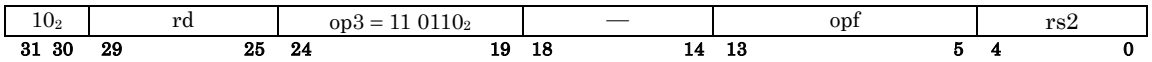
表 7-8　レジスタ表記の意味

frs1: Element[2s^1'b0::urs1<2>] urs1<1:0>::rs1<5:1>::1'b0	$frs1_i$: Element[2s^1'b0::¬urs1<2>] urs1<1:0>::rs1<5:1>::1'b0
frs2: Element[2s^1'b0::urs2<2>] urs2<1:0>::rs2<5:1>::1'b0	$frs2_i$: Element[2s^1'b0:: ¬urs2<2>] urs2<1:0>::rs2<5:1>::1'b0
$frs3_b$: Element[2s] urs3<1:0>::rs3<5:1>::1'b0	$frs3_e$: Element[2s+1] urs3<1:0>::rs3<5:1>::1'b0
frd_b: Element[2s] urd<1:0>::rd<5:1>::1'b0	frd_e: Element[2s+1] urd<1:0>::rd<5:1>::1'b0
c:　urs3<2>	
n:　urd<2>	s = 0 or 1 である。

例外		対象命令	検出条件
illegal_instruction		すべて	size = 11_2 かつ var ≠ 11_2 (4 倍精度用 FMA 命令に予約)。
fp_disabled		すべて	PSTATE.pef = 0 または FPRS.fef = 0
fp_exception_ieee_754	NV, NX, OF, UF	すべて	
fp_exception_other (FSR.ftt = *unfinished_FPop*)		すべて	第 8 章を参照

7.8. Integer Sign/Zero Extension

命令	opf	操作	HPC-ACE2		アセンブリ言語表記
			Regs	SIMD	
FSEXTW	$1\ 0000\ 0000_2$	倍精度浮動小数点レジスタの 32 ビット整数を符号拡張	✓	✓	`fsextw` *freg*$_{rs2}$, *freg*$_{rd}$
FZEXTW	$1\ 0000\ 0001_2$	倍精度浮動小数点レジスタの 32 ビット整数をゼロ拡張	✓	✓	`fzextw` *freg*$_{rs2}$, *freg*$_{rd}$



動作説明

FSEXTW 命令は、Fd[rs2]<31:0>の整数データを符号拡張し、Fd[rd]に 64 ビット符号付き整数データとして格納する。Fd[rd]<31:0>は Fd[rs2]<31:0>がコピーされ、Fd[rd]<63:32>は Fd[rs2]<31>の値がコピーされる。Fd[rs2]<63:32>に値が入っていた場合、その値は無視される。

FZEXTW 命令は、Fd[rs2]の上位 32 ビットを 0 クリアし、Fd[rd]に格納する。Fd[rd]<31:0>は Fd[rs2]<31:0>がコピーされ、Fd[rd]<63:32>は All 0 が格納される。Fd[rs2]<63:32>に値が入っていた場合、その値は無視される。

FSEXTW, FZEXTW 命令は FSR のどのフィールドも更新しない。

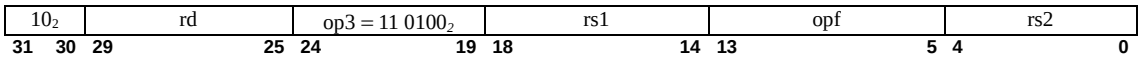
SIMD 動作

有効な要素においては上記の動作を行い、有効ではない要素の Fd[rd]には不定値が書き込まれる。

例外	対象命令	検出条件
illegal_instruction	全て	*reserved* が 0 でないとき
fp_disabled	全て	PSTATE.pef = 0 または FPRS.fef = 0
illegal_action	全て	XAR.v = 1 かつ、下記のいずれかが成立している場合 • XAR.urs3 ≠ 0 • XAR.simd = 1 かつ XAR.urs2<2> ≠ 0 • XAR.urs1 ≠ 0 • XAR.simd = 1 かつ XAR.urd<2> ≠ 0

7.9. Floating-Point Add and Subtract Dual Single Precision

命令	opf	操作	HPC-ACE2		アセンブリ言語表記	
			Regs.	SIMD		
FADDds	0 0100 0000_2	単精度倍幅加算	✓	✓	faddds	$freg_{rs1}$, $freg_{rs2}$, $freg_{rd}$
FSUBds	0 0100 0100_2	単精度倍幅減算	✓	✓	fsubds	$freg_{rs1}$, $freg_{rs2}$, $freg_{rd}$



動作説明

F{ADD|SUB}ds 命令は倍精度レジスタを単精度倍幅利用する命令である（単精度倍幅利用については 5.3.3 を参照）。上位 4 バイトと下位 4 バイトで 2 つの演算をそれぞれ同時に実行する。

FADDds 命令は、Fd[rs1]<63:32>の浮動小数点数と Fd[rs2]<63:32>の浮動小数点数を加算し、結果を Fd[rd]<63:32>に書き込む。同時に Fd[rs1]<31:0>の浮動小数点数と Fd[rs2]<31:0>の浮動小数点数を加算し、結果を Fd[rd]<31:0>に書き込む。

FSUBds 命令は、Fd[rs1]<63:32>の浮動小数点数から Fd[rs2]<63:32>の浮動小数点数を減算し、結果を Fd[rd]<63:32>に書き込む。同時に Fd[rs1]<31:0>の浮動小数点数から Fd[rs2]<31:0>の浮動小数点数を減算し、結果を Fd[rd]<31:0>に書き込む。

これらの命令によって実行される丸めの方法は、FSR.rd または GSR.irnd によって決まる。

fp_exception_ieee_754, *fp_exeception_other* については演算ごとに例外が検出されることがあるが、例外の優先順位、FSR.ns, FSR.tem, XASR.fed の設定値により処理される。両方の演算 *fp_exception_ieee_754* 例外を検出した場合、2 つの要因が表示・加算されることがある。また 2 つの演算のそれぞれで *fp_exception_ieee_754* と *fp_exception_other* を同時に検出したとき、表 5-9 の優先順位に基づき *fp_exception_other* が優先される。

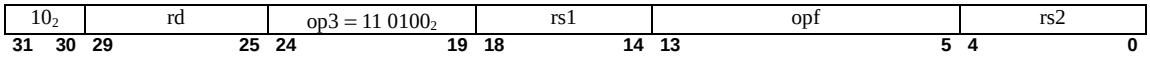
SIMD 動作

有効な要素においては上記の動作を行い、有効でない要素の Fd[rd]には不定値が格納される。

例外		対象命令	検出条件
fp_disabled		すべて	PSTATE.pef = 0 または FPRS.fef = 0
illegal_action		すべて	XAR.v = 1 かつ、下記のいずれかが成立している場合 • XAR.urs3 ≠ 0 • XAR.simd = 1 かつ XAR.urs1<2> ≠ 0 • XAR.simd = 1 かつ XAR.urs2<2> ≠ 0 • XAR.simd = 1 かつ XAR.urd<2> ≠ 0
fp_exception_ieee_754	OF, UF, NX, NV	すべて	IEEE 754 に準拠 同時に 2 つの例外を検出することがある
fp_exception_other (FSR.ftt = *unfinished_FPop*)		FADDds	第 8 章を参照 上位 4 バイトの演算及び下位 4 バイト演算それぞれ、FADDs 命令の検出条件に準ずる。
		FSUBds	第 8 章を参照 上位 4 バイトの演算及び下位 4 バイト演算それぞれ、FSUBs 命令の検出条件に準ずる。

7.10. Floating-Point Multiply Dual Single Precision

命令	opf	操作	HPC-ACE2 Regs	HPC-ACE2 SIMD	アセンブリ言語表記	
FMULds	0 0100 1000_2	単精度倍幅の乗算	✓	✓	fmulds	$freg_{rs1}$, $freg_{rs2}$, $freg_{rd}$



動作説明

FMULds 命令は倍精度レジスタを単精度倍幅利用する命令である（単精度倍幅利用については 5.3.3 を参照）。Fd[rs1]<63:32>の浮動小数点数と Fd[rs2]<63:32>の浮動小数点数を乗算し、結果を Fd[rd]<63:32>に書き込む。同時に Fd[rs1]<31:0>の浮動小数点数と Fd[rs2]<31:0>の浮動小数点数を乗算し、結果を Fd[rd]<31:0>に書き込む。

fp_exception_ieee_754, *fp_exeception_other* については演算ごとに例外が検出されることがあるが、例外の優先順位、FSR.ns, FSR.tem, XASR.fed の設定値により処理される。両方の演算 *fp_exception_ieee_754* 例外を検出した場合、2 つの要因が表示・加算されることがある。また 2 つの演算のそれぞれで *fp_exception_ieee_754* と *fp_exception_other* を同時に検出したとき、表 5-9 の優先順位に基づき *fp_exception_other* が優先される。

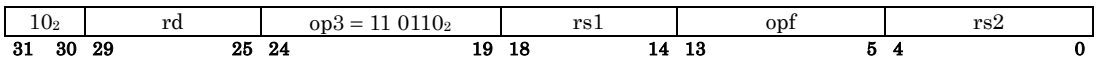
SIMD 動作

有効な要素においては上記の動作を行い、有効でない要素の Fd[rd]には不定値が格納される。

例外		対象命令	検出条件
fp_disabled		すべて	PSTATE.pef = 0 または FPRS.fef = 0
illegal_action		すべて	XAR.v = 1 かつ、下記のいずれかが成立している場合 • XAR.urs3 ≠ 0 • XAR.simd = 1 かつ XAR.urs1<2> ≠ 0 • XAR.simd = 1 かつ XAR.urs2<2> ≠ 0 • XAR.simd = 1 かつ XAR.urd<2> ≠ 0
fp_exception_ieee_754	NV, OF, UF, NX	すべて	IEEE 754 に準拠 同時に 2 つの例外を検出することがある
fp_exception_other (FSR.ftt = *unfinished_FPop*)		すべて	第 8 章を参照 上位 4 バイトの演算及び下位 4 バイト演算それぞれ、FMULs 命令の検出条件に準ずる。

7.11. Floating-Point Multiply-Add/Subtract Dual Single Precision

命令	opf	操作	HPC-ACE2		アセンブリ言語表記
			Regs	SIMD	
FMADD1ds	0 1111 0000	単精度倍幅の乗算と加算	✓	✓	fmadd1ds *freg*$_{rs1}$, *freg*$_{rs2}$, *freg*$_{rd}$
FMADD2ds	0 1111 0100	単精度倍幅の乗算と加算	✓	✓	fmadd2ds *freg*$_{rs1}$, *freg*$_{rs2}$, *freg*$_{rd}$
FMSUB1ds	0 1111 0001	単精度倍幅の乗算と減算	✓	✓	fmsub1ds *freg*$_{rs1}$, *freg*$_{rs2}$, *freg*$_{rd}$
FMSUB2ds	0 1111 0101	単精度倍幅の乗算と減算	✓	✓	fmsub2ds *freg*$_{rs1}$, *freg*$_{rs2}$, *freg*$_{rd}$
FNMSUB1ds	0 1111 0010	単精度倍幅の乗算と減算後符号反転	✓	✓	fnmsub1ds *freg*$_{rs1}$, *freg*$_{rs2}$, *freg*$_{rd}$
FNMSUB2ds	0 1111 0110	単精度倍幅の乗算と減算後符号反転	✓	✓	fnmsub2ds *freg*$_{rs1}$, *freg*$_{rs2}$, *freg*$_{rd}$
FNMADD1ds	0 1111 0011	単精度倍幅の乗算と加算後符号反転	✓	✓	fnmadd1ds *freg*$_{rs1}$, *freg*$_{rs2}$, *freg*$_{rd}$
FNMADD2ds	0 1111 0111	単精度倍幅の乗算と加算後符号反転	✓	✓	fnmadd2ds *freg*$_{rs1}$, *freg*$_{rs2}$, *freg*$_{rd}$



処理	演算
FMADD1ds	Fd[rd]<63:32> ← Fd[rs1]<63:32> × Fd[rs2]<63:32> + Fd[rd]<63:32> Fd[rd]<31:0> ← Fd[rs1]<31:0> × Fd[rs2]<31:0> + Fd[rd]<31:0>
FMADD2ds	Fd[rd]<63:32> ← Fd[rd]<63:32> × Fd[rs1]<63:32> + Fd[rs2]<63:32> Fd[rd]<31:0> ← Fd[rd]<31:0> × Fd[rs1]<31:0> + Fd[rs2]<31:0>
FMSUB1ds	Fd[rd]<63:32> ← Fd[rs1]<63:32> × Fd[rs2]<63:32> – Fd[rd]<63:32> Fd[rd]<31:0> ← Fd[rs1]<31:0> × Fd[rs2]<31:0> – Fd[rd]<31:0>
FMSUB2ds	Fd[rd]<63:32> ← Fd[rd]<63:32> × Fd[rs1]<63:32> – Fd[rs2]<63:32> Fd[rd]<31:0> ← Fd[rd]<31:0> × Fd[rs1]<31:0> – Fd[rs2]<31:0>
FNMSUB1ds	Fd[rd]<63:32> ← –(Fd[rs1]<63:32> × Fd[rs2]<63:32> – Fd[rd]<63:32>) Fd[rd]<31:0> ← –(Fd[rs1]<31:0> × Fd[rs2]<31:0> – Fd[rd]<31:0>)
FNMSUB2ds	Fd[rd]<63:32> ← –(Fd[rd]<63:32> × Fd[rs1]<63:32> – Fd[rs2]<63:32>) Fd[rd]<31:0> ← –(Fd[rd]<31:0> × Fd[rs1]<31:0> – Fd[rs2]<31:0>)
FNMADD1ds	Fd[rd]<63:32> ← –(Fd[rs1]<63:32> × Fd[rs2]<63:32> + Fd[rd]<63:32>) Fd[rd]<31:0> ← –(Fd[rs1]<31:0> × Fd[rs2]<31:0> + Fd[rd]<31:0>)
FNMADD2ds	Fd[rd]<63:32> ← –(Fd[rd]<63:32> × Fd[rs1]<63:32> + Fd[rs2]<63:32>) Fd[rd]<31:0> ← –(Fd[rd]<31:0> × Fd[rs1]<31:0> + Fd[rs2]<31:0>)

動作説明

FM{ADD|SUB}{1|2}ds, FNM{ADD|SUB}{1|2}ds 命令は倍精度レジスタを単精度倍幅利用する命令である（単精度倍幅利用については 5.3.3 を参照）。上位 4 バイトと下位 4 バイトで 2 つの演算をそれぞれ同時に実行する。ソースとして使用したレジスタをデスティネーションと共用している。

FMADD1ds は Fd[rs1]<63:32>と Fd[rs2]<63:32>を乗じた値に Fd[rd]<63:32>を加算し結果を Fd[rd]<63:32>に格納する。同時に Fd[rs1]<31:0>と Fd[rs2]<31:0>を乗じた値に Fd[rd]<31:0>を加算し、結果を Fd[rd]<31:0>に格納する。

FMADD2ds は Fd[rd]<63:32>と Fd[rs1]<63:32>を乗じた値に Fd[rs2]<63:32>を加算し結果を Fd[rd]<63:32>に格納する。同時に Fd[rd]<31:0>と Fd[rs1]<31:0>を乗じた値に Fd[rs2]<31:0>を加算し、結果を Fd[rd]<31:0>に格納する。

FMSUB1ds は Fd[rs1]<63:32>と Fd[rs2]<63:32>を乗じた値から Fd[rd]<63:32>を減算し結果を Fd[rd]<63:32>に格納する。同時に Fd[rs1]<31:0>と Fd[rs2]<31:0>を乗じた値から Fd[rd]<31:0>を減算し、結果を Fd[rd]<31:0>に格納する。

FMSUB2ds は Fd[rd]<63:32>と Fd[rs1]<63:32>を乗じた値から Fd[rs2]<63:32>を減算し結果を Fd[rd]<63:32>に格納する。同時に Fd[rd]<31:0>と Fd[rs1]<31:0>を乗じた値から Fd[rs2]<31:0>を減算し、結果を Fd[rd]<31:0>に格納する。

FNMSUB1ds は Fd[rs1]<63:32>と Fd[rs2]<63:32>を乗じた値から Fd[rd]<63:32>を減算し、その符号を反転させた結果を Fd[rd]<63:32>に格納する。同時に Fd[rs1]<31:0>と Fd[rs2]<31:0>を乗じた値から Fd[rd]<31:0>を減算し、その符号を反転させた結果を Fd[rd]<31:0>に格納する。

FNMSUB2ds は Fd[rd]<63:32>と Fd[rs1]<63:32>を乗じた値から Fd[rs2]<63:32>を減算し、その符号を反転させた結果を Fd[rd]<63:32>に格納する。同時に Fd[rd]<31:0>と Fd[rs1]<31:0>を乗じた値から Fd[rs2]<31:0>を減算し、その符号を反転させた結果を Fd[rd]<31:0>に格納する。

FNMADD1ds は Fd[rs1]<63:32>と Fd[rs2]<63:32>を乗じた値に Fd[rd]<63:32>を加算し、その符号を反転させた結果を Fd[rd]<63:32>に格納する。同時に Fd[rs1]<31:0>と Fd[rs2]<31:0>を乗じた値に Fd[rd]<31:0>を加算し、その符号を反転させた結果を Fd[rd]<31:0>に格納する。

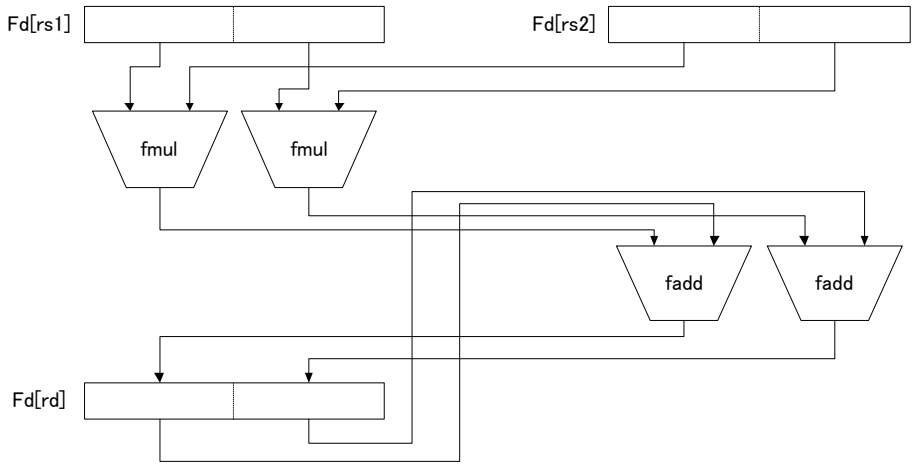
FNMADD2ds は Fd[rd]<63:32>と Fd[rs1]<63:32>を乗じた値に Fd[rs2]<63:32>を加算し、その符号を反転させた結果を Fd[rd]<63:32>に格納する。同時に Fd[rd]<31:0>と Fd[rs1]<31:0>を乗じた値に Fd[rs2]<31:0>を加算し、その符号を反転させた結果を Fd[rd]<31:0>に格納する。

通常の浮動小数点積和演算命令と同様に一つの連続した(fused)命令として処理される。つまり、乗算の結果は内部的に丸められることなく無限の精度を持つとして扱われ、加減算ののちに丸め処理が行われる。したがって、丸め処理による誤差が発生するのは一回だけである。

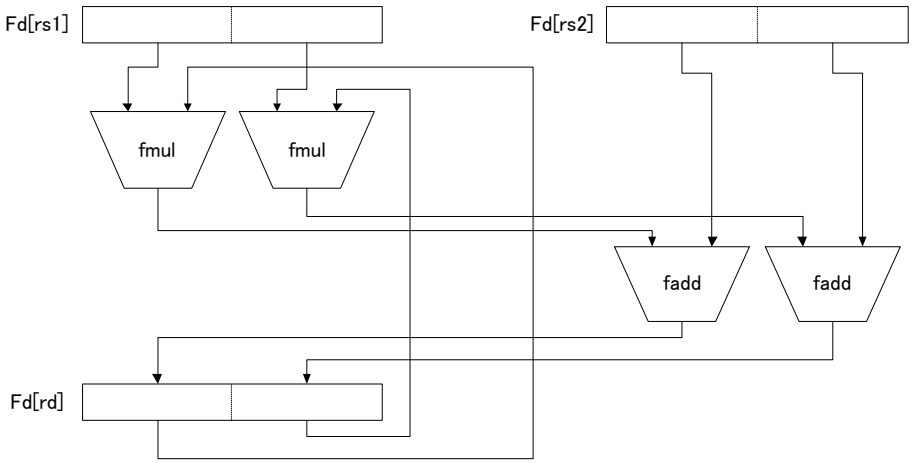
fp_exception_ieee_754, *fp_exeception_other* については演算ごとに例外が検出されることがあるが、例外の優先順位、FSR.ns, FSR.tem, XASR.fed の設定値により処理される。両方の演算 *fp_exception_ieee_754* 例外を検出した場合、2 つの要因が表示・加算されることがある。また 2 つの演算のそれぞれで *fp_exception_ieee_754* と *fp_exception_other* を同時に検出したとき、表 5-9 の優先順位に基づき *fp_exception_other* が優先される。演算で発生する例外の条件については、Floating-Point Multiply-Add/Subtract(ページ 66)と同一である。

FMADD1ds 命令と FMADD2ds 命令の演算イメージを示す。

FMADD1ds



FMADD2ds

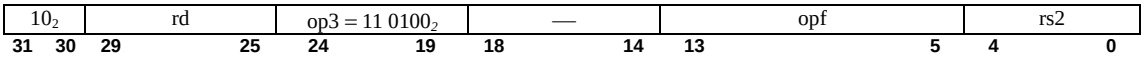


SIMD 動作　　有効な要素においては上記の動作を行い、有効でない要素の Fd[rd]には不定値が格納される。

例外		対象命令	検出条件
fp_disabled		すべて	PSTATE.pef = 0 または FPRS.fef = 0
illegal_action		すべて	XAR.v = 1 かつ、下記のいずれかが成立している場合 • XAR.urs3 ≠ 0 • XAR.simd = 1 かつ XAR.urs1<2> ≠ 0 • XAR.simd = 1 かつ XAR.urs2<2> ≠ 0 • XAR.simd = 1 かつ XAR.urd<2> ≠ 0
fp_exception_ieee_754	NV, NX, OF, UF	すべて	
fp_exception_other (FSR.ftt = *unfinished_FPop*)		すべて	第 8 章を参照 上位 4 バイトの演算及び下位 4 バイト演算それぞれ、乗算部分は FMULs、加算部分は FADDs の検出条件に準ずる。

7.12. Convert Integer to Floating-Point

命令	opf	urs1 <0>	操作	HPC-ACE2		アセンブリ言語表記	
				Regs.	SIMD		
FiTOs	0 1100 0100$_2$	0$_2$	左寄せ 32 ビット整数を単精度浮動小数点数に変換	✓	✓	fitos	*freg$_{rs2}$, freg$_{rd}$*
FiTOd	0 1100 1000$_2$	0$_2$	左寄せ 32 ビット整数を倍精度浮動小数点数に変換	✓	✓	fitod	*freg$_{rs2}$, freg$_{rd}$*
FwTOs	0 1100 0100$_2$	1$_2$	右寄せ 32 ビット整数を単精度浮動小数点数に変換	※	✓	fwtos	*freg$_{rs2}$, freg$_{rd}$*
FwTOd	0 1100 1000$_2$	1$_2$	右寄せ 32 ビット整数を倍精度浮動小数点数に変換	※	✓	fwtod	*freg$_{rs2}$, freg$_{rd}$*
FiTOq	0 1100 1100$_2$	0$_2$	左寄せ 32 ビット整数を 4 倍精度浮動小数点数に変換	✓		fitoq	*freg$_{rs2}$, freg$_{rd}$*
FxTOs	0 1000 0100$_2$	0$_2$	64 ビット整数を単精度浮動小数点数に変換	✓	✓	fxtos	*freg$_{rs2}$, freg$_{rd}$*
FxTOd	0 1000 1000$_2$	0$_2$	64 ビット整数を倍精度浮動小数点数に変換	✓	✓	fxtod	*freg$_{rs2}$, freg$_{rd}$*
FxTOq	0 1000 1100$_2$	0$_2$	64 ビット整数を 4 倍精度浮動小数点数に変換	✓		fxtoq	*freg$_{rs2}$, freg$_{rd}$*



動作説明

FiTOs, FwTOs は、F[rs2]の 32 ビット符号付き整数を単精度浮動小数点数に変換し、F[rd]に格納する。

FiTOd, FwTOd,は、F[rs2]の 32 ビット符号付き整数を倍精度浮動小数点数に変換し、Fd[rd]に格納する。

FiTOq は、F[rs2]の 32 ビット符号付き整数を 4 倍精度の浮動小数点数に変換し、Fq[rd]に格納する。

FiTOs, FiTOd が XAR 拡張されるとき、XAR.urs1<0> (XAR.right_justify) により 32 ビット符号付き整数が格納されている位置の指示を行う。XAR.urs1<0> = 0 のとき、Fd[rs2]<63:32>を 32 ビット符号付き整数として扱う。このとき、Fd[rs2]<31:0>は無視される。XAR.urs1<0> = 1 のとき、FwTOs, FwTOd となり Fd[rs2]<31:0>を 32 ビット符号付き整数として扱う。このとき、Fd[rs2]<63:32>は無視される。

Compatibility Note XAR.urs1<0>で整数の格納位置を選択する仕様は SPARC64™ XIfx で追加された。XAR.urs1<0> = 0 での動作は、SPARC64™ VIIIfx, SPARC64™ IXfx と互換である。

Programming Note SPARC64™ XIfx で追加された FwTOs, FwTOd 命令を使用するためには、SXAR の前置が必要である。

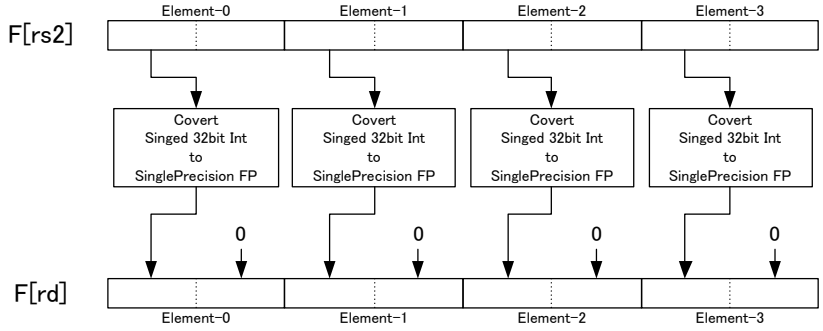
FxTOs, FxTOd, FxTOq は、Fd[rs2]の 64 ビット符号付きをそれぞれ単精度、倍精度、4 倍精度の浮動小数点数に変換し、Fd[rd]に格納する。FxTOs, FxTOd 命令では結果は FSR.rd または GSR.irnd にしたがって丸められる。

SIMD 動作説明

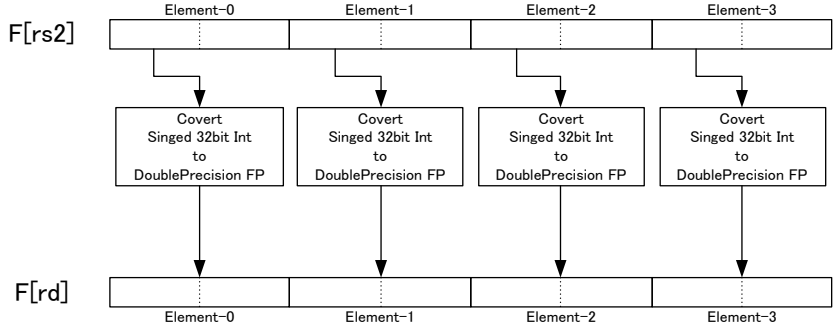
FiTOs, FiTOd, FwTOs, FwTOd, FxTOs, FxTOd は SIMD 拡張される。有効な要素においては上記の動作を行い、有効ではない要素の Fd[rd]には不定値が書き込まれる。

以下に XASR.simd_mode = 1, XAR.v = 1, XAR.simd = 1 の時の FiTOs, FiTOd, FwTOs, FwTOd の動作を示す。

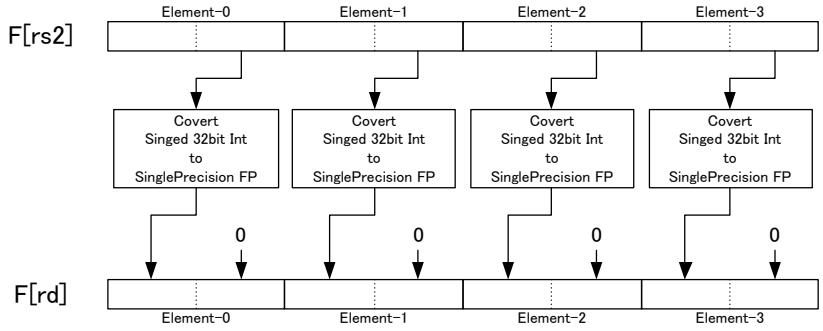
FiTOs (XAR.urs1<0> = 0)



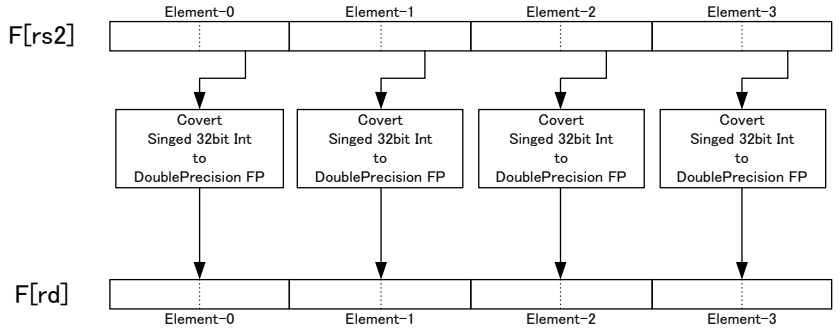
FiTOd (XAR.urs1<0> = 0)



FwTOs (XAR.urs1<0> = 1)



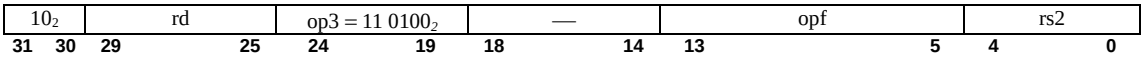
FwTOd (XAR.urs1<0> = 1)



例外		対象命令	検出条件
illegal_instruction		FiTOs, FiTOd, FwTOs, FwTOd, FxTOs, FxTOd	*reserved* フィールドが 0 でない
		FiTOq, FxTOq	常に これらの命令に対する例外は、CPU からは *illegal_instruction* のみを検出する。それより優先度の低い例外は、命令エミュレーションのための仕様である
fp_disabled		すべて	PSTATE.pef = 0 または FPRS.fef = 0
illegal_action		FiTOs, FiTOd FwTOs, FwTOd,	XAR.v = 1 かつ、下記のいずれかが成立している場合 • XAR.urs1<2:1> ≠ 00_2 • XAR.urs3 ≠ 0 • XAR.simd = 1 かつ XAR.urs2<2> ≠ 0 • XAR.simd = 1 かつ XAR.urd<2> ≠ 0
		FxTOs, FxTOd	XAR.v = 1 かつ、下記のいずれかが成立している場合 • XAR.urs1 ≠ 0 • XAR.urs3 ≠ 0 • XAR.simd = 1 かつ XAR.urs2<2> ≠ 0 • XAR.simd = 1 かつ XAR.urd<2> ≠ 0
		FiTOq, FxTOq	XAR.v = 1 かつ、下記のいずれかが成立している場合 • XAR.simd = 1 • XAR.urs1 ≠ 0 • XAR.urs3 ≠ 0
fp_exception_ieee_754	NX	FxTOs, FxTOd, FiTOs, FwTOs	IEEE 754 に準拠
fp_exception_other (FSR.ftt = *invalid_fp_register*)		FxTOq, FiTOq	rs2<1> ≠ 0

7.13. Convert Floating-Point to Integer

命令	opf	urs1 <0>	操作	HPC-ACE2		アセンブリ言語表記	
				Regs.	SIMD		
FsTOx	0 1000 0001_2	0_2	単精度浮動小数点数を 64 ビット整数に変換	✓	✓	fstox	*freg*$_{rs2}$, *freg*$_{rd}$
FdTOx	0 1000 0010_2	0_2	倍精度浮動小数点数を 64 ビット整数に変換	✓	✓	fdtox	*freg*$_{rs2}$, *freg*$_{rd}$
FqTOx	0 0100 0011_2	0_2	4 倍精度浮動小数点数を 64 ビット整数に変換	✓		fqtox	*freg*$_{rs2}$, *freg*$_{rd}$
FsTOi	0 1101 0001_2	0_2	単精度浮動小数点数を 32 ビット整数に変換	✓	✓	fstoi	*freg*$_{rs2}$, *freg*$_{rd}$
FdTOi	0 1101 0010_2	0_2	倍精度浮動小数点数を 32 ビット整数に変換	✓	✓	fdtoi	*freg*$_{rs2}$, *freg*$_{rd}$
FsTOw	0 1101 0001_2	1_2	単精度浮動小数点数を 32 ビット整数に変換し右寄せで格納する	※	✓	fstow	*freg*$_{rs2}$, *freg*$_{rd}$
FdTOw	0 1101 0010_2	1_2	倍精度浮動小数点数を 32 ビット整数に変換し右寄せで格納する	※	✓	fdtow	*freg*$_{rs2}$, *freg*$_{rd}$
FqTOi	0 1101 0011_2	0_2	4 倍精度浮動小数点数を 32 ビット整数に変換	✓		fqtoi	*freg*$_{rs2}$, *freg*$_{rd}$



動作説明

FsTOx, FdTOx, FqTOx は、F[rs2]の浮動小数点数を符号付き 64 ビット整数値に変換し、Fd[rd]に格納する。

FsTOi, FdTOi, FqTOi, FsTOw, FdTOw は、F[rs2]の浮動小数点数を符号付き 32 ビット整数値に変換し、F[rd]に格納する。

FsTOi, FdTOi が XAR 拡張されるとき、XAR.urs1<0> (XAR.right_justify) で結果を格納する位置の指示を行う。XAR.urs1<0> = 0 のとき、結果は Fd[rd]<63:32>に格納され Fd[rd]<31:0>は 0 クリアされる。XAR.urs1<0> = 1 のとき、FsTOw, FdTOw となり結果は 64 ビットに符号拡張され Fd[rd]<63:0>に格納される。つまり、Fd[rd]<63:32>は Fd[rd]<31>で埋められ、Fd[rd]<31:0>に結果が格納される。

Compatibility Note XAR.urs1<0>で整数の格納位置を選択する仕様は SPARC64™ XIfx で追加された。XAR.urs1<0> = 0 での動作は、SPARC64™ VIIIfx, SPARC64™ IXfx と互換である。

Programming Note SPARC64™ XIfx で追加された FsTOw, FdTOw 命令を使用するためには、SXAR の前置が必要である。

結果は、いつも 0 への丸めとされる。すなわち、FSR.rd, GSR.irnd は無視される。

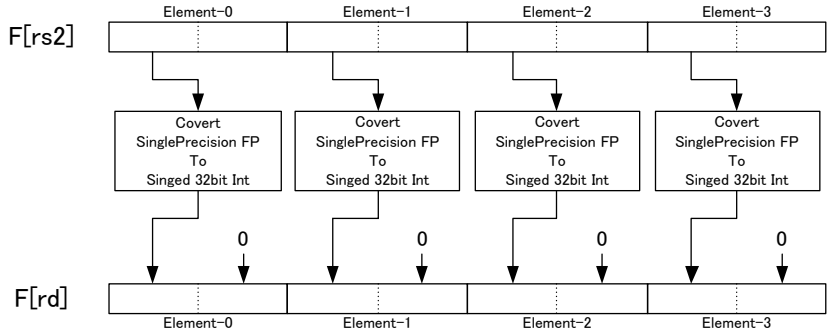
Note SPARC64™ XIfx では 4 倍精度浮動小数点レジスタを参照するハードウェア命令を実装しない。FqTOx, FqTOi 命令は、特権ソフトウェアがその命令をエミュレートする事を可能とするために *illegal_instruction* 例外を生じる。

SIMD 動作説明

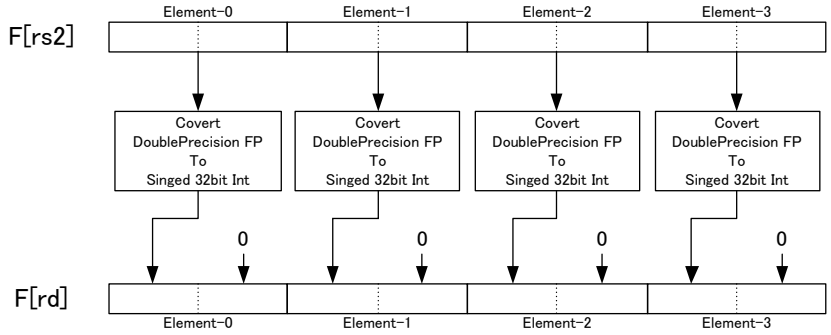
FsTOx, FdTOx, FsTOi, FdTOi, FsTOw, FdTOw は SIMD 拡張される。有効な要素においては上記の動作を行い、有効ではない要素の Fd[rd]には不定値が書き込まれる。

以下に XASR.simd_mode = 1, XAR.v = 1, XAR.simd = 1 の時の FsTOi, FdTOi, FsTOw, FdTOw の動作を示す。

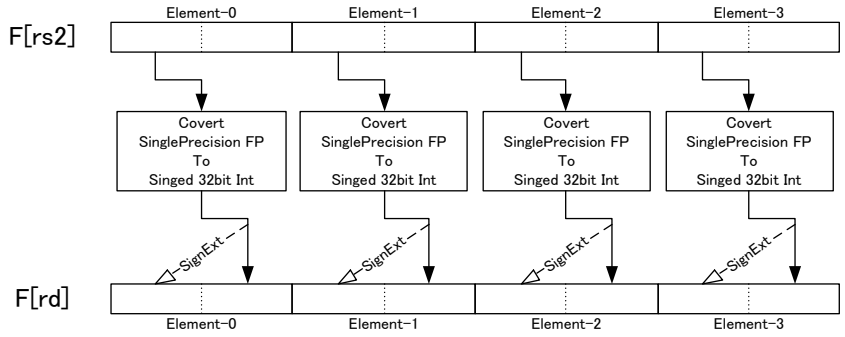
FsTOi (XAR.urs1<0> = 0)



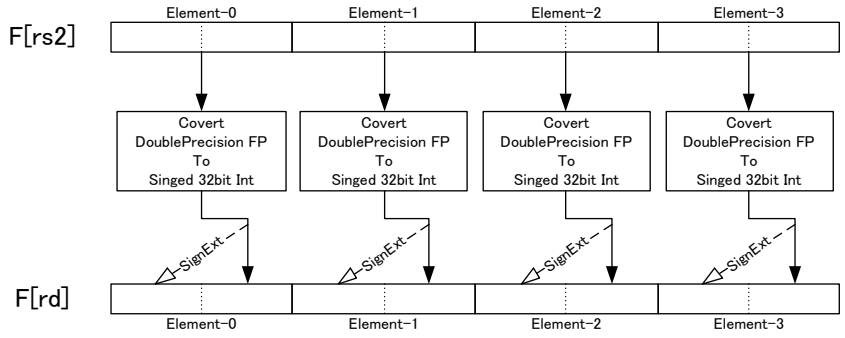
FdTOi (XAR.urs1<0> = 0)



FsTOw (XAR.urs1<0> = 1)



FdTOw (XAR.urs1<0> = 1)

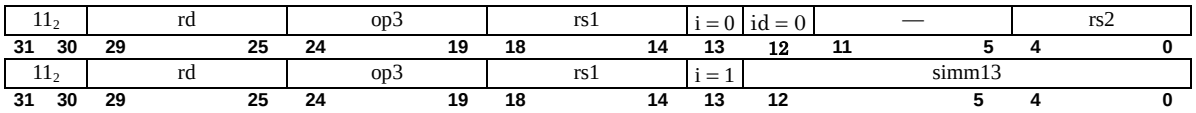


例外		対象命令	検出条件
illegal_instruction		`FsTOx`, `FdTOx`, `FsTOi`, `FdTOi`, `FsTOw`, `FdTOw`	*reserved* フィールドが 0 でない
		`FqTOx`, `FqTOi`	常に これらの命令に対する例外は、CPU からは *illegal_instruction* のみを検出する。それより優先度の低い例外は、命令エミュレーションのための仕様である
fp_disabled		すべて	PSTATE.pef = 0 または FPRS.fef = 0
illegal_action		`FsTOi`, `FdTOi`, `FsTOw`, `FdTOw`	XAR.v = 1 かつ、下記のいずれかが成立している場合 • XAR.urs1<2:1> ≠ 00_2 • XAR.urs3 ≠ 0 • XAR.simd = 1 かつ XAR.urs2<2> ≠ 0 • XAR.simd = 1 かつ XAR.urd<2> ≠ 0
		`FsTOx`, `FdTOx`,	XAR.v = 1 かつ、下記のいずれかが成立している場合 • XAR.urs1 ≠ 0 • XAR.urs3 ≠ 0 • XAR.simd = 1 かつ XAR.urs2<2> ≠ 0 • XAR.simd = 1 かつ XAR.urd<2> ≠ 0
		`FqTOx`, `FqTOi`	XAR.v = 1 かつ、下記のいずれかが成立している場合 • XAR.simd = 1 • XAR.urs1 ≠ 0 • XAR.urs3 ≠ 0
fp_exception_ieee_754	NV, NX	すべて	IEEE 754 に準拠
fp_exception_other (FSR.ftt = *invalid_fp_register*)		`FqTOx`, `FqTOi`	rs2<1> ≠ 0

Compatibility Note *fp_exception_other* (FSR.ftt = *invalid_fp_register*) は UA2011 準拠。JPS1 では 4 倍精度命令の実行で *fp_exception_other* (FSR.ftt = *unimplemented_FPop*) 例外を検出していた。

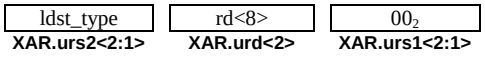
7.14. Load Floating-Point

命令	op3	rd[xv]	urs2 <2:1>	操作	HPC-ACE2 Regs	HPC-ACE2 SIMD	アセンブリ言語表記
LDF	$10\ 0000_2$	0 – 31	—	メモリから単精度浮動小数点レジスタへの読み出し (XAR.v = 0)			ld [*address*], *freg*$_{rd}$
LDF	$10\ 0000_2$	0 – 510	00_2	メモリから倍精度浮動小数点レジスタへの読み出し (XAR.v = 1)	✓	✓	ld [*address*], *freg*$_{rd}$
LDDF	$10\ 0011_2$	0 – 510	00_2	メモリから倍精度浮動小数点レジスタへの読み出し	✓	✓	ldd [*address*], *freg*$_{rd}$
LDQF	$10\ 0010_2$	0 – 510	00_2	メモリから 4 倍精度浮動小数点レジスタへの読み出し	✓		ldq [*address*], *freg*$_{rd}$
LDFUW	$10\ 0000_2$	0 – 510	01_2	メモリから倍精度浮動小数点レジスタへ符号なしワードの読み出し	※	✓	lduw [*address*], *freg*$_{rd}$
LDFIB	$10\ 0000_2$	0 – 510	10_2	メモリから倍精度浮動小数点レジスタの上位と下位へ同一ワードを読み出し	※	✓	ldib [*address*], *freg*$_{rd}$
LDFSW	$10\ 0000_2$	0 – 510	11_2	メモリから倍精度浮動小数点レジスタへ符号付きワードの読み出し	※	✓	ldsw [*address*], *freg*$_{rd}$
LDDFDS	$10\ 0011_2$	0 – 510	01_2	メモリから 2 つの単精度浮動小数を倍精度レジスタへ読み出し	※	✓	lddds[*address*], *freg*$_{rd}$

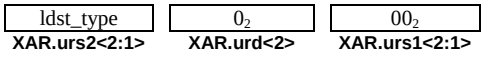


XAR 拡張フィールド (XAR.v = 1 の時のみ有効)

・XAR.simd = 0 のとき



・XAR.simd = 1 のとき



Non-SIMD 動作 LDF 命令は、メモリの 4 バイト境界にある 4 バイト領域の内容を F[rd]に読み出す。XAR.v = 0 のときは単精度浮動小数点レジスタに読み出し、XAR.v = 1 のときは倍精度浮動小数点レジスタの上位 4 バイトに読み出す。

LDDF 命令は、メモリの 4 バイト境界にある 8 バイト領域の内容を Fd[rd]に読み出す。

LDQF 命令は、メモリの 4 バイト境界にある 16 バイト領域の内容を Fq[rd]に読み出す。

LDFUW 命令は、メモリの 4 バイト境界にある 4 バイト領域の内容を符号なしワードとして Fd[rd]に読み出す。

LDFIB 命令は、メモリの 4 バイト境界にある 4 バイト領域の内容を、Fd[rd]の上位 4 バイトと下位 4 バイトに読み出す。

[xv] 5.3.4 "Floating-Point Registers Number Encoding" (39 ページ)で定義される浮動小数点レジスタエンコーディングに従う。

LDFSW 命令は、メモリの 4 バイト境界にある 4 バイト領域の内容を符号拡張し 8 バイト符号付き整数として Fd[rd]に読み出す。

LDDFDS 命令は、メモリの 4 バイト境界にある 8 バイト領域の内容を Fd[rd]に読み出す。

これらの浮動小数点ロード命令は、暗黙の ASI を使いメモリにアクセスする。読み出すアドレスは、i = 0 のときは “R[rs1] + R[rs2]” で、i = 1 のときは “R[rs1] + sign_ext(simm13)” で計算される。

LDF, LDDF, LDDFDS, LDFUW, LDFIB, LDFSW および LDQF 命令は、4 バイト境界にないアドレスにアクセスすると *mem_address_not_aligned* 例外を発生する。

LDDF 命令は、アクセスアドレスは 4 バイト境界にあればよい。しかし、4 バイト境界だが 8 バイト境界にないアドレスにアクセスすると *LDDF_mem_address_not_aligned* 例外を発生する。この例外については、トラップハンドラは LDDF 命令をエミュレートする必要がある。

LDQF 命令は SPARC V9 で定義された命令であるが、SPARC64™ XIfx では実装されていないので、実行すると *illegal_instruction* 例外を発生する。

LDDFDS 命令ではエンディアン変換は、4 バイトごとに行われる。エンディアンが異なるページに属し、それぞれのページのエンディアンが異なる場合も、エンディアンが異なるページの 4 バイト単位のデータのみエンディアン変換が行われる。

LDDFDS 命令ではキャッシャブル領域からのみロードできる。ノンキャッシャブル領域に対してロードを実行しようとすると、*DAE_nc_page* 例外を検出する。

Programming Note　LDF, LDDF 命令のアドレス指定（rs1, rs2）に HPC-ACE 拡張整数レジスタ xg[0] - xg[31]を使う場合、ロードデータを保持するレジスタは倍精度レジスタ Fd[rd]となる。これは XAR.v = 1 のときの rd のデコード定義から来る制約であり、rs1, rs2 に拡張レジスタを指定し、rd に SPARC V9 単精度レジスタ（奇数番号レジスタ）を指定する方法はない。

Programming Note　LDFUW, LDFSW, LDFIB, LDDFDS 命令は XAR.v = 1 のときのみ使用可能である。XAR.v = 0 のとき、XAR 拡張フィールドは All0 として振舞う。

SIMD 動作

SPARC64™ XIfx では LDF, LDDF, LDFU および LDFW 命令は SIMD 拡張される。

LDF 命令は 4 バイト境界の 4×有効な要素数バイト領域の内容を、低位アドレス側から順に有効な要素の Fd[rd]に読み出す。

LDDF 命令は 8 バイト境界の 8×有効な要素数バイト領域の内容を、低位アドレス側から順に有効な要素の Fd[rd]に読み出す。

LDFUW 命令は 4 バイト境界の 4×有効な要素数バイト領域の内容を、低位アドレス側から順に符号なしワードとして有効な要素の Fd[rd]に読み出す。

LDFIB 命令は 4 バイト境界の 4×有効な要素数バイト領域の内容を、低位アドレス側から順に上位 4 バイト下位 4 バイトを同一データで埋めて有効な要素の Fd[rd]に読み出す。

LDFSW 命令は 4 バイト境界の 4×有効な要素数バイト領域の内容を、低位アドレス側から順に符号拡張し 8 バイト符号付き整数として有効な要素の Fd[rd]に読み出す。

LDDFDS 命令は 4 バイト境界の 4×2×有効な要素数バイト領域の内容を、低位アドレス側から順に有効な要素の Fd[rd]に読み出す。

有効でない要素の Fd[rd]は不定値が格納される。

これらの浮動小数点ロード命令は、暗黙の ASI を使いメモリにアクセスする。

アクセス境界違反には *mem_address_not_aligned* 例外が発生する。

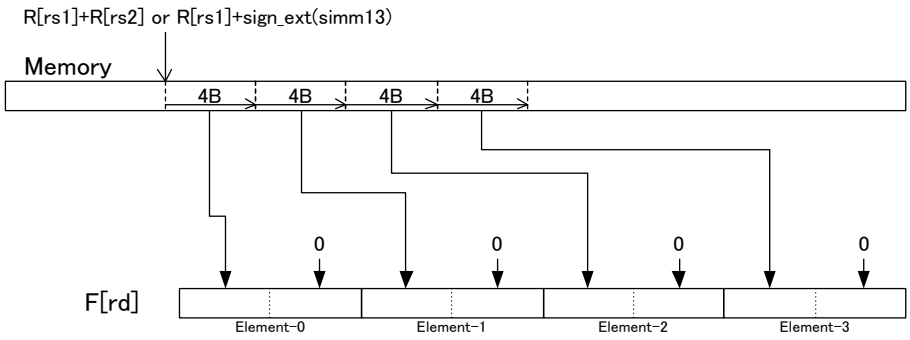
SIMD LDF, LDDF, LDFUW, LDFIB, LDFSW, LDDFDS 命令ではキャッシャブル領域からのみロードできる。ノンキャッシャブル領域に対して SIMD ロードを実行しようとすると、*DAE_nc_page* 例外を検出する。

メモリアクセスのセマンティクスは通常のロード命令と同じく、TSO を遵守する。SIMD LDF, LDDF, LDFUW, LDFIB, LDFSW, LDDFDS 命令では 1 命令で複数の要素を同時にロードするが、要素間でも TSO が遵守される。

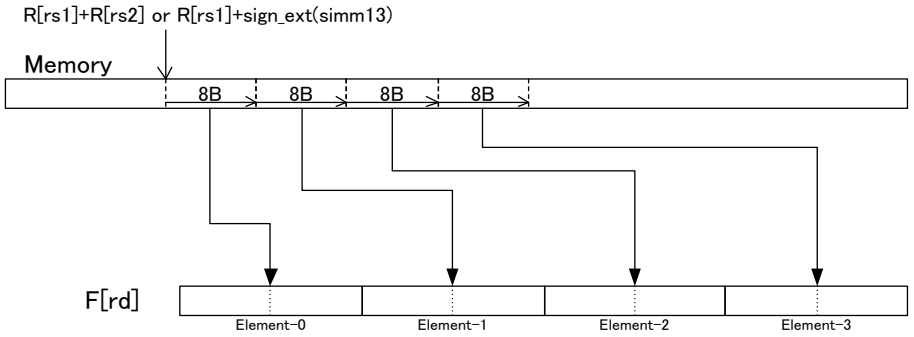
SIMD LDF, LDDF, LDFUW, LDFIB, LDFSW 命令ではエンディアン変換は、要素ごとに個別に行われる。各要素が異なるページに属し、それぞれのページのエンディアンが異なる場合も、エンディアンが異なるページの要素のみエンディアン変換が行われる。LDDFDS 命令ではエンディアン変換は、4 バイトごとに行われる。エンディアンが異なるページに属し、それぞれのページのエンディアンが異なる場合も、エンディアンが異なるページの 4 バイト単位のデータのみエンディアン変換が行われる。

SIMD ロードはどの要素でもウォッチポイントを検出する。

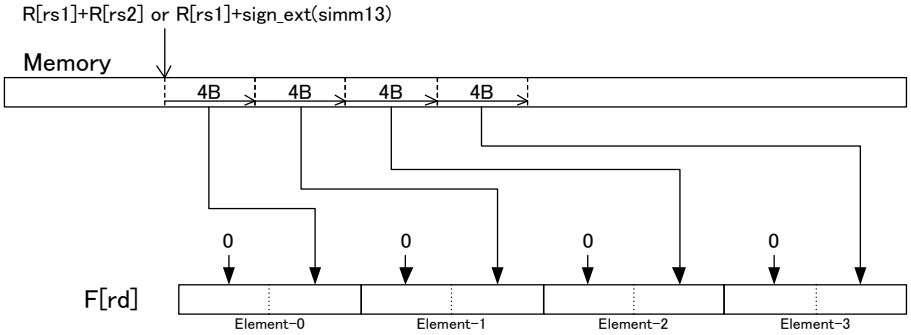
LDF (XAR.v = 1)



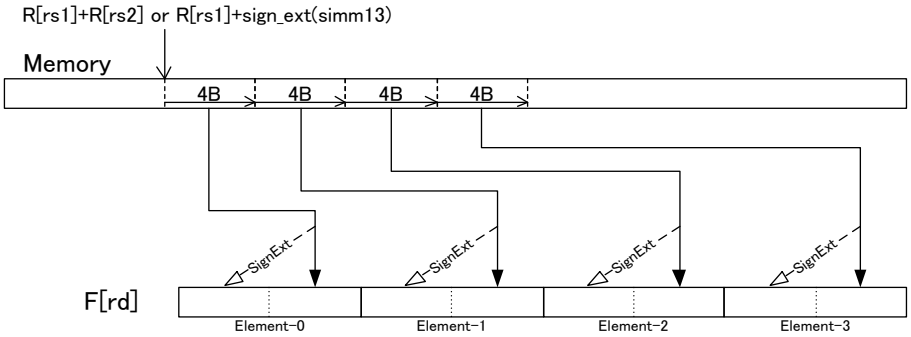
LDDF



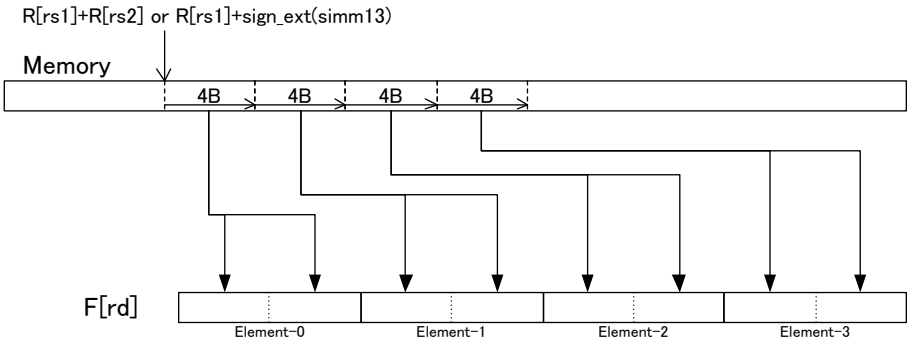
LDFUW



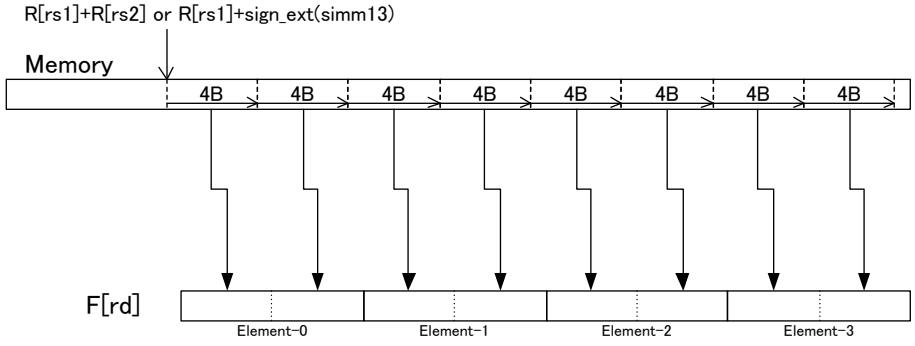
LDFSW



LDFIB



LDDFDS



仮想的な言語を使用した動作記述（SIMD）

```
BIT_32 Load32 (ADDRESS){
  TMP<31:0> ← MEM[ADDRESS,4]
  RETURN TMP
}

BIT_64 Load64 (ADDRESS){
  TMP<63:0> ← MEM[ADDRESS,8]
  RETURN TMP
}

BIT_64 Sign_Ext(DATA32){
  TMP<31:0> ← DATA32
  TMP<63:32> ← DATA32<31> ? 32'hFFFFFFFF : 32'h00000000
  RETURN TMP
}

IF(XASR.simd_mode==0)
  SIMD_WIDTH ← 2
ELSEIF(XASR.simd_mode==1)
  SIMD_WIDTH ← 4
FPR_WIDTH ← 4
```

##LDF

```
ADDR_TMP ← IW.i ? R[rs1]+IW.simm13 : R[rs1]+R[rs2]
FOR (ELEMENT ← 0; ELEMENT<SIMD_WIDTH; ELEMENT++){
  ADDR ← ADDR_TMP + (ELEMENT * 4)
  F[rd][ELEMENT]<63:32> ← Load32(ADDR)
  F[rd][ELEMENT]<31:0> ← 32'h00000000
}
FOR (ELEMENT ← SIMD_WIDTH; ELEMENT<FPR_WIDTH; ELEMENT++)
  F[rd][ELEMENT]<63:0> ← UNDEFINED
```

##LDDF

```
ADDR_TMP ← IW.i ? R[rs1]+IW.simm13 : R[rs1]+R[rs2]
FOR (ELEMENT ← 0; ELEMENT<SIMD_WIDTH; ELEMENT++){
  ADDR ← ADDR_TMP + (ELEMENT * 8)
  F[rd][ELEMENT]<63:0> ← Load64(ADDR)
}
FOR (ELEMENT ← SIMD_WIDTH; ELEMENT<FPR_WIDTH; ELEMENT++)
  F[rd][ELEMENT]<63:0> ← UNDEFINED
```

##LDFUW

```
ADDR_TMP ← IW.i ? R[rs1]+IW.simm13 : R[rs1]+R[rs2]
FOR (ELEMENT ← 0; ELEMENT<SIMD_WIDTH; ELEMENT++){
  ADDR ← ADDR_TMP + (ELEMENT * 4)
  F[rd][ELEMENT]<63:32> ← 32'h00000000
  F[rd][ELEMENT]<31:0> ← Load32(ADDR)
}
FOR (ELEMENT ← SIMD_WIDTH; ELEMENT<FPR_WIDTH; ELEMENT++)
  F[rd][ELEMENT]<63:0> ← UNDEFINED
```

##LDFIB

```
ADDR_TMP ← IW.i ? R[rs1]+IW.simm13 : R[rs1]+R[rs2]
FOR (ELEMENT ← 0; ELEMENT<SIMD_WIDTH; ELEMENT++){
  ADDR ← ADDR_TMP + (ELEMENT * 4)
  DATA_TMP ← Load32(ADDR)
  F[rd][ELEMENT]<63:32> ← DATA_TMP
  F[rd][ELEMENT]<31:0> ← DATA_TMP
}
FOR (ELEMENT ← SIMD_WIDTH; ELEMENT<FPR_WIDTH; ELEMENT++)
  F[rd][ELEMENT]<63:0> ← UNDEFINED
```

##LDFSW

```
ADDR_TMP ← IW.i ? R[rs1]+IW.simm13 : R[rs1]+R[rs2]
FOR (ELEMENT ← 0; ELEMENT<SIMD_WIDTH; ELEMENT++){
  ADDR ← ADDR_TMP + (ELEMENT * 4)
  DATA_TMP ← Load32(ADDR)
  F[rd][ELEMENT]<63:0> ← Sign_Ext(DATA_TMP)
}
FOR (ELEMENT ← SIMD_WIDTH; ELEMENT<FPR_WIDTH; ELEMENT++)
  F[rd][ELEMENT]<63:0> ← UNDEFINED
```

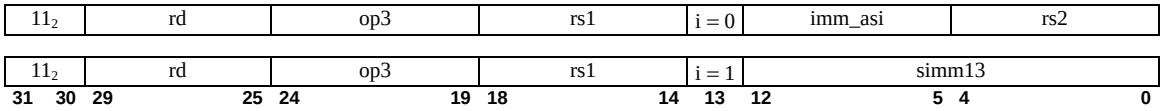
##LDDFDS

```
ADDR_TMP ← IW.i ? R[rs1]+IW.simm13 : R[rs1]+R[rs2]
FOR (ELEMENT ← 0; ELEMENT<SIMD_WIDTH; ELEMENT++){
  ADDR_H ← ADDR_TMP + (ELEMENT * 8)
  ADDR_L ← ADDR_TMP + (ELEMENT * 8) + 4
  F[rd][ELEMENT]<63:32> ← Load32(ADDR_H)
  F[rd][ELEMENT]<31:0> ← Load32(ADDR_L)
}
FOR (ELEMENT ← SIMD_WIDTH; ELEMENT<FPR_WIDTH; ELEMENT++)
  F[rd][ELEMENT]<63:0> ← UNDEFINED
```

例外	対象命令	検出条件
illegal_instruction	LDF, LDDF, LDFUW, LDFSW, LDFIB, LDDFDS	*reserved* が 0 でないとき。
	LDQF	常に これらの命令に対する例外は、CPU からは *illegal_instruction* のみを検出する。それより優先度の低い例外は、命令エミュレーションのための仕様である
fp_disabled	すべて	PSTATE.pef = 0 または FPRS.fef = 0
illegal_action	LDF, LDFUW, LDFSW, LDFIB	XAR.v = 1 かつ、下記のいずれかが成立している場合 • XAR.simd = 0 かつ、XAR.urs1<2:1> ≠ 00_2 • i = 1 かつ XAR.urs2<0> ≠ 0
	LDDF, LDDFDS	XAR.v = 1 かつ、下記のいずれかが成立している場合 • XAR.simd = 0 かつ、XAR.urs1<2:1> ≠ 00_2 • XAR.urs2<2> ≠ 0 • i = 1 かつ XAR.urs2<0> ≠ 0
	LDQF	XAR.v = 1 かつ、下記のいずれかが成立している場合 • XAR.urs1<2:1> ≠ 00_2 • i = 0 かつ XAR.urs2<2:1> ≠ 00_2 • i = 1 かつ XAR.urs2 ≠ 0 • XAR.simd = 1
fp_exception_other (FSR.ftt = *invalid_fp_register*)	LDQF	rd<1> ≠ 0
LDDF_mem_address_not_aligned	LDDF	XAR.v = 0 または XAR.simd = 0 で、4 バイト境界だが 8 バイト境界ではないアドレスにアクセスしたとき。
mem_address_not_aligned	LDF, LDQF, LDFUW, LDFSW, LDFIB, LDDFDS	4 バイト境界ではないアドレスにアクセスしたとき。
	LDDF	下記のいずれかが成立している場合。 • XAR.v = 1 かつ XAR.simd = 1 で、8 バイト境界ではないアドレスにアクセスしたとき。 • XAR.v = 0 もしくは XAR.simd = 0 で、4 バイト境界ではないアドレスにアクセスしたとき。
VA_watchpoint	すべて	
DAE_privilege_violation	すべて	
DAE_nc_page	LDF, LDDF, LDFUW, LDFSW, LDFIB	XAR.v = 1 かつ XAR.simd = 1 で、ノンキャッシャブル空間にアクセスしたとき。
	LDDFDS	ノンキャッシャブル空間にアクセスしたとき。
DAE_nfo_page	すべて	

7.15. Load Floating-Point from Alternate Space

命令	op3	rd[xvi]	操作	HPC-ACE2 Regs	HPC-ACE2 SIMD	アセンブリ言語表記	
LDFA$^{P_{ASI}}$	$11\ 0000_2$	0 – 31	別空間から単精度浮動小数点レジスタへの読み出し（XAR.v = 0）			lda lda	[*address*] ***imm_asi***, *freg*$_{rd}$ [*address*] %asi, *freg*$_{rd}$
LDFA$^{P_{ASI}}$	$11\ 0000_2$	0 – 510	別空間から倍精度浮動小数点レジスタへの読み出し（XAR.v = 1）	✓	✓	lda lda	[*address*] ***imm_asi***, *freg*$_{rd}$ [*address*] %asi, *freg*$_{rd}$
LDDFA$^{P_{ASI}}$	$11\ 0011_2$	0 – 510	別空間から倍精度浮動小数点レジスタへの読み出し	✓	✓	ldda ldda	[*address*] ***imm_asi***, *freg*$_{rd}$ [*address*] %asi, *freg*$_{rd}$
LDQFA$^{P_{ASI}}$	$11\ 0010_2$	0 – 510	別空間から4倍精度浮動小数点レジスタへの読み出し	✓		ldqa ldqa	[*address*] ***imm_asi***, *freg*$_{rd}$ [*address*] %asi, *freg*$_{rd}$



Non-SIMD 動作　LDFA 命令は、別空間の 4 バイト境界にある 4 バイト領域の内容を F[rd]に読み出す。XAR.v = 0 のときは単精度浮動小数点レジスタに読み出し、XAR.v = 1 のときは倍精度浮動小数点レジスタの上位 4 バイトに読み出す。

LDDF 命令は、別空間の 4 バイト境界にある 8 バイト領域の内容を Fd[rd]に読み出す。

LDQF 命令は、別空間の 4 バイト境界にある 16 バイト領域の内容を Fq[rd]に読み出す。

これらの浮動小数点ロード命令は、空間識別子（ASI）を必要とする。ASI は、i = 0 のときは imm_asi フィールドで指示され、i = 1 のときは ASI レジスタの値が使われる。読み出しアドレスは、i = 0 のときは “R[rs1] + R[rs2]” で、i = 1 のときは “R[rs1] + sign_ext(simm13)” で計算される。

LDFA, LDDFA および LDQFA 命令を非特権モードで実行する場合、ASI のビット 7 が 0 だと *privileged_action* 例外を発生する。詳細は 10.Address Space Identifiers(321 ページ)参照。

LDFA, LDDFA および LDQFA 命令は、4 バイト境界にないアドレスにアクセスすると *mem_address_not_aligned* 例外を発生する。

LDDFA 命令は、アクセスアドレスは 4 バイト境界にあればよい。しかし、4 バイト境界だが 8 バイト境界にないアドレスにアクセスすると *LDDF_mem_address_not_aligned* 例外を発生する。この例外については、トラップハンドラは LDDFA 命令をエミュレートする必要がある。

LDQFA 命令は SPARC V9 で定義された命令であるが、SPARC64™ XIfx では実装されていないので、実行すると *illegal_instruction* 例外を発生する。

> **Note**　LDFA, LDDFA, LDQFA 命令は、指定された ASI によらず HPC-ACE2 で拡張された型修飾子及びオペレーション修飾子の対象とはならない。

SIMD 動作　SPARC64™ XIfx では LDFA および LDDFA 命令は SIMD 拡張される。

[xvi] 5.3.4 "Floating-Point Registers Number Encoding” (39 ページ)で定義される浮動小数点レジスタエンコーディングに従う。

LDFA 命令は別空間の 4 バイト境界の 4×有効な要素数バイト領域の内容を、低位アドレス側から順に有効な要素の Fd[rd]に読み出す。

LDDFA 命令は別空間の 8 バイト境界の 8×有効な要素数領域の内容を、低位アドレス側から順に有効な要素の Fd[rd]に読み出す。

有効でない要素の Fd[rd]は不定値が格納される。

アクセス境界違反には *mem_address_not_aligned* 例外が発生する。

SIMD LDFA, LDDFA 命令はキャッシャブル領域からのみロードできる。SIMD LDFA, LDDFA 命令でノンキャッシャブル空間からロードを実行しようとすると、*DAE_nc_page* 例外を検出する。

メモリアクセスのセマンティクスは通常のロード命令と同じく、TSO を遵守する。SIMD LDFA, LDDFA 命令では 1 命令で複数の要素を同時にロードするが、要素間でも TSO が遵守される。

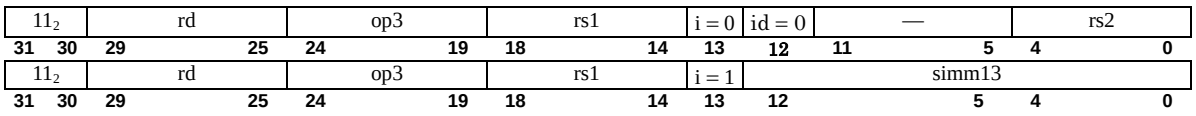
SIMD LDFA, LDDFA 命令におけるエンディアン変換は、エンディアン変換は、要素ごとに個別に行われる。各要素が異なるページに属し、それぞれのページのエンディアンが異なる場合も、エンディアンが異なるページの要素のみエンディアン変換が行われる。

SIMD ロードはどの要素でもウォッチポイントを検出する。

例外	対象命令	検出条件
illegal_instruction	LDQFA	常に これらの命令に対する例外は、CPU からは *illegal_instruction* のみを検出する。それより優先度の低い例外は、命令エミュレーションのための仕様である
fp_disabled	すべて	PSTATE.pef = 0 または FPRS.fef = 0
illegal_action	LDFA, LDDFA	XAR.v = 1 かつ、下記のいずれかが成立している場合 • XAR.urs1<2:1> ≠ 00_2 • i = 0 かつ XAR.urs2<2:1> ≠ 00_2 • i = 1 かつ XAR.urs2 ≠ 0 • XAR.simd = 1 かつ XAR.urd<2> ≠ 0
	LDQFA	XAR.v = 1 かつ、下記のいずれかが成立している場合 • XAR.urs1<2:1> ≠ 00_2 • i = 0 かつ XAR.urs2<2:1> ≠ 00_2 • i = 1 かつ XAR.urs2 ≠ 0 • XAR.simd = 1
fp_exception_other (FSR.ftt = *invalid_fp_register*)	LDQFA	rd<1> ≠ 0
LDDF_mem_address_not_aligned	LDDFA	XAR.v = 0 または XAR.simd = 0 で、4 バイト境界だが 8 バイト境界ではないアドレスにアクセスしたとき。
mem_address_not_aligned	LDFA, LDQFA	4 バイト境界ではないアドレスにアクセスしたとき。
	LDDFA	下記のいずれかが成立している場合。 • XAR.v = 1 かつ XAR.simd = 1 で、8 バイト境界ではないアドレスにアクセスしたとき。 • XAR.v = 0 もしくは XAR.simd = 0 で、4 バイト境界ではないアドレスにアクセスしたとき。
privileged_action	すべて	本文参照
VA_watchpoint	すべて	
DAE_invalid_asi	すべて	本文参照
DAE_privilege_violation	すべて	
DAE_nc_page	すべて	XAR.v = 1 かつ XAR.simd = 1 で、ノンキャッシャブル空間にアクセスしたとき。
DAE_nfo_page	すべて	
DAE_side_effect_page	すべて	

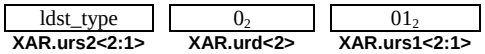
7.16. Broadcast Load Floating-Point

命令	op3	rd[xv]	urs2 <2:1>	操作	HPC-ACE2 Regs	HPC-ACE2 SIMD	アセンブリ言語表記
LDFBC	10 0000$_2$	0 – 254	00$_2$	メモリから倍精度浮動小数点レジスタへの単精度浮動小数点データのブロードキャスト読み出し		✓	ldbc [*address*], *freg*$_{rd}$
LDDFBC	10 0011$_2$	0 – 254	00$_2$	メモリから倍精度浮動小数点レジスタへのブロードキャスト読み出し		✓	lddbc[*address*], *freg*$_{rd}$
LDFBCUW	10 0000$_2$	0 – 254	01$_2$	メモリから倍精度浮動小数点レジスタへの符号なしワードのブロードキャスト読み出し		✓	ldbcuw [*address*], *freg*$_{rd}$
LDFBCIB	10 0000$_2$	0 – 254	10$_2$	メモリから倍精度浮動小数点レジスタの上位と下位へ同一ワードのブロードキャスト読み出し		✓	ldbcib [*address*], *freg*$_{rd}$
LDFBCSW	10 0000$_2$	0 – 254	11$_2$	メモリから倍精度浮動小数点レジスタへの符号付きワードのブロードキャスト読み出し		✓	ldbcsw [*address*], *freg*$_{rd}$



XAR 拡張フィールド

・XAR.simd = 1 のときのみ



動作説明

SPARC64™ XIfx ではこれらの命令は SIMD 拡張でのみ使用される。

命令は単一データをロードして Fd[rd]の各要素に同一データをブロードキャストする。暗黙の ASI を使いメモリにアクセスし、読み出すアドレスは i = 0 のときは “R[rs1] + R[rs2]” で i = 1 のときは“R[rs1] + sign_ext(simm13)” で計算される。

LDFBC 命令は 4 バイト境界の 4 バイト領域の内容を、有効な全要素の Fd[rd]に読み出す。

LDDFBC 命令は 8 バイト境界の 8 バイト領域の内容を、有効な全要素の Fd[rd]に読み出す。

LDFBCSW 命令は 4 バイト境界の 4 バイト領域の内容を、符号拡張し 8 バイト符号付き整数として有効な全要素の Fd[rd]に読み出す。

LDFBCIB 命令は 4 バイト境界の 4 バイト領域の内容を、上位 4 バイト下位 4 バイトを同一データで埋めて有効な要素の Fd[rd]に読み出す。

LDFBCUW 命令は 4 バイト境界の 4 バイト領域の内容を、符号なし整数として有効な全要素の Fd[rd]に読み出す。

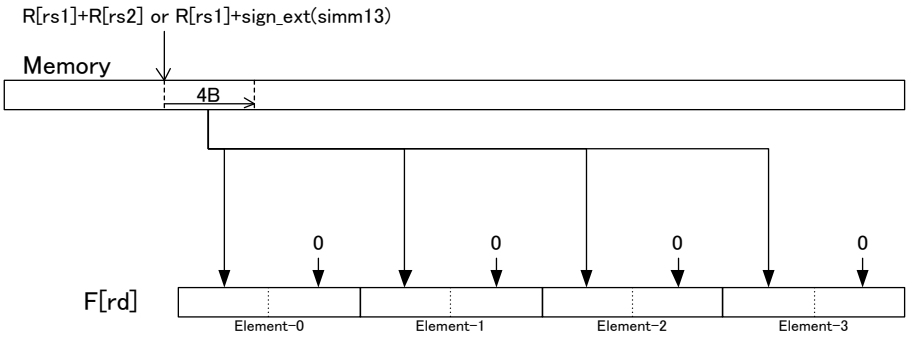
有効でない要素の Fd[rd]は不定値が格納される。

これらの浮動小数点ロード命令は、暗黙の ASI を使いメモリにアクセスする。

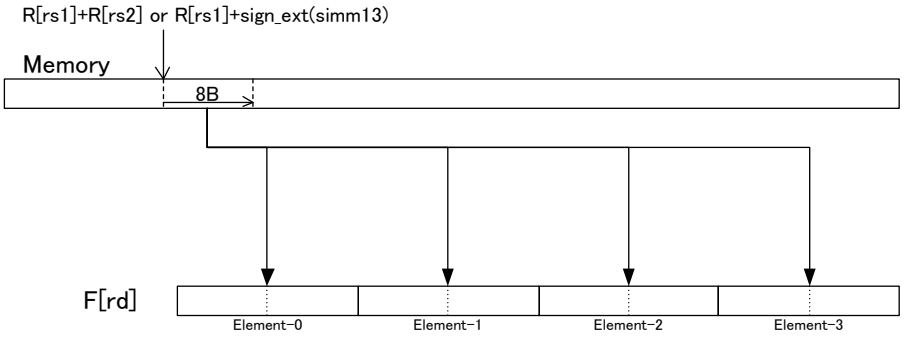
アクセス境界違反には *mem_address_not_aligned* 例外が発生する。

LDFBC, LDDFBC, LDFBCUW, LDFBCIB, LDFBCSW 命令ではキャッシャブル領域からのみロードできる。ノンキャッシャブル領域に対して SIMD ロードを実行しようとすると、*DAE_nc_page* 例外を検出する。

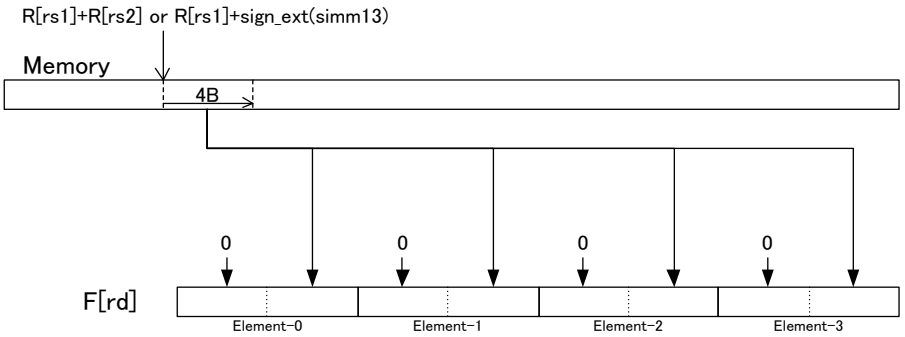
LDFBC



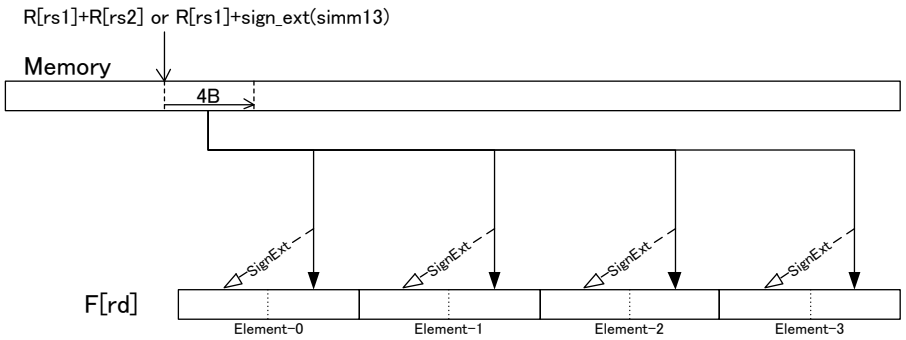
LDDFBC



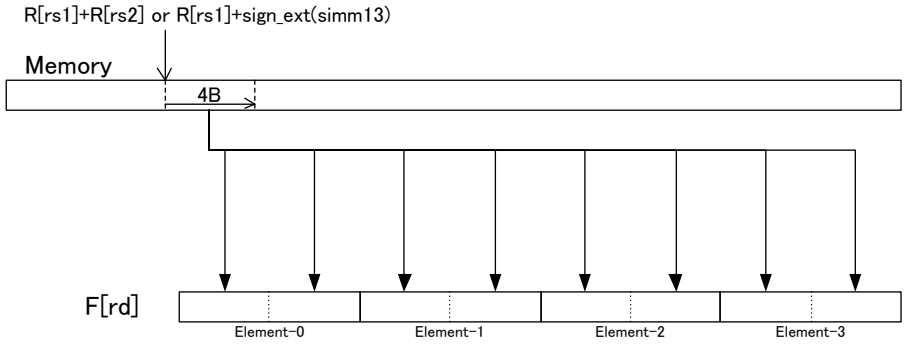
LDFBCUW



LDFBCSW



LDFBCIB



仮想的な言語を使用した動作記述（SIMD）

```
BIT_32 Load32 (ADDRESS){
  TMP<31:0> ← MEM[ADDRESS,4]
  RETURN TMP
}

BIT_64 Load64 (ADDRESS){
  TMP<63:0> ← MEM[ADDRESS,8]
  RETURN TMP
}

BIT_64 Sign_Ext(DATA32){
  TMP<31:0> ← DATA32
  TMP<63:32> ← DATA32<31> ? 32'hFFFFFFFF : 32'h00000000
  RETURN TMP
}

IF(XASR.simd_mode==0)
  SIMD_WIDTH ← 2
ELSEIF(XASR.simd_mode==1)
  SIMD_WIDTH ← 4
FPR_WIDTH ← 4
```

##LDFBC

```
ADDR ← IW.i ? R[rs1]+IW.simm13 : R[rs1]+R[rs2]
TMP<63:32> ← Load32(ADDR)
TMP<31:0> ← 32'h00000000
FOR (ELEMENT ← 0; ELEMENT<SIMD_WIDTH; ELEMENT++)
```

```
  F[rd][ELEMENT]<63:0> ← TMP<63:0>
FOR (ELEMENT ← SIMD_WIDTH; ELEMENT<FPR_WIDTH; ELEMENT++)
  F[rd][ELEMENT]<63:0> ← UNDEFINED
```

##LDDFBC

```
ADDR ← IW.i ? R[rs1]+IW.simm13 : R[rs1]+R[rs2]
TMP<63:0> ← Load64(ADDR)
FOR (ELEMENT ← 0; ELEMENT<SIMD_WIDTH; ELEMENT++)
  F[rd][ELEMENT]<63:0> ← TMP<63:0>
FOR (ELEMENT ← SIMD_WIDTH; ELEMENT<FPR_WIDTH; ELEMENT++)
  F[rd][ELEMENT]<63:0> ← UNDEFINED
```

##LDFBCUW

```
ADDR ← IW.i ? R[rs1]+IW.simm13 : R[rs1]+R[rs2]
TMP<63:32> ← 32'h00000000
TMP<31:0> ← Load32(ADDR)
FOR (ELEMENT ← 0; ELEMENT<SIMD_WIDTH; ELEMENT++)
  F[rd][ELEMENT]<63:0> ← TMP<63:0>
FOR (ELEMENT ← SIMD_WIDTH; ELEMENT<FPR_WIDTH; ELEMENT++)
  F[rd][ELEMENT]<63:0> ← UNDEFINED
```

##LDFBCIB

```
ADDR ← IW.i ? R[rs1]+IW.simm13 : R[rs1]+R[rs2]
DATA_TMP ← Load32(ADDR)
TMP<63:32> ← DATA_TMP
TMP<31:0> ← DATA_TMP
FOR (ELEMENT ← 0; ELEMENT<SIMD_WIDTH; ELEMENT++)
  F[rd][ELEMENT]<63:0> ← TMP<63:0>
FOR (ELEMENT ← SIMD_WIDTH; ELEMENT<FPR_WIDTH; ELEMENT++)
  F[rd][ELEMENT]<63:0> ← UNDEFINED
```

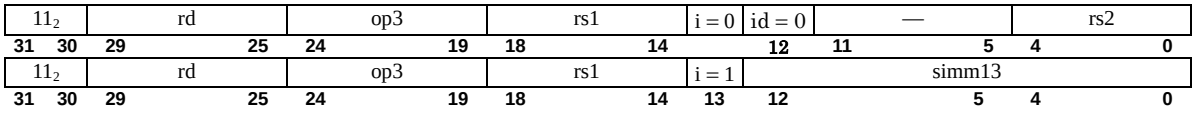
##LDFBCSW

```
ADDR ← IW.i ? R[rs1]+IW.simm13 : R[rs1]+R[rs2]
TMP<31:0> ← Load32(ADDR)
FOR (ELEMENT ← 0; ELEMENT<SIMD_WIDTH; ELEMENT++)
  F[rd][ELEMENT]<63:0> ← Sign_Ext(TMP<31:0>)
FOR (ELEMENT ← SIMD_WIDTH; ELEMENT<FPR_WIDTH; ELEMENT++)
  F[rd][ELEMENT]<63:0> ← UNDEFINED
```

例外	対象命令	検出条件
illegal_instruction	すべて	*reserved* が 0 でないとき。
fp_disabled	すべて	PSTATE.pef = 0 または FPRS.fef = 0
illegal_action	`LDFBC`, `LDFBCUW`, `LDFBCIB`, `LDFBCSW`	下記のいずれかが成立している場合 • i = 1 かつ XAR.urs2<0> ≠ 0
	`LDDFBC`	下記のいずれかが成立している場合 • i = 0 かつ XAR.urs2<2:1> ≠ 00_2 • i = 1 かつ XAR.urs2 ≠ 0
mem_address_not_aligned	`LDFBC`, `LDFBCUW`, `LDFBCSW`, `LDFBCIB`,	4 バイト境界ではないアドレスにアクセスしたとき。
	`LDDFBC`	8 バイト境界ではないアドレスにアクセスしたとき。
VA_watchpoint	すべて	
DAE_privilege_violation	すべて	
DAE_nc_page	すべて	ノンキャッシャブル空間にアクセスしたとき。
DAE_nfo_page	すべて	

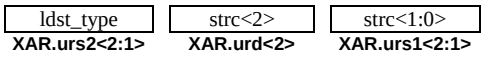
7.17. Stride Load Floating-Point

命令	op3	rd[xv]	urs2 <2:1>	操作	HPC-ACE2		アセンブリ言語表記
					Regs	SIMD	
LDFST	$10\ 0000_2$	0 – 254	00_2	メモリから倍精度浮動小数点レジスタへの単精度浮動小数点データの一定アドレス間隔読み出し		✓	ldst [*address*] @*strc, freg*$_{rd}$
LDDFST	$10\ 0011_2$	0 – 254	00_2	メモリから倍精度浮動小数点レジスタへの一定アドレス間隔読み出し		✓	lddst [*address*] @*strc, freg*$_{rd}$
LDFSTUW	$10\ 0000_2$	0 – 254	01_2	メモリから倍精度浮動小数点レジスタへの符号なし整数の一定アドレス間隔読み出し		✓	ldstuw [*address*] @*strc, freg*$_{rd}$
LDFSTIB	$10\ 0000_2$	0 – 254	10_2	メモリから倍精度浮動小数点レジスタの上位と下位へ同一ワードで埋める一定アドレス間隔読み出し		✓	ldstib [*address*] @*strc, freg*$_{rd}$
LDFSTSW	$10\ 0000_2$	0 – 254	11_2	メモリから倍精度浮動小数点レジスタへの符号付き整数の一定アドレス間隔読み出し		✓	ldstsw [*address*] @*strc, freg*$_{rd}$



XAR 拡張フィールド

・XAR.simd = 1 のときのみ



動作説明

SPARC64™ XIfx ではこれらの命令は SIMD 拡張でのみ使用される。

メモリ上に一定アドレス間隔で複数個配置されているデータの読み出しを行う。アドレスの間隔は strc で指定される。strc = XAR.ldst_SIMD_op (2≦XAR.ldst_SIMD_op≦7) である。LDFST, LDFSTSW, LDFSTIB, LDFSTUW では 8, 12, 16, 20, 24, 28 アドレス間隔での読み出しが可能である。LDDFST では 16, 24, 32, 40, 48, 56 アドレス間隔での読み出しが可能である。これを超えるものについては、Indirect Load Floating-Point を利用する。暗黙の ASI を使いメモリにアクセスし、読み出すアドレスはそれぞれの要素(Element)で、i =0 のときは “R[rs1] + R[rs2] + (Element * strc * <byte>)” で、i = 1 のときは“R[rs1] + sign_ext(simm13) + (Element * strc * <byte>)” で計算される。<byte>には LDFST, LDFSTSW, LDFSTIB, LDFSTUW では 4 を使用し、LDDFST では 8 を使用する。

LDFST 命令は 4 バイト境界の 4 バイト領域の内容を一定アドレス間隔ごとに、有効な全要素の Fd[rd]に読み出す。

LDDFST 命令は 8 バイト境界の 8 バイト領域の内容を一定アドレス間隔ごとに、有効な全要素の Fd[rd]に読み出す。

LDFSTUW 命令は 4 バイト境界の 4 バイト領域の内容を一定アドレス間隔ごとに、符号なし整数として有効な全要素の Fd[rd]に読み出す

LDFSTIB 命令は 4 バイト境界の 4 バイト領域の内容を一定アドレス間隔ごとに、上位 4 バイト下位 4 バイトを同一データで埋めて有効な要素の Fd[rd]に読み出す。

LDFSTSW 命令は 4 バイト境界の 4 バイト領域の内容を一定アドレス間隔ごとに、符号拡張し 8 バイト符号付き整数として有効な全要素の Fd[rd]に読み出す。

有効でない要素の Fd[rd]は不定値が格納される。

アクセス境界違反には *mem_address_not_aligned* 例外が発生する。

これらの浮動小数点ロード命令は、暗黙の ASI を使いメモリにアクセスする。

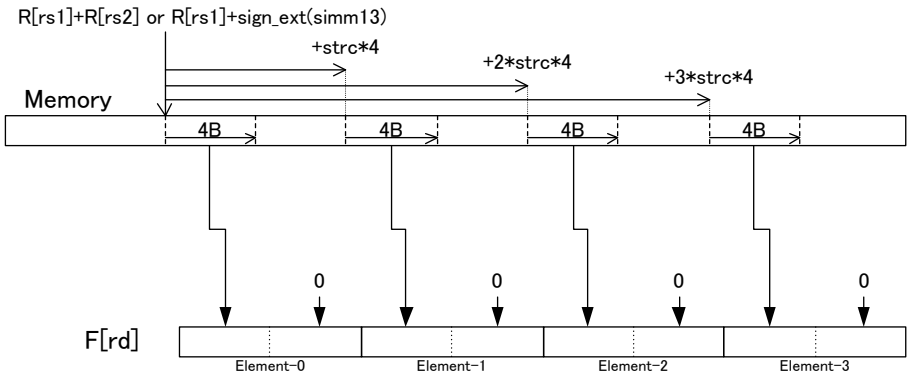
LDFST, LDDFST, LDFSTUW, LDFSTIB, LDFSTSW 命令ではキャッシャブル領域からのみロードできる。ノンキャッシャブル領域に対して SIMD ロードを実行しようとすると、*DAE_nc_page* 例外を検出する。

メモリアクセスのセマンティクスは通常のロード命令と同じく、TSO を遵守する。SIMD LDFST,LDDFST, LDFSTUW, LDFSTIB, LDFSTSW 命令では 1 命令で複数の要素を同時にロードするが、要素間でも TSO が遵守される。

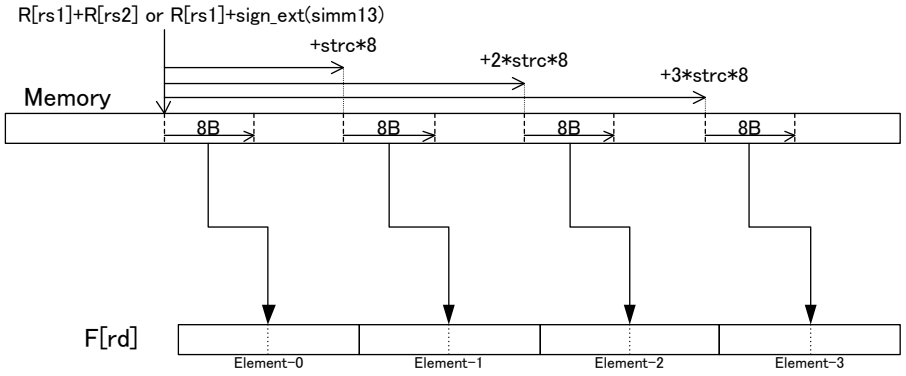
LDFST,LDDFST, LDFSTUW, LDFSTIB, LDFSTSW 命令ではエンディアン変換は、要素ごとに個別に行われる。各要素が異なるページに属し、それぞれのページのエンディアンが異なる場合も、エンディアンが異なるページの要素のみエンディアン変換が行われる。

DFST,LDDFST, LDFSTUW, LDFSTIB, LDFSTSW 命令はどの要素でもウォッチポイントを検出する。

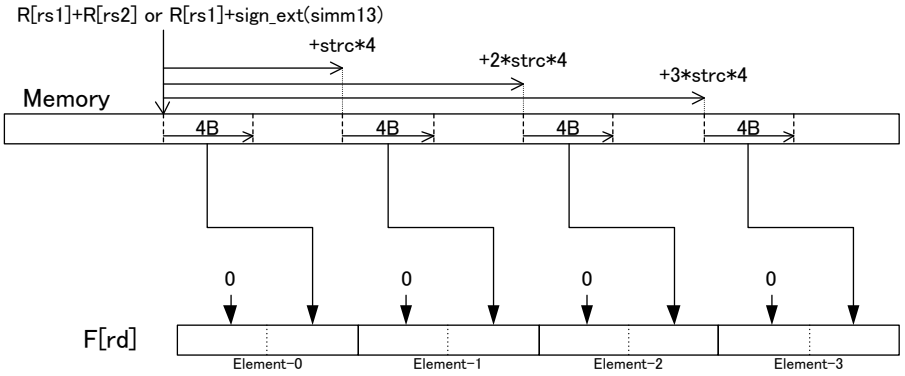
LDFST



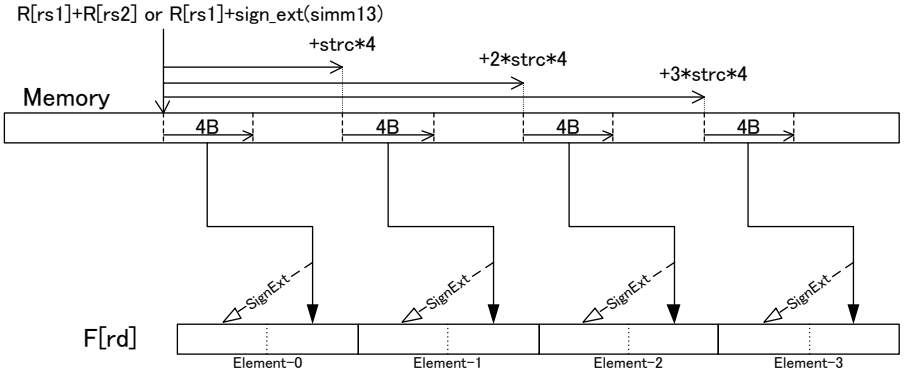
LDDFST



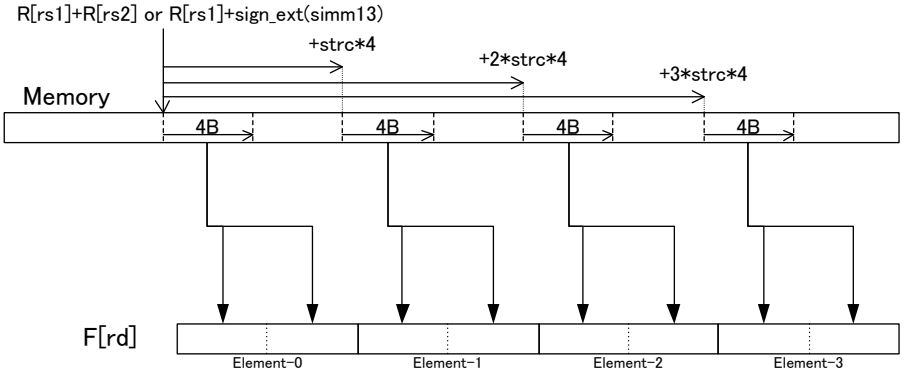
LDFSTUW



LDFSTSW



LDFSTIB



仮想的な言語を使用した動作記述（SIMD）

```
BIT_32 Load32 (ADDRESS){
  TMP<31:0> ← MEM[ADDRESS,4]
  RETURN TMP
}
```

```
BIT_64 Load64 (ADDRESS){
  TMP<63:0> ← MEM[ADDRESS,8]
  RETURN TMP
}

BIT_64 Sign_Ext(DATA32){
  TMP<31:0> ← DATA32
  TMP<63:32> ← DATA32<31> ? 32'hFFFFFFFF : 32'h00000000
  RETURN TMP
}

IF(XASR.simd_mode==0)
  SIMD_WIDTH ← 2
ELSEIF(XASR.simd_mode==1)
  SIMD_WIDTH ← 4
FPR_WIDTH ← 4

STRC ← XAR.ldst_SIMD_op # XAR.ldst_SIMD_op > 1
```

##LDFST

```
ADDR_TMP ← IW.i ? R[rs1]+IW.simm13 : R[rs1]+R[rs2]
FOR (ELEMENT ← 0; ELEMENT<SIMD_WIDTH; ELEMENT++){
  ADDR ← ADDR_TMP + (ELEMENT * STRC * 4)
  F[rd][ELEMENT] ← Load32(ADDR)
  F[rd][ELEMENT] ← 32'h00000000
}
FOR (ELEMENT ← SIMD_WIDTH; ELEMENT<FPR_WIDTH; ELEMENT++)
  F[rd][ELEMENT]<63:0> ← UNDEFINED
```

##LDDFST

```
ADDR_TMP ← IW.i ? R[rs1]+IW.simm13 : R[rs1]+R[rs2]
FOR (ELEMENT ← 0; ELEMENT<SIMD_WIDTH; ELEMENT++){
  ADDR ← ADDR_TMP + (ELEMENT * STRC * 8)
  F[rd][ELEMENT]<63:0> ← Load64(ADDR)
}
FOR (ELEMENT ← SIMD_WIDTH; ELEMENT<FPR_WIDTH; ELEMENT++)
  F[rd][ELEMENT]<63:0> ← UNDEFINED
```

##LDFSTUW

```
ADDR_TMP ← IW.i ? R[rs1]+IW.simm13 : R[rs1]+R[rs2]
FOR (ELEMENT ← 0; ELEMENT<SIMD_WIDTH; ELEMENT++){
  ADDR ← ADDR_TMP + (ELEMENT * STRC * 4)
  F[rd][ELEMENT]<63:32> ← 32'h00000000
  F[rd][ELEMENT]<31:0> ← Load32(ADDR)
}
FOR (ELEMENT ← SIMD_WIDTH; ELEMENT<FPR_WIDTH; ELEMENT++)
  F[rd][ELEMENT]<63:0> ← UNDEFINED
```

##LDFSTIB

```
ADDR_TMP ← IW.i ? R[rs1]+IW.simm13 : R[rs1]+R[rs2]
FOR (ELEMENT ← 0; ELEMENT<SIMD_WIDTH; ELEMENT++){
  ADDR ← ADDR_TMP + (ELEMENT * STRC * 4)
  DATA_TMP ← Load32(ADDR)
  F[rd][ELEMENT]<63:32> ← DATA_TMP
  F[rd][ELEMENT]<31:0> ← DATA_TMP
}
FOR (ELEMENT ← SIMD_WIDTH; ELEMENT<FPR_WIDTH; ELEMENT++)
  F[rd][ELEMENT]<63:0> ← UNDEFINED
```

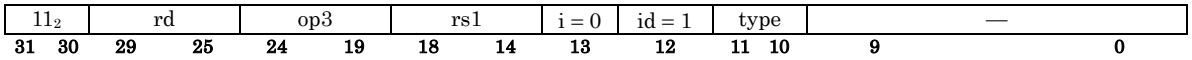
##LDFSTSW

```
ADDR_TMP ← IW.i ? R[rs1]+IW.simm13 : R[rs1]+R[rs2]
FOR (ELEMENT ← 0; ELEMENT<SIMD_WIDTH; ELEMENT++){
  ADDR ← ADDR_TMP + (ELEMENT * STRC * 4)
  TMP<31:0> ← Load32(ADDR)
  F[rd][ELEMENT]<63:0> ← Sign_Ext(TMP<31:0>)
}
FOR (ELEMENT ← SIMD_WIDTH; ELEMENT<FPR_WIDTH; ELEMENT++)
  F[rd][ELEMENT]<63:0> ← UNDEFINED
```

例外	対象命令	検出条件
illegal_instruction	すべて	*reserved* が 0 でないとき。
fp_disabled	すべて	PSTATE.pef = 0 または FPRS.fef = 0
illegal_action	LDFST, LDFSTUW, LDFSTIB, LDFSTSW	下記のいずれかが成立している場合 • i = 1 かつ XAR.urs2<0> ≠ 0
	LDDFST	下記のいずれかが成立している場合 • i = 0 かつ XAR.urs2<2:1> ≠ 00_2 • i = 1 かつ XAR.urs2 ≠ 0
mem_address_not_aligned	LDFST, LDFSTUW, LDFSTSW, LDFSTIB	4 バイト境界ではないアドレスにアクセスしたとき
	LDDFST	8 バイト境界ではないアドレスにアクセスしたとき。
VA_watchpoint	すべて	
DAE_privilege_violation	すべて	
DAE_nc_page	すべて	ノンキャッシャブル空間にアクセスしたとき。
DAE_nfo_page	すべて	

7.18. Indirect Load Floating-Point

命令	op3	rs1, rd[xv]	type	操作	HPC-ACE2 Regs	HPC-ACE2 SIMD	アセンブリ言語表記
LDFID	$10\ 0000_2$	0 – 510	00_2	要素ごとにメモリから倍精度浮動小数点レジスタへの単精度浮動小数点データの読み出し	※	✓	ldid [*freg*$_{rs1}$], *freg*$_{rd}$
LDDFID	$10\ 0011_2$	0 – 510	00_2	要素ごとにメモリから倍精度浮動小数点レジスタへ読み出し	※	✓	lddid [*freg*$_{rs1}$], *freg*$_{rd}$
LDFIDUW	$10\ 0000_2$	0 – 510	01_2	要素ごとにメモリから倍精度浮動小数点レジスタへの符号なし整数の読み出し	※	✓	ldiduw [*freg*$_{rs1}$], *freg*$_{rd}$
LDFIDSW	$10\ 0000_2$	0 – 510	11_2	要素ごとにメモリから倍精度浮動小数点レジスタへの符号付き整数の読み出し	※	✓	ldidsw [*freg*$_{rs1}$], *freg*$_{rd}$



Non-SIMD 動作　Indirect Load Floating-Point 命令は XAR.v = 1 のときのみ使用可能である。XAR.v = 0 で実行される場合、*illegal_action* 例外を検出する。

LDFID 命令は、メモリの 4 バイト境界にある 4 バイト領域の内容を Fd[rd] の上位 4 バイトに読み出す。

LDDFID 命令は、メモリの 4 バイト境界にある 8 バイト領域の内容を Fd[rd]に読み出す。

LDFIDUW 命令は、メモリの 4 バイト境界にある 4 バイト領域の内容を符号なしワードとして Fd[rd]に読み出す。

LDFIDSW 命令は、メモリの 4 バイト境界にある 4 バイト領域の内容を符号拡張し 8 バイト符号付き整数として Fd[rd]に読み出す。

これらの浮動小数点ロード命令は、暗黙の ASI を使いメモリにアクセスする。読み出すアドレスは、“Fd[rs1]” で指定される。LDFID, LDDFID, LDFIDUW および LDFIDSW 命令は、4 バイト境界にないアドレスにアクセスすると *mem_address_not_aligned* 例外を発生する。

> **Comment**　倍精度浮動小数点レジスタの上位 4 バイトと下位 4 バイトを埋める型修飾子 “ib”は Indirect Load 命令では使用できない。

LDDFID 命令は、アクセスアドレスは 4 バイト境界にあればよい。しかし、4 バイト境界だが 8 バイト境界にないアドレスにアクセスすると *LDDF_mem_address_not_aligned* 例外を発生する。この例外については、トラップハンドラは LDDFID 命令をエミュレートする必要がある。

SIMD 動作　SPARC64™ XIfx ではこれらの命令は SIMD 拡張される。

メモリのデータを要素ごとに計算されるアドレスから読み出す。暗黙の ASI を使いメモリにアクセスし、読み出すアドレスは、“Fd[rs1][Element]” で指定される。

LDFID 命令は有効な要素ごとにそれぞれ 4 バイト境界の 4 バイト領域の内容を Fd[rd]に読み出す。

LDDFID 命令は有効な要素ごとにそれぞれ 8 バイト境界の 8 バイト領域の内容を Fd[rd]に読み出す。

LDFIDUW 命令は有効な要素ごとにそれぞれ 4 バイト境界の 4 バイト領域の内容を符号なし整数として Fd[rd]に読み出す。

LDFIDSW 命令は有効な要素ごとにそれぞれ 4 バイト境界の 4 バイト領域の内容を符号拡張し 8 バイト符号付き整数として Fd[rd]に読み出す。

有効でない要素の Fd[rd]は不定値が格納される。

アクセス境界違反には *mem_address_not_aligned* 例外が発生する。

SIMD LDFID, LDDFID, LDFIDUW, LDFIDSW 命令ではキャッシャブル領域からのみロードできる。ノンキャッシャブル領域に対して SIMD ロードを実行しようとすると、*DAE_nc_page* 例外を検出する。

メモリアクセスのセマンティクスは通常のロード命令と同じく、TSO を遵守する。SIMD LDFID, LDDFID, LDFIDUW, LDFIDSW 命令では 1 命令で複数の要素を同時にロードするが、これらの要素間における TSO は保証されない。他の SIMD ロード命令と異なることに注意が必要である。

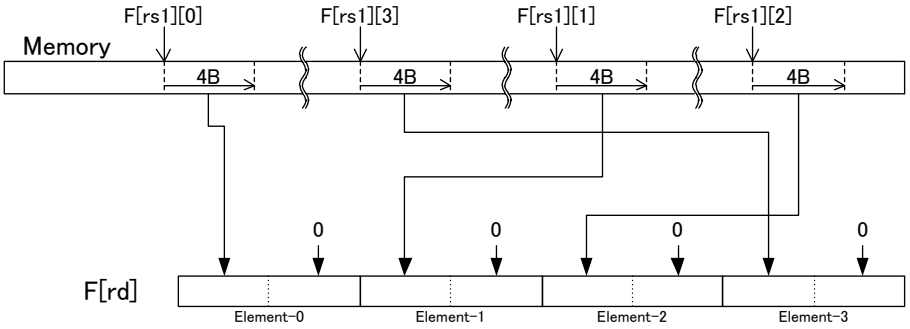
> **Programming Note**　SPARC64™ XIfx ではメモリアクセスのセマンティクスは通常のロード命令と同じく、TSO を遵守する。SIMD LDFID, LDDFID, LDFIDUW, LDFIDSW 命令では 1 命令で複数の要素を同時にロードするが、要素間でも TSO が遵守される。

SIMD LDFID, LDDFID, LDFIDUW, LDFIDSW 命令ではエンディアン変換は、要素ごとに個別に行われる。各要素が異なるページに属し、それぞれのページのエンディアンが異なる場合も、エンディアンが異なるページの要素のみエンディアン変換が行われる。

SIMD LDFID, LDDFID, LDFIDUW, LDFIDSW 命令はどの要素でもウォッチポイントを検出する。

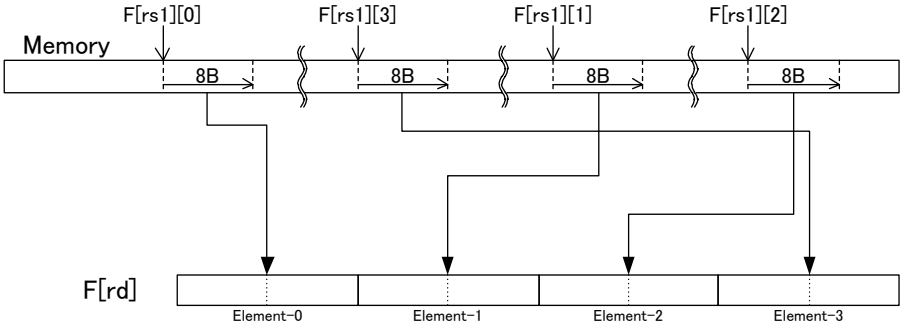
> **Programming Note**　Indirect Load Floating-Point 命令は Stride Load Floating-Point 命令を包含しているが、Stride Load Floating-Point 命令を使用する方が性能面で有利である。

LDFID



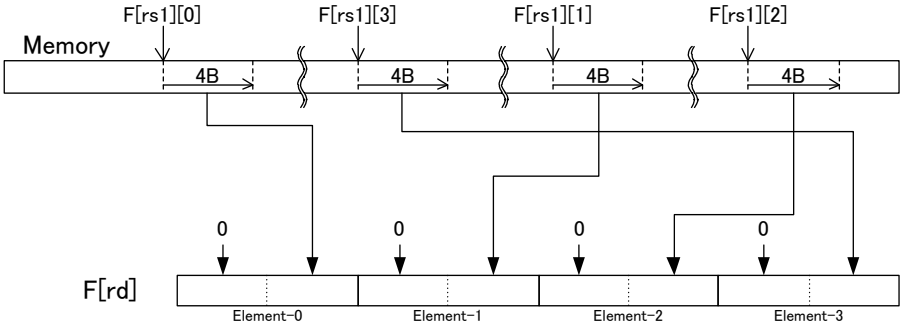
上記例では F[rs1][0] < F[rs1][3] < F[rs1][1] < F[rs1][2]

LDDFID



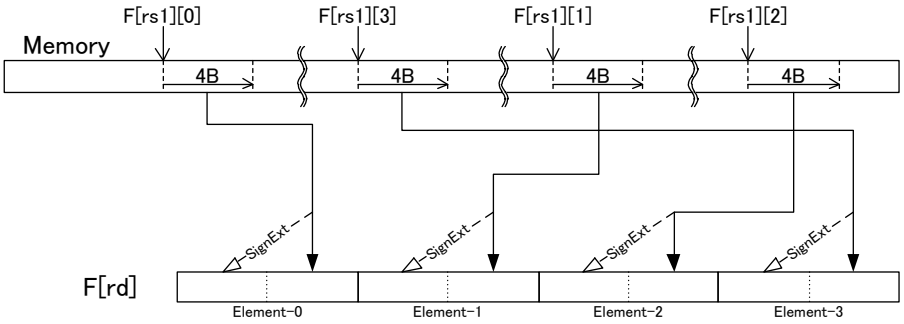
上記例では F[rs1][0] < F[rs1][3] < F[rs1][1] < F[rs1][2]

LDFIDUW



上記例では F[rs1][0] < F[rs1][3] < F[rs1][1] < F[rs1][2]

LDFIDSW



上記例では F[rs1][0] < F[rs1][3] < F[rs1][1] < F[rs1][2]

仮想的な言語を使用した動作記述（SIMD）

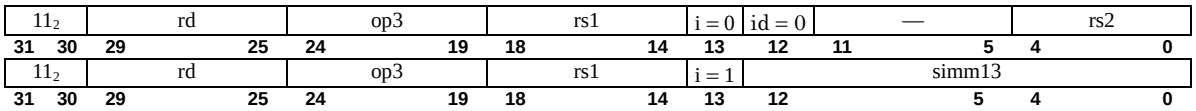
```
BIT_32 Load32 (ADDRESS){
  TMP<31:0> ← MEM[ADDRESS,4]
  RETURN TMP
}

BIT_64 Load64 (ADDRESS){
  TMP<63:0> ← MEM[ADDRESS,8]
  RETURN TMP
```

```
}

BIT_64 Sign_Ext(DATA32){
  TMP<31:0> ← DATA32
  TMP<63:32> ← DATA32<31> ? 32'hFFFFFFFF : 32'h00000000
  RETURN TMP
}

IF(XASR.simd_mode==0)
  SIMD_WIDTH ← 2
ELSEIF(XASR.simd_mode==1)
  SIMD_WIDTH ← 4
FPR_WIDTH ← 4


##LDFID
FOR (ELEMENT ← 0; ELEMENT<SIMD_WIDTH; ELEMENT++){
  ADDR ← F[rs1][ELEMENT]
  F[rd][ELEMENT]<63:32> ← Load32(ADDR)
  F[rd][ELEMENT]<31:0> ← 32'h00000000
}
FOR (ELEMENT ← SIMD_WIDTH; ELEMENT<FPR_WIDTH; ELEMENT++)
  F[rd][ELEMENT]<63:0> ← UNDEFINED

##LDDFID
FOR (ELEMENT ← 0; ELEMENT<SIMD_WIDTH; ELEMENT++){
  ADDR ← F[rs1][ELEMENT]
  F[rd][ELEMENT]<63:0> ← Load64(ADDR)
}
FOR (ELEMENT ← SIMD_WIDTH; ELEMENT<FPR_WIDTH; ELEMENT++)
  F[rd][ELEMENT]<63:0> ← UNDEFINED

##LDFIDUW
FOR (ELEMENT ← 0; ELEMENT<SIMD_WIDTH; ELEMENT++){
  ADDR ← F[rs1][ELEMENT]
  F[rd][ELEMENT]<63:32> ← 32'h00000000
  F[rd][ELEMENT]<31:0> ← Load32(ADDR)
}
FOR (ELEMENT ← SIMD_WIDTH; ELEMENT<FPR_WIDTH; ELEMENT++)
  F[rd][ELEMENT]<63:0> ← UNDEFINED

##LDFIDSW
FOR (ELEMENT ← 0; ELEMENT<SIMD_WIDTH; ELEMENT++){
  ADDR ← F[rs1][ELEMENT]
  TMP<31:0> ← Load32(ADDR)
  F[rd][ELEMENT]<63:0> ← Sign_Ext(TMP<31:0>)
}
FOR (ELEMENT ← SIMD_WIDTH; ELEMENT<FPR_WIDTH; ELEMENT++)
  F[rd][ELEMENT]<63:0> ← UNDEFINED
```

例外	対象命令	検出条件
illegal_instruction	LDFID, LDFIDUW, LDFIDSW	*reserved* が 0 でないとき type = 2 のとき
	LDDFID	*reserved* が 0 でないとき type ≠ 0 のとき
fp_disabled	すべて	PSTATE.pef = 0 または FPRS.fef = 0
illegal_action	すべて	XAR.v = 0 のとき または、XAR.v = 1 かつ、下記のいずれかが成立している場合 • XAR.simd = 1 かつ XAR.urs1<2> ≠ 0 • XAR.simd = 1 かつ XAR.urd<2> ≠ 0 • XAR.urs2 ≠ 0
LDDF_mem_address_not_aligned	LDDFID	XAR.v = 0 または XAR.simd = 0 で、4 バイト境界だが 8 バイト境界ではないアドレスにアクセスしたとき。
mem_address_not_aligned	LDFID, LDFIDUW, LDFIDSW,	4 バイト境界ではないアドレスにアクセスしたとき
	LDDFID	下記のいずれかが成立している場合 • XAR.v = 0 または XAR.simd = 0 で、4 バイト境界ではないアドレスにアクセスしたとき。 • XAR.v = 1 かつ XAR.simd = 0 で、8 バイト境界ではないアドレスにアクセスしたとき。
VA_watchpoint	すべて	
DAE_privilege_violation	すべて	
DAE_nc_page	すべて	XAR.v = 1 かつ XAR.simd = 1 で、ノンキャッシャブル空間にアクセスしたとき。
DAE_nfo_page	すべて	

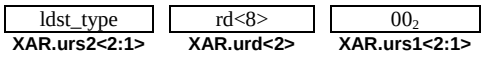
7.19. Store Floating-Point

命令	op3	rd[xv]	urs2 <2:1>	操作	HPC-ACE2		アセンブリ言語表記
					Regs	SIMD	
STF	$10\ 0100_2$	0 – 31	—	単精度浮動小数点レジスタのメモリ書き込み (XAR.v = 0)			st *freg*$_{rd}$, [*address*]
STF	$10\ 0100_2$	0 – 510	00_2	倍精度浮動小数点レジスタ上の単精度データのメモリ書き込み (XAR.v = 1)	✓	✓	st *freg*$_{rd}$, [*address*]
STDF	$10\ 0111_2$	0 – 510	00_2	倍精度浮動小数点レジスタのメモリ書き込み	✓	✓	std *freg*$_{rd}$, [*address*]
STQF	$10\ 0110_2$	0 – 510	00_2	4 倍精度浮動小数点レジスタのメモリ書き込み	✓		stq *freg*$_{rd}$, [*address*]
STFUW	$10\ 0100_2$	0 – 510	01_2	倍精度浮動小数点レジスタ上の符号なしワードをメモリへ書き込み	※	✓	stuw *freg*$_{rd}$, [*address*]
STDFDS	$10\ 0111_2$	0 – 510	01_2	倍精度浮動小数点レジスタの単精度データ×2 のメモリ書き込み	※	✓	stdds *freg*$_{rd}$, [*address*]

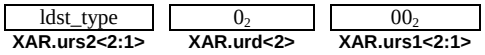


XAR 拡張フィールド (XAR.v = 1 の時のみ有効)

・XAR.simd = 0 のとき



・XAR.simd = 1 のとき



Non-SIMD 動作

STF 命令は、F[rd]の内容を 4 バイト境界の 4 バイト領域に書き込む。XAR.v = 0 のときは単精度浮動小数点レジスタの内容を書き込み、XAR.v = 1 のときは Fd[rd]の上位 4 バイトを書き込む。

STDF 命令は、Fd[rd]の内容を 4 バイト境界の 8 バイト領域に書き込む。

STQF 命令は、Fq[rd]の内容を 4 バイト境界の 16 バイト領域に書き込む。

STFUW 命令は、Fd[rd]の下位 4 バイトの内容を 4 バイト境界の 4 バイト領域に書き込む。

STDFDS 命令は、Fd[rd]の内容を 4 バイト×2 として 4 バイト境界の 8 バイト領域に書き込む。

これらの浮動小数点ストア命令は、暗黙の ASI を使いメモリにアクセスする。書き込みアドレスは、i = 0 のときは “R[rs1] + R[rs2]” で、i = 1 のときは “R[rs1] + sign_ext(simm13)” で計算される。

STF, STDF, STDFDS, STQF および STFUW 命令は、4 バイト境界にないアドレスにアクセスすると *mem_address_not_aligned* 例外を発生する。

STDF 命令は、アクセスアドレスは 4 バイト境界にあればよい。しかし、4 バイト境界だが 8 バイト境界にないアドレスにアクセスすると *STDF_mem_address_not_aligned* 例外を発生する。この例外については、トラップハンドラは STDF 命令をエミュレートする必要がある。

> **Programming Note**　SPARC V8 では、倍精度または 4 倍精度データが適切にアラインされている保証がないとき、複数の単精度ストア命令で処理するコードを生成するコンパイラがあった。アラインされていない命令のエミュレーションは高速に行われるべきなので、SPARC V9 では、複数の単精度ストア命令で処理するのは、倍精度または 4 倍精度データが適切にアラインされていないことが確実なときのみにすることを推奨する。

STQF 命令は SPARC V9 で定義された命令であるが、SPARC64™ XIfx では実装されていないので、実行すると *illegal_instruction* 例外を発生する。

> **Programming Note**　STF 命令のアドレス指定（rs1, rs2）に HPC-ACE 拡張整数レジスタ xg[0] – xg[31]を使う場合、ストアデータを保持するレジスタは倍精度レジスタ Fd[rd]となる。これは XAR.v = 1 のときの rd のデコード定義から来る制約であり、rs1, rs2 に拡張レジスタを指定し、rd に SPARC V9 単精度レジスタ（奇数番号レジスタ）を指定する方法はない。
>
> **Programming Note**　STFUW, STDFDS 命令は XAR.v = 1 のときのみ使用可能である。XAR.v = 0 のとき、XAR 拡張フィールドは All 0 として振舞う。

STDFDS 命令ではエンディアン変換は、4 バイトごとに行われる。エンディアンが異なるページに属し、それぞれのページのエンディアンが異なる場合も、エンディアンが異なるページの 4 バイト単位のデータのみエンディアン変換が行われる。

STDFDS 命令はキャッシャブル空間にのみ書き込みができる。ノンキャッシャブル空間に書き込もうとすると、*DAE_nc_page* 例外が発生する。

SIMD 動作

SPARC64™ XIfx では STF, STDF, STFUW および STDFDS 命令は SIMD 拡張される。

STF は 4 バイト境界の 4×有効な要素数バイト領域に対し、低位アドレス側から順に有効な要素の Fd[rd]の上位 4 バイトを書き込む。

STDF は 8 バイト境界の 8×有効な要素数バイト領域に対し、低位アドレス側から順に有効な要素の Fd[rd]を書き込む。

STFUW は 4 バイト境界の 4×有効な要素数バイト領域に対し、低位アドレス側から順に有効な要素の Fd[rd]の下位 4 バイトを書き込む。

STDFDS は 4 バイト境界の 4×2×有効な要素数バイト領域に対し、低位アドレス側から順に有効な要素の Fd[rd]を書き込む。

SIMD STF, STDF, STFUW, STDFDS 命令では、アクセス境界違反には *mem_address_not_aligned* 例外が発生する。

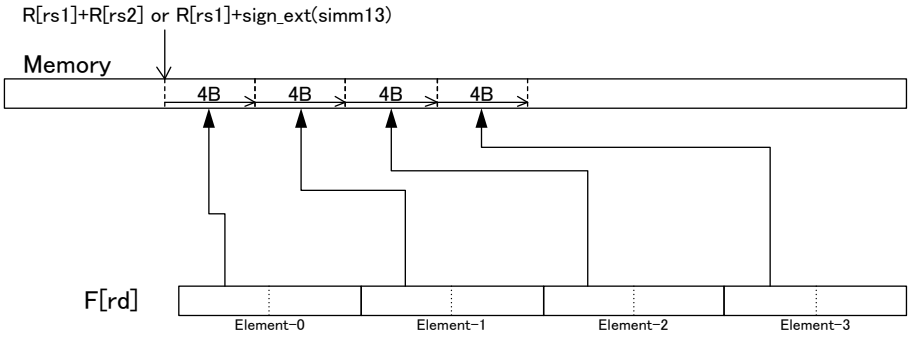
STF, STDF, STFUW および STDFDS 命令はキャッシャブル空間にのみ書き込みができる。ノンキャッシャブル空間に書き込もうとすると、*DAE_nc_page* 例外が発生する。

メモリアクセスのセマンティクスは通常のストア命令と同じく、TSO を遵守する。SIMD STF, STDF, STFUW, STDFDS 命令では 1 命令で複数の要素を同時にストアするが、要素間でも TSO が遵守される。

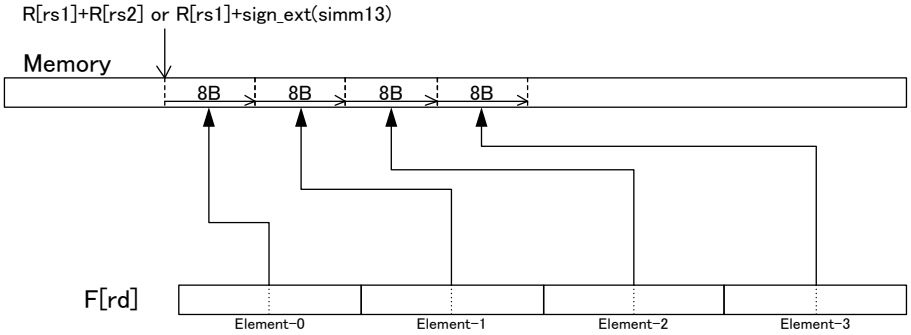
SIMD STF, STDF, STFUW 命令ではエンディアン変換は、要素ごとに個別に行われる。各要素が異なるページに属し、それぞれのページのエンディアンが異なる場合も、エンディアンが異なるページの要素のみエンディアン変換が行われる。STDFDS 命令ではエンディアン変換は、4 バイトごとに行われる。エンディアンが異なるページに属し、それぞれのページのエンディアンが異なる場合も、エンディアンが異なるページの 4 バイト単位のデータのみエンディアン変換が行われる。

SIMD STF, STDF, STFUW, STDFDS 命令はどの要素でもウォッチポイントを検出する。

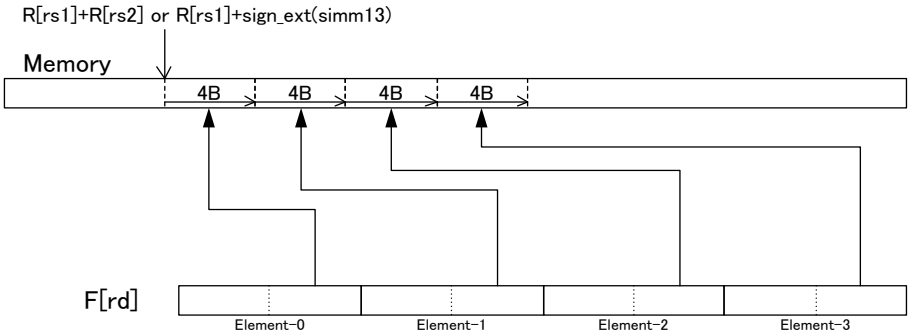
STF (XAR.v = 1)



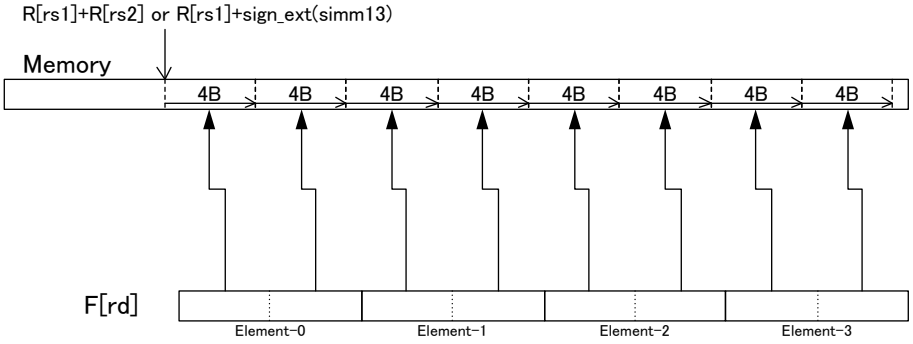
STDF



STFUW



STDFDS



仮想的な言語を使用した動作記述（SIMD）

```
VOID Store32 (ADDRESS,DATA32){
  MEM[ADDRESS,4] ← DATA32
}

VOID Store64 (ADDRESS,DATA64){
  MEM[ADDRESS,8] ← DATA64
}

IF(XASR.simd_mode==0)
  SIMD_WIDTH ← 2
ELSEIF(XASR.simd_mode==1)
  SIMD_WIDTH ← 4
FPR_WIDTH ← 4
```

##STF

```
ADDR_TMP ← IW.i ? R[rs1]+IW.simm13 : R[rs1]+R[rs2]
FOR (ELEMENT ← 0; ELEMENT<SIMD_WIDTH; ELEMENT++){
  ADDR ← ADDR_TMP + (ELEMENT * 4)
  Store32(ADDR,F[rd][ELEMENT]<63:32>)
}
```

##STDF

```
ADDR_TMP ← IW.i ? R[rs1]+IW.simm13 : R[rs1]+R[rs2]
FOR (ELEMENT ← 0; ELEMENT<SIMD_WIDTH; ELEMENT++){
  ADDR ← ADDR_TMP + (ELEMENT * 8)
  Store64(ADDR,F[rd][ELEMENT]<63:0>)
}
```

##STFUW

```
ADDR_TMP ← IW.i ? R[rs1]+IW.simm13 : R[rs1]+R[rs2]
FOR (ELEMENT ← 0; ELEMENT<SIMD_WIDTH; ELEMENT++){
  ADDR ← ADDR_TMP + (ELEMENT * 4)
  Store32(ADDR,F[rd][ELEMENT]<31:0>)
}
```

##STDFDS

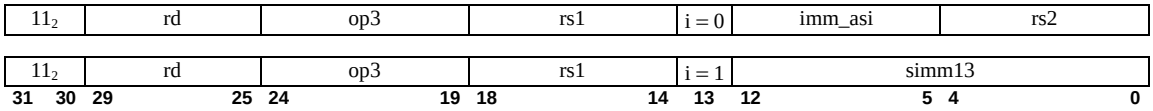
```
ADDR_TMP ← IW.i ? R[rs1]+IW.simm13 : R[rs1]+R[rs2]
FOR (ELEMENT ← 0; ELEMENT<SIMD_WIDTH; ELEMENT++){
  ADDR_H ← ADDR_TMP + (ELEMENT * 8)
  ADDR_L ← ADDR_TMP + (ELEMENT * 8) + 4
  Store32(ADDR_H,F[rd][ELEMENT]<63:32>)
  Store32(ADDR_L,F[rd][ELEMENT]<31:0>)
```

```
}
```

例外	対象命令	検出条件
illegal_instruction	STF, STDF, STFUW, STDFDS	*reserved* が 0 でないとき
	STQF	常に これらの命令に対する例外は、CPU からは *illegal_instruction* のみを検出する。それより優先度の低い例外は、命令エミュレーションのための仕様である
fp_disabled	すべて	PSTATE.pef = 0 または FPRS.fef = 0
illegal_action	STF, STFUW, STDF, STDFDS	XAR.v = 1 かつ下記のいずれかが成立している場合 • XAR.simd = 0 かつ、XAR.urs1<2:1> ≠ 00_2 • i = 1 かつ XAR.urs2<0> ≠ 0 • XAR.urs2<2> ≠ 0 • XAR.simd = 1 かつ、XAR.urd<2> = 0 かつ、XAR.urs1<2:1> = 01_2
	STQF	XAR.v = 1 かつ下記のいずれかが成立している場合 • XAR.urs1<2:1> ≠ 00_2 • i = 0 かつ XAR.urs2<2:1> ≠ 00_2 • i = 1 かつ XAR.urs2 ≠ 0 • XAR.urs2<2:1> ≠ 00_2 • XAR.simd = 1
fp_exception_other (FSR.ftt = *invalid_fp_register*)	STQF	rd<1> ≠ 0
STDF_mem_address_not_aligned	STDF	XAR.v = 0 または XAR.simd = 0 で、4 バイト境界だが 8 バイト境界ではないアドレスに書き込もうとしたとき。
mem_address_not_aligned	STF, STQF, STFUW, STDFDS	4 バイト境界ではないアドレスにアクセスしたとき。
	STDF	以下のいずれかが成立している場合 • XAR.v = 1 かつ XAR.simd = 1 で、8 バイト境界ではないアドレスにアクセスしたとき • XAR.v = 0 または XAR.simd = 0 で、4 バイト境界ではないアドレスにアクセスしたとき
VA_watchpoint	すべて	
DAE_privilege_violation	すべて	
DAE_nc_page	STF, STDF, STFUW	XAR.v = 1 かつ、XAR.simd = 1 で、ノンキャッシャブル空間にアクセスしたとき。
	STDFDS	ノンキャッシャブル空間にアクセスしたとき。
DAE_nfo_page	すべて	

7.20. Store Floating-Point into Alternate Space

命令	op3	rd[xvii]	操作	HPC-ACE2 Regs	HPC-ACE2 SIMD	アセンブリ言語表記	
STFA$^{P_{ASI}}$	$11\ 0100_2$	0 – 31	単精度浮動小数点レジスタの別空間への書き込み（XAR.v = 0）			sta sta	*freg$_{rd}$,* [*address*] ***imm_asi*** *freg$_{rd}$,* [*address*] %asi
STFA$^{P_{ASI}}$	$11\ 0100_2$	0 – 510	単精度浮動小数点レジスタの別空間への書き込み（XAR.v = 1）	✓	✓	sta sta	*freg$_{rd}$,* [*address*] ***imm_asi*** *freg$_{rd}$,* [*address*] %asi
STDFA$^{P_{ASI}}$	$11\ 0111_2$	0 – 510	倍精度浮動小数点レジスタの別空間への書き込み	✓	✓	stda stda	*freg$_{rd}$,* [*address*] ***imm_asi*** *freg$_{rd}$,* [*address*] %asi
STQFA$^{P_{ASI}}$	$11\ 0110_2$	0 – 510	4 倍精度浮動小数点レジスタの別空間への書き込み	✓		stqa stqa	*freg$_{rd}$,* [*address*] ***imm_asi*** *freg$_{rd}$,* [*address*] %asi



Non-SIMD 動作 STFA 命令は F[rd]の内容を、指定された別空間の 4 バイト境界の 4 バイト領域に書き込む。XAR.v = 0 のときは単精度浮動小数点レジスタの内容を書き込み、XAR.v = 1 のときは Fd[rd]の上位 4 バイトを書き込む。

STDFA 命令は Fd[rd]の内容を、指定された別空間の 4 バイト境界の 8 バイト領域に書き込む。

STQFA 命令は Fq[rd]の内容を､指定された別空間の 4 バイト境界の 16 バイト領域に書き込む。

STFA, STDFA および STQFA 命令は、空間識別子（ASI）を必要とする。ASI は、i = 0 のときは imm_asi フィールドで指示され、i = 1 のときは ASI レジスタの値が使われる。書き込みアドレスは、i = 0 のときは “R[rs1] + R[rs2]” で、i = 1 のときは “R[rs1] + sign_ext(simm13)” で計算される。

STFA, STDFA および STQFA 命令を非特権モードで実行する場合、ASI のビット 7 が 0 だと *privileged_action* 例外を発生する。詳細は 10.Address Space Identifiers(321 ページ)参照。

STFA, STDFA および STQFA 命令は、4 バイト境界にないアドレスにアクセスすると *mem_address_not_aligned* 例外を発生する。

STDFA 命令は、アクセスアドレスは 4 バイト境界にあればよい。しかし、4 バイト境界だが 8 バイト境界にないアドレスにアクセスすると *STDF_mem_address_not_aligned* 例外を発生する。この例外については、トラップハンドラは STDF 命令をエミュレートする必要がある。

> **Programming Note** SPARC V8 では、倍精度または 4 倍精度データが適切にアラインされている保証がないとき、複数の単精度ストア命令で処理するコードを生成するコンパイラがあった。アラインされていない命令のエミュレーションは高速に行われるべきなので、SPARC V9 では、複数の単精度ストア命令で処理するのは、倍精度または 4 倍精度データが適切にアラインされていないことが確実なときのみにすることを推奨する。

STQFA 命令は SPARC V9 で定義された命令であるが、SPARC64™ XIfx では実装されていないので、実行すると *illegal_instruction* 例外を発生する。

[xvii] 5.3.4 "Floating-Point Registers Number Encoding” (39 ページ)で定義される浮動小数点レジスタエンコーディングに従う。

Note STFA, STDFA，STQFA 命令は、指定された ASI によらず HPC-ACE2 で拡張された型修飾子及びオペレーション修飾子の対象とはならない。

SIMD 動作

SPARC64™ XIfx では STFA および STDFA 命令は SIMD 拡張される。

STFA は別空間の 4 バイト境界の 4×有効な要素数バイト領域に対し、低位アドレス側から順に有効な要素の Fd[rd]の上位 4 バイトを書き込む。

STDFA は別空間の 8 バイト境界の 8×有効な要素数バイト領域に対し、低位アドレス側から順に有効な要素の Fd[rd]を書き込む。

SIMD STFA, STDFA 命令はアクセス境界違反には *mem_address_not_aligned* 例外が発生する。

SIMD STFA, STDFA 命令はキャッシャブル空間にのみ書き込みができる。ノンキャッシャブル空間に書き込もうとすると、*DAE_nc_page* 例外が発生する。SIMD STFA, STDFA 命令を non-translating ASI に対し使用すると、*DAE_invalid_ASI* 例外が発生する。

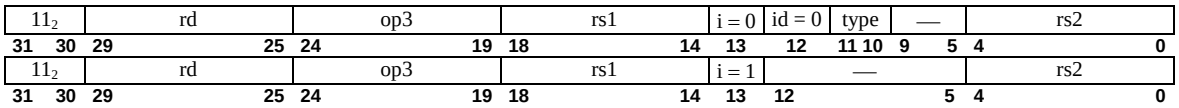
SIMD STFA，STDFA 命令ではエンディアン変換は、要素ごとに個別に行われる。各要素が異なるページに属し、それぞれのページのエンディアンが異なる場合も、エンディアンが異なるページの要素のみエンディアン変換が行われる。

SIMD STFA，STDFA 命令はどの要素でもウォッチポイントを検出する。

例外	対象命令	検出条件
illegal_instruction	STQFA	常に これらの命令に対する例外は、CPU からは *illegal_instruction* のみを検出する。それより優先度の低い例外は、命令エミュレーションのための仕様である
fp_disabled	すべて	PSTATE.pef = 0 または FPRS.fef = 0
illegal_action	STFA, STDFA	XAR.v = 1 かつ、下記のいずれかが成立している場合 • XAR.urs1<2:1> ≠ 00_2 • i = 0 かつ XAR.urs2<2:1> ≠ 00_2 • i = 1 かつ XAR.urs2 ≠ 0 • XAR.simd = 1 かつ XAR.urd<2> ≠ 0
	STQFA	XAR.v = 1 かつ、下記のいずれかが成立している場合 • XAR.urs1<2:1> ≠ 00_2 • i = 0 かつ XAR.urs2<2:1> ≠ 00_2 • i = 1 かつ XAR.urs2 ≠ 0 • XAR.simd = 1
fp_exception_other (FSR.ftt = *invalid_fp_register*)	STQFA	rd<1> ≠ 0
STDF_mem_address_not_aligned	STDFA	XAR.v = 0 または XAR.simd = 0 で、4 バイト境界だが 8 バイト境界ではないアドレスに書き込もうとしたとき。
mem_address_not_aligned	STFA, STQFA	4 バイト境界ではないアドレスにアクセスしたとき。
	STDFA	下記のいずれかが成立している場合。 • XAR.v = 1 かつ XAR.simd = 1 で、8 バイト境界ではないアドレスにアクセスしたとき。 • XAR.v = 0 もしくは XAR.simd = 0 で、4 バイト境界ではないアドレスにアクセスしたとき。
privileged_action	すべて	本文参照
VA_watchpoint	すべて	
DAE_invalid_asi	すべて	本文参照
DAE_privilege_violation	すべて	
DAE_nc_page	すべて	XAR.v = 1 かつ XAR.simd = 1 で、ノンキャッシャブル空間にアクセスしたとき。
DAE_nfo_page	すべて	

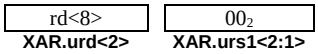
7.21. Store Floating-Point Register on Register Condition

命令	op3	rs2, rd	Type	i	操作	HPC-ACE2		アセンブリ言語表記
						Regs	SIMD	
STFR	$10\ 1100_2$	0 – 31	00_2	0_2, 1_2	条件付単精度浮動小数点レジスタのメモリ書き込み（XAR.v = 0）			`stfr` *freg$_{rd}$, freg$_{rs2}$,* [*reg$_{rs1}$*]
STFR	$10\ 1100_2$	0 – 510	00_2	0_2, 1_2	条件付単精度浮動小数点レジスタのメモリ書き込み（XAR.v = 1）	✓	✓	`stfr` *freg$_{rd}$, freg$_{rs2}$,* [*reg$_{rs1}$*]
STDFR	$10\ 1111_2$	0 – 510	00_2	0_2, 1_2	条件付倍精度浮動小数点レジスタのメモリ書き込み	✓	✓	`stdfr` *freg$_{rd}$, freg$_{rs2}$,* [*reg$_{rs1}$*]
STFRUW	$10\ 1100_2$	0 – 510	01_2	0_2	条件付倍精度浮動小数点レジスタ上の符号無しワードのメモリ書き込み	※	✓	`stfruw` *freg$_{rd}$, freg$_{rs2}$,* [*reg$_{rs1}$*]

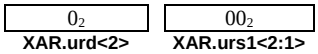


XAR 拡張フィールド

・XAR.simd = 0 のとき



・XAR.simd = 1 のとき



Non-SIMD 動作　STFR 命令は、XAR.v = 0 の場合、F[rs2]<31>=1 のとき、F[rd]の内容を 4 バイト境界の 4 バイト領域に書き込む。XAR.v = 1 の場合、Fd[rs2]<63>=1 のとき、Fd[rd]の上位 4 バイトの内容を 4 バイト境界の 4 バイト領域に書き込む。i = 0 および i = 1 で定義され、どちらを使用した場合も動作に差異はない。

STDFR 命令は、Fd[rs2]<63>=1 のとき、Fd[rd]の内容を 4 バイト境界の 8 バイト領域に書き込む。i = 0 および i = 1 で定義され、どちらを使用した場合も動作に差異はない。

STFRUW 命令は、XAR.v = 0 の場合 *illegal_action* 例外を検出する。XAR.v = 1 の場合、Fd[rs2]<63>=1 のとき、Fd[rd]の下位 4 バイトの内容を 4 バイト境界の 4 バイト領域に書き込む。i = 0 でのみで定義され、i = 1 では *illegal_instruction* 例外を検出する。

これらの浮動小数点ストア命令は、暗黙の ASI を使いメモリにアクセスする。書き込みアドレスは、“R[rs1]”で計算される。

Programming Note　STFR, STDFR に関しては、i = 0 で使用することを推奨する。

STFR 命令は、XAR.v = 0 の場合、F[rs2]<31>=0 のとき、もしくは XAR.v = 1 の場合、Fd[rs2]<63>=0 のとき *illegal_instruction*, *fp_disabled*, *illegal_action* 以外のすべての例外を検出しない。

STDFR 及び STFRUW 命令は、Fd[rs2]<63>=0 のとき、*illegal_instruction*, *fp_disabled*, *illegal_action* 以外の例外をすべて検出しない。

以下の例外は、F[rs2]最上位ビットが 0 であるとき検出されない

4 バイト境界にないアドレスにアクセスすると *mem_address_not_aligned* 例外を発生する。

SIMD 拡張されていない STDFR は、4 バイト境界だが 8 バイト境界にないアドレスにアクセスすると *STDF_mem_address_not_aligned* 例外を発生する。

Programming Note STFR 命令のアドレス指定（rs1, rs2）に HPC-ACE 拡張整数レジスタ xg[0] – xg[31]を使う場合、ストアデータを保持するレジスタは倍精度レジスタ Fd[rd]となる。これは XAR.v = 1 のときの rd のデコード定義から来る制約であり、rs1, rs2 に拡張レジスタを指定し、rd に SPARC V9 単精度レジスタ（奇数番号レジスタ）を指定する方法はない。

Compatibility Note STFR, STDFR 命令は従来 i = 1 でのみ定義されていた。SPARC64™ XIfx での拡張は i = 0 で定義し、STFR, STDFR 命令においては、i = 0 と i = 1 の動作は同一である。

Note SPARC64™ VIIIfx, SPARC64™ IXfx で Non-SIMD の STDFR 命令は 8 バイト境界が明示されていたが、これは誤りであり 4 バイト境界が正しい仕様である。

常に検出する例外	Fs[rs2]もしくは Fd[rs2]の最上位ビットが 1 の時のみ検出する例外
Illegal_instruction *fp_disabled* *illegal_action*	*mem_address_not_aligned* *STDF_mem_address_not_aligned* *VA_watchpoint* *DAE_privilege_violation* *DAE_nfo_page*

SIMD 動作

SPARC64™ XIfx では STFR, STDFR および STFRUW は SIMD 拡張される。

STFR 命令は、4 バイト境界の 4×有効な要素数バイト領域に対し、有効な要素かつ Fd[rs2][Element]<63>=1 を満たすとき、低位アドレス側から順に Fd[rd][Element]の上位 4 バイトを書き込む。もし、Fd[rs2][Element]<63>=0 である場合は、該当領域に対しては書き込みを行わない。

STDFR 命令は、8 バイト境界の 8×有効な要素数バイト領域に対し、有効な要素かつ Fd[rs2][Element]<63>=1 を満たすとき、低位アドレス側から順に Fd[rd]を書き込む。もし、Fd[rs2][Element]<63>=0 である場合は、該当領域に対しては書き込みを行わない。

STFRUW 命令は、4 バイト境界の 4×有効な要素数バイト領域に対し、有効な要素かつ Fd[rs2][Element]<63>=1 を満たすとき、低位アドレス側から順に Fd[rd]の下位 4 バイトを書き込む。

これらの浮動小数点ストア命令は、暗黙の ASI を使いメモリにアクセスする。

SIMD STFR, STDFR, STFRUW 命令は *illegal_instruction*, *fp_disabled*, *illegal_action* の例外は発生条件を満たした場合常に検出する。

illegal_instruction, *fp_disabled*, *illegal_action* を除くすべての例外は有効かつ発生条件を満たした要素において、Fd[rs2][Element]<63>=1 である場合にのみ例外を検出する。すなわち、例外発生条件を満たした場合においても、Fd[rs2][Element]<63>=0 であるとき例外は検出されない。

有効な要素かつ Fd[rs2][Element]<63>=1 の要素でのみ、ウォッチポイントは検出される。

有効な要素かつ Fd[rs2][Element]<63>=1 の要素において、アクセス境界違反がある場合 *mem_address_not_aligned* 例外を検出する。

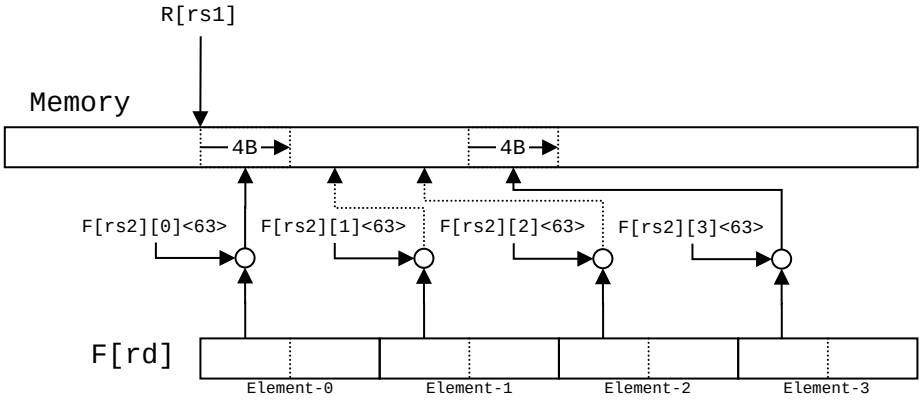
STFR, STDFR および STFRUW 命令はキャッシャブル空間にのみ書き込みができる。ノンキャッシャブル空間に書き込もうとすると、*DAE_nc_page* 例外が発生する。

メモリアクセスのセマンティクスは通常のストア命令と同じく、TSO を遵守する。SIMD STFR, STDFR, STFRUW 命令では 1 命令で複数の要素を同時にストアするが、要素間でも TSO が遵守される。

SIMD STFR, STDFR, STFRUW 命令ではエンディアン変換は、要素ごとに個別に行われる。各要素が異なるページに属し、それぞれのページのエンディアンが異なる場合も、エンディアンが異なるページの要素のみエンディアン変換が行われる。

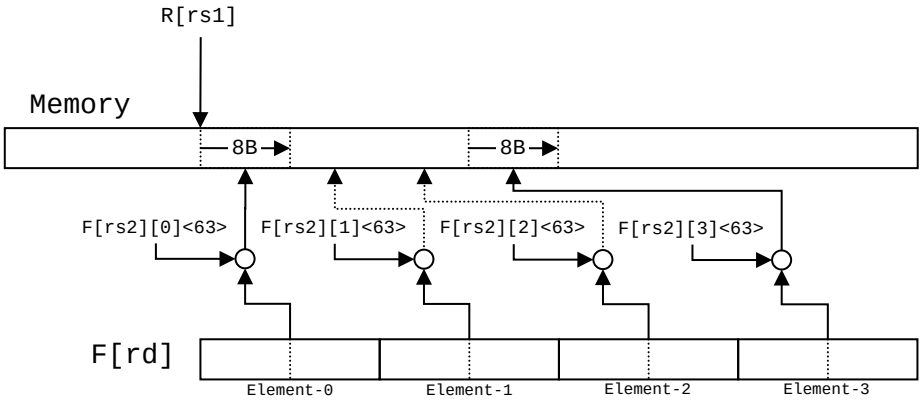
常に検出する例外	各要素の Fd[rs2]の最上位ビットが 1 のとき、該当する要素に対して検出する例外
Illegal_instruction *fp_disabled* *illegal_action*	*mem_address_not_aligned* *VA_watchpoint* *DAE_privilege_violation* *DAE_nc_page* *DAE_nfo_page*

STFR



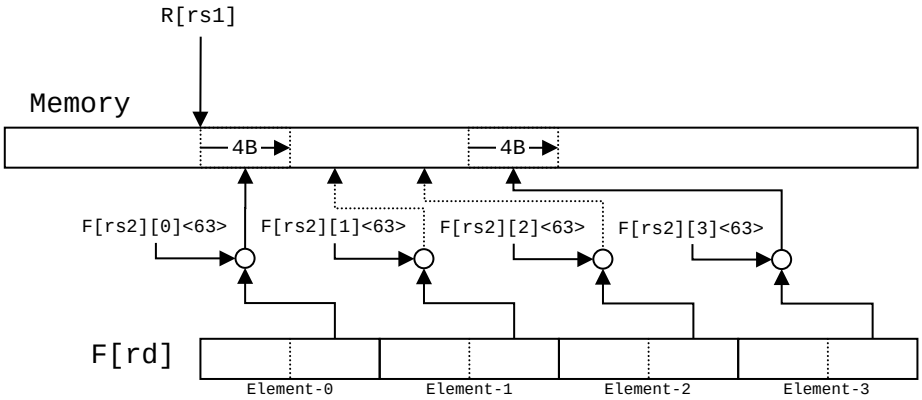
上記例では、F[rs2][0]<63>=1, F[rs2][1]<63>=0, F[rs2][2]<63>=0, F[rs2][3]<63>=1 であり、Element-1, Element-2 の Store は実行されない。

STDFR



上記例では、F[rs2][0]<63>=1, F[rs2][1]<63>=0, F[rs2][2]<63>=0, F[rs2][3]<63>=1 であり、Element-1, Element-2 の Store は実行されない。

STFRUW



上記例では、F[rs2][0]<63>=1, F[rs2][1]<63>=0, F[rs2][2]<63>=0, F[rs2][3]<63>=1 であり、Element-1, Element-2 の Store は実行されない。

仮想的な言語を使用した動作記述（SIMD）

```
VOID Store32 (ADDRESS,DATA32){
  MEM[ADDRESS,4] ← DATA32
}

VOID Store64 (ADDRESS,DATA64){
  MEM[ADDRESS,8] ← DATA64
}

IF(XASR.simd_mode==0)
  SIMD_WIDTH ← 2
ELSEIF(XASR.simd_mode==1)
  SIMD_WIDTH ← 4
FPR_WIDTH ← 4

##STFR
FOR (ELEMENT=0; ELEMENT<SIMD_WIDTH; ELEMENT++){
  IF (F[rs2][ELEMENT]<63>){
```

```
    ADDR ← R[rs1] + (ELEMENT * 4)
    Store32(ADDR,F[rd][ELEMENT]<63:32>)
  }
}
```

##STDFR

```
FOR (ELEMENT=0; ELEMENT<SIMD_WIDTH; ELEMENT++){
  IF (F[rs2][ELEMENT]<63>){
    ADDR ← R[rs1] + (ELEMENT * 8)
    Store64(ADDR,F[rd][ELEMENT]<63:0>)
  }
}
```

##STFRUW

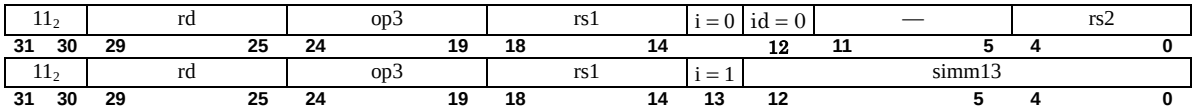
```
FOR (ELEMENT=0; ELEMENT<SIMD_WIDTH; ELEMENT++){
  IF (F[rs2][ELEMENT]<63>){
    ADDR ← R[rs1] + (ELEMENT * 4)
    Store32(ADDR,F[rd][ELEMENT]<31:0>)
  }
}
```

例外	対象命令	検出条件
illegal_instruction	STFR STFRUW	*reserved* が 0 でないとき または i = 0 かつ type<1> = 1 のとき
	STDFR	*reserved* が 0 でないとき または i = 0 かつ type<1:0> ≠ 00_2 のとき
fp_disabled	すべて	PSTATE.pef = 0 または FPRS.fef = 0
illegal_action	STFR STFRUW	XAR.v = 0 かつ type<0> ≠ 0 または、i = 0 かつ XAR.v = 1 かつ下記のいずれかが成立している場合 • XAR.simd = 0 かつ XAR.urs1<2:1> ≠ 00_2 • XAR.simd = 1 かつ XAR.urs2<2> ≠ 0 • XAR.simd = 1 かつ XAR.urd<2> = 0 かつ XAR.urs1<2:1> = 01_2 または、i = 1 かつ XAR.v = 1 かつ下記のいずれかが成立している場合 • XAR.urs1<2:1> ≠ 00_2 • XAR.simd = 1 かつ XAR.urs2<2> ≠ 0 • XAR.simd = 1 かつ XAR.urd<2> ≠ 0
	STDFR	i = 0 かつ XAR.v = 1 かつ下記のいずれかが成立している場合 • XAR.simd = 0 かつ XAR.urs1<2:1> ≠ 00_2 • XAR.simd = 1 かつ XAR.urs2<2> ≠ 0 • XAR.simd = 1 かつ XAR.urd<2> = 0 かつ XAR.urs1<2:1> = 01_2 または、i = 1 かつ XAR.v = 1 かつ下記のいずれかが成立している場合 • XAR.urs1<2:1> ≠ 00_2 • XAR.simd = 1 かつ XAR.urs2<2> ≠ 0 • XAR.simd = 1 かつ XAR.urd<2> ≠ 0
STDF_mem_address_not_aligned	STDFR	XAR.v = 0 または XAR.simd = 0 で 4 バイト境界だが 8 バイト境界ではないアドレスに書き込もうとしたとき。
mem_address_not_aligned	STDFR	以下のいずれかが成立している場合 • XAR.v = 0 または XAR.simd = 0 で、4 バイト境界ではないアドレスにアクセスしたとき。 • XAR.v = 1 かつ XAR.simd = 1 で、8 バイト境界ではないアドレスにアクセスしたとき。
	STFR, STFRUW	4 バイト境界ではないアドレスにアクセスしたとき。
VA_watchpoint	すべて	
DAE_privilege_violation	すべて	
DAE_nc_page	すべて	XAR.v = 1 かつ XAR.simd = 1 で、ノンキャッシャブル空間にアクセスしたとき。

例外	対象命令	検出条件
DAE_nfo_page	すべて	

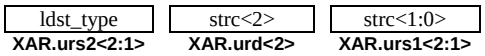
7.22. Stride Store Floating-Point

命令	op3	rd[xv]	urs2 <2:1>	操作	HPC-ACE2		アセンブリ言語表記	
					Regs	SIMD		
STFST	$10\ 0100_2$	0 – 254	00_2	倍精度浮動小数点レジスタ上の単精度浮小数点データのメモリへの一定アドレス間隔書き込み		✓	stst	*freg$_{rd}$,* [*address*] *@strc*
STDFST	$10\ 0111_2$	0 – 254	00_2	倍精度浮動小数点レジスタのメモリへの一定アドレス間隔書き込み		✓	stdst	*freg$_{rd}$,* [*address*] *@strc*
STFSTUW	$10\ 0100_2$	0 – 254	01_2	倍精度浮動小数点レジスタ上の符号なし整数のメモリへの一定アドレス間隔書き込み		✓	ststuw	*freg$_{rd}$,* [*address*] *@strc*



XAR 拡張フィールド

・XAR.simd = 1 のときのみ



動作説明

SPARC64™ XIfx ではこれらの命令は SIMD 拡張でのみ使用される。

メモリ上に一定アドレス間隔で複数個 Fd[rd]のデータを書き込む。アドレスの間隔は XAR.ldst_SIMD_op (2≦XAR.ldst_SIMD_op≦7) で指定する。STFST, STFSTUW では 8, 12, 16, 20, 24, 28 アドレス間隔での書き込みが可能である。STDFST では 16, 24, 32, 40, 48, 56 アドレス間隔での書き込みが可能である。これを超えるものについては、Indirect Store Floating-Point を利用する。暗黙の ASI を使いメモリにアクセスし、書き込むアドレスはそれぞれの要素(Element)で、i = 0 のときは “R[rs1] + R[rs2] + (Element * strc * <byte>)” で、i = 1 のときは“R[rs1] + sign_ext(simm13) + (Element * strc * <byte>)” で計算される。<byte>には STFST, STFSTUW では 4 を使用し、STDFST では 8 を使用する。

STFST 命令は有効な要素の Fd[rd]の上位 4 バイトをメモリ上の 4 バイト境界の 4 バイト領域へ、一定アドレス間隔で書き込む。

STDFST 命令は有効な要素の Fd[rd]の 8 バイトをメモリ上の 8 バイト境界の 8 バイト領域へ、一定アドレス間隔で書き込む。

STFSTUW 命令は有効な要素の Fd[rd]の下位 4 バイトをメモリ上の 4 バイト境界の 4 バイト領域へ、一定アドレス間隔で書き込む。

アクセス境界違反には *mem_address_not_aligned* 例外が発生する。

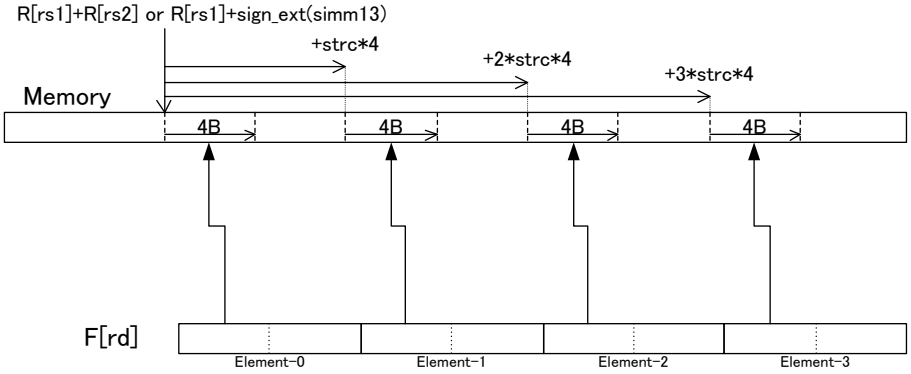
STFST, STDFST および STFSTUW 命令はキャッシャブル空間にのみ書き込みができる。ノンキャッシャブル空間に書き込もうとすると、*DAE_nc_page* 例外が発生する。

メモリアクセスのセマンティクスは通常のストア命令と同じく、TSO を遵守する。STFST, STDFST, STFSTUW 命令では 1 命令で複数の要素を同時にストアするが、要素間でも TSO が遵守される。

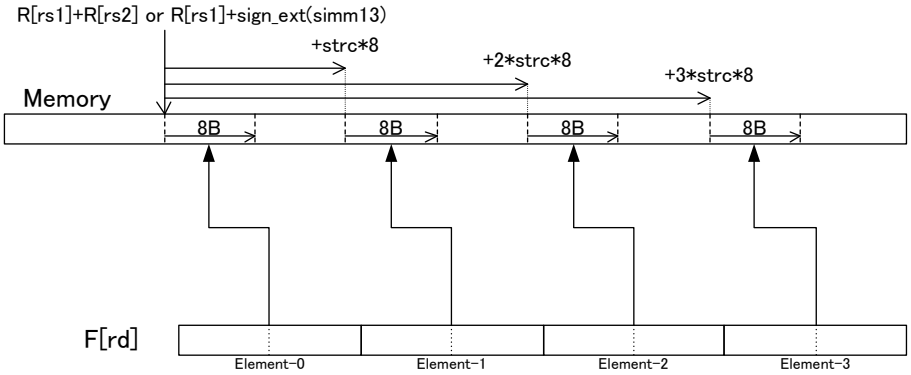
STFST, STDFST, STFSTUW 命令ではエンディアン変換は、要素ごとに個別に行われる。各要素が異なるページに属し、それぞれのページのエンディアンが異なる場合も、エンディアンが異なるページの要素のみエンディアン変換が行われる。

STFST, STDFST, STFSTUW 命令はどの要素でもウォッチポイントを検出する。

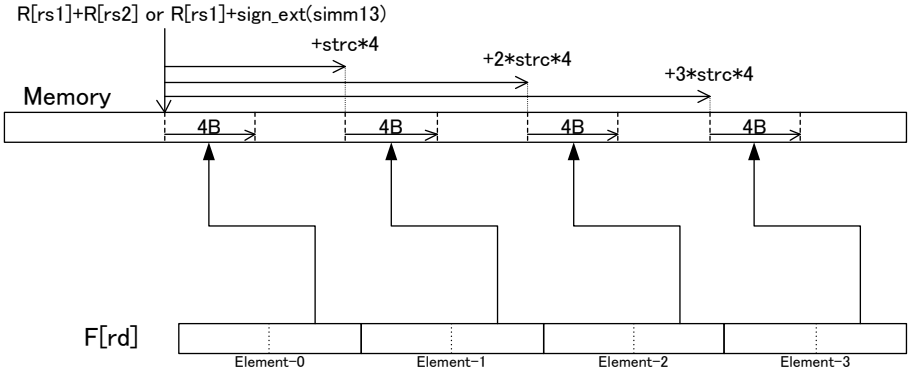
STFST



STDFST



STFSTUW



仮想的な言語を使用した動作記述（SIMD）

```
VOID Store32 (ADDRESS,DATA32){
  MEM[ADDRESS,4] ← DATA32
}

VOID Store64 (ADDRESS,DATA64){
  MEM[ADDRESS,8] ← DATA64
}

IF(XASR.simd_mode==0)
  SIMD_WIDTH ← 2
ELSEIF(XASR.simd_mode==1)
  SIMD_WIDTH ← 4
FPR_WIDTH ← 4

STRC ← XAR.ldst_SIMD_op
```

##STFST

```
ADDR_TMP ← IW.i ? R[rs1]+IW.simm13 : R[rs1]+R[rs2]
FOR (ELEMENT=0; ELEMENT<SIMD_WIDTH; ELEMENT++){
  ADDR ← ADDR_TMP + (ELEMENT * STRC * 4)
  Store32(ADDR,F[rd][ELEMENT]<63:32>)
}
```

##STDFST

```
ADDR_TMP ← IW.i ? R[rs1]+IW.simm13 : R[rs1]+R[rs2]
FOR (ELEMENT=0; ELEMENT<SIMD_WIDTH; ELEMENT++){
  ADDR ← ADDR_TMP + (ELEMENT * STRC * 8)
  Store64(ADDR,F[rd][ELEMENT]<63:0>)
}
```

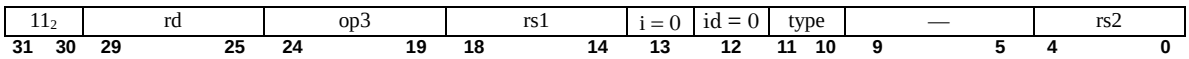
##STFSTUW

```
ADDR_TMP ← IW.i ? R[rs1]+IW.simm13 : R[rs1]+R[rs2]
FOR (ELEMENT=0; ELEMENT<SIMD_WIDTH; ELEMENT++){
  ADDR ← ADDR_TMP + (ELEMENT * STRC * 4)
  Store32(ADDR,F[rd][ELEMENT]<31:0>)
}
```

例外	対象命令	検出条件
illegal_instruction	すべて	*reserved* が 0 でないとき
fp_disabled	すべて	PSTATE.pef = 0 または FPRS.fef = 0
illegal_action	STFST, STFSTUW	下記のいずれかが成立している場合 • i = 1 かつ XAR.urs2<0> ≠ 0 • XAR.urs2<2> ≠ 0 • XAR.urd<2> = 0 かつ XAR.urs1<2:1> = 01_2
	STDFST	下記のいずれかが成立している場合 • i = 1 かつ XAR.urs2<0> ≠ 0 • XAR.urs2<2:1> ≠ 00_2 • XAR.urd<2> = 0 かつ XAR.urs1<2:1> = 01_2
mem_address_not_aligned	STFST, STFSTUW	4 バイト境界ではないアドレスにアクセスしたとき。
	STDFST	8 バイト境界ではないアドレスにアクセスしたとき
VA_watchpoint	すべて	
DAE_privilege_violation	すべて	
DAE_nc_page	すべて	ノンキャッシャブル空間にアクセスしたとき。
DAE_nfo_page	すべて	

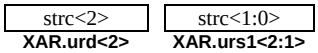
7.23. Stride Store Floating-Point Register on Register Condition

命令	op3	rs2, rd	type	操作	HPC-ACE2		アセンブリ言語表記
					Regs	SIMD	
STFRST	$10\ 1100_2$	0 – 254	00_2	倍精度浮動小数点レジスタ上の単精度浮小数点データを条件付きでメモリの一定アドレス間隔ごとに書き込む		✓	`stfrst` *freg*$_{rd}$, *freg*$_{rs2}$, [*reg*$_{rs1}$] @*strc*
STDFRST	$10\ 1111_2$	0 – 254	00_2	倍精度浮動小数点レジスタを条件付きでメモリの一定アドレス間隔ごとに書き込む		✓	`stdfrst` *freg*$_{rd}$, *freg*$_{rs2}$, [*reg*$_{rs1}$] @*strc*
STFRSTUW	$10\ 1100_2$	0 – 254	01_2	倍精度浮動小数点レジスタ上の整数を条件付きでメモリの一定アドレス間隔ごとに書き込む		✓	`stfrstuw` *freg*$_{rd}$, *freg*$_{rs2}$, [*reg*$_{rs1}$] @strc



XAR 拡張フィールド

・XAR.simd = 1 のときのみ



動作説明

SPARC64™ XIfx ではこれらの命令は SIMD 拡張でのみ使用される。

Fd[rs2]<63>=1 のときメモリ上に一定アドレス間隔で複数個 Fd[rd]のデータを書き込む。アドレスの間隔は XAR.ldst_SIMD_op (2≦XAR.ldst_SIMD_op≦7) で指定する。STFRST, STFRSTUW では 8, 12, 16, 20, 24, 28 アドレス間隔での書き込みが可能である。STDFRST では 16, 24, 32, 40, 48, 56 アドレス間隔での書き込みが可能である。暗黙の ASI を使いメモリにアクセスし、アドレスはそれぞれの要素(Element)で “R[rs1] + (Element * strc * <byte>)” で計算される。<byte>には STFRST，STFRSTUW では 4 を使用し、STDFRST では 8 を使用する。

STFRST 命令は有効な要素かつ Fd[rs2][Element]<63>=1 を満たすとき、Fd[rd]の上位 4 バイトをメモリ上の 4 バイト境界の 4 バイト領域へ、一定アドレス間隔で書き込む。もし、Fd[rs2][Element]<63>=0 である場合は、該当領域に対しては書き込みを行わない。

STDFRST 命令は有効な要素かつ Fd[rs2]<63>=1 のとき、Fd[rd]の 8 バイトをメモリ上の 8 バイト境界の 8 バイト領域へ、一定アドレス間隔で書き込む。もし、Fd[rs2][Element]<63>=0 である場合は、該当領域に対しては書き込みを行わない。

STFRSTUW 命令は有効な要素かつ Fd[rs2]<63>=1 のとき、Fd[rd]の下位 4 バイトをメモリ上の 4 バイト境界の 4 バイト領域へ、一定アドレス間隔で書き込む。もし、Fd[rs2][Element]<63>=0 である場合は、該当領域に対しては書き込みを行わない。

STFRST, STDFRST， STFRSTUW 命令は *illegal_instruction*, *fp_disabled*, *illegal_action* の例外は発生条件を満たした場合常に検出する。

illegal_instruction, *fp_disabled*, *illegal_action* を除くすべての例外は有効かつ発生条件を満たした要素において、Fd[rs2][Element]<63>=1 である場合にのみ例外を検出する。すなわち、例外発生条件を満たした場合においても、Fd[rs2][Element]<63>=0 であるとき例外は検出されない。

有効な要素かつ Fd[rs2][Element]<63>=1 の要素でのみ、ウォッチポイントは検出される。

有効な要素かつ Fd[rs2][Element]<63>=1 の要素において、アクセス境界違反がある場合 *mem_address_not_aligned* 例外を検出する。

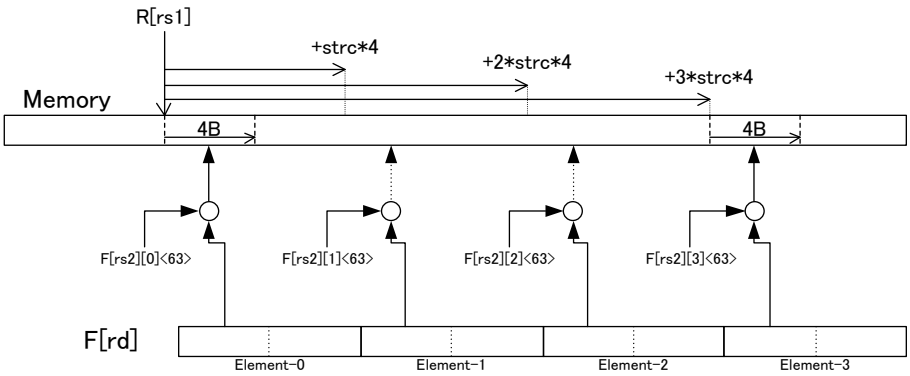
STFRST, STDFRST および STFRSTUW 命令はキャッシャブル空間にのみ書き込みができる。ノンキャッシャブル空間に書き込もうとすると、*DAE_nc_page* 例外が発生する。

メモリアクセスのセマンティクスは通常のストア命令と同じく、TSO を遵守する。SIMD STFRST,STDFRST, STFRSTUW 命令では 1 命令で複数の要素を同時にストアするが、要素間でも TSO が遵守される。

STFRST, STDFRST, STFRSTUW 命令ではエンディアン変換は、要素ごとに個別に行われる。各要素が異なるページに属し、それぞれのページのエンディアンが異なる場合も、エンディアンが異なるページの要素のみエンディアン変換が行われる。

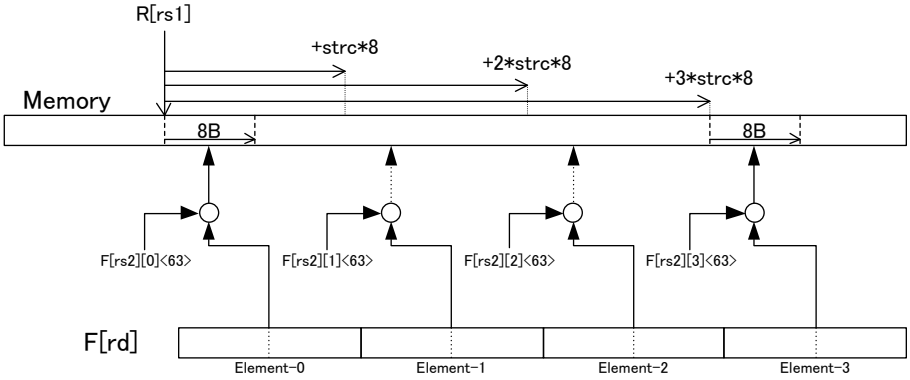
常に検出する例外	各要素の Fd[rs2]の最上位ビットが 1 のとき、該当する要素に対して検出する例外
Illegal_instruction *fp_disabled* *illegal_action*	*mem_address_not_aligned* *VA_watchpoint* *DAE_privilege_violation* *DAE_nc_page* *DAE_nfo_page*

STFRST



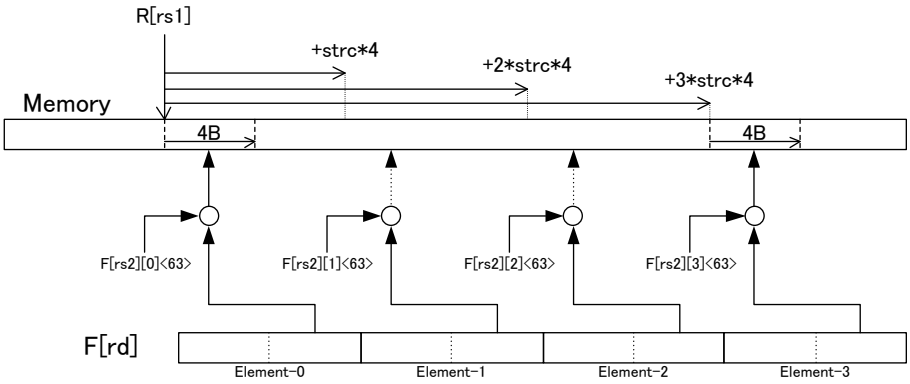
上記例では、F[rs2][0]<63>=1, F[rs2][1]<63>=0, F[rs2][2]<63>=0, F[rs2][3]<63>=1 であり、Element-1, Element-2 の Store は実行されない。

STDFRST



上記例では、F[rs2][0]<63>=1, F[rs2][1]<63>=0, F[rs2][2]<63>=0, F[rs2][3]<63>=1 であり、Element-1, Element-2 の Store は実行されない。

STFRSTUW



上記例では、F[rs2][0]<63>=1, F[rs2][1]<63>=0, F[rs2][2]<63>=0, F[rs2][3]<63>=1 であり、Element-1, Element-2 の Store は実行されない。

仮想的な言語を使用した動作記述（SIMD）

```
VOID Store32 (ADDRESS,DATA32){
  MEM[ADDRESS,4] ← DATA32
}

VOID Store64 (ADDRESS,DATA64){
  MEM[ADDRESS,8] ← DATA64
}

IF(XASR.simd_mode==0)
  SIMD_WIDTH ← 2
ELSEIF(XASR.simd_mode==1)
  SIMD_WIDTH ← 4
FPR_WIDTH ← 4

STRC ← XAR.ldst_SIMD_op

##STFRST
ADDR_TMP ←R[rs1]
```

```
FOR (ELEMENT ← 0; ELEMENT<SIMD_WIDTH; ELEMENT++){
  IF (F[rs2][ELEMENT]<63>){
    ADDR ← ADDR_TMP + (ELEMENT * STRC * 4)
    Store32(ADDR,F[rd][ELEMENT]<63:32>)
  }
}
```

##STDFRST

```
ADDR_TMP ←R[rs1]
FOR (ELEMENT ← 0; ELEMENT<SIMD_WIDTH; ELEMENT++){
  IF (F[rs2][ELEMENT]<63>){
    ADDR ← ADDR_TMP + (ELEMENT * STRC * 8)
    Store64(ADDR,F[rd][ELEMENT]<63:0>)
  }
}
```

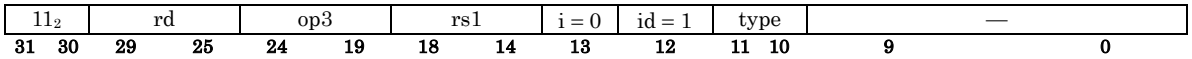
##STFRSTUW

```
ADDR_TMP ←R[rs1]
FOR (ELEMENT ← 0; ELEMENT<SIMD_WIDTH; ELEMENT++){
  IF (F[rs2][ELEMENT]<63>){
    ADDR ← ADDR_TMP + (ELEMENT * STRC * 4)
    Store32(ADDR,F[rd][ELEMENT]<31:0>)
  }
}
```

例外	対象命令	検出条件
illegal_instruction	STFRST STFRSTUW	*reserved* が 0 でないとき または Type<1> = 1 のとき
	STDFRST	*reserved* が 0 でないとき または Type<1:0> ≠ 00_2 のとき
fp_disabled	すべて	PSTATE.pef = 0 または FPRS.fef = 0
illegal_action	すべて	下記のいずれかが成立している場合 • i = 1 • XAR.simd = 1 かつ XAR.urs2<2> ≠ 0 • XAR.simd = 1 かつ XAR.urd<2> = 0 かつ XAR.urs1<2:1> = 01_2
mem_address_not_aligned	STFRST STFRSTUW	4 バイト境界ではないアドレスにアクセスしたとき。
	STDFRST	8 バイト境界ではないアドレスにアクセスしたとき。
VA_watchpoint	すべて	
DAE_privilege_violation	すべて	
DAE_nc_page	すべて	ノンキャッシャブル空間にアクセスしたとき。
DAE_nfo_page	すべて	

7.24. Indirect Store Floating-Point

命令	op3	rs2,rd[xv]	type	操作	HPC-ACE2		アセンブリ言語表記
					Regs	SIMD	
STFID	$10\ 0100_2$	0 – 510	00_2	要素ごとに倍精度浮動小数点レジスタ上の単精度浮小数点データをメモリへ書き込み	※	✓	stid *freg*$_{rd}$, [*freg*$_{rs1}$]
STDFID	$10\ 0111_2$	0 – 510	00_2	要素ごとに倍精度浮動小数点データをメモリへ書き込み	※	✓	stdid *freg*$_{rd}$, [*freg*$_{rs1}$]
STFIDUW	$10\ 0100_2$	0 – 510	01_2	要素ごとに倍精度浮動小数点レジスタ上の符号なしワードをメモリへ書き込み	※	✓	stiduw *freg*$_{rd}$, [*freg*$_{rs1}$]



Non-SIMD 動作　Indirect Store Floating-Point 命令は XAR.v = 1 のときのみ使用可能である。XAR.v = 0 で実行される場合、*illegal_action* 例外を検出する。

STFID 命令は、Fd[rd] の上位 4 バイトの内容を 4 バイト境界にある 4 バイト領域に書き込む。

STDFID 命令は、Fd[rd] の内容をメモリ上の 4 バイト境界にある 8 バイト領域に書き込む。

STFIDUW 命令は、Fd[rd] の下位 4 バイトの内容をメモリ上の 4 バイト境界にある 4 バイト領域に書き込む。

これらの浮動小数点ストア命令は、暗黙の ASI を使いメモリにアクセスする。書き込むアドレスは、“Fd[rs1]” で指定される。

STFID, STDFID および STFIDUW 命令は、4 バイト境界にないアドレスにアクセスすると *mem_address_not_aligned* 例外を発生する。

STDFID 命令は、アクセスアドレスは 4 バイト境界にあればよい。しかし、4 バイト境界だが 8 バイト境界にないアドレスにアクセスすると *STDF_mem_address_not_aligned* 例外を発生する。この例外については、トラップハンドラは STDFID 命令をエミュレートする必要がある。

SIMD 動作　SPARC64™ XIfx ではこれらの命令は SIMD 拡張される。

メモリ上の要素ごとに計算されるアドレスに Fd[rd]のデータを書き込む。暗黙の ASI を使いメモリにアクセスし、書き込むアドレスは、“Fd[rs1][Element]” で指定される。

STFID 命令は有効な要素ごとに Fd[rd]の上位 4 バイトをそれぞれ 4 バイト境界の 4 バイト領域に書き込む。

STDFID 命令は有効な要素ごとに Fd[rd]の 8 バイトをそれぞれ 8 バイト境界の 8 バイト領域に書き込む。

STFIDUW 命令は有効な要素ごとに Fd[rd]の下位 4 バイトをそれぞれ 4 バイト境界の 4 バイト領域に書き込む。

アクセス境界違反には *mem_address_not_aligned* 例外が発生する。

SIMD STFID ,STDFID, STFIDUW 命令ではキャッシャブル領域からのみストアできる。ノンキャッシャブル領域に対して SIMD ストアを実行しようとすると、*DAE_nc_page* 例外を検出する。

メモリアクセスのセマンティクスは通常のストア命令と同じく、TSO を遵守する。SIMD STFID, STDFID, STFIDUW 命令では 1 命令で複数の要素を同時にストアするが、これらの要素間における TSO は保証されない。他の SIMD ストア命令と異なることに注意が必要である。

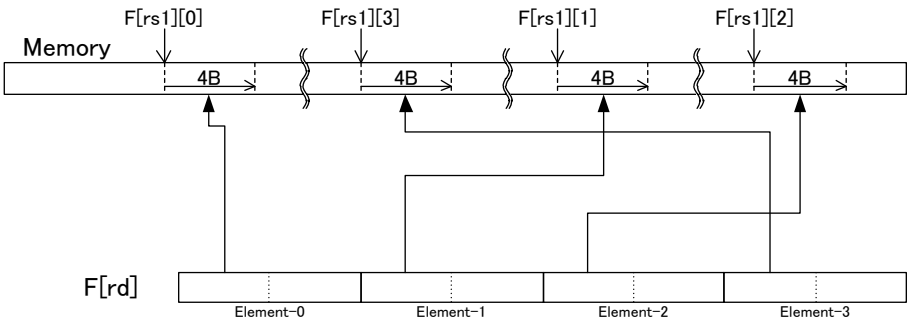
Programming Note　SPARC64™ XIfx ではメモリアクセスのセマンティクスは通常のストア命令と同じく、TSO を遵守する。SIMD STFID, STDFID, STFIDUW 命令では 1 命令で複数の要素を同時にストアするが、要素間でも TSO が遵守される。

SIMD STFID, STDFID, STFIDUW命令ではエンディアン変換は､要素ごとに個別に行われる。各要素が異なるページに属し、それぞれのページのエンディアンが異なる場合も、エンディアンが異なるページの要素のみエンディアン変換が行われる。

SIMD インダイレクトストアはどの要素でもウォッチポイントを検出する。

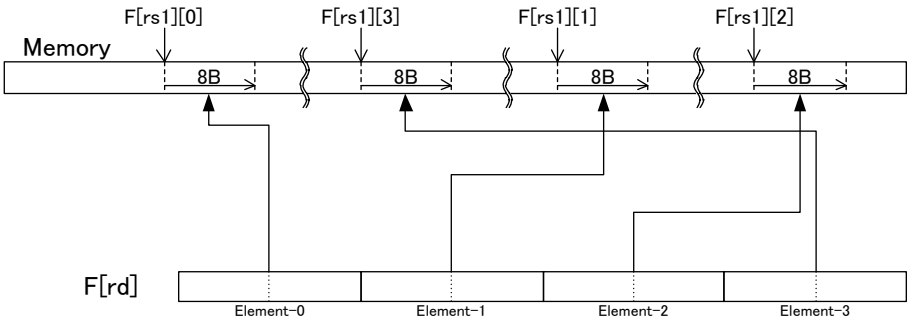
Programming Note　Indirect Store Floating-Point 命令は Stride Store Floating-Point 命令を包含しているが、Stride Store Floating-Point 命令を使用する方が性能面で有利である。

STFID



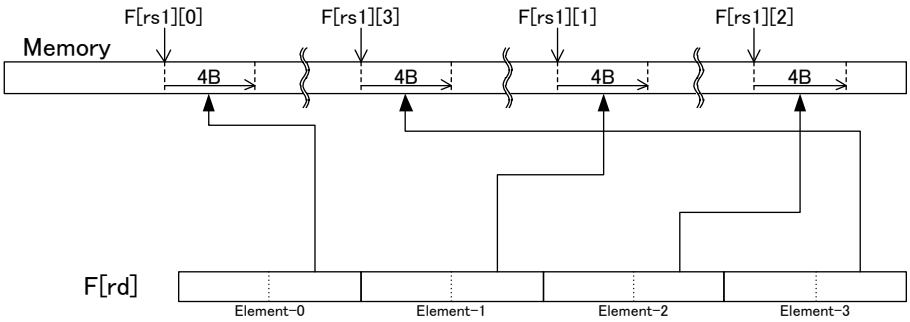
上記例では F[rs1][0] < F[rs1][3] < F[rs1][1] < F[rs1][2]

STDFID



上記例では F[rs1][0] < F[rs1][3] < F[rs1][1] < F[rs1][2]

STFIDUW



上記例では F[rs1][0] < F[rs1][3] < F[rs1][1] < F[rs1][2]

仮想的な言語を使用した動作記述 （SIMD）

```
VOID Store32 (ADDRESS,DATA32){
  MEM[ADDRESS,4] ← DATA32
}

VOID Store64 (ADDRESS,DATA64){
  MEM[ADDRESS,8] ← DATA64
}

IF(XASR.simd_mode==0)
  SIMD_WIDTH ← 2
ELSEIF(XASR.simd_mode==1)
  SIMD_WIDTH ← 4
FPR_WIDTH ← 4

##STFID
FOR (ELEMENT ← 0; ELEMENT<SIMD_WIDTH; ELEMENT++){
  ADDR ← F[rs1][ELEMENT]
  Store32(ADDR,F[rd][ELEMENT]<63:32>)
}

##STDFID
FOR (ELEMENT ← 0; ELEMENT<SIMD_WIDTH; ELEMENT++){
  ADDR ← F[rs1][ELEMENT]
  Store64(ADDR,F[rd][ELEMENT]<63:0>)
}

##STFIDUW
FOR (ELEMENT ← 0; ELEMENT<SIMD_WIDTH; ELEMENT++){
  ADDR ← F[rs1][ELEMENT]
  Store32(ADDR,F[rd][ELEMENT]<31:0>)
}
```

例外	対象命令	検出条件
illegal_instruction	`STFID`, `STFIDUW`	*reserved* が 0 でないとき または type<1> = 1 のとき。
	`STDFID`	*reserved* が 0 でないとき または type<1:0> ≠ 00_2 のとき
fp_disabled	すべて	PSTATE.pef = 0 または FPRS.fef = 0
illegal_action	すべて	XAR.v = 0 のとき または、XAR.v = 1 かつ下記のいずれかが成立している場合 • XAR.urs2 ≠ 0 • XAR.simd = 1 かつ XAR.urd<2> ≠ 0 • XAR.simd = 1 かつ XAR.urs1<2> ≠ 0
STDF_mem_address_not_aligned	`STDFID`	XAR.v = 0 または XAR.simd = 0 で、4 バイト境界だが 8 バイト境界ではないアドレスに書き込もうとしたとき。
mem_address_not_aligned	`STFID`, `STFIDUW`	4 バイト境界ではないアドレスにアクセスしたとき。
	`STDFID`	以下のいずれかが成立している場合 • XAR.v = 0 または XAR.simd = 0 で、4 バイト境界ではないアドレスにアクセスしたとき。 • XAR.v = 1 かつ XAR.simd = 1 で、8 バイト境界ではないアドレスにアクセスしたとき。
VA_watchpoint	すべて	
DAE_privilege_violation	すべて	
DAE_nc_page	すべて	XAR.v = 1 かつ XAR.simd = 1 で、ノンキャッシャブル空間にアクセスしたとき。
DAE_nfo_page	すべて	

7.25. Indirect Store Floating-Point Register on Register Condition

命令	op3	rs1, rs2,rd[xv]	type	操作	HPC-ACE2		アセンブリ言語表記
					Regs	SIMD	
STFRID	$10\ 1100_2$	0 – 510	00_2	条件付き、要素ごとに倍精度浮動小数点レジスタ上の単精度浮小数点データをメモリへ書き込み	※	✓	`stfrid` *freg*$_{rd}$, *freg*$_{rs2}$, [*freg*$_{rs1}$]
STDFRID	$10\ 1111_2$	0 – 510	00_2	条件付き、要素ごとに倍精度浮動小数点データをメモリへ書き込み	※	✓	`stdfrid` *freg*$_{rd}$, *freg*$_{rs2}$, [*freg*$_{rs1}$]
STFRIDUW	$10\ 1100_2$	0 – 510	01_2	条件付き、要素ごとに倍精度浮動小数点レジスタ上の符号なしワードをメモリへ書き込み	※	✓	`stfriduw` *freg*$_{rd}$, *freg*$_{rs2}$, [*freg*$_{rs1}$]

11_2	rd	op3	rs1	i = 0	id = 1	type	—	rs2
31 30	29 25	24 19	18 14	13	12	11 10	9 5	4 0

Non-SIMD 動作 Indirect Store Floating-Point Register on Register Condition 命令は XAR.v = 1 のときのみ使用可能である。XAR.v = 0 で実行される場合、*illegal_action* 例外を検出する。

STFRID 命令は、Fd[rs2]<63>=1 のとき、Fd[rd]の上位 4 バイトの内容を 4 バイト境界の 4 バイト領域に書き込む。

STDFRID 命令は、Fd[rs2]<63>=1 のとき、Fd[rd]の内容を 4 バイト境界の 8 バイト領域に書き込む。

STFRIDUW 命令は、Fd[rs2]<63>=1 のとき、Fd[rd]の下位 4 バイトの内容を 4 バイト境界の 4 バイト領域に書き込む。

これらの浮動小数点ストア命令は、暗黙の ASI を使いメモリにアクセスする。書き込むアドレスは、“Fd[rs1]” で指定される。

STFRID, STDFRID 及び STFRIDUW 命令は、Fd[rs2]<63>=0 のとき、*illegal_instruction*, *fp_disabled*, *illegal_action* 以外の例外をすべて検出しない。

以下の例外は、Fd[rs2]<63>=0 であるとき検出されない

4 バイト境界にないアドレスにアクセスすると *mem_address_not_aligned* 例外を発生する。

STDFRID 命令は、アクセスアドレスは 4 バイト境界にあればよい。しかし、4 バイト境界だが 8 バイト境界にないアドレスにアクセスすると *STDF_mem_address_not_aligned* 例外を発生する。この例外については、トラップハンドラは STDFRID 命令をエミュレートする必要がある。

常に検出する例外	Fs[rs2]もしくは Fd[rs2]の最上位ビットが 1 の時のみ検出する例外
Illegal_instruction *fp_disabled* *illegal_action*	*mem_address_not_aligned* *STDF_mem_address_not_aligned* *VA_watchpoint* *DAE_privilege_violation* *DAE_nfo_page*

SIMD 動作

SPARC64™ XIfx ではこれらの命令は SIMD 拡張される。

メモリ上の要素ごとに計算されるアドレスに Fd[rd]のデータを書き込む。暗黙の ASI を使いメモリにアクセスし、書き込むアドレスは、“ F[rs1][Element]” で指定される。

STFRID 命令は有効な要素かつ Fd[rs2][Element] <63>=1 を満たすとき、Fd[rd]の上位 4 バイトをそれぞれ 4 バイト境界の 4 バイト領域に書き込む。もし、Fd[rs2][Element] <63>=0 である場合は、該当領域に対しては書き込みを行わない。

STDFRID 命令は有効な要素かつ Fd[rs2][Element] <63>=1 を満たすとき、Fd[rd]の 8 バイトをそれぞれ 8 バイト境界の 8 バイト領域に書き込む。もし、Fd[rs2][Element] <63>=0 である場合は、該当領域に対しては書き込みを行わない。

STFRIDUW 命令は有効な要素かつ Fd[rs2][Element] <63>=1 を満たすとき、Fd[rd]の下位 4 バイトをそれぞれ 4 バイト境界の 4 バイト領域に書き込む。もし、Fd[rs2][Element] <63>=0 である場合は、該当領域に対しては書き込みを行わない。

SIMD STFRID, STDFRID, STFRIDUW 命令は *illegal_instruction*, *fp_disabled*, *illegal_action* の例外は発生条件を満たした場合常に検出する。

illegal_instruction, *fp_disabled*, *illegal_action* を除くすべての例外は有効かつ発生条件を満たした要素において、Fd[rs2][Element]<63>=1 である場合にのみ例外を検出する。すなわち、例外発生条件を満たした場合においても、Fd[rs2][Element]<63>=0 であるとき例外は検出されない。

有効な要素かつ Fd[rs2][Element]<63>=1 の要素でのみ、ウォッチポイントは検出される。

有効な要素かつ Fd[rs2][Element]<63>=1 で Fd[rs1][Element]に指定されたアドレスがアクセス境界違反の場合には *mem_address_not_aligned* 例外が発生する。

STFRID, STDFRID および STFRIDUW 命令はキャッシャブル空間にのみ書き込みができる。ノンキャッシャブル空間に書き込もうとすると、*DAE_nc_page* 例外が発生する。

メモリアクセスのセマンティクスは通常のストア命令と同じく、TSO を遵守する。SIMD STFRID, STDFRID, STFRIDUW 命令では 1 命令で複数の要素を同時にストアするが、これらの要素間における TSO は保証されない。他の SIMD ストア命令と異なることに注意が必要である。

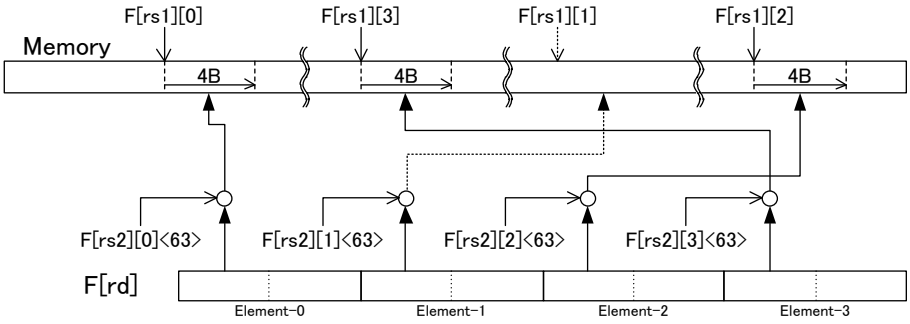
> **Programming Note** SPARC64™ XIfx ではメモリアクセスのセマンティクスは通常のストア命令と同じく、TSO を遵守する。SIMD STFRID,STDFRID, STFRIDUW 命令では 1 命令で複数の要素を同時にストアするが、要素間でも TSO が遵守される。

SIMD STFR, STDFR, STFRUW 命令ではエンディアン変換は、要素ごとに個別に行われる。各要素が異なるページに属し、それぞれのページのエンディアンが異なる場合も、エンディアンが異なるページの要素のみエンディアン変換が行われる。

常に検出する例外	各要素の Fd[rs2]の最上位ビットが 1 のとき、該当する要素に対して検出する例外
Illegal_instruction *fp_disabled* *illegal_action*	*mem_address_not_aligned* *VA_watchpoint* *DAE_privilege_violation* *DAE_nc_page* *DAE_nfo_page*

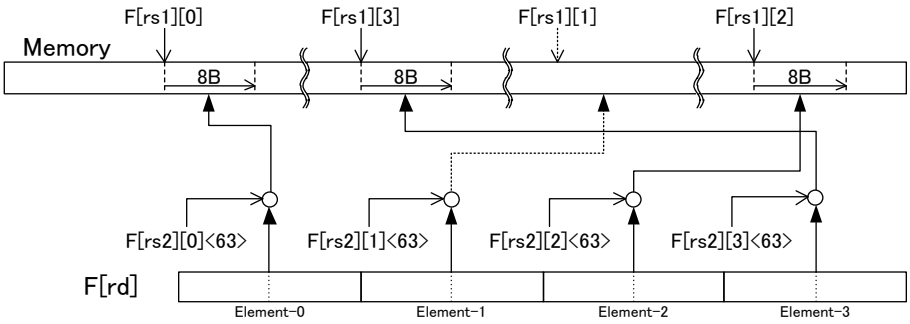
Programming Note Indirect Store Floating-Point on Register Condition 命令は Stride Store Floating-Point on Register Condition 命令を包含しているが、Stride Store Floating-Point on Register Condition 命令を使用する方が性能面で有利である。

STFRID



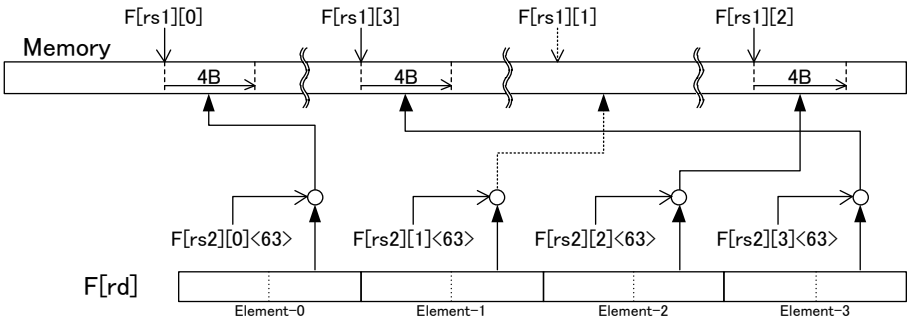
上記例では F[rs1][0] < F[rs1][3] < F[rs1][1] < F[rs1][2] かつ F[rs2][0]<63>=1, F[rs2][1]<63>=0, F[rs2][2]<63>=1, F[rs2][3]<63>=1

STDFRID



上記例では F[rs1][0] < F[rs1][3] < F[rs1][1] < F[rs1][2] かつ F[rs2][0]<63>=1, F[rs2][1]<63>=0, F[rs2][2]<63>=1, F[rs2][3]<63>=1

STFRIDUW



上記例では F[rs1][0] < F[rs1][3] < F[rs1][1] < F[rs1][2] かつ F[rs2][0]<63>=1, F[rs2][1]<63>=0, F[rs2][2]<63>=1, F[rs2][3]<63>=1

仮想的な言語を使用した動作記述（SIMD）

```
VOID Store32 (ADDRESS,DATA32){
  MEM[ADDRESS,4] ← DATA32
}

VOID Store64 (ADDRESS,DATA64){
  MEM[ADDRESS,8] ← DATA64
}

IF(XASR.simd_mode==0)
  SIMD_WIDTH ← 2
ELSEIF(XASR.simd_mode==1)
  SIMD_WIDTH ← 4
FPR_WIDTH ← 4
```

##STFRID

```
FOR (ELEMENT ← 0; ELEMENT<SIMD_WIDTH; ELEMENT++){
  IF (F[rs2][ELEMENT]<63>){
    ADDR ← F[rs1][ELEMENT]
    Store32(ADDR,F[rd][ELEMENT]<63:32>)
  }
}
```

##STDFRID

```
FOR (ELEMENT ← 0; ELEMENT<SIMD_WIDTH; ELEMENT++){
  IF (F[rs2][ELEMENT]<63>){
    ADDR ← F[rs1][ELEMENT]
    Store64(ADDR,F[rd][ELEMENT]<63:0>)
  }
}
```

##STFRIDUW

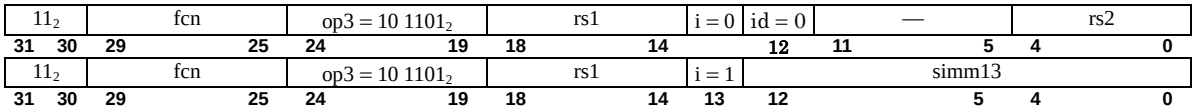
```
FOR (ELEMENT ← 0; ELEMENT<SIMD_WIDTH; ELEMENT++){
  IF (F[rs2][ELEMENT]<63>){
    ADDR ← F[rs1][ELEMENT]
    Store32(ADDR,F[rd][ELEMENT]<31:0>)
  }
}
```

例外	対象命令	検出条件
illegal_instruction	`STFRID`, `STFRIDUW`	*reserved* が 0 でないときまたは type<1> = 1 のとき。
	`STDFRID`	*reserved* が 0 でないときまたは type<1:0> ≠ 00_2 のとき。
fp_disabled	すべて	PSTATE.pef = 0 または FPRS.fef = 0
illegal_action	すべて	XAR.v = 0 のとき または、XAR.v = 1 かつ下記のいずれかが成立している場合 • XAR.simd = 1 かつ XAR.urs2 <2>≠ 0 • XAR.simd = 1 かつ XAR.urd<2> ≠ 0 • XAR.simd = 1 かつ XAR.urs1<2> ≠ 0
STDF_mem_address_not_aligned	`STDFRID`	XAR.v = 0 または XAR.simd = 0 で、4 バイト境界だが 8 バイト境界ではないアドレスに書き込もうとしたとき。
mem_address_not_aligned	`STFRID`, `STFRIDUW`	4 バイト境界ではないアドレスにアクセスしたとき
	`STDFRID`	以下のいずれかが成立している場合 • XAR.v = 0 または XAR.simd = 0 で、4 バイト境界ではないアドレスにアクセスしたとき。 • XAR.v = 1 かつ XAR.simd = 1 で、8 バイト境界ではないアドレスにアクセスしたとき。
VA_watchpoint	すべて	
DAE_privilege_violation	すべて	
DAE_nc_page	すべて	XAR.v = 1 かつ XAR.simd = 1 で、ノンキャッシャブル空間にアクセスしたとき。
DAE_nfo_page	すべて	

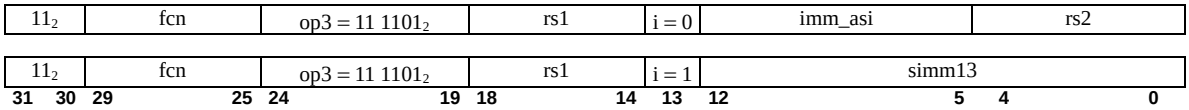
7.26. Prefetch

命令	op3	操作	HPC-ACE2		アセンブリ言語表記	
			Regs	SIMD		
PREFETCH	$10\ 1101_2$	データのプリフェッチ	✓		`prefetch`	[*address*], *prefetch_fcn*
PREFETCHA$^{P_{ASI}}$	$11\ 1101_2$	別空間からのデータのプリフェッチ	✓		`prefetch` `prefetch`	[*regaddr*], ***imm_asi***, *prefetch_fcn* [*reg_plus_imm*] `%asi`, *prefetch_fcn*

PREFETCH



PREFETCHA



動作説明

PREFETCH{|A}命令は、近い将来ソフトウェアがアクセスするデータを、あらかじめキャッシュに載せておき、その後に行われるロードやストアのメモリレイテンシを低減するための命令である。命令実行が成功すると、指定したアドレスを含むメモリブロックの内容がキャッシュにコピーされる。

PREFETCH 命令は、暗黙の ASI を使いアクセスする。メモリアドレスは i = 0 のとき “R[rs1] + R[rs2]” 、i = 1 のとき “R[rs1] + sign_ext(simm13)” である。

PREFETCHA 命令は、空間識別子（ASI）を必要とする。ASI は、i = 0 のときは imm_asi フィールドで指示され、i = 1 のときは ASI レジスタの値が使われる。メモリアドレスは i = 0 のとき “R[rs1] + R[rs2]” 、i = 1 のとき “R[rs1] + sign_ext(simm13)” である。

命令で指定するアドレスは、任意のアドレスが指定できる。指定されたアドレスを含む 1 キャッシュブロック（256バイト）がコピーされる。*mem_address_not_aligned* 例外は発生しない。

指定されたアドレスがノンキャッシャブル空間の場合、または、キャッシャブル空間だが存在しないアドレスの場合、PREFETCH{|A}命令は NOP となる。

PREFETCHA 命令で指定できる ASI は下表に示すとおりである。これ以外の ASI が指定された場合、PREFETCHA 命令は NOP となる。

表 7-9　PREFETCHA 命令に指定できる ASI 一覧

ASI_PRIMARY	ASI_PRIMARY_LITTLE
ASI_SECONDARY	ASI_SECONDARY_LITTLE

プリフェッチ命令は、キャッシュにデータブロックを持ってくること以外の副作用を持たない。

プリフェッチ命令は、ハードウェア内部の状況により実行されないことがある（プリフェッチロスト）。プリフェッチ命令が実行されたかロストしたかを知る方法はない。

7.26.1. プリフェッチ種類

プリフェッチの種類は命令の fcn により選択される。プリフェッチ種類は、CPU に対してヒントを与えるために、コンパイラやプログラマの意図によって使い分けられる。ここが他の命令と異なるところで、他の命令は、その命令の動作がソフトウェアにとって必須なときに使われる。

JPS1 のプリフェッチ種類は、特定の CPU の実装に依存しないよう一般的な使用法を想定して定義されている。表 7-10 に SPARC64™ XIfx で使用できる fcn とその動作を示す。

表 7-10　PREFETCH, PREFETCHA の fcn

fcn	JPS1 の定義	SPARC64™ XIfx の動作
0	使用頻度の高いデータを、読み出し用にプリフェッチする。	L1 データキャッシュに 256 バイトデータをコピーする。
1	使用頻度の低いデータを、読み出し用にプリフェッチする。	L2 キャッシュに 256 バイトデータをコピーする。
2	使用頻度の高いデータを、書き込み用にプリフェッチする。	L1 データキャッシュに 256 バイトデータを、排他使用権つきでコピーする。
3	使用頻度の低いデータを、書き込み用にプリフェッチする。	L2 キャッシュに 256 バイトデータを、排他使用権つきでコピーする。
4	特権ソフトウェアに、ページ割り当てを指示する。	NOP
5 – 15 (05_{16} – $0F_{16}$)	*illegal_instruction* 例外が検出される。	同左。
16 – 19 (10_{16} – 13_{16})	実装依存。	NOP
20 (14_{16})	使用頻度の高いデータを、読み出し用にプリフェッチする。ストロングプリフェッチとして扱われる。	L1 データキャッシュに 256 バイトデータをコピーする。ストロングプリフェッチとして扱われる。
21 (15_{16})	使用頻度の低いデータを、読み出し用にプリフェッチする。	L2 キャッシュに 256 バイトデータをコピーする。ストロングプリフェッチとして扱われる。
22 (16_{16})	使用頻度の高いデータを、書き込み用にプリフェッチする。	L1 データキャッシュに 256 バイトデータを、排他使用権つきでコピーする。ストロングプリフェッチとして扱われる。
23 (17_{16})	使用頻度の低いデータを、書き込み用にプリフェッチする。	L2 キャッシュに 256 バイトデータを、排他使用権つきでコピーする。ストロングプリフェッチとして扱われる。
24 – 28 (18_{16} – $1C_{16}$)	実装依存。	NOP
29 ($1D_{16}$)		L2 キャッシュに 256 バイトデータをコピーする。ストロングプリフェッチとして扱われる。
30 ($1E_{16}$)		NOP
31 ($1F_{16}$)		L2 キャッシュに 256 バイトデータを、排他使用権つきでコピーする。ストロングプリフェッチとして扱われる。

Compatibility Note　fcn $1D_{16}$, fcn $1F_{16}$ は SPARC64™ VIIIfx, SPARC64™ IXfx では 2 ラインプリフェッチを行うファンクションとして実装されていた。SPARC64™ XIfx では他の通常のプリフェッチと同様の 1 ラインプリフェッチとして動作する。

7.26.2. “strong” プリフェッチと “weak” プリフェッチ

有効なプリフェッチ fcn は、“strong”, “weak” のどちらかの属性を持つ。両者の違いは、プリフェッチロストの起きやすさと関係している。“weak” プリフェッチは、CPU 内部の資源の状況によりロストすることがある。一方、“strong” プリフェッチは、内部資源が不足していればそれが空くまで待ってプリフェッチを実行する。また、後続のロードおよびストア命令の実行に影響がある場合でもロストせず実行される。

Programming Note ストロングプリフェッチは後続のロード、ストア命令の実行を阻害するかもしれないので、プリフェッチしたデータが使われることが確実なときにのみ使うべきである。

例外	対象命令	検出条件
illegal_instruction	すべて	以下のいずれかが成立している場合。 *reserved* フィールドが 0 でない。 fcn = 5 – 15
illegal_action	すべて	XAR.v = 1 かつ下記のいずれかが成立している場合。 • XAR.simd = 1 • XAR.urs1<2:1> ≠ 00_2 • i = 0 かつ XAR.urs2<2:1> ≠ 00_2 • i = 1 かつ XAR.urs2 ≠ 0 • XAR.urd ≠ 0

7.27. Indirect Prefetch

命令	op3	操作	HPC-ACE2		アセンブリ言語表記
			Regs	SIMD	
PREFETCHID	$10\ 1101_2$	複数データのプリフェッチ	※	✓	`prefetchid` [*fregrs1*], *prefetch_fcn*

11_2	fcn	$op3 = 10\ 1101_2$	rs1	i = 0	id = 1	—
31 30	29 25	24 19	18 14	13	12	11 0

Non-SIMD 動作　Indirect Prefetch 命令は XAR.v = 1 のときのみ使用可能である。XAR.v = 0 で実行される場合、*illegal_action* 例外を検出する。

PREFETCHID 命令は、PREFETCH{|A}命令と同様に、近い将来ソフトウェアがアクセスするデータを、あらかじめキャッシュに載せておき、その後に行われるロードやストアのメモリレイテンシを低減するための命令である。命令実行が成功すると、指定したアドレスを含むメモリブロックの内容がキャッシュにコピーされる。

PREFETCHID 命令は、暗黙の ASI を使いアクセスし、“Fd[rs1]” で指定されたアドレスを含むメモリブロックをキャッシュにコピーする。

命令で指定するアドレスは、任意のアドレスが指定できる。指定されたアドレスを含む 1 キャッシュブロック（256 バイト)がコピーされる。*mem_address_not_aligned* 例外は発生しない。

指定できる fcn は表 7-10 を参照。

SIMD 動作　PREFETCHID 命令は SIMD 拡張される。

暗黙の ASI を使いアクセスし、メモリアドレスは要素ごとに“Fd[rs1][Element]”で指定される。

命令で指定するアドレスは、任意のアドレスが指定できる。要素ごとに指定されたアドレスを含む 1 キャッシュブロックがコピーされる。*mem_address_not_aligned* 例外は発生しない。

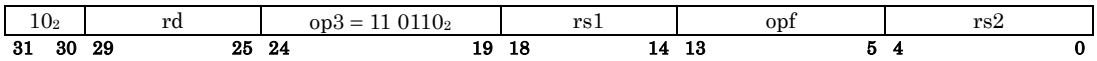
SIMD 実行時にキャッシュにコピーされる順番は保証されない。

指定できる fcn は表 7-10 を参照。

例外	対象命令	検出条件
illegal_instruction	すべて	以下のいずれかが成立している場合。 *reserved* フィールドが 0 でない。 fcn = 5 – 15
fp_disabled	すべて	PSTATE.pef = 0 または FPRS.fef = 0
illegal_action	すべて	XAR.v = 0 のとき、もしくは、 XAR.v = 1 かつ下記のいずれかが成立している場合。 • XAR.simd = 1 かつ XAR.urs1<2> ≠ 0 • XAR.urs2 ≠ 0 • XAR.urd ≠ 0

7.28. Full Element Permutation

命令	opf	操作	HPC-ACE2		アセンブリ言語表記
			Regs	SIMD	
FEPERMD	1 1000 0000$_2$	倍精度浮動小数点レジスタの要素を並べ替える	☆	✓	`fepermd` *freg$_{rs1}$, freg$_{rs2}$, freg$_{rd}$*



Non-SIMD 動作　FEPERMD 命令は、SIMD 要素間の並べ替えやマスクを実現する。SIMD 拡張を行わない場合も動作するが、本来の意図する動作とは異なる。

FEPERMD 命令は、Fd[rs2]の最上位ビットにより、Fd[rs1]か All0 を選択し、その値を Fd[rd]に格納する。Fd[rs2]<63>が 1 なら All 0 を、0 なら Fd[rs1]を選択する。このとき、Fd[rs2]<1:0>は 0 でなければならない。もし、0 でない場合 Fd[rd]に格納される値は保証されない。

本命令は、XASR.fed によって例外の検出条件が変更される。XASR.fed = 0 のとき、Fd[rs2]<62:2> ≠ 0 の値が設定された場合、*illegal_instruction* 例外が検出される。XASR.fed = 1 のとき、Fd[rs2]<62:2> ≠ 0 の値が設定された場合、*illegal_instruction* 例外は発生しないが Fd[rd]に格納される値は保証されない。

XAR.v = 1 のとき、rs1 及び rs2 には F[0]-F[510]を指定することができるが、rd には F[0]-F[254]のみ指定可能である。

FEPERMD 命令は FSR のどのフィールドも更新しない。

SIMD 動作説明　SPARC64™ XIfx では FEPERMD 命令は SIMD 拡張される。

有効な各要素において、Fd[rs2]の値により Fd[rs1]の任意要素か All 0 を選択し、その値を Fd[rd]に格納する。

Fd[rd]に格納する値は、Fd[rs2]<63>及び Fd[rs2]<1:0>により選択される。Fd[rs2]<63>が 1 なら All 0 を、Fd[rs2]<63>が 0 なら、Fd[rs2]<1:0>によって指定される要素の Fd[rs1]の値が選択される。Fd[rs2]<1:0>によって、現在有効ではない要素が選択された場合も Fd[rd]に格納される値は保証されない。

本命令は、XASR.fed によって例外の検出条件が変更される。XASR.fed = 0 のとき、Fd[rs2]<62:2> ≠ 0 の値が設定された場合、*illegal_instruction* 例外が検出される。XASR.fed = 1 のとき、Fd[rs2]<62:2> ≠ 0 の値が設定された場合、*illegal_instruction* 例外は発生しないが Fd[rd]に格納される値は保証されない。

表 7-11 に Fd[rs2]のビットの意味とそれにより Fd[rd]に格納される値の関係を示す。

Programming Note　将来の拡張のため、SIMD 拡張で FEPERMD 命令を使用する場合、Fd[rs2]<62:2>は 0 を保証しなければならない。

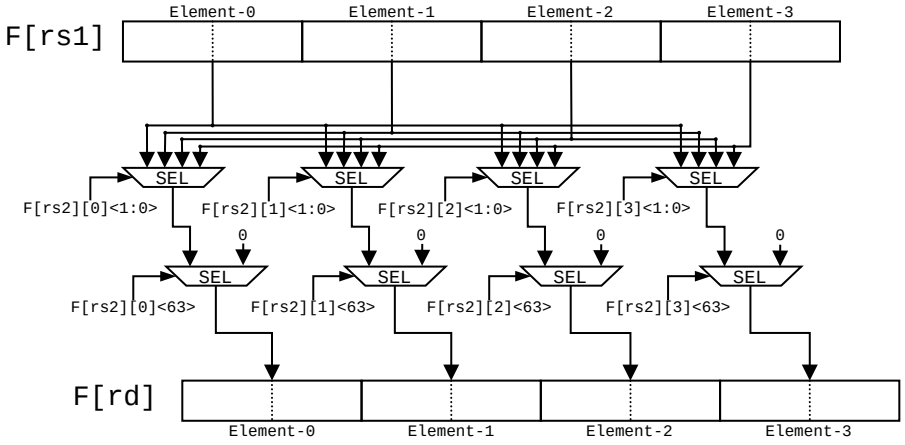
表 7-11　Fd[rs2]のビットの意味と Fd[rd]に格納される値

mask	—	element_select
63	62 … 2	1 … 0

ビット	フィールド	説明
63	mask	0: Fd[rd]に格納する値に element_select を使用する 1: Fd[rd]に All 0 を格納する
1:0	element_select	

element_select	Fd[rd]に格納される値
00_2	Fd[rs1][0]
01_2	Fd[rs1][1][xviii]
10_2	Fd[rs1][2][xix]
11_2	Fd[rs1][3][xx]

FEPERMD



仮想的な言語を使用した動作記述（Non-SIMD, XASR.fed = 1）

```
##FEPERMD
UNDEF_MASK ← 64'h7fffffff_ffffffff

MASK<0> ← F[rs2]<63>
IF (F[rs2] & UNDEF_MASK)
  F[rd] ← UNDEFINED
ELSE
  F[rd] ← MASK ? 64'h00000000_00000000 : F[rs1]
```

[xviii] XAR.simd = 0 の時に指定した場合、格納される値は保証されない
[xix] XAR.simd = 0 もしくは XASR.simd_mode = 0 の時に指定した場合、格納される値は保証されない
[xx] XAR.simd = 0 もしくは XASR.simd_mode = 0 の時に指定した場合、格納される値は保証されない

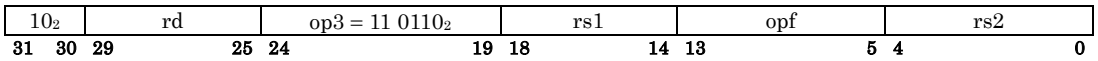
仮想的な言語を使用した動作記述（SIMD, XASR.fed = 1）

```
IF(XASR.simd_mode==0){
  SIMD_WIDTH ← 2
  UNDEF_MASK ← 64'h7fffffff_fffffffe
}ELSEIF(XASR.simd_mode==1){
  SIMD_WIDTH ← 4
  UNDEF_MASK ← 64'h7fffffff_fffffffc
}
FPR_WIDTH ← 4

##FEPERMD
FOR (ELEMENT=0; ELEMENT<SIMD_WIDTH; ELEMENT++){
  ELE_SEL<1:0> ← F[rs2][ELEMENT]<1:0>
  MASK<0> ← F[rs2][ELEMENT]<63>
  IF (F[rs2][ELEMENT] & UNDEF_MASK)
    F[rd][ELEMENT] ← UNDEFINED
  ELSE
    F[rd][ELEMENT] ← MASK ? 64'h00000000_00000000 : F[rs1][ELE_SEL]
}
FOR (ELEMENT ← SIMD_WIDTH; ELEMENT<FPR_WIDTH; ELEMENT++)
  F[rd][ELEMENT]<63:0> ← UNDEFINED
```

例外	対象命令	検出条件
illegal_instruction	全て	XASR.fed = 0 のとき、Fd[rs2]<62:2> ≠ 0
fp_disabled	全て	PSTATE.pef = 0 または FPRS.fef = 0
illegal_action	全て	XAR.v = 1 のとき、下記のいずれかが成立している場合 • XAR.urs3 ≠ 0 • XAR.simd = 1 かつ XAR.urs1<2> ≠ 0 • XAR.simd = 1 かつ XAR.urs2<2> ≠ 0 • XAR.urd<2> ≠ 0

7.29. Element Concatenate Shift Left

命令	opf	操作	HPC-ACE2		アセンブリ言語表記
			Regs	SIMD	
FECSLD	1 1000 0001_2	倍精度浮動小数点レジスタの有効な全要素を連結して左シフトを行う	4		`fecsld` *freg$_{rs1}$, freg$_{rs2}$, shc, freg$_{rd}$*



動作説明

SPARC64™ XIfx では FECSLD 命令は 4-wide SIMD 拡張でのみ使用される。

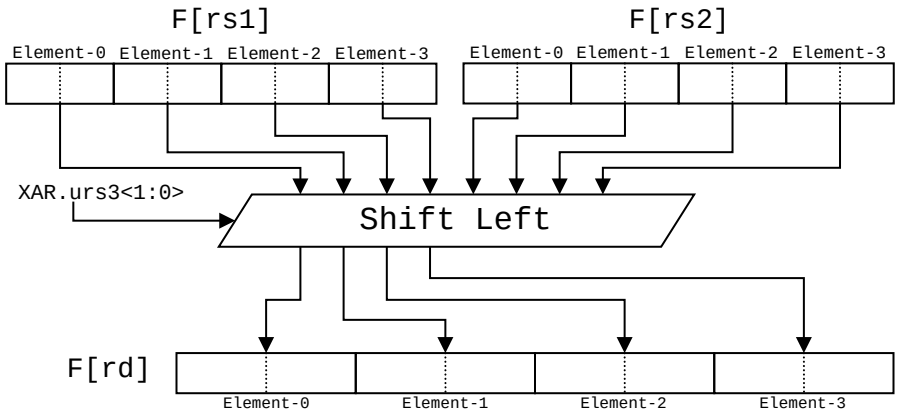
Fd[rs1]と Fd[rs2]で指定される Element-0, 1, 2, 3 の 4 要素(32 バイト)を結合した 64 バイトのデータを XAR.urs3<1:0>で指定される shc 要素数(shc * 8 バイト)左シフトを行う。その結果の上位から 32 バイトを Fd[rd]の Element-0,1,2,3 に格納する。表 7-12 に XAR.urs3<1:0>で指定する値と、Fd[rd]の各要素に格納される値の関係を示す。

表 7-12 Element Concatenate Shift Left 格納データ

urs3<1:0>	Fd[rd][0]	Fd[rd][1]	Fd[rd][2]	Fd[rd][3]
00_2	Fd[rs1][0]	Fd[rs1][1]	Fd[rs1][2]	Fd[rs1][3]
01_2	Fd[rs1][1]	Fd[rs1][2]	Fd[rs1][3]	Fd[rs2][0]
10_2	Fd[rs1][2]	Fd[rs1][3]	Fd[rs2][0]	Fd[rs2][1]
11_2	Fd[rs1][3]	Fd[rs2][0]	Fd[rs2][1]	Fd[rs2][2]

FECSLD 命令は FSR のどのフィールドも更新しない。

FECSLD



仮想的な言語を使用した動作記述（SIMD）

```
SIMD_WIDTH ← 4
SHC ←(XAR.urs3<1:0>)

##FECSLD
```

```
FOR (ELEMENT ← 0; ELEMENT<SIMD_WIDTH; ELEMENT++){
  IF ((ELEMENT + SHC) < SIMD_WIDTH)
    F[rd][ELEMENT] ← F[rs1][ELEMENT + SHC]
  ELSE
    F[rd][ELEMENT] ← F[rs2][ELEMENT + SHC - SIMD_WIDTH]
}
```

例外	対象命令	検出条件
fp_disabled	全て	PSTATE.pef = 0 または FPRS.fef = 0
illegal_action	全て	XAR.v ≠ 1 もしくは XAR.simd ≠ 1 もしくは XASR.simd_mode = 0 いずれかが成立している場合 XAR.v = 1 かつ XAR.simd = 1 かつ XASR.simd_mode = 1 のとき、下記のいずれかが成立している場合 • XAR.urs3<2> ≠ 0 • XAR.urs1<2> ≠ 0 • XAR.urs2<2> ≠ 0 • XAR.urd<2> ≠ 0

7.30. Element Sum Mask

命令	opf	操作	HPC-ACE2		アセンブリ言語表記
			Regs	SIMD	
FESUMMD	1 1000 0010$_2$	倍精度浮動小数点レジスタの有効な要素のマスク数を計算	☆	✓	fesummd *freg$_{rs2}$, freg$_{rd}$*

10$_2$	rd	op3 = 11 0110$_2$	—	opf	rs2
31 30	29 25	24 19	18 14	13 5	4 0

動作説明　FESUMMD 命令は Fd[rs2]<63>が 1 である場合、Fd[rd]に 1 を格納する。Fd[rs2]<63>が 0 である場合 Fd[rd]に 0 を格納する。

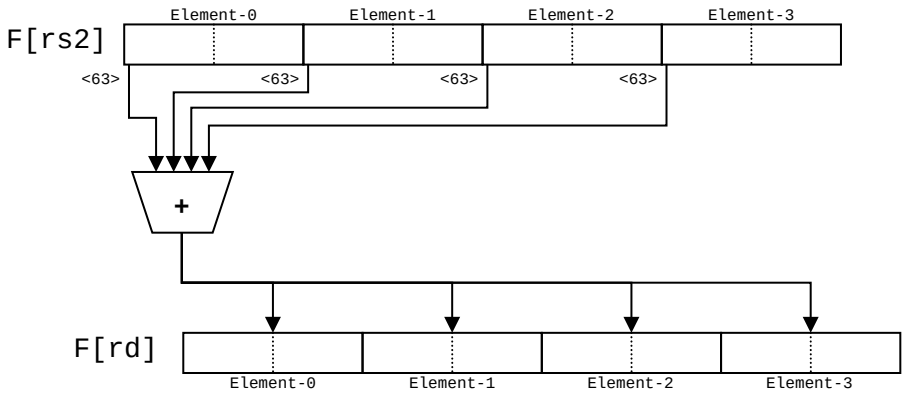
XAR.v = 1 のとき、rs2 には F[0]-F[510]を指定することができるが、rd には F[0]-F[254]のみ指定可能である。

FESUMMD 命令は FSR のどのフィールドも更新しない。

SIMD 動作説明　SPARC64™ XIfx では FESUMMD 命令は SIMD 拡張される。

有効な要素の中で Fd[rs2]<63>が 1 である要素数を数え、結果を有効な全要素の Fd[rd]に格納する。

FESUMMD 動作



仮想的な言語を使用した動作記述（SIMD）

```
IF(XASR.simd_mode==0)
  SIMD_WIDTH ← 2
ELSEIF(XASR.simd_mode==1)
  SIMD_WIDTH ← 4
FPR_WIDTH ← 4

##FESUMMD
TMP ← 0
FOR (ELEMENT ← 0; ELEMENT<SIMD_WIDTH; ELEMENT++)
  TMP ← TMP + F[rs2][ELEMENT]<63>

FOR (ELEMENT ← 0; ELEMENT<SIMD_WIDTH; ELEMENT++)
  F[rd][ELEMENT] ← TMP
```

```
FOR (ELEMENT ← SIMD_WIDTH; ELEMENT<FPR_WIDTH; ELEMENT++)
  F[rd][ELEMENT]<63:0> ← UNDEFINED
```

例外	対象命令	検出条件
illegal_instruction	全て	*reserved* が 0 でないとき
fp_disabled	全て	PSTATE.pef = 0 または FPRS.fef = 0
illegal_action	全て	XAR.v = 1 のとき、下記のいずれかが成立している場合 • XAR.urs3 ≠ 0 • XAR.urs1 ≠ 0 • XAR.simd = 1 かつ XAR.urs2<2> ≠ 0 • XAR.urd<2> ≠ 0

7.31. Element Compress

命令	Opf	操作	HPC-ACE2		アセンブリ言語表記
			Regs	SIMD	
FECPD	1 1000 0011$_2$	倍精度浮動小数点レジスタのマスクされた要素を集める	☆	✓	fecpd *freg$_{rs1}$, freg$_{rs2}$, freg$_{rd}$*

10_2	rd	op3 = 11 0110$_2$	rs1	opf	rs2
31 30	29 25	24 19	18 14	13 5	4 0

動作説明　FECPD 命令は Fd[rs2]<63>が 1 だった場合、Fd[rd]に Fd[rs1]を格納する。Fd[rs2]<63>が 0 だった場合、Fd[rd]に All0 を格納する。

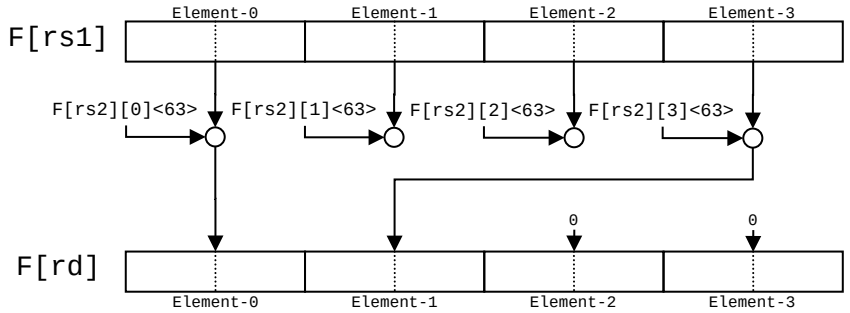
XAR.v = 1 のとき、rs1 及び rs2 には F[0]-F[510]を指定することができるが、rd には F[0]-F[254]のみ指定可能である。

FECPD 命令は FSR のどのフィールドも更新しない。

SIMD 動作説明　SPARC64™ XIfx では FECPD 命令は SIMD 拡張される。

有効な要素の中で Fd[rs2]<63>が 1 である要素の Fd[rs1]を Fd[rd]の要素 0 から順に格納する。格納するものがない要素は All 0 を格納する。有効でない要素の Fd[rd]には不定値が格納される。

FECPD



上記例では、F[rs2][0]<63>=1, F[rs2][1]<63>=0, F[rs2][2]<63>=0, F[rs2][3]<63>=1 である。

仮想的な言語を使用した動作記述（SIMD）

```
IF(XASR.simd_mode==0)
  SIMD_WIDTH ← 2
ELSEIF(XASR.simd_mode==1)
  SIMD_WIDTH ← 4
FPR_WIDTH ← 4

##FECPD
TMP ← 0
FOR (ELEMENT ← 0; ELEMENT<SIMD_WIDTH; ELEMENT++){
  IF (F[rs2][ELEMENT]<63>){
    F[rd][TMP] ← F[rs1][ELEMENT]
    TMP++
  }
}
FOR (ELEMENT ← TMP; ELEMENT<SIMD_WIDTH; ELEMENT++){
```

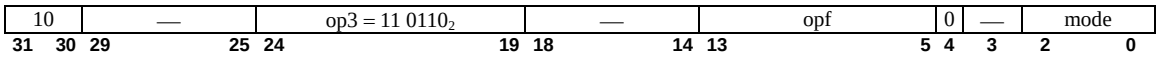
```
  F[rd][ELEMENT] ← 64'h00000000_00000000
}

FOR (ELEMENT ← SIMD_WIDTH; ELEMENT<FPR_WIDTH; ELEMENT++)
  F[rd][ELEMENT] ← UNDEFINED
```

例外	対象命令	検出条件
fp_disabled	全て	PSTATE.pef = 0 または FPRS.fef = 0
illegal_action	全て	XAR.v = 1 のとき、下記のいずれかが成立している場合 • XAR.urs3 ≠ 0 • XAR.simd = 1 かつ XAR.urs1<2> ≠ 0 • XAR.simd = 1 かつ XAR.urs2<2> ≠ 0 • XAR.urd<2> ≠ 0

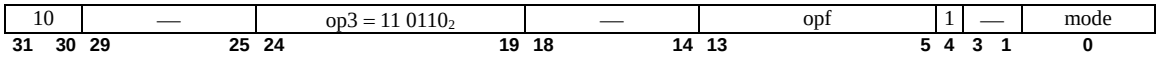
7.32. Set SIMD Arithmetic Mode

命令	opf	iw<4>	操作	HPC-ACE2 Regs SIMD	アセンブリ言語表記
SSM	0 1000 1101$_2$	0$_2$	SIMD モードを設定する		ssm *simd_mode*
SNF	0 1000 1101$_2$	1$_2$	Non-Faulting モードを設定する		snf *nf_mode*

SSM



SNF



SSM 命令は浮動小数点演算器で実行する SIMD 演算幅を設定する。

XASR.simd_mode ←mode<2:0>

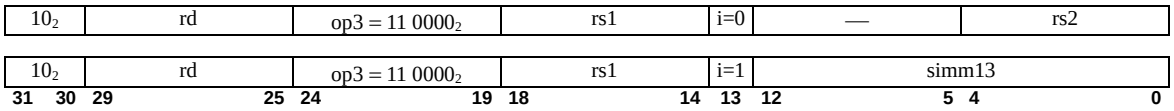
SNF 命令は Non-Faulting モードを設定する。

XASR.nf ←mode<0>

例外	対象命令	検出条件
illegal_instruction	SSM	以下のいずれかが成立する場合。 • *reserved* フィールドが 0 でない。 • mode<2:1> ≠ 00$_2$
	SNF	以下のいずれかが成立する場合。 *reserved* フィールドが 0 でない。
fp_disabled	すべて	PSTATE.pef = 0 または FPRS.fef = 0
illegal_action	すべて	XAR.v = 1 のとき

7.33. Write Ancillary State Register (WRASR)

命令	rd	操作	HPC-ACE2		アセンブリ言語表記	
			Regs	SIMD		
WRY[D]	0	Y レジスタの更新。(使用を推奨しない)	✓		wr	*reg*$_{rs1}$, *reg_or_imm*, %y
WRCCR	2	CCR レジスタの更新。	✓		wr	*reg*$_{rs1}$, *reg_or_imm*, %ccr
WRASI	3	ASI レジスタの更新。	✓		wr	*reg*$_{rs1}$, *reg_or_imm*, %asi
WRFPRS	6	FPRS レジスタの更新。	✓		wr	*reg*$_{rs1}$, *reg_or_imm*, %fprs
WRPCR[PPCR]	16	PCR レジスタの更新。	✓		wr	*reg*$_{rs1}$, *reg_or_imm*, %pcr
WRPIC[PPCR]	17	PIC レジスタの更新。	✓		wr	*reg*$_{rs1}$, *reg_or_imm*, %pic
WRGSR	19	GSR レジスタの更新。	✓		wr	*reg*$_{rs1}$, *reg_or_imm*, %gsr
WRXAR	29	XAR レジスタの更新。	✓		wr	*reg*$_{rs1}$, *reg_or_imm*, %xar
WRXASR	30	XASR レジスタの更新。	✓		wr	*reg*$_{rs1}$, *reg_or_imm*, %xasr



WRASR 命令は、Ancillary State レジスタの有効なフィールドに値を設定する命令である。Ancillary State レジスタの詳細は Section **5.5** (31 ページ)を参照。

WRASR により設定される値は、命令語の i フィールドが 0 のときは “R[rs1] **xor** R[rs2]” で、i フィールドが 1 のときは “R[rs1] **xor** sign_ext(simm13)” である。XOR に注意。

WRASR は直ちに結果が反映される命令である。設定後の値は、命令列上これらの命令の直後に位置する命令から有効となる。

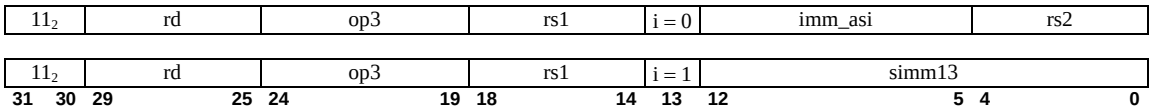
- WRY は、Y レジスタに値を設定する命令である。Y レジスタを使う命令はすべて、使用が推奨されない命令(deprecated)となっている。
- WRFPRS は、先行するすべての命令の実行が完了した後で、FPRS に値が設定される。
- WRXAR で XAR の *reserved* に 0 以外の値を設定しようとすると、*illegal_instruction* 例外が発生する。ただし、この理由による *illegal_instruction* と *illegal_action* 例外では、*illegal_action* 例外の優先順位の方が高い。
- WRXASR 命令は simd_mode<2:1>に 0 以外の値を設定すると、*illegal_instruction* 例外を発生する。

例外	対象命令	検出条件
illegal_instruction	—	rd = 1, 4 – 5, 7 – 14, 18, 26 – 28
	すべて	i = 0 かつ iw<12:5> ≠ $0000\ 0000_2$
	WRXASR	simd_mode<2:1>≠ 00_2 のとき
fp_disabled	WRGSR	PSTATE.pef = 0 または FPRS.fef = 0
illegal_action	すべて	XAR.v = 1 かつ、下記のいずれかが成立している場合 • XAR.simd = 1 • XAR.urs1<2:1> ≠ 00_2 • i = 0 かつ XAR.urs2<2:1> ≠ 00_2 • i = 1 かつ XAR.urs2 ≠ 0 • XAR.urs3 ≠ 0 • XAR.urd ≠ 0

7.34. Cache Line Fill with Undetermined Values

命令	ASI	op3	操作	HPC-ACE2		アセンブリ言語表記
				Regs	SIMD	
XFILL[N]	ASI_XFILL_P ASI_XFILL_S	$01\ 1110_2$ $01\ 0111_2$ $11\ 0111_2$ $01\ 0100_2$ $01\ 0101_2$ $01\ 0110_2$ $11\ 0100_2$	nop	✓		stxa *reg$_{rd}$* , [*address*] %*asi* stxa *reg$_{rd}$* , [*address*] *imm_asi* stwa *reg$_{rd}$* , [*address*] %*asi* stwa *reg$_{rd}$* , [*address*] *imm_asi* stha *reg$_{rd}$* , [*address*] %*asi* stha *reg$_{rd}$* , [*address*] *imm_asi* sttwa *reg$_{rd}$* , [*address*] %*asi* sttwa *reg$_{rd}$* , [*address*] *imm_asi* stba *reg$_{rd}$* , [*address*] %*asi* stba *reg$_{rd}$* , [*address*] *imm_asi* sta *freg$_{rd}$* , [*address*] %*asi* sta *freg$_{rd}$* , [*address*] *imm_asi* stda *freg$_{rd}$* , [*address*] %*asi* stda *freg$_{rd}$* , [*address*] *imm_asi*
XFILL256	ASI_XFILL256_P	$01\ 1110_2$ $01\ 0111_2$ $11\ 0111_2$ $01\ 0100_2$ $01\ 0101_2$ $01\ 0110_2$ $11\ 0100_2$	プライマリ空間の指定されたアドレスをキャッシュにのせ、不定値で埋める。	✓		stxa *reg$_{rd}$* , [*address*] %*asi* stxa *reg$_{rd}$* , [*address*] *imm_asi* stwa *reg$_{rd}$* , [*address*] %*asi* stwa *reg$_{rd}$* , [*address*] *imm_asi* stha *reg$_{rd}$* , [*address*] %*asi* stha *reg$_{rd}$* , [*address*] *imm_asi* sttwa *reg$_{rd}$* , [*address*] %*asi* sttwa *reg$_{rd}$* , [*address*] *imm_asi* stba *reg$_{rd}$* , [*address*] %*asi* stba *reg$_{rd}$* , [*address*] *imm_asi* sta *freg$_{rd}$* , [*address*] %*asi* sta *freg$_{rd}$* , [*address*] *imm_asi* stda *freg$_{rd}$* , [*address*] %*asi* stda *freg$_{rd}$* , [*address*] *imm_asi*

命令	ASI	op3	操作	HPC-ACE2		アセンブリ言語表記
				Regs	SIMD	
XFILL256	ASI_XFILL256_S	$01\ 1110_2$ $01\ 0111_2$ $11\ 0111_2$ $01\ 0100_2$ $01\ 0101_2$ $01\ 0110_2$ $11\ 0100_2$	セカンダリ空間の指定されたアドレスをキャッシュにのせ、不定値で埋める。	✓		stxa *reg*$_{rd}$, [*address*] %*asi* stxa *reg*$_{rd}$, [*address*] *imm_asi* stwa *reg*$_{rd}$, [*address*] %*asi* stwa *reg*$_{rd}$, [*address*] *imm_asi* stha *reg*$_{rd}$, [*address*] %*asi* stha *reg*$_{rd}$, [*address*] *imm_asi* sttwa *reg*$_{rd}$, [*address*] %*asi* sttwa *reg*$_{rd}$, [*address*] *imm_asi* stba *reg*$_{rd}$, [*address*] %*asi* stba *reg*$_{rd}$, [*address*] *imm_asi* sta *freg*$_{rd}$, [*address*] %*asi* sta *freg*$_{rd}$, [*address*] *imm_asi* stda *freg*$_{rd}$, [*address*] %*asi* stda *freg*$_{rd}$, [*address*] *imm_asi*



動作説明　　STXA, STWA, STHA, STTWA, STBA, STFA, STDFA にこれらの ASI を指定すると、指定されたアドレスに対応するキャッシュラインをキャッシュ上に書き込み用に確保し、そのアドレスを含むキャッシュラインを不定値で埋める。メモリから CPU へのデータ転送は発生しない。命令で指定するアドレスは、キャッシュライン上の任意のアドレスを指定してよい。

Compatibility Note　SPARC64™ XIfx では SPARC64™ VIIIfx, SPARC64™ IXfx とキャッシュラインサイズが変更されたためバイナリ互換を維持して XFILL の動作を実装することが困難となった。
そのため、既存の XFILL 命令（ASI に ASI_XFILL_{P|S}を指定した場合）は NOP として処理する。そして、SPARC64™ XIfx の XFILL 命令を XFILL256 として新規に定義する。この XFILL256 命令は新規追加 ASI である ASI_XFILL256_{P|S}を使用する。

Compatibility Note　SPARC64™ VIIIfx, SPARC64™ IXfx では 8 バイト境界にアクセスするストア命令のみ使用可能であった。SPARC64™ XIfx では制限を緩和し、STWA, STBA, STHA, STFA, STBA 命令が使用可能となった。これにより、境界を気にせず XFILL 命令を実行することが可能である。

Programming Note　アラインを意識せず、XFILL256 命令を使用する場合、stba を使うこと。今後、XFILL256 を使用する場合は STBA 命令を使用することを推奨する。

STXA, STTWA は、8 バイト境界でないアドレスを指定すると *mem_address_not_aligned* を通知する。

STFA, STWA は、4 バイト境界でないアドレスを指定すると *mem_address_not_aligned* を通知する。

STHA は、2 バイト境界でないアドレスを指定すると *mem_address_not_aligned* を通知する。

STDFA は、4 バイト境界だが 8 バイト境界でないアドレスを指定すると *STDF_mem_address_not_aligned* を通知し、8 バイト境界でも 4 バイト境界でもないアドレスを指定すると *mem_address_not_aligned* を通知する。

XFILL256_{S|P} ではハードウェアプリフェッチの on/off 指定は意味を持たない。XAR.dis_hw_pf の値は無視される。

XFILL256_{S|P} と後続のメモリアクセスの間のメモリオーダは TSO が遵守される。

ノンキャッシャブル空間に対する XFILL256_{S|P}は、ウォッチポイント、アラインメント、プロテクション違反は検出されるが、キャッシュラインの不定値フィルは行われない。

XFILL256_{S|P} はキャッシュライン 256 バイト全てでウォッチポイントの一致を検出する。

キャッシュラインフィル中に、当該キャッシュラインに対する後続アクセスが発生すると、後続アクセスはキャッシュラインフィルが完了するまで保留される。

Compatibility Note SPARC64™ IXfx, SPARC64™ VIIIfx では、キャッシュラインが 128B であった。SPARC64™ XIfx ではキャッシュラインが 256B であるため、命令動作が異なることに注意が必要である。

Programming Note XFILL256 と後続アクセスの間に MEMBAR は不要。後続アクセスが保留されると性能が低下するので、高速なプログラムを作成しようとするとき XFILL256 は実際のストアに十分先立って実行する必要がある。しかし、XFILL256 が完了するまでにかかる時間はシステムに依存するので、あるシステムで十分先立って実行されていたものが、将来のプロセッサでは不十分になる可能性がある。

XFILL 命令(XFILL256 命令を除く)は、SPACR64™ VIIIfx, SPARC64™ IXfx との互換性のために残してある。キャッシュライン全体(128 バイト)を不定値で更新する命令だったが、SPARC64™ XIfx ではメモリ、キャッシュに対し何の操作も行わない。ただし、メモリアクセスに関連する例外は検出する。

XFILL 命令(XFILL256 命令を除く)は、互換性のため 128 バイト全てでウォッチポイントの一致を検出する。

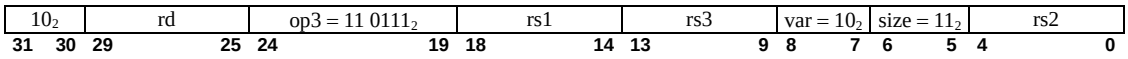
ノンキャッシャブル空間に対する XFILL は *DAE_nc_page* 例外を通知しない。

例外	対象命令	検出条件
illegal_instruction	STTWA	rd に偶数以外を指定した場合
fp_disabled	STDFA, STFA	PSTATE.pef = 0 または FPRS.fef = 0
illegal_action	STXA, STTWA, STWA, STBA, STHA	XAR.v = 1 かつ下記のいずれかが成立している場合 • XAR.urs1<2:1> ≠ 00_2 • i = 0 かつ XAR.urs2<2:1> ≠ 00_2 • i = 1 かつ XAR.urs2 ≠ 0 • XAR.urd<2:1> ≠ 00_2 • XAR.simd = 1

	STFA, STDFA	XAR.v = 1 かつ下記のいずれかが成立している場合 • XAR.urs1<2:1> ≠ 00_2 • i = 0 かつ XAR.urs2<2:1> ≠ 00_2 • i = 1 かつ XAR.urs2 ≠ 0 • XAR.simd = 1
STDF_mem_address_not_aligned	STDFA	XAR.v = 0 または XAR.simd = 0 で、4 バイト境界だが 8 バイト境界ではないアドレスに書き込もうとしたとき。
mem_address_not_aligned	STXA, STTWA	8 バイト境界ではないアドレスに書き込もうとしたとき。
	STDFA, STFA, STWA	4 バイト境界ではないアドレスに書き込もうとしたとき。
	STHA	2 バイト境界ではないアドレスに書き込もうとしたとき。
VA_watchpoint	ASI_XFILL256_P ASI_XFILL256_S	アクセスしたアドレスを含む 256 バイト境界 256 バイト領域の任意アドレスにウォッチポイントアドレスがマッチした場合
	ASI_XFILL_P ASI_XFILL_S	アクセスしたアドレスを含む 128 バイト境界 128 バイト領域の任意アドレスにウォッチポイントアドレスがマッチした場合
DAE_privilege_violation	ASI_XFILL_P ASI_XFILL_S ASI_XFILL256_P ASI_XFILL256_S	
DAE_nfo_page	すべて	

7.35. Shift Mask Or

命令	var	size	操作	HPC-ACE2		アセンブリ言語表記
				Regs	SIMD	
FSHIFTORX	10_2	11_2	2 つの倍精度浮動小数点レジスタの値を結合する	✓	✓	`fshiftorx` *freg*$_{rs1}$, *freg*$_{rs2}$, *freg*$_{rs3}$, *freg*$_{rd}$



非 SIMD 動作

FSHIFTORX 命令は、Fd[rs3]で指定するパラメータに従って、以下のいずれかの操作を行う。A)では 2 つのフィールドを使用し、B)では 3 つのフィールドを使用する。

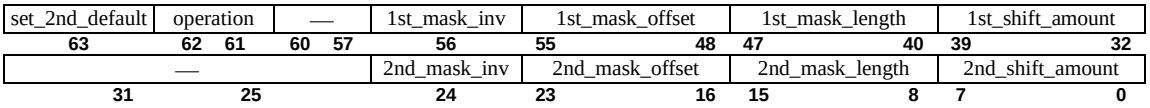
A) 1st フィールドの Fd[rs1]の内容を右または左にシフトし、一部分だけを取り出したものと、2nd フィールドの Fd[rs2]の内容を右または左にシフトし、一部分だけを取り出したものとをビット単位で OR 演算を行い、結果を Fd[rd]に格納する。

B) 1st フィールドの Fd[rs1]の内容を右または左にシフトし、一部分だけを取り出したものと、2nd フィールドの Fd[rs1]の内容を右または左にシフトし、一部分だけを取り出したものとをビット単位で OR 演算を行い、さらにその結果と、3rd フィールドの Fd[rs2]の内容に算術演算（AND、OR 又は XOR）を行った結果を、Fd[rd]に格納する。

本命令は、XASR.fed により例外の検出条件が変更される。XASR.fed = 0 のとき Fd[rs3]に以下の値を指定すると *illegal_instruction* 例外が発生する。
1) *reserved* フィールドに 1 を指定
2) set_rs2_default に 1 を指定し、<31:0>に 0 以外を指定

XASR.fed = 1 のとき Fd[rs3]に上記の値を設定した場合、*illegal_instruction* 例外は発生しないが、Fd[rd]に格納される値は保証されない。

表 7-13 Fd[rs3]のビットの意味

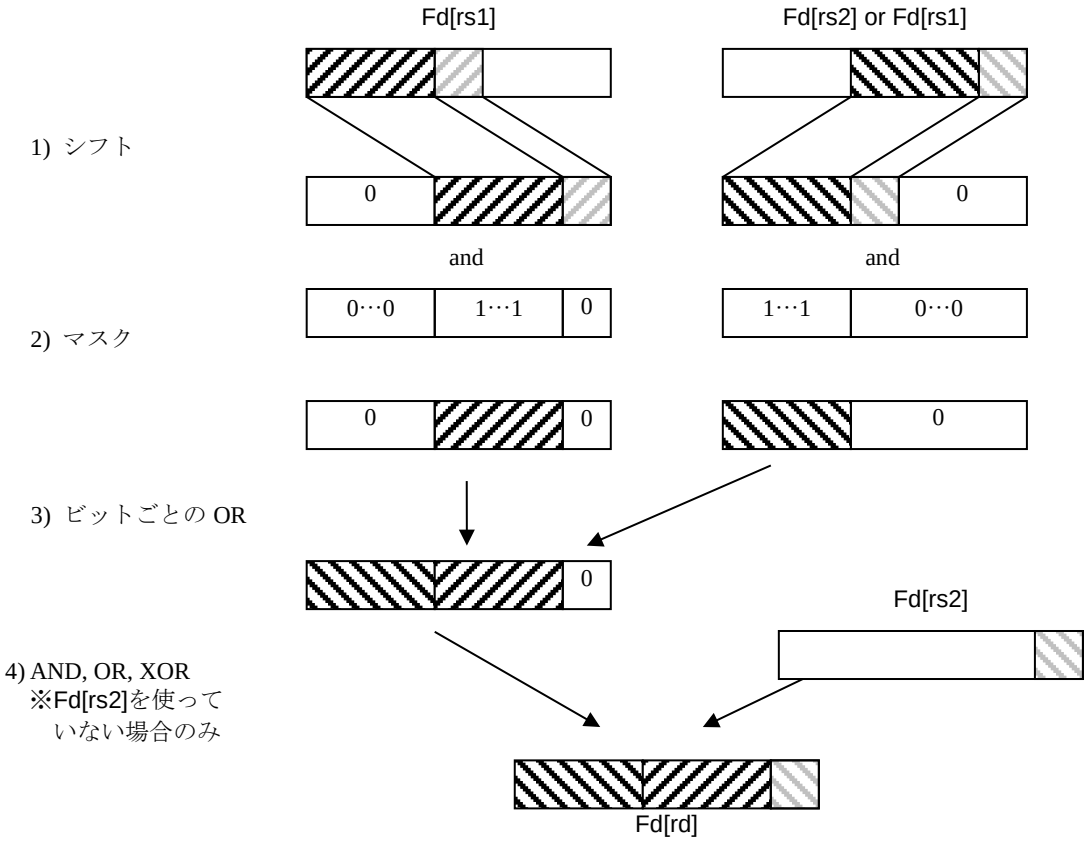


ビット		フィールド	説明
rs1	rs2		
63	—	set_2nd_default	rs2 用フィールドの値を自動計算するかどうか指定する 0: 2nd フィールドの設定値を使用する 1: 1st フィールドから自動計算した値を使用する
62:61	—	operation	操作パターンを指定する。表 7-14 参照。
56	24	{1st|2nd}_mask_inv	マスクビットを反転させるかどうかを指定する 0: 反転させない 1: 反転させる
55:48	23:16	{1st|2nd}_mask_offset	マスクビット開始位置（最上位からのビット数） 8-bit 符号つき整数
47:40	15:8	{1st|2nd}_mask_length	マスクビット長 8-bit 符号つき整数
39:32	7:0	{1st|2nd}_shift_amount	シフト量（正の数は左シフト、負の数は右シフト） 8-bit 符号つき整数

表 7-14　operation フィールドによる操作パターン

operation	1st フィールド	2nd フィールド	3rd フィールド	1st フィールドと 2nd フィールドで生成された値と、3rd フィールドとの算術演算
00_2	rs1	rs2	—	—
01_2	rs1	rs1	rs2	AND
10_2	rs1	rs1	rs2	OR
11_2	rs1	rs1	rs2	XOR

FSHIFTORX 命令の動作は、1）シフト、2）マスク、3) OR 、4）AND OR XOR の 4 つに分けられる。



シフト操作で行われるのは論理シフトである。左（MSB 方向）へシフトする場合は、空いた右側（LSB 側）には 0 が入る。右（LSB 方向）へシフトする場合は、空いた左側（MSB 側）には 0 が入る。{1st|2nd}_shift_amount はシフト方向とシフト量を指示するフィールドで、正の数なら左方向、負の数なら右方向にシフトする。

シフト後のレジスタから特定のビット列を抜き出すのが、マスク操作である。マスクパターンは{1st|2nd}_mask_offset, {1st|2nd}_mask_length および{1st|2nd}_mask_inv で作成され、マスクパターンとビット毎の AND を取る。

マスクパターンは、任意長の連続する 1 が 64 ビット内のどこか一箇所に存在するパターン(またはどこにも存在しないパターン)、およびその反転のみが指定可能である。マスクの開始位置を左（MSB）からのビット数で{1st|2nd}_mask_offset に、連続する 1 のビット数を {1st|2nd}_mask_length に指定する。こうして作られたマスクパターンの全ビットを反転させるには、{1st|2nd}_mask_inv に 1 を指定する。

Note {1st|2nd}_mask_offset が 0 以上かつ 64 未満、かつ {1st|2nd}_mask_length が 0 より大きければ、1 を含むマスクパターンが生成される。{1st|2nd}_mask_offset + {1st|2nd}_mask_length が 64 以上となる場合は、{1st|2nd}_mask_offset から LSB まで 1 が連続するマスクパターンが生成される。

Note {1st|2nd}_mask_offset が負の数であっても、1 を含むマスクパターンが生成される場合がある。{1st|2nd}_mask_offset の絶対値が {1st|2nd}_mask_length より小さければ、MSB から連続する {1st|2nd}_mask_length − |{1st|2nd}_mask_offset| ビットのマスクパターンが生成される。

Note {1st|2nd}_mask_offset と{1st|2nd}_mask_length が以下の条件のいずれかを満たすと、64 ビット全て 0 のマスクパターンが生成される。このとき {1st|2nd}_mask_inv に 1 を指定すると、64 ビット全て 1 のマスクパターンが生成される。

- {1st|2nd}_mask_length ≤ 0
- {1st|2nd}_mask_offset + rs{1|2}_mask_length ≤ 0
- {1st|2nd}_mask_offset ≥ 64

A)のパターンの場合、このようにして得られた Fd[rs1]の一部分と、Fd[rs2]の一部分をビット毎に OR し、その結果を Fd[rd]に格納する。B)のパターンの場合、このようにして得られた Fd[rs1]の一部分と、別に生成した Fd[rs1]の一部分をビット毎に OR し、さらにその結果に Fd[rs2]と論理演算（例：OR）した結果を、Fd[rd]に格納する。

SIMD 動作

FSHIFTORX 命令を SIMD で使用する場合、Fd[rs1]には Element-0, Element-1 のすべての浮動小数点レジスタを指定することができる。

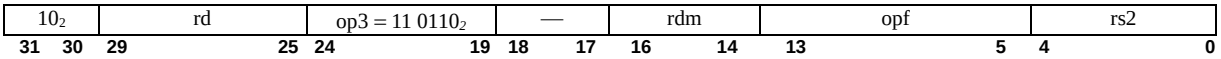
Fd[rs1]に Fd[2n](n = 128 ~ 255)のレジスタを指定すると、実際に使用するレジスタは Fd[2n-128]であるが、各 Element の演算で Fd[rs1]に使用するレジスタの Element 番号の LSB 1bit を反転させる。

たとえば、Fd[rs1]に Element-1 側レジスタ Fd[256] – Fd[510]を指定すると Element-0 側の演算に Element-1 の Fd[n-256]、Element-1 側の演算に Element-0 の Fd[n – 256]が使われる。同様に、Element-2 側の演算に Element-3 の Fd[n-256]、Element-3 側の演算に Element-2 の Fd[n – 256]が使われる。これに対し、Fd[rs2], Fd[rs3], Fd[rd]には Fd[0] – Fd[254]のみを指定できる。

例外	対象命令	検出条件
fp_disabled	すべて	PSTATE.pef = 0 または FPRS.fef = 0
illegal_instruction	すべて	本文参照。
illegal_action	すべて	XAR.v = 1 かつ XAR.simd = 1 で、下記のいずれかが成立している場合 • XAR.urs2<2> = 1 • XAR.urs3<2> = 1 • XAR.urd<2> = 1

7.36. Floating-Point Round-Off

命令	opf	操作	HPC-ACE2		アセンブリ言語表記	
			Regs.	SIMD		
FRDs	$1\ 0111\ 1111_2$	単精度浮動小数の丸め	✓	✓	frds	$freg_{rs2}$, rdm, $freg_{rd}$
FRDd	$1\ 0111\ 1110_2$	倍精度浮動小数の丸め	✓	✓	frdd	$freg_{rs2}$, rdm, $freg_{rd}$



動作説明

FRD{s|d}命令は浮動小数点数を即値で指定される丸めモードにしたがって整数に丸め、それを浮動小数点数としてレジスタに格納する

FRDs 命令は、F[rs2]の単精度浮動小数点数を rdm<2:0>で指定された値の丸めモードに従い、整数に丸め、それを F[rd]に単精度浮動小数点数として格納する。丸めモードを表 7-15 に示す。

FRDd 命令は、Fd[rs2]の倍精度浮動小数点数を rdm<2:0>で指定された値の丸めモードに従い、整数に丸め、それを Fd[rd]に倍精度浮動小数点数として格納する。丸めモードを表 7-15 に示す。

FRD{s|d}命令は rdm<2:0> ≧ 5 を指定したとき、*illegal_instruction* 例外を検出する。

FRD{s|d}の演算結果と例外条件を表 7-16 に示す。結果の欄の上段が例外、下段は例外が通知されないときの値である。

表 7-15 浮動小数点演算の丸め方法

rdm<2:0>	丸め方法
000_2	近いほう、等しいときは偶数（Nearest, Ties to Even）
001_2	0 方向（Round toward 0）
010_2	+∞方向（Round toward +∞）
011_2	-∞方向（Round toward -∞）
100_2	近いほう、等しいときは 0 から遠い方向（Nearest, Ties to Away）
101_2 - 111_2	*reserved*

表 7-16　FRD{s|d}命令の演算結果

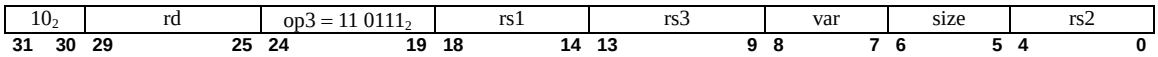
rs2	rd
-∞	— -∞
-N	— rdm により正しく丸められる
-D	— rdm により正しく丸められる
-0	— -0
+0	— +0
+D	— rdm により正しく丸められる
+N	— rdm により正しく丸められる
+∞	— +∞
QNaN	— QNaN
SNaN	NV QSNaN

SIMD 動作　　有効な要素においては上記の動作を行い、有効でない要素の Fd[rd]には不定値が格納される。

例外		対象命令	検出条件
illegal_instruction		すべて	*reserved* が 0 でないとき もしくは rdm<2> = 1_2 かつ rdm<1:0> ≠ 00_2
fp_disabled		すべて	PSTATE.pef = 0 または FPRS.fef = 0
illegal_action		すべて	XAR.v = 1 かつ、下記のいずれかが成立している場合 • XAR.urs3 ≠ 0 • XAR.urs1 ≠ 0 • XAR.simd = 1 かつ XAR.urs2<2> ≠ 0 • XAR.simd = 1 かつ XAR.urd<2> ≠ 0
fp_exception_ieee_754	NV	すべて	入力値が SNaN のとき

7.37. Integer Multiply-Add

命令	var	size	操作	HPC-ACE2 Regs	HPC-ACE2 SIMD	アセンブリ言語表記	
FPMADDX	00_2	00_2	符号なし整数の積和演算の下位 8 バイト	✓	✓	fpmaddx	$freg_{rs1},freg_{rs2},freg_{rs3},freg_{rd}$
FPMADDXHI	01_2	00_2	符号なし整数の積和演算の上位 8 バイト	✓	✓	fpmaddxhi	$freg_{rs1},freg_{rs2},freg_{rs3},freg_{rd}$



動作説明

整数積和演算は、浮動小数点レジスタに格納された符号なし 8 バイト整数の乗算と加算を行う。

FPMADDX は、Fd[rs1]の符号なし 8 バイト整数と Fd[rs2]の符号なし 8 バイト整数を乗じ、その結果に Fd[rs3]の符号なし 8 バイト整数を加算し、結果の下位 8 バイトを Fd[rd]に格納する。

FPMADDXHI は、Fd[rs1]の符号なし 8 バイト整数と Fd[rs2]の符号なし 8 バイト整数を乗じ、その結果に Fd[rs3]の符号なし 8 バイト整数を加算し、結果の上位 8 バイトを Fd[rd]に格納する。

FPMADDX, FPMADDXHI は FSR のどのフィールドも更新しない。

SIMD 動作説明

FPMADDX, FPMADDXHI 命令は SPARC64™ XIfx の SIMD 拡張では、FMADD 系命令と同様に制限つきではあるが他要素のレジスタを使用することが可能となっている。

> **Note** ここに書かれているのは XAR.simd = 1 のときのことである。XAR.simd = 0 のときは rs1, rs2, rs3, rd にすべての浮動小数点レジスタが使える。

FPMADDX, FPMADDXHI 命令では、rs1, rs2 に Element-0, Element-1 の全てのレジスタを指定することができる。Fd[2n](n = 128 ~ 255)のレジスタを指定すると、実際に使用するレジスタは Fd[2n-128]であるが、使用するレジスタの Element 番号 LSB 1bit を反転させる。たとえば、Fd[2n] (n = 128 ~ 255)を使用すると、Element-0 側の演算では、Element-1 のレジスタを、Element-1 側の演算では、Element-0 のレジスタをそれぞれ使用する。同様に Element-2 側の演算では、Element-3 のレジスタを、Element-3 側の演算では Element-2 のレジスタを使用する。

これに対し rs3 と rd は他の SIMD 拡張と同じで、Fd[2n] (n = 0 ~ 127)という制限のままである。したがって、urs3<2>と urd<2>はレジスタ指示には使われない。urs3<2>は別の機能拡張に使用される。urs3<2> = 1 のとき、Element 番号が Odd 側の演算の rs1 には Element 番号が Even 側と同じものが使われる。

> **Note** FMADD 命令では urd<2>に negate_mul の機能が割り当てられていたが、FPMADDX, FPMADDXHI 命令では使用できない。

FPMADDX, FPMADDXHI 命令の SIMD 演算での XAR.urs1, XAR.urs2, XAR.urs3, XAR.urd の意味をまとめると以下のようになる。

- XAR.urs1<2> rs1 に使うレジスタの Element 番号 LSB 1bit を反転
- XAR.urs2<2> rs2 に使うレジスタの Element 番号 LSB 1bit を反転
- XAR.urs3<2> rs1 に使うレジスタの Odd 側 Element 番号のみ LSB 1bit を反転

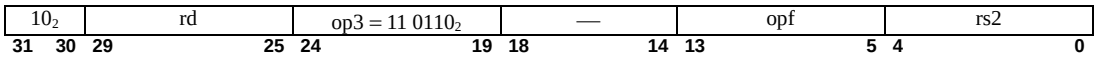
仮想的な言語を使用した動作記述（SIMD）

```
IF(XASR.simd_mode==0)
  SIMD_WIDTH ← 2
ELSEIF(XASR.simd_mode==1)
 SIMD_WIDTH ← 4
FPR_WIDTH ← 4

##FPMADDX
RS1_ELE_XOR<1:0> ← {1'b0 , XAR.urs1<2>}
RS2_ELE_XOR<1:0> ← {1'b0 , XAR.urs2<2>}
RS_COPY<1:0> ← {1'b0 , XAR.urs3<2>}

FOR (ELEMENT ← 0; ELEMENT<SIMD_WIDTH; ELEMENT++){
  RS1_ELE ← ELEMENT ^ RS1_ELE_XOR
  IF (ELEMENT<0> == 1){
    RS2_ELE ← ELEMENT ^ RS2_ELE_XOR ^ RS_COPY
  ELSE
    RS2_ELE ← ELEMENT ^ RS2_ELE_XOR

  TMP_RD<127:0> ←  F[rs1][RS1_ELE] * F[rs2][RS2_ELE] + F[rs3][ELEMENT]
  F[rd][ELEMENT] ← TMP_RD<63:0>
}

FOR (ELEMENT ← SIMD_WIDTH; ELEMENT<FPR_WIDTH; ELEMENT++)
  F[rd][ELEMENT]<63:0> ← UNDEFINED


##FPMADDXHI
RS1_ELE_XOR<1:0> ← {1'b0 , XAR.urs1<2>}
RS2_ELE_XOR<1:0> ← {1'b0 , XAR.urs2<2>}
RS_COPY<1:0> ← {1'b0 , XAR.urs3<2>}

FOR (ELEMENT ← 0; ELEMENT<SIMD_WIDTH; ELEMENT++){
  RS1_ELE ← ELEMENT ^ RS1_ELE_XOR
  IF (ELEMENT<0> == 1){
    RS2_ELE ← ELEMENT ^ RS2_ELE_XOR ^ RS_COPY
  ELSE
    RS2_ELE ← ELEMENT ^ RS2_ELE_XOR

  TMP_RD<127:0> ←  F[rs1][RS1_ELE] * F[rs2][RS2_ELE] + F[rs3][ELEMENT]
  F[rd][ELEMENT] ← TMP_RD<127:64>
}

FOR (ELEMENT ← SIMD_WIDTH; ELEMENT<FPR_WIDTH; ELEMENT++)
  F[rd][ELEMENT]<63:0> ← UNDEFINED
```

例外	対象命令	検出条件
fp_disabled	すべて	PSTATE.pef = 0 または FPRS.fef = 0
illegal_action	すべて	XAR.v = 1 かつ XAR.simd = 1 かつ XAR.urd<2> ≠ 0 のとき

7.38. Floating-Point Reciprocal Approximation

命令	opf	操作	HPC-ACE2 Regs	HPC-ACE2 SIMD	アセンブラ言語表記	
FRCPAd	$1\ 0111\ 0100_2$	倍精度逆数の近似値	✓	✓	`frcpad`	$freg_{rs2}$, $freg_{rd}$
FRCPAs	$1\ 0111\ 0101_2$	単精度逆数の近似値	✓	✓	`frcpas`	$freg_{rs2}$, $freg_{rd}$
FRSQRTAd	$1\ 0111\ 0110_2$	倍精度平方根逆数の近似値	✓	✓	`frsqrtad`	$freg_{rs2}$, $freg_{rd}$
FRSQRTAs	$1\ 0111\ 0111_2$	単精度平方根逆数の近似値	✓	✓	`frsqrtas`	$freg_{rs2}$, $freg_{rd}$



動作説明

FRCPA{s|d}は、F[rs2]の値の逆数近似値を求め、F[rd]に格納する。得られる結果は近似値だが、丸めモード設定の影響は受けない。結果が正規化数のとき、その精度は 1/256 未満、つまり

$$\left|\frac{frcpa(x)-1/x}{1/x}\right| < \frac{1}{256}$$

となる。

FRCPA{s|d}の演算結果と例外条件を表 7-17 に示す。結果の欄の上段が例外、下段は例外が通知されないときの値である。*fp_exception_ieee_754* 例外の要因が複数あるものの優先順位は、本書および JPS1 Commonality の Appendix B を参照。

Compatibility Note SPARC64™ XIfx では、SPARC64™VIIIfx, SPARC64™ IXfx から FRSQRTAs, FRSQRTAd 命令の動作を変更した。

表 7-17　FRCPA{s|d}の演算結果

op2	例外と演算結果	
	FSR.ns = 0	FSR.ns = 1
+∞	— 0	— 0
+N　(単精度では $N \geq 2^{126}$ 倍精度では $N \geq 2^{1022}$)	UF[xxi] +1/N の近似値 (非正規化数)[xxii]	UF, NX +0
+N　(単精度では$+N_{min} \leq N < 2^{126}$ 倍精度では$+N_{min} \leq N < 2^{1022}$	— +1/N の近似値	— +1/N の近似値
+D	*unfinished_FPop* —	DZ +∞
+0	DZ +∞	DZ +∞
−∞	DZ −∞	DZ −∞
−D	*unfinished_FPop* —	DZ −∞
−N　(単精度では$+N_{min} \leq N < 2^{126}$ 倍精度では$+N_{min} \leq N < 2^{1022}$	— −1/N の近似値	— −1/N の近似値
−N　(単精度では $N \geq 2^{126}$ 倍精度では $N \geq 2^{1022}$)	UF[xxi] −1/N の近似値 (非正規化数) [xxiii]	UF, NX +0
−∞	— −0	— −0
SNaN	NV QSNaN2	NV QSNaN2
QNaN	— op2	— op2

N	正の正規化数(ゼロ、NaN, 無限大を除く)
D	正の非正規化
Nmin	正の正規化数の最小値
dNaN	符号は 0, 指数部と仮数部が全ビット 1 の QNaN
QSNaN2	JPS1 Commonality の TABLE B-1 参照

FRSQRTA{s|d}は、F[rs2]の平方根の逆数近似値を求め、F[rd]に格納する。得られる結果は近似値だが、丸めモード設定の影響を受けない。結果が正規化数のとき、近似値の精度は 1/256 未満、つまり

$$\left| \frac{frsqrta(x) - 1/\left(\sqrt{x}\right)}{1/\left(\sqrt{x}\right)} \right| < \frac{1}{256}$$

となる。

FRSQRTA{s|d}の演算結果と例外条件を表 7-18 に示す。結果の欄の上段が例外、下段は例外が通知されないときの値である。*fp_exception_ieee_754* 例外の要因が複数あるものの優先順位は本書および JPS1 Commonality の Appendix B を参照。

[xxi] FSR.tem.ufm = 0 の場合も NX は伴わない
[xxii] 結果が非正規化数のとき、精度は 1/256 より大きくなり得る。
[xxiii] 結果が非正規化数のとき、精度は 1/256 より大きくなり得る。

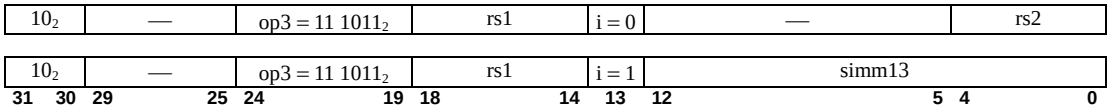
表 7-18　FRSQRTA{s | d}の演算結果

op2	例外と演算結果	
	FSR.ns = 0	FSR.ns = 1
+∞	— +∞	— +∞
+N	— $+1/\left(\sqrt{N}\right)$	— $+1/\left(\sqrt{N}\right)$
+D	*unfinished_FPop* —	— +0
+0	— +0	— +0
–0	— +0	— +0
–D	NV dNaN	NV dNaN
–N	NV dNaN	NV dNaN
–∞	NV dNaN	NV dNaN
SNaN	NV QSNaN2	NV QSNaN2
QNaN	— op2	— op2

例外		対象命令	検出条件
illegal_instruction		すべて	*reserved* フィールドが 0 でない (iw<18:14> ≠ $0\ 0000_2$)
fp_disabled		すべて	PSTATE.pef = 0 または FPRS.fef = 0
illegal_action		すべて	XAR.v = 1 かつ、下記のいずれかが成立している場合 • XAR.urs1 ≠ 0 • XAR.urs3 ≠ 0 • XAR.simd = 1 かつ XAR.urs2<2> ≠ 0 • XAR.simd = 1 かつ XAR.urd<2> ≠ 0
fp_exception_ieee_754	NV, DZ, UF, NX	FRCPAs, FRCPAd,	IEEE 754 に準拠
	NV,	FRSQRTs, FRSQRTAd	IEEE 754 に準拠
fp_exception_other (FSR.ftt = *unfinished_FPop*)		すべて	

7.39. Flush Instruction Memory

命令	op3	操作	HPC-ACE2 Regs	HPC-ACE2 SIMD	アセンブリ言語表記	
FLUSH	$11\ 1011_2$	命令メモリのフラッシュ	✓		flush	[*address*]



Note FLUSH 命令の仕様は、JPS1 と UA2011 で微妙に異なっている。両仕様に差異がある部分には注釈をつける。SPARC64™ XIfx は主として JPS1 仕様に準拠する。

動作説明

FLUSH 命令は、命令で指定されたアドレスを含む 8 バイトアラインされた 8 バイトが、CPU チップ内すべてのキャッシュで同一の値になることを保証する。全 VCPU で全キャッシュが同一の値になる。

FLUSH 命令が作用するアドレスは、i = 0 のときは “R[rs1] + R[rs2]” で、i = 1 のときは、“R[rs1] + sign_ext(simm13)” で計算される。どちらの場合も下位 3 ビットは無視され、常に 8 バイトアラインのアドレスが使用される。

Compatibility Note JPS1 仕様、UA2011 仕様とも下位 3 ビットは無視されるが、JPS1 仕様ではソフトウェアに対し、下位 2 ビットに 0 を指定するよう要請している。

SPARC V9 仕様では、命令用メモリ [xxiv]とデータ用メモリの間の同一性を保証していない。ソフトウェアが、命令として実行される可能性のある領域に対して書き込み [xxv]（自己書き換え）を行うと、潜在的なメモリの矛盾という問題が起こりうる。FLUSH 命令はこれを解決する命令である。命令用メモリの書き換えの後 FLUSH 命令を実行することで、命令用メモリとデータ用メモリの間の同一性が保証される。

VCPU は先行する（キャッシャブル）ストアがすべて完了するのを待ってから FLUSH 命令を発行する。メモリオーダリングの観点では、FLUSH 命令はストアと同様に扱われる。

SPARC64™ XIfx は、命令キャッシュとデータキャッシュの値の同一性を保証しているので、FLUSH 命令を実行しなくても、両方のキャッシュが持つ値はいずれ同一になる。したがって命令で指定されるアドレスは使用せず、アドレスに関連する例外も発生しない。しかし、命令をアウトオブオーダで実行する SPARC64™ XIfx では、命令キャッシュが更新される前の命令がパイプラインに送り込まれることはありうる。SPARC64™ XIfx の FLUSH 命令は、パイプライン中の命令についても同一性を保証するためのパイプラインフラッシュを行う。

また、FLUSH 命令はメモリへのアクセスを行う命令であるが、セクタキャッシュの明示的な指定はできない。

Programming Note すべての SPARC V9 プロセッサでの互換性を保障するために、FLUSH 命令には有効な実アドレスを指定するべきである。

以後の説明では、P_{FLUSH} は FLUSH 命令を実行した VCPU を意味する。

[xxiv] ここではキャッシュを意味している。
[xxv] 自 VCPU、他 VCPU によるストア命令の実行や、DMA その他いかなる意味の書き込みも含む。

FLUSH 命令は VCPU 内の同期を提供する。この同期は、P_{FLUSH} による特定アドレスの命令フェッチは、P_{FLUSH} で FLUSH 命令より前に実行されたすべてのロード、ストア、およびアトミック命令が完了してから行われることを保証する。複数の CPU チップが搭載されたシステムでは、FLUSH 命令は、他の CPU チップの命令フェッチにおいてもいずれは P_{FLUSH} が命令フェッチするものと同じ値（命令語）が見えるようになることを保証する。メモリバリアの観点では、FLUSH 命令はストアと同様の働きをする。

アドレス A に対するストア S_A があり、メモリオーダ観点でその後に FLUSH F_A が続き、さらにメモリオーダ観点でその後にストア S_B が続くとする。もし、アドレス B からフェッチされた命令 I_B がストア S_B によって書き換えられた命令だったとすると、プログラムオーダ的にその後にアドレス A からフェッチされる命令 I_A は、ストア S_A により書き換えられる前のアドレス A の内容を実行することはありえない。

上の文は、UltraSPARC Architecture 仕様に準拠するプロセッサが満たすべきオーダリング要件である。命令を書き換えるストア命令 2 つの間に FLUSH 命令をはさむことで、そのストア間のアトミシティが保証される。すなわち、ある VCPU が後のストア命令によって変更された命令を実行したとすると、その VCPU は前のストアによって変更される前の命令を実行することはありえないことが保証される。

Programming Note

1. FLUSH 命令は、自己書き換えコードで使われるのが典型例である。しかし、自己書き換えコードの使用は薦めない。

2. 自己書き換えコードを含むプログラムがどのプロセッサでも動くようにするためには、すべての 8 バイト単位の書き込みごとに FLUSH 命令を実行する（またはそれと同等の効果を持つ特権コードの呼び出しを行う）こと。

3. メモリ書き換えの順序はストア命令とアトミック命令の間に FLUSH 命令と MEMBAR 命令を適切に配置することで制御できる。FLUSH 命令はストア命令とその後その変更されたデータを命令フェッチする場合にのみ必要である。複数のプロセスが同時に生きている（実行される可能性のある）コードを変更するかもしれない場合、プログラマは、変更の順序がプログラムを意義的に正しい状態に維持することを保証しなければならない。

4. SPARC64™ XIfx のメモリモデルは、1 プロセッサの場合、FLUSH 命令がなくても、ストア後のデータロードは必ず新しいデータが見えることを保証している。

5. FLUSH 命令は実行に時間がかかる命令である。

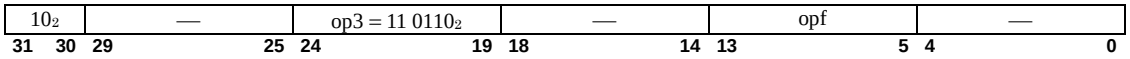
6. 複数プロセッサからなるシステムでは、FLUSH 命令の結果が global visible になるのは、それに続くストア命令の結果が global visible になる前である。

7. FLUSH 命令は 8 バイト境界の 8 バイトデータに対して作用するものと定義されているので、システムソフトウェアは、非特権ソフトウェア用に任意サイズのフラッシュを行うルーチンを用意する必要がある。

例外	検出条件
illegal_instruction	*reserved* フィールドが 0 でない
illegal_action	XAR.v = 1 かつ、下記のいずれかが成立している場合 • XAR.simd = 1 • XAR.urs1<2:1> ≠ 00_2 • i = 0 かつ XAR.urs2<2:1> ≠ 00_2 • i = 1 かつ XAR.urs2 ≠ 0 • XAR.urs3 ≠ 0 • XAR.urd ≠ 0

7.40. Sleep

命令	Opf	操作	HPC-ACE2 Regs.	HPC-ACE2 SIMD	アセンブリ言語表記
SLEEP	$0\ 1000\ 0011_2$	VCPU を一定時間停止させる			sleep



SLEEP 命令は、ペンディングされている例外がないとき、VCPU を一定時間停止させる。

実行を停止している VCPU はまた、以下のいずれかの条件により実行を再開する。

- 実装により決まる一定時間経過後。
- 例外が発生、またはペンディングされたとき。
- バリアの窓 ASI が BB に割り当てられており、その BB の LBSY が更新されたとき。

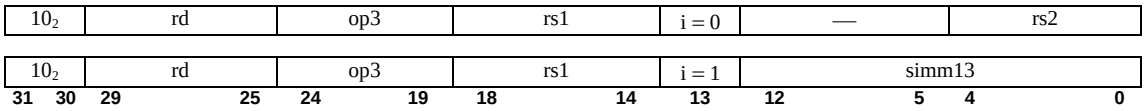
Programming Note ソフトウェアは、SLEEP 命令が常に一定時間停止することを期待してはいけない。

Compatibility Note SPARC64™ IXfx までは例外を検出したときに実行を再開することになっていたが、SPARC64™ XIfx では例外を検出するような事象で実行再開する(例外発生ではないことに注意)。

例外	命令	検出条件
illegal_instruction	すべて	*reserved* が 0 でない
illegal_action	すべて	XAR.v = 1

7.41. ADD

命令	op3	操作	HPC-ACE2 Regs.	HPC-ACE2 SIMD	アセンブリ言語表記	
ADD	$00\ 0000_2$	整数加算	✓		add	*reg$_{rs1}$, reg_or_imm, reg$_{rd}$*
ADDcc	$01\ 0000_2$	整数加算と cc の更新	✓		addcc	*reg$_{rs1}$, reg_or_imm, reg$_{rd}$*
ADDC	$00\ 1000_2$	キャリーを含む整数加算	✓		addc	*reg$_{rs1}$, reg_or_imm, reg$_{rd}$*
ADDCcc	$01\ 1000_2$	キャリーを含む整数加算と cc の更新	✓		addccc	*reg$_{rs1}$, reg_or_imm, reg$_{rd}$*



動作説明

i = 0 のとき、ADD と ADDcc は、“R[rs1] + R[rs2]”を処理する。i = 1 のときは、“R[rs1] + sign_ext(simm13)”を処理する。どちらの場合も合計は、R[rd]に書き込まれる。

32 ビット演算結果のキャリーを入力とする ADD（ADDC と ADDCcc）では、CCR レジスタのキャリービット(icc.c)も加える。すなわち、i = 0 のときは、“R[rs1] + R[rs2] + icc.c”を処理し、i = 1 のときは、“R[rs1] + sign_ext(simm13) + icc.c”を処理する。どちらの場合も合計は、R[rd]に書き込まれる。

ADDcc と ADDCcc は、整数条件コード（CCR.icc と CCR.xcc）を変更する。加算において、両方のオペランドが同じ符号であり、その合計の符号がオペランドの符号と異なる場合には、オーバーフローが生じる。

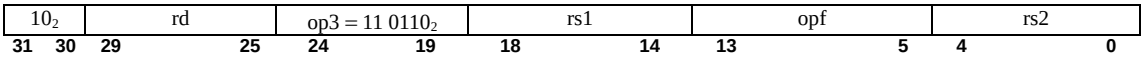
Programming Note ADDC と ADDCcc は、32 ビット条件コードのキャリービット（CCR.icc.c）を読み出す。64 ビット条件コードのキャリービット（CCR.xcc.c）は読み出さない。

Compatibility Note ADDC と ADDCcc は、SPARC V8 では、それぞれ、ADDX, ADDXcc と名づけられていた。

例外	対象命令	検出条件
illegal_instruction	すべて	*reserved* フィールドが 0 でない (i = 0 かつ iw<12:5> = $0000\ 0000_2$)
illegal_action	すべて	XAR.v = 1 かつ、下記のいずれかが成立している場合 • XAR.simd = 1 • XAR.urs1<2:1> ≠ 00_2 • i = 0 かつ XAR.urs2<2:1> ≠ 00_2 • i = 1 かつ XAR.urs2 ≠ 0 • XAR.urs3 ≠ 0 • XAR.urd<2:1> ≠ 00_2

7.42. Align Address

命令	opf	操作	HPC-ACE2	アセンブリ言語表記
			Regs. SIMD	
ALIGNADDRESS	0 0001 1000$_2$	アラインされていないデータにアクセスするためにアドレスを計算する		alignaddr *reg$_{rs1}$, reg$_{rs2}$, reg$_{rd}$*
ALIGNADDRESS_LITTLE	0 0001 1010$_2$	アラインされていないリトルエンディアン型データにアクセスするためにアドレスを計算する		alignaddrl *reg$_{rs1}$, reg$_{rs2}$, reg$_{rd}$*



動作説明

ALIGNADDRESS は、２つの整数値 R[rs1]と R[rs2] を加算し、加算結果の低位 3 ビットを 0 にして、整数レジスタ R[rd]に格納する。さらに、結果の低位 3 ビットは、GSR.align フィールドに格納される。

ALIGNADDRESS_LITTLE は、加算結果の低位 3 ビットの 2 の補数値を GSR.align フィールドに格納する。それ以外の処理は、ALIGNADDRESS と同じである。

> **Note** ALIGNADDRESS_LITTLE は、後続の FALIGNDATA 処理のために、反対のエンディアンバイト順序を生成する。

バイトアラインされた 64 ビットロードは、以下のように行うことができる。

```
/* GSR.align にオフセットを格納 */
alignaddr Address, Offset, Address
/* アラインされていないデータを含む 16B のデータをロードする */
ldd    [Address], %d0
ldd    [Address + 8], %d2
/* 16B データから、GSR.align （オフセット）B 分ずらした 8B データを抜き出す */
faligndata   %d0, %d2, %d4
```

例外	対象命令	検出条件
fp_disabled	すべて	FPRS.fef = 0 または PSTATE.pef = 0
illegal_action	すべて	XAR.v = 1

7.43. Three-Dimensional Array Addressing

ARRAY8, ARRY16, ARRAY32 は UA2011 7.8 参照

例外	対象命令	検出条件
illegal_action	すべて	XAR.v = 1

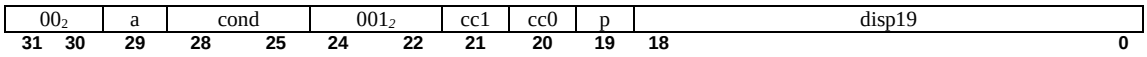
7.44. Byte Mask and Shuffle

BMASK, BSHUFFLE は UA2011 7.10 参照

例外	対象命令	検出条件
fp_disabled	すべて	FPRS.fef = 0 または PSTATE.pef = 0
illegal_action	すべて	XAR.v = 1

7.45. Branch on Integer Condition Codes with Prediction (BPcc)

命令	cond	分岐条件	テスト (icc or xcc)	HPC-ACE2 Regs.	HPC-ACE2 SIMD	アセンブリ言語表記	
BPA	1000_2	常に分岐する	1			`ba{,a}{,pt|,pn}`	*i_or_x_cc, label*
BPN	0000_2	決して分岐しない	0			`bn{,a}{,pt|,pn}`	*i_or_x_cc, label*
BPNE	1001_2	等しくないとき	**not** Z			`bne{,a}{,pt|,pn}` `bnz{,a}{,pt|,pn}`	*i_or_x_cc, label* *i_or_x_cc, label*
BPE	0001_2	等しいとき	Z			`be{,a}{,pt|,pn}` `bz{,a}{,pt|,pn}`	*i_or_x_cc, label* *i_or_x_cc, label*
BPG	1010_2	より大きいとき	**not** (Z **or** (N **xor** V))			`bg{,a}{,pt|,pn}`	*i_or_x_cc, label*
BPLE	0010_2	以下のとき	Z **or** (N **xor** V)			`ble{,a}{,pt|,pn}`	*i_or_x_cc, label*
BPGE	1011_2	以上のとき	**not** (N **xor** V)			`bge{,a}{,pt|,pn}`	*i_or_x_cc, label*
BPL	0011_2	より小さいとき	N **xor** V			`bl{,a}{,pt|,pn}`	*i_or_x_cc, label*
BPGU	1100_2	符号なし整数で、より大きいとき	**not** (C **or** Z)			`bgu{,a}{,pt|,pn}`	*i_or_x_cc, label*
BPLEU	0100_2	符号なし整数で、以下のとき	C **or** Z			`bleu{,a}{,pt|,pn}`	*i_or_x_cc, label*
BPCC	1101_2	キャリークリア（符号なし整数で以上）のとき	**not** C			`bcc{,a}{,pt|,pn}` `bgeu{,a}{,pt|,pn}`	*i_or_x_cc, label* *i_or_x_cc, label*
BPCS	0101_2	キャリーセット（符号なし整数でより小さい）のとき	C			`bcs{,a}{,pt|,pn}` `blu{,a}{,pt|,pn}`	*i_or_x_cc, label* *i_or_x_cc, label*
BPPOS	1110_2	正のとき	**not** N			`bpos{,a}{,pt|,pn}`	*i_or_x_cc, label*
BPNEG	0110_2	負のとき	N			`bneg{,a}{,pt|,pn}`	*i_or_x_cc, label*
BPVC	1111_2	オーバーフローしていないとき	**not** V			`bvc{,a}{,pt|,pn}`	*i_or_x_cc, label*
BPVS	0111_2	オーバーフローのとき	V			`bvs{,a}{,pt|,pn}`	*i_or_x_cc, label*



cc1	cc0	条件コード
0	0	CCR.icc
0	1	—
1	0	CCR.xcc
1	1	—

Programming Note BPcc 命令に対して無効ビット(a ビット)をセットする場合は、“,a”をオペコードのニモニックに付加する。例えば、“bgu,a *%icc,label*” とする。上記フィールドの説明において、アセンブラ言語表記で{,a}となっているのは、無効ビットのセットが任意である事を表す。分岐予測ビットに関しては、分岐すると予測する場合は “,pt”を、分岐しないと予測する場合は“,pn”をオペコードのニモニックに付加する(どちらも付加されていないならば、アセンブラは “,pt”と解釈する)。適切な整数条件コードを選択するためには、ラベルの前に“*%icc*” または“*%xcc*” を含めること。

動作説明

無条件分岐と条件分岐は以下に記述した:

- 無条件分岐(BPA, BPN)—この分岐タイプ(op2 = 1)に対して BPN(予測付きで決して分岐しない)命令は、命令プリフェッチとして SPARC V9 アーキテクチャで使われるかもしれない ; すなわち、有効アドレス(PC + (4 × sign_ext(disp19)))は、すぐに実行される事を期待される命令のアドレスを明示する。もし、BPN の無効ビットが 1(a = 1)であるならば、次の命令(delay 命令)は、無効とされる(実行されない)。もし、無効ビットが 0 (a = 0)ならば、次の命令(delay 命令)は実行される。BPN では、決して制御転送先には分岐しない。

  BPA(予測付きでいつも分岐)は、アドレス “PC + (4 × sign_ext(disp19))” へ無条件に PC 相対遅延制御転送を生じる。もし、分岐命令の無効(a)ビットが 1 ならば、遅延命令は無効とされる(実行されない)。もし、無効ビットが 0 (a = 0)ならば、遅延命令は実行される。

- 条件分岐—(BPA と BPN を除く)条件 BPcc 命令は、命令の cond フィールドに従い、cc0 と cc1 によって選択される二つの整数条件コード(CCR.icc または CCR.xcc)のひとつを評価し、真または偽のどちらかを結果として生成する。もし、真であるならば分岐が行われる、すなわち命令は、アドレス “PC + (4 × sign_ext(disp19))” へ PC 相対遅延制御転送を生じる。もし、偽であるならば分岐は行われない。

  もし、条件分岐が行われるならば、遅延命令は無効フィールドの値に関わらず、いつも実行される。もし、条件分岐が行われない、かつ、無効ビットが 1(a = 1)であるならば、遅延命令は無効とされる(実行されない)。

Note 無効ビットは、無条件分岐と条件分岐では異なる作用をもつ。

予測ビット(p)は、分岐が行われると期待されるかどうかについてハードウェアにヒントを与えるために使われる。p ビットが 1 である事は、分岐が行われると期待されている事を示す。p ビットが 0 である事は、分岐が行われないと期待されている事を示す。

SPARC64™ XIfx では、制御転送機構におけるトラップが実装されている(page363)。そのため、PSTATE.tct = 1 であり、BPcc 命令が制御転送(BPA または taken となる条件分岐)を生じるならば、そのとき BPcc は、制御転送を起こさずに *control_transfer_instruction* 例外を生成する。*control_transfer_instruction* トラップが生じるとき、PC(BPcc 命令のアドレス)は TPC[TL]に保存され、BPcc が実行される前の NPC の値は TNPC[TL]に保存される。

BPN は、*control_transfer_instruction* 例外を決して生じない事に注意すること。

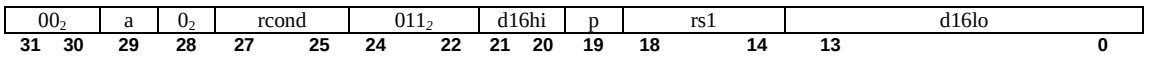
例外	対象命令	検出条件
illegal_instruction	すべて	*reserved* フィールドが 0 でない (cc0 = 1)
illegal_action	すべて	XAR.v = 1
control_transfer_instruction	BPN 以外	条件分岐が成立し、PSTATE.tct = 1 の場合。 BPA 命令は常に条件分岐が成立しているとする。

関連項目

Branch on Integer Register with Prediction (BPr) (page 180)

7.46. Branch on Integer Register with Prediction (BPr)

命令	rcond	分岐条件	HPC-ACE2 Regs.	HPC-ACE2 SIMD	アセンブリ言語表記	
—	000_2	*reserved*	—			
BRZ	001_2	0 のとき	R[rs1] = 0		`brz{,a}{,pt|,pn}`	*reg_{rs1}, label*
BRLEZ	010_2	0 以下	R[rs1] ≤ 0		`brlez{,a}{,pt|,pn}`	*reg_{rs1}, label*
BRLZ	011_2	0 より小さいとき	R[rs1] < 0		`brlz{,a}{,pt|,pn}`	*reg_{rs1}, label*
—	100_2	*reserved*	—			
BRNZ	101_2	0 以外	R[rs1] ≠ 0		`brnz{,a}{,pt|,pn}`	*reg_{rs1}, label*
BRGZ	110_2	0 より大きいとき	R[rs1] > 0		`brgz{,a}{,pt|,pn}`	*reg_{rs1}, label*
BRGEZ	111_2	0 以上	R[rs1] ≥ 0		`brgez{,a}{,pt|,pn}`	*reg_{rs1}, label*



Programming Note BPr 命令に対して無効ビット（a ビット）をセットする場合は、“,a”をオペコートのニモニックに付加する。例えば、“`brz,a` *%i3, label*” とする。上記のフィールドの説明において、アセンブラ言語表記で{,a}となっているのは、無効ビットのセットが任意である事を表す。分岐予測ビットに関しては、分岐すると予測する場合は “`,pt`”を、分岐しないと予測する場合は “`,pn`”をオペコードのニモニックに付加する（どちらも付加されていない場合は、アセンブラは、“`,pt`”と解釈する）。

動作説明

これらの命令は、R[rs1]の内容に基づいて分岐する。また、R[rs1]の内容は、符号付整数値として扱われる。

BPr 命令は、命令の rcond フィールドにしたがって R[rs1]の全 64 ビットを評価し、真または偽のどちらかの結果を生成する。もし、真であるならば分岐が行われる、すなわち命令は、アドレス “PC + (4 × sign_ext(d16hi::d16lo))” へ PC 相対遅延制御転送を生じる。もし、偽であるならば分岐は行われない。

もし、分岐が行われるならば、遅延命令は無効(a)ビットの値に関わらず、いつも実行される。もし、分岐が行われない、かつ、無効ビットが 1(a = 1)であるならば、遅延命令は無効とされる(実行されない)。

予測ビット(p)は、分岐が行われると期待されるかどうかについてハードウェアにヒントを与えるために使われる。p ビットが 1 である事は、分岐が行われると期待されている事を示す。p ビットが 0 である事は、分岐が行われないと期待されている事を示す。

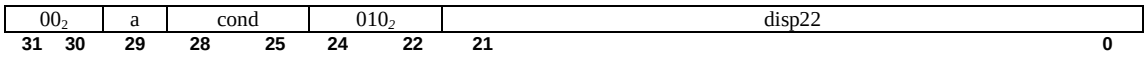
SPARC64™ XIfx では、制御転送機構におけるトラップが実装されている(page363)。そのため、PSTATE.tct = 1 であり、BPr 命令が制御転送(taken となる条件分岐)を生じるならば、そのとき BPr は、制御転送を起こさずに *control_transfer_instruction* トラップを生成する。

例外	対象命令	検出条件
illegal_instruction	すべて	*reserved* フィールドが 0 でない (rcond = (000_2 または 100_2))
illegal_action	すべて	XAR.v = 1
control_transfer_instruction	すべて	条件分岐が成立し、PSTATE.tct = 1 の場合。

関連項目　Branch on Integer Condition Codes with Prediction (BPcc) (page 178)

7.47. Branch on Integer Condition Codes (Bicc)

命令	cond	分岐条件	テスト (icc)	HPC-ACE2 Regs.	HPC-ACE2 SIMD	アセンブリ言語表記	
BA	1000_2	常に分岐する	1			ba{,a}	*label*
BN	0000_2	決して分岐しない	0			bn{,a}	*label*
BNE	1001_2	等しくないとき	**not** Z			bne{,a} bnz{,a}	*label* *label*
BE	0001_2	等しいとき	Z			be{,a} bz{,a}	*label* *label*
BG	1010_2	より大きいとき	**not** (Z **or** (N **xor** V))			bg{,a}	*label*
BLE	0010_2	以下のとき	Z **or** (N **xor** V)			ble{,a}	*label*
BGE	1011_2	以上のとき	**not** (N **xor** V)			bge{,a}	*label*
BL	0011_2	より小さいとき	N **xor** V			bl{,a}	*label*
BGU	1100_2	符号なし整数で、より大きいとき	**not** (C **or** Z)			bgu{,a}	*label*
BLEU	0100_2	符号なし整数で、以下のとき	C **or** Z			bleu{,a}	*label*
BCC	1101_2	キャリークリア（符号なし整数で以上）のとき	**not** C			bcc{,a} bgeu{,a}	*label* *label*
BCS	0101_2	キャリーセット（符号なし整数でより小さい）のとき	C			bcs{,a} blu{,a}	*label* *label*
BPOS	1110_2	正のとき	**not** N			bpos{,a}	*label*
BNEG	0110_2	負のとき	N			bneg{,a}	*label*
BVC	1111_2	オーバーフローしていないとき	**not** V			bvc{,a}	*label*
BVS	0111_2	オーバーフローのとき	V			bvs{,a}	*label*



Programming Note Bicc 命令に対して無効ビット（a ビット）をセットする場合は、“,a”をオペコードのニモニックに付加する。例えば、“bgu,a *label*”とする。上記のフィールドの説明において、アセンブラ言語表記で{,a}となっているのは、無効ビットのセットが任意である事を示す。

動作説明

無条件分岐と icc 条件分岐は、以下に記述した:

- 無条件分岐(BA, BN)—もし、無効ビットが 0 (a = 0)ならば、BN(決して分岐しない)命令は、NOP として扱われる。もし、無効ビットが 1 (a = 1)ならば、次の命令(delay 命令)は、無効とされる(実行されない)。どちらの場合も、制御転送先には分岐しない。

  BA(いつも分岐)は、アドレス “PC + (4 × sign_ext(disp22))” へ無条件に PC 相対遅延制御転送を生じる。もし、分岐命令の無効(a)ビットが 1 ならば、遅延命令は無効とされる(実行されない)。もし、無効ビットが 0 (a = 0)ならば、遅延命令は実行される。

- icc 条件分岐—条件 Bicc 命令(BA と BN を除く全て)は、真または偽の結果のどちらかを生成するために命令の cond フィールドにしたがって、32 ビット整数条件コード(CCR.icc)を評価する。もし、真であるならば分岐が行われる、すなわち命令は、アドレス “PC + (4 × sign_ext(disp22))” へ PC 相対遅延制御転送を生じる。もし、偽であるならば分岐は行われない。

もし、条件分岐が行われるならば、遅延命令は無効フィールドの値に関わらず、いつも実行される。もし、条件分岐が行われない、かつ、無効ビットが 1(a = 1)であるならば、遅延命令は無効とされる(実行されない)。

Note 無効ビットは、無条件分岐と条件分岐では異なる作用をもつ。

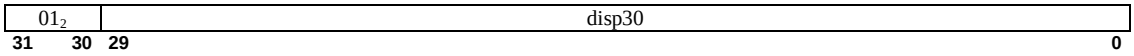
SPARC64™ XIfx では、制御転送機構におけるトラップが実装されている(page363)。そのため、PSTATE.tct = 1 であり、`Bicc` 命令が制御転送(BA または taken となる条件分岐)を生じるならば、そのとき `Bicc` は、制御転送を起こさずに *control_transfer_instruction* 例外を生成する。*control_transfer_instruction* トラップが生じるとき、PC(`Bicc` 命令のアドレス)は TPC[TL]に保存され、`Bicc` が実行される前の NPC の値は TNPC[TL]に保存される。

BN は、*control_transfer_instruction* 例外を決して生じない事に注意すること。

例外	対象命令	検出条件
illegal_action	すべて	XAR.v = 1
control_transfer_instruction	BN 以外	条件分岐が成立し、PSTATE.tct = 1 の場合。 BA 命令は常に条件分岐が成立しているとする。

7.48. Call and Link

命令	op	操作	HPC-ACE2 Regs.	HPC-ACE2 SIMD	アセンブリ言語表記	
CALL	01_2	関数コール			call	*label*



動作説明

CALL 命令は遅延スロットを有す PC 相対無条件ジャンプ命令である。CCR とは無関係にカレント PC からの相対アドレス、PC + (4 × sign_ext(disp30))に、PC を更新する。disp30 は 30 ビット長のフィールドを持ち、相対アドレスの範囲は-2^{31}から$2^{31} - 4$バイトである。相対アドレスは、30 ビット長の disp30 フィールドを 62 ビットに符号拡張し、下位 2 ビットにゼロを追加し 64 ビット長とする。

PC の更新と同時にカレント PC を R[15]レジスタ(out register 7)にコピーする。

PSTATE.am = 1 のとき、R[15]には PC の上位 32 ビットを 0 にした値が設定される。R[15]の更新は直ちに行われ、ディレイスロットの命令には新しい値が見える。

SPARC64™ XIfx では、制御転送機構におけるトラップが実装されている(page363)。そのため、PSTATE.tct = 1 であるならば、CALL 命令は制御転送を起こさずに *control_transfer_instruction* 例外を生成する。*control_transfer_instruction* トラップが生じるとき、PC(CALL 命令のアドレス)は TPC[TL]に保存され、CALL 命令が実行される前の NPC の値は TNPC[TL]に保存される。

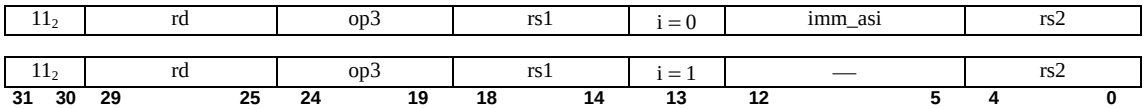
例外	対象命令	検出条件
illegal_action	すべて	XAR.v = 1
control_transfer_instruction	すべて	PSTATE.tct = 1

関連項目

JMPL（238 ページ）

7.49. Compare and Swap

命令	op3	操作	HPC-ACE2		アセンブリ言語表記	
			Regs.	SIMD		
CASA$^{P_{ASI}}$	$11\ 1100_2$	別空間に対する 32 ビットデータの比較交換	✓		casa casa	*[reg$_{rs1}$] imm_asi, reg$_{rs2}$, reg$_{rd}$* *[reg$_{rs1}$] %asi, reg$_{rs2}$, reg$_{rd}$*
CASXA$^{P_{ASI}}$	$11\ 1110_2$	別空間に対する 64 ビットデータの比較交換	✓		casxa casxa	*[reg$_{rs1}$] imm_asi, reg$_{rs2}$, reg$_{rd}$* *[reg$_{rs1}$] %asi, reg$_{rs2}$, reg$_{rd}$*



動作説明

並列プロセスが同期とメモリ更新のためにこの命令を使う。比較交換の活用は、スピンロック処理、共有コンテンツの更新、連結リストポインタの更新を含む。共有コンテンツの更新と連結リストポインタの更新は、無待機(ノンロッキング)処理プロトコルを使うことができる。

CASXA 命令は、R[rs2]の値と R[rs1]の 64 ビットアドレス値によって示されるメモリの 64 ビット値を比較する。もし、比較結果が等しいならば R[rd]の値が R[rs1]の 64 ビットアドレス値によって示されるメモリの 64 ビット値と交換される。もし、比較結果が等しくないならば、R[rs1]によって示されるメモリの値は R[rd]の値を置き換える、しかし、メモリの内容は、変更されずに残る。

CASA 命令は、R[rs2]の下位 32 ビット値と R[rs1]の 32 ビットアドレス値によって示されるメモリの 32 ビット値を比較する。もし、比較結果が等しいならば R[rd]の下位 32 ビット値が R[rs1]の 32 ビットアドレス値によって示されるメモリの 32 ビット値と交換される、このとき R[rd]の上位 32 ビット値は、0 にセットされる。もし、比較結果が等しくないならば、R[rs1]によって示されるメモリの 32 ビット値は R[rd]の下位 32 ビット値を置き換える、このとき R[rd]の上位 32 ビット値は、0 にセットされる、しかし、メモリの内容は、変更されずに残る。

比較交換命令は、３つの操作：ロード、比較、交換から構成される。全命令がアトミックである。すなわち、介入する割り込みはない、そして、遅延トラップは VCPU によって認識されない、比較交換、交換、ロード、ロードストアアンサインドバイト、ストア命令の結果に介入する更新はない。これらは、メモリシステムによって保障されている。

比較交換操作は、いくつかのメモリバリア動作を暗に含んではいない。比較交換が、同期のために使われるとき、あたかもロード、ストア、交換命令が使われたかのように、同じ配慮がメモリバリアに対して与えられるべきである。

比較交換操作は、まるで、R[rd]からの新しい値か、もしくはメモリの以前の値のどちらかをストアしたかのように動作する。メモリの値と R[rs2]の値が等しくないかどうかに関わらず、アドレスされた場所は書き込み可能でなければならない。

もし、i = 0 であるならば、メモリ配置のアドレス空間は、imm_asi フィールドで明示される。もし、i = 1 であるならば、アドレス空間は、ASI レジスタで明示される。

> **Programming Note** Compare and Swap (CAS) と Compare and Swap Extended (CASX) 合成命令は、“ビッグエンディアン”メモリアクセスに使用できる。

比較交換命令は、条件コードに作用しない。

比較交換命令は、*privileged_action* 例外に記述した特権モード規則に従い、以下の ASI で使うことができる。それ以外の ASI での使用は、*DAE_invalid_asi* 例外を生じる。

CASA と CASXA 命令で有効な ASI	
ASI_PRIMARY	ASI_PRIMARY_LITTLE
ASI_SECONDARY	ASI_SECONDARY_LITTLE

例外	対象命令	検出条件
illegal_instruction	すべて	*reserved* フィールドが 0 でない (i = 1 かつ iw<12:5> ≠ $0000\ 0000_2$)
illegal_action	すべて	XAR.v = 1 かつ、下記のいずれかが成立している場合 • XAR.simd = 1 • XAR.urs1<2:1> ≠ 00_2 • XAR.urs2<2:1> ≠ 00_2 • XAR.urd<2:1> ≠ 00_2
mem_address_not_aligned	CASXA	R[rs1]で示されるアドレス値が 8 バイト境界でないとき
	CASA	R[rs1]で示されるアドレス値が 4 バイト境界でないとき
privileged_action	すべて	PSTATE.priv = 0 のとき、下記のいずれかが成立している場合 • i = 0 かつ ASI<7> = 0 • i = 1 かつ imm_asi<7> = 0
	すべて	PSTATE.priv = 1 のとき、下記のいずれかが成立している場合 • i = 0 かつ 30_{16} ≤ ASI ≤ $7F_{16}$ • i = 1 かつ 30_{16} ≤ imm_asi ≤ $7F_{16}$
VA_watchpoint	すべて	
DAE_invalid_asi	すべて	
DAE_privilege_violation	すべて	
DAE_nc_page	すべて	ノンキャッシャブル空間にアクセスしたとき。
DAE_nfo_page	すべて	
DAE_side_effect_page	すべて	

7.50. Edge Handling Instructions

`EDGE8cc`, `EDGE8Lcc`, `EDGE16cc`, `EDGE16Lcc`, `EDGE32cc`, `EDGE32Lcc` は UA2011 7.23 参照

例外	対象命令	検出条件
illegal_action	すべて	XAR.v = 1

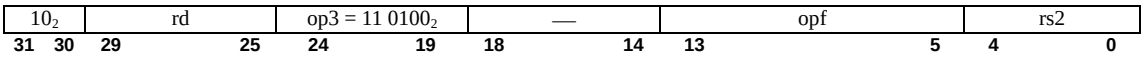
7.51. Edge Handling Instructions (noCC)

EDGE8N, EDGE8LN, EDGE16N, EDGE16LN, EDGE32N, EDGE32LN は UA2011 7.24 参照

例外	対象命令	検出条件
illegal_action	すべて	XAR.v = 1

7.52. Convert Between Floating-Point Formats

命令	opf	操作	HPC-ACE2		アセンブリ言語表記	
			Regs.	SIMD		
FsTOd	$0\ 1100\ 1001_2$	単精度浮動小数点数を倍精度浮動小数点数に変換	✓	✓	fstod	$freg_{rs2}$, $freg_{rd}$
FsTOq	$0\ 1100\ 1101_2$	単精度浮動小数点数を 4 倍精度浮動小数点数に変換	✓		fstoq	$freg_{rs2}$, $freg_{rd}$
FdTOs	$0\ 1100\ 0110_2$	倍精度浮動小数点数を単精度浮動小数点数に変換	✓	✓	fdtos	$freg_{rs2}$, $freg_{rd}$
FdTOq	$0\ 1100\ 1110_2$	倍精度浮動小数点数を 4 倍精度浮動小数点数に変換	✓		fdtoq	$freg_{rs2}$, $freg_{rd}$
FqTOs	$0\ 1100\ 0111_2$	4 倍精度浮動小数点数を単精度浮動小数点数に変換	✓		fqtos	$freg_{rs2}$, $freg_{rd}$
FqTOd	$0\ 1100\ 1011_2$	4 倍精度浮動小数点数を倍精度浮動小数点数に変換	✓		fqtod	$freg_{rs2}$, $freg_{rd}$



動作説明

これらの命令は、F[rs2]の値を指定された精度の浮動小数点数に変換する。結果は、F[rd]に格納される。これらの命令によって実行される丸めの方法は、FSR.rd または GSR.irnd によって決まる。

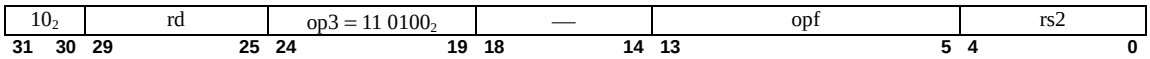
SPARC64™ XIfx では 4 倍精度浮動小数点レジスタを参照するハードウェア命令を実装しない。FsTOq, FdTOq, FqTOs, FqTOd 命令は、特権ソフトウェアがその命令をエミュレートする事を可能とするために *illegal_instruction* 例外を生じる。

例外		対象命令	検出条件
illegal_instruction		FsTOd, FdTOs	*reserved* フィールドが 0 でない
		FsTOq, FdTOq, FqTOs, FqTOd	常に これらの命令の *illegal_instruction* より優先度の低い例外は、エミュレーション用の仕様。
fp_disabled		すべて	PSTATE.pef = 0 または FPRS.fef = 0
illegal_action		FsTOd, FdTOs	XAR.v = 1 かつ、下記のいずれかが成立している場合 • XAR.urs1 ≠ 0 • XAR.urs3 ≠ 0 • XAR.simd = 1 かつ XAR.urs2<2> ≠ 0 • XAR.simd = 1 かつ XAR.urd<2> ≠ 0
		FsTOq, FdTOq, FqTOs, FqTOd	XAR.v = 1 かつ、下記のいずれかが成立している場合 • XAR.simd = 1 • XAR.urs1 ≠ 0 • XAR.urs3 ≠ 0
fp_exception_ieee_754	OF, UF, NX	FqTOs, FqTOd, FdTOs	IEEE 754 に準拠
	NV	すべて	F[rs2]が SNaN
fp_exception_other (FSR.ftt = *invalid_fp_register*)		FsTOq, FdTOq	rd<1> ≠ 0
		FqTOs, FqTOd	rs2<1> ≠ 0
fp_exception_other (FSR.ftt = *unfinished_FPop*)		FsTOd, FdTOs	第 8 章を参照

Compatibility Note *fp_exception_other* (FSR.ftt = *invalid_fp_register*) は UA2011 準拠。JPS1 では 4 倍精度命令の実行で *fp_exception_other* (FSR.ftt = *unimplemented_FPop*) 例外を検出していた。

7.53. Floating-Point Absolute Value

命令	opf	操作	HPC-ACE2		アセンブリ言語表記	
			Regs.	SIMD		
FABSs	$0\ 0000\ 1001_2$	単精度浮動小数点数の絶対値	✓	✓	`fabss`	*freg*$_{rs2}$, *freg*$_{rd}$
FABSd	$0\ 0000\ 1010_2$	倍精度浮動小数点数の絶対値	✓	✓	`fabsd`	*freg*$_{rs2}$,*freg*$_{rd}$
FABSq	$0\ 0000\ 1011_2$	4 倍精度浮動小数点数の絶対値	✓		`fabsq`	*freg*$_{rs2}$,*freg*$_{rd}$



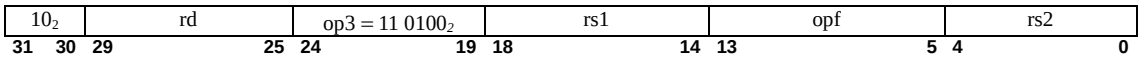
FABS 命令は、浮動小数点数の絶対値を求める命令である。FABSs 命令は Fs[rs2]<31> (符号ビット) を 0 にした値を Fs[rd]に書き込む。FABSd 命令は Fd[rs2]<63> (符号ビット) を 0 にした値を Fd[rd]に書き込む。FABSq 命令は Fq[rs2]<127> (符号ビット) を 0 にした値を Fq[rd]に書き込む。入力が NaN であるかどうかによらず、符号ビットを 0 にした値を出力する。

これらの命令を実行すると、FSR.cexc および FSR.ftt には 0 がセットされ、FSR.aexc は更新されない。

例外	対象命令	検出条件
illegal_instruction	FABSs, FABSd	*reserved* フィールドが 0 でない
	FABSq	常に この命令の *illegal_instruction* より優先度の低い例外は、エミュレーション用の仕様。
fp_disabled	すべて	PSTATE.pef = 0 または FPRS.fef = 0
illegal_action	FABSs, FABSd	XAR.v = 1 かつ、下記のいずれかが成立している場合 • XAR.urs1 ≠ 0 • XAR.urs3 ≠ 0 • XAR.simd = 1 かつ XAR.urs2<2> ≠ 0 • XAR.simd = 1 かつ XAR.urd<2> ≠ 0
	FABSq	XAR.v = 1 かつ、下記のいずれかが成立している場合 • XAR.simd = 1 • XAR.urs1 ≠ 0 • XAR.urs3 ≠ 0
fp_exception_other (FSR.ftt = *invalid_fp_register*)	FABSq	下記のいずれかが成立している場合 • rs2<1> ≠ 0 • rd<1> ≠ 0

7.54. Floating-Point Add and Subtract

命令	opf	操作	HPC-ACE2		アセンブリ言語表記	
			Regs.	SIMD		
FADDs	$0\ 0100\ 0001_2$	32 ビット浮動小数点加算	✓	✓	fadds	$freg_{rs1}$, $freg_{rs2}$, $freg_{rd}$
FADDd	$0\ 0100\ 0010_2$	64 ビット浮動小数点加算	✓	✓	faddd	$freg_{rs1}$, $freg_{rs2}$, $freg_{rd}$
FADDq	$0\ 0100\ 0011_2$	128 ビット浮動小数点加算	✓		faddq	$freg_{rs1}$, $freg_{rs2}$, $freg_{rd}$
FSUBs	$0\ 0100\ 0101_2$	32 ビット浮動小数点減算	✓	✓	fsubs	$freg_{rs1}$, $freg_{rs2}$, $freg_{rd}$
FSUBd	$0\ 0100\ 0110_2$	64 ビット浮動小数点減算	✓	✓	fsubd	$freg_{rs1}$, $freg_{rs2}$, $freg_{rd}$
FSUBq	$0\ 0100\ 0111_2$	128 ビット浮動小数点減算	✓		fsubq	$freg_{rs1}$, $freg_{rs2}$, $freg_{rd}$



動作説明

FADD{s|d|q}命令は、F[rs1]の浮動小数点数と F[rs2]の浮動小数点数を加算し、結果を F[rd]に書き込む。

FSUB{s|d|q}命令は、F[rs1]の浮動小数点数から F[rs2]の浮動小数点数を減算し、結果を F[rd]に書き込む。

これらの命令によって実行される丸めの方法は、FSR.rd または GSR.irnd によって決まる。

Note SPARC64™ XIfx では 4 倍精度浮動小数点レジスタを参照するハードウェア命令を実装しない。FADDq, FSUBq 命令は、特権ソフトウェアがその命令をエミュレートする事を可能とするために *illegal_instruction* 例外を生じる。

例外		対象命令	検出条件
illegal_instruction		FADDs, FADDd, FSUBs, FSUBd	*reserved* フィールドが 0 でない
		FADDq, FSUBq,	常に これらの命令の *illegal_instruction* より優先度の低い例外は、エミュレーション用の仕様。
fp_disabled		すべて	PSTATE.pef = 0 または FPRS.fef = 0
illegal_action		FADDs, FADDd, FSUBs, FSUBd	XAR.v = 1 かつ、下記のいずれかが成立している場合 • XAR.urs3 ≠ 0 • XAR.simd = 1 かつ XAR.urs1<2> ≠ 0 • XAR.simd = 1 かつ XAR.urs2<2> ≠ 0 • XAR.simd = 1 かつ XAR.urd<2> ≠ 0
		FADDq, FSUBq,	XAR.v = 1 かつ、以下のいずれかが成立している場合 • XAR.simd = 1 • XAR.urs3 ≠ 0
fp_exception_ieee_754	OF, UF, NX, NV	すべて	IEEE 754 に準拠
fp_exception_other (FSR.ftt = *invalid_fp_register*)		FADDq FSUBq	下記のいずれかが成立している場合 • rs1<1> ≠ 0 • rs2<1> ≠ 0 • rd<1> ≠ 0
fp_exception_other (FSR.ftt = *unfinished_FPop*)		FADDs, FADDd, FSUBs, FSUBd	第 8 章を参照

Compatibility Note *fp_exception_other* (FSR.ftt = *invalid_fp_register*) は UA2011 準拠。JPS1 では 4 倍精度命令の実行で *fp_exception_other* (FSR.ftt = *unimplemented_FPop*) 例外を検出していた。

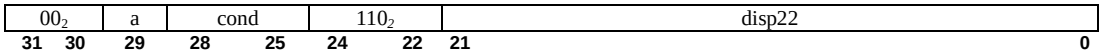
7.55. Align Data

FALIGNDATA は UA2011 7.27 FALIGNDATAg 参照

例外	対象命令	検出条件
fp_disabled	FALIGNDATA	PSTATE.pef = 0 または FPRS.fef = 0
illegal_action	FALIGNDATA	XAR.v = 1

7.56. Branch on Floating-Point Condition Codes (FBfcc)

命令	cond	分岐条件	fcc の値 (分岐時)	HPC-ACE2 Regs.	HPC-ACE2 SIMD	アセンブリ言語表記	
FBA^{D}	1000_2	常に分岐する	0, 1, 2, 3			`fba{,a}`	*label*
FBN^{D}	0000_2	決して分岐しない	—			`fbn{,a}`	*label*
FBU^{D}	0111_2	比較不能のとき	3			`fbu{,a}`	*label*
FBG^{D}	0110_2	より大きいとき	2			`fbg{,a}`	*label*
FBUG^{D}	0101_2	比較不能またはより大きいとき	2, 3			`fbug{,a}`	*label*
FBL^{D}	0100_2	より小さいとき	1			`fbl{,a}`	*label*
FBUL^{D}	0011_2	比較不能またはより小さいとき	1, 3			`fbul{,a}`	*label*
FBLG^{D}	0010_2	より小さいまたはより大きいとき	1, 2			`fblg{,a}`	*label*
FBNE^{D}	0001_2	等しくないとき	1, 2, 3			`fbne{,a}` `fbnz{,a}`	*label* *label*
FBE^{D}	1001_2	等しいとき	0			`fbe{,a}` `fbz{,a}`	*label* *label*
FBUE^{D}	1010_2	比較不能または等しいとき	0, 3			`fbue{,a}`	*label*
FBGE^{D}	1011_2	以上のとき	0, 2			`fbge{,a}`	*label*
FBUGE^{D}	1100_2	比較不能または以上のとき	0, 2, 3			`fbuge{,a}`	*label*
FBLE^{D}	1101_2	以下のとき	0, 1			`fble{,a}`	*label*
FBULE^{D}	1110_2	比較不能または以下のとき	0, 1, 3			`fbule{,a}`	*label*
FBO^{D}	1111_2	比較可能のとき	0, 1, 2			`fbo{,a}`	*label*



Programming Note FBfcc 命令に対して無効ビット（a ビット）をセットする場合は、 “,a”をオペコードのニモニックに付加する。例えば、 “fbl,a *label*” とする。上記のフィールドの説明において、アセンブラ言語表記で{,a}となっているのは、無効ビットのセットが任意であることを意味する。

動作説明

FBN は、無効ビットが 0 (a = 0) ならば NOP として扱われる。無効ビットが 1 (a = 1) ならば、次の命令（遅延命令）は、無効とされる（実行されない)。どちらの場合も、制御転送先には分岐しない。

FBA は、浮動小数点条件ビットの値に関わらず、アドレス “PC + (4 · sign_ext(disp22))” へ無条件に PC 相対遅延制御転送を生じる。もし、分岐命令の無効 (a) ビットが 1 ならば、遅延命令は無効とされる（実行されない)。もし、無効ビットが 0 (a = 0) ならば、遅延命令は実行される。

FBA と FBN 以外の FBfcc 命令は、真または偽の結果のどちらかを生成するために、命令の cond フィールドにしたがって、浮動小数点条件コード (FSR.fcc0) を評価する。もし、真であるならば分岐が行われる、すなわち命令は、アドレス “PC + (4 · sign_ext(disp22))” へ PC 相対遅延制御転送を生じる。もし、偽であるならば分岐は行われない。もし制御転送が行われるならば、

遅延命令は無効ビットの値に関わらず、いつも実行される。もし、制御転送が行われない、かつ、無効ビットが 1 (a = 1) であるならば、遅延命令は無効とされる(実行されない)。

Note FBA 命令と、他の FBfcc 命令の無効ビットの作用は異なる。

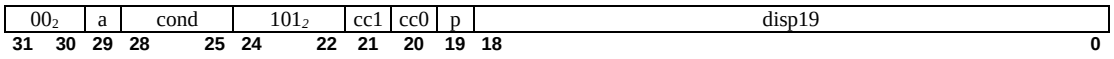
SPARC64™ XIfx では、制御転送機構におけるトラップが実装されている (page363)。そのため、PSTATE.tct = 1 であり、FBfcc 命令が制御転送 (FBA または taken となる条件分岐) を生じるならば、そのとき FBfcc は、制御転送を起こさずに *control_transfer_instruction* 例外を生成する。*control_transfer_instruction* トラップが生じるとき、PC (FBfcc 命令のアドレス) は TPC[TL]に保存され、FBfcc が実行される前の NPC の値は TNPC[TL]に保存される。

FBN は、*control_transfer_instruction* 例外を決して生じない事に注意すること。

例外	対象命令	検出条件
fp_disabled	すべて	PSTATE.pef = 0 または FPRS.fef = 0
illegal_action	すべて	XAR.v = 1
control_transfer_instruction	FBN 以外	条件分岐が成立し、PSTATE.tct = 1 の場合。 FBA 命令は常に条件分岐が成立しているとする。

7.57. Branch on Floating-Point Condition Code with Prediction (FBPfcc)

命令	cond	分岐条件	fcc の値 (分岐時)	HPC-ACE2 Regs.	HPC-ACE2 SIMD	アセンブリ言語表記	
FBPA	1000_2	常に分岐する	0, 1, 2, 3			`fba{,a}{,pt|,pn}`	%fcc*n, label*
FBPN	0000_2	決して分岐しない	—			`fbn{,a}{,pt|,pn}`	%fcc*n*, *label*
FBPU	0111_2	比較不能のとき	3			`fbu{,a}{,pt|,pn}`	%fcc*n*, *label*
FBPG	0110_2	より大きいとき	2			`fbg{,a}{,pt|,pn}`	%fcc*n*, *label*
FBPUG	0101_2	比較不能またはより大きいとき	2, 3			`fbug{,a}{,pt|,pn}`	%fcc*n*, *label*
FBPL	0100_2	より小さいとき	1			`fbl{,a}{,pt|,pn}`	%fcc*n*, *label*
FBPUL	0011_2	比較不能またはより小さいとき	1, 3			`fbul{,a}{,pt|,pn}`	%fcc*n*, *label*
FBPLG	0010_2	より小さいまたはより大きいとき	1, 2			`fblg{,a}{,pt|,pn}`	%fcc*n*, *label*
FBPNE	0001_2	等しくないとき	1, 2, 3			`fbne{,a}{,pt|,pn}` `fbnz{,a}{,pt|,pn}`	%fcc*n*, *label* %fcc*n*, *label*
FBPE	1001_2	等しいとき	0			`fbe{,a}{,pt|,pn}` `fbz{,a}{,pt|,pn}`	%fcc*n*, *label* %fcc*n*, *label*
FBPUE	1010_2	比較不能または等しいとき	0, 3			`fbue{,a}{,pt|,pn}`	%fcc*n*, *label*
FBPGE	1011_2	以上のとき	0, 2			`fbge{,a}{,pt|,pn}`	%fcc*n*, *label*
FBPUGE	1100_2	比較不能または以上のとき	0, 2, 3			`fbuge{,a}{,pt|,pn}`	%fcc*n*, *label*
FBPLE	1101_2	以下のとき	0, 1			`fble{,a}{,pt|,pn}`	%fcc*n*, *label*
FBPULE	1110_2	比較不能または以下のとき	0, 1, 3			`fbule{,a}{,pt|,pn}`	%fcc*n*, *label*
FBPO	1111_2	比較可能のとき	0, 1, 2			`fbo{,a}{,pt|,pn}`	%fcc*n*, *label*



cc1	cc0	条件コード
0	0	fcc0
0	1	fcc1
1	0	fcc2
1	1	fcc3

Programming Note　FBPfcc 命令に対して無効ビット（a ビット）をセットする場合は、“,a”をオペコードのニモニックに付加する。例えば、“fbl,a %fcc*n*, *label*”とする。 上記のフィールドの説明において、アセンブラ言語表記で{,a}となっているのは、無効ビットのセットが任意であることを示す。分岐予測ビットに関しては、分岐すると予測する場合は、“,pt”を、分岐しないと予測する場合には “,pn”をオペコードのニモニックに付加する（どちらも付加されていない場合は、アセンブラは“,pt”と解釈する）。浮動小数点比較の結果は“%fcc0” , “%fcc1” , “%fcc2” , または “%fcc3”のいずれかを、分岐先ラベルの前に置く。

動作説明

FBPN は、無効ビットが 0 (a = 0) ならば NOP として扱われる。無効ビットが 1 (a = 1) ならば、次の命令（遅延命令）は、無効とされる（実行されない）。どちらの場合も、制御転送先には分岐しない。

FBPA は、浮動小数点条件ビットの値に関わらず、アドレス “PC + (4 · sign_ext(disp19))” へ無条件に PC 相対遅延制御転送を生じる。もし、分岐命令の無効 (a) ビットが 1 ならば、遅延命令は無効とされる（実行されない）。もし、無効ビットが 0 (a = 0)ならば、遅延命令は実行される。

FBPA と FBPN 以外の FBPfcc 命令は、真または偽の結果のどちらかを生成するために、命令の cond フィールドにしたがって、浮動小数点条件コードを評価する。浮動小数点条件コードは、cc0 および cc1 で指定される。もし、真であるならば分岐が行われる、すなわち命令は、アドレス “PC + (4 · sign_ext(disp19))” へ PC 相対遅延制御転送を生じる。もし、偽であるならば分岐は行われない。もし条件分岐が行われるならば、遅延命令は無効ビットの値に関わらず、いつも実行される。もし、条件分岐が行われない、かつ、無効ビットが 1 (a = 1)で あるならば、遅延命令は無効とされる（実行されない）。

Note　FBPA 命令と、他の FBPfcc 命令の無効ビットの作用は異なる。

分岐予測ビット (p) は、ハードウェアに対し、条件分岐が成立するかしないかのヒントを与える。p = 1 ならば条件分岐が成立すると予測され、p = 0 ならば条件分岐が成立しないと予測される。

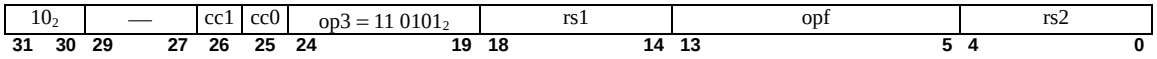
SPARC64™ XIfx では、制御転送機構におけるトラップが実装されている（page363）。そのため、PSTATE.tct = 1 であり、FBPfcc 命令が制御転送（FBPA または taken となる条件分岐）を生じるならば、そのとき FBPfcc は、制御転送を起こさずに *control_transfer_instruction* 例外を生成する。*control_transfer_instruction* トラップが生じるとき、PC（FBPfcc 命令のアドレス）は TPC[TL]に保存され、FBPfcc が実行される前の NPC の値は TNPC[TL]に保存される。

FBPN は、*control_transfer_instruction* 例外を決して生じない事に注意が必要である。

例外	対象命令	検出条件
fp_disabled	すべて	PSTATE.pef = 0 または FPRS.fef = 0
illegal_action	すべて	XAR.v = 1
control_transfer_instruction	FBPN 以外	条件分岐が成立し、PSTATE.tct = 1 の場合。 FBPA 命令は常に条件分岐が成立しているとする。

7.58. Floating-Point Compare

命令	opf	操作	HPC-ACE2		アセンブリ言語表記
			Regs	SIMD	
FCMPs	$0\ 0101\ 0001_2$	単精度浮動小数点数の比較	✓		`fcmps %fccn,` *freg*$_{rs1}$*, freg*$_{rs2}$
FCMPd	$0\ 0101\ 0010_2$	倍精度浮動小数点数の比較	✓		`fcmpd %fccn,` *freg*$_{rs1}$*, freg*$_{rs2}$
FCMPq	$0\ 0101\ 0011_2$	4 倍精度浮動小数点数の比較	✓		`fcmpq %fccn,` *freg*$_{rs1}$*, freg*$_{rs2}$
FCMPEs	$0\ 0101\ 0101_2$	単精度浮動小数点数の比較 比較不能なら例外発生	✓		`fcmpes %fccn,` *freg*$_{rs1}$*, freg*$_{rs2}$
FCMPEd	$0\ 0101\ 0110_2$	倍精度浮動小数点数の比較 比較不能なら例外発生	✓		`fcmped %fccn,` *freg*$_{rs1}$*, freg*$_{rs2}$
FCMPEq	$0\ 0101\ 0111_2$	4 倍精度浮動小数点数の比較 比較不能なら例外発生	✓		`fcmpeq %fccn,` *freg*$_{rs1}$*, freg*$_{rs2}$



cc1	cc0	条件コード
0	0	fcc0
0	1	fcc1
1	0	fcc2
1	1	fcc3

動作説明

これらの命令は、F[rs1]と F[rs2]を浮動小数点数として比較し、命令で指定した浮動小数点条件コードフィールド FSR.fcc*n* に格納する。FSR.fcc*n* に格納される値は表 5-7(24 ページ) 参照。

FCMP 命令は、入力のどちらかが sNaN のとき、*fp_exception_ieee_754* の無効例外 (NV) を生成する。FCMPE{s|d|q}命令は、入力のどちらかが sNaN または qNaN のとき、*fp_exception_ieee_754* の無効例外 (NV) を生成する。

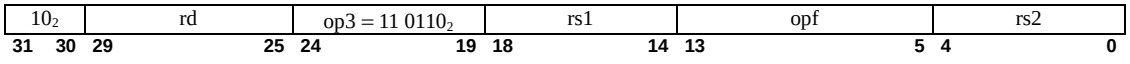
Compatibility Note SPARC V9 仕様は SPARC V8 仕様と異なり、浮動小数点数の比較命令とその結果による分岐命令の間に命令を必要としない。

Compatibility Note SPARC V8 仕様では、FCMP 命令は rd = 0 になっている必要があった。SPARC V9 仕様では、命令語のビット 26 と 25 で fcc*n* を指定する。したがって SPARC V8 仕様に準拠したプログラムは、SPARC V9 では fcc0 を指定したものと解釈され、正しく動作する。

例外		対象命令	検出条件
illegal_instruction		FCMPs, FCMPd, FCMPEs, FCMPEd	*reserved* フィールドが 0 でない
		FCMPq, FCMPEq	常に この命令の *illegal_instruction* より優先度の低い例外は、エミュレーション用の仕様。
fp_disabled		すべて	PSTATE.pef = 0 または FPRS.fef = 0
illegal_action		すべて	XAR.v = 1 かつ、下記のいずれかが成立している場合 • XAR.simd = 1 • XAR.urs3 ≠ 0 • XAR.urd ≠ 0
fp_exception_ieee_754	NV	すべて	IEEE 754 に準拠
fp_exception_other (FSR.ftt = *invalid_fp_register*)		FCMPq, FCMPEq	下記のいずれかが成立している場合 • rs1<1> ≠ 0 • rs2<1> ≠ 0

7.59. Floating-Point Conditional Compare to Register

命令	opf	操作	HPC-ACE2 Regs	HPC-ACE2 SIMD	アセンブリ言語表記	
FCMPEQd	$1\ 0110\ 0000_2$	Fd[rs1] = Fd[rs2]	✓	✓	fcmpeqd	*freg*$_{rs1}$, *freg*$_{rs2}$, *freg*$_{rd}$
FCMPEQEd	$1\ 0110\ 0010_2$	Fd[rs1] = Fd[rs2] 比較不能なら例外発生	✓	✓	fcmpeqed	*freg*$_{rs1}$, *freg*$_{rs2}$, *freg*$_{rd}$
FCMPLEEd	$1\ 0110\ 0100_2$	Fd[rs1] ≤ Fd[rs2] 比較不能なら例外発生	✓	✓	fcmpleed	*freg*$_{rs1}$, *freg*$_{rs2}$, *freg*$_{rd}$
FCMPLTEd	$1\ 0110\ 0110_2$	Fd[rs1] < Fd[rs2] 比較不能なら例外発生	✓	✓	fcmpleted	*freg*$_{rs1}$, *freg*$_{rs2}$, *freg*$_{rd}$
FCMPNEd	$1\ 0110\ 1000_2$	Fd[rs1] ≠ Fd[rs2]	✓	✓	fcmpned	*freg*$_{rs1}$, *freg*$_{rs2}$, *freg*$_{rd}$
FCMPNEEd	$1\ 0110\ 1010_2$	Fd[rs1] ≠ Fd[rs2] 比較不能なら例外発生	✓	✓	fcmpneed	*freg*$_{rs1}$, *freg*$_{rs2}$, *freg*$_{rd}$
FCMPGTEd	$1\ 0110\ 1100_2$	Fd[rs1] > Fd[rs2] 比較不能なら例外発生	✓	✓	fcmpgted	*freg*$_{rs1}$, *freg*$_{rs2}$, *freg*$_{rd}$
FCMPGEEd	$1\ 0110\ 1110_2$	Fd[rs1] ≥ Fd[rs2] 比較不能なら例外発生	✓	✓	fcmpgeed	*freg*$_{rs1}$, *freg*$_{rs2}$, *freg*$_{rd}$
FCMPEQs	$1\ 0110\ 0001_2$	Fs[rs1] = Fs[rs2]	✓	✓	fcmpeqs	*freg*$_{rs1}$, *freg*$_{rs2}$, *freg*$_{rd}$
FCMPEQEs	$1\ 0110\ 0011_2$	Fs[rs1] = Fs[rs2] 比較不能なら例外発生	✓	✓	fcmpeqes	*freg*$_{rs1}$, *freg*$_{rs2}$, *freg*$_{rd}$
FCMPLEEs	$1\ 0110\ 0101_2$	Fs[rs1] ≤ Fs[rs2] 比較不能なら例外発生	✓	✓	fcmplees	*freg*$_{rs1}$, *freg*$_{rs2}$, *freg*$_{rd}$
FCMPLTEs	$1\ 0110\ 0111_2$	Fs[rs1] < Fs[rs2] 比較不能なら例外発生	✓	✓	fcmpltes	*freg*$_{rs1}$, *freg*$_{rs2}$, *freg*$_{rd}$
FCMPNEs	$1\ 0110\ 1001_2$	Fs[rs1] ≠ Fs[rs2]	✓	✓	fcmpnes	*freg*$_{rs1}$, *freg*$_{rs2}$, *freg*$_{rd}$
FCMPNEEs	$1\ 0110\ 1011_2$	Fs[rs1] ≠ Fs[rs2] 比較不能なら例外発生	✓	✓	fcmpnees	*freg*$_{rs1}$, *freg*$_{rs2}$, *freg*$_{rd}$
FCMPGTEs	$1\ 0110\ 1101_2$	Fs[rs1] > Fs[rs2] 比較不能なら例外発生	✓	✓	fcmpgtes	*freg*$_{rs1}$, *freg*$_{rs2}$, *freg*$_{rd}$
FCMPGEEs	$1\ 0110\ 1111_2$	Fs[rs1] ≥ Fs[rs2] 比較不能なら例外発生	✓	✓	fcmpgees	*freg*$_{rs1}$, *freg*$_{rs2}$, *freg*$_{rd}$



動作説明

これらの命令は、F[rs1]と F[rs2]を浮動小数点数として比較し、条件が成立していれば all1、不成立なら all0 を、F[rd]に格納する。

入力値のどちらかが SNaN または QNaN のときの例外と結果は以下のようになる。exception の列は *fp_exception_ieee_754* 例外発生時 FSR.cexc にセットされる値を、F[rd]の列は例外が通知されない場合に結果として格納される値を示す。

命令	SNaN		QNaN	
	例外	F[rd]	例外	F[rd]
FCMPGTE{s|d}, FCMPLTE{s|d}, FCMPGEE{s|d}, FCMPLEE{s|d}	NV	all0	NV	all0
FCMPEQE{s|d}	NV	all0	NV	all0
FCMPNEE{s|d}	NV	all1	NV	all1
FCMPEQ{s|d}	NV	all0	—	all0
FCMPNE{s|d}	NV	all1	—	all1

Programming Note　この命令は、FSELMOV{s|d}, ST{D}FR, VIS の論理演算命令と組み合わせて使うことを想定している。

例外		対象命令	検出条件
fp_disabled		全て	PSTATE.pef = 0 または FPRS.fef = 0
illegal_action		全て	XAR.v = 1 かつ、下記のいずれかが成立している場合 • XAR.urs3 ≠ 0 • XAR.simd = 1 かつ XAR.urs1<2> ≠ 0 • XAR.simd = 1 かつ XAR.urs2<2> ≠ 0 • XAR.simd = 1 かつ XAR.urd<2> ≠ 0
fp_exception_ieee_754	NV	全て	Unordered

7.60. Partitioned Signed Compare

Compatibility Note SPARC64™ XIfx では UA2011 仕様において Partitioned Signed Compare として定義されてしている命令及び Partitioned Unsigned Compare として定義している命令について一部古いニーモニックにて定義されている。

Compatibility Note UA2011 の Partitioned Signed Compare として定義されている FPCMPNE8, FPCMPEQ8 は Partitioned Unsigned Compare として定義されている FPCMPUNE8, FPCMPUEQ8 のニーモニックにのみ対応している。Partitioned Unsigned Compare については 301 ページ参照。

Compatibility Note UA2011 の Partitioned Signed Compare として定義されている FPCMPEQ16, FPCMPNE16, FPCMPLE16, FPCMPGT16, FPCMPEQ32, FPCMPNE32, FPCMPLE32, FPCMPGT32 は古いニーモニックである FCMPEQ16, FCMPNE16, FCMPLE16, FCMPGT16, FCMPEQ32, FCMPNE32, FCMPLE32, FCMPGT32 にのみ対応している。

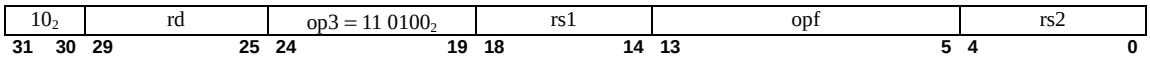
命令の動作は以下の UA2011 仕様を参照。

命令	参照先
FCMPEQ16	UA2011 7.54 FPCMPEQ16 参照
FCMPNE16	UA2011 7.54 FPCMPNE16 参照
FCMPLE16	UA2011 7.54 FPCMPLE16 参照
FCMPGT16	UA2011 7.54 FPCMPGT16 参照
FCMPEQ32	UA2011 7.54 FPCMPEQ32 参照
FCMPNE32	UA2011 7.54 FPCMPNE32 参照
FCMPLE32	UA2011 7.54 FPCMPLE32 参照
FCMPGT32	UA2011 7.54 FPCMPGT32 参照

例外	対象命令	検出条件
fp_disabled	全て	PSTATE.pef = 0 または FPRS.fef = 0
illegal_action	全て	XAR.v = 1

7.61. Floating-Point Divide

命令	opf	操作	HPC-ACE2		アセンブリ言語表記	
			Regs	SIMD		
FDIVs	$0\ 0100\ 1101_2$	単精度浮動小数点数の除算	☆		fdivs	$freg_{rs1}$,$freg_{rs2}$, $freg_{rd}$
FDIVd	$0\ 0100\ 1110_2$	倍精度浮動小数点数の除算	☆		fdivd	$freg_{rs1}$,$freg_{rs2}$,$freg_{rd}$
FDIVq	$0\ 0100\ 1111_2$	4 倍精度浮動小数点数の除算	☆		fdivq	$freg_{rs1}$,$freg_{rs2}$,$freg_{rd}$

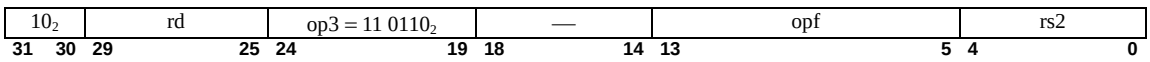


FDIV 命令は、F[rs1]の浮動小数点数を F[rs2]の浮動小数点数で除した結果を F[rd]に書き込む。FDIV 命令によって実行される丸めの方法は、FSR.rd または GSR.irnd によって決まる。

例外		対象命令	検出条件
illegal_instruction		FDIVq	常に この命令の *illegal_instruction* より優先度の低い例外は、エミュレーション用の仕様。
fp_disabled		すべて	PSTATE.pef = 0 または FPRS.fef = 0
illegal_action		すべて	XAR.v = 1 かつ、下記のいずれかが成立している場合 • XAR.simd = 1 • XAR.urs3 ≠ 0 • XAR.urd<2> ≠ 0
fp_exception_ieee_754	OF, UF, DZ, NV, NX	すべて	IEEE 754 に準拠
fp_exception_other (FSR.ftt = *invalid_fp_register*)		FDIVq	下記のいずれかが成立している場合 • rs1<1> ≠ 0 • rs2<1> ≠ 0
fp_exception_other (FSR.ftt = *unfinished_FPop*)		FDIVs, FDIVd	第 8 章を参照

7.62. Floating-Point Exponential Auxiliary

命令	opf	操作	HPC-ACE2 Regs	HPC-ACE2 SIMD	アセンブリ言語表記	
FEXPAd	$1\ 0111\ 1100_2$	指数関数補助命令	✓	✓	fexpad	*freg*$_{rs2}$, *freg*$_{rd}$



FEXPAd 命令は、指数関数 exp(x)の値を求める一連の演算の補助命令である。Fd[rs2]の内容に従いテーブルを引き、結果を Fd[rd]に格納する。

Fd[rd] = 1'b0 :: Fd[rs2]<16:6> :: Texp[Fd[rs2]<5:0>]

この命令を実行すると、FSR.cexc および FSR.ftt には 0 がセットされ、FSR.aexc は更新されない。

Texp は倍精度浮動小数点数の仮数部（52 ビット）を保持する 64 エントリのテーブルである。

表 7-19 Texp[k]の表

k	Texp[k]	k	Texp[k]	k	Texp[k]	k	Texp[k]
0	0x0000000000000	16	0x306FE0A31B715	32	0x6A09E667F3BCD	48	0xAE89F995AD3AD
1	0x02C9A3E778061	17	0x33C08B26416FF	33	0x6DFB23C651A2F	49	0xB33A2B84F15FB
2	0x059B0D3158574	18	0x371A7373AA9CB	34	0x71F75E8EC5F74	50	0xB7F76F2FB5E47
3	0x0874518759BC8	19	0x3A7DB34E59FF7	35	0x75FEB564267C9	51	0xBCC1E904BC1D2
4	0x0B5586CF9890F	20	0x3DEA64C123422	36	0x7A11473EB0187	52	0xC199BDD85529C
5	0x0E3EC32D3D1A2	21	0x4160A21F72E2A	37	0x7E2F336CF4E62	53	0xC67F12E57D14B
6	0x11301D0125B51	22	0x44E086061892D	38	0x82589994CCE13	54	0xCB720DCEF9069
7	0x1429AAEA92DE0	23	0x486A2B5C13CD0	39	0x868D99B4492ED	55	0xD072D4A07897C
8	0x172B83C7D517B	24	0x4BFDAD5362A27	40	0x8ACE5422AA0DB	56	0xD5818DCFBA487
9	0x1A35BEB6FCB75	25	0x4F9B2769D2CA7	41	0x8F1AE99157736	57	0xDA9E603DB3285
10	0x1D4873168B9AA	26	0x5342B569D4F82	42	0x93737B0CDC5E5	58	0xDFC97337B9B5F
11	0x2063B88628CD6	27	0x56F4736B527DA	43	0x97D829FDE4E50	59	0xE502EE78B3FF6
12	0x2387A6E756238	28	0x5AB07DD485429	44	0x9C49182A3F090	60	0xEA4AFA2A490DA
13	0x26B4565E27CDD	29	0x5E76F15AD2148	45	0xA0C667B5DE565	61	0xEFA1BEE615A27
14	0x29E9DF51FDEE1	30	0x6247EB03A5585	46	0xA5503B23E255D	62	0xF50765B6E4540
15	0x2D285A6E4030B	31	0x6623882552225	47	0xA9E6B5579FDBF	63	0xFA7C1819E90D8

FEXPAd 命令は Fd[rs2]を浮動小数点数として扱わない。Fd[rs2]が NaN であっても特別扱いしない。

例外	検出条件
illegal_instruction	*reserved* フィールドが 0 でない
fp_disabled	PSTATE.pef = 0 または FPRS.fef = 0
illegal_action	XAR.v = 1 かつ、下記のいずれかが成立している場合 • XAR.urs1 ≠ 0 • XAR.urs3 ≠ 0 • XAR.simd = 1 かつ XAR.urs2<2> ≠ 0 • XAR.simd = 1 かつ XAR.urd<2> ≠ 0

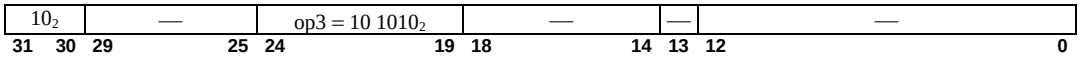
7.63. FEXPAND

FEXPAND は UA2011 7.33 参照

例外	検出条件
illegal_instruction	*reserved* フィールドが 0 でない
fp_disabled	PSTATE.pef = 0 または FPRS.fef = 0
illegal_action	XAR.v = 1

7.64. Flush Register Windows

命令	op3	操作	HPC-ACE2 Regs	HPC-ACE2 SIMD	アセンブリ言語表記
FLUSHW	$10\ 1010_2$	レジスタウィンドウをフラッシュする			flushw



FLUSHW 命令は現在のウィンドウ以外の有効なウィンドウレジスタの内容を、メモリに書き出す。現在のウィンドウ以外に有効なウィンドウレジスタがない場合、FLUSHW 命令は何もしない。FLUSHW 命令が完了すると、有効なウィンドウレジスタは現在のウィンドウだけになる。

Programming Note FLUSHW 命令はソフトウェアがメモリにレジスタ内容を書き出し、スタックを切り替えたりレジスタ内容を調べたりするために使うことができる。

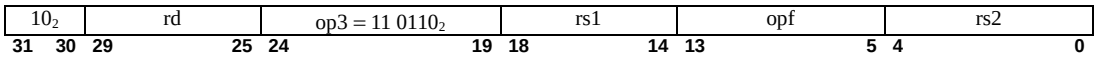
FLUSHW 命令は、CANSAVE = N_REG_WINDOWS – 2 のときは NOP になる。それ以外では、現在のウィンドウ以外にも有効なウィンドウがあるので、spill 例外が発生する。spill 例外のトラップベクタは、その時点の OTHERWIN と WSTATE の設定により決まる。spill ハンドラが呼び出される際には、CWP には書き出されるウィンドウ番号
(CWP + CANSAVE + 2) **mod** N_REG_WINDOWS が設定される。

Programming Note 通常、spill ハンドラはレジスタウィンドウをひとつメモリスタックに書き出し、CANSAVE, CANRESTORE を調整して終了し、FLUSHW 命令が再実行される。よって、現在のウィンドウ以外がすべてメモリに書き出されるまで、spill 例外によるトラップが次々と発生する。

例外	検出条件
illegal_instruction	*reserved* フィールドが 0 でない
illegal_action	XAR.v = 1
spill_n_normal	
spill_n_other	

7.65. Floating-Point Minimum and Maximum

命令	opf	操作	HPC-ACE2 Regs	HPC-ACE2 SIMD	アセンブリ言語表記	
FMAXd	$1\,0111\,0000_2$	倍精度最大値	✓	✓	fmaxd	$freg_{rs1}$, $freg_{rs2}$, $freg_{rd}$
FMAXs	$1\,0111\,0001_2$	単精度最大値	✓	✓	fmaxs	$freg_{rs1}$, $freg_{rs2}$, $freg_{rd}$
FMINd	$1\,0111\,0010_2$	倍精度最小値	✓	✓	fmind	$freg_{rs1}$, $freg_{rs2}$, $freg_{rd}$
FMINs	$1\,0111\,0011_2$	単精度最小値	✓	✓	fmins	$freg_{rs1}$, $freg_{rs2}$, $freg_{rd}$



動作説明

FMAX{s|d}は、F[rs1]と F[rs2]を浮動小数点数として比較し、F[rs1] > F[rs2]なら F[rs1]を、そうでなければ F[rs2]を、F[rd]に格納する。

FMIN{s|d}は、F[rs1]と F[rs2]を浮動小数点数として比較し、F[rs1] < F[rs2]なら F[rs1]を、そうでなければ F[rs2]を、F[rd]に格納する。

FMIN, FMAX はゼロの符号は無視する。F[rs1], F[rs2]がそれぞれ+0, -0 または-0, +0 のときは、F[rs2]の値が出力される。

入力値の一方が QNaN で、もう一方が QNaN, SNaN 以外の数ときは、F[rd]には QNaN 以外の数が格納される。

Note FMIN, FMAX は他の命令と異なり、NaN を伝播しない。

入力値の一方または両方が SNaN、または入力値の両方が QNaN のときは、F[rd]には JPS1 Commonality の Table B-1 で定義された値が格納される。また、入力値の少なくとも一方に QNaN, SNaN が含まれる場合、*fp_exception_ieee_754* 例外を検出する。

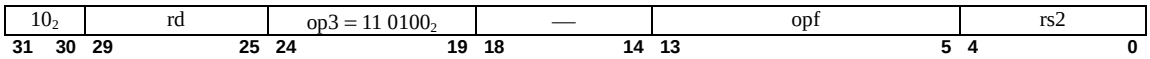
表 7-20 FMIN{s|d}, FMAX{s|d}の演算結果

		F[rs2]							
		-∞	-Fn	-0	+0	+Fn	+∞	QNaN	SNaN
F[rs1]	-∞	— min(F[rs1], F[rs2]), or max(F[rs1], F[rs2])						NV F[rs1]	NV QSNaN2
	-Fn								
	-0								
	+0								
	+Fn								
	+∞								
	QNaN	NV F[rs2]							
	SNaN	NV QSNaN1							

例外		対象命令	検出条件
fp_disabled		すべて	PSTATE.pef = 0 または FPRS.fef = 0
illegal_action		すべて	XAR.v = 1 かつ、下記のいずれかが成立している場合 • XAR.urs3 ≠ 0 • XAR.simd = 1 かつ XAR.urs1<2> ≠ 0 • XAR.simd = 1 かつ XAR.urs2<2> ≠ 0 • XAR.simd = 1 かつ XAR.urd<2> ≠ 0
fp_exception_ieee_754	NV	すべて	Unordered

7.66. Floating-Point Move

命令	opf	操作	HPC-ACE2 Regs	HPC-ACE2 SIMD	アセンブリ言語表記	
FMOVs	$0\ 0000\ 0001_2$	単精度レジスタの移動	✓	✓	fmovs	$freg_{rs2}$, $freg_{rd}$
FMOVd	$0\ 0000\ 0010_2$	倍精度レジスタの移動	✓	✓	fmovd	$freg_{rs2}$,$freg_{rd}$
FMOVq	$0\ 0000\ 0011_2$	4 倍精度レジスタの移動	✓		fmovq	$freg_{rs2}$,$freg_{rd}$



動作説明

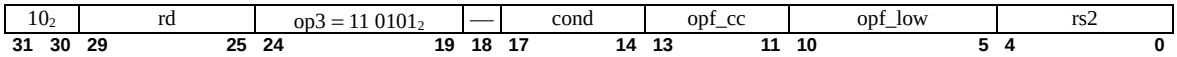
FMOV 命令は、F[rs2]の内容を F[rd]に書き込む。データ形式が何であるかを問わないので、NaN もその他の値と同じように扱われる。

これらの命令を実行すると、FSR.cexc および FSR.ftt には 0 がセットされ、FSR.aexc は更新されない。

例外	対象命令	検出条件
illegal_instruction	FMOVs, FMOVd	*reserved* フィールドが 0 でない
	FMOVq	常に この命令の *illegal_instruction* より優先度の低い例外は、エミュレーション用の仕様。
fp_disabled	すべて	PSTATE.pef = 0 または FPRS.fef = 0
illegal_action	FMOVs, FMOVd	XAR.v = 1 かつ、下記のいずれかが成立している場合 • XAR.urs1 ≠ 0 • XAR.urs3 ≠ 0 • XAR.simd = 1 かつ XAR.urs2<2> ≠ 0 • XAR.simd = 1 かつ XAR.urd<2> ≠ 0
	FMOVq	XAR.v = 1 かつ、下記のいずれかが成立している場合 • XAR.simd = 1 • XAR.urs1 ≠ 0 • XAR.urs3 ≠ 0
fp_exception_other (FSR.ftt = *invalid_fp_register*)	FMOVq	下記のいずれかが成立している場合 • rs2<1> ≠ 0 • rd<1> ≠ 0

7.67. Move Floating-Point Register on Condition (FMOVcc)

命令	opf_cc	opf_low	操作	HPC-ACE2 Regs	HPC-ACE2 SIMD	アセンブリ言語表記	
FMOVSicc	100_2	$00\ 0001_2$	単精度レジスタの移動（icc による）			fmovs*icc*	%icc, *freg*$_{rs2}$, *freg*$_{rd}$
FMOVDicc	100_2	$00\ 0010_2$	倍精度レジスタの移動（icc による）			fmovd*icc*	%icc, *freg*$_{rs2}$, *freg*$_{rd}$
FMOVQicc	100_2	$00\ 0011_2$	4 倍精度レジスタの移動（icc による）			fmovq*icc*	%icc, *freg*$_{rs2}$, *freg*$_{rd}$
FMOVSxcc	110_2	$00\ 0001_2$	単精度レジスタの移動（xcc による）			fmovs*xcc*	%xcc, *freg*$_{rs2}$, *freg*$_{rd}$
FMOVDxcc	110_2	$00\ 0010_2$	倍精度レジスタの移動（xcc による）			fmovd*xcc*	%xcc, *freg*$_{rs2}$, *freg*$_{rd}$
FMOVQxcc	110_2	$00\ 0011_2$	4 倍精度レジスタの移動（xcc による）			fmovq*xcc*	%xcc, *freg*$_{rs2}$, *freg*$_{rd}$
FMOVSfcc	$0xx_2$	$00\ 0001_2$	単精度レジスタの移動（fcc による）			fmovs*fcc*	%fccn, *freg*$_{rs2}$, *freg*$_{rd}$
FMOVDfcc	$0xx_2$	$00\ 0010_2$	倍精度レジスタの移動（fcc による）			fmovd*fcc*	%fccn, *freg*$_{rs2}$, *freg*$_{rd}$
FMOVQfcc	$0xx_2$	$00\ 0011_2$	4 倍精度レジスタの移動（fcc による）			fmovq*fcc*	%fccn, *freg*$_{rs2}$, *freg*$_{rd}$



cond	条件	テスト (icc or xcc)	アセンブリ言語表記
1000_2	常に	1	a
0000_2	決してない	0	n
1001_2	等しくないとき	**not** Z	ne, nx
0001_2	等しいとき	Z	e, z
1010_2	より大きいとき	**not** (Z **or** (N **xor** V))	g
0010_2	以下のとき	Z **or** (N **xor** V)	le
1011_2	以上のとき	**not** (N **xor** V)	ge
0011_2	より小さいとき	N **xor** V	l
1100_2	符号なし整数で、より大きいとき	**not** (C **or** Z)	gu
0100_2	符号なし整数で、以下のとき	C **or** Z	leu
1101_2	キャリークリア（符号なし整数で以上）のとき	**not** C	cc, geu
0101_2	キャリーセット（符号なし整数でより小さい）のとき	C	cs, lu
1110_2	正のとき	**not** N	pos
0110_2	負のとき	N	neg
1111_2	オーバーフローしていないとき	**not** V	vc
0111_2	オーバーフローのとき	V	vs

cond	条件	テスト (fcc)	アセンブリ言語表記
1000_2	常に分岐する	0, 1, 2, 3	a
0000_2	決して分岐しない	—	n
0111_2	比較不能のとき	3	u
0110_2	より大きいとき	2	g
0101_2	比較不能またはより大きいとき	2, 3	ug
0100_2	より小さいとき	1	l
0011_2	比較不能またはより小さいとき	1, 3	ul
0010_2	より小さいまたはより大きいとき	1, 2	lg
0001_2	等しくないとき	1, 2, 3	ne, nz
1001_2	等しいとき	0	e, z
1010_2	比較不能または等しいとき	0, 3	ue
1011_2	以上のとき	0, 2	ge
1100_2	比較不能または以上のとき	0, 2, 3	uge
1101_2	以下のとき	0, 1	le
1110_2	比較不能または以下のとき	0, 1, 3	ule
1111_2	比較可能のとき	0, 1, 2	o

opf_cc	命令	条件
000_2 001_2 010_2 011_2	FMOV{s|d|q}fcc	fcc0 fcc1 fcc2 fcc3
100_2	FMOV{s|d|q}icc	icc
110_2	FMOV{s|d|q}xcc	xcc

動作説明

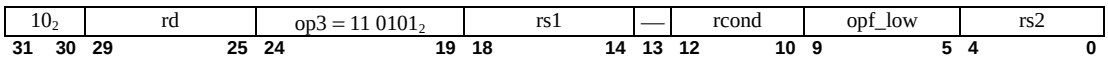
これらの命令は、opf_cc で指定された条件フラグが cond で指定された条件を満たしているとき、F[rs2]の内容を F[rd]に書き込む。条件フラグは更新されない。

これらの命令を実行すると、FSR.cexc および FSR.ftt には 0 がセットされ、FSR.aexc は更新されない。これらの命令はレジスタの内容により動作を変えないので、NaN は特別扱いされない。

例外	対象命令	検出条件
illegal_instruction	FMOV{S|D}icc, FMOV{S|D}xcc, FMOV{S|D}fcc	*reserved* フィールドが 0 でない
	FMOVQicc, FMOVQxcc, FMOVQfcc	常に この命令の *illegal_instruction* より優先度の低い例外は、エミュレーション用の仕様。
	—	下記のいずれかが成立している場合 • opf_cc = 101_2 • opf_cc = 111_2
fp_disabled	すべて	PSTATE.pef = 0 または FPRS.fef = 0
illegal_action	すべて	XAR.v = 1
fp_exception_other (FSR.ftt = *invalid_fp_register*)	FMOVQicc, FMOVQxcc, FMOVQfcc	下記のいずれかが成立している場合 • rs2<1> ≠ 0 • rd<1> ≠ 0

7.68. Move Floating-Point Register on Integer Register Condition (FMOVR)

命令	rcond	操作	HPC-ACE2 Regs	HPC-ACE2 SIMD	アセンブリ言語表記	
—	000_2	*reserved*			—	
FMOVR{s|d|q}Z	001_2	R[rs1] = 0 のとき移動			fmovr{s|d|q}z fmovr{s|d|q}e	reg_{rs1}, $freg_{rs2}$, $freg_{rd}$ reg_{rs1}, $freg_{rs2}$, $freg_{rd}$
FMOVR{s|d|q}LEZ	010_2	R[rs1] ≤ 0 のとき移動			fmovr{s|d|q}lez	reg_{rs1}, $freg_{rs2}$, $freg_{rd}$
FMOVR{s|d|q}LZ	011_2	R[rs1] < 0 のとき移動			fmovr{s|d|q}lz	reg_{rs1}, $freg_{rs2}$, $freg_{rd}$
—	100_2	*reserved*			—	
FMOVR{s|d|q}NZ	101_2	R[rs1] ≠ 0 のとき移動			fmovr{s|d|q}nz fmovr{s|d|q}ne	reg_{rs1}, $freg_{rs2}$, $freg_{rd}$ reg_{rs1}, $freg_{rs2}$, $freg_{rd}$
FMOVR{s|d|q}GZ	110_2	R[rs1] > 0 のとき移動			fmovr{s|d|q}gz	reg_{rs1}, $freg_{rs2}$, $freg_{rd}$
FMOVR{s|d|q}GEZ	111_2	R[rs1] ≥ 0 のとき移動			fmovr{s|d|q}gez	reg_{rs1}, $freg_{rs2}$, $freg_{rd}$



opf_low	精度
0 0101_2	単精度
0 0110_2	倍精度
0 0111_2	4 倍精度

動作説明

これらの命令は、整数レジスタ R[rs1]の値が rcond で指定される条件を満たしているとき、浮動小数点レジスタ F[rs2]の内容を F[rd]に移動する。R[rs1]の内容は符号付き整数として扱われる。CCR は更新されない。

これらの命令を実行すると、FSR.cexc と FSR.ftt にはゼロがセットされる。浮動小数点レジスタの値は丸められることはないし、NaN を特別に扱うこともしない。

例外	対象命令	検出条件
illegal_instruction	FMOVR{s|d}Z, FMOVR{s|d}LEZ, FMOVR{s|d}LZ, FMOVR{s|d}NZ, FMOVR{s|d}GZ, FMOVR{s|d}GEZ	*reserved* フィールドが 0 でないとき。
	FMOVRqZ, FMOVRqLEZ, FMOVRqLZ, FMOVRqNZ, FMOVRqGZ, FMOVRqGEZ	常に この命令の *illegal_instruction* より優先度の低い例外は、エミュレーション用の仕様。
	—	下記のいずれかが成立している場合 • rcond = 000_2 • rcond = 100_2 • opf_low が $0\ 0101_2$, $0\ 0110_2$, $0\ 0111_2$ 以外
fp_disabled	すべて	PSTATE.pef = 0 または FPRS.fef = 0
illegal_action	すべて	XAR.v = 1
fp_exception_other (FSR.ftt = *invalid_fp_register*)	FMOVRqZ, FMOVRqLEZ, FMOVRqLZ, FMOVRqNZ, FMOVRqGZ, FMOVRqGEZ	下記のいずれかが成立している場合 • rs2<1> ≠ 0 • rd<1> ≠ 0

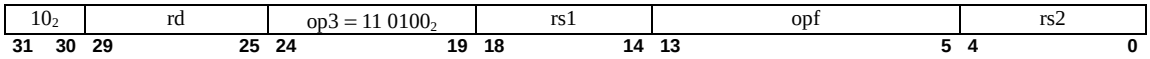
7.69. Partitioned Multiply Instructions

FMUL8x16, FMUL8x16AU, FMUL8x16AL, FMUL8SUx16, FMUL8ULx16, FMULD8SUx16, FMULD8ULx16 は UA2011 7.45 参照

例外	対象命令	検出条件
fp_disabled	すべて	PSTATE.pef = 0 または FPRS.fef = 0
illegal_action	すべて	XAR.v = 1

7.70. Floating-Point Multiply

命令	opf	操作	HPC-ACE2 Regs	HPC-ACE2 SIMD	アセンブリ言語表記	
FMULs	0 0100 1001_2	単精度の乗算	✓	✓	fmuls	$freg_{rs1}$, $freg_{rs2}$, $freg_{rd}$
FMULd	0 0100 1010_2	倍精度の乗算	✓	✓	fmuld	$freg_{rs1}$, $freg_{rs2}$, $freg_{rd}$
FMULq	0 0100 1011_2	4 倍精度の乗算	✓		fmulq	$freg_{rs1}$, $freg_{rs2}$, $freg_{rd}$
FsMULd	0 0110 1001_2	単精度を乗算し結果を倍精度で返す	✓	✓	fsmuld	$freg_{rs1}$, $freg_{rs2}$, $freg_{rd}$
FdMULq	0 0110 1110_2	倍精度を乗算し結果を 4 倍精度で返す	✓		fdmulq	$freg_{rs1}$, $freg_{rs2}$, $freg_{rd}$



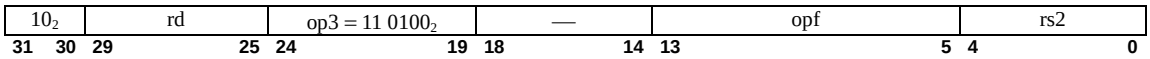
FMUL{s|d|q}命令は、F[rs1]と F[rs2]を浮動小数点数として乗算し、結果を F[rd]に書き込む。結果は FSR.rd または GSR.irnd にしたがって丸められる。

FsMULd 命令は、Fs[rs1]と Fs[rs2]を単精度浮動小数点数として乗算し、正確な結果を倍精度浮動小数点数として Fd[rd]に書き込む。丸めやオーバーフロー、アンダーフローは起こらない。同様に、FdMULq 命令は、Fd[rs1]と Fd[rs2]を倍精度浮動小数点数として乗算し、正確な結果を 4 倍精度浮動小数点数として Fq[rd]に書き込む。丸めやオーバーフロー、アンダーフローは起こらない。

例外		対象命令	検出条件
illegal_instruction		FMULq, FdMULq	常に この命令の *illegal_instruction* より優先度の低い例外は、エミュレーション用の仕様。
fp_disabled		すべて	PSTATE.pef = 0 または FPRS.fef = 0
illegal_action		FMULs, FMULd, FsMULd	XAR.v = 1 かつ、下記のいずれかが成立している場合 • XAR.urs3 ≠ 0 • XAR.simd = 1 かつ XAR.urs1<2> ≠ 0 • XAR.simd = 1 かつ XAR.urs2<2> ≠ 0 • XAR.simd = 1 かつ XAR.urd<2> ≠ 0
		FMULq, FdMULq	XAR.v = 1 かつ、下記のいずれかが成立している場合 • XAR.simd = 1 • XAR.urs3 ≠ 0
fp_exception_ieee_754	NV	すべて	IEEE 754 に準拠
	OF, UF, NX	FMULs, FMULd, FMULq	
fp_exception_other (FSR.ftt = *invalid_fp_register*)		FMULq	下記のいずれかが成立している場合 • rs1<1> ≠ 0 • rs2<1> ≠ 0 • rd<1> ≠ 0
		FdMULq	rd<1> ≠ 0
fp_exception_other (FSR.ftt = *unfinished_FPop*)		FMULs, FMULd, FsMULd	第 8 章を参照

7.71. Floating-Point Negative

命令	opf	操作	HPC-ACE2		アセンブリ言語表記	
			Regs.	SIMD		
FNEGs	$0\ 0000\ 0101_2$	単精度浮動小数点数の符号反転	✓	✓	fnegs	*freg*$_{rs2}$, *freg*$_{rd}$
FNEGd	$0\ 0000\ 0110_2$	倍精度浮動小数点数の符号反転	✓	✓	fnegd	*freg*$_{rs2}$,*freg*$_{rd}$
FNEGq	$0\ 0000\ 0111_2$	4 倍精度浮動小数点数の符号反転	✓		fnegq	*freg*$_{rs2}$,*freg*$_{rd}$



FNEG 命令は、浮動小数点数の符号ビットを反転させる命令である。FNEGs 命令は Fs[rs2]<31> を反転させた値を Fs[rd]に書き込む。FNEGd 命令は Fd[rs2]<63>を反転させた値を Fd[rd]に書き込む。FNEGq 命令は Fq[rs2]<127>を反転させた値を Fq[rd]に書き込む。入力が NaN であるかどうかによらず、符号ビットを反転させた値を出力する。

これらの命令を実行すると、FSR.cexc および FSR.ftt には 0 がセットされ、FSR.aexc は更新されない。

例外	対象命令	検出条件
illegal_instruction	FNEGs, FNEGd	*reserved* フィールドが 0 でない
	FNEGq	常に この命令の *illegal_instruction* より優先度の低い例外は、エミュレーション用の仕様。
fp_disabled	すべて	PSTATE.pef = 0 または FPRS.fef = 0
illegal_action	FNEGs, FNEGd	XAR.v = 1 かつ、下記のいずれかが成立している場合 • XAR.urs1 ≠ 0 • XAR.urs3 ≠ 0 • XAR.simd = 1 かつ XAR.urs2<2> ≠ 0 • XAR.simd = 1 かつ XAR.urd<2> ≠ 0
	FNEGq	XAR.v = 1 かつ、下記のいずれかが成立している場合 • XAR.simd = 1 • XAR.urs1 ≠ 0 • XAR.urs3 ≠ 0
fp_exception_other (FSR.ftt = *invalid_fp_register*)	FNEGq	下記のいずれかが成立している場合 • rs2<1> ≠ 0 • rd<1> ≠ 0

7.72. FPACK

FPACK16, FPACK32, FPACKFIX は UA2011 7.51 参照

例外		検出条件
illegal_instruction	FPACK16, FPACKFIX	iw<18:14> ≠ $0\ 0000_2$
fp_disabled	すべて	PSTATE.pef = 0 または FPRS.fef = 0
illegal_action	すべて	XAR.v = 1

7.73. Fixed-Point Partitioned Add

命令	opf	操作	HPC-ACE2		アセンブリ言語表記	
			Regs.	SIMD		
FPADD16	0 0101 0000_2	4 つの 16 ビット加算	✓	✓	fpadd16	*freg*$_{rs1}$, *freg*$_{rs2}$, *freg*$_{rd}$
FPADD16S	0 0101 0001_2	2 つの 16 ビット加算	✓	✓	fpadd16s	*freg*$_{rs1}$, *freg*$_{rs2}$, *freg*$_{rd}$
FPADD32	0 0101 0010_2	2 つの 32 ビット加算	✓	✓	fpadd32	*freg*$_{rs1}$, *freg*$_{rs2}$, *freg*$_{rd}$
FPADD32S	0 0101 0011_2	1 つの 32 ビット加算	✓	✓	fpadd32s	*freg*$_{rs1}$, *freg*$_{rs2}$, *freg*$_{rd}$

動作説明　　浮動小数点レジスタに格納された 16 ビット整数または 32 ビット整数の加算を行う。

FPADD16 命令は、Fd[rs1]の 4 つの 16 ビット整数と Fd[rs2]の 4 つの 16 ビット整数を、それぞれ同じ位置の要素同士で加算し、結果を Fd[rd]に格納する。

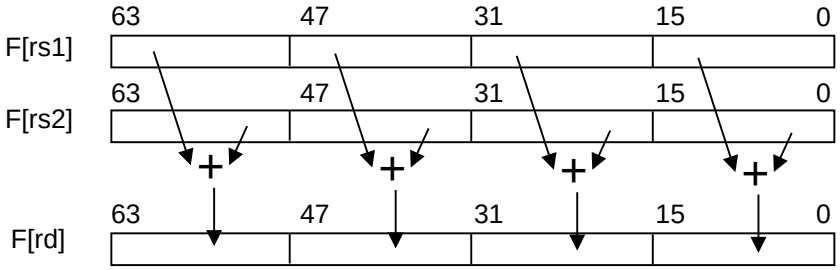
FPADD16S 命令は、Fs[rs1]の 2 つの 16 ビット整数と Fs[rs2]の 2 つの 16 ビット整数をそれぞれ同じ位置の要素同士で加算し、結果を Fs[rd]に格納する。

FPADD32 命令は、Fd[rs1]の 2 つの 32 ビット整数と Fd[rs2]の 2 つの 32 ビット整数をそれぞれ同じ位置の要素同士で加算し、結果を Fd[rd]に格納する。

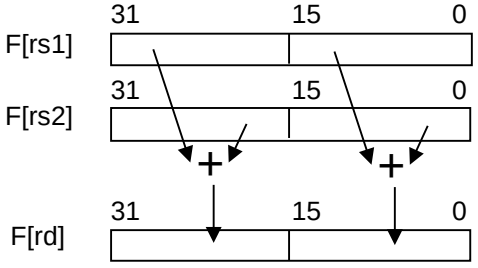
FPADD32S 命令は､Fs[rs1]の 1 つの 32 ビット整数と Fs[rs2]の 1 つの 32 ビット整数を加算し、結果を Fs[rd]に格納する。

FPADD{16|32}{S}命令は、FSR のどのフィールドも更新しない。

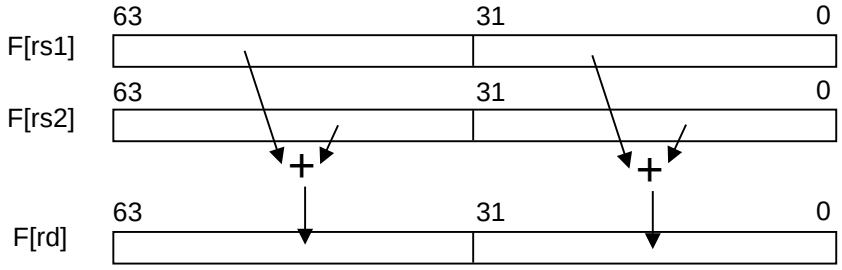
FPADD16 の動作



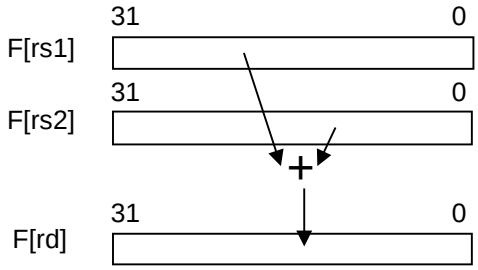
FPADD16S の動作



FPADD32 の動作



FPADD32S の動作



例外	検出条件
fp_disabled	PSTATE.pef = 0 または FPRS.fef = 0
illegal_action	XAR.v = 1 かつ、下記のいずれかが成立している場合 • XAR.urs3 ≠ 0 • XAR.simd = 1 かつ XAR.urs1<2> ≠ 0 • XAR.simd = 1 かつ XAR.urs2<2> ≠ 0 • XAR.simd = 1 かつ XAR.urd<2> ≠ 0

7.74. FPMERGE

FPMERGE は UA2011 7.57 参照

例外	検出条件
fp_disabled	PSTATE.pef = 0 または FPRS.fef = 0
illegal_action	XAR.v = 1

7.75. Fixed-point Partitioned Subtract (64-bit)

命令	opf	操作	HPC-ACE2		アセンブリ言語表記	
			Regs.	SIMD		
FPSUB16	0 0101 0100$_2$	4 つの 16 ビット減算	✓	✓	fpsub16	*freg*$_{rs1}$, *freg*$_{rs2}$, *freg*$_{rd}$
FPSUB16S	0 0101 0101$_2$	2 つの 16 ビット減算	✓	✓	fpsub16s	*freg*$_{rs1}$, *freg*$_{rs2}$, *freg*$_{rd}$
FPSUB32	0 0101 0110$_2$	2 つの 32 ビット減算	✓	✓	fpsub32	*freg*$_{rs1}$, *freg*$_{rs2}$, *freg*$_{rd}$
FPSUB32S	0 0101 0111$_2$	1 つの 32 ビット減算	✓	✓	fpsub32s	*freg*$_{rs1}$, *freg*$_{rs2}$, *freg*$_{rd}$

動作説明　　浮動小数点レジスタに格納された 16 ビット整数または 32 ビット整数の減算を行う。

FPSUB16 命令は、Fd[rs1]の 4 つの 16 ビット整数から Fd[rs2]の 4 つの 16 ビット整数を、それぞれ同じ位置の要素同士で減算し、結果を Fd[rd]に格納する。

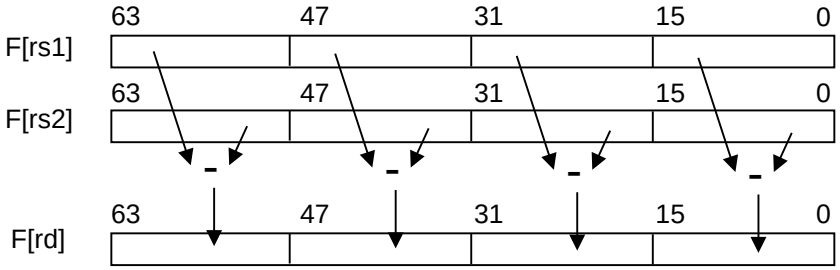
FPSUB16S 命令は、Fs[rs1]の 2 つの 16 ビット整数から Fs[rs2]の 2 つの 16 ビット整数をそれぞれ同じ位置の要素同士で減算し、結果を Fs[rd]に格納する。

FPSUB32 命令は、Fd[rs1]の 2 つの 32 ビット整数から Fd[rs2]の 2 つの 32 ビット整数をそれぞれ同じ位置の要素同士で減算し、結果を Fd[rd]に格納する。

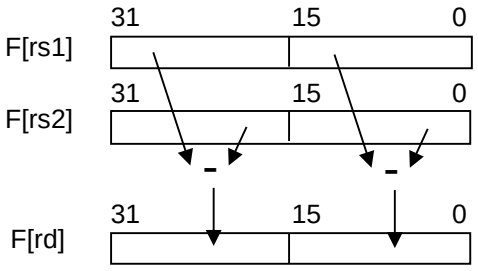
FPSUB32S 命令は、Fs[rs1]の 1 つの 32 ビット整数から Fs[rs2]の 1 つの 32 ビット整数を減算し、結果を Fs[rd]に格納する。

FPSUB{16|32}{S}命令は、FSR のどのフィールドも更新しない。

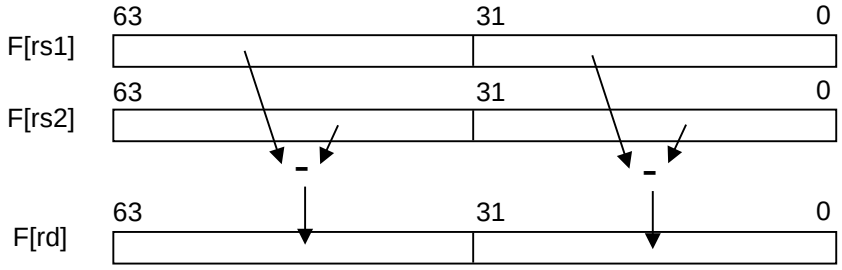
FPSUB16 の動作



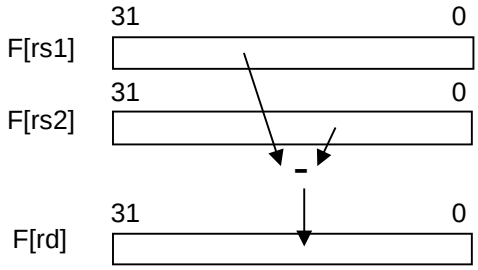
FPSUB16S の動作



FPSUB32 の動作



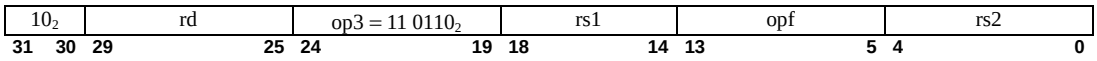
FPSUB32S の動作



例外	検出条件
fp_disabled	PSTATE.pef = 0 または FPRS.fef = 0
illegal_action	XAR.v = 1 かつ、下記のいずれかが成立している場合 • XAR.urs3 ≠ 0 • XAR.simd = 1 かつ XAR.urs1<2> ≠ 0 • XAR.simd = 1 かつ XAR.urs2<2> ≠ 0 • XAR.simd = 1 かつ XAR.urd<2> ≠ 0

7.76. F Register Logical Operate

命令	opf	操作	HPC-ACE2		アセンブリ言語表記	
			Regs.	SIMD		
FZERO	0 0110 0000_2	Fd[rd]の全ビットに'0'をセットする	✓	✓	fzero	$freg_{rd}$
FZEROs	0 0110 0001_2	Fs[rd]の全ビットに'0'をセットする	✓	✓	fzeros	$freg_{rd}$
FONE	0 0111 1110_2	Fd[rd]の全ビットに'1'をセットする	✓	✓	fone	$freg_{rd}$
FONEs	0 0111 1111_2	Fs[rd]の全ビットに'1'をセットする	✓	✓	fones	$freg_{rd}$
FSRC1	0 0111 0100_2	Fd[rs1]を Fd[rd]にコピーする	✓	✓	fsrc1	$freg_{rs1}$, $freg_{rd}$
FSRC1s	0 0111 0101_2	Fs[rs1]を Fs[rd]にコピーする	✓	✓	fsrc1s	$freg_{rs1}$, $freg_{rd}$
FSRC2	0 0111 1000_2	Fd[rs2]を Fd[rd]にコピーする	✓	✓	fsrc2	$freg_{rs2}$, $freg_{rd}$
FSRC2s	0 0111 1001_2	Fs[rs2]を Fs[rd]にコピーする	✓	✓	fsrc2s	$freg_{rs2}$, $freg_{rd}$
FNOT1	0 0110 1010_2	Fd[rs1]の全ビットを反転させる	✓	✓	fnot1	$freg_{rs1}$, $freg_{rd}$
FNOT1s	0 0110 1011_2	Fs[rs1]の全ビットを反転させる	✓	✓	fnot1s	$freg_{rs1}$, $freg_{rd}$
FNOT2	0 0110 0110_2	Fd[rs2]の全ビットを反転させる	✓	✓	fnot2	$freg_{rs2}$, $freg_{rd}$
FNOT2s	0 0110 0111_2	Fs[rs2] の全ビットを反転させる	✓	✓	fnot2s	$freg_{rs2}$, $freg_{rd}$
FOR	0 0111 1100_2	Fd[rs1]と Fd[rs2]の論理 OR	✓	✓	for	$freg_{rs1}$, $freg_{rs2}$, $freg_{rd}$
FORs	0 0111 1101_2	Fs[rs1]と Fs[rs2]の論理 OR	✓	✓	fors	$freg_{rs1}$, $freg_{rs2}$, $freg_{rd}$
FNOR	0 0110 0010_2	Fd[rs1]と Fd[rs2]の論理 NOR	✓	✓	fnor	$freg_{rs1}$, $freg_{rs2}$, $freg_{rd}$
FNORs	0 0110 0011_2	Fs[rs1]と Fs[rs2]の論理 NOR	✓	✓	fnors	$freg_{rs1}$, $freg_{rs2}$, $freg_{rd}$
FAND	0 0111 0000_2	Fd[rs1]と Fd[rs2]の論理 AND	✓	✓	fand	$freg_{rs1}$, $freg_{rs2}$, $freg_{rd}$
FANDs	0 0111 0001_2	Fs[rs1]と Fs[rs2]の論理 AND	✓	✓	fands	$freg_{rs1}$, $freg_{rs2}$, $freg_{rd}$
FNAND	0 0110 1110_2	Fd[rs1]と Fd[rs2]の論理 NAND	✓	✓	fnand	$freg_{rs1}$, $freg_{rs2}$, $freg_{rd}$
FNANDs	0 0110 1111_2	Fs[rs1]と Fs[rs2]の論理 NAND	✓	✓	fnands	$freg_{rs1}$, $freg_{rs2}$, $freg_{rd}$
FXOR	0 0110 1100_2	Fd[rs1]と Fd[rs2]の論理 XOR	✓	✓	fxor	$freg_{rs1}$, $freg_{rs2}$, $freg_{rd}$
FXORs	0 0110 1101_2	Fs[rs1]と Fs[rs2]の論理 XOR	✓	✓	fxors	$freg_{rs1}$, $freg_{rs2}$, $freg_{rd}$
FXNOR	0 0111 0010_2	Fd[rs1]と Fd[rs2]の論理 XNOR	✓	✓	fxnor	$freg_{rs1}$, $freg_{rs2}$, $freg_{rd}$
FXNORs	0 0111 0011_2	Fs[rs1]と Fs[rs2]の論理 XNOR	✓	✓	fxnors	$freg_{rs1}$, $freg_{rs2}$, $freg_{rd}$
FORNOT1	0 0111 1010_2	(**not** Fd[rs1]) **or** Fd[rs2]	✓	✓	fornot1	$freg_{rs1}$, $freg_{rs2}$, $freg_{rd}$
FORNOT1s	0 0111 1011_2	(**not** Fs[rs1]) **or** Fs[rs2]	✓	✓	fornot1s	$freg_{rs1}$, $freg_{rs2}$, $freg_{rd}$
FORNOT2	0 0111 0110_2	Fd[rs1] **or** (**not** Fd[rs2])	✓	✓	fornot2	$freg_{rs1}$, $freg_{rs2}$, $freg_{rd}$
FORNOT2s	0 0111 0111_2	Fs[rs1] **or** (**not** Fs[rs2])	✓	✓	fornot2s	$freg_{rs1}$, $freg_{rs2}$, $freg_{rd}$
FANDNOT1	0 0110 1000_2	(**not** Fd[rs1]) **and** Fd[rs2]	✓	✓	fandnot1	$freg_{rs1}$, $freg_{rs2}$, $freg_{rd}$
FANDNOT1s	0 0110 1001_2	(**not** Fs[rs1]) **and** Fs[rs2]	✓	✓	fandnot1s	$freg_{rs1}$, $freg_{rs2}$, $freg_{rd}$
FANDNOT2	0 0110 0100_2	Fd[rs1] **and** (**not** Fd[rs2])	✓	✓	fandnot2	$freg_{rs1}$, $freg_{rs2}$, $freg_{rd}$
FANDNOT2s	0 0110 0101_2	Fs[rs1] **and** (**not** Fs[rs2])	✓	✓	fandnot2s	$freg_{rs1}$, $freg_{rs2}$, $freg_{rd}$

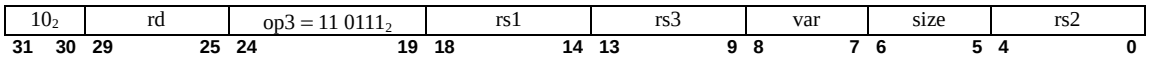


動作説明　　これらの命令は､浮動小数点レジスタに対する論理操作を行う｡操作は 16 種類定義されており､それぞれ 64 ビット用と 32 ビット用がある。すべての命令が、ビット毎の操作になっている。結果は F[rd]に格納される。

例外	対象命令	検出条件
fp_disabled	すべて	PSTATE.pef = 0 または FPRS.fef = 0
illegal_instruction	FZERO, FZEROs, FONE, FONEs	iw<18:14> ≠ 0 00000_2 または iw<4:0> ≠ 0 0000_2
	FSRC1, FSRC1s, FNOT1, FNOT1s	iw<4:0> ≠ 0 0000_2
	FSRC2, FSRC2s, FNOT2, FNOT2s	iw<18:14> ≠ 0 0000_2
illegal_action	FZERO, FZEROs, FONE, FONEs	XAR.v = 1 かつ、下記のいずれかが成立している場合 • XAR.urs1 ≠ 0 • XAR.urs2 ≠ 0 • XAR.urs3 ≠ 0 • XAR.simd = 1 かつ XAR.urd<2> ≠ 0
	FSRC1, FSRC1s, FNOT1, FNOT1s	XAR.v = 1 かつ 、下記のいずれかが成立している場合 • XAR.urs2 ≠ 0 • XAR.urs3 ≠ 0 • XAR.simd = 1 かつ XAR.urs1<2> ≠ 0 • XAR.simd = 1 かつ XAR.urd<2> ≠ 0
	FSRC2, FSRC2s, FNOT2, FNOT2s	XAR.v = 1 かつ、下記のいずれかが成立している場合 • XAR.urs1 ≠ 0 • XAR.urs3 ≠ 0 • XAR.simd = 1 かつ XAR.urs2<2> ≠ 0 • XAR.simd = 1 かつ XAR.urd<2> ≠ 0
	FOR, FORs, FNOR, FNORs, FAND, FANDs, FNAND, FNANDs, FXOR, FXORs, FXNOR, FXNORs, FORNOT1, FORNOT1s, FORNOT2, FORNOT2s, FANDNOT1, FANDNOT1s, FANDNOT2, FANDNOT2s	XAR.v = 1 かつ、下記のいずれかが成立している場合 • XAR.urs3 ≠ 0 • XAR.simd = 1 かつ XAR.urs1<2> ≠ 0 • XAR.simd = 1 かつ XAR.urs2<2> ≠ 0 • XAR.simd = 1 かつ XAR.urd<2> ≠ 0

7.77. Move Selected Floating-Point Register on Floating-Point Register's Condition

命令	var	size	操作	HPC-ACE2 Regs	HPC-ACE2 SIMD	アセンブリ言語表記
FSELMOVd	11_2	00_2	倍精度レジスタを選択し移動	✓	✓	fselmovd *freg*$_{rs1}$, *freg*$_{rs2}$, *freg*$_{rs3}$, *freg*$_{rd}$
FSELMOVs	11_2	11_2	単精度レジスタを選択し移動	✓	✓	fselmovs *freg*$_{rs1}$, *freg*$_{rs2}$, *freg*$_{rs3}$, *freg*$_{rd}$



動作説明

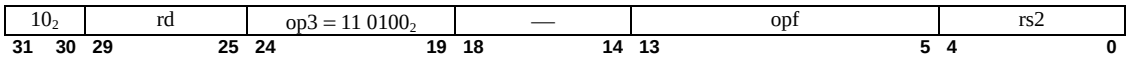
FSELMOV{s|d}は、F[rs3]の最上位ビットにより F[rs1]か F[rs2]を選択し、その値を F[rd]に格納する。

FSELMOVd では、Fd[rs3]<63>が 1 なら Fd[rs1]を、0 なら Fd[rs2]を選択する。FSELMOVs では、Fs[rs3]<31>が 1 なら Fs[rs1]を、0 なら Fs[rs2]を選択する。

例外	対象命令	検出条件
fp_disabled	すべて	PSTATE.pef = 0 または FPRS.fef = 0
illegal_action	すべて	XAR.v = 1 かつ、下記のいずれかが成立している場合 • XAR.simd = 1 かつ XAR.urs1<2> ≠ 0 • XAR.simd = 1 かつ XAR.urs2<2> ≠ 0 • XAR.simd = 1 かつ XAR.urs3<2> ≠ 0 • XAR.simd = 1 かつ XAR.urd<2> ≠ 0

7.78. Floating-Point Square Root

命令	opf	操作	HPC-ACE2 Regs. SIMD	アセンブリ言語表記	
FSQRTs	0 0010 1001$_2$	単精度浮動小数点数の平方根	☆	fsqrts	*freg*$_{rs2}$, *freg*$_{rd}$
FSQRTd	0 0010 1010$_2$	倍精度浮動小数点数の平方根	☆	fsqrtd	*freg*$_{rs2}$,*freg*$_{rd}$
FSQRTq	0 0010 1011$_2$	4 倍精度浮動小数点数の平方根	☆	fsqrtq	*freg*$_{rs2}$,*freg*$_{rd}$



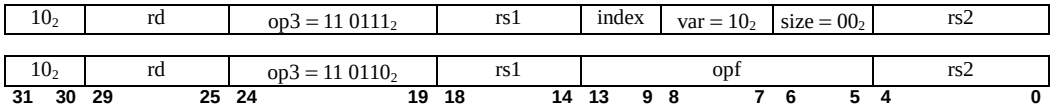
動作説明

FSQRT 命令は、F[rs2]の浮動小数点数の平方根を F[rd]に書き込む。FSQRT 命令によって実行される丸めの方法は、FSR.rd または GSR.irnd によって決まる。

例外		対象命令	検出条件
illegal_instruction		FSQRTs, FSQRTd	*reserved* フィールドが 0 でない
		FSQRTq	常に この命令の *illegal_instruction* より優先度の低い例外は、エミュレーション用の仕様。
fp_disabled		すべて	PSTATE.pef = 0 または FPRS.fef = 0
illegal_action		すべて	XAR.v = 1 かつ、下記のいずれかが成立している場合 • XAR.simd = 1 • XAR.urs1 ≠ 0 • XAR.urs3 ≠ 0 • XAR.urd<2> ≠ 0
fp_exception_ieee_754	NV, NX	すべて	IEEE 754 に準拠
fp_exception_other (FSR.ftt = *invalid_fp_register*)		FSQRTq	下記のいずれかが成立している場合 • rs2<1> ≠ 0 • rd<1> ≠ 0
fp_exception_other (FSR.ftt = *unfinished_FPop*)		FSQRTs, FSQRTd	第 8 章を参照

7.79. Floating-Point Trigonometric Functions

命令	op3	opf	操作	HPC-ACE2		アセンブリ言語表記	
				Regs	SIMD		
FTRIMADDd	$11\ 0111_2$	—	三角関数補助演算	✓	✓	ftrimaddd	$freg_{rs1}$, $freg_{rs2}$, *index*, $freg_{rd}$
FTRISMULd	$11\ 0110_2$	$1\ 0111\ 1010_2$	FTRIMADDd の初期値算出	✓	✓	ftrismuld	$freg_{rs1}$, $freg_{rs2}$, $freg_{rd}$
FTRISSELd	$11\ 0110_2$	$1\ 0111\ 1000_2$	FTRIMADDd の初期値算出	✓	✓	ftrisseld	$freg_{rs1}$, $freg_{rs2}$, $freg_{rd}$



命令	演算
FTRIMADDd	Fd[rd] ← Fd[rs1] × abs(Fd[rs2]) + T[Fd[rs2]<63>][index]
FTRISMULd	Fd[rd] ← (Fd[rs2]<0> << 63) ^ (Fd[rs1] × Fd[rs1])
FTRISSELd	Fd[rd] ← (Fd[rs2]<1> << 63) ^ (Fd[rs2]<0> ? 1.0 : Fd[rs1])

動作説明

これらの 3 命令は sin(x)を求めるための級数計算の補助命令である。FTRIMADDd は、$-\pi/4 < x \leq \pi/4$ の範囲でテーラー級数の級数部分の計算を行い、FTRISMULd と FTRISSELd は、任意の x について sin(x) を求めるため FTRIMADDd の前処理を行う。これらの命令は倍精度命令のみが定義されている。図 7-1 にこれら 3 命令が行う計算式を示す。

$$\sin x \cong x - \frac{1}{3!}x^3 + \frac{1}{5!}x^5 - \frac{1}{7!}x^7 + \frac{1}{9!}x^9 - \frac{1}{11!}x^{11} + \frac{1}{13!}x^{13} - \frac{1}{15!}x^{15}$$

$$= x\left(1 - \frac{1}{3!}x^2 + \frac{1}{5!}x^4 - \frac{1}{7!}x^6 + \frac{1}{9!}x^8 - \frac{1}{11!}x^{10} + \frac{1}{13!}x^{12} - \frac{1}{15!}x^{14}\right)$$

$$= x \cdot \underbrace{\left(\left(\left(\left(\left(\left(\left(0 \cdot x^2 - \frac{1}{15!}\right)x^2 + \frac{1}{13!}\right)x^2 - \frac{1}{11!}\right)x^2 + \frac{1}{9!}\right)x^2 - \frac{1}{7!}\right)x^2 + \frac{1}{5!}\right)x^2 - \frac{1}{3!}\right)x^2 + 1\right)}$$

$$\cos x \cong 1 - \frac{1}{2!}x^2 + \frac{1}{4!}x^4 - \frac{1}{6!}x^6 + \frac{1}{8!}x^8 - \frac{1}{10!}x^{10} + \frac{1}{12!}x^{12} - \frac{1}{14!}x^{14}$$

$$= 1 \cdot \underbrace{\left(\left(\left(\left(\left(\left(\left(0 \cdot x^2 - \frac{1}{14!}\right)x^2 + \frac{1}{12!}\right)x^2 - \frac{1}{10!}\right)x^2 + \frac{1}{8!}\right)x^2 - \frac{1}{6!}\right)x^2 + \frac{1}{4!}\right)x^2 - \frac{1}{2!}\right)x^2 + 1\right)}$$

図 7-1　三角関数補助演算

FTRIMADDd は、Fd[rs1]と、Fd[rs2]の絶対値を乗じ、さらに演算器内にあるテーブルから index で指定される倍精度数を取り出して加算し、結果を Fd[rd]に格納する。FTRIMADDd は sin(x), cos(x)の級数部分の計算を行う。

FTRISMULd は、Fd[rs1]を二乗した値に、Fd[rs2]<0>を符号として結合し、結果を Fd[rd]に格納する。FTRISMULd は FTRIMADDd の初期値を求めるために使われる。

FTRISSELd は、Fd[rs2]<0>により Fd[rs1]か 1.0 を選択し、その値に Fd[rs2]<1>を符合として排他的論理和（exclusive or）を取り、結果を Fd[rd]に格納する。FTRISSELd は級数部分の計算結果に乗ずる最後の項を求めるために使われる。

FTRIMADDd は sin(x), cos(x)の級数部分の計算を行う。初期値として Fd[rs1]にゼロ、Fd[rs2]に $-\pi/4 < x \leq \pi/4$ の x^2 をセットし、FTRIMADDd を 8 回実行すると、級数部分が倍精度浮動小数点数として十分な精度で計算できる。図 7-1 の式からわかる通り、sin(x)と cos(x)の級数部分は係数が異なるだけなので、FTRIMADDd は、rs2 の符号をどちらの係数を使うかの識別に利用する。

- Fd[rs2]<63> = 0 のとき、sin(x)の係数テーブルを使う
- Fd[rs2]<63> = 1 のとき、con(x)の係数テーブルを使う

FTRIMADDd は下の例の使われ方を想定しており、この使い方のときに精度誤差が小さくなるような係数を採用しているため、係数が数学的に正しいものとは異なっている。表 7-21、表 7-22 に FTRIMADDd の係数テーブルの内容を示す。

表 7-21　sin(x) (Fd[rs2]<63> = 0)の係数テーブル

Index	演算に使われる係数		数学的に正しい係数
	16 進表記	10 進表記	
0	3ff0 0000 0000 0000_{16}	1.0	= 1/1!
1	bfc5 5555 5555 5543_{16}	–0.1666666666666661	> –1/3!
2	3f81 1111 1110 $f30c_{16}$	0.8333333333320002e–02	< 1/5!
3	bf2a 01a0 19b9 $2fc6_{16}$	–0.1984126982840213e–03	> –1/7!
4	3ec7 1de3 51f3 $d22b_{16}$	0.2755731329901505e–05	< 1/9!
5	be5a e5e2 b60f $7b91_{16}$	–0.2505070584637887e–07	> –1/11!
6	3de5 d840 8868 $552f_{16}$	0.1589413637195215e–09	< 1/13!
7	0000 0000 0000 0000_{16}	0	> –1/15!

表 7-22　cos(x) (Fd[rs2]<63> = 1)の係数テーブル

Index	演算に使われる係数		数学的に正しい係数
	16 進表記	10 進表記	
0	3ff0 0000 0000 0000_{16}	1.0	= 1/0!
1	bfe0 0000 0000 0000_{16}	–0.5000000000000000	= –1/2!
2	3fa5 5555 5555 5536_{16}	0.4166666666666645e–01	< 1/4!
3	bf56 c16c 16c1 $3a0b_{16}$	–0.1388888888886111e–02	> –1/6!
4	3efa 01a0 19b1 $e8d8_{16}$	0.2480158728388683e–04	< 1/8!
5	be92 7e4f 7282 $f468_{16}$	–0.2755731309913950e–06	> –1/10!
6	3e21 ee96 d264 $1b13_{16}$	0.2087558253975872e–08	< 1/12!
7	bda8 f763 80fb $b401_{16}$	–0.1135338700720054e–10	> –1/14!

FTRISMULd は FTRIMADDd の初期値を計算する。Fd[rs1]に x、Fd[rs2]に図 7-2 の Q を与えると、x^2 の値に、係数テーブル選択するための情報を符号として付加した値を返す。Q は浮動小数点数ではなく整数で与える。Fd[rs2]<63:1>は使われない。Fd[rs2]が NaN であっても例外を検出しない。

FTRISSELd は FTRIMADDd の結果に乗ずる値を計算する。Fd[rs1]に x、Fd[rs2]に図 7-2 の Q を与えると、x または 1.0 のどちらかに適切な符号をつけた値を返す。Q は浮動小数点数ではなく整数で与える。Fd[rs2]<63:2>は使われない。Fd[rs2]が NaN であっても例外を検出しない。

$$q : (2q-1)\cdot\frac{\pi}{4} < x \le (2q+1)\cdot\frac{\pi}{4}$$

$$Q : q \bmod 4$$

$$R : x - q\cdot\frac{\pi}{2}\left(-\frac{\pi}{4} < R \le \frac{\pi}{4}\right)$$

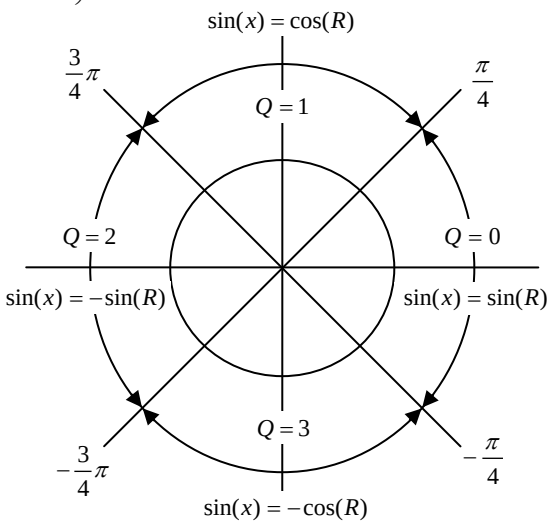


図 7-2　三角関数補助演算を用いた sin(x)の計算

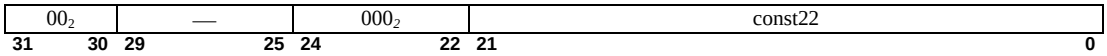
Example: sin(x)を求める

```
/*
 * 入力値: x
 * q: (2q-1)*π/4 < x <= (2q+1)*π/4 なる q
 * Q: q%4
 * R: x - q * π/2
 */

ftrismuld R, Q, M
ftrisseld R, Q, N

/*
 * M ←R2[63]=table_type, R2[62:0]=R2
 * (R2 は必ず正となるので sign ビット(bit63) は常に 0 となる。
 * この sign ビットを ftrimaddd の table_type として使う)
 * N ←最後に掛ける値 (1.0 or R) * sign
 * S ←0
 */

ftrimaddd S, M, 7, S
ftrimaddd S, M, 6, S
ftrimaddd S, M, 5, S
ftrimaddd S, M, 4, S
ftrimaddd S, M, 3, S
ftrimaddd S, M, 2, S
ftrimaddd S, M, 1, S
ftrimaddd S, M, 0, S
fmuld S, N, S

/*
 * S ←結果
 */
```

例外		対象命令	検出条件
illegal_instruction		FTRIMADDd	index > 7
fp_disabled		すべて	PSTATE.pef = 0 または FPRS.fef = 0
illegal_action		すべて	XAR.v = 1 かつ、下記のいずれかが成立している場合 • XAR.urs3 ≠ 0 • XAR.simd = 1 かつ XAR.urs1<2> ≠ 0 • XAR.simd = 1 かつ XAR.urs2<2> ≠ 0 • XAR.simd = 1 かつ XAR.urd<2> ≠ 0
fp_exception_ieee754	NV	FTRIMADDd, FTRISMULd	IEEE 754 に準拠 FTRISMULd は rs1 のみ
	NX	FTRIMADDd, FTRISMULd	IEEE 754 に準拠
	OF	FTRIMADDd, FTRISMULd	IEEE 754 に準拠
	UF	FTRIMADDd, FTRISMULd	IEEE 754 に準拠
fp_exception_other (FSR.ftt = *unfinished_FPop*)		FTRIMADDd, FTRISMULd	

7.80. Illegal Instruction Trap

命令	操作	HPC-ACE2		アセンブリ言語表記
		Regs.	SIMD	
ILLTRAP	*illegal_instruction* 例外を発生させる			illtrap



動作説明

ILLTRAP 命令は、実行すると *illegal_instruction* 例外を発生する命令である。const22 は無視される。このフィールドは、将来のアーキテクチャ用に予約されてはいない。

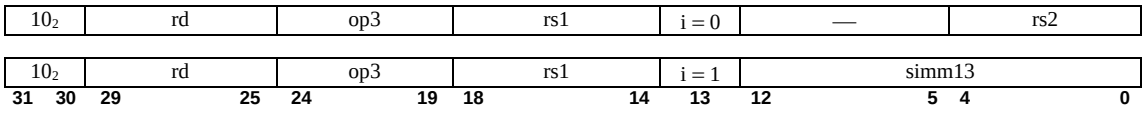
Compatibility Note 命令の名前以外は、SPARC V8 の UNIMP 命令と同じである。

reserved が 0 でない場合は *illegal_instruction* 例外が発生する。しかし、将来のアーキテクチャでは変更されるかもしれないので、ソフトウェアはこの仕様を前提としてはいけない。

例外	対象命令	検出条件
illegal_instruction	すべて	常に

7.81. Integer Logical Operation

命令	op3	操作	HPC-ACE2		アセンブリ言語表記	
			Regs.	SIMD		
AND	$00\ 0001_2$	論理積	✓		and	*reg$_{rs1}$, reg_or_imm, reg$_{rd}$*
ANDcc	$01\ 0001_2$	論理積と cc の更新	✓		andcc	*reg$_{rs1}$, reg_or_imm, reg$_{rd}$*
ANDN	$00\ 0101_2$	入力 2 の論理否定との論理積	✓		andn	*reg$_{rs1}$, reg_or_imm, reg$_{rd}$*
ANDNcc	$01\ 0101_2$	入力 2 の論理否定との論理積と cc の更新	✓		andncc	*reg$_{rs1}$, reg_or_imm, reg$_{rd}$*
OR	$00\ 0010_2$	論理和	✓		or	*reg$_{rs1}$, reg_or_imm, reg$_{rd}$*
ORcc	$01\ 0010_2$	論理和と cc の更新	✓		orcc	*reg$_{rs1}$, reg_or_imm, reg$_{rd}$*
ORN	$00\ 0110_2$	入力 2 の論理否定との論理和	✓		orn	*reg$_{rs1}$, reg_or_imm, reg$_{rd}$*
ORNcc	$01\ 0110_2$	入力 2 の論理否定との論理和と cc の更新	✓		orncc	*reg$_{rs1}$, reg_or_imm, reg$_{rd}$*
XOR	$00\ 0011_2$	排他論理和	✓		xor	*reg$_{rs1}$, reg_or_imm, reg$_{rd}$*
XORcc	$01\ 0011_2$	排他論理和と cc の更新	✓		xorcc	*reg$_{rs1}$, reg_or_imm, reg$_{rd}$*
XNOR	$00\ 0111_2$	排他論理和の論理否定	✓		xnor	*reg$_{rs1}$, reg_or_imm, reg$_{rd}$*
XNORcc	$01\ 0111_2$	排他論理和の論理否定と cc の更新	✓		xnorcc	*reg$_{rs1}$, reg_or_imm, reg$_{rd}$*



動作説明　これらの整数論理演算命令は、ビット単位の論理演算を行う。i = 0 のときは、“R[rs1] **op** R[rs2]” を処理し、i = 1 のときは、 “R[rs1] **op** sign_ext(simm13)” を処理する。結果は、R[rd]に書かれる。表 7-23 にビット単位の論理演算の真偽値を示す。

表 7-23　ビット単位の真偽値

入力 1	0	0	1	1
入力 2	0	1	0	1
AND, ANDcc	0	0	0	1
ANDN, ANDNcc	0	0	1	0
OR, ORcc	0	1	1	1
ORN, ORNcc	1	0	1	1
XOR, XORcc	0	1	1	0
XNOR, XNORcc	1	0	0	1

Note　ANDN, ANDNcc, ORN, および ORNcc 命令は、それぞれ AND, ANDcc, OR, および ORcc 命令を、入力 2 を論理否定して実行する。XNOR および XNORcc 命令は、XOR および XORcc 命令の結果を論理否定するが、ANDN, ANDNcc, ORN, および ORNcc 命令と同様、入力 2 を論理否定して実行すると考えても真偽値は同じ。

ANDcc, ANDNcc, ORcc, ORNcc, XORcc, および XNORcc 命令は、整数条件コード（CCR.icc と CCR.xcc）を変更する。条件コードは、以下の通りにセットされる：

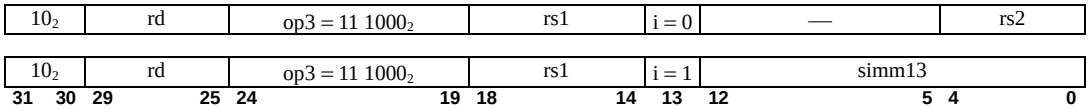
- icc.v, icc.c, xcc.v, xcc.c には 0 がセットされる
- icc.n には、結果のビット 31 がコピーされる。

- xcc.n には、結果のビット 63 がコピーされる。
- icc.z には、結果のビット<31:0>がゼロならば 1 がセットされる。そうでなければ 0 がセットされる。
- xcc.z には、結果の全 64 ビットがゼロならば 1 がセットされる。そうでなければ 0 がセットされる。

例外	対象命令	検出条件
illegal_instruction	すべて	*reserved* フィールドが 0 でない (i = 0 かつ iw<12:5> ≠ $0000\ 0000_2$)
illegal_action	すべて	XAR.v = 1 かつ、下記のいずれかが成立している場合 • XAR.simd = 1 • XAR.urs1<2:1> ≠ 00_2 • i = 0 かつ XAR.urs2<2:1> ≠ 00_2 • i = 1 かつ XAR.urs2 ≠ 0 • XAR.urs3 ≠ 0 • XAR.urd<2:1> ≠ 00_2

7.82. Jump and Link

命令	op3	操作	HPC-ACE2 Regs SIMD	アセンブリ言語表記
JMPL	$11\ 1000_2$	レジスタ間接ジャンプ		jmpl *address*, *reg*$_{rd}$



動作説明

JMPL 命令は遅延スロットを有すレジスタ間接ジャンプ命令である。i = 0 のときに “R[rs1] + R[rs2]” に、i = 1 のときに “R[rs1] + sign_ext(simm13)” に PC を更新する。同時にカレント PC を R[rd]にコピーする。

PSTATE.am = 1 のとき、R[rd]には PC の上位 32 ビットを 0 にした値が設定される。R[rd]の更新は直ちに行われ、遅延スロットの命令には新しい値が見える。

ジャンプ先アドレスの下位 2 ビットが 0 でない場合、*mem_address_not_aligned* 例外が発生する。

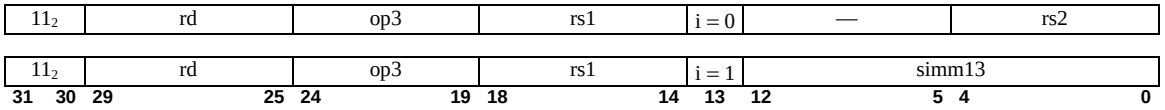
Programming Note rd = 15 に指定すると、CALL 命令と同様に o7 レジスタにコピーされる。

rd = 0 の JMPL は関数からの復帰に用いることができる。リーフでない関数(SAVE 命令を使用する関数)からの復帰では R[31] + 8 に復帰し、リーフ関数(SAVE 命令を使用しない関数) からの復帰では R[15] + 8 に復帰する。

例外	検出条件
illegal_instruction	*reserved* フィールドが 0 でない
illegal_action	XAR.v = 1
mem_address_not_aligned	ジャンプ先アドレスの下位 2 ビットが 0 でないとき

7.83. Load Integer

命令	op3	操作	HPC-ACE2 Regs	HPC-ACE2 SIMD	アセンブリ言語表記	
LDSB	$00\ 1001_2$	符号付き 1 バイトデータの読み出し	✓		ldsb	[*address*], *reg*$_{rd}$
LDSH	$00\ 1010_2$	符号付き 2 バイトデータの読み出し	✓		ldsh	[*address*], *reg*$_{rd}$
LDSW	$00\ 1000_2$	符号付き 4 バイトデータの読み出し	✓		ldsw	[*address*], *reg*$_{rd}$
LDUB	$00\ 0001_2$	符号なし 1 バイトデータの読み出し	✓		ldub	[*address*], *reg*$_{rd}$
LDUH	$00\ 0010_2$	符号なし 2 バイトデータの読み出し	✓		lduh	[*address*], *reg*$_{rd}$
LDUW	$00\ 0000_2$	符号なし 4 バイトデータの読み出し	✓		lduw ld	[*address*], *reg*$_{rd}$ [*address*], *reg*$_{rd}$
LDX	$00\ 1011_2$	8 バイトデータの読み出し	✓		ldx	[*address*], *reg*$_{rd}$



動作説明

整数ロード命令は、メモリから 1,2,4, および 8 バイトのデータを読み出して R[rd]に格納する。読み出した 1,2,4, および 8 バイトデータは、命令により符号拡張またはゼロ拡張される。

整数ロード命令は、暗黙の ASI (UA2011 6.3.1.3 参照) を使いメモリにアクセスする。読み出すアドレスは、i = 0 のときは “R[rs1] + R[rs2]” で、i = 1 のときは “R[rs1] + sign_ext(simm13)” で計算される。

整数ロード命令が正常に実行されるとき、メモリアクセスはアトミックに行なわれる。

LDUH, LDSH 命令は、2 バイト境界にないアドレスにアクセスすると *mem_address_not_aligned* 例外を検出する。LDUW, LDSW 命令は、4 バイト境界にないアドレスにアクセスすると *mem_address_not_aligned* 例外を検出する。LDX 命令は、8 バイト境界にないアドレスにアクセスすると *mem_address_not_aligned* 例外を検出する。

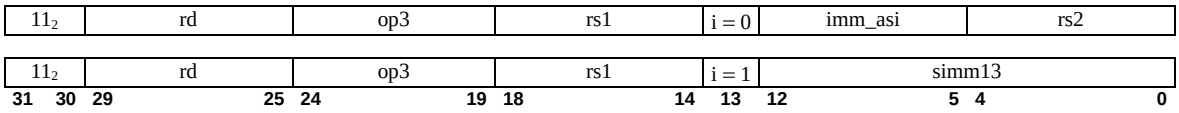
Compatibility Note SPARC V8 仕様の LD 命令は、LDUW に名前を変更した。LDSW は SPARC V9 で新規に定義された。

例外	対象命令	検出条件
illegal_instruction	すべて	*reserved* が 0 でないとき。
illegal_action	すべて	XAR.v = 1 かつ、下記のいずれかが成立している場合 • XAR.simd = 1 • XAR.urs1<2:1> ≠ 00_2 • i = 0 かつ XAR.urs2<2:1> ≠ 00_2 • i = 1 かつ XAR.urs2 ≠ 0 • XAR.urd<2:1> ≠ 00_2
mem_address_not_aligned	LDUH, LDSH, LDUW, LDSW, LDX	本文参照。
VA_watchpoint	すべて	
DAE_privilege_violation	すべて	
DAE_nfo_page	すべて	

関連項目　　LDTW（250 ページ）

7.84. Load Integer from Alternate Space

命令	op3	操作	HPC-ACE2 Regs	HPC-ACE2 SIMD	アセンブリ言語表記	
LDSBA$^{P_{ASI}}$	01 1001$_2$	別空間から符号付き 1 バイトデータの読み出し	✓		ldsba ldsba	[*address*] ***imm_asi***, *reg*$_{rd}$ [*address*] %asi, *reg*$_{rd}$
LDSHA$^{P_{ASI}}$	01 1010$_2$	別空間から符号付き 2 バイトデータの読み出し	✓		ldsha ldsha	[*address*] ***imm_asi***, *reg*$_{rd}$ [*address*] %asi, *reg*$_{rd}$
LDSWA$^{P_{ASI}}$	01 1000$_2$	別空間から符号付き 4 バイトデータの読み出し	✓		ldswa ldswa	[*address*] ***imm_asi***, *reg*$_{rd}$ [*address*] %asi, *reg*$_{rd}$
LDUBA$^{P_{ASI}}$	01 0001$_2$	別空間から符号なし 1 バイトデータの読み出し	✓		lduba lduba	[*address*] ***imm_asi***, *reg*$_{rd}$ [*address*] %asi, *reg*$_{rd}$
LDUHA$^{P_{ASI}}$	01 0010$_2$	別空間から符号なし 2 バイトデータの読み出し	✓		lduha lduha	[*address*] ***imm_asi***, *reg*$_{rd}$ [*address*] %asi, *reg*$_{rd}$
LDUWA$^{P_{ASI}}$	01 0000$_2$	別空間から符号なし 4 バイトデータの読み出し	✓		lduwa lduwa lda lda	[*address*] ***imm_asi***, *reg*$_{rd}$ [*address*] %asi, *reg*$_{rd}$ [*address*] ***imm_asi***, *reg*$_{rd}$ [*address*] %asi, *reg*$_{rd}$
LDXA$^{P_{ASI}}$	01 1011$_2$	別空間から 8 バイトデータの読み出し	✓		ldxa ldxa	[*address*] ***imm_asi***, *reg*$_{rd}$ [*address*] %asi, *reg*$_{rd}$



動作説明

別空間からの整数ロード命令は、メモリから 1,2,4, および 8 バイトのデータを読み出して R[rd] に格納する。読み出した 1,2,4, および 8 バイトデータは、命令により符号拡張またはゼロ拡張される。

これらの別空間からの整数ロード命令は、空間識別子（ASI）を必要とする。ASI は、i = 0 のときは imm_asi フィールドで指示され、i = 1 のときは ASI レジスタの値が使われる。読み出しアドレスは、i = 0 のときは “R[rs1] + R[rs2]” で、i = 1 のときは “R[rs1] + sign_ext(simm13)” で計算される。

整数ロード命令が正常に実行されるとき、メモリアクセスはアトミックに行なわれる。

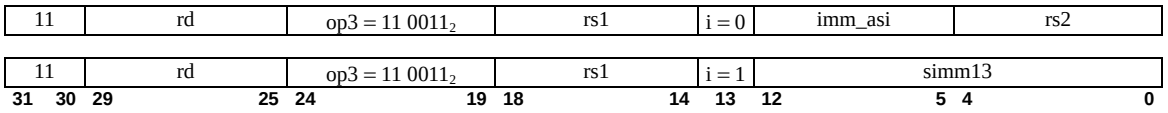
LDUHA, LDSHA 命令は、2 バイト境界にないアドレスにアクセスすると *mem_address_not_aligned* 例外を検出する。LDUWA, LDSWA 命令は、4 バイト境界にないアドレスにアクセスすると *mem_address_not_aligned* 例外を検出する。LDXA 命令は、8 バイト境界にないアドレスにアクセスすると *mem_address_not_aligned* 例外を検出する。

例外	対象命令	検出条件
illegal_action	すべて	XAR.v = 1 かつ、下記のいずれかが成立している場合 • XAR.simd = 1 • XAR.urs1<2:1> ≠ 00_2 • i = 0 かつ XAR.urs2<2:1> ≠ 00_2 • i = 1 かつ XAR.urs2 ≠ 0 • XAR.urd<2:1> ≠ 00_2
mem_address_not_aligned	LDUHA, LDSHA, LDUWA, LDSWA, LDXA	本文参照。
privileged_action	すべて	• PSTATE.priv = 0 で、00_{16} – $7F_{16}$ の ASI を使おうとしたとき。 • PSTATE.priv = 1 で、30_{16} – $7F_{16}$ の ASI を使おうとしたとき。
VA_watchpoint	すべて	
DAE_invalid_asi	すべて	
DAE_privilege_violation	すべて	
DAE_nfo_page	すべて	

関連項目　　LDTWA（252 ページ）

7.85. Block Load

命令	ASI	操作	HPC-ACE2		アセンブリ言語表記	
			Regs	SIMD		
LDBLOCKF	$F0_{16}$	プライマリアドレス空間に 64 バイトブロッククロードを実行する。	✓		ldda ldda	[*regaddr*] ASI_BLK_P, *freg*$_{rd}$ [*reg_plus_imm*] %asi, *freg*$_{rd}$
LDBLOCKF	$F1_{16}$	セカンダリアドレス空間に 64 バイトブロッククロードを実行する。	✓		ldda ldda	[*regaddr*] ASI_BLK_S, *freg*$_{rd}$ [*reg_plus_imm*] %asi, *freg*$_{rd}$
LDBLOCKF	$F8_{16}$	プライマリアドレス空間に 64 バイトブロッククロードを実行する。リトルエンディアン。	✓		ldda ldda	[*regaddr*] ASI_BLK_PL, *freg*$_{rd}$ [*reg_plus_imm*] %asi, *freg*$_{rd}$
LDBLOCKF	$F9_{16}$	セカンダリアドレス空間に 64 バイトブロッククロードを実行する。リトルエンディアン。	✓		ldda ldda	[*regaddr*] ASI_BLK_SL, *freg*$_{rd}$ [*reg_plus_imm*] %asi, *freg*$_{rd}$



動作説明

LDBLOCKF 命令は、LDDFA 命令にブロックロード用に定義された ASI を指定することで実行される。LDBLOCKF 命令でアクセスするメモリ空間は、通常のロードが行われるメモリ空間と異なり、キャッシャブル空間のみである。LDBLOCKF 命令をノンキャッシャブル空間に対して実行することはできない。

LDBLOCKF 命令は、64 バイト境界の 64 バイト領域の内容を、連続番号の倍精度浮動小数点レジスタ 8 つに読み出す。読み出すアドレスは、i = 0 のときは “R[rs1] + R[rs2]” で、i = 1 のときは “R[rs1] + sign_ext(simm13)” で計算される。命令で指定されたアドレス（最低位アドレス）の内容が、命令で指定された倍精度浮動小数点レジスタ（一番小さい番号）に読み出され、最低位アドレス+8 の内容が、次に小さい番号の倍精度浮動小数点レジスタの内容に読み出され、以後順に全 64 バイト領域の内容が 8 つの倍精度浮動小数点レジスタに読み出される。リトルエンディアンのブロックロードでは、ひとつの倍精度浮動小数点レジスタの 8 バイト内でエンディアンの変換が行われる。

SPARC64™ XIfx の LDBLOCKF 命令は、前後のロード・ストア命令間で TSO を遵守する。ブロックロードの各 8 バイトロード間でも TSO を遵守する。

SPARC64™ XIfx の LDBLOCKF 命令は、他のロード命令と同様、レジスタ依存関係にある命令のプログラム実行順序を保障する。

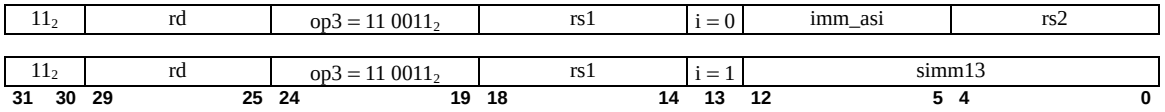
LDBLOCKF 命令のキャッシュに対する作用は、通常のロード命令と同じである。すなわち、L1D キャッシュ上にデータがあれば L1D キャッシュから読み出し、L1D キャッシュ上になければ L1D キャッシュに読み込んだ上で当該データを読み出す。

LDBLOCKF 命令は最初の 8 バイトロードに対してのみ、*VA_watchpoint* 例外を検出する。

例外	対象命令	検出条件
illegal_instruction	すべて	rd に 8 の倍数以外のレジスタが指定された場合。
fp_disabled	すべて	PSTATE.pef = 0 または FPRS.fef = 0
illegal_action	すべて	XAR.v = 1 かつ下記のいずれかが成立している場合 • XAR.simd = 1 • XAR.urs1<2:1> ≠ 00_2 • i = 0 かつ XAR.urs2<2:1> ≠ 00_2 • i = 1 かつ XAR.urs2 ≠ 0
mem_address_not_aligned	すべて	アドレスが 64 バイト境界にないとき。
VA_watchpoint	すべて	最低位アドレスの 8 バイトに対するアクセスのみ。
DAE_privilege_violation	すべて	PSTATE.priv = 0 かつ TTE.p = 1
DAE_nc_page	すべて	ノンキャッシャブル空間にアクセスしたとき。
DAE_nfo_page	すべて	

7.86. Short Floating-Point Load

命令	ASI 番号	操作	HPC-ACE2 Regs	HPC-ACE2 SIMD	アセンブリ言語表記
LDSHORTF	$D0_{16}$	プライマリ空間から8ビットデータを浮動小数点レジスタへ読み出し			ldda [*address*] ASI_FL_8_P, *freg*$_{rd}$ ldda [*address*] %asi, *freg*$_{rd}$
LDSHORTF	$D1_{16}$	セカンダリ空間から8ビットデータを浮動小数点レジスタへ読み出し			ldda [*address*] ASI_FL_8_S, *freg*$_{rd}$ ldda [*address*] %asi, *freg*$_{rd}$
LDSHORTF	$D8_{16}$	プライマリ空間から8ビットデータを浮動小数点レジスタへ読み出し、リトルエンディアン			ldda [*address*] ASI_FL_8_PL, *freg*$_{rd}$ ldda [*address*] %asi, *freg*$_{rd}$
LDSHORTF	$D9_{16}$	プライマリ空間から8ビットデータを浮動小数点レジスタへ読み出し、リトルエンディアン			ldda [*address*] ASI_FL_8_SL, *freg*$_{rd}$ ldda [*address*] %asi, *freg*$_{rd}$
LDSHORTF	$D2_{16}$	プライマリ空間から 16 ビットデータを浮動小数点レジスタへ読み出し			ldda [*address*] ASI_FL_16_P, *freg*$_{rd}$ ldda [*address*] %asi, *freg*$_{rd}$
LDSHORTF	$D3_{16}$	セカンダリ空間から 16 ビットデータを浮動小数点レジスタへ読み出し			ldda [*address*] ASI_FL_16_S, *freg*$_{rd}$ ldda [*address*] %asi, *freg*$_{rd}$
LDSHORTF	DA_{16}	プライマリ空間から 16 ビットデータを浮動小数点レジスタへ読み出し、リトルエンディアン			ldda [*address*] ASI_FL_16_PL, *freg*$_{rd}$ ldda [*address*] %asi, *freg*$_{rd}$
LDSHORTF	DB_{16}	プライマリ空間から 16 ビットデータを浮動小数点レジスタへ読み出し、リトルエンディアン			ldda [*address*] ASI_FL_16_SL, *freg*$_{rd}$ ldda [*address*] %asi, *freg*$_{rd}$



動作説明

LDSHORTF 命令は、LDDFA 命令で ASI $D0_{16}$ – $D3_{16}$, および $D8_{16}$ – DB_{16} を(ASI) を指定した場合と等価である。これ以外の ASI を使用した場合は LDSHORTF 命令ではないので、ASI 番号に応じた他の説明を参照。

8 ビットの LDSHORTF 命令は、メモリ上の 8 ビットデータを倍精度浮動小数点レジスタに読み出す。読み出されたデータは Fd[rd]の最下位 8 ビットに格納され、上位 56 ビットにはゼロがセットされる。

16 ビットの LDSHORTF 命令は、メモリ上の 16 ビットデータを倍精度浮動小数点レジスタに読み出す。読み出されたデータは Fd[rd]の最下位 16 ビットに格納され、上位 48 ビットにはゼロがセットされる。16 ビットの LDSHORTF 命令が 2 バイト境界にないアドレスにアクセスすると、*mem_address_not_aligned* 例外を検出する。

ASI は、i = 0 のときは imm_asi フィールドで指示され、i = 1 のときは ASI レジスタの値が使われる。読み出しアドレスは、i = 0 のときは “R[rs1] + R[rs2]” で、i = 1 のときは “R[rs1] + sign_ext(simm13)” で計算される。

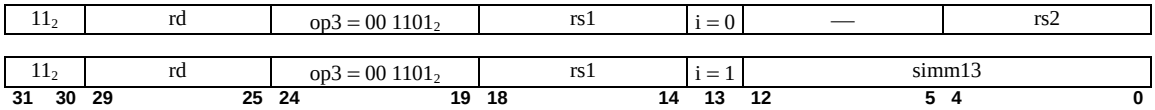
リトルエンディアン ASI は、メモリからのデータ読み出し時に、バイト単位でリトルエンディアンに置き換えてレジスタに格納する。

Programming Note　LDSHORTF 命令の典型的な使い方は、FALIGNDATA 命令とともに使用し、不連続なコンポーネントからの 64 ビットデータを構成することである。

例外	検出条件
fp_disabled	PSTATE.pef = 0 または FPRS.fef = 0
illegal_action	XAR.v = 1
mem_address_not_aligned	本文参照。
VA_watchpoint	本文参照。
DAE_privilege_violation	
DAE_nfo_page	

7.87. Load-Store Unsigned Byte

命令	op3	操作	HPC-ACE2		アセンブリ言語表記
			Regs	SIMD	
LDSTUB	$00\ 1101_2$	1バイト符号なしアトミック操作	✓		`ldstub` [*address*], *reg*$_{rd}$



動作説明

LDSTUB 命令は、指定されたアドレスの 1 バイトを読み出し、同時にその 1 バイトすべてに'1'を書き込む。読み出したデータは R[rd]の最小位バイトに格納され、上位 7 バイトにはゼロが格納される。

メモリ読み出しと書き込みは不可分に行なわれ、間にインタラプトや deferred トラップが入ることはない。マルチプロセッサシステムにおいては、複数のプロセッサが同時に、同一ダブルワードまたはその一部を指定してアトミック命令を実行した場合に、それらの命令が、順序は不定だが必ずひとつずつ実行されることが保証される。

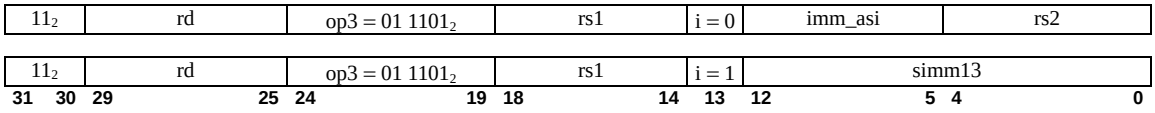
LDSTUB 命令は、暗黙の ASI (UA2011 6.3.1.3 参照) を使いメモリにアクセスする。読み出すアドレスは、i = 0 のときは "R[rs1] + R[rs2]" で、i = 1 のときは "R[rs1] + sign_ext(simm13)" で計算される。

プロセッサと I/O DMA の間でメモリ内容の同一性とアトミック性が保証されるかどうかはこの仕様書では定義しない。

例外	検出条件
illegal_instruction	*reserved* が 0 でないとき。
illegal_action	XAR.v = 1 かつ、下記のいずれかが成立している場合 • XAR.simd = 1 • XAR.urs1<2:1> ≠ 00_2 • i = 0 かつ XAR.urs2<2:1> ≠ 00_2 • i = 1 かつ XAR.urs2 ≠ 0 • XAR.urd<2:1> ≠ 00_2
VA_watchpoint	
DAE_privilege_violation	
DAE_nc_page	
DAE_nfo_page	

7.88. Load-Store Unsigned Byte to Alternate Space

命令	op3	操作	HPC-ACE2 Regs	HPC-ACE2 SIMD	アセンブリ言語表記
LDSTUBA$^{P_{ASI}}$	$01\ 1101_2$	別空間に対する 1 バイト符号なしアトミック操作	✓		`ldstuba` [*address*] ***imm_asi***, *reg*$_{rd}$ `ldstuba` [*address*] `%asi`, *reg*$_{rd}$



動作説明

LDSTUBA 命令は、指定されたアドレスの 1 バイトを読み出し、同時にその 1 バイトすべてに'1'を書き込む。読み出したデータは R[rd]の最小位バイトに格納され、上位 7 バイトにはゼロが格納される。

メモリ読み出しと書き込みは不可分に行なわれ、間にインタラプトや deferred トラップが入ることはない。マルチプロセッサシステムにおいては、複数のプロセッサが同時に、同一ダブルワードまたはその一部を指定してアトミック命令を実行した場合に、それらの命令が、順序は不定だが必ずひとつずつ実行されることが保証される。

LDSTUBA 命令は、空間識別子（ASI）を必要とする。ASI は、i = 0 のときは imm_asi フィールドで指示され、i = 1 のときは ASI レジスタの値が使われる。読み出しアドレスは、i = 0 のときは "R[rs1] + R[rs2]" で、i = 1 のときは "R[rs1] + sign_ext(simm13)" で計算される。

プロセッサと I/O DMA の間でメモリ内容の同一性とアトミック性が保証されるかどうかはこの仕様書では定義しない。

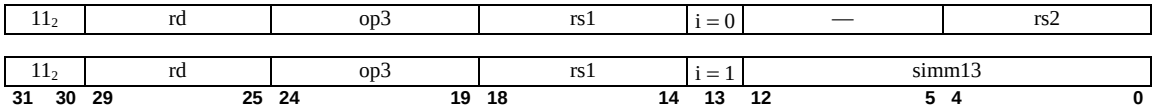
LDSTUBA 命令は、*privileged_action* 例外に記述した特権モード規則に従い、以下の ASI で使うことができる。それ以外の ASI での使用は、*DAE_invalid_asi* 例外を生じる。

LDSTUBA 命令で有効な ASI	
ASI_PRIMARY	ASI_PRIMARY_LITTLE
ASI_SECONDARY	ASI_SECONDARY_LITTLE

例外	検出条件
illegal_action	XAR.v = 1 かつ、下記のいずれかが成立している場合 • XAR.simd = 1 • XAR.urs1<2:1> ≠ 00_2 • i = 0 かつ XAR.urs2<2:1> ≠ 00_2 • i = 1 かつ XAR.urs2 ≠ 0 • XAR.urd<2:1> ≠ 00_2
privileged_action	• PSTATE.priv = 0 で、00_{16} – $7F_{16}$ の ASI を使おうとしたとき。 • PSTATE.priv = 1 で、40_{16} – $7F_{16}$ の ASI を使おうとしたとき。
VA_watchpoint	
DAE_invalid_asi	本文参照
DAE_privilege_violation	
DAE_nc_page	
DAE_nfo_page	

7.89. Load Integer Twin Word

命令	op3	操作	HPC-ACE2 Regs	HPC-ACE2 SIMD	アセンブリ言語表記
LDTW[D]	$00\,0011_2$	4 バイトデータを 2 つ読み出す		✓	ldtw[xxvi][*address*], *reg*$_{rd}$



動作説明

LDTW 命令は、8 バイト境界にある 4 バイトのデータ 2 つを読み出し、2 つの整数レジスタに格納する。命令では偶数番号のレジスタを指定し、命令で指定したアドレスにある 4 バイトが偶数番号のレジスタの下位 32 ビットに、命令で指定したアドレス+4 にある 4 バイトが、命令で指定した番号+1 の奇数番号のレジスタの下位 32 ビットに格納される。両方のレジスタとも、上位 32 ビットにはゼロが格納される。

Note LDTW 命令で rd = 0 を指定すると、アドレス+4 の 4 バイトが R[1]に格納される。

LDTW 命令は、暗黙の ASI (UA2011 6.3.1.3 参照) を使いメモリにアクセスする。読み出すアドレスは、i = 0 のときは “R[rs1] + R[rs2]” で、i = 1 のときは “R[rs1] + sign_ext(simm13)” で計算される。

LDTW 命令は、8 バイト境界にないアドレスにアクセスすると *mem_address_not_aligned* 例外を検出する。

LDTW 命令はリトルエンディアン変換に対し、2 つの 4 バイトアクセスと同じように振舞う。

LDTW 命令が正常に実行されるとき、メモリアクセスはアトミックに行なわれる。

Programming Note LDTW 命令は SPARC V8 仕様に準拠したソフトウェアの互換性のために用意されている。SPARC V9 使用に準拠したプロセッサでは、ハードウェア実装の制約から低速で実行されるかもしれない。

Compatibility Note LDTW 命令は SPARC V8 仕様、SPARC V9 仕様では LDD 命令と不正確な名前で呼ばれていた。この命令はダブルワードをロードするのではなく、2 ワードを 2 レジスタにロードするので、ふさわしい名前に変更した。

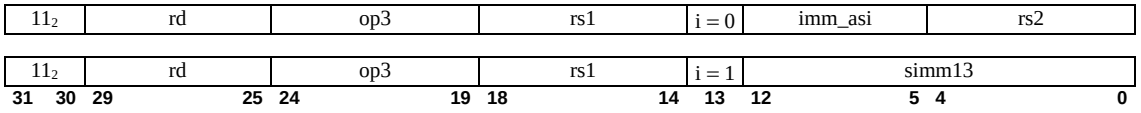
[xxvi] もともとこの命令のアセンブリ言語表記は ldd で、今では非推奨となっている。いずれアセンブラは新しい命令表記に対応すると思われるが、古い命令表記にしか対応していないアセンブラが生き残っている場合もありうる。

例外	検出条件
illegal_instruction	下記のいずれかが成立している場合 • *reserved* が 0 でないとき。 • rd が奇数のとき。
illegal_action	XAR.v = 1 かつ、下記のいずれかが成立している場合 • XAR.simd = 1 • XAR.urs1<2:1> ≠ 00_2 • i = 0 かつ XAR.urs2<2:1> ≠ 00_2 • i = 1 かつ XAR.urs2 ≠ 0 • XAR.urd<2:1> ≠ 00_2
mem_address_not_aligned	本文参照。
VA_watchpoint	
DAE_privilege_violation	
DAE_nfo_page	

関連項目　　LDX（239 ページ）
STTW（289 ページ）

7.90. Load Integer Twin Word from Alternate Space

命令	op3	操作	HPC-ACE2 Regs	HPC-ACE2 SIMD	アセンブリ言語表記
LDTWA$^{D,P_{ASI}}$	$01\ 0011_2$	別空間から 4 バイトデータを 2 つ読み出す	✓		ldtwa[xxvii] [*address*] **imm_asi**, *reg*$_{rd}$ ldtwa [*address*] %asi, *reg*$_{rd}$



動作説明

LDTWA 命令は、8 バイト境界にある 4 バイトのデータ 2 つを読み出し、2 つの整数レジスタに格納する。命令では偶数番号のレジスタを指定し、命令で指定したアドレスにある 4 バイトが偶数番号のレジスタの下位 32 ビットに、命令で指定したアドレス+4 にある 4 バイトが、命令で指定した番号+1 の奇数番号のレジスタの下位 32 ビットに格納される。両方のレジスタとも、上位 32 ビットにはゼロが格納される。

Note LDTWA 命令で rd = 0 を指定すると、アドレス+4 の 4 バイトが R[1]に格納される。

LDTWA 命令は、空間識別子（ASI）を必要とする。ASI は、i = 0 のときは imm_asi フィールドで指示され、i = 1 のときは ASI レジスタの値が使われる。読み出しアドレスは、i = 0 のときは “R[rs1] + R[rs2]” で、i = 1 のときは “R[rs1] + sign_ext(simm13)” で計算される。

LDTWA 命令は、8 バイト境界にないアドレスにアクセスすると *mem_address_not_aligned* 例外を検出する。

LDTWA 命令はリトルエンディアン変換に対し、2 つの 4 バイトアクセスと同じように振舞う。

LDTWA 命令が正常に実行されるとき、メモリアクセスはアトミックに行なわれる。

Programming Note LDTWA 命令は SPARC V8 仕様に準拠したソフトウェアの互換性のために用意されている。SPARC V9 使用に準拠したプロセッサでは、ハードウェア実装の制約から低速で実行されるかもしれない。

Compatibility Note LDTWA 命令は SPARC V8 仕様、SPARC V9 仕様では LDDA 命令と不正確な名前で呼ばれていた。この命令はダブルワードをロードするのではなく、2 ワードを 2 レジスタにロードするので、ふさわしい名前に変更した。

LDTWA 命令は、*privileged_action* 例外に記述した特権モード規則に従い、以下の ASI で使うことができる。それ以外の ASI での使用は、*DAE_invalid_asi* 例外を生じる。なお、ASI_TWINX* については 254 ページを参照。

[xxvii] もともとこの命令のアセンブリ言語表記は lddа で、今では非推奨となっている。いずれアセンブラは新しい命令表記に対応すると思われるが、古い命令表記にしか対応していないアセンブラが生き残っている場合もありうる。

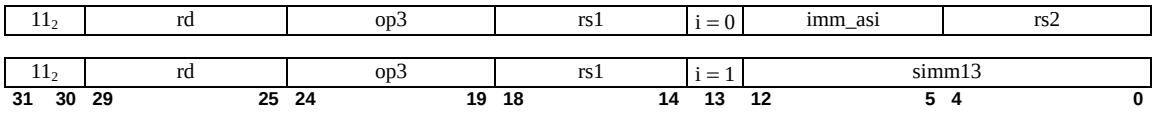
LDTWA 命令で有効な ASI	
ASI_PRIMARY	ASI_PRIMARY_LITTLE
ASI_SECONDARY	ASI_SECONDARY_LITTLE
ASI_PRIMARY_NO_FAULT	ASI_PRIMARY_NO_FAULT_LITTLE
ASI_SECONDARY_NO_FAULT	ASI_SECONDARY_NO_FAULT_LITTLE

例外	検出条件
illegal_instruction	下記のいずれかが成立している場合 • rd が奇数のとき。
illegal_action	XAR.v = 1 かつ、下記のいずれかが成立している場合 • XAR.simd = 1 • XAR.urs1<2:1> ≠ 00_2 • i = 0 かつ XAR.urs2<2:1> ≠ 00_2 • i = 1 かつ XAR.urs2 ≠ 0 • XAR.urd<2:1> ≠ 00_2
mem_address_not_aligned	本文参照。
privileged_action	• PSTATE.priv = 0 で、00_{16} – $7F_{16}$ の ASI を使おうとしたとき。 • PSTATE.priv = 1 で、30_{16} – $7F_{16}$ の ASI を使おうとしたとき。
VA_watchpoint	
DAE_invalid_asi	本文参照
DAE_privilege_violation	
DAE_nfo_page	

関連項目　LDXA（241 ページ）
STTWA（290 ページ）

7.91. Load Integer Twin Extended Word from Alternate Space

命令	ASI	操作	HPC-ACE2 Regs	HPC-ACE2 SIMD	アセンブリ言語表記
LDTXA^N	$E2_{16}$	8 バイトデータを 2 つ読み出す	✓		`ldtxa` [*regaddr*]`#ASI_TWINX_P`, *reg*$_{rd}$
	$E3_{16}$	8 バイトデータを 2 つ読み出す	✓		`ldtxa` [*regaddr*]`#ASI_TWINX_S`, *reg*$_{rd}$
	EA_{16}	8 バイトデータを 2 つ読み出す	✓		`ldtxa` [*regaddr*]`#ASI_TWINX_PL`, *reg*$_{rd}$
	EB_{16}	8 バイトデータを 2 つ読み出す	✓		`ldtxa` [*regaddr*]`#ASI_TWINX_SL`, *reg*$_{rd}$



動作説明

LDTXA 命令は、16 バイト境界にある 8 バイトデータ 2 つを読み出し、2 つの整数レジスタに格納する。命令では偶数番号のレジスタを指定し、命令で指定したアドレスにある 8 バイトが偶数番号のレジスタに、命令で指定したアドレス+8 にある 8 バイトが、命令で指定した番号+1 の奇数番号のレジスタに格納される。

Note LDTXA 命令で rd = 0 を指定すると、アドレス+8 の 8 バイトが R[1]に格納される。

LDTXA 命令は、空間識別子（ASI）を必要とする。ASI は、i = 0 のときは imm_asi フィールドで指示され、i = 1 のときは ASI レジスタの値が使われる。読み出しアドレスは、i = 0 のときは “R[rs1] + R[rs2]” で、i = 1 のときは “R[rs1] + sign_ext(simm13)” で計算される。

LDTXA 命令はリトルエンディアン変換に対し、2 つの 8 バイトアクセスと同じように振舞う。

LDTXA 命令が正常に実行されるとき、メモリアクセスはアトミックに行なわれる。

Programming Note LDTXA 命令は、TSB TTE の 1 エントリをアトミックに読み出すときに使うことができる。

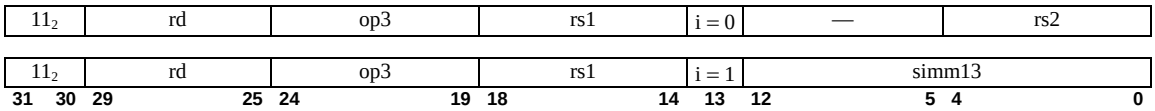
LDTXA 命令で指定できるのは ASI 番号 $E2_{16}$, $E3_{16}$, EA_{16}, EB_{16} のみである。他の ASI 番号との組み合わせは LDTWA 命令になる。VA でのアクセスになる。

Compatibility Note UA2011 では、i = 0 のみ定義されているが、SPARC64™ XIfx では、i = 1 でも動作する。

例外	対象命令	検出条件
illegal_instruction	すべて	rd に奇数番号レジスタを指定した場合
illegal_action	すべて	XAR.v = 1 かつ、下記のいずれかが成立している場合 • XAR.simd = 1 • XAR.urs1<2:1> ≠ 00_2 • i = 0 かつ XAR.urs2<2:1> ≠ 00_2 • i = 1 かつ XAR.urs2 ≠ 0 • XAR.urd<2:1> ≠ 00_2
mem_address_not_aligned	すべて	*regaddr* が 16 バイトアライン境界にないとき
VA_watchpoint	すべて	先頭 8 バイトのみ。 12.3.1.34 参照。
DAE_privilege_violation	すべて	PSTATE.priv = 0 で TTE.p = 1 のページにアクセスしたとき
DAE_nc_page	すべて	
DAE_nfo	すべて	

7.92. Load Floating-Point State Register

命令	op3	rd	操作	HPC-ACE2 Regs	HPC-ACE2 SIMD	アセンブリ言語表記	
LDFSR[D]	$10\ 0001_2$	0	メモリから FSR に読み出し（下位 32 ビットのみ）	✓		ld	[*address*], %fsr
LDXFSR	$10\ 0001_2$	1	メモリから FSR に読み出し	✓		ldx	[*address*], %fsr
—	$10\ 0001_2$	2 – 31	*reserved*				



動作説明

LDFSR 命令は、未完了のすべての浮動小数点演算の完了を待ってから、4 バイト境界の 4 バイト領域を読み出し FSR の下位 32 ビットに格納する。上位 32 ビットは変更されない。また、ver, ftt, qne, および *reserved* フィールドは更新されない。

LDXFSR 命令は、未完了のすべての浮動小数点演算の完了を待ってから、8 バイト境界の 8 バイト領域を読み出し FSR の全 64 ビットに格納する。ver, ftt, qne, および *reserved* フィールドは更新されない。

> **Programming Note** 将来の互換性のために、ソフトウェアは *reserved* フィールドには 0 をロードすべきである。

LDFSR および LDXFSR 命令は、暗黙の ASI（UA2011 6.3.1.3 参照）を使いメモリにアクセスする。書き込みアドレスは、i = 0 のときは “R[rs1] + R[rs2]” で、i = 1 のときは “R[rs1] + sign_ext(simm13)” で計算される。

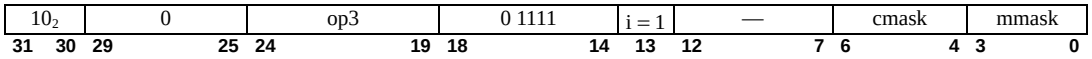
LDFSR 命令は、4 バイト境界にないアドレスにアクセスすると *mem_address_not_aligned* 例外を発生する。LDXFSR 命令は、8 バイト境界にないアドレスにアクセスすると *mem_address_not_aligned* 例外を発生する。

LDFSR および LDXFSR 命令の実行により例外が発生しトラップが起きた場合、FSR は更新されない。

例外	対象命令	検出条件
illegal_instruction	LDFSR, LDXFSR	i = 0 かつ *reserved* が 0 でないとき。
	—	rd = 2–31
fp_disabled	すべて	PSTATE.pef = 0 または FPRS.fef = 0
illegal_action	すべて	XAR.v = 1 かつ下記のいずれかが成立している場合 • XAR.simd = 1 • XAR.urs1<2:1> ≠ 00_2 • i = 0 かつ XAR.urs2<2:1> ≠ 00_2 • i = 1 かつ XAR.urs2 ≠ 0 • XAR.urd ≠ 0
mem_address_not_aligned	LDFSR	4 バイト境界ではないアドレスにアクセスしたとき。
	LDXFSR	8 バイト境界ではないアドレスにアクセスしたとき。
VA_watchpoint	すべて	
DAE_privilege_violation	すべて	
DAE_nfo_page	すべて	

7.93. Memory Barrier

命令	op3	操作	HPC-ACE2 Regs	HPC-ACE2 SIMD	アセンブリ言語表記
MEMBAR	$10\ 1000_2$	メモリバリア			membar *membar_mask*



動作説明

メモリバリア命令 MEMBAR は、メモリ参照の順序制御と完了の明示的な制御という 2 つの相異なる機能を持つ。アセンブリ言語の文法における membar_mask フィールドで、命令フィールドの cmask と mmask を指示する。

mmask は命令のビット 3 から 0 にあるフィールドである。表 7-24 で mmask の各ビットについて説明する。これはメモリ参照に関して MEMBAR を入れることで付加されるメモリ順序制御を意味する。

表 7-24 mmask ビットで指示できる順序制御

マスクビット	名前	説明
mmask<3>	#StoreStore	MEMBAR の前に現れるストアの結果が、すべてのプロセッサにおいて、MEMBAR の後に現れるストアの結果より前に観測可能であることを保証する。非推奨の STBAR 命令と同じ役割を果たす。 SPARC64™ XIfx ではすべてのストア命令がプログラム順序で実行されるので、このビットは意味を持たない。
mmask<2>	#LoadStore	MEMBAR の前に現れるロード命令が、MEMBAR の後に現れるストアがいずれかのプロセッサで観測可能になる前に、実行されることを保証する。 SPARC64™ XIfx ではすべてのストア命令がプログラム順序で実行され、ロード命令との順序は保証されているので、このビットは意味を持たない。
mmask<1>	#StoreLoad	MEMBAR の前に現れるストア命令が、すべてのプロセッサにおいて、MEMBAR の後に現れるロード命令の実行前に観測可能であることを保証する。
mmask<0>	#LoadLoad	MEMBAR の前に現れるすべてのロードの実行が、MEMBAR の後に現れるロードの実行より前に完了していることを保証する。 SPARC64™ XIfx ではすべてのロード命令がプログラム順序で実行されるので、このビットは意味を持たない。

cmask は命令のビット 6 から 4 にあるフィールドである。表 7-25 で cmask の各ビットについて説明する。これはメモリ参照と命令の実行に関する制限を付加するためのものである。もし cmask がゼロなら、MEMBAR は mmask フィールドで指定される部分的な順序制御を指示する。もし cmask がゼロでなければ、部分的な順序制御と完了制御が適用される。

表 7-25　cmask ビットで指示できる順序制御

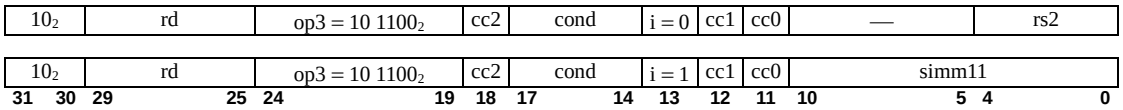
マスクビット	機能	名前	説明
cmask<2>	同期バリア	#Sync	MEMBAR より前に実行されるべきすべての命令（メモリアクセス以外の命令も）が実行され、例外があれば、MEMBAR の後の命令の実行が開始される前に通知される。
cmask<1>	メモリ参照バリア	#MemIssue	MEMBAR の前に現れるすべてのメモリ参照命令が、MEMBAR の後に現れるどのメモリ参照命令の実行開始よりも前に実行されることを保証する。SPARC64™ XIfx では#Sync と同じ。
cmask<0>	ルックアサイドバリア	#Lookaside	MEMBAR の前のストアが MEMBAR の後の同一アドレスを参照するロードより前に完了していることを保証する。SPARC64™ XIfx では#Sync と同じ。

例外	検出条件
illegal_instruction	*reserved* フィールドが 0 でないとき。
illegal_action	XAR.v = 1

7.94. Move Integer Register on Condition (MOVcc)

命令	cond	操作	テスト	HPC-ACE2		アセンブリ言語表記	
				Regs.	SIMD		
MOVA	1000_2	常に移動する	1	✓		mova	*i_or_x_cc, reg_or_imm11, reg*$_{rd}$
MOVN	0000_2	決して移動しない	0	✓		movn	*i_or_x_cc, reg_or_imm11, reg*$_{rd}$
MOVNE	1001_2	等しくないとき	**not** Z	✓		movne movnz	*i_or_x_cc, reg_or_imm11, reg*$_{rd}$ *i_or_x_cc, reg_or_imm11, reg*$_{rd}$
MOVE	0001_2	等しいとき	Z	✓		move movz	*i_or_x_cc, reg_or_imm11, reg*$_{rd}$ *i_or_x_cc, reg_or_imm11, reg*$_{rd}$
MOVG	1010_2	より大きいとき	**not** (Z **or** (N **xor** V))	✓		movg	*i_or_x_cc, reg_or_imm11, reg*$_{rd}$
MOVLE	0010_2	以下のとき	Z **or** (N **xor** V)	✓		movle	*i_or_x_cc, reg_or_imm11, reg*$_{rd}$
MOVGE	1011_2	以上のとき	**not** (N **xor** V)	✓		movge	*i_or_x_cc, reg_or_imm11, reg*$_{rd}$
MOVL	0011_2	より小さいとき	N **xor** V	✓		movl	*i_or_x_cc, reg_or_imm11, reg*$_{rd}$
MOVGU	1100_2	符号なし整数で、より大きいとき	**not** (C **or** Z)	✓		movgu	*i_or_x_cc, reg_or_imm11, reg*$_{rd}$
MOVLEU	0100_2	符号なし整数で、以下のとき	C **or** Z	✓		movleu	*i_or_x_cc, reg_or_imm11, reg*$_{rd}$
MOVCC	1101_2	キャリークリア（符号なし整数で以上）のとき	**not** C	✓		movcc movgeu	*i_or_x_cc, reg_or_imm11, reg*$_{rd}$ *i_or_x_cc, reg_or_imm11, reg*$_{rd}$
MOVCS	0101_2	キャリーセット（符号なし整数でより小さい）のとき	C	✓		movcs movlu	*i_or_x_cc, reg_or_imm11, reg*$_{rd}$ *i_or_x_cc, reg_or_imm11, reg*$_{rd}$
MOVPOS	1110_2	正のとき	**not** N	✓		movpos	*i_or_x_cc, reg_or_imm11, reg*$_{rd}$
MOVNEG	0110_2	負のとき	N	✓		movneg	*i_or_x_cc, reg_or_imm11, reg*$_{rd}$
MOVVC	1111_2	オーバーフローしていないとき	**not** V	✓		movvc	*i_or_x_cc, reg_or_imm11, reg*$_{rd}$
MOVVS	0111_2	オーバーフローのとき	V	✓		movvs	*i_or_x_cc, reg_or_imm11, reg*$_{rd}$

命令	cond	操作	fcc の値	HPC-ACE2		アセンブリ言語表記	
				Regs.	SIMD		
MOVFA	1000_2	常に移動する	0, 1, 2, 3	✓		mova	%fcc*n*, *reg_or_imm11*, *reg*$_{rd}$
MOVFN	0000_2	決して移動しない	—	✓		movn	%fcc*n*, *reg_or_imm11*, *reg*$_{rd}$
MOVFU	0111_2	比較不能のとき	3	✓		movu	%fcc*n*, *reg_or_imm11*, *reg*$_{rd}$
MOVFG	0110_2	より大きいとき	2	✓		movg	%fcc*n*, *reg_or_imm11*, *reg*$_{rd}$
MOVFUG	0101_2	比較不能またはより大きいとき	2, 3	✓		movug	%fcc*n*, *reg_or_imm11*, *reg*$_{rd}$
MOVFL	0100_2	より小さいとき	1	✓		movl	%fcc*n*, *reg_or_imm11*, *reg*$_{rd}$
MOVFUL	0011_2	比較不能またはより小さいとき	1, 3	✓		movul	%fcc*n*, *reg_or_imm11*, *reg*$_{rd}$
MOVFLG	0010_2	より小さいまたはより大きいとき	1, 2	✓		movlg	%fcc*n*, *reg_or_imm11*, *reg*$_{rd}$
MOVFNE	0001_2	等しくないとき	1, 2, 3	✓		movne movnz	%fcc*n*, *reg_or_imm11*, *reg*$_{rd}$ %fcc*n*, *reg_or_imm11*, *reg*$_{rd}$
MOVFE	1001_2	等しいとき	0	✓		move movz	%fcc*n*, *reg_or_imm11*, *reg*$_{rd}$ %fcc*n*, *reg_or_imm11*, *reg*$_{rd}$
MOVFUE	1010_2	比較不能または等しいとき	0, 3	✓		movue	%fcc*n*, *reg_or_imm11*, *reg*$_{rd}$
MOVFGE	1011_2	以上のとき	0, 2	✓		movge	%fcc*n*, *reg_or_imm11*, *reg*$_{rd}$
MOVFUGE	1100_2	比較不能または以上のとき	0, 2, 3	✓		movuge	%fcc*n*, *reg_or_imm11*, *reg*$_{rd}$
MOVFLE	1101_2	以下のとき	0, 1	✓		movle	%fcc*n*, *reg_or_imm11*, *reg*$_{rd}$
MOVFULE	1110_2	比較不能または以下のとき	0, 1, 3	✓		movule	%fcc*n*, *reg_or_imm11*, *reg*$_{rd}$
MOVFO	1111_2	比較可能のとき	0, 1, 2	✓		movo	%fcc*n*, *reg_or_imm11*, *reg*$_{rd}$



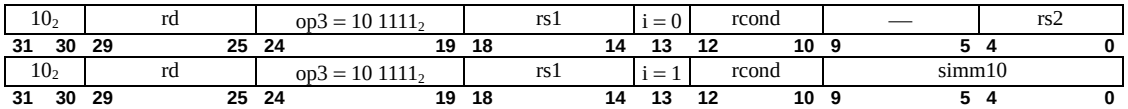
cc2	cc1	cc0	条件コード
0	0	0	fcc0
0	0	1	fcc1
0	1	0	fcc2
0	1	1	fcc3
1	0	0	icc
1	1	0	xcc

動作説明　　これらの命令は、cc0, cc1, cc2 で指定された条件フラグが cond で指定された条件を満たしているとき、i = 0 の場合は R[rs2]の内容を、i = 1 の場合は sign_ext(simm11)を R[rd]に書き込む。CCR は更新されない。条件が成立しない場合、R[rd]の内容は変更されない。

例外	対象命令	検出条件
illegal_instruction	すべて	*reserved* フィールドが 0 でない
	—	下記のいずれかが成立している場合 • cc2::cc1::cc0 = 101_2 • cc2::cc1::cc0 = 111_2
fp_disabled	MOVF*	PSTATE.pef = 0 または FPRS.fef = 0
illegal_action	すべて	XAR.v = 1 かつ以下のいずれかが成立している場合 • XAR.simd = 1 • XAR.urs1 ≠ 0 • i = 0 かつ XAR.urs2<2:1> ≠ 00_2 • i = 1 かつ XAR.urs2 ≠ 0 • XAR.urs3 ≠ 0 • XAR.urd<2:1> ≠ 00_2

7.95. Move Integer Register on Register Condition (MOVr)

命令	rcond	操作	HPC-ACE2 Regs	HPC-ACE2 SIMD	アセンブリ言語表記	
—	000_2	*reserved*			—	
MOVRZ	001_2	R[rs1] = 0 のとき移動		✓	movrz	reg_{rs1}, *reg_or_imm10*, reg_{rd}
					movre	reg_{rs1}, *reg_or_imm10*, reg_{rd}
MOVRLEZ	010_2	R[rs1] ≤ 0 のとき移動		✓	movrlez	reg_{rs1}, *reg_or_imm10*, reg_{rd}
MOVRLZ	011_2	R[rs1] < 0 のとき移動		✓	movrlz	reg_{rs1}, *reg_or_imm10*, reg_{rd}
—	100_2	*reserved*			—	
MOVRNZ	101_2	R[rs1] ≠ 0 のとき移動		✓	movrnz	reg_{rs1}, *reg_or_imm10*, reg_{rd}
					movrne	reg_{rs1}, *reg_or_imm10*, reg_{rd}
MOVRGZ	110_2	R[rs1] > 0 のとき移動		✓	movrgz	reg_{rs1}, *reg_or_imm10*, reg_{rd}
MOVRGEZ	111_2	R[rs1] ≥ 0 のとき移動		✓	movrgez	reg_{rs1}, *reg_or_imm10*, reg_{rd}



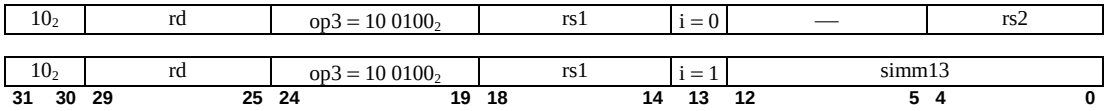
動作説明

これらの命令は、整数レジスタ R[rs1]の値が rcond で指定される条件を満たしているとき、i = 0 の場合は R[rs2]の内容を、i = 1 の場合は sign_ext(simm10)を R[rd]に移動する。R[rs1]の内容は符号付き整数として扱われる。CCR は更新されない。条件を満たしていないときは、R[rd]の内容は変更されない。

例外	対象命令	検出条件
illegal_instruction	すべて	*reserved* フィールドが 0 でないとき。
	—	下記のいずれかが成立している場合 • rcond = 000_2 • rcond = 100_2
illegal_action	すべて	XAR.v = 1 かつ下記のいずれかが成立している場合 • XAR.urs1<2:1> ≠ 00_2 • i = 0 かつ XAR.urs2<2:1> ≠ 00_2 • i = 1 かつ XAR.urs2 ≠ 0 • XAR.urs3 ≠ 0 • XAR.urd<2:1> ≠ 00_2 • XAR.simd = 1

7.96. Multiply Step

命令	op3	操作	HPC-ACE2 Regs	HPC-ACE2 SIMD	アセンブリ言語表記	
MULScc[D]	$10\ 0100_2$	ステップ乗算と cc 更新	✓		mulscc	*reg*$_{rs1}$, *reg_or_imm*, *reg*$_{rd}$



動作説明

MULScc 命令は乗算の補助命令である。MULScc 命令は R[rs1]の下位 32 ビットと Y レジスタの下位 32 ビットを、R[rs1]の最下位ビットが Y の最上位ビットと接しているかのようにつなげ、右シフト可能な 64 ビットレジスタとして扱い、Y の最下位ビットに基づき加算を行なう。

乗算を開始するに当たり、Y レジスタには乗数を、R[rs1]には結果の上位ビットを、R[rs2]には被乗数を入れておく。乗算が完了すると結果が Y レジスタに格納される。

Note MULScc 命令は通常、rs = rd で使われる。

1. 被乗数は、i = 0 なら R[rs2], i = 1 なら sign_ext(simm13)である。
2. R[rs1]を 1 ビット右シフトし、ビット 31 に “CCR.icc.n **xor** CCR.icc.v” を入れる（これは一つ前のステップ乗算の適切な符号である)。
3. Y レジスタの最下位ビットが 1 のとき、2 で得られた値と被乗数が加算される。Y の最下位ビットが 0 のとき、2 で得られた値に 0 が加算される。
4. レジスタに以下の値が設定される。

レジスタフィールド	MULScc がセットする値
CCR.icc	上記 3 の結果により設定される
R[rd]<63:33>	0
R[rd]<32>	CCR.icc.c
R[rd]<31:0>	上記 3 の結果の下位 32 ビット
CCR.xcc.n	0
CCR.xcc.v	0
CCR.xcc.c	0
CCR.xcc.z	R[rd] = 0 なら 1, それ以外なら 0

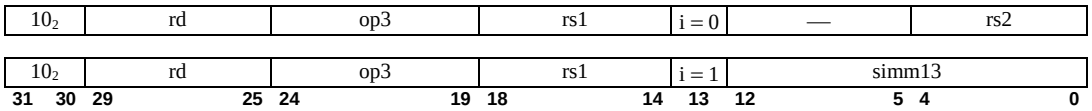
Compatibility Note SPARC V9, および JPS1 では、R[rd]の上位 32 ビットと CCR.xcc は不定とされていた。

5. Y レジスタが右に 1 ビットシフトされ、R[rs1]の最下位ビットが Y<31>に入れられる。

例外	検出条件
illegal_instruction	*reserved* フィールドが 0 でないとき。
illegal_action	XAR.v = 1 かつ下記のいずれかが成立している場合 • XAR.urs1<2:1> ≠ 00_2 • i = 0 かつ XAR.urs2<2:1> ≠ 00_2 • i = 1 かつ XAR.urs2 ≠ 0 • XAR.urs3 ≠ 0 • XAR.urd<2:1> ≠ 00_2 • XAR.simd = 1

7.97. Multiply and Divide (64-bit)

命令	op3	操作	HPC-ACE2		アセンブリ言語表記	
			Regs	SIMD		
MULX	$00\ 1001_2$	符号つきおよび符号なし整数乗算	✓		mulx	*reg*$_{rs1}$, *reg_or_imm*, *reg*$_{rd}$
SDIVX	$10\ 1101_2$	符号つき整数除算	✓		sdivx	*reg*$_{rs1}$, *reg_or_imm*, *reg*$_{rd}$
UDIVX	$00\ 1101_2$	符号なし整数除算	✓		udivx	*reg*$_{rs1}$, *reg_or_imm*, *reg*$_{rd}$



動作説明

MULX 命令は、i = 0 のとき “R[rs1] × R[rs2]” を、i = 1 のとき “R[rs1] × sign_ext(simm13)” を計算し、結果を R[rd]に格納する。MULX 命令は符号付き、符号なしのどちらの 64 ビット計算も行なえる（結果は同じ）。

SDIVX, UDIVX 命令は、i = 0 のとき “R[rs1] ÷ R[rs2]” を、i = 1 のとき “R[rs1] ÷ sign_ext(simm13)” を計算し、結果を R[rd]に格納する。SDIVX 命令は引数を符号付き整数として扱い、符号付き整数の結果を出力する。UDIVX 命令は引数を符号なし整数として扱い、符号なし整数の結果を出力する。

SDIVX 命令では、負の最大値を-1 で割ると、結果は以下のように負の最大値となる。

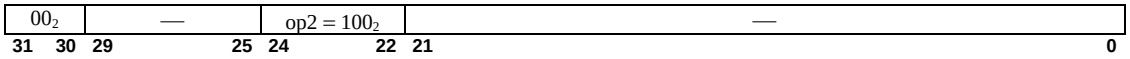
$$8000\ 0000\ 0000\ 0000_{16} \div FFFF\ FFFF\ FFFF\ FFFF_{16} = 8000\ 0000\ 0000\ 0000_{16}$$

これらの命令は CCR を更新しない。

例外	対象命令	検出条件
illegal_instruction	すべて	*reserved* フィールドが 0 でないとき。
illegal_action	すべて	XAR.v = 1 かつ下記のいずれかが成立している場合 • XAR.urs1<2:1> ≠ 00_2 • i = 0 かつ XAR.urs2<2:1> ≠ 00_2 • i = 1 かつ XAR.urs2 ≠ 0 • XAR.urs3 ≠ 0 • XAR.urd<2:1> ≠ 00_2 • XAR.simd = 1
division_by_zero	SDIVX, UDIVX	除数が 0 のとき。

7.98. No Operation

命令	op2	操作	HPC-ACE2 Regs.	HPC-ACE2 SIMD	アセンブリ言語表記
NOP	100_2	何もしない	✓		nop



動作説明

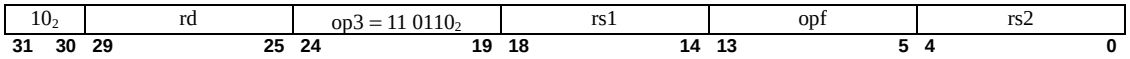
NOP 命令は、SETHI 命令で imm22 = 0 かつ rd = 0 を指定した場合と等価である。

NOP 命令は PC および nPC 以外のソフトウェアから見える状態を変化させない。

例外	対象命令	検出条件
illegal_instruction	すべて	*reserved* フィールドが 0 でない
illegal_action	すべて	XAR.v = 1 かつ下記のいずれかが成立している場合 • XAR.urs1 ≠ 0 • XAR.urs2 ≠ 0 • XAR.urs3 ≠ 0 • XAR.urd<2:1> ≠ 00_2 • XAR.simd = 1

7.99. Partitioned Add

命令	opf	操作	HPC-ACE2		アセンブリ言語表記	
			Regs.	SIMD		
PADD32	$0\ 1000\ 1001_2$	2 つの 32 ビット加算		✓	padd32	reg_{rs1}, reg_{rs2}, reg_{rd}



動作説明

PADD32 命令は、R[rs1]の 2 つの符号付き 32 ビット整数と R[rs2]の 2 つの符号付き 32 ビット整数をそれぞれ加算し、結果を R[rd]に格納する。各 32 ビット加算のキャリーは捨てられる。CCR は更新しない。

例外	検出条件
illegal_action	XAR.v = 1 かつ下記のいずれかが成立している場合 • XAR.urs1<2:1> ≠ 00_2 • XAR.urs2<2:1> ≠ 00_2 • XAR.urs3 ≠ 0 • XAR.urd<2:1> ≠ 00_2 • XAR.simd = 1

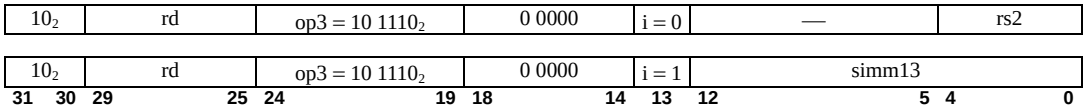
7.100. Pixel Component Distance (with Accumulation)

PDIST は UA2011 7.101 参照

例外	検出条件
fp_disabled	PSTATE.pef = 0 または FPRS.fef = 0
illegal_action	XAR.v = 1

7.101. Population Count

命令	op3	操作	HPC-ACE2 Regs	HPC-ACE2 SIMD	アセンブリ言語表記	
POPC	$10\ 1110_2$	1 がセットされているビットを数える		✓	popc	*reg_or_imm*, reg_{rd}



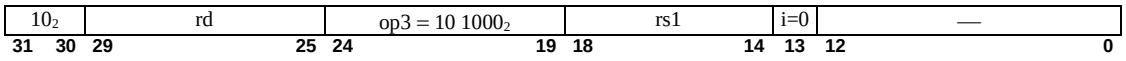
動作説明　POPC 命令は、i = 0 のときは R[rs2]の、i = 1 のときは sign_ext(simm13)の、1 がセットされているビットを数え、結果を R[rd]に格納する。CCR は更新しない。

Compatibility Note　命令語のビット 18 から 14 は 0 でなくてはならない。このフィールドが 0 でない命令は、将来の SPARC アーキテクチャで他の命令が定義されるかもしれない。

例外	検出条件
illegal_instruction	*reserved* フィールドおよび iw<18:14>が 0 でない
illegal_action	XAR.v = 1 かつ下記のいずれかが成立している場合 • XAR.urs1 ≠ 0 • i = 0 かつ XAR.urs2<2:1> ≠ 00_2 • i = 1 かつ XAR.urs2 ≠ 0 • XAR.urs3 ≠ 0 • XAR.urd<2:1> ≠ 00_2 • XAR.simd = 1

7.102. Read Ancillary State Register (RDASR)

命令	rs1	操作	HPC-ACE2		アセンブリ言語表記	
			Regs	SIMD		
RDY[D]	0	Y レジスタの読み出し。(使用を推奨しない)	✓		rd	%y, *reg*$_{rd}$
RDCCR	2	CCR レジスタの読み出し。	✓		rd	%ccr, *reg*$_{rd}$
RDASI	3	ASI レジスタの読み出し。	✓		rd	%asi, *reg*$_{rd}$
RDTICK[PNPT]	4	TICK レジスタの読み出し。	✓		rd	%tick, *reg*$_{rd}$
RDPC	5	PC レジスタの読み出し。	✓		rd	%pc, *reg*$_{rd}$
RDFPRS	6	FPRS レジスタの読み出し。	✓		rd	%fprs, *reg*$_{rd}$
MEMBAR	15	MEMBAR (258 ページ)参照。				
RDPCR[PPCR]	16	PCR レジスタの読み出し。	✓		rd	%pcr, *reg*$_{rd}$
RDPIC[PPCR]	17	PIC レジスタの読み出し。	✓		rd	%pic, *reg*$_{rd}$
RDGSR	19	GSR レジスタの読み出し。	✓		rd	%gsr, *reg*$_{rd}$
RDSTICK[PNPT]	24	STICK レジスタの読み出し。	✓		rd	%stick, *reg*$_{rd}$
RDXASR	30	XASR レジスタの読み出し。	✓		rd	%xasr, *reg*$_{rd}$



RDASR 命令は、Ancillary State レジスタの内容を汎用整数レジスタに転送する命令である。Ancillary State レジスタの詳細は 5.5 "Ancillary State Registers" (31 ページ)を参照。なお、rs1 = 15 には MEMBAR が割り当てられているが、MEMBAR に関しては 258 ページに記述することとし、この節では触れない。

- RDY は、Y レジスタの内容を読み出す命令である。Y レジスタを使う命令はすべて使用が推奨されない命令(deprecated)となっている。
- RDPC は、命令を実行する時点の PSTATE.am の設定により読み出される内容が異なる。
  - PSTATE.am = 0 のときは、PC の全 64 ビットが R[rd]に転送される。
  - PSTATE.am = 1 のときは、PC<31:0>が R[rd]<31:0>に転送され、R[rd]<63:32>には 0 が設定される。
- RDFPRS は、先行するすべての命令の実行が完了した後で、FPRS の内容が転送される。
- RDPCR, RDPIC を非特権モードで実行した場合、PCR.priv = 1 だと *privileged_action* 例外が発生する。

rd = 15 に関する例外は MEMBAR 参照。

例外	対象命令	検出条件
illegal_instruction	—	下記のいずれかが成立している場合 • rs1 = 1, 7 – 14, 18, 20 – 21, 26 – 29 • i = 1 • iw<12:0> ≠ $0\ 0000\ 0000\ 0000_2$
fp_disabled	RDGSR	PSTATE.pef = 0 または FPRS.fef = 0
Illegal_action	MEMBAR	XAR.v = 1
	MEMBAR 以外	XAR.v = 1 かつ、下記のいずれかが成立している場合 • XAR.urs1 ≠ 0 • XAR.urs2 ≠ 0 • XAR.urs3 ≠ 0 • XAR.urd<2:1> ≠ 00_2 • XAR.simd = 1
privileged_action	RDTICK	PSTATE.priv = 0 かつ TICK.npt = 0
	RDPCR, RDPIC	PSTATE.priv = 0 かつ PCR.priv = 1
	RDSTICK	PSTATE.priv = 0 かつ STICK.npt = 1

7.103. Return

RETURN は UA2011 7.110 参照

例外	検出条件
illegal_instruction	i = 0 のとき、iw<29:25> ≠ $0\ 0000_2$ または iw<12:5> ≠ $0000\ 0000_2$ i = 1 のとき、iw<29:25> ≠ $0\ 0000_2$
illegal_action	XAR.v = 1
fill_n_normal	
fill_n_other	
mem_address_not_aligned	アドレスの下位 2 ビットが 0 でないとき
control_transfer_instruction	PSTATE.tct = 1

7.104. SAVE and RESTORE

SAVE は UA2011 7.111 参照

RESTORE は UA2011 7.107 参照

<SAVE>

例外	検出条件
illegal_instruction	i = 0 かつ iw<12:5> ≠ 0000 0000_2
illegal_action	i = 0 かつ XAR.v = 1 かつ以下のいずれかの条件を満たしたとき • XAR.simd = 1 • XAR.urs1<2:1> ≠ 00_2 • XAR.urs2<2:1> ≠ 00_2 • XAR.urs3 ≠ 0 • XAR.urd<2:1> ≠ 00_2 i = 1 かつ XAR.v = 1 かつ以下のいずれかの条件を満たしたとき • XAR.simd = 1 • XAR.urs1<2:1> ≠ 00_2 • XAR.urs2 ≠ 0 • XAR.urs3 ≠ 0 • XAR.urd<2:1> ≠ 00_2
spill_n_normal	
spill_n_other	
clean_window	

<RESTORE>

例外	検出条件
illegal_instruction	i = 0 かつ iw<12:5> ≠ 0000 0000_2
illegal_action	i = 0 かつ XAR.v = 1 かつ以下のいずれかの条件を満たしたとき • XAR.simd = 1 • XAR.urs1<2:1> ≠ 00_2 • XAR.urs2<2:1> ≠ 00_2 • XAR.urs3 ≠ 0 • XAR.urd<2:1> ≠ 00_2 i = 1 かつ XAR.v = 1 かつ以下のいずれかの条件を満たしたとき • XAR.simd = 1 • XAR.urs1<2:1> ≠ 00_2 • XAR.urs2 ≠ 0 • XAR.urs3 ≠ 0 • XAR.urd<2:1> ≠ 00_2
fill_n_normal	
fill_n_other	

7.105. Signed Divide (64-bit ÷ 32-bit)

SDIV^{D}, SDIVcc^{D} は UA2011 7.113 参照

例外	検出条件
illegal_instruction	i = 0 かつ iw<12:5> ≠ 0000 0000_2
illegal_action	i = 0 かつ XAR.v = 1 かつ以下のいずれかの条件を満たしたとき • XAR.simd = 1 • XAR.urs1<2:1> ≠ 00_2 • XAR.urs2<2:1> ≠ 00_2 • XAR.urs3 ≠ 0 • XAR.urd<2:1> ≠ 00_2 i = 1 かつ XAR.v = 1 かつ以下のいずれかの条件を満たしたとき • XAR.simd = 1 • XAR.urs1<2:1> ≠ 00_2 • XAR.urs2 ≠ 0 • XAR.urs3 ≠ 0 • XAR.urd<2:1> ≠ 00_2
division_by_zero	除数が 0 のとき

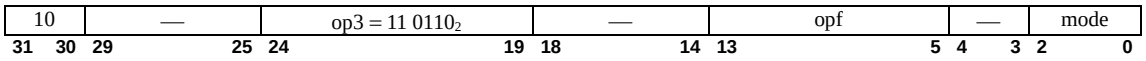
7.106. SETHI

SETHI は UA2011 7.114 参照

例外	検出条件
illegal_action	XAR.v = 1 かつ以下のいずれかの条件を満たしたとき • XAR.simd = 1 • XAR.urs1 ≠ 0 • XAR.urs2 ≠ 0 • XAR.urs3 ≠ 0 • XAR.urd<2:1> ≠ 00_2

7.107. Set Interval Arithmetic Mode

命令	opf	操作	HPC-ACE2 Regs	HPC-ACE2 SIMD	アセンブリ言語表記	
SIAM	$0\ 1000\ 0001_2$	2 進浮動小数点演算のインターバルモードの設定をする			siam	*siam_mode*



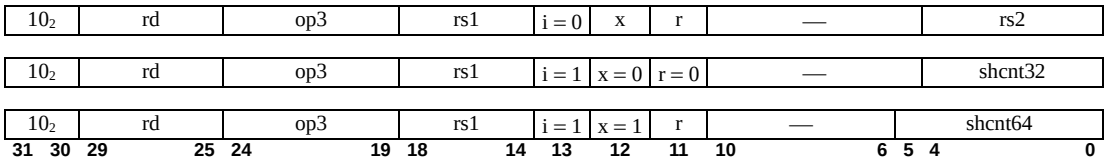
SIAM 命令は浮動小数点演算のインターバルモードの設定をする。

GSR.im ← mode<2>
GSR.irnd ← mode<1:0>

例外	検出条件
illegal_instruction	*reserved* フィールドが 0 でない。
fp_disabled	PSTATE.pef = 0 または FPRS.fef = 0
illegal_action	XAR.v = 1

7.108. Shift

命令	op3	X	r	操作	HPC-ACE2 Regs	HPC-ACE2 SIMD	アセンブリ言語表記	
SLL	$10\,0101_2$	0	0	左論理シフト − 32 ビット	✓		sll	*reg*$_{rs1}$, *reg_or_shcnt*, *reg*$_{rd}$
SRL	$10\,0110_2$	0	0	右論理シフト − 32 ビット	✓		srl	*reg*$_{rs1}$, *reg_or_shcnt*, *reg*$_{rd}$
SRA	$10\,0111_2$	0	0	右算術シフト − 32 ビット	✓		sra	*reg*$_{rs1}$, *reg_or_shcnt*, *reg*$_{rd}$
SLLX	$10\,0101_2$	1	0	左論理シフト − 64 ビット	✓		sllx	*reg*$_{rs1}$, *reg_or_shcnt*, *reg*$_{rd}$
SRLX	$10\,0110_2$	1	0	右論理シフト − 64 ビット	✓		srlx	*reg*$_{rs1}$, *reg_or_shcnt*, *reg*$_{rd}$
SRAX	$10\,0111_2$	1	0	右算術シフト − 64 ビット	✓		srax	*reg*$_{rs1}$, *reg_or_shcnt*, *reg*$_{rd}$
ROLX	$10\,0101_2$	1	1	左ローテート − 64 ビット	✓		rolx	*reg*$_{rs1}$, *reg_or_shcnt*, *reg*$_{rd}$



動作説明

これらの命令は、R[rs1]のデータを右または左にシフトし、結果を R[rd]に格納する。

シフト量は、i = 0 のときは R[rs2]で指定する。x = 0 なら R[rs2]の下位 5 ビットが、x = 1 なら R[rs2]の下位 6 ビットがシフト量として使われる。

i = 1 のときは命令語でシフト量を指定する。x = 0 なら命令語の下位 5 ビットが、x = 1 なら命令語の下位 6 ビットがシフト量として使われる。

SLL 命令と SLLX 命令は、R[rs1]の全 64 ビットを左（上位方向）にシフトし、結果を R[rd]に格納する。シフトにより空いた右側（下位側）のビットには 0 が入る。シフト量が 32 ビット以下のとき、SLL 命令と SLLX 命令は等価である。

SRL 命令は、R[rs1]の下位 32 ビットを右（下位方向）にシフトし、結果を R[rd]に格納する。ビット 31 以下のシフトにより空いたビットには 0 が入り、上位 32 ビットはゼロクリアされる。

SRLX 命令は、R[rs1]の全 64 ビットを右（下位方向）にシフトし、結果を R[rd]に格納する。シフトにより空いた左側（上位側）のビットには 0 が入る。

SRA 命令は、R[rs1]の下位 32 ビットを右（下位方向）にシフトし、結果を R[rd]に格納する。ビット 31 以下のシフトにより空いたビットには、R[rs1]のビット 31 の値がコピーされ、上位 32 ビットにも R[rs1]のビット 31 の値がコピーされる。

SRAX 命令は、R[rs1]の全 64 ビットを右（下位方向）にシフトし、結果を R[rd]に格納する。シフトにより空いた左側（上位側）のビットには、R[rs1]のビット 63 の値がコピーされる。

ROLX 命令は、R[rs1]の全 64 ビットを左（上位方向）にローテートし、結果を R[rd]に格納する。シフト命令と異なりローテート命令では、最左からあふれたビットは最右に入れられる。

> **Compatibility Note** ROLX 命令は SPARC64™ XIfx 固有の拡張命令である。SPARC V9 では命令語のビット 11 は *reserved* である。

シフト量が 0 のときは、シフトは行なわれないが、32 ビットシフト命令では上位 32 ビットに上で書いたような影響がある。

これらの命令は整数条件コード CCR を更新しない。

Programming Note　1 ビットの左算術シフト（とオーバフロー）は、ADDcc 命令で実現できる。

Programming Note　“sra reg_{rs1}, 0, reg_{rd}” で符号付き 32 ビットを 64 ビットに変換することができる。また、“srl reg_{rs1}, 0, reg_{rd}” で上位 32 ビットをクリアすることができる。

例外	対象命令	検出条件
illegal_instruction	すべて	*reserved* フィールドが 0 でない
	SLL, SLLX, ROLX	x = 0 かつ r = 1
	SRL, SRA, SRLX, SRAX	r = 1
illegal_action	すべて	XAR.v = 1 かつ、下記のいずれかが成立している場合 • XAR.urs1<2:1> ≠ 00_2 • i = 0 かつ XAR.urs2<2:1> ≠ 00_2 • i = 1 かつ XAR.urs2 ≠ 0 • XAR.urs3 ≠ 0 • XAR.urd<2:1> ≠ 00_2 • XAR.simd = 1

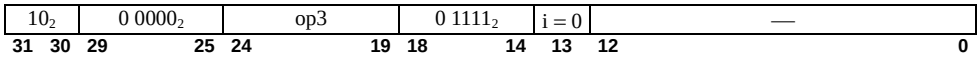
7.109. Signed Multiply (32-bit)

SMUL[D], SMULcc[D] は UA2011 7.118 参照

例外	検出条件
illegal_instruction	*reserved* フィールドが 0 でない
illegal_action	XAR.v = 1 かつ、下記のいずれかが成立している場合 • XAR.urs1<2:1> ≠ 00_2 • i = 0 かつ XAR.urs2<2:1> ≠ 00_2 • i = 1 かつ XAR.urs2 ≠ 0 • XAR.urs3 ≠ 0 • XAR.urd<2:1> ≠ 00_2 • XAR.simd = 1

7.110. Store Barrier

命令	op3	操作	HPC-ACE2 Regs	HPC-ACE2 SIMD	アセンブリ言語表記
STBAR	$10\ 1000_2$	nop			`stbar`



SPARC64™ XIfx においては STBAR は NOP として動作する。これは SPARC64™ XIfx ハードウェアのメモリモデルが、すべてのメモリアクセスにこの命令を挟んでいるのと等価なためである。

例外	検出条件
illegal_instruction	*reserved* フィールドが 0 でない
illegal_action	XAR.v = 1

7.111. Store Integer

STB, STH, STW, STX は UA2011 7.119 参照

例外	対象命令	検出条件
illegal_instruction	すべて	i = 0 かつ iw<12:5> ≠ 0000 0000_2
illegal_action	すべて	i = 0 かつ XAR.v = 1 かつ以下のいずれかの条件を満たしたとき • XAR.simd = 1 • XAR.urs1<2:1> ≠ 00_2 • XAR.urs2<2:1> ≠ 00_2 • XAR.urd<2:1> ≠ 00_2 i = 1 かつ XAR.v = 1 かつ以下のいずれかの条件を満たしたとき • XAR.simd = 1 • XAR.urs1<2:1> ≠ 00_2 • XAR.urs2 ≠ 0 • XAR.urd<2:1> ≠ 00_2
mem_address_not_aligned	STH	2 バイト境界でないとき
	STW	4 バイト境界でないとき
	STX	8 バイト境界でないとき
VA_watchpoint	すべて	
DAE_privilege_violation	すべて	
DAE_nfo_page	すべて	

7.112. Store Integer into Alternate Space

STBA^{PASI}, STHA^{PASI}, STWA^{PASI}, STXA^{PASI} は UA2011 7.120 参照

例外	対象命令	検出条件
illegal_action	すべて	i = 0 かつ XAR.v = 1 かつ以下のいずれかの条件を満たしたとき • XAR.simd = 1 • XAR.urs1<2:1> ≠ 00_2 • XAR.urs2<2:1> ≠ 00_2 • XAR.urd<2:1> ≠ 00_2 i = 1 かつ XAR.v = 1 かつ以下のいずれかの条件を満たしたとき • XAR.simd = 1 • XAR.urs1<2:1> ≠ 00_2 • XAR.urs2 ≠ 0 • XAR.urd<2:1> ≠ 00_2
mem_address_not_aligned	STHA	2 バイト境界でないとき
	STWA	4 バイト境界でないとき
	STXA	8 バイト境界でないとき
privileged_action	すべて	
VA_watchpoint	すべて	
DAE_invalid_asi	すべて	
DAE_privilege_violation	すべて	
DAE_nfo_page	すべて	

7.113. Block Initializing Store

UA2011 仕様では ASI_STBI_*の ASI が定義されている。SPARC64™ XIfx ではこれらの ASI を指定した場合、通常のストアとして動作する。STBA, STHA, STWA, STXA, STTWA 命令でこれらの ASI が指定された場合の動作は以下の通りである。

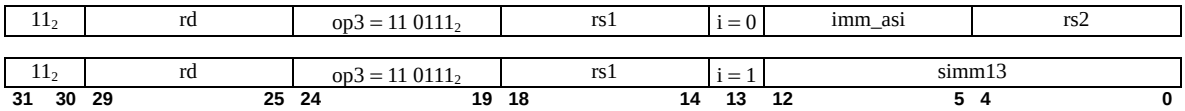
ASI 番号	ASI 名	整数ストア（STBA, STHA, STWA, STXA, STTWA）の動作
$E2_{16}$	ASI_STBI_P	ASI_P
$E3_{16}$	ASI_STBI_S	ASI_S
EA_{16}	ASI_STBI_PL	ASI_PL
EB_{16}	ASI_STBI_SL	ASI_SL

例外は *DAE_invalid_ASI* か *mem_address_not_aligned* のいずれか（*DAE_invalid_asi* 以外の *DAE_**例外は優先順位が低いので起こりえない)。

例外	対象命令	検出条件
illegal_action	すべて	i = 0 かつ XAR.v = 1 かつ以下のいずれかの条件を満たしたとき • XAR.simd = 1 • XAR.urs1<2:1> ≠ 00_2 • XAR.urs2<2:1> ≠ 00_2 • XAR.urd<2:1> ≠ 00_2 i = 1 かつ XAR.v = 1 かつ以下のいずれかの条件を満たしたとき • XAR.simd = 1 • XAR.urs1<2:1> ≠ 00_2 • XAR.urs2 ≠ 0 • XAR.urd<2:1> ≠ 00_2
mem_address_not_aligned	STHA	2 バイト境界でないとき
	STWA	4 バイト境界でないとき
	STXA	8 バイト境界でないとき
privileged_action	すべて	
VA_watchpoint	すべて	
DAE_invalid_asi	すべて	
DAE_privilege_violation	すべて	
DAE_nfo_page	すべて	

7.114. Block Store

命令	ASI	操作	HPC-ACE2 Regs	HPC-ACE2 SIMD	アセンブリ言語表記	
STBLOCKF	$F0_{16}$	プライマリアドレス空間に 64 バイトブロックストアを実行する。	✓		stda stda	*freg*$_{rd}$, [*regaddr*] ASI_BLK_P *freg*$_{rd}$, [*reg_plus_imm*] %asi
STBLOCKF	$F1_{16}$	セカンダリアドレス空間に 64 バイトブロックストアを実行する。	✓		stda stda	*freg*$_{rd}$, [*regaddr*] ASI_BLK_S *freg*$_{rd}$, [*reg_plus_imm*] %asi
STBLOCKF	$F8_{16}$	プライマリアドレス空間に 64 バイトブロックストアを実行する。リトルエンディアン。	✓		stda stda	*freg*$_{rd}$, [*regaddr*] ASI_BLK_PL *freg*$_{rd}$, [*reg_plus_imm*] %asi
STBLOCKF	$F9_{16}$	セカンダリアドレス空間に 64 バイトブロックストアを実行する。リトルエンディアン。	✓		stda stda	*freg*$_{rd}$, [*regaddr*] ASI_BLK_SL *freg*$_{rd}$, [*reg_plus_imm*] %asi
STBLOCKF	$E0_{16}$	プライマリアドレス空間に 64 バイトブロックコミットストアを実行する。	✓		stda stda	*freg*$_{rd}$, [*regaddr*] ASI_BLK_COMMIT_P *freg*$_{rd}$, [*reg_plus_imm*] %asi
STBLOCKF	$E1_{16}$	セカンダリアドレス空間に 64 バイトブロックコミットストアを実行する。	✓		stda stda	*freg*$_{rd}$, [*regaddr*] ASI_BLK_COMMIT_S *freg*$_{rd}$, [*reg_plus_imm*] %asi



STBLOCKF 命令は、STDFA 命令にブロックストア用に定義された ASI を指定することで実行される。STBLOCKF 命令でアクセスするメモリ空間は、通常のストアが行われるメモリ空間と同じである。

STBLOCKF 命令は、連続番号の倍精度浮動小数点レジスタ 8 つの内容を、64 バイト境界の 64 バイト領域に書き込む。書き込みアドレスは、i = 0 のときは “R[rs1] + R[rs2]” で、i = 1 のときは “R[rs1] + sign_ext(simm13)” で計算される。命令で指定されたアドレス（最低位アドレス）に命令で指定された倍精度浮動小数点レジスタ（一番小さい番号)の内容が書き込まれ、最低位アドレス+8 に次に小さい番号の倍精度浮動小数点レジスタの内容が書き込まれ、以後順に全 8 つのレジスタが 64 バイト領域に書き込まれる。リトルエンディアンのブロックストアでは、ひとつの倍精度浮動小数点レジスタの 8 バイト内でエンディアンの変換が行われる。

SPARC64™ XIfx ではブロックストア命令とブロックコミットストア命令は完全に同じ動作をする。

> **Compatibility Note** ブロックコミットストア命令のキャッシュに対する副作用は従来仕様とは大きく異なる。従来仕様では、キャッシュにデータがあれば全部消して、データをメモリに書く仕様だった。

STBLOCKF 命令でアトミシティが保障されるのは、レジスタ毎の 8 バイトである。

SPARC64™ XIfx の STBLOCKF 命令は、前後のロード・ストア命令間で TSO を遵守する。ブロックストアの各 8 バイトストア間でも TSO を遵守する。

SPARC64™ XIfx の STBLOCKF 命令は、他のストア命令と同様、レジスタ依存関係にある命令のプログラム実行順序を保障する。

STBLOCKF 命令のキャッシュに対する作用は、通常のストア命令と同じである。すなわち、L1D キャッシュ上にデータがあればL1D キャッシュに書きこみ、L1D キャッシュ上になければL1D キャッシュに読み込んだ上で当該データを更新する。

STBLOCKF 命令をノンキャッシャブル空間に対して実行することは可能である。

STBLOCKF 命令は最初の 8 バイトストアに対してのみ、*VA_watchpoint* 例外を検出する。

例外	対象命令	検出条件
illegal_instruction	すべて	rd に 16 の倍数以外のレジスタが指定された場合。
fp_disabled	すべて	PSTATE.pef = 0 または FPRS.fef = 0
illegal_action	すべて	XAR.v = 1 かつ下記のいずれかが成立している場合 • XAR.simd = 1 • XAR.urs1<2:1> ≠ 00_2 • i = 0 かつ XAR.urs2<2:1> ≠ 00_2 • i = 1 かつ XAR.urs2 ≠ 0
mem_address_not_aligned	すべて	アドレスが 64 バイト境界にないとき。
VA_watchpoint	すべて	最低位アドレスの 8 バイトに対するアクセスのみ。
DAE_privilege_violation	すべて	PSTATE.priv = 0 かつ TTE.p = 1
DAE_nfo_page	すべて	

7.115. Store Partial Floating-Point

STPARTIALF は UA2011 7.125 参照

例外	検出条件
illegal_instruction	i = 1
fp_disabled	PSTATE.pef = 0 または FPRS.fef = 0
illegal_action	XAR.v = 1
STDF_mem_address_not_aligned	4 バイト境界だが 8 バイト境界でないとき
mem_address_not_aligned	4 バイト境界でないとき
VA_watchpoint	
DAE_privilege_violation	
DAE_nfo_page	

7.116. Store Short Floating-Point

STSHORTF は UA2011 7.126 参照

例外	対象命令	検出条件
fp_disabled	すべて	PSTATE.pef = 0 または FPRS.fef = 0
illegal_action	すべて	XAR.v = 1
mem_address_not_aligned	ASI = 0xD2, 0xD3, 0xDA,0x DB を指定した時	2 バイト境界でないとき
VA_watchpoint	すべて	
DAE_privilege_violation	すべて	
DAE_nfo_page	すべて	

7.117. Store Integer Twin Word

STTW[D] は UA2011 7.127 参照

例外	検出条件
illegal_instruction	• rd に奇数番号のレジスタが指定されたとき • i = 0 かつ iw<12:5> ≠ $0000\ 0000_2$ のとき
illegal_action	i = 0 かつ XAR.v = 1 かつ以下のいずれかの条件を満たしたとき • XAR.simd = 1 • XAR.urs1<2:1> ≠ 00_2 • XAR.urs2<2:1> ≠ 00_2 • XAR.urd<2:1> ≠ 00_2 i = 1 かつ XAR.v = 1 かつ以下のいずれかの条件を満たしたとき • XAR.simd = 1 • XAR.urs1<2:1> ≠ 00_2 • XAR.urs2 ≠ 0 • XAR.urd<2:1> ≠ 00_2
mem_address_not_aligned	8 バイト境界でないとき
VA_watchpoint	
DAE_privilege_violation	
DAE_nfo_page	

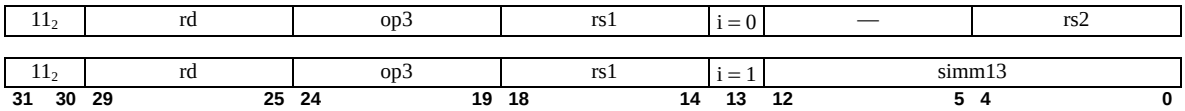
7.118. Store Integer Twin Word into Alternate Space

STTWA^{D,PASI} は UA2011 7.128 参照

例外	検出条件
illegal_instruction	rd に奇数番号のレジスタが指定されたとき
illegal_action	i = 0 かつ XAR.v = 1 かつ以下のいずれかの条件を満たしたとき • XAR.simd = 1 • XAR.urs1<2:1> ≠ 00_2 • XAR.urs2<2:1> ≠ 00_2 • XAR.urd<2:1> ≠ 00_2 i = 1 かつ XAR.v = 1 かつ以下のいずれかの条件を満たしたとき • XAR.simd = 1 • XAR.urs1<2:1> ≠ 00_2 • XAR.urs2 ≠ 0 • XAR.urd<2:1> ≠ 00_2
mem_address_not_aligned	8 バイト境界でないとき
privileged_action	
VA_watchpoint	
DAE_invalid_asi	
DAE_privilege_violation	
DAE_nfo_page	

7.119. Store Floating-Point State Register

命令	op3	rd	操作	HPC-ACE2 Regs	HPC-ACE2 SIMD	アセンブリ言語表記	
STFSR[D]	$10\,0101_2$	0	FSR のメモリ書き込み（下位 32 ビットのみ）	✓		`st`	%fsr, [*address*]
STXFSR	$10\,0101_2$	1	FSR のメモリ書き込み	✓		`stx`	%fsr, [*address*]
—	$10\,0101_2$	2 – 31	*reserved*				



動作説明

STFSR 命令は、未完了のすべての浮動小数点演算の完了を待ってから、FSR の下位 32 ビットを 4 バイト境界の 4 バイト領域に書き込む。書き込み後、FSR.ftt はゼロクリアされる。

STXFSR 命令は、未完了のすべての浮動小数点演算の完了を待ってから、FSR の全 64 ビットを 4 バイト境界の 8 バイト領域に書き込む。書き込み後、FSR.ftt はゼロクリアされる。

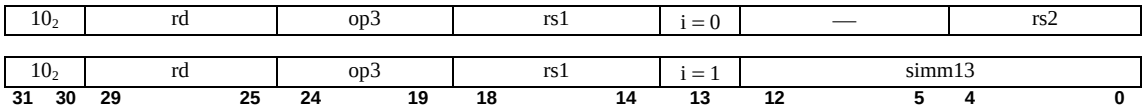
STFSR および STXFSR 命令は、暗黙の ASI（UA2011 6.3.1.3 参照）を使いメモリにアクセスする。書き込みアドレスは、i = 0 のときは “R[rs1] + R[rs2]” で、i = 1 のときは “R[rs1] + sign_ext(simm13)” で計算される。

STFSR 命令は、4 バイト境界にないアドレスにアクセスすると *mem_address_not_aligned* 例外を発生する。STXFSR 命令は、8 バイト境界にないアドレスにアクセスすると *mem_address_not_aligned* 例外を発生する。

例外	対象命令	検出条件
illegal_instruction	STFSR, STXFSR	i = 0 かつ *reserved* が 0 でないとき。
	—	rd = 2–31
fp_disabled	すべて	PSTATE.pef = 0 または FPRS.fef = 0
illegal_action	すべて	XAR.v = 1 かつ下記のいずれかが成立している場合 • XAR.simd = 1 • XAR.urs1<2:1> ≠ 00_2 • i = 0 かつ XAR.urs2<2:1> ≠ 00_2 • i = 1 かつ XAR.urs2 ≠ 0 • XAR.urd ≠ 0
mem_address_not_aligned	STFSR	4 バイト境界ではないアドレスにアクセスしたとき。
	STXFSR	8 バイト境界ではないアドレスにアクセスしたとき。
VA_watchpoint	すべて	
DAE_privilege_violation	すべて	
DAE_nfo_page	すべて	

7.120. Subtract

命令	op3	操作	HPC-ACE2 Regs.	HPC-ACE2 SIMD	アセンブリ言語表記	
SUB	$00\ 0100_2$	整数減算	✓		sub	*reg_{rs1}, reg_or_imm, reg_{rd}*
SUBcc	$01\ 0100_2$	整数減算と cc の更新	✓		subcc	*reg_{rs1}, reg_or_imm, reg_{rd}*
SUBC	$00\ 1100_2$	キャリーを含む整数減算	✓		subc	*reg_{rs1}, reg_or_imm, reg_{rd}*
SUBCcc	$01\ 1100_2$	キャリーを含む整数減算と cc の更新	✓		subccc	*reg_{rs1}, reg_or_imm, reg_{rd}*



動作説明

i = 0 のとき、SUB と SUBcc は、 “R[rs1] – R[rs2]” を処理する。i = 1 のときは、“R[rs1] – sign_ext(simm13)” を処理する。どちらの場合も結果は、R[rd]に書き込まれる。

32 ビット演算結果のキャリーを入力とする SUB（SUBC と SUBCcc）では、CCR レジスタのキャリービット(icc.c)も加える。すなわち、i = 0 のときは、 “R[rs1] – R[rs2] – icc.c” を処理し、i = 1 のときは、“R[rs1] – sign_ext(simm13) – icc.c” を処理する。どちらの場合も合計は、R[rd]に書き込まれる。

SUBcc と SUBCcc は、整数条件コード（CCR.icc と CCR.xcc）を変更する。

- 32 ビットオーバーフロー（CCR.icc.v）は、入力オペランド 2 つのそれぞれのビット 31 (符号ビット）が異なり、減算結果のビット 31 (符号ビット）が R[rs1]<31>と異なるときに、1 がセットされる。
- 64 ビットオーバーフロー（CCR.xcc.v）は、入力オペランド 2 つのそれぞれのビット 63 (符号ビット）が異なり、減算結果のビット 63 (符号ビット）が R[rs1]<63>と異なるときに、1 がセットされる。

Programming Note rd = 0 の SUBcc 命令は、符号ありまたは符号なし整数の比較に使われる。

Programming Note SUBC と SUBCcc は、32 ビット条件コードのキャリービット（CCR.icc.c）を読み出す。64 ビット条件コードのキャリービット (CCR.xcc.c）は読み出さない。

例外	対象命令	検出条件
illegal_instruction	すべて	*reserved* フィールドが 0 でない (i = 0 かつ iw<12:5> ≠ $0000\ 0000_2$)
illegal_action	すべて	XAR.v = 1 かつ、下記のいずれかが成立している場合 • XAR.simd = 1 • XAR.urs1<2:1> ≠ 00_2 • i = 0 かつ XAR.urs2<2:1> ≠ 00_2 • i = 1 かつ XAR.urs2 ≠ 0 • XAR.urs3 ≠ 0 • XAR.urd<2:1> ≠ 00_2

7.121. Swap Register with Memory

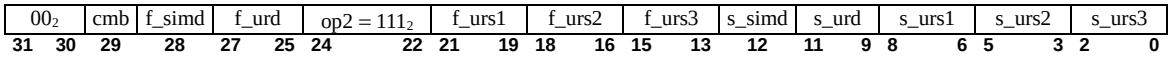
SWAP^{D} は UA2011 7.131 参照

SWAPA$^{D, PASI}$ は UA2011 7.132 参照

例外	検出条件
illegal_instruction	i = 0 かつ iw<12:5> ≠ 0000 0000_2
illegal_action	i = 0 かつ XAR.v = 1 かつ以下のいずれかの条件を満たしたとき • XAR.simd = 1 • XAR.urs1<2:1> ≠ 00_2 • XAR.urs2<2:1> ≠ 00_2 • XAR.urd<2:1> ≠ 00_2 i = 1 かつ XAR.v = 1 かつ以下のいずれかの条件を満たしたとき • XAR.simd = 1 • XAR.urs1<2:1> ≠ 00_2 • XAR.urs2 ≠ 0 • XAR.urd<2:1> ≠ 00_2
mem_address_not_aligned	4 バイト境界でないとき
VA_watchpoint	
DAE_privilege_violation	
DAE_nfo_page	

7.122. Set XAR (SXAR)

命令	op2	cmb	操作	HPC-ACE2 Regs	HPC-ACE2 SIMD	アセンブリ言語表記
SXAR1	111_2	0	後続 1 命令の XAR 指定			sxar1
SXAR2	111_2	1	後続 2 命令の XAR 指定			sxar2



動作説明

SXAR 命令は XAR を更新する命令である。XAR が 2 命令までの値を保持できるのに対応し、1 命令分指定できる SXAR1 と 2 命令分指定できる SXAR2 がある。f_で始まるフィールドは SXAR 命令の直後に実行される命令に適用され、s_で始まるフィールドはその次に実行される命令に適用される。

SXAR1 命令の s_で始まるフィールドは、0 でなくてはならない。s_で始まるフィールドに 0 以外の値が入っていると、*illegal_instruction* 例外が発生する。

Compatibility Note SPARC64™ VIIIfx および SPARC64™ IXfx では、SXAR1 命令の s_で始まるフィールドに 0 以外の値が入っていても無視された。

SXAR 命令は後続の命令で、SPARC64™ XIfx で拡張された整数・浮動小数点レジスタを使う場合や SIMD 拡張動作を指示する場合に使われる。また、メモリアクセス命令でハードウェアプリフェッチやセクタキャッシュの指示を行う場合にも使われる。命令フィールドの、*_simd、*_urs1、*_urs2、*_urs3 および*_urd に全て 0 が書かれていたとしても、SXAR1 命令を実行すると XAR.f_v = 1 となり、SXAR2 命令を実行すると XAR.f_v = 1 かつ XAR.s_v = 1 となる。

SXAR 命令と後続命令がメモリ上連続していない場合、たとえば SXAR がディレイスロットに置かれていたり、Tcc 命令をはさんでいると、命令実行性能は低下する可能性がある。

SXAR 命令自身は XAR 対象命令ではないので、実行時に XAR.v = 1 だと *illegal_action* 例外が発生する。

Compatibility Note op = 00_2, op2 = 111_2 は SPARC V9 では *reserved* だが、SPARC V8 では FBcc 命令が定義されていた。SPARC V8 のアプリケーションを SPARC64™ XIfx で実行すると、動作が異なる可能性がある。

Programming Note SXAR 命令は命令語中に XAR にセットする値を抱えているが、アセンブリ言語の表記上は引数を持たない。HPC-ACE2 拡張動作は後続命令のニーモニックにサフィックスで記述し、アセンブラが SXAR に落とし込むこと。

例外	対象命令	検出条件
illegal_instruction	SXAR1	s_* ≠ 0
illegal_action	すべて	XAR.v = 1

7.123. Tagged Add and Subtract

TADDcc は UA2011 7.133 参照

TSUBcc は UA2011 7.136 参照

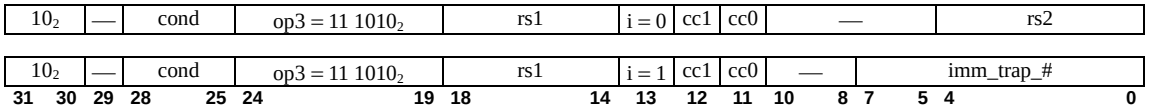
TADDccTV^{D} は UA2011 7.134 参照

TSUBccTV^{D} は UA2011 7.137 参照

例外	検出条件
illegal_instruction	i = 0 かつ iw<12:5> ≠ 0000 0000_2
illegal_action	i = 0 かつ XAR.v = 1 かつ以下のいずれかの条件を満たしたとき • XAR.simd = 1 • XAR.urs1<2:1> ≠ 00_2 • XAR.urs2<2:1> ≠ 00_2 • XAR.urs3 ≠ 0 • XAR.urd<2:1> ≠ 00_2 i = 1 かつ XAR.v = 1 かつ以下のいずれかの条件を満たしたとき • XAR.simd = 1 • XAR.urs1<2:1> ≠ 00_2 • XAR.urs2 ≠ 0 • XAR.urs3 ≠ 0 • XAR.urd<2:1> ≠ 00_2

7.124. Trap on Integer Condition Code (Tcc)

命令	cond	操作	条件 (icc または xcc)	HPC-ACE2 Regs.	HPC-ACE2 SIMD	アセンブリ言語表記
TA	1000_2	常にトラップする	1			`ta` *i_or_xcc*, *software_trap_number*
TN	0000_2	常にトラップしない	0			`tn` *i_or_xcc*, *software_trap_number*
TNE	1001_2	等しくないときにトラップ	**not** Z			`tne` *i_or_xcc*, *software_trap_number* `tnz` *i_or_xcc*, *software_trap_number*
TE	0001_2	等しいときトラップ	Z			`te` *i_or_xcc*, *software_trap_number* `tz` *i_or_xcc*, *software_trap_number*
TG	1010_2	より大きいときトラップ	**not** (Z **or** (N **xor** V))			`tg` *i_or_xcc*, *software_trap_number*
TLE	0010_2	以下のときトラップ	Z **or** (N **xor** V)			`tle` *i_or_xcc*, *software_trap_number*
TGE	1011_2	以上のときトラップ	**not** (N **xor** V)			`tge` *i_or_xcc*, *software_trap_number*
TL	0011_2	より小さいときトラップ	N **xor** V			`tl` *i_or_xcc*, *software_trap_number*
TGU	1100_2	符号なし整数としての比較で、より大きいときトラップ	**not** (C **or** Z)			`tgu` *i_or_xcc*, *software_trap_number*
TLEU	0100_2	符号なし整数としての比較で、以下のときトラップ	C **or** Z			`tleu` *i_or_xcc*, *software_trap_number*
TCC	1101_2	キャリークリア（符号なし整数としての比較で以上）のときトラップ	**not** C			`tcc` *i_or_xcc*, *software_trap_number* `tgeu` *i_or_xcc*, *software_trap_number*
TCS	0101_2	キャリーセット（符号なし整数でより小さい）のときトラップ	C			`tcs` *i_or_xcc*, *software_trap_number* `tlu` *i_or_xcc*, *software_trap_number*
TPOS	1110_2	正のときトラップ	**not** N			`tpos` *i_or_xcc*, *software_trap_number*
TNEG	0110_2	負のときトラップ	N			`tneg` *i_or_xcc*, *software_trap_number*
TVC	1111_2	オーバーフロークリアのときトラップ	**not** V			`tvc` *i_or_xcc*, *software_trap_number*
TVS	0111_2	オーバーフローのときトラップ	V			`tvs` *i_or_xcc*, *software_trap_number*



cc1	cc0	条件コード
0	0	CCR.icc
0	1	—
1	0	CCR.xcc
1	1	—

動作説明　　Tcc 命令は、icc または xcc の条件が成立した場合に、*trap_instruction* トラップを発生させる。

Tcc 命令の動作は、XAR の影響を受けない。XAR.v = 1 であっても *illegal_action* 例外を検出しない。

icc または xcc の条件が成立しない場合は、トラップは発生させないが、XAR の設定はクリアする。すなわち、XAR.f_v = 1 ならば XAR.f_*が 0 クリアされ、XAR.f_v = 0 かつ XAR.s_v = 1 ならば XAR.s_*が 0 クリアされる。

Programming Note Tcc 命令は、ブレークポイント、トレース、特権モードソフトウェアの呼び出しに使われる。また、実行時のチェック、たとえば配列の範囲チェックや整数オーバフローチェックなどにも使われる。デバッガのブレークポイントとして使うためには、命令列のどの場所にも挿入できなくてはいけないので、XAR を無視する仕様になっている。

以下の説明ではソフトウェアトラップ番号を SWTN と表記する。SWTN の有効な値は特権レベルにより異なる。

- 非特権モードのとき、SWTN は、i = 0 だと “R[rs1] + R[rs2]” , i = 1 だと “R[rs1] + imm_trap_#” で求められた値の下位 7 ビットとなる。よって非特権モードでは 0 – 127 の範囲のトラップ番号が使える。上位 57 ビットは使われないが、ソフトウェアは 0 を指定すること。

表 7-26 SWTN、特権レベルと発生するトラップの関係

特権レベル	SWTN	
	0 – 127	128 – 255
非特権モード PSTATE.priv = 0	*trap_instruction*	— (SWTN の有効ビットは 7 ビットなので、起こらない。)

Tcc 命令は、トラップ発生条件が成立して PSTATE.tct = 1 の場合、制御転送を起こさずに *control_transfer_instruction* 例外を生成する。*control_transfer_instruction* トラップが生じるとき、PC (Tcc 命令のアドレス) は TPC[TL]に保存され、Tcc が実行される前の NPC の値は TNPC[TL]に保存される。TN は常にトラップを発生しないので、*control_transfer_instruction* トラップを生じることもない。

Tcc 命令が *trap_instruction* トラップを発生させる場合、SWTN に 256 を足した値が TT[TL]に書き込まれ、通常のトラップ処理が行われる。

トラップが発生するとき、TPC[TL]および TNPC[TL]に保存される値は、PSTATE.am の影響を受ける。

トラップ処理の詳細は 372,ページを参照。

例外	対象命令	検出条件
illegal_instruction	すべて	以下のいずれかが成立するとき • *reserved* が 0 でない。 • cc0 = 1
illegal_action	すべて	XAR.v = 1
control_transfer_instruction	TN 以外	条件分岐が成立し、PSTATE.tct = 1 の場合。 TA 命令は常に条件分岐が成立しているとする。
trap_instruction	TN 以外	本文参照。

7.125. Unsigned Divide (64-bit ÷ 32-bit)

UDIV^{D}, UDIVcc^{D} は UA2011 7.138 参照

例外	検出条件
illegal_instruction	i = 0 かつ iw<12:5> ≠ 0000 0000_2
illegal_action	i = 0 かつ XAR.v = 1 かつ以下のいずれかの条件を満たしたとき • XAR.simd = 1 • XAR.urs1<2:1> ≠ 00_2 • XAR.urs2<2:1> ≠ 00_2 • XAR.urs3 ≠ 0 • XAR.urd<2:1> ≠ 00_2 i = 1 かつ XAR.v = 1 かつ以下のいずれかの条件を満たしたとき • XAR.simd = 1 • XAR.urs1<2:1> ≠ 00_2 • XAR.urs2 ≠ 0 • XAR.urs3 ≠ 0 • XAR.urd<2:1> ≠ 00_2
division_by_zero	除数が 0 のとき

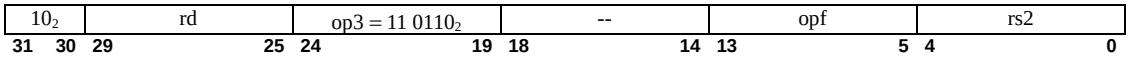
7.126. Unsigned Multiply (32-bit)

UMUL[D], UMULcc[D] は UA2011 7.139 参照

例外	検出条件
illegal_instruction	i = 0 かつ iw<12:5> ≠ $0000\ 0000_2$
illegal_action	i = 0 かつ XAR.v = 1 かつ以下のいずれかの条件を満たしたとき • XAR.simd = 1 • XAR.urs1<2:1> ≠ 00_2 • XAR.urs2<2:1> ≠ 00_2 • XAR.urs3 ≠ 0 • XAR.urd<2:1> ≠ 00_2 i = 1 かつ XAR.v = 1 かつ以下のいずれかの条件を満たしたとき • XAR.simd = 1 • XAR.urs1<2:1> ≠ 00_2 • XAR.urs2 ≠ 0 • XAR.urs3 ≠ 0 • XAR.urd<2:1> ≠ 00_2

7.127. Leading Zero Detect

命令	opf	操作	HPC-ACE2		アセンブリ言語表記	
			Regs.	SIMD		
LZD	$0\ 0001\ 0111_2$	R[rs2]の左から連続する 0 の数のカウント	✓		lzd	*reg*$_{rs2}$, *reg*$_{rd}$



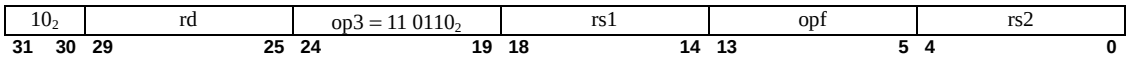
動作説明　R[rs2]の左から連続する 0 の数を検出し（0～64）、R[rd]に書き込む。1 が検出されたらその時点でカウントを中止する。R[rd]は下位 7 ビットのみを用い、上位 57 ビットには 0 が書き込まれる。

Programming Note　LZD 命令の実行前に R[rs2]を 1 の補数にすることで、R[rs2]の左から連続する 1 の数を検出することも可能である。

例外	検出条件
illegal_instruction	iw<18:14> ≠ $0\ 0000_2$
illegal_action	XAR.v = 1 かつ、下記のいずれかが成立している場合 • XAR.simd = 1 • XAR.urs1 ≠ 0 • XAR.urs2<2:1> ≠ 00_2 • XAR.urs3 ≠ 0 • XAR.urd<2:1> ≠ 00_2

7.128. Partitioned Unsigned Compare

命令	opf	操作	HPC-ACE2	アセンブリ言語表記	
			Regs. SIMD		
FPCMPULE8	$1\ 0010\ 0000_2$	8 つの 8-bit 符号なし整数比較 *src1* ≤ *src2* ならば 1		fpcmpule8	$freg_{rs1}$, $freg_{rs2}$, reg_{rd}
FPCMPUNE8	$1\ 0010\ 0010_2$	8 つの 8-bit 符号なし整数比較 *src1* ≠ *src2* ならば 1		fpcmpune8	$freg_{rs1}$, $freg_{rs2}$, reg_{rd}
FPCMPUGT8	$1\ 0010\ 1000_2$	8 つの 8-bit 符号なし整数比較 *src1* > *src2* ならば 1		fpcmpugt8	$freg_{rs1}$, $freg_{rs2}$, reg_{rd}
FPCMPUEQ8	$1\ 0010\ 1010_2$	8 つの 8-bit 符号なし整数比較 *src1* = *src2* ならば 1		fpcmpueq8	$freg_{rs1}$, $freg_{rs2}$, reg_{rd}



動作説明

SIMD 符号なし比較命令は 2 つの浮動小数点レジスタ Fd[rs1], Fd[rs2]に含まれる複数の要素を比較し、結果を整数レジスタ R[rd]に格納する。各要素の比較結果は R[rd]の下位 8 ビットに左から順に格納され、それ以外のビットには 0 が格納される。

64-bit の入力レジスタには 8 要素が含まれる。要素数および要素のビット位置を以下に示す。

表 7-50 要素数および要素のビット位置

要素数	要素 1	要素 2	要素 3	要素 4	要素 5	要素 6	要素 7	要素 8
8	63:56	55:48	47:40	39:32	31:24	23:16	15:8	7:0

Fd[rs1]と Fd[rs2]に含まれる同じ位置の要素が比較され、その結果が R[rd]の対応するビットに格納される。それぞれの要素の比較結果を格納する R[rd]のビットを以下に示す。

表 7-51 各要素の比較結果格納ビット位置

	要素 1	要素 2	要素 3	要素 4	要素 5	要素 6	要素 7	要素 8
R[rd]	7	6	5	4	3	2	1	0

FPCMPULE8 は Fd[rs1]と Fd[rs2]の同じ位置の要素を符号なし整数として比較し、Fd[rs1]の要素が Fd[rs2]の要素より小さいか一致すれば R[rd]の対応するビットに 1 を格納する。

FPCMPUNE8 は Fd[rs1]と Fd[rs2]の同じ位置の要素を符号なし整数として比較し、一致しなければ R[rd]の対応するビットに 1 を格納する。

FPCMPGT8 は Fd[rs1]と Fd[rs2]の同じ位置の要素を符号なし整数として比較し、Fd[rs1]の要素が Fd[rs2]の要素より大きければ R[rd]の対応するビットに 1 を格納する。

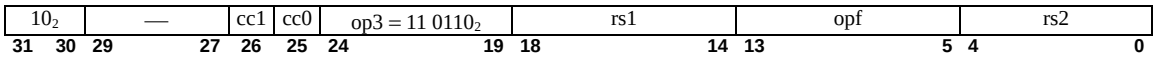
FPCMPUEQ8 は Fd[rs1]と Fd[rs2]の同じ位置の要素を符号なし整数として比較し、一致すれば R[rd]の対応するビットに 1 を格納する。

FPCMPU{LE|NE|GT|EQ}8 命令は FSR のどのフィールドも更新しない。

例外	検出条件
fp_disabled	PSTATE.pef = 0 または FPRS.fef = 0
illegal_action	XAR.v = 1

7.129. Floating-Point Lexicographic Compare

命令	opf	操作	HPC-ACE2		アセンブリ言語表記	
			Regs.	SIMD		
FLCMPs	$1\ 0101\ 0001_2$	単精度浮動小数点数の辞書式比較	✓		flcmps	%fccn, *freg*$_{rs1}$, *freg*$_{rs2}$
FLCMPd	$1\ 0101\ 0010_2$	倍精度浮動小数点数の辞書式比較	✓		flcmpd	%fccn, *freg*$_{rs1}$, *freg*$_{rs2}$



cc1	cc0	条件コード
0	0	fcc0
0	1	fcc1
1	0	fcc2
1	1	fcc3

動作説明

FLCMPs 命令は、Fs[rs1]と Fs[rs2]を単精度浮動小数点として辞書式比較を行い、命令で指定した浮動小数点条件コードフィールド FSR.fccn に格納する。

FLCMPd 命令は、Fd[rs1]と Fd[rs2]を倍精度浮動小数点として辞書式比較を行い、命令で指定した浮動小数点条件コードフィールド FSR.fccn に格納する。

–0 と+0 を異なる数値として扱う。–0 と+0 を比較すると、–0 が小さい。

入力値の関係	%fccn 出力
F[rs1] ≥ F[rs2] (どちらも NaN ではない)	0
F[rs1] < F[rs2] (どちらも NaN ではない)	1
F[rs1] が NaN かつ F[rs2]が NaN ではない	2
F[rs2]が NaN(F[rs1]に依存しない)	3

Programming Note FLCMP{s|d}命令の実行で FSR.cexc と FSR.aexc の値は変化しない。また、*fp_exception_other* 例外は発生しない。

Programming Note FLCMP{s|d}の FSR.fccn への出力は、他の SPARC V9 の浮動小数点比較命令とは異なる。そのため、FLCMP{s|d}に依存する MOVE 命令や分岐命令は、(他の SPARCV9 の浮動小数点命令に対して) ニーモニックの意味が異なる。

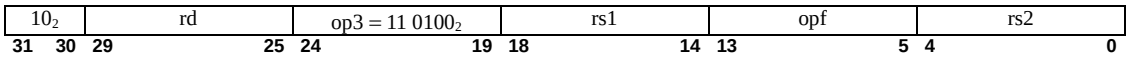
表 7-42　FLCMP{s|d}の演算結果

		F[rs2]							
		-∞	-N	-0	+0	+N	+∞	QNaN2	SNaN2
F[rs1]	-∞					—			
	-N		—			1			
			0, 1						
	-0								
	+0							3	
	+N		—			—			
			0			0, 1			
	+∞								
	QNaN1								
	SNaN1		2						

例外	検出条件
fp_disabled	PSTATE.pef = 0 または FPRS.fef = 0
Illegal_instruction	iw<29:27> ≠ 000_2
illegal_action	XAR.v = 1 かつ、下記のいずれかが成立している場合 • XAR.simd = 1 • XAR.urs3 ≠ 0 • XAR.urd ≠ 0

7.130. Floating-Point Negative Add

命令	opf	操作	HPC-ACE2		アセンブリ言語表記	
			Regs.	SIMD		
FNADDs	0 0101 0001_2	浮動小数点加算結果の符合を反転（単精度）	✓	✓	fnadds	*freg*$_{rs1}$, *freg*$_{rs2}$, *freg*$_{rd}$
FNADDd	0 0101 0010_2	浮動小数点加算結果の符合を反転（倍精度）	✓	✓	fnaddd	*freg*$_{rs1}$, *freg*$_{rs2}$, *freg*$_{rd}$



動作説明

FNADDs 命令は、Fs[rs1]の単精度浮動小数点数と Fs[rs2]の単精度浮動小数点数を加算し、その結果の符合を反転して Fs[rd]へ格納する。

FNADDd 命令は、Fd[rs1]の倍精度浮動小数点数と Fd[rs2]の倍精度浮動小数点数を加算し、その結果の符合を反転して Fd[rd]へ格納する。

丸めは、加算の後ではなく、符号反転後に行われる。

NaN に関しての扱いは、以下の通りである。

- 入力として QNaN が使用された場合、符号ビットは変化しない
- 出力として QNaN が生成された場合、符号ビットは 0 となる
- 入力として SNaN が使用され、出力として QNaN が生成された場合、QNaN は SNaN の符号ビットを保持する

FSR.cexc と FSR.aexc の ofa/ofc, ufa/ufc, nxa/nxc のビットは、演算の最終結果で更新される（途中の加算の結果では更新されない）。FSR.cexc と FSR.aexc の nva/nvc ビットは、加算と符号反転が 2 つの独立した命令で実行されたものとして更新される。nva/nvc ビットは、以下の条件を満たすとき 1 がセットされる。

- F[rs1]か F[rs2]が SNaN のとき
- ∞ – ∞のとき

表 7-27　FNADD{s|d}の演算結果

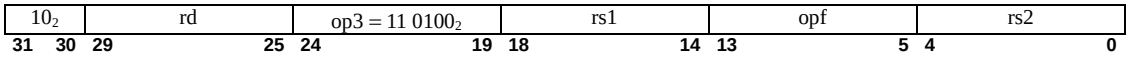
<table>
<tr><td colspan="2" rowspan="2"></td><td colspan="8">F[rs2]</td></tr>
<tr><td>-∞</td><td>-N</td><td>-0</td><td>+0</td><td>+N</td><td>+∞</td><td>QNaN2</td><td>SNaN2</td></tr>
<tr><td rowspan="9">F[rs1]</td><td>-∞</td><td colspan="5" rowspan="2">—
+∞</td><td>NV
dQNaN</td><td rowspan="8">—
QNaN2</td><td rowspan="8">NV
QSNaN2</td></tr>
<tr><td>-N</td><td>—
-(F[rs1] + F[rs2])</td><td colspan="2">—
-F[rs1]</td><td>—
-(F[rs1] + F[rs2])[xxviii]</td><td rowspan="6"></td></tr>
<tr><td>-0</td><td rowspan="2">—
-F[rs2]</td><td>—
-0</td><td>—
+0[xxix]</td><td rowspan="2">—
-F[rs2]</td></tr>
<tr><td>+0</td><td>—
+0[xxix]</td><td>—
+0</td></tr>
<tr><td>+N</td><td>—
-(F[rs1] + F[rs2])[xxviii]</td><td colspan="2">—
-F[rs1]</td><td>—
-(F[rs1] + F[rs2])</td></tr>
<tr><td>+∞</td><td>NV
dQNaN</td><td colspan="4">—
-∞</td></tr>
<tr><td>QNaN1</td><td colspan="6">—
QNaN1</td></tr>
<tr><td>SNaN1</td><td colspan="7">NV
QSNaN1</td></tr>
</table>

例外	検出条件
fp_disabled	PSTATE.pef = 0 または FPRS.fef = 0
illegal_action	XAR.v = 1 かつ、下記のいずれかが成立している場合 • XAR.urs3 ≠ 0 • XAR.simd = 1 かつ XAR.urs1<2> ≠ 0 • XAR.simd = 1 かつ XAR.urs2<2> ≠ 0 • XAR.simd = 1 かつ XAR.urd<2> ≠ 0
fp_exception_ieee_754	FADD{s|d}と同じ
fp_exception_other	FADD{s|d}と同じ

[xxviii] 演算結果がゼロのときは、脚注 xxix に従う。
[xxix] 丸めモードが toward -∞のときは-0

7.131. Floating-Point Negative Multiply

命令	opf	操作	HPC-ACE2		アセンブリ言語表記	
			Regs.	SIMD		
FNMULs	0 0101 1001_2	乗算結果の符合を反転（単精度）	✓	✓	fnmuls	$freg_{rs1}$, $freg_{rs2}$, $freg_{rd}$
FNMULd	0 0101 1010	乗算結果の符合を反転（倍精度）	✓	✓	fnmuld	$freg_{rs1}$, $freg_{rs2}$, $freg_{rd}$
FNsMULd	0 0111 1001	乗算結果を倍精度に変換して符号を反転	✓	✓	fnsmuld	$freg_{rs1}$, $freg_{rs2}$, $freg_{rd}$



動作説明

FNMULs 命令は、Fs[rs1]の単精度浮動小数点数と Fs[rs2]の単精度浮動小数点数を乗算し、その結果の符合を反転して Fs[rd]へ格納する。

FNMULd 命令は、Fd[rs1]の倍精度浮動小数点数と Fd[rs2]の倍精度浮動小数点数を乗算し、その結果の符合を反転して Fd[rd]へ格納する。

FNsMULd 命令は、Fs[rs1]の単精度浮動小数点数と Fs[rs2]の単精度浮動小数点数を乗算し、倍精度浮動小数点数で結果を出力し、符号を反転して Fd[rd]へ格納する。

丸めは、乗算の後ではなく、符号反転後に行われる。

NaN に関しての扱いは、以下の通りである。

- 入力として QNaN が使用された場合、符号ビットは変化しない
- 出力として QNaN が生成された場合、符号ビットは 0 となる
- 入力として SNaN が使用され、出力として QNaN が生成された場合、QNaN は SNaN の符号ビットを保持する

FSR.cexc と FSR.aexc の ofa/ofc, ufa/ufc, nxa/nxc のビットは、演算の最終結果で更新される（途中の乗算の結果では更新されない）。FSR.cexc と FSR.aexc の nva/nvc ビットは、乗算と符号反転が 2 つの独立した命令で実行されたものとして更新される。nva/nvc ビットは、以下の条件を満たすとき 1 がセットされる。

- F[rs1]か F[rs2]が SNaN のとき
- $\infty \times 0$ のとき

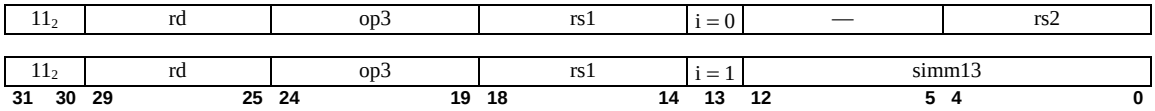
表 7-28　**FNMUL{s|d}, FNsMULd の演算結果**

<table>
<tr><td colspan="2" rowspan="2"></td><td colspan="8">F[rs2]</td></tr>
<tr><td>-∞</td><td>-N</td><td>-0</td><td>+0</td><td>+N</td><td>+∞</td><td>QNaN2</td><td>SNaN2</td></tr>
<tr><td rowspan="8">F[rs1]</td><td>-∞</td><td rowspan="2">—
-∞</td><td colspan="2" rowspan="2"></td><td colspan="2">NV
dNaN</td><td colspan="2" rowspan="2"></td><td rowspan="2">—
+∞</td><td rowspan="8">—
QNaN2</td><td rowspan="8">NV
QSNaN2</td></tr>
<tr><td>-N</td><td rowspan="2">—
-(F[rs1] × F[rs2])</td><td rowspan="2"></td><td rowspan="2"></td><td rowspan="2">—
-(F[rs1] × F[rs2])</td></tr>
<tr><td>-0</td><td rowspan="3">NV
dQNaN</td><td colspan="2" rowspan="2">—
-0</td><td rowspan="2">—
+0</td><td rowspan="3">NV
dQNaN</td></tr>
<tr><td>+0</td><td colspan="2" rowspan="2">—
+0</td><td rowspan="2">—
-0</td></tr>
<tr><td>+N</td><td rowspan="2">—
-(F[rs1] × F[rs2])</td><td rowspan="2"></td><td rowspan="2">—
-(F[rs1] × F[rs2])</td></tr>
<tr><td>+∞</td><td>—
+∞</td><td colspan="2">NV
dNaN</td><td>—
-∞</td></tr>
<tr><td>QNaN1</td><td colspan="7">—
QNaN1</td></tr>
<tr><td>SNaN1</td><td colspan="7">NV
QSNaN1</td></tr>
</table>

例外	検出条件
fp_disabled	PSTATE.pef = 0 または FPRS.fef = 0
illegal_action	XAR.v = 1 かつ、下記のいずれかが成立している場合 • XAR.urs3 ≠ 0 • XAR.simd = 1 かつ XAR.urs1<2> ≠ 0 • XAR.simd = 1 かつ XAR.urs2<2> ≠ 0 • XAR.simd = 1 かつ XAR.urd<2> ≠ 0
fp_exception_ieee_754	`FMUL{s|d}, FsMULd` と同じ
fp_exception_other	`FMUL{s|d}, FsMULd` と同じ

7.132. Load Entire Floating-Point State Register

命令	op3	rd 操作	HPC-ACE2 Regs	HPC-ACE2 SIMD	アセンブリ言語表記	
LDXEFSR	$10\ 0001_2$	3 メモリから FSR に読み出し	✓		ldx	[*address*], %efsr



動作説明

LDXEFSR 命令は、未完了のすべての浮動小数点演算の完了を待ってから、8 バイト境界の 8 バイト領域を読み出し FSR の全 64 ビットに格納する。ftt フィールドは更新するが、ver, qne, および *reserved* フィールドは更新しない。

Programming Note 将来の互換性のために、ソフトウェアは *reserved* フィールドには 0 をロードすべきである。

LDXEFSR 命令は、暗黙の ASI (UA2011 6.3.1.3 参照) を使いメモリにアクセスする。書き込みアドレスは、i = 0 のときは “R[rs1] + R[rs2]” で、i = 1 のときは “R[rs1] + sign_ext(simm13)” で計算される。

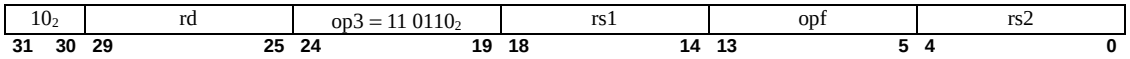
LDXEFSR 命令は、8 バイト境界にないアドレスにアクセスすると *mem_address_not_aligned* 例外を発生する。

LDXEFSR 命令の実行により例外が発生しトラップが起きた場合、FSR は更新されない。

例外	検出条件
illegal_instruction	i = 0 かつ *reserved* が 0 でないとき。
fp_disabled	PSTATE.pef = 0 または FPRS.fef = 0
illegal_action	XAR.v = 1 かつ下記のいずれかが成立している場合 • XAR.simd = 1 • XAR.urs1<2:1> ≠ 00_2 • i = 0 かつ XAR.urs2<2:1> ≠ 00_2 • i = 1 かつ XAR.urs2 ≠ 0 • XAR.urd ≠ 0
mem_address_not_aligned	8 バイト境界ではないアドレスにアクセスしたとき。
VA_watchpoint	
DAE_privilege_violation	
DAE_nfo_page	

7.133. Partitioned Move Selected Floating-Point Register on Floating-Point Register's Condition

命令	opf	操作	HPC-ACE2		アセンブリ言語表記	
			Regs.	SIMD		
FPSELMOV8X	0 1001 0101_2	レジスタから 8 ビットの値を選択して移動	✓	✓	fpselmov8x	*freg*$_{rs1}$, *freg*$_{rs2}$, *freg*$_{rd}$
FPSELMOV16X	0 1001 0110_2	レジスタから 16 ビットの値を選択して移動	✓	✓	fpselmov16x	*freg*$_{rs1}$, *freg*$_{rs2}$, *freg*$_{rd}$
FPSELMOV32X	0 1001 0111_2	レジスタから 32 ビットの値を選択して移動	✓	✓	fpselmov32x	*freg*$_{rs1}$, *freg*$_{rs2}$, *freg*$_{rd}$



動作説明

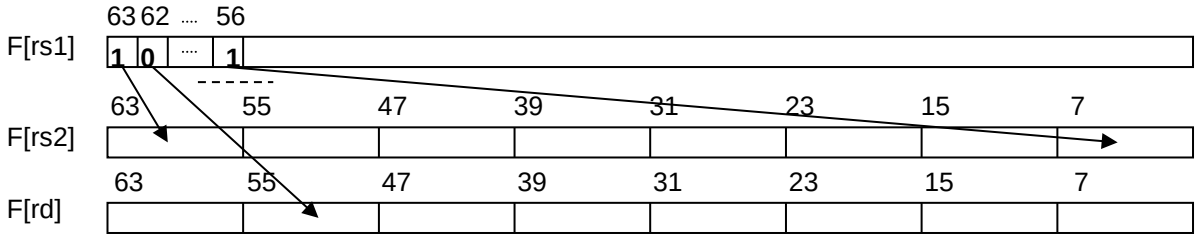
Fd[rs1]の最上位から n 番目のビットにより、Fd[rs2]か Fd[rd]の n 番目の要素を選択し、その値を Fd[rd]に格納する。Fd[rs1]のビット(63 – n)が 1 ならば Fd[rs2]、Fd[rs1]の 63 – n ビットが 0 ならば Fd[rd]を選択する。

Fd[rs1]の各ビットに対応する、Fd[rs2]と Fd[rd]の要素を以下の表に示す。

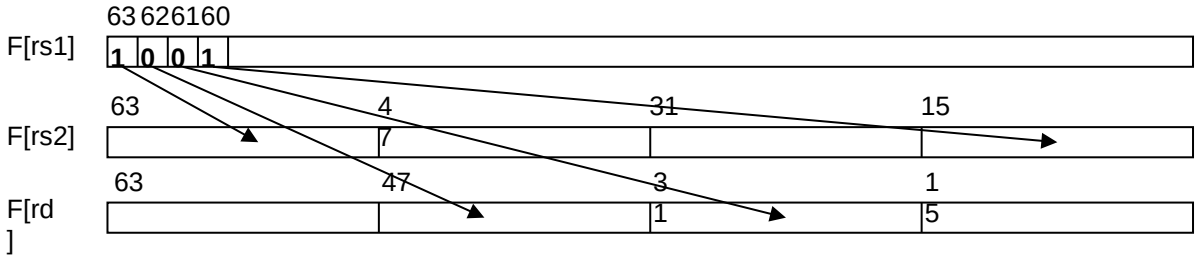
Note FPSELMOV8X の Fd[rs1]<55:0>、FPSELMOV16X の Fd[rs1]<59:0>、FPSELMOV32X の Fd[rs1]<61:0>に値を書き込んでも無視され、動作には影響を与えない。

	F[rs1] 63bit	F[rs1] 62bit	F[rs1] 61bit	F[rs1] 60bit	F[rs1] 59bit	F[rs1] 58bit	F[rs1] 57bit	F[rs1] 56bit
FPSELMOV8X 対応要素	<63:56>	<55:48>	<47:40>	<39:32>	<31:24>	<23:16>	<15:8>	<7:0>
FPSELMOV16X 対応要素	<63:48>	<47:32>	<31:16>	<15:0>				
FPSELMOV32X 対応要素	<63:32>	<31:0>						

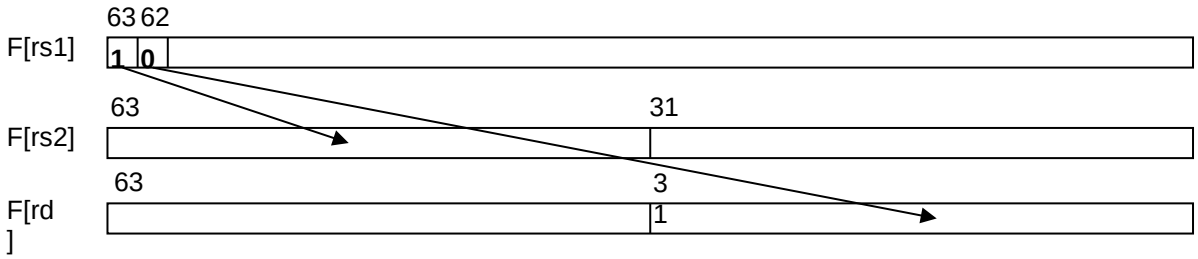
FPSELMOV8X の動作例



FPSELMOV16X の動作例



FPSELMOV32X の動作例



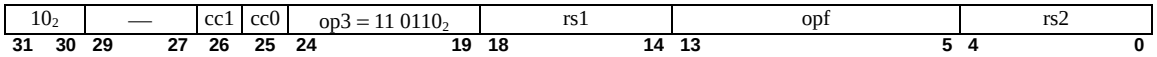
Note 64bit の FPSELMOV 命令は、FSELMOVd 命令(228 ページ参照)と同じ動作であるため定義しない。但し、FSELMOVd 命令は FSR のフィールドを更新するので注意が必要である。

FPSELMOV{8|16|32}X 命令は FSR のどのフィールドも更新しない。

例外	検出条件
fp_disabled	PSTATE.pef = 0 または FPRS.fef = 0
illegal_action	XAR.v = 1 かつ、下記のいずれかが成立している場合 • XAR.urs3 ≠ 0 • XAR.simd = 1 かつ XAR.urs1<2> ≠ 0 • XAR.simd = 1 かつ XAR.urs2<2> ≠ 0 • XAR.simd = 1 かつ XAR.urd<2> ≠ 0

7.134. 64-bit Integer Compare on Floating-Point Register

命令	opf	操作	HPC-ACE2		アセンブリ言語表記	
			Regs.	SIMD		
FPCMP64X	$1\ 0000\ 0100_2$	符号あり 64 ビット整数の比較	✓		fpcmp64x	%fccn, *freg*$_{rs1}$, *freg*$_{rs2}$
FPCMPU64X	$1\ 0000\ 0101_2$	符号なし 64 ビット整数の比較	✓		fpcmpu64x	%fccn, *freg*$_{rs1}$, *freg*$_{rs2}$



cc1	cc0	条件コード
0	0	fcc0
0	1	fcc1
1	0	fcc2
1	1	fcc3

動作説明

Fd[rs1]と Fd[rs2]を 64 ビットの整数として比較し、命令で指定した浮動小数点条件コードフィールド FSR.fccn に格納する。

入力値の関係	%fccn 出力
F[rs1] = F[rs2]	0
F[rs1] < F[rs2]	1
F[rs1] > F[rs2]	2
	3 出力されない

Programming Note FPCMP{64|U64}X 命令の実行で、FSR.cexc と FSR.aexc の値は変化しない。また、*fp_exception_other* 例外は発生しない

例外	検出条件
fp_disabled	PSTATE.pef = 0 または FPRS.fef = 0
Illegal_instruction	iw<29:27> ≠ 000_2
illegal_action	XAR.v = 1 かつ、下記のいずれかが成立している場合 • XAR.simd = 1 • XAR.urs3 ≠ 0 • XAR.urd ≠ 0

8. IEEE Std. 754-1985 Requirements for SPARC-V9

8.1. Floating-Point Nonstandard Mode

この節では、*fp_exception_other* 例外を FSR.ftt = *unfinished_FPop* で通知する境界条件と、SPARC64™ XIfx の IEEE 754-1985 非準拠モードの動作について説明する。前者は SPARC64™ XIfx では IEEE 準拠モード（FSR.ns = 0）のときのみ起こる例外である。

SPARC64™ XIfx の浮動小数点演算器は一定範囲の数のみを扱うことができる。演算の入力になる数や中間結果から、演算結果がその範囲から外れるだろうと予想される場合、SPARC64™ XIfx は演算を中断し *fp_exception_other* 例外を FSR.ftt = 02_{16}（*unfinished_FPop*）で通知し、後の処理をソフトウェアに委ねる。そしてエミュレーションルーチンは、IEEE 754-1985 に準拠する形で演算を完了させる（impl. dep. #3）。

SPARC64™ XIfx は IEEE 754-1985 非準拠モードで動作することもできる。これは FSR.ns = 1 により指示する（ページ 24 参照）。FSR.ns の値により、SPARC64™ XIfx の動作は変わってくる。

8.1.1. *fp_exception_other* Exception（ftt = *unfinished_FPop*）

SPARC64™ XIfx のほとんどの浮動小数点演算命令は、*fp_exception_other* 例外を FSR.ftt = *unfinished_FPop* で通知する可能性がある（詳細は各命令の仕様を参照）。例外発生の境界条件は以下の通りである。

1) 入力オペランドの 1 つが非正規化数で、それ以外がゼロでない正規化数（NaN と無限大を除く）のとき、*fp_exception_other* 例外が *unfinished_FPop* で通知される。ただし、結果がゼロかオーバーフローのときはこの限りではない。

2) すべての入力オペランドが非正規化数で、結果がゼロかオーバーフロー以外のとき、*fp_exception_other* 例外が *unfinished_FPop* で通知される。

3) すべての入力オペランドが正規化数、丸める前の演算結果が非正規化数で、FSR.tem.ufm = 0、かつ結果がゼロでない場合、*fp_exception_other* 例外が *unfinished_FPop* で通知される。

結果が定数、たとえばゼロや無限大になると予想され、そのための演算が簡単な場合、*unfinished_FPop* を通知せずに演算を行う。

境界条件検出のための、結果の指数部を求める式を表 8-1 に示す。ここで Er は丸め前の計算値の指数部（バイアス込み）で、入力データの指数部（esrc1, esrc2）だけから計算される。

表 8-1　*unfinished_FPop* 例外検出のための結果の指数部の計算式

処理	計算式
`fmuls, fmulds`	Er = esrc1 + esrc2 – 126

fmuld	Er = esrc1 + esrc2 – 1022
fdivs	Er = esrc1 – esrc2 + 126
fdivd	Er = esrc1 – esrc2 + 1022

esrc1, esrc2 はバイアス込みの入力データの指数部である。非正規化数の場合、この値は 0 である。

Er から eres が求められる。eres は仮数部の桁合わせ後、丸め前の指数部である。すなわち、演算後の仮数部を暗黙の 1 が小数点の左にくるまで右または左にシフトし、そのシフト量を Er に加算または減算する操作後の値である。表 8-2

表 8-2 に、全浮動小数点演算命令の *unfinished_FPop* 検出条件をまとめた。

表 8-2 *unfinished_FPop* 検出条件

処理	検出条件
FdTOs	-25 < eres < 1 かつ FSR.tem.ufm = 0 のとき。
FsTOd	第二オペランド（rs2）が非正規数のとき。
FADD{s|d|ds}, FSUB{s|d|ds}, FNADD{s|d}, FNSUB{s|d}	1) 入力のひとつが非正規化数、もうひとつがゼロ以外の正規化数（無限大と NaN を除く）[xxx]のとき。 2) 入力が両方とも非正規化数。 3) 入力が両方ともゼロ以外の正規化数（無限大と NaN を除く）で、eres < 1 かつ FSR.tem.ufm = 0 のとき。
FMUL{s|d|ds}, FNMUL{s|d}	1) 入力のひとつが非正規化数、もうひとつがゼロ以外の正規化数（無限大と NaN を除く）のとき、 単精度演算： -25 < Er 倍精度演算： -54 < Er 2) 入力が両方ともゼロ以外の正規化数（無限大と NaN を除く）で FSR.tem.ufm = 0 のとき、 単精度演算： -25 < eres < 1 倍精度演算： -54 < eres < 1
F{|N}sMULd	1) 入力のひとつが非正規化数、もうひとつがゼロ以外の正規化数（無限大と NaN を除く）のとき。 2) 入力が両方とも非正規化数のとき。
FDIV{s|d}	1) 被除数（rs1）がゼロ以外の正規化数（無限大と NaN を除く）、除数（rs2）が非正規化数のとき、 単精度演算： Er < 255 倍精度演算： Er < 2047 2) 被除数（rs1）が非正規化数、除数（rs2）がゼロ以外の正規化数（無限大と NaN を除く）のとき、 単精度演算： -25 < Er 倍精度演算： -54 < Er 3) 被除数、除数ともに非正規化数のとき。 4) 被除数、除数ともゼロ以外の正規化数（無限大と NaN を除く）で FSR.tem.ufm = 0 のとき、 単精度演算： -25 < eres < 1 倍精度演算： -54 < eres < 1
FSQRT{s|d}	入力（rs2）が正のゼロでない非正規化数のとき。

[xxx] 入力が非正規化数とゼロのときは、IEEE754-1985 で規定されている通りの結果を生成する。

FMADD{s|d}, FMSUB{s|d}, FNMADD{s|d}, FNMSUB{s|d}, FMADD{1|2}ds, FMSUM{1|2}ds, FNMADD{1|2}ds, FNMSUM{1|2}ds	乗算部分は FMUL{s|d}と同じ。加算部分は FADD{s|d}と同じ。
FTRIMADDd	乗算部分は FMUL{s|d}と同じ。加算部分では起きない。
FTRISMULd	rs1 がゼロ以外の正規化数（無限大と NaN を除く）で FSR.tem.ufm = 0 のとき、 -54 < eres < 1
FRCPA{s|d}	入力が非正規化数のとき。
FRSQRTA{s|d}	入力が正のゼロでない非正規化数のとき。

Conditions for a Zero Result

表 8-3 の条件が成立するとき、SPARC64™ XIfx は演算結果として、FSR.rd に応じて 0 または最小の非正規化数を返す。この値を pessimistic zero と呼ぶ。

表 8-3 Pessimistic Zero 発生条件

処理	条件		
	オペランドの一つが非正規化数 [xxxi]	両方のオペランドが非正規化数	両方が正規化数 [xxxii]
FdTOs	常に	—	—
FMUL{s|d|ds}, FNMUL{s|d}	単精度　Er ≤ -25 倍精度　Er ≤ -54	常に	単精度　eres ≤ -25 倍精度　eres ≤ -54
F{N}M{ADD|SUB}{s|d}, F{N}M{ADD|SUB}{1|2}ds	F[rs3] = 0、かつ以下の条件を満たすとき 単精度　Er ≤ -25 倍精度　Er ≤ -54	F[rs3] = 0 のとき、常に	F[rs3] = 0、かつ以下の条件を満たすとき 単精度　eres ≤ -25 倍精度　eres ≤ -54
FDIV{s|d}	単精度　Er ≤ -25 倍精度　Er ≤ -54	起こらない	単精度　eres ≤ -25 倍精度　eres ≤ -54
FTRIMADDd	Fd[rs2]<63> = 0 か つ index = 7 かつ Er ≤ -54	Fd[rs2]<63> = 0 か つ index = 7 のとき、常に	Fd[rs2]<63> = 0 か つ index = 7 かつ eres ≤ -54
FTRISMULd	Fd[rs1]が非正規化数の場合は常に	—	Fd[rs1]が正規化数 か つ eres ≤ -54

Conditions for an Overflow Result

表 8-4 の条件が成立するとき、SPARC64™ XIfx は演算でオーバーフローが起きたものとみなす。

[xxxi] 両方ともゼロ、NaN、無限大以外。
[xxxii] 両方がゼロでもよいが、どちらも無限大、NaN ではない。

表 8-4 Pessimistic Overflow 発生条件

処理	条件
FDIVs	除数（rs2）が非正規化数で Er ≥ 255 のとき。
FDIVd	除数（rs2）が非正規化数で Er ≥ 2047 のとき。

8.1.2. Behavior when FSR.ns = 1

Compatibility Note UA2011 の 8.4 章では、FSR.ns の値に関わらず、常に IEEE Std. 754 に従って演算が行われる命令について記載してある（FADD, FDIV, FMUL など）。しかしながら、SPARC64™ XIfx においては、これらの命令は FSR.ns = 1 の時は IEEE Std.754 に従わず、演算の処理が変わる (24 ページ参照)。

FSR.ns = 1（IEEE 非準拠モード）の場合、SPARC64™ XIfx は入力あるいは演算結果が非正規化数になった場合、それをゼロに置き換えて処理を続けようとする。この置き換えにより動作がどのように変わるかを以下に示す。

- 入力の一つが非正規化数で、すべてがゼロ、NaN、無限大以外のとき、非正規化数は同符号のゼロで置き換えられて演算が行われる。演算後、FSR.cexc.nxc = 1 がセットされる。ただし、以下の条件に該当する場合は FSR.cexc.nxc = 0 になる。
  - 演算が FDIV{s|d}で、ゼロ除算か無効演算が検出されたとき
  - 演算が FSQRT{s|d}で、無効演算が検出されたとき
  - 演算が FRPCA{s|d}, FRSQRTA{s|d}のとき

  FSR.cexc.nxc = 1 かつ FSR.tem.nxm = 1 のとき、*fp_exception_ieee_754* 例外が通知される。
- 丸める前の演算結果が非正規化数のときは、同符号のゼロで置き換えられる。
  FSR.tem.ufm = 1 ならば FSR.cexc.ufc = 1 が、FSR.tem.ufm = 0 で FSR.tem.nxm = 1 なら FSR.cexc.nxc = 1 がセットされ、*fp_exception_ieee_754* 例外が通知される。
  FSR.tem.ufm = 0 かつ FSR.tem.nxm = 0 のときは、FSR.cexc.ufc = 1, FSR.cexc.nxc = 1 がセットされる。

上で述べたとおり、FSR.ns = 1 のとき SPARC64™ XIfx は *unfinished_FPop* 例外を起こさないし、演算結果として非正規化数を返すこともない。

表 8-5 は、表 8-2 であげた浮動小数点演算命令[xxxiii]について、入力と結果の数の種類と FSR.tem によりどの例外が通知されるかを、FSR.ns = 0, 1 それぞれの場合についてまとめたものである。条件にしたがって表を左から右へたどると、Result の列に起こりうる例外と例外がマスクされている場合の演算結果が得られる。

FSR.ns = 1 で入力が非正規化数の場合は表 8-6 を参照。表 8-5 で Result が網掛けの部分は、IEEE754-1985 に準拠していることを示す。

Note 表 8-5、表 8-6 を通じて、英小文字の例外条件、nx, uf, of, dv, nv は、FSR.tem のそれぞれに対応するビットが 0 のため *fp_exception_ieee_754* 例外が通知されないことを示す。大文字の NX, UF, OF, DV, NV は *fp_exception_ieee_754* 例外が通知されることを示す。

[xxxiii] FTRISMULd の rs2 は浮動小数点ではないので、Source Denormal の対象外。

表 8-5 浮動小数点例外の発生条件と結果

FSR.ns	入力が非正規化数 [xxxiv]	結果が非正規化数 [xxxv]	Zero Result	Overflow Result	UFM	OFM	NXM	結果
0	No	Yes	Yes	—	1	—	—	UF
					0	—	1	NX
							0	uf + nx, 演算結果は丸め前と同符号のゼロか Dmin[xxxvi]
			No	—	1	—	—	UF
					0	—	—	*unfinished_FPop*[xxxvii]
		No	—	—	—	—	—	IEEE754-1985 準拠
	Yes	—	Yes	—	1	—	—	UF
					0	—	1	NX
							0	uf + nx, 演算結果は丸め前と同符号のゼロか Dmin
			No	Yes	—	1	—	OF
					—	0	1	NX
							0	of + nx, 演算結果は丸め前と同符号の無限大か Nmax[xxxviii]
				No	—	—	—	*unfinished_FPop*
1	No	Yes	—	—	1	—	—	UF
					0	—	1	NX
							0	uf + nx, 演算結果は丸め前と同符号のゼロ
		No	—	—	—	—	—	IEEE754-1985 準拠
	Yes	—	—	—	—	—	—	表 8-6

表 8-6 は、表 8-2 であげた浮動小数点演算命令について、FSR.ns = 1 で入力が非正規化数のときの SPARC64™ XIfx の動作をまとめたものである。表 8-6 で Result が網掛けの部分は、IEEE754-1985 に準拠していることを示す。

表 8-6 FSR.ns = 1 における非正規化数の演算

命令	入力値			FSR.tem				結果
	op1	op2	op3	UFM	NXM	DVM	NVM	

[xxxiv] 入力のひとつが非正規化数。他は正規化数（0, NaN, 無限大を除く）でも非正規化数でもよい。
[xxxv] 丸め前の演算結果が非正規化数になった場合。
[xxxvi] Dmin = 最小の非正規化数。
[xxxvii] 演算が、FADD{s|d}か FSUB{s|d}で入力がゼロと非正規化数の場合、*unfinished_FPop* 例外を通知せず、IEEE754-1985 に準拠した演算を行う。
[xxxviii] Nmax = 最大の正規化数。

FsTOd	—	Denorm	—	—	1	—	—	NX
					0	—	—	nx, 丸め前と同符号のゼロ
FdTOs	—	Denorm	—	1	—	—	—	UF
				0	1	—	—	NX
					0	—	—	uf + nx, 丸め前と同符号のゼロ
FADD{s|d|ds}, FSUB{s|d|ds}, FNADD{s|d}, FNSUB{s|d}	Denorm	Normal	—	—	1	—	—	NX
					0	—	—	nx, op2
	Normal	Denorm	—		1	—	—	NX
					0	—	—	nx, op1
	Denorm	Denorm	—		1	—	—	NX
					0	—	—	nx, 丸め前と同符号のゼロ
FMUL{s|d|ds}, FsMULd, FNMUL{s|d}, FNsMULd	Denorm	—	—	—	1	—	—	NX
					0	—	—	nx, 丸め前と同符号のゼロ
	—	Denorm	—		1	—	—	NX
					0	—	—	nx, 丸め前と同符号のゼロ
FDIV{s|d}	Denorm	Normal	—	—	1	—	—	NX
					0	—	—	nx, 丸め前と同符号のゼロ
	Normal	Denorm	—		—	1	—	DZ
						0	—	dz, 丸め前と同符号の無限大
	Denorm	Denorm	—		—	—	1	NV
							0	nv, dNaN[xxxix]
FSQRT{s|d}	—	Denorm and op2 > 0	—	—	1	—	—	NX
					0	—	—	nx, ゼロ
		Denorm and op2 < 0	—		—	—	1	NV
							0	nv, dNaN[xxxix]
FMADD{s|d}, FMSUB{s|d},	Denorm	—	Normal	—	1	—	—	NX
					0	—	—	nx, op3

[xxxix] 単精度の dNaN は $7FFFFFFF_{16}$、倍精度の dNaN は $7FFFFFFFFFFFFFFF_{16}$。

FNMADD{s|d}, FNMSUB{s|d}, FMADD{1|2}ds, FMSUB{1|2}ds, FNMADD{1|2}ds, FNMSUB{1|2}ds, FTRIMADDd[xl]			Denorm	—	1	—	—	NX
					0	—	—	nx, 丸め前と同符号のゼロ
	—	Denorm	Normal	—	1	—	—	NX
					0	—	—	nx, op3
			Denorm	—	1	—	—	NX
					0	—	—	nx, 丸め前と同符号のゼロ
	Normal	Normal	Denorm	—	1	—	—	NX
					0	—	—	nx, op1 × op2[xli]
FTRISMULd	Denorm	—	—	—	1	—	—	NX
					0	—	—	nx, op2<0>を符号とするゼロ
FRCPA{s|d}	—	Denorm	—	—	—	1	—	DZ
						0	—	dz, 丸め前と同符号の無限大
FRSQRTA{s|d}	—	Denorm	—	—	—	1	—	DZ
						0	—	dz, 丸め前と同符号の無限大

8.2. Floating-Point Exception Disable

SPARC64™ XIfx で追加された Floating-Point Exception Disable(fed)は浮動小数点演算例外の検出をマスクする。

このモードを設定することで(XASR.fed = 1)、演算結果は IEEE 754-1985 非準拠モードとして出力され、Nonstandard Mode(313 ページ)と同一の動作となる。また浮動小数点演算に関連する例外(*fp_exception_ieee_754* と *fp_exception_other* (ftt=*unfinished_FPop*)及び一部 *illegal_instruction*)を検出しない。詳細は、XASR.fed(38 ページ)及び FSR(23 ページ)参照。

[xl] op3 は演算器内のテーブルから引いてくる。常に正規化数。
[xli] op1 × op2 が Denorm のときは、op1 × op2 と同符号のゼロ。

9. Memory Models

SPARC64™ VIIIfx Extensions Chapter8 参照。

Compatibility Note SPARC64™ XIfx では PSTATE.MM は削除されている。SPARC64™ VIIIfx と同様にメモリモデルは Total Store Order 仕様に準拠する。

9.1.1. Coherence Domains

Compatibility Note UA2011 仕様ではキャッシャブル空間であるなしに関わらず TSO 遵守。

10. Address Space Identifiers

10.1. ASI Assignment

10.1.1. Supported ASIs

表 10-1　ASI 一覧で使われる表記

列	記号	意味
タイプ	Trans.	特権レベルと MMU 設定により変換モードが決まる
	Real	アドレスを RA として扱う
	non-T	MMU による変換を行わない。VA ウォッチポイント対象外
共有 (non-T のみ)	Chip	レジスタを CPU チップ全体で共有する
	CMG	レジスタをコアメモリグループで共有する
	Core	レジスタをコア内 VCPU で共有する (SPARC64™ XIfx では 1Core=1VCPU であるため、VCPU と読み替えてもよい)
	VCPU	VCPU ごとに独立したレジスタを持つ

表 10-2　ASI 一覧

ASI	VA	ASI 名	アクセス	タイプ	共有	ページ
80_{16}	—	ASI_PRIMARY (ASI_P)	RW	Trans.	—	
81_{16}	—	ASI_SECONDARY (ASI_S)	RW	Trans.	—	
82_{16}	—	ASI_PRIMARY_NO_FAULT (ASI_PNF)	RO	Trans.	—	
83_{16}	—	ASI_SECONDARY_NO_FAULT (ASI_SNF)	RO	Trans.	—	
84_{16} – 87_{16}	—	—	—	—	—	
88_{16}	—	ASI_PRIMARY_LITTLE (ASI_PL)	RW	Trans.	—	
89_{16}	—	ASI_SECONDARY_LITTLE (ASI_SL)	RW	Trans.	—	
$8A_{16}$	—	ASI_PRIMARY_NO_FAULT_LITTLE (ASI_PNFL)	RO	Trans.	—	
$8B_{16}$	—	ASI_SECONDARY_NO_FAULT_LITTLE (ASI_SNFL)	RO	Trans.	—	
$8C_{16}$ – BF_{16}	—	—	—	—	—	

ASI	VA	ASI 名	アクセス	タイプ	共有	ページ
$C0_{16}$	—	ASI_PST8_PRIMARY (ASI_PST8_P)	WO	Trans.	—	325
$C1_{16}$	—	ASI_PST8_SECONDARY (ASI_PST8_S)	WO	Trans.	—	325
$C2_{16}$	—	ASI_PST16_PRIMARY (ASI_PST16_P)	WO	Trans.	—	325
$C3_{16}$	—	ASI_PST16_SECONDARY (ASI_PST16_S)	WO	Trans.	—	325
$C4_{16}$	—	ASI_PST32_PRIMARY (ASI_PST32_P)	WO	Trans.	—	325
$C5_{16}$	—	ASI_PST32_SECONDARY (ASI_PST32_S)	WO	Trans.	—	325
$C6_{16}$ – $C7_{16}$	—	—	—	—	—	
$C8_{16}$	—	ASI_PST8_PRIMARY_LITTLE (ASI_PST8_PL)	WO	Trans.	—	325
$C9_{16}$	—	ASI_PST8_SECONDARY_LITTLE (ASI_PST8_SL)	WO	Trans.	—	325
CA_{16}	—	ASI_PST16_PRIMARY_LITTLE (ASI_PST16_PL)	WO	Trans.	—	325
CB_{16}	—	ASI_PST16_SECONDARY_LITTLE (ASI_PST16_SL)	WO	Trans.	—	325
CC_{16}	—	ASI_PST32_PRIMARY_LITTLE (ASI_PST32_PL)	WO	Trans.	—	325
CD_{16}	—	ASI_PST32_SECONDARY_LITTLE (ASI_PST32_SL)	WO	Trans.	—	325
CE_{16} – CF_{16}	—	—	—	—	—	
$D0_{16}$	—	ASI_FL8_PRIMARY (ASI_FL8_P)	RW	Trans.	—	325
$D1_{16}$	—	ASI_FL8_SECONDARY (ASI_FL8_S)	RW	Trans.	—	325
$D2_{16}$	—	ASI_FL16_PRIMARY (ASI_FL16_P)	RW	Trans.	—	325
$D3_{16}$	—	ASI_FL16_SECONDARY (ASI_FL16_S)	RW	Trans.	—	325
$D4_{16}$ – $D7_{16}$	—	—	—	—	—	
$D8_{16}$	—	ASI_FL8_PRIMARY_LITTLE (ASI_FL8_PL)	RW	Trans.	—	325
$D9_{16}$	—	ASI_FL8_SECONDARY_LITTLE (ASI_FL8_SL)	RW	Trans.	—	325
DA_{16}	—	ASI_FL16_PRIMARY_LITTLE (ASI_FL16_PL)	RW	Trans.	—	325
DB_{16}	—	ASI_FL16_SECONDARY_LITTLE (ASI_FL16_SL)	RW	Trans.	—	325
DC_{16} – DF_{16}	—	—	—	—	—	
$E0_{16}$	—	ASI_BLOCK_COMMIT_PRIMARY (ASI_BLK_COMMIT_P)	WO	Trans.	—	325
$E1_{16}$	—	ASI_BLOCK_COMMIT_SECONDARY (ASI_BLK_COMMIT_S)	WO	Trans.	—	325
$E2_{16}$	—	ASI_TWINX_P/ASI_STBI_P	RW	Trans.	—	
$E3_{16}$	—	ASI_TWINX_S/ASI_STBI_S	RW	Trans.	—	

ASI	VA	ASI 名	アクセス	タイプ	共有	ページ
$E4_{16}$ – $E6_{16}$	—	—	—	—	—	
$E7_{16}$	00_{16}	ASI_VSCCR	RW	non-T	VCPU	406
	200_{16}	ASI_HWPF_CTRL	RW	non-T	VCPU	412
$E8_{16}$ – $E9_{16}$	—	—	—	—	—	
EA_{16}	—	ASI_TWINX_PL/ASI_STBI_PL	RW	Trans.	—	254, 284
EB_{16}	—	ASI_TWINX_SL/ASI_STBI_SL	RW	Trans.	—	254, 284
EC_{16} – EE_{16}	—	—	—	—	—	
EF_{16}	00_{16} – 38_{16}	ASI_LBSY, ASI_BST	RW	non-T	VCPU	394
$F0_{16}$	—	ASI_BLOCK_PRIMARY (ASI_BLK_P)	RW	Trans.	—	325
$F1_{16}$	—	ASI_BLOCK_SECONDARY (ASI_BLK_S)	RW	Trans.	—	325
$F2_{16}$	—	ASI_XFILL_PRIMARY (ASI_XFILL_P)	WO	Trans.	—	156
$F3_{16}$	—	ASI_XFILL_SECONDARY (ASI_XFILL_S)	WO	Trans.	—	156
$F4_{16}$	—	ASI_XFILL256_PRIMARY (ASI_XFILL256_P)	WO	Trans.	—	156
$F5_{16}$	—	ASI_XFILL256_SECONDARY (ASI_XFILL256_S)	WO	Trans.	—	156
$F4_{16}$ – $F7_{16}$	—	—	—	—	—	
$F8_{16}$	—	ASI_BLOCK_PRIMARY_LITTLE (ASI_BLK_PL)	RW	Trans.	—	325
$F9_{16}$	—	ASI_BLOCK_SECONDARY_LITTLE (ASI_BLK_SL)	RW	Trans.	—	325
FA_{16} – FF_{16}	—	—	—	—	—	

10.1.2. ASI アクセス例外

10.1.2.1. ASIと命令の組み合わせ

SPARC64™ XIfx では、未定義の ASI や無効な命令と ASI の組み合わせにより起こる例外が、JPS1 Commonality の定義と一部異なる。この節では SPARC64™ XIfx での定義を、実際に通知される優先順位に沿って説明する。

1. LDBLOCKF, STBLOCKF, STPARTIALF 命令では *illegal_instruction* が通知される場合がある。詳細は各命令の定義を参照。LDTWA, STTWA 命令の rd に奇数番号のレジスタを指定した場合も、*illegal_instruction* が通知される。
2. 命令によって決まるアラインメント条件が検査され、違反していると *mem_address_not_aligned*, **_mem_address_not_aligned* が通知される。
   a) LDBLOCKF, STBLOCKF 命令は 64 バイトアラインメントを要求する命令なので、64 バイト境界にないアドレスをアクセスすると *mem_address_not_aligned* が通知される。*LDDF_mem_address_not_aligned*, *STDF_mem_address_not_aligned* は通知されない。
   だたし、LDDFA 命令でブロックコミットストア ASI(ASI 番号 $E0_{16}$, $E1_{16}$)を指定した場合は、この項には当てはまらない。
   b) 16 ビット LDSHORTF, STSHORTF 命令(ASI 番号 $D0_{16}$, $D1_{16}$, $D8_{16}$, $D9_{16}$)は 2 バイトアラインメントを要求する命令なので、2 バイト境界にないアドレスをアクセスす

ると *mem_address_not_aligned* が通知される。*LDDF_mem_address_not_aligned*, *STDF_mem_address_not_aligned* は通知されない。

c) 8 ビット LDSHORTF, STSHORTF 命令(ASI 番号 $D2_{16}$, $D3_{16}$, DA_{16}, DB_{16})は 1 バイトアラインメントを要求する命令なので、アラインメント違反は起きない。

d) STPARTIALF 命令は 8 バイトアラインメントを要求する命令なので、8 バイト境界にないアドレスをアクセスすると、*mem_address_not_aligned* が通知される。*LDDF_mem_address_not_aligned*, *STDF_mem_address_not_aligned* は通知されない。
LDDFA 命令で Partial Store ASI(ASI 番号 $C0_{16}$–$C5_{16}$, $C8_{16}$–CD_{16})を指定した場合は、STPARTIALF 命令ではないので、この項には当てはまらない。

e) LDDFA, STDFA 命令で上記以外の ASI を指定し、8 バイト境界にない 4 バイト境界にアクセスすると、それぞれ、*LDDF_mem_address_not_aligned*, *STDF_mem_address_not_aligned* が通知される。

f) 上記以外のアラインメント違反には、*mem_address_not_aligned* が通知される。

3. ASI と命令の組み合わせが正しくない場合、*DAE_invalid_asi* が通知される。
ただし PREFETCHA 命令では *DAE_invalid_asi* が通知されず、NOP として処理される。

10.1.2.2. 未定義ASI空間

未定義 ASI 空間にアクセスすると、*privileged_action* または *DAE_invalid_asi* が通知される。通知される例外は、ASI 番号とアクセス時の特権モードによって決定される。表 10-3 に各特権モードにおける例外一覧を示す。ただし、未定義 ASI であっても、アラインメント違反が通知される場合は、*privileged_action, DAE_invalid_asi* は通知されない。

表 10-3 未定義 ASI 空間にアクセスしたときの例外

ASI	非特権
00_{16} – $2F_{16}$	*privileged_action*
30_{16} – $7F_{16}$	*privileged_action*
80_{16} – FF_{16}	*DAE_invalid_asi*

10.2. Special Memory Access ASI

この節では、Translating ASI, Real ASI について述べる。アドレス変換の情報は 13.2 アドレス変換（374 ページ）も参照。

ASIs $E2_{16}$, $E3_{16}$, EA_{16}, EB_{16} (Nonprivileged Load Integer Twin Extended Word)

ASI $E2_{16}$, $E3_{16}$, EA_{16}, EB_{16} は、(nonportable な) アトミック Load Integer Twin Extended Word (LDTXA) 命令を使用するために存在する。(Load Integer Twin Extended Word from Alternate Space （254 ページ参照）) これらの ASI は、アクセスするアドレス空間（Primary or Secondary）と、エンディアンにより複数定義される。

LDTXA でこれらの ASI が指定された際に、オペランドが 16 バイトアラインされていない場合には *mem_address_not_aligned* 例外が発生する。

10.2.2. Block Load and Store ASIs

ASI $E0_{16}$, $E1_{16}$, $F0_{16}$, $F1_{16}$, $F8_{16}$, $F9_{16}$ は、ブロックロード（LDBLOCKF）やブロックストア（STBLOCKF）として LDDFA や STDFA を使用するために存在する。(Block Load, Block Store（285 ページ参照）)

ブロックロード（ストア）として LDDFA（STDFA）が使用された際に、オペランドアドレスが 64 バイトアラインされていない場合には *mem_address_not_aligned* 例外が発生する。

ASI $E0_{16}$, $E1_{16}$ は、ブロックコミットストア命令のためだけに使用される（285 ページ参照）。ASI $E0_{16}$, $E1_{16}$ はいずれも LDDFA オペコードとして使用されるべきではない。LDDFA オペコードとして使用された場合は、その結果は LDDFA で定義（89 ページ参照)されたものになる。

10.2.3. Partial Store ASIs

ASI $C0_{16}$ – $C5_{16}$ と $C8_{16}$ – CD_{16} はパーシャルストア（STPARTIALF）として STDFA 命令を使用するために存在する。(Store Partial Floating-Point （287 ページ参照）)

パーシャルストアとして STDFA が使用された際は、オペランドアドレスが 8 バイトアラインされていない場合には *mem_address_not_aligned* 例外が発生する。ASI レジスタにパーシャルストア ASI を指定し、命令のビット i = 1 である場合には *illegal_instruction* が発生する。

ASI $C0_{16}$ – $C5_{16}$ と $C8_{16}$ – CD_{16} はパーシャルストア命令のみに定義されている。(287 ページ参照）いずれも LDDFA オペコードとして使用されるべきではない。LDDFA オペコードとして使用された場合は、その結果は LDDFA で定義（89 ページ参照）されたものになる。

10.2.4. Short Floating-Point Load and Store ASI

ASI $D0_{16}$ – $D3_{16}$ と $D8_{16}$ – DB_{16} は Short Floating-Point Load and Store 命令 LDDFA と STDFA のために存在する。(Load Short Floating-Point 82 ページ, Store Short Floating-Point 107 ページ参照)

ASI $D2_{16}$, $D3_{16}$, DA_{16}, DB_{16} が LDDFA (STDFA）とともに 16 ビット Short Floating-Point Load (Store）として使用され、そのオペランドアドレスが halfword アラインされていない場合、*mem_address_not_aligned* 例外が発生する。

これらの ASI が、別空間へのロード、ストア、アトミックロードストア、別空間へのプリフェッチ命令とともに使用されると、*DAE_invalid_asi* 例外が常に発生し、*mem_address_not_aligned* は発生しない。

10.3. Non-Faulting モード

SPARC64™ XIfx では、XASR.nf = 1 のとき指定した ASI 番号に関わらず、ロード命令が ASI の*_NO_FAULT*を指定したかのように振舞う Non-Faulting モードを追加する。

このモードは、TL = 0 の時のみ有効であり、TL > 0 では無効化される。特権レベルに関わらず TL = 0 では有効である。

ロード命令で ASI として*_NO_FAULT*が指定され、かつ XASR.nf = 1 の場合も互いに干渉することなく*_NO_FAULT*を指定したかのようにアクセスされる。Non-Faulting モードの影響範囲は表 10-2 のモードが Trans.のものである。

この Non-Faulting モードによるロード命令の振る舞いは、ASI の*_NO_FAULT*が指定されたときと互換であり、side_effect や non-fault only などの属性に対しても同様に作用する。

XASR.nf = 0 のとき、このモードは無効となり、従来のロードと互換の動作となる。

Non-Faulting モードの対象となるロード命令を表 10-4 に示す。表に示されていない、メモリからのロードの操作を含むが、アトミックにメモリへのストアが行われる可能性のある命令については、Non-Faulting モードの対象とならない。

表 10-4　Non-Faulting モードの対象となる命令

命令
LDUW{A}
LDUB{A}
LDUH{A}
LDTW{A}
LDTXA
LDSW{A}
LDSB{A}
LDSH{A}
LDX{A}
LDF{|UW|SW|IB|A}, LDFBC{|UW|SW|IB}, LDFST{|UW|SW|IB}, LDFID{|UW|SW}
LDDF{|DS|A}, LDDFBC, LDDFST, LDDFID
LDQF{A}
LDFSR, LDXFSR, LDXEFSR
LDBLOCKF, LDSHORTF

11. Performance Instrumentation

11.1. Overview

Performance counters comprise one “Performance Control Register (PCR) (ASR 16)” and multiple instances of “Performance Instrumentation Counter Register (PIC) (ASR 17)”.

SPARC64™ XIfx implements 4 PIC registers, which are selected by PCR.SC and accessed via ASR 17. Each PIC register contains two counters.

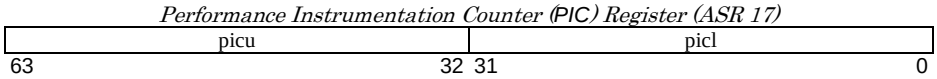
Performance Control Register (PCR) (ASR 16)

—	toe	—	ovf	ovro	ulro	—	nc	su	sl	—	sc	ht	ut	st	priv
63 56	55 48	47 40	39 32	31	30	29 27	26 24	23 16	15 8	7	6 4	3	2	1	0

Bits	Field	Access	Description
55:48	toe	RW	Controls whether an overflow exception is generated for performance counters. A write updates the field and a read returns the current settings. If toe<i> is 1 and the counter corresponding to ovf<i> overflows, ovf<i>=1 and a pic_overflow exception is generated. If toe<i> is 0 and the counter corresponding to ovf<i> overflows, ovf<i>=1 but a pic_overflow exception is not generated. When ovf<i>=1 and the value of toe<i> is changed to 1, a pic_overflow exception is not generated.
39:32	ovf	RW	Overflow Clear/Set/Status. A read by RDPCR returns the overflow status of the counters, and a write by WRPCR clears or sets the overflow status bits. The following figure shows the PIC counters corresponding to the OVF bits. A write of 0 to an OVF bit clears the overflow status of the corresponding counter. U3 L3 U2 L2 U1 L1 U0 L0 7 6 5 4 3 2 1 0
31	ovro	RW	Overflow Read-Only. A write to the PCR register with write data containing a value of ovro =0 updates the PCR.ovf field with the OVF write data. If the write data contains a value of ovro=1, the OVF write data is ignored and the PCR.ovf field is not updated. Reads of the PCR.ovro field return 0. The PCR.ovro field allows PCR to be updated without changing the overflow status. Hardware maintains the most recent state in PCR.ovf such that a subsequent read of the PCR returns the current overflow status.
30	ulro	RW	su/sl Read-Only. A write to the PCR register with write data containing a value of ulro = 0 updates the PCR.su and PCR.sl fields with the su/sl write data. If the write data contains a value of ulro = 1, the su/sl write data is ignored and the PCR.su and PCR.sl fields are not updated. Reads of the PCR.ulro field return 0.

			The PCR.ulro field allows the PIC pair selection field to be updated without changing the PCR.su and PCR.sl settings.
26:24	nc	RO	This read-only field indicates the number of PIC counter pairs.
23:16	su	RW	This field selects the event counted by PIC<63:32>. A write updates the setting, and a read returns the current setting.
15:8	sl	RW	This field selects the event counted by PIC<31:0>. A write updates the setting, and a read returns the current setting.
6:4	sc	RW	PIC Pair Selection. A write updates which PIC counter pair is selected, and a read returns the current selection. When a "1" is written to bit<6>, no counter pair is selected and a subsequent read returns "0"
3	ht	RO (priv) RO (user)	If ht = 1, events that occur while in hypervisor mode are counted. Writes to this field are ignored.
2	ut	RW	User Mode. When PSTATE.priv =0 and ut=1, events are counted.
1	st	RW	System Mode. When PSTATE.priv =1 and st=1, events are counted. If both PCR.ut and PCR.st are 1, all events are counted. If both PCR.ut and PCR.st are 0, counting is disabled. PCR.ut and PCR.st are global fields; that is, they apply to all PIC counter pairs.
0	priv	RW	Privileged. If PCR.priv = 1, executing a RDPCR, WRPCR, RDPIC, or WRPIC instruction in non-privileged mode (PSTATE.priv = 0) causes a privileged_action exception. If PCR.priv = 0, a non-privileged (PSTATE.priv = 0) attempt to update PCR.priv (that is, to write a value of 1) via a WRPCR instruction causes a privileged_action exception.

Performance Instrumentation Counter (PIC) Register (ASR 17)



Bits	Field	Access	Description
63:32	picu	RW	32-bit counter for the event selected by PCR.su.
31:0	picl	RW	32-bit counter for the event selected by PCR.sl.

11.1.1. Sample Pseudo-codes

11.1.1.1. Counter Clear/Set

The counter fields in the PIC registers are read/write registers. Writing zero clears a counter; writing any other value sets the counter to that value. The following pseudo-code clears all PIC registers (assuming privileged access):

```
/* Clear PICs without updating SL/SU values */
pic_init = 0x0;
pcr = rd_pcr();
pcr.ulro = 0x1;          /* don't update SU/SL on write    */
pcr.ovf = 0x0;           /* clear overflow bits            */
pcr.ut = 0x0;
pcr.st = 0x0;            /* disable counts                 */
pcr.ht = 0x0;            /* non- hypervisor mode           */
pcr.priv = 0x0;          /* privileged access              */
for (i=0; i<=pcr.nc; i++) {
/* select the PIC to be written */
   pcr.sc = i;
   wr_pcr(pcr);
   wr_pic(pic_init);     /* clear PIC[i]                   */
}
```

11.1.1.2. Select and Enable Counter Events

Counter events are selected using the PCR.sc and PCR.su/PCR.sl fields. The following pseudo-code selects events and enables counters (assuming privileged access):

```
pcr.ut = 0x0;            /* Disable user counts            */
pcr.st = 0x0;            /* Disable system counts also     */
pcr.ht = 0x0;            /* non- hypervisor mode           */
pcr.priv = 0x0;          /* privileged access              */
pcr.ulro = 0x0;          /* Make SU/SL writeable           */
pcr.ovro = 0x1;          /* Overflow is read-only          */
/* Select events without enabling counters */
for(i=0; i<=pcr.nc; i++) {
   pcr.sc = i;
   pcr.sl = select an event;
   pcr.su = select an event;
   wr_pcr(pcr);
}
/* Start counting */
pcr.ut = 0x1;
pcr.st = 0x1;
pcr.ulro = 0x1;          /* SU/SL is read-only             */
/* Clear overflow bits here if needed */
wr_pcr(pcr);
```

11.1.1.3. Stop Counter and Read

The following pseudo-code disables counters and reads their values (assuming privileged access):

```
pcr.ut = 0x0;            /* Disable user counts            */
pcr.st = 0x0;            /* Disable system counts, too     */
pcr.ht = 0x0;            /* non- hypervisor mode           */
pcr.priv = 0x0;          /* privileged access              */
pcr.ulro = 0x1;          /* Make SU/SL read-only           */
pcr.ovro = 0x1;          /* Overflow is read-only          */
for(i=0; i<=pcr.nc; i++) {
   pcr.sc = i;
   wr_pcr(pcr);
```

```
    pic = rd_pic();
    picl[i] = pic.picl;
    picu[i] = pic.picu;
}
```

11.2. Description of PA Events

The performance counter (PA) events can be divided into the following groups:

1. Instruction and trap statistics
2. MMU and L1 cache events
3. L2 cache events
4. Bus transaction events

There are 2 types of PA events that can be measured in SPARC64™ XIfx, standard and supplemental events.

Standard events on SPARC64™ XIfx have been verified for correct behavior; they are guaranteed to be compatible with future processors.

Supplemental events are primarily intended for debugging the hardware.

a) The behavior of supplemental events may not be fully verified. There is a possibility that some of these events may not behave as specified in this document.

b) The definition of these events may change without notice. Compatibility with future processors is not guaranteed.

Table 11-1,

Table 11-2, and Table 11-3 show PA events defined on SPARC64™ XIfx. Shaded events are supplemental events.

For details of each event, refer to the descriptions in the following sections. Unless otherwise indicated, speculative instructions are also counted by the PA events.

Table 11-1 PA Events and Encodings (1/3)

Encoding (bin)	Counter							
	pic u0	pic l0	pic u1	pic l1	pic u2	pic l2	pic u3	pic l3
000_0000	cycle_counts							
000_0001	instruction_counts							
000_0010	instruction_ flow_counts	*Reserved*			instruction_ flow_counts	d_move_wait	*Reserved*	
000_0011	iwr_empty	*Reserved*			iwr_empty	*Reserved*		
000_0100	*Reserved*							
000_0101	op_stv_wait							
000_0110	effective_instruction_counts							
000_0111	SIMD_load_store_ instructions	SIMD_floating_ instructions	SIMD_fma_ instructions	sxar1_ instructions	sxar2_ instructions	unpack_sxar1	unpack_sxar2	*Reserved*
000_1000	load_store_instructions							
000_1001	branch_instructions							
000_1010	floating_instructions							
000_1011	fma_instructions							
000_1100	prefetch_instructions							
000_1101	*Reserved*	ex_load_ instructions	ex_store_ instructions	fl_load_ instructions	fl_store_ instructions	SIMD_fl_load_ instructions	SIMD_fl_store_ instructions	*Reserved*
000_1110	*Reserved*							
000_1111	*Reserved*							
001_0000	*Reserved*							
001_0001	*Reserved*							
001_0010	rs1	flush_rs	*Reserved*					
001_0011	1iid_use	2iid_use	3iid_use	4iid_use	*Reserved*	sync_intlk	regwin_intlk	*Reserved*
001_0100	*Reserved*							
001_0101	*Reserved*	toq_rsbr_phantom	*Reserved*	flush_rs	*Reserved*		rs1	*Reserved*
001_0110	trap_all	*Reserved*	trap_int_level	trap_spill	trap_fill	trap_trap_inst	*Reserved*	*Reserved*
001_0111	*Reserved*	*Reserved*	*Reserved*					
001_1000	*Reserved*						op_stv_wait_ pfp_busy_swpf	op_stv_ wait_sxmiss
001_1001	*Reserved*							
001_1010	*Reserved*	*Reserved*	single_sxar_commit	*Reserved*				suspend_cycle
001_1011	rsf_pmmi	*Reserved*	op_stv_wait_nc_pend	0iid_use	flush_rs	*Reserved*		decode_all_intlk
001_1100	*Reserved*			*Reserved*	*Reserved*	*Reserved*	*Reserved*	*Reserved*
001_1101	op_stv_wait_ pfp_busy_ex	op_stv_wait_ l1d_cache_miss	op_stv_wait_ sxmiss_ex	op_stv_wait_ nc_pend	cse_window_ empty_sp_full	op_stv_wait_ pfp_busy	*Reserved*	
001_1110	cse_window_empty	eu_comp_wait	branch_comp_wait	0endop	op_stv_wait_ex	fl_comp_wait	1endop	2endop
001_1111	op_stv_wait_l1d_ cache_miss_ex	*Reserved*			3endop	*Reserved*	sleep_cycle	op_stv_wait_swpf

Table 11-2 PA Events and Encodings (2/3)

Encoding (bin)	counter							
	pic u0	pic l0	pic u1	pic l1	pic u2	pic l2	pic u3	pic l3
010_0000	ITLB_write	DTLB_write	uITLB_miss	uDTLB_miss	L1I_miss	L1D_miss	L1I_wait_all	L1D_wait_all
010_0001	*Reserved*							L1D_miss_qpf
010_0010	*Reserved*							
010_0011	L1I_thrashing	L1D_thrashing	*Reserved*			L1D_miss_dm	L1D_miss_pf	*Reserved*
010_0100	swpf_success_all	swpf_fail_all	*Reserved*		swpf_lbs_hit	*Reserved*		
010_0101	*Reserved*							
010_0110	*Reserved*							
010_0111	*Reserved*							
010_1000	*Reserved*			L1D_pipe0_valid	L1D_pipe1_valid	*Reserved*		
010_1001	*Reserved*							
010_1010	*Reserved*							
010_1011	*Reserved*							
010_1100	*Reserved*							
010_1101	*Reserved*							
010_1110	*Reserved*							
010_1111	*Reserved*							
011_0000	*Reserved*		L2_miss_dm	L2_miss_pf	L2_read_dm	L2_read_pf	L2_wb_dm	L2_wb_pf
011_0001	bi_counts	cpi_counts	cpb_counts	cpd_counts	cpu_mem_ read_counts	cpu_mem_ write_counts	IO_mem_ read_counts	IO_mem_ write_counts
011_0010	L2_miss_wait_ dm_bank0	L2_miss_wait_ pf_bank0	L2_miss_counts_ dm_bank0	L2_miss_counts_ pf_bank0	L2_miss_wait_ dm_bank1	L2_miss_wait_ pf_bank1	L2_miss_counts_ dm_bank1	L2_miss_counts_ pf_bank1
011_0011	L2_miss_counts_ dm_bank2	L2_miss_counts_ pf_bank2	L2_miss_wait_ dm_bank2	L2_miss_wait_ pf_bank2	L2_miss_counts_ dm_bank3	L2_miss_counts_ pf_bank3	L2_miss_wait_ dm_bank3	L2_miss_wait_ pf_bank3
011_0100	lost_pf_pfp_full	lost_pf_by_abort	IO_pst_counts	*Reserved*				
011_0101	*Reserved*							
011_0110	*Reserved*							
011_0111	*Reserved*							
011_1000	*Reserved*							
011_1001	*Reserved*							
011_1010	*Reserved*							
011_1011	*Reserved*							
011_1100	*Reserved*							
011_1101	*Reserved*							
011_1110	*Reserved*							
011_1111	Disabled(No PIC is counted up)							

Table 11-3 PA Events and Encodings (3/3)

Encoding (bin)	counter							
	pic u0	pic l0	pic u1	pic l1	pic u2	pic l2	pic u3	pic l3
100_0000	1FLOPS_ instructions	2FLOPS_ instructions	4FLOPS_ instructions	8FLOPS_ instructions	16FLOPS_ instructions	*Reserved*		
100_0001	4SIMD_load_ store_instructions	4SIMD_floating_ instructions	4SIMD_fma_ instructions	4SIMD_fl_load_ instructions	4SIMD_fl_store_ instructions	*Reserved*	4SIMD_load_ store_instructions	*Reserved*
100_0010	floating_DSP_ instructions	fma_DSP_ instructions	fl_move_ instructions	fl_movcc_r_sel_ instructions	fixed_point_ instructions	fepermd_ instructions	elememt_ summask_comp_ instructions	fecsld_ instructions
100_0011	frcpa_ instructions	frsqrta_ instructions	trigonometric_ exponential_ instructions	xma-inst	flogical_ instructions	fpcmp_ instructions	*Reserved*	
100_0100	indirect_load_ fl_instructions	*Reserved*	*Reserved*	*Reserved*	indirect_store_ fl_instructions	store_fl_condition _instructions	stfruw_ instructions	*Reserved*
100_0101	indirect_store_ fl_condition_ instructions	ldfuw_sw_ib_ instructions	stfuw_ instructions	lddfds_ instructions	stdfds_ instructions	indirect_prefetch_ instructions	*Reserved*	store_with_xfill_ instructions
100_0110	SIMD_ floating_DSP_ instructions	SIMD_ fma_DSP_ instructions	SIMD_ fl_move_ instructions	SIMD_ fselmov_ instructions	SIMD_ fixed_point_ instructions	SIMD_ fepermd_ instructions	SIMD_elememt_ summask_comp_ instructions	*Reserved*
100_0111	SIMD_ frcpa_ instructions	SIMD_ frsqrta_ instructions	SIMD_ trigonometric_ exponential_ instructions	SIMD_ xma_inst	SIMD_ flogical_ instructions	SIMD_ fpcmp_ instructions	*Reserved*	
100_1000	SIMD_ indirect_load_ fl_instructions	SIMD_ stride_load_ fl_instructions	SIMD_ broadcast_load_ fl_instructions	SIMD_ stride_store_ fl_instructions	SIMD_ indirect_store_ fl_instructions	SIMD_ store_fl_condition _instructions	SIMD_ stfruw_ instructions	SIMD_stride_ store_fl_condition _instructions
100_1001	SIMD_indirect_ store_fl_condition _instructions	SIMD_ ldfuw_sw_ib_ instructions	SIMD_ stfuw_ instructions	SIMD_ lddfds_ instructions	SIMD_ stdfds_ instructions	SIMD_ indirect_prefetch_ instructions	*Reserved*	
100_1010	4SIMD_ floating_DSP_ instructions	4SIMD_ fma_DSP_ instructions	4SIMD_ fl_move_ instructions	4SIMD_ fselmov_ instructions	4SIMD_ fixed_point_ instructions	4SIMD_ fepermd_ instructions	4SIMD_elememt_ summask_comp_ instructions	*Reserved*
100_1011	4SIMD_ frcpa_ instructions	4SIMD_ frsqrta_ instructions	4SIMD_ trigonometric_ exponential_ instructions	4SIMD_ xma_inst	4SIMD_ flogical_ instructions	4SIMD_ fpcmp_ instructions	*Reserved*	
100_1100	4SIMD_ indirect_load_ fl_instructions	4SIMD_ stride_load_ fl_instructions	4SIMD_ broadcast_load_ fl_instructions	4SIMD_ stride_store_ fl_instructions	4SIMD_ indirect_store_ fl_instructions	4SIMD_ store_fl_condition _instructions	4SIMD_ stfruw_ instructions	4SIMD_stride_ store_fl_condition _instructions
100_1101	4SIMD_ indirect_store_ fl_condition_ instructions	4SIMD_ ldfuw_sw_ib_ instructions	4SIMD_ stfuw_ instructions	4SIMD_ lddfds_ instructions	4SIMD_ stdfds_ instructions	4SIMD_ indirect_prefetch_ instructions	*Reserved*	
100_1110	XSIMD_load_ store_instructions	XSIMD_floating_ instructions	XSIMD_fma_ instructions	*Reserved*	XSIMD_fixed_ point_instructions	nonSIMD_XSIMD _indirect_prefetch _instructions	*Reserved*	
100_1111	*Reserved*							
111_1111	Disabled(No PIC is counted up)							

11.2.1. Instruction and Trap Statistics

Standard PA Events

1 cycle_counts

Counts the number of cycles when the performance counter is enabled. This counter is similar to the TICK register but can count user cycles and system cycles separately, based on the settings of PCR.ut and PCR.st.

2 instruction_counts (Non-Speculative)

Counts the number of committed instructions, including SXAR1 and SXAR2. SPARC64™ XIfx commits up to 4 instructions per cycle; however, this number normally does not include SXAR1 and SXAR2. That is, instruction_counts/cycle_counts can be greater than 4.

3 effective_instruction_counts (Non-Speculative)

Counts the number of committed instructions. SXAR1 and SXAR2 are not included. Instructions per cycle (IPC) can be derived from this event and cycle_counts.

IPC = effective_instruction_counts / cycle_counts

If effective_instruction_counts and cycle_counts are collected for user or system mode, the IPC in either user or system mode can be calculated.

4 load_store_instructions (Non-Speculative)

Counts the number of committed “non-SIMD load/store” instructions. Also counts atomic load-store instructions.

5 branch_instructions (Non-Speculative)

Counts the number of committed “branch” instructions. Also counts the CALL, JMPL, and RETURN instructions.

6 floating_instructions (Non-Speculative)

Counts the number of committed “non-SIMD floating point” instructions. The counted instructions are FPop1 (exclude F{ADD|SUB|MUL}{s|d}), FPop2 ,FSELMOV{s|d} and IMPDEP1 with opf<8:4> = 0x16 and 0x17.

7 fma_instructions (Non-Speculative)

Counts the number of committed “non-SIMD floating point multiply and add/sub” and “floating point trigonometric” instructions. The counted instructions are FM{ADD|SUB}{s|d}, FNM{ADD|SUB}{s|d}, and FTRIMADDd. Two operations are executed per instruction; the number of operations is obtained by multiplying by 2.

8 prefetch_instructions (Non-Speculative)

Counts the number of committed “prefetch” instructions. (exclude “indirect prefetch” instruction)

9 SIMD_load_store_instructions (Non-Speculative)

Counts the number of committed “SIMD load/store” instructions. The counted instructions are the same as load_store_instructions.

10 SIMD_floating_instructions (Non-Speculative)

Counts the number of committed “SIMD floating point” instructions. The counted instructions are the same as floating_instructions. Two operations are executed per instruction; the number of operations is obtained by multiplying by 2.

11 SIMD_fma_instructions (Non-Speculative)

Counts the number of committed “SIMD floating-point multiply and add/sub” and “SIMD floating point trigonometric” instructions. The counted instructions are the same as fma_instructions. Four operations are executed per instruction; the number of operations is obtained by multiplying by 4.

12 sxar1_instructions (Non-Speculative)

Counts the number of committed “SXAR1” instructions.

13 sxar2_instructions (Non-Speculative)

Counts the number of committed “SXAR2” instructions.

14 trap_all (Non-Speculative)

Counts the occurrences of all trap events.

16 trap_int_level (Non-Speculative)

Counts the occurrences of *interrupt_level_n*.

17 trap_spill (Non-Speculative)

Counts the occurrences of *spill_n_normal* and *spill_n_other*.

18 trap_fill (Non-Speculative)

Counts the occurrences of *fill_n_normal* and *fill_n_other*.

19 trap_trap_inst (Non-Speculative)

Counts the occurrences of trap_instruction.

Supplemental PA Events

23 4SIMD_load_store_instructions (Non-Speculative)

Counts the number of committed “4SIMD load/store” instructions. The counted instructions are the same as load_store_instructions.

24 4SIMD_floating_instructions (Non-Speculative)

Counts the number of committed “4SIMD floating point” instructions. The counted instructions are the same as floating_instructions. Four operations are executed per instruction; the number of operations is obtained by multiplying by 4.

25 4SIMD_fma_instructions (Non-Speculative)

Counts the number of committed “4SIMD floating point multiply and add/sub” and “4SIMD floating point trigonometric” instructions. The counted instructions are the same as fma_instructions. Eight operations are executed per instruction; the number of operations is obtained by multiplying by 8.

26 XSIMD_load_store_instructions (Non-Speculative)

Counts the number of committed “SIMD and 4SIMD load/store” instructions. The counted instructions are the same as load_store_instructions.

27 XSIMD_floating_instructions (Non-Speculative)

Counts the number of committed “SIMD and 4SIMD floating point” instructions. The counted instructions are the same as floating_instructions and floating_DSP_instructions (floating Dual Single Precision instructions).

28 XSIMD_fma_instructions (Non-Speculative)

Counts the number of committed “SIMD and 4SIMD floating point multiply and add/sub” and “SIMD and 4SIMD floating point trigonometric” instructions. The counted instructions are the same as fma_instructions and fma_DSP_instructions.

29 floating_DSP_instructions (Non-Speculative)

Counts the number of committed “non-SIMD floating-point Dual Single Precision” instructions. The counted instructions are `F{ADD|SUB|MUL}ds`. Two operations are executed per instruction; the number of operations is obtained by multiplying by 2.

30 fma_DSP_instructions (Non-Speculative)

Counts the number of committed “non-SIMD floating point multiply and_add/sub Dual Single Precision” instructions. The counted instructions are `FM{ADD|SUB}{1|2}ds` and `FNM{ADD|SUB}{1|2}ds`. Four operations are executed per instruction; the number of operations is obtained by multiplying by 4.

31 SIMD_floating_DSP_instructions (Non-Speculative)

Counts the number of committed “SIMD floating-point Dual Single Precision” instructions. The counted instructions are the same as floating_DSP_instructions. Four operations are executed per instruction; the number of operations is obtained by multiplying by 4.

32 SIMD_fma_DSP_instructions (Non-Speculative)

Counts the number of committed “SIMD floating-point multiply and add/sub Dual Single Precision” instructions. The counted instructions are the same as fma_DSP_instructions. Eight operations are executed per instruction; the number of operations is obtained by multiplying by 8.

33 4SIMD_floating_DSP_instructions (Non-Speculative)

Counts the number of committed “4SIMD floating point Dual Single Precision” instructions. The counted instructions are the same as floating_DSP_instructions. Eight operations are executed per instruction; the number of operations is obtained by multiplying by 8.

34 4SIMD_fma_DSP_instructions (Non-Speculative)

Counts the number of committed “4SIMD floating point multiply and_add/sub Dual Single Precision” instructions. The counted instructions are the same as fma_DSP_instructions. Sixteen operations are executed per instruction; the number of operations is obtained by multiplying by 16.

35 xma_inst (Non-Speculative)

Counts the number of committed “non-SIMD integer multiply add” instructions. The counted instructions are `FPMADDX` and `FPMADDXHI`.

36 SIMD_xma_inst (Non-Speculative)

Counts the number of committed “SIMD integer multiply add” instructions. The counted instructions are the same as xma_inst.

37 4SIMD_xma_inst (Non-Speculative)

Counts the number of committed “4SIMD integer multiply add” instructions. The counted instructions are the same as xma_inst.

38 unpack_sxar1 (Non-Speculative)

Counts the number of unpacked `SXAR1` instructions that are committed.

39 unpack_sxar2 (Non-Speculative)

Counts the number of unpacked SXAR2 instructions that are committed.

40 instruction_flow_counts (Non-Speculative)

Counts the number of committed instruction flows. On SPARC64™ XIfx, some instructions are processed internally as several separate instructions, called instruction flows. This event does not count packed SXAR1 and SXAR2 instructions.

41 ex_load_instructions (Non-Speculative)

Counts the number of committed “integer-load” instructions. Counts the LD{S|U}B{A}, LD{S|U}H{A}, LD{S|U|T}W{A}, LDTW{A}, LDX{A} and LDTXA instructions.

42 ex_store_instructions (Non-Speculative)

Counts the number of committed “integer-store and atomic” instructions. Counts the STB{A}, STH{A}, STW{A}, STTW{A}, STX{A}, LDSTUB{A}, SWAP{A}, and CAS{X}A instructions..

43 fl_load_instructions (Non-Speculative)

Counts the number of committed “non-SIMD floating-point load” instructions. Counts the LDF{A}, LDDF{A}, and LD{|X|XE}FSR instructions. This event does not count LDQF{A}.

44 fl_store_instructions (Non-Speculative)

Counts the number of committed “non-SIMD floating-point store” instructions. Counts the STF{A}, STDF{A}, STFR, STDFR, and ST{X}FSR instructions. This event does not count STQF{A}.

45 SIMD_fl_load_instructions (Non-Speculative)

Counts the number of committed “SIMD floating-point load” instructions. The counted instructions are the same as fl_load_instructions.

46 SIMD_fl_store_instructions (Non-Speculative)

Counts the number of committed “SIMD floating-point store” instructions. The counted instructions are the same as fl_store_instructions.

47 4SIMD_fl_load_instructions (Non-Speculative)

Counts the number of committed "4SIMD floating point load" instructions. The counted instructions are the same as fl_load_instructions.

48 4SIMD_fl_store_instructions (Non-Speculative)

Counts the number of committed “4SIMD floating point store” instructions. The counted instructions are the same as fl_store_instructions.

49 iwr_empty

Counts the number of cycles that the IWR (Issue Word Register) is empty. IWR is a four entry register that holds instructions during instruction decode; the IWR may be empty if an instruction cache miss prevents instruction fetch.

50 rs1 (Non-Speculative)

Counts the number of cycles in which normal execution is halted due to the following:

- a trap or interrupt
- update of privileged registers
- to guarantee memory ordering
- RAS-initiated hardware retry

51 flush_rs (Non-Speculative)

Counts the number of pipeline flushes due to branch misprediction. Since SPARC64™ XIfx supports speculative execution, instructions that should not have been executed may be in flight. When it is determined that the predicted path is incorrect, these instructions are cancelled. A pipeline flush occurs at this time.

misprediction rate = flush_rs / branch_instructions

52 0iid_use

Counts the number of cycles where no instruction is issued. SPARC64™ XIfx issues up to four instructions per cycle; when no instruction is issued, 0iid_use is incremented. On SPARC64™ XIfx, some instructions are processed internally as several separate instructions, called instruction flows. Each of these instruction flows is counted. SXAR instructions are also counted.

53 1iid_use

Counts the number of cycles where one instruction is issued.

54 2iid_use

Counts the number of cycles where two instructions are issued.

55 3iid_use

Counts the number of cycles where three instructions are issued.

56 4iid_use

Counts the number of cycles where four instructions are issued.

57 sync_intlk

Counts the number of cycles where instruction issue is blocked by a pipeline sync.

58 regwin_intlk

Counts the number of cycles where instruction issue is blocked by a register window switch.

59 decode_all_intlk

Counts the number of cycles where instruction issue is blocked by a static interlock condition at the decode stage. decode_all_intlk includes sync_intlk and regwin_intlk; stall cycles due to dynamic conditions (such as reservation station full) are not counted.

60 rsf_pmmi (Non-Speculative)

Counts the number of cycles where mixing single-precision and double-precision floating-point operations prevents instructions from issuing.

61 toq_rsbr_phantom

Counts the number of instructions that are predicted taken but are not actually branch instructions. Branch prediction on SPARC64™ XIfx is done prior to instruction decode; in other words, branch prediction occurs regardless of whether the instruction is actually a branch. Instructions that are not branch instructions may be incorrectly predicted as taken branches.

62 1FLOPS_instructions (Non-Speculative)

Counts the number of committed "floating-point" instructions that execute one floating-point operation; `FTRISMULd`, `F{SQRT|ADD|SUB|MUL|DIV|NADD|NMUL}{s|d}` and `F{N}sMULd.`

63 2FLOPS_instructions (Non-Speculative)

Counts the number of committed "floating-point" instructions that execute two floating-point operations; SIMD `FTRISMULd`, SIMD `F{ADD|SUB|MUL|NADD|NMUL}{s|d}`, SIMD `F{N}sMULd`, `F{ADD|SUB|MUL}ds` and fma_instructions. The number of operations is obtained by multiplying by 2.

64 4FLOPS_instructions (Non-Speculative)

Counts the number of committed "floating-point" instructions that execute four floating-point operations; 4SIMD `FTRISMULd`, 4SIMD `F{ADD|SUB|MUL|NADD|NMUL}{s|d}`, 4SIMD `F{N}sMULd`, SIMD `F{ADD|SUB|MUL}ds`, SIMD fma_instructions and fma_DSP_instructions. The number of operations is obtained by multiplying by 4.

65 8FLOPS_instructions (Non-Speculative)

Counts the number of committed "floating-point" instructions that execute eight floating-point operations; 4SIMD `F{ADD|SUB|MUL}ds`, SIMD_fma_DSP_instructions and 4SIMD_fma_instructions. The number of operations is obtained by multiplying by 8.

66 16FLOPS_instructions (Non-Speculative)

Counts the number of committed "floating-point" instructions that execute sixteen floating-point operations; 4SIMD_fma_DSP_instructions. The number of operations is obtained by multiplying by 16.

67 fl_move_instructions (Non-Speculative)

Counts the number of committed “non-SIMD floating-point move” instructions. The counted instructions are `FMOV{s|d}` ,`FRD{d|s}`, `F{MAX|MIN}{d|s}`, `FNEG{s|d}`, `FABS{s|d}` and convert instructions(FPop1 with opf<8:4> = 0x08,0x0C and 0x0D).

68 fl_movecc_r_sel_instructions (Non-Speculative)

Counts the number of committed “non-SIMD floating-point movecc, mover and selmov” instructions. The counted instructions are `FMOV{s|d}cc`, `FMOV{s|d}r`, and `FSELMOV{s|d}`.

69 SIMD_fl_move_instructions (Non-Speculative)

Counts the number of committed “SIMD floating-point move” instructions. The counted instructions are the same as fl_move_instructions.

70 SIMD_fselmove_instructions (Non-Speculative)

Counts the number of committed “SIMD floating-point selmov” instructions. The counted instructions are `FSELMOV{s|d}` instructions.

71 4SIMD_fl_move_instructions (Non-Speculative)

Counts the number of committed “4SIMD floating-point move” instructions. The counted instructions are the same as fl_move_instructions.

72 4SIMD_fselmove_instructions (Non-Speculative)

Counts the number of committed “4SIMD floating-point selmov” instructions. The counted instructions are the same as SIMD_fselmove_instructions.

73 fixed_point_instructions (Non-Speculative)

Counts the number of committed “non-SIMD fixed point partitioned add/sub and integer multiply-add” instructions. The counted instructions are

FP{ADD,SUB}{16|32|64}{S},FPMUL64 and FPMADDX{HI}. For FP{ADD|SUB}{16|32} instructions, four or two operations are executed per instruction, but each instruction is counted as "1" per committed.

74 SIMD_fixed_point_instructions (Non-Speculative)

Counts the number of committed “SIMD fixed point partitioned add/sub and integer multiply add” instructions. The counted instructions are the same as fixed_point_instructions. For SIMD versions of FP{ADD|SUB}{16|32} instructions, eight or four operations are executed per instruction, but each instruction is counted as "1" per committed.

75 4SIMD_fixed_point_instructions (Non-Speculative)

Counts the number of committed “4SIMD fixed point partitioned add/sub and integer multiply add” instructions. The counted instructions are the same as fixed-point instructions. For 4SIMD versions of FP{ADD|SUB}{16|32} instructions, sixteen or eight operations are executed per instruction, but each instruction is counted as "1" per committed.

76 XSIMD_fixed_point_instructions (Non-Speculative)

Counts the number of committed “SIMD and 4SIMD fixed point partitioned add/sub and integer multiply add” instructions. The counted instructions are the same as fixed_point_instructions.

77 fepermd_instructions (Non-Speculative)

Counts the number of committed “non-SIMD full element permutation” instructions. The counted instruction is FEPERMD.

78 element_summask_comp_instructions (Non-Speculative)

Counts the number of committed “non-SIMD element sum mask” and ”element compress” instructions. The counted instructions are FESUMMD and FECPD instruction.

79 fecsld_instructions (Non-Speculative)

Counts the number of committed “4SIMD element concatenate shift left” instructions. The counted instruction is FECSLD instruction.

80 SIMD_fepermd_instructions (Non-Speculative)

Counts the number of committed “SIMD full element permutation” instructions.

81 SIMD_ element_summask_comp_instructions (Non-Speculative)

Counts the number of committed “SIMD element sum mask” and ”SIMD element compress” instructions.

82 4SIMD_fepermd_instructions (Non-Speculative)

Counts the number of committed “4SIMD full element permutation” instructions.

83 4SIMD_ element_summask_comp_instructions (Non-Speculative)

Counts the number of committed “4SIMD element sum mask” and ”4SIMD element compress” instructions.

84 frcpa_instructions (Non-Speculative)

Counts the number of committed “non-SIMD reciprocal approximation” instructions. The counted instructions are `FRCPA{s|d}`.

85 frsqrta_instructions (Non-Speculative)

Counts the number of committed “non-SIMD reciprocal square approximation” instructions. The counted instructions are `FRSQRTA{s|d}`.

86 trigonometric_exponential_instructions (Non-Speculative)

Counts the number of committed “non-SIMD trigonometric” and “exponential auxiliary” instructions. The counted instructions are `FTRISSELd`, `FTRISMULd` and `FEXPAd`.

87 flogical_instructions (Non-Speculative)

Counts the number of committed “non-SIMD floating register logical operate” instructions. The counted instructions are IMPDEP1 with opf<8:4>=0x06 and 0x07.

88 fpcmp_instructions (Non-Speculative)

Counts the number of committed “non-SIMD compare” instructions. The counted instructions are IMPDEP1 with opf<8:4>=0x0C and 0x0D.

89 SIMD_frcpa_instructions (Non-Speculative)

Counts the number of committed “SIMD reciprocal approximation” instructions. The counted instructions are the same as frcpa_instructions.

90 SIMD_frsqrta_instructions (Non-Speculative)

Counts the number of committed “SIMD reciprocal square approximation” instructions. The counted instructions are the same as frsqrta_instructions.

91 SIMD_ trigonometric_exponential_instructions (Non-Speculative)

Counts the number of committed “SIMD trigonometric” and “SIMD exponential auxiliary” instructions. The counted instructions are the same as trigonometric_exponential_instructions.

92 SIMD_flogical_instructions (Non-Speculative)

Counts the number of committed “SIMD floating register logical operate” instructions. The counted instructions are the same as flogical_instructions.

93 SIMD_fpcmp_instructions (Non-Speculative)

Counts the number of committed “SIMD compare” instructions. The counted instructions are the same as fpcmp_instructions.

94 4SIMD_frcpa_instructions (Non-Speculative)

Counts the number of committed “4SIMD reciprocal approximation” instructions. The counted instructions are the same as frcpa_instructions.

95 4SIMD_frsqrta_instructions (Non-Speculative)

Counts the number of committed “4SIMD reciprocal square approximation” instructions. The counted instructions are the same as frsqrta_instructions.

96 4SIMD_ trigonometric_exponential_instructions (Non-Speculative)

Counts the number of committed “4SIMD trigonometric” and “4SIMD exponential auxiliary” instructions. The counted instructions are the same as trigonometric_exponential_instructions.

97 4SIMD_flogical_instructions (Non-Speculative)

Counts the number of committed “4SIMD floating register logical operate” instructions. The counted instructions are the same as flogical_instructions.

98 4SIMD_fpcmp_instructions (Non-Speculative)

Counts the number of committed “4SIMD compare” instructions. The counted instructions are the same as fpcmp_instructions.

99 indirect_prefetch_instructions (Non-Speculative)

Counts the number of committed “non-SIMD indirect prefetch” instructions. The counted instruction is `PREFETCHID` instruction.

100 SIMD_indirect_prefetch_instructions (Non-Speculative)

Counts the number of committed “SIMD indirect prefetch” instructions.

101 4SIMD_indirect_prefetch_instructions (Non-Speculative)

Counts the number of committed “4SIMD indirect prefetch” instructions.

102 nonSIMD_XSIMD_indirect_prefetch_instructions (Non-Speculative)

Counts the number of committed “Non-SIMD, SIMD and 4SIMD of indirect prefetch” instructions. The counted instruction is `PREFETCHID` instruction.

103 indirect_load_fl_instructions (Non-Speculative)

Counts the number of committed “non-SIMD indirect load floating-point” instructions. The counted instructions are `LD{D}FID` and `LDFID{U|S}W`.

104 indirect_store_fl_instructions (Non-Speculative)

Counts the number of committed “non-SIMD indirect store floating-point” instructions. The counted instructions are `ST{D}FID` and `STFID{U|S}W`.

105 store_fl_condition_instructions (Non-Speculative)

Counts the number of committed “non-SIMD store floating-point register on register condition” instructions. The counted instructions are `ST{D}FR`.

106 stfruw_instructions (Non-Speculative)

Counts the number of committed “non-SIMD store floating-point register on register condition unsigned word data” instructions. The counted instruction is `STFRUW` instruction.

107 indirect_store_fl_condition_instructions (Non-Speculative)

Counts the number of committed “non-SIMD indirect store floating-point register on register condition” instructions. The counted instructions are `ST{D}FRID` and `STFRIDUW`.

108 ldfuw_sw_ib_instructions (Non-Speculative)

Counts the number of committed “non-SIMD load floating-point register unsigned and signed word data” and “load floating-point register on internal broadcast” instructions. The counted instructions are `LDF{U|S}W` and `LDFIB`.

109 stfuw_instructions (Non-Speculative)

Counts the number of committed “non-SIMD store floating-point register on unsigned word data” instructions. The counted instruction is `STFUW`.

110 lddfds_instructions (Non-Speculative)

Counts the number of committed “non-SIMD load dual single-floating point register” instruction. The counted instruction is `LDDFDS`.

111 stdfds_instructions (Non-Speculative)

Counts the number of committed “non-SIMD store dual floating-point register for double single floating” instruction. The counted instruction is `STDFDS`.

112 SIMD_indirect_load_fl_instructions (Non-Speculative)

Counts the number of committed “SIMD indirect load floating point” instructions. The counted instructions are the same as indirect_load_fl_instructions.

113 SIMD_stride_load_fl_instructions (Non-Speculative)

Counts the number of committed “SIMD stride load floating point” instructions. The counted instructions are `LD{D}FST`, `LDFST{U|S}W` and `LDFSTIB`.

114 SIMD_broadcast_load_fl_instructions (Non-Speculative)

Counts the number of committed “SIMD broadcast load floating-point” instructions. The counted instructions are `LD{D}FBC`, `LDFBC{U|S}W` and `LDFBCIB`.

115 SIMD_stride_store_fl_instructions (Non-Speculative)

Counts the number of committed “SIMD stride store floating-point” instructions. The counted instructions are `ST{D}ST` and `STFSTUW`.

116 SIMD_indirect_store_fl_instructions (Non-Speculative)

Counts the number of committed “SIMD indirect store floating-point” instructions. The counted instructions are the same as indirect_store_fl_instructions.

117 SIMD_store_fl_condition_instructions (Non-Speculative)

Counts the number of committed “SIMD store floating-point register on register condition” instructions. The counted instructions are the same as store_fl_condition_instructions.

118 SIMD_stfruw_instructions (Non-Speculative)

Counts the number of committed “SIMD store floating-point register on register condition unsigned word data” instructions. The counted instructions are the same as stfruw_instruction.

119 SIMD_stride_store_fl_condition_instructions (Non-Speculative)

Counts the number of committed “SIMD stride store floating-point register on register condition” instructions. The counted instructions are `ST{D}FRST` and `STFRSTUW`.

120 SIMD_indirect_store_fl_condition_instructions (Non-Speculative)

Counts the number of committed “SIMD indirect store floating-point register on register condition” instructions. The counted instructions are the same as indirect_store_fl_condition_instructions.

121 SIMD_ldfuw_sw_ib_instructions (Non-Speculative)

Counts the number of committed “SIMD load floating point register on unsigned and signed word date” and “SIMD load floating point register on internal broadcast” instructions. The counted instructions are the same as ldfuw_sw_ib_instructions.

122 SIMD_stfuw_instructions (Non-Speculative)

Counts the number of committed “SIMD store floating point register on unsigned word data” instructions.

123 SIMD_lddfds_instructions (Non-Speculative)

Counts the number of committed “SIMD load dual single floating-point register” instruction.

124 SIMD_stdfds_instructions (Non-Speculative)

Counts the number of committed “SIMD store dual floating-point register for double single floating” instruction.

125 4SIMD_indirect_load_fl_instructions (Non-Speculative)

Counts the number of committed “4SIMD indirect load floating-point” instructions. The counted instructions are the same as indirect_load_fl_instructions.

126 4SIMD_stride_load_fl_instructions (Non-Speculative)

Counts the number of committed “4SIMD stride load floating-point” instructions. The counted instructions are the same as SIMD_stride_load_fl_instructions.

127 4SIMD_broadcast_load_fl_instructions (Non-Speculative)

Counts the number of committed “4SIMD broadcast load floating-point” instructions. The counted instructions are the same as SIMD_broadcast_load_fl_instructions.

128 4SIMD_stride_store_fl_instructions (Non-Speculative)

Counts the number of committed “4SIMD stride store floating-point” instructions. The counted instructions are the same as SIMD_stride_store_fl_instructions.

129 4SIMD_indirect_store_fl_instructions (Non-Speculative)

Counts the number of committed “4SIMD indirect store floating-point” instructions. The counted instructions are the same as indirect_store_fl_instructions.

130 4SIMD_store_fl_condition_instructions (Non-Speculative)

Counts the number of committed “4SIMD store floating-point register on register condition” instructions. The counted instructions are the same as store_fl_condition_instructions.

131 4SIMD_stfruw_instructions (Non-Speculative)

Counts the number of committed “4SIMD store floating-point register on register condition unsigned word data” instructions. The counted instructions are the same as stfruw_instruction.

132 4SIMD_stride_store_fl_condition_instructions (Non-Speculative)

Counts the number of committed “4SIMD stride store floating-point register on register condition” instructions. The counted instructions are the same as SIMD_stride_store_fl_condition_instructions.

133 4SIMD_indirect_store_fl_condition_instructions (Non-Speculative)

Counts the number of committed “4SIMD indirect store floating-point register on register condition” instructions. The counted instructions are the same as indirect_store_fl_condition_instructions.

134 4SIMD_ldfuw_sw_ib_instructions (Non-Speculative)

Counts the number of committed “4SIMD load floating-point register on unsigned and signed word date” and “4SIMD load floating-point register on internal broadcast” instructions. The counted instructions are the same as ldfuw_sw_ib_instructions.

135 4SIMD_stfuw_instructions (Non-Speculative)

Counts the number of committed “4SIMD store floating-point register on unsigned word data” instructions.

136 4SIMD_lddfds_instructions (Non-Speculative)

Counts the number of committed “4SIMD load dual single floating-point register” instruction.

137 4SIMD_stdfds_instructions (Non-Speculative)

Counts the number of committed “4SIMD store dual floating-point register for double single floating” instruction.

138 store_with_xfill_instructions (Non-Speculative)

Counts the number of committed “cache line fill with undetermined values” instructions. The counted instructions are store instructions with ASI.

139 op_stv_wait (Non-Speculative)

Counts the number of cycles where no instructions are committed because the oldest, uncommitted instruction is a memory access waiting for data. op_stv_wait does not count cycles where a store instruction is waiting for data (atomic instructions are counted).

Note that op_stv_wait does not measure the cache-miss latency, since any cycles prior to becoming the oldest, uncommitted instruction are not counted.

140 op_stv_wait_nc_pend (Non-Speculative)

Counts op_stv_wait for noncacheable accesses.

141 op_stv_wait_ex (Non-Speculative)

Counts op_stv_wait for integer memory access instructions. Does not distinguish between the L1 cache and L2 cache misses.

142 op_stv_wait_sxmiss (Non-Speculative)

Counts op_stv_wait caused by an L2 cache miss. Does not distinguish between integer and floating-point loads.

143 op_stv_wait_l1d_cache_miss (Non-Speculative)

Counts op_stv_wait caused by an L1D cache miss. Does not distinguish between integer and floating-point loads.

144 op_stv_wait_sxmiss_ex (Non-Speculative)

Counts op_stv_wait caused by an integer-load L2 cache miss.

145 op_stv_wait_l1d_cache_miss_ex (Non-Speculative)

Counts op_stv_wait caused by an integer-load L1 cache miss.

146 op_stv_wait_pfp_busy (Non-Speculative)

Counts op_stv_wait caused by a memory access instruction that cannot be executed due to the lack of an available prefetch port.

147 op_stv_wait_pfp_busy_ex (Non-Speculative)

Counts op_stv_wait caused by an integer memory access instruction that cannot be executed due to the lack of an available prefetch port.

148 op_stv_wait_swpf (Non-Speculative)

Counts op_stv_wait caused by a prefetch instruction.

149 op_stv_wait_pfp_busy_swpf (Non-Speculative)

Counts op_stv_wait caused by a prefetch instruction that cannot be executed due to the lack of an available prefetch port.

150 cse_window_empty_sp_full (Non-Speculative)

Counts the number of cycles where no instructions are committed because the CSE is empty and the store ports are full.

151 cse_window_empty (Non-Speculative)

Counts the number of cycles where no instructions are committed because the CSE is empty.

152 branch_comp_wait (Non-Speculative)

Counts the number of cycles where no instructions are committed and the oldest, uncommitted instruction is a branch instruction. Measuring branch_comp_wait has a lower priority than measuring eu_comp_wait.

153 eu_comp_wait (Non-Speculative)

Counts the number of cycles where no instructions are committed and the oldest, uncommitted instruction is an integer or floating-point instruction. Measuring eu_comp_wait has a higher priority than measuring branch_comp_wait.

154 fl_comp_wait (Non-Speculative)

Counts the number of cycles where no instructions are committed and the oldest, uncommitted instruction is a floating-point instruction.

155 0endop (Non-Speculative)

Counts the number of cycles where no instructions are committed. 0endop also counts cycles where the only instruction that commits is an SXAR instruction.

156 1endop (Non-Speculative)

Counts the number of cycles where one instruction is committed.

157 2endop (Non-Speculative)

Counts the number of cycles where two instructions are committed.

158 3endop (Non-Speculative)

Counts the number of cycles where three instructions are committed.

159 suspend_cycle (Non-Speculative)

Counts the number of cycles where the instruction unit is halted.

160 sleep_cycle (Non-Speculative)

Counts the number of cycles where the instruction unit is halted by a SLEEP instruction

161 single_sxar_commit (Non-Speculative)

Counts the number of cycles where the only instruction committed is an unpacked SXAR instruction. These cycles are also counted by 0endop.

162 d_move_wait (non-speculative)

Counts the number of cycles where no instructions are committed while waiting for the register window to be updated.

11.2.2. MMU and L1 cache Events

Standard PA Events

1 uITLB_miss

Counts the occurrences of instruction uTLB misses.

2 uDTLB_miss

Counts the occurrences of data uTLB misses.

3 L1I_miss

Counts the occurrences of L1 instruction cache misses.

4 L1D_miss

Counts the occurrences of L1 data cache misses.

5 L1I_wait_all

Counts the total time spent processing L1 instruction cache misses (i.e., the total miss latency). On SPARC64™ XIfx, the L1 cache is a non-blocking cache that can process multiple cache misses in parallel; L1I_wait_all only counts the miss latency for one of these misses. That is, the overlapped miss latencies are not counted.

6 L1D_wait_all

Counts the total time spent processing L1 data cache misses (i.e., the total miss latency). On SPARC64™ XIfx, the L1 cache is a non-blocking cache that can process multiple cache misses in parallel; L1D_wait_all only counts the miss latency for one of these misses. That is, the overlapped miss latencies are not counted.

Supplemental PA Events

7 ITLB_write

Counts the number of ITLB writes caused by an instruction fetch ITLB miss.

8 DTLB_write

Counts the number of DTLB writes caused by a data access DTLB miss.

9 swpf_success_all

Counts the number of prefetch instructions that are not lost in the SU and are sent to the SX .

10 swpf_fail_all

Counts the number of prefetch instructions that are lost in the SU.

11 swpf_lbs_hit

Counts the number of prefetch instructions that hit in the L1 cache.

The number of prefetch instructions sent to the SU =
swpf_success_all + swpf_fail_all + swpf_lbs_hit

12 L1I_thrashing

Counts the occurrences of an L2 read request being issued twice in the period between acquiring and releasing a store port. When instruction fetch causes an L1 instruction cache miss, the requested data is updated in the L1I cache. This counter is incremented if the updated data is evicted before it can be read.

13 L1D_thrashing

Counts the occurrences of an L2 read request being issued twice in the period between acquiring and releasing a store port. When a memory access instruction causes an L1 data cache miss, the requested data is updated in the L1D cache. This counter is incremented if the updated data is evicted before it can be read.

14 L1D_miss_dm

Counts the occurrences of L1 data cache misses for load/store instructions.

15 L1D_miss_pf

Counts the occurrences of L1 data cache misses for prefetch instructions.

16 L1D_miss_qpf

Counts the occurrences of L1 data cache misses for hardware prefetch requests.

17 L1D_pipe0_valid

Counts the number of L1 data cache busy at pipeline0.

18 L1D_pipe1_valid

Counts the number of L1 data cache busy at pipeline1.

11.2.3. L2 cache Events

L2 cache events may be due to the actions of VCPU, I/O or inter-CMG requests. Events caused by VCPUs are counted separately for each VCPU; those caused by I/O or inter-CMG requests are counted for all VCPUs on the CMG. This means 2 CMGs has separate counters for this kind of requests.

Most L2 cache events are categorized as either demand (dm) or prefetch (pf) events, but these categories do not directly correspond to load/store/atomic and prefetch instructions, for the following reasons.

- When a load/store instruction cannot be executed due to a lack of resources needed to move data into the L1 cache, data is first moved into the L2 cache. Once L1 cache resources become available, the load/store instruction is executed. That is, the request to move data into the L2 cache is processed as a prefetch request.
- The hardware prefetch mechanism generates prefetch requests.

- L1 cache prefetch instructions are processed as demand requests.

Instead, demand and prefetch L2 cache events correspond to the following:

- A demand (dm) request to the L2 cache is an instruction fetch, load/store instruction, or L1 prefetch instruction that successfully acquired the resources needed to access memory.
- A prefetch (pf) request to the L2 cache is an instruction fetch, load/store instruction, or L1 prefetch instruction that could not acquire the resources needed to access memory; L2 prefetch instructions and hardware prefetch are also prefetch requests.

Standard PA Events

1 L2_read_dm

Counts the number of L2 cache references by demand requests.

Inter-CMG cache-reference requests are not counted.

2 L2_read_pf

Counts the number of L2 cache references by prefetch requests.

3 L2_miss_dm

Counts the number of L2 cache misses caused by demand requests.

This counter is the sum of the L2_miss_counts_dm_bank{0, 1, 2, 3}.

4 L2_miss_pf

Counts the number of L2 cache misses caused by prefetch requests.

This counter is the sum of the L2_miss_counts_pf_bank {0, 1, 2, 3}.

5 L2_miss_counts_dm_bank {0, 1, 2, 3}

Counts the number of L2 cache misses for each bank caused by demand requests.

When an L2 cache miss causes a prefetch request for an address to be issued, and then a demand request for the same address is issued before the data is returned from memory on the CMG or the other CMG, the later demand request is not counted in L2_miss_counts_dm_bank{0,1,2,3}.

6 L2_miss_counts_pf_bank {0, 1, 2, 3}

Counts the number of L2 cache misses for each bank caused by prefetch requests.

7 L2_miss_wait_dm_bank {0, 1, 2, 3}

Counts the total time spent processing L2 cache misses for each bank caused by demand requests (i.e., the total miss latency for each bank). The latency of each memory access request is counted.

When an L2 cache miss causes prefetch request for an address to be issued and then a demand request for the same address is issued before the data is returned from memory on the CMG or the other CMG, the successive cycle counts to the data reply are counted in L2_miss_wait_dm_bank{0,1,2,3}.

8 L2_miss_wait_pf_bank {0, 1, 2, 3}

Counts the total time spent processing L2 cache misses for each bank caused by prefetch requests, (i.e., the total miss latency for each bank). The latency of each memory access request is counted.

The L2 cache miss latency can be derived by summing L2_miss_wait_* and dividing by the sum of L2_miss_counts_*. If individual L2 cache-miss latencies are calculated for pf/dm requests, the value obtained for the miss latency for dm requests may be higher than expected.

9 L2_wb_dm

Counts the occurrences of writeback to memory caused by L2 cache misses for demand requests.

10 L2_wb_pf

Counts the occurrences of writeback to memory caused by L2 cache misses for prefetch requests.

Supplemental PA Events

11 lost_pf_pfp_full

Counts the number of weak prefetch requests lost due to prefetch port full.

12 lost_pf_by_abort

Counts the number of weak prefetch requests lost due to L2- pipe abort.

11.2.4. Bus Transaction Events

Standard PA Events

1 cpu_mem_read_counts

Counts the number of memory read requests issued by VCPUs (both intra- and inter-CMG requests).

For this event, the same value is counted by all VCPUs on the CMG.

2 cpu_mem_write_counts

Counts the number of memory write requests issued by VCPUs (both intra- and inter-CMG requests).

For this event, the same value is counted by all VCPUs on the CMG.

3 IO_mem_read_counts

Counts the number of memory read requests issued by I/O.

For this event, the same value is counted by all VCPUs on the CMG.

4 IO_mem_write_counts

Counts the number of memory write requests issued by I/O.

Only IO-FST is counted by this event. IO-PST can be counted using IO_pst_counts.

For this event, the same value is counted by all VCPUs on the CMG.

5 bi_counts

Counts the number of inter-CMG or I/O cache-invalidate requests received by the CMG.

IO cache-invalidate requests means cache-invalidate action caused by internal IO-FST/PST requests.

These requests do not check the cache data before invalidating.

For this event, the same value is counted by all VCPUs on the CMG.

6 cpi_counts

Counts the number of inter-CMG cache-copy-and-invalidate requests received by the CMG.

For this event, the same value is counted by all VCPUs on the CMG.

7 cpb_counts

Counts the number of inter-CMG cache-copyback requests received by the CMG.

For this event, the same value is counted by all VCPUs on the CMG.

8 cpd_counts

Counts the number of I/O cache-read requests (DMA read requests) received by the CMG.

For this event, the same value is counted by all VCPUs on the CMG.

Supplemental PA Events

9 IO_pst_counts

Counts the number of memory write requests (IO-PST) issued by I/O.

For this event, the same value is counted by all VCPUs on the CMG.

11.3. Cycle Accounting

Cycle accounting is a method for analyzing performance bottlenecks. The total time (number of CPU cycles) required to execute an instruction sequence can be divided into time spent in various CPU execution states (executing instructions, waiting for a memory access, waiting for execution to complete, and so on).

SPARC64™ XIfx defines a large number of PA events that record detailed information about CPU execution states, enable efficient analysis of bottlenecks, and are useful for performance tuning.

In this document, cycle accounting is specifically defined as the analysis of instructions as they are committed in order. SPARC64™ XIfx executes instructions out-of-order and have multiple execution units; the CPU is generally in a state where executing instructions and waiting instructions are thoroughly mixed together. One instruction may be waiting for data from memory, another executing a floating-point multiply, and yet another waiting for confirmation of the branch direction. Simply analyzing the reasons why individual instructions are waiting is not useful. Instead, cycle accounting classifies cycles by the number of instructions committed; when a cycle commits no instructions, the conditions that prevented instructions from committing are analyzed.

SPARC64™ XIfx commits up to 4 instructions per cycle. The more cycles that commit the maximum number of instructions, the better the execution efficiency. Cycles that do not commit any instructions have an extremely negative effect on performance, so it is important to perform a detailed analysis of these cycles. The main causes are:

- Waiting for a memory access to return data.
- Waiting for instruction execution to complete.
- Instruction fetch is unable to supply the pipeline with instructions.

Table 11-4 highlights some useful PA events and describes how these PA events can be used to analyze execution efficiency.

Figure 11-1 shows the relationship between the various op_stv_wait_* events. The PA events marked with a † in the table and in the figure are synthetic events, which are calculated from other PA events.

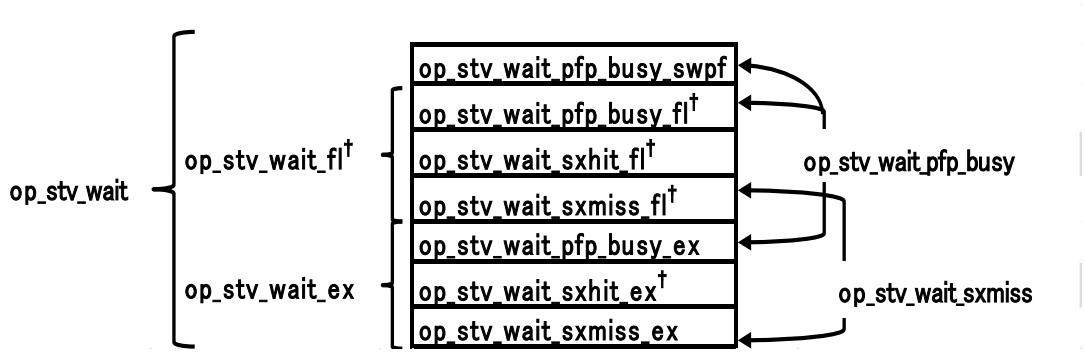


Figure 11-1 Breakdown of op_stv_wait

Table 11-4 Useful Performance Events for Cycle Accounting

Instructions Committed per Cycle	Cycles	Remarks
4	*cycle_counts* - *3endop* - *2endop* - *1endop* - *0endop*	N/A (Four instructions are committed in a cycle)
3	*3endop*	*inh_cmit_gpr_2write* measures one of the conditions that can prevent subsequent instruction(s) from committing.
2	*2endop*	
1	*1endop*	
0	Execution: *eu_comp_wait* + *branch_comp_wait*	*eu_comp_wait* = *ex_comp_wait*†+ *fl_comp_wait*
	Instruction Fetch: *cse_window_empy*	*cse_window_empty* = *cse_window_empty_sp_full* + *sleep_cycle* + misc.†
	L1D cache miss: *op_stv_wait* - L2 cache miss (see below)	
	L2 cache miss: *op_stv_wait_sxmiss* + *op_stv_wait_nc_pend*	
	Others: *0endop* - *op_stv_wait* - *cse_window_empy* - *eu_comp_wait* - *branch_comp_wait* -(*instruction_flow_counts* - *instruction_counts*)	

12. Traps

トラップとは、ある特権レベルで動作しているソフトウェアの制御を、より高い特権レベルに変更し、同時にある特定の実行命令列に変更することである。非特権モードでトラップが発生すると、特権モードまたはハイパーバイザモードに遷移し、特権モードでトラップが発生すると、特権モードまたはハイパーバイザモードに遷移する。

トラップにより実行される命令列は、トラップハンドラの先頭 8 命令（*clean_window*, *spill_n_**, *fill_n_**では 32 命令）である。トラップハンドラは、特権モードに遷移する際、TBA レジスタで示される仮想アドレスに位置する。トラップテーブル内の命令列の位置は、トラップの種類（TT）とトラップレベル（TL）によって決まる。トラップテーブルの半分はハードウェアによって発生するトラップ用で、残り半分は Tcc 命令で発生するソフトウェアトラップ用である。

トラップは、予期しない関数呼び出しのように振舞う。トラップ発生時のハードウェア動作は、

1. トラップ発生時点の VCPU の状態（PC, CWP, ASI, PSTATE, TT など）をレジスタにセーブする。
2. PSTATE に予め決められた値が設定され、特権モードに遷移する。
3. トラップテーブルの命令の実行を開始する。

トラップハンドラの実行が終了すると、元の命令列に制御が戻る。

トラップは Tcc 命令、命令実行による例外、リセット、ハードウェアエラー、さらには、命令とは関係ない非同期な割り込みにより発生する。これに対し VCPU は、各命令の実行に先立ち、ペンディングしている例外や割り込みがあるかを検査するように振舞わなくてはならない。ペンディングしている例外や割り込みが複数あるならば、一番優先度の高いものを選び、トラップを発生させる。

つまり“例外”とは、VCPU がその時点で実行している命令列を、ソフトウェアの介在なしにそれ以上続けることができなくなるような事象を指す。これに対し“トラップ”とは、例外、割り込み、リセット、または Tcc 命令実行の結果、VCPU が実行命令列を変更するという動作を指す。“割り込み”とは、VCPU 外部からの要求を指す。

12.1. Virtual Processor Privilege Modes

プロセッサの特権レベルを以下に示す。

- 非特権モード（nonprivileged）
- 特権モード（privileged）
- ハイパーバイザモード（hyperprivileged）

非特権モードより特権モードが、特権モードよりハイパーバイザモードがより権限の大きい特権レベルである（特権レベルが高い）。

本ドキュメントではハイパーバイザモードについて言及しない。

特権レベルは PSTATE.priv の設定で決まる。

表 12-1　特権レベルの設定

	PSTATE.priv
非特権モード	0
特権モード	1

特権レベルの変更は、トラップか、PSTATE.priv を表 12-1 の通りに設定することで行う。低い特権レベルから高い特権レベルへはトラップでのみ遷移できる。トラップで高い特権レベルから低い特権レベルへ遷移することはない。

トラップと遷移する特権レベルの関係は固定されている。どのトラップがどの特権レベルに遷移するかは表 12-2 (360 ページ)を参照。高い特権レベルで動作中の VCPU で、低い特権レベルに遷移する例外が起きた場合、VCPU の特権レベルが、その例外が遷移する特権レベルより低くなるまでペンディングされる。特権レベルと TL の関係については、UA2011 12.1 を参照。

12.2. トラップ制御

12.2.1. Trap Type (TT)

SPARC64™ XIfx では、*instruction_breakpoint* と *illegal_instruction* の優先順位は UA2011 に準拠する。また、*control_transfer_instruction* が特権モードで起きた場合、特権モードへ割り込む仕様とする。

12.3. トラップ一覧と優先順位

記号	説明
-x-	このモードではトラップは発生しない。
P	priv モードに遷移する。
P(ie)	PSTATE.ie = 1 のとき priv モードに遷移する。
H	hpriv モードに遷移する。

表 12-2 トラップ一覧 (TT 順)

TT	トラップ名	種類	優先順位	トラップ後の特権レベル	定義
000_{16}	*reserved*	—	—	—	—
006_{16}	*reserved*	—	—	—	—
007_{16}	*reserved*	—	—	—	—
008_{16}	*IAE_privilege_violation*	precise	3.1	H	368
$00B_{16}$	*IAE_unauth_access*	precise	2.7	H	368
$00C_{16}$	*IAE_nfo_page*	precise	3.3	H	368
$00D_{16}$	*reserved*	—	—	—	—

$00E_{16}$	*reserved*	—	—	—	—
$00F_{16}$	*reserved*	—	—	—	—
010_{16}	*illegal_instruction*	precise	6.2	H	369
011_{16}	*privileged_opcode*	precise	7	P	371
012_{16}	*reserved*	—	—	—	—
013_{16}	*reserved*	—	—	—	—
014_{16}	*DAE_invalid_asi*	precise	12.1	H	364
015_{16}	*DAE_privilege_violation*	precise	12.5	H	365
016_{16}	*DAE_nc_page*	precise	12.6	H	365
017_{16}	*DAE_nfo_page*	precise	12.7	H	365
018_{16}-$01F_{16}$	*reserved*	—	—	—	—
020_{16}	*fp_disabled*	precise	8	P	367
021_{16}	*fp_exception_ieee_754*	precise	11.1	P	367
022_{16}	*fp_exception_other*	precise	11.1	P	367
023_{16}	*tag_overflow*	precise	14	P	372
024_{16}	*clean_window*	precise	10.1	P	363
025_{16}-027_{16}	*reserved*	—	—	—	—
028_{16}	*division_by_zero*	precise	15	P	366
029_{16}	*reserved*	—	—	—	—
$02C_{16}$	*reserved*				—
$02D_{16}$	*reserved*	—	—	—	—
$02E_{16}$	*reserved*	—	—	—	—
$02F_{16}$	*reserved*	—	—	—	—
030_{16}	*DAE_side_effect_page*	precise	12.7	H	366
033_{16}	*reserved*	—	—	—	—
034_{16}	*mem_address_not_aligned*	precise	10.2	H	370
035_{16}	*LDDF_mem_address_not_aligned*	precise	10.1	H	369
036_{16}	*STDF_mem_address_not_aligned*	precise	10.1	H	372
037_{16}	*privileged_action*	precise	11.1	H	370
038_{16}	*reserved*	—	—	—	—
039_{16}	*reserved*	—	—	—	—
$03C_{16}$	*reserved*	—	—	—	—
$03D_{16}$	*reserved*	—	—	—	—
041_{16}-$04F_{16}$	*interrupt_level_n* (n = 1-15) (*interrupt_level_15* は *pic_overflow* とも表記される)	disrupting	32-n[i]	P(ie)	369, 370
050_{16}-$05D_{16}$	*reserved*	—	—	—	—
062_{16}	*VA_watchpoint*	precise	11.2	H	372

[i] UA2011 は level15 と pic_overflow で優先度が異なるが、SPARC64™ XIfx ではどちらも 17 とする。

065_{16}-067_{16}	*reserved*	—	—	—	—
069_{16}-$06B_{16}$	*reserved*	—	—	—	—
$06D_{16}$-070_{16}	*reserved*	—	—	—	—
073_{16}	*illegal_action*	precise	8.5	H	369
074_{16}	*control_transfer_instruction*	precise	11.1	P	363
075_{16}	*reserved*	—	—	—	—
078_{16}-$07B_{16}$	*reserved*	—	—	—	—
$07C_{16}$	*cpu_mondo*	disrupting	16.8	P(ie)	364
$07D_{16}$	*dev_mondo*	disrupting	16.11	P(ie)	366
$07E_{16}$	*resumable_error*	disrupting	33.3	P(ie)	371
$07F_{16}$	*nonresumable_error* (not by hardware)		—	—	370
080_{16}-$09C_{16}$	*spill_n_normal* (n = 0-7)	precise	9	P	371
$0A0_{16}$-$0BC_{16}$	*spill_n_other* (n = 0-7)	precise	9	P	371
$0C0_{16}$-$0DC_{16}$	*fill_n_normal* (n = 0-7)	precise	9	P	367
$0E0_{16}$- $0FC_{16}$	*fill_n_other* (n = 0-7)	precise	9	P	367
100_{16}-$17F_{16}$	*trap_instruction*	precise	16.2	P	372

表 12-3 トラップ一覧（優先順位順）

TT	トラップ名	種類	優先順位	トラップ後の特権レベル	定義
$00B_{16}$	*IAE_unauth_access*	precise	2.7	H	368
008_{16}	*IAE_privilege_violation*	precise	3.1	H	368
$00C_{16}$	*IAE_nfo_page*	precise	3.3	H	368
010_{16}	*illegal_instruction*	precise	6.2	H	369
011_{16}	*privileged_opcode*	precise	7	P	371
020_{16}	*fp_disabled*	precise	8	P	367
073_{16}	*illegal_action*	precise	8.5	H	369
080_{16}-$09C_{16}$	*spill_n_normal* (n = 0-7)	precise	9	P	371
$0A0_{16}$-$0BC_{16}$	*spill_n_other* (n = 0-7)	precise	9	P	371
$0C0_{16}$-$0DC_{16}$	*fill_n_normal* (n = 0-7)	precise	9	P	367
$0E0_{16}$- $0FC_{16}$	*fill_n_other* (n = 0-7)	precise	9	P	367
024_{16}	*clean_window*	precise	10.1	P	363
035_{16}	*LDDF_mem_address_not_aligned*	precise	10.1	H	369
036_{16}	*STDF_mem_address_not_aligned*	precise	10.1	H	372
034_{16}	*mem_address_not_aligned*	precise	10.2	H	370
021_{16}	*fp_exception_ieee_754*	precise	11.1	P	367
022_{16}	*fp_exception_other*	precise	11.1	P	367
037_{16}	*privileged_action*	precise	11.1	H	370

074_{16}	*control_transfer_instruction*	precise	11.1	P	363
062_{16}	*VA_watchpoint*	precise	11.2	H	372
014_{16}	*DAE_invalid_asi*	precise	12.1	H	364
015_{16}	*DAE_privilege_violation*	precise	12.5	H	365
016_{16}	*DAE_nc_page*	precise	12.6	H	365
017_{16}	*DAE_nfo_page*	precise	12.7	H	365
030_{16}	*DAE_side_effect_page*	precise	12.7	H	366
023_{16}	*tag_overflow*	precise	14	P	372
028_{16}	*division_by_zero*	precise	15	P	366
100_{16}-$17F_{16}$	*trap_instruction*	precise	16.2	P	372
$07C_{16}$	*cpu_mondo*	disrupting	16.8	P(ie)	364
$07D_{16}$	*dev_mondo*	disrupting	16.11	P(ie)	366
041_{16}-$04F_{16}$	*interrupt_level_n* (n = 1-15) (*interrupt_level_15* は *pic_overflow* とも表記される)	disrupting	32-n[ii]	P(ie)	369, 370
$07E_{16}$	*resumable_error*	disrupting	33.3	P(ie)	371
$07F_{16}$	*nonresumable_error* (not by hardware)		—	—	370

12.3.1. トラップ説明

12.3.1.1. *clean_window*

TT	024_{16} – 027_{16}
優先順位	10.1
トラップ種別	precise
例外検出と特権レベル遷移	priv

SAVE 命令実行時に CLEANWIN = 0 だった場合（新しいレジスタウィンドウに別コンテキストのデータがある場合)、この例外が発生する。

Compatibility Note JPS1, UA2011 ではウィンドウクリーンをハードウェアで行うことも許容しているが(Impl. Dep. #102)、SPARC64™ XIfx では例外を発生させる。

12.3.1.2. *control_transfer_instruction*

[ii] UA2011 は level15 と pic_overflow で優先度が異なるが、SPARC64™ XIfx ではどちらも 17 とする。

TT	074_{16}
優先順位	11.1
トラップ種別	precise
例外検出と 特権レベル遷移	priv

PSTATE.tct = 1 であり、かつ、制御転送命令による分岐が taken である場合に、制御転送命令を完了する代わりにこの例外が発生する。対象となる制御転送命令のケースを下記に記す。

- 条件分岐命令(`Bicc, BPcc, BPr, FBfcc, FBPfcc, Cbcond`)であり、分岐が taken であるケース
- 無条件分岐命令(`BA, BPA, FBA, FBPA`)
- `CALL, JMPL, RETURN` 命令
- `Tcc` 命令が条件を満たし、taken であるケース

TPC[TL], TPC[TL]には制御転送命令実行前の設定が保存される。すなわち、TPC[TL]は制御転送命令の命令アドレスで更新され、TNPC[TL]は制御転送命令実行前の NPC の値で更新される。トラップ通知時 PSTATE.tct は 0 に設定される。例外が非特権モードまたは特権モードで起きた場合、トラップにより特権モードに遷移する。

12.3.1.3. *cpu_mondo*

TT	$07C_{16}$
優先順位	16.8
トラップ種別	disrupting
例外検出と 特権レベル遷移	priv (PSTATE.ie = 1 のとき)

他の VCPU から当該 VCPU にインタラプトがかけられたことを、特権ソフトウェアに通知するためのトラップである。PSTATE.ie = 1 で、CPU_MONDO キューの HEAD と TAIL が一致していないとき、この例外が発生する。

12.3.1.4. *DAE_invalid_asi*

TT	014_{16}
優先順位	12.1
トラップ種別	precise
例外検出と 特権レベル遷移	hpriv

命令と ASI の組み合わせが無効な場合、この例外が発生する。

- load, store, load-store 命令で、未定義 ASI 番号や未定義 VA を指定したとき。
- ASI と命令の組み合わせを間違ったとき。
  - `LDTXA` など。
  - non-translating ASI を `LDXA, LDDFA, STXA, STDFA` 以外の命令で指定した場合。
  - 読み出しのみ可能な ASI に書き込もうとした場合
  - 書き込みのみ可能な ASI を読み出そうとした場合。

命令と ASI の有効な組み合わせの詳細は命令の定義を参照。

12.3.1.5. *DAE_nc_page*

TT	016_{16}
優先順位	12.6
トラップ種別	precise
例外検出と特権レベル遷移	hpriv

ノンキャッシャブル空間に、アトミックロードストア命令、LDTXA, LDBLOCKF, LDDFDS, STDFDS, SIMD ロード命令, SIMD ストア命令でアクセスすると、この例外を検出する。

> **Note** SPARC64™ XIfx で追加された LDDFDS, STDFDS 命令は SIMD 拡張するしない及びアクセスするアドレスが 4 バイト境界もしくは 8 バイト境界に関わらず、ノンキャッシャブル空間にアクセスを行った場合、*DAE_nc_page* 例外を検出する。STPARTIALF 命令では *DAE_nc_page* を検出しない。

12.3.1.6. *DAE_nfo_page*

TT	017_{16}
優先順位	12.7
トラップ種別	precise
例外検出と特権レベル遷移	hpriv

ノンフォールティングロードアクセスのみが許されたページ（TTE.nfo = 1）に、下記の命令以外でアクセスしようとした場合、この例外が発生する。

- ASI_PRIMARY_NO_FAULT{_LITTLE}, ASI_SECONDARY_NO_FAULT{_LITTLE}を指示したロード命令
- PREFETCH, PREFETCHA 命令
- XASR.nf = 1 でのロード命令

つまり、上記 ASI 以外もしくは XASR.nf = 0 でのロード命令や、ASI が何であるかには係わりなくストア命令やアトミックロードストア命令、および FLUSH 命令でアクセスにより、この例外が発生する。

> **Note** ストア命令、アトミックロードストア命令で ASI_PRIMARY_NO_FAULT{_LITTLE}, ASI_SECONDARY_NO_FAULT{_LITTLE}を指示した場合、*DAE_invalid_asi* が検出される。

12.3.1.7. *DAE_privilege_violation*

TT	015_{16}
優先順位	12.5
トラップ種別	precise
例外検出と 特権レベル遷移	hpriv

特権モードのみアクセスが許されたページ（TTE.p = 1）に、非特権モードにより、ロード命令、ストア命令、アトミックロードストア命令でアクセスした場合、この例外が発生する。FLUSH, PREFETCH{A}では *DAE_privilege_violation* 例外は発生しない。

特権モードで、ASI_*_AS_IF_USER_PRIMARY{_LITTLE}, ASI_*_AS_IF_SECONDARY{_LITTLE}を指示してアクセスした場合にも *DAE_privilege_violation* 例外が発生する。

12.3.1.8. *DAE_side_effect_page*

TT	030_{16}
優先順位	12.7
トラップ種別	precise
例外検出と 特権レベル遷移	hpriv

副作用があるページ（TTE.e = 1）に、ASI_PRIMARY_NO_FAULT{_LITTLE}, ASI_SECONDARY_NO_FAULT{_LITTLE}を指示したロード命令もしくは XASR.nf = 1 でロード命令によりアクセスした場合、この例外が発生する。

12.3.1.9. *dev_mondo*

TT	$07D_{16}$
優先順位	16.11
トラップ種別	disrupting
例外検出と 特権レベル遷移	priv (PSTATE.ie = 1 のとき)

I/O デバイスから当該 VCPU にインタラプトがかけられたことを、特権ソフトウェアに通知するためのトラップである。PSTATE.ie = 1 で、DEV_MONDO キューの HEAD と TAIL が一致していないとき、この例外が発生する。

12.3.1.10. *division_by_zero*

TT	028_{16}
優先順位	15
トラップ種別	precise
例外検出と特権レベル遷移	priv

整数除算において、0 で除算しようとした場合、この例外が発生する。

12.3.1.11. *fill_n_normal, fill_n_other*

TT	$0C0_{16}$, $0C4_{16}$, $0C8_{16}$, $0CC_{16}$, $0D0_{16}$, $0D4_{16}$, $0D8_{16}$, $0DC_{16}$, $0E0_{16}$, $0E4_{16}$, $0E8_{16}$, $0EC_{16}$, $0F0_{16}$, $0F4_{16}$, $0F8_{16}$, $0FC_{16}$
優先順位	9
トラップ種別	precise
例外検出と特権レベル遷移	priv

RESTORE, RETURN 命令実行時に、ウィンドウレジスタの内容をメモリから復元する必要があるとき、この例外が発生する。

12.3.1.12. *fp_disabled*

TT	020_{16}
優先順位	8
トラップ種別	precise
例外検出と特権レベル遷移	priv

浮動小数点ユニット（FPU）が無効な状態（PSTATE.pef = 0 または FPRS.fef = 0）で浮動小数点演算、浮動小数点レジスタへのロードやストア、fcc による条件分岐命令を実行した場合、この例外が発生する。

12.3.1.13. *fp_exception_ieee_754*

TT	021_{16}
優先順位	11.1
トラップ種別	precise
例外検出と特権レベル遷移	priv

浮動小数点演算により IEEE 754 で定義されている例外を検出し、その例外がマスクされていないとき、この例外が発生する。例外のマスクに関しては FSR を参照 (23 ページ)。

12.3.1.14. *fp_exception_other*

TT	022_{16}
優先順位	11.1
トラップ種別	precise
例外検出と特権レベル遷移	priv

浮動小数点演算により IEEE 754 で定義されている例外以外の例外を検出した場合、この例外が発生する。Section 8, “IEEE Std. 754-1985 Requirements for SPARC-V9” (313 ページ) を参照。

12.3.1.16. *IAE_nfo_page*

TT	$00C_{16}$
優先順位	3.3
トラップ種別	precise
例外検出と特権レベル遷移	hpriv

ノンフォールティングロードアクセスのみが許されたページ（TTE.nfo = 1）に命令フェッチをした場合、この例外が発生する。

12.3.1.17. *IAE_privilege_violation*

TT	008_{16}
優先順位	3.1
トラップ種別	precise
例外検出と特権レベル遷移	hpriv

特権モードのみアクセスが許されたページ（TTE.p = 1）に、非特権モードで命令フェッチをした場合、この例外が発生する。

また、非特権モードかつ TL > 0 で命令フェッチをした場合も、この例外が発生する。このとき優先順位は表 12-3 と異なる。

12.3.1.18. *IAE_unauth_access*

TT	$00B_{16}$
優先順位	2.7
トラップ種別	precise
例外検出と特権レベル遷移	hpriv

命令実行可能とされていないページ（TTE.ep = 0）に命令フェッチをした場合、この例外が発生する。

12.3.1.19. *illegal_action*

TT	073_{16}
優先順位	8.5
トラップ種別	precise
例外検出と 特権レベル遷移	hpriv

実行しようとする命令が XAR 対象外の命令だが、XAR.v = 1 が指定されている場合、または、実行しようとする命令が XAR 対象命令だが XAR の指定が間違っている場合に通知される。SXAR で XAR をセットする場合は、後続の命令を実行しようとした時点で通知される。

WRASR(154 ページ), FSHIFTORX(160 ページ)及び FEPERMD (144 ページ)では優先順位の高い *illegal_instruction* ではなく *illegal_action* が通知されることがある。詳細は各命令の説明参照。

12.3.1.20. *illegal_instruction*

TT	010_{16}
優先順位	6.2
トラップ種別	precise
例外検出と 特権レベル遷移	hpriv

ILLTRAP 命令を実行した場合や、未実装のオペコード、*reserved* フィールドが 0 でない命令を実行した場合、この例外が発生する。

12.3.1.21. *interrupt_level_n* (n = 1 – 15)

TT	041_{16} – $04F_{16}$
優先順位	17 – 31 (32 – n)
トラップ種別	disrupting
例外検出と 特権レベル遷移	priv (PSTATE.ie = 1 かつ PIL < n のとき)

PSTATE.ie = 1 かつ PIL < n である n に対応する SOFTINT<n>に 1 がセットされているとき、n に対応する *interrupt_level_n* 例外が発生する。

Interrupt_level_14 は、PIL < 14 で SOFTINT.sm = 1 または SOFTINT.tm = 1 でも発生する。

Interrupt_level_15 は、PA カウンタのオーバーフローを通知するための例外 *pic_overflow* と番号を共有している。

12.3.1.22. *LDDF_mem_address_not_aligned*

TT	035_{16}
優先順位	10.1
トラップ種別	precise
例外検出と特権レベル遷移	hpriv

Non-SIMD の LDDF, LDDFA または LDDFID 命令で、アクセスするアドレスが 4 バイトアラインだが 8 バイトアラインではない場合、この例外が発生する。

12.3.1.23. *mem_address_not_aligned*

TT	034_{16}
優先順位	10.2
トラップ種別	precise
例外検出と特権レベル遷移	hpriv

メモリアクセス命令を実行時、必要とされるアラインメントに違反しているとき、または、JMPL, RETURN 命令の分岐先が 4 バイトアラインメントでない場合、この例外が発生する。

12.3.1.24. *nonresumable_error*

TT	$07F_{16}$
優先順位	—
トラップ種別	—
例外検出と特権レベル遷移	検出しない

この例外はハードウェアが発生させるものではない。

12.3.1.25. *PIC_overflow*

TT	$04F_{16}$
優先順位	17
トラップ種別	disrupting
例外検出と特権レベル遷移	priv (PSTATE.ie = 1 かつ PIL < 15 のとき)

PA カウンタのオーバーフローが起き、オーバーフローによる例外がマスクされていないとき、この例外が発生する。SPARC64™ XIfx では *interrupt_level_15* (369 ページ)と同じ。

12.3.1.26. *privileged_action*

TT	037_{16}
優先順位	11.1
トラップ種別	precise
例外検出と特権レベル遷移	hpriv

特権モードまたはハイパーバイザモードに許された動作を非特権モード（PSTATE.priv = 0）で行おうとした場合、また、ハイパーバイザモードのみに許された動作を行おうとした場合この例外が発生する。

privileged_action は、命令語と PSTATE だけでは判別できない特権レベル違反に対して発生する。例えば

- ある特権レベルでは使用できない ASI 番号を使おうとした
- 動作時の設定により非特権モードでのアクセスが禁止されているレジスタ(TICK, STICK, PIC, PCR など)にアクセスしようとした

などである。

12.3.1.27. *privileged_opcode*

TT	011_{16}
優先順位	7
トラップ種別	precise
例外検出と特権レベル遷移	priv

オペランドや ASI 番号によらず、特権モードでのみ実行可能な命令を、非特権モード（PSTATE.priv = 0）で実行しようとした場合、この例外が発生する。

privileged_opcode は、命令語と PSTATE.priv だけで判別できる特権レベル違反を検出する。特例として、命令語として正しいが TL = 0 では実行できない命令を非特権モードで実行したことも検出する。

12.3.1.29. *resumable_error*

TT	$07E_{16}$
優先順位	33.3
トラップ種別	disrupting
例外検出と特権レベル遷移	priv (PSTATE.ie = 1 のとき)

特権モードのソフトウェアに対して、エラーが起きたことと、そのエラーは命令の実行継続を妨げるような性質のものではないことを通知する。PSTATE.ie = 1 で、RESUMABLE_ERROR キューの HEAD と TAIL が一致していないとき、この例外が発生する。

12.3.1.30. *spill_n_normal, spill_n_other*

TT	080_{16}, 084_{16}, 088_{16}, $08C_{16}$, 090_{16}, 094_{16}, 098_{16}, $09C_{16}$, $0A0_{16}$, $0A4_{16}$, $0A8_{16}$, $0AC_{16}$, $0A0_{16}$, $0A4_{16}$, $0A8_{16}$, $0AC_{16}$
優先順位	9
トラップ種別	precise
例外検出と特権レベル遷移	priv

SAVE, FLUSHW 命令実行時に、ウィンドウレジスタの内容をメモリに退避する必要があるとき、この例外が発生する。

12.3.1.31. *STDF_mem_address_not_aligned*

TT	036_{16}
優先順位	10.1
トラップ種別	precise
例外検出と特権レベル遷移	hpriv

Non-SIMD の STDF, STDFA, STDFR, STDFID, STDFRID 命令で、アクセスするアドレスが 4 バイトアラインだが 8 バイトアラインではない場合、この例外が発生する。

12.3.1.32. *tag_overflow*

TT	023_{16}
優先順位	14
トラップ種別	precise
例外検出と特権レベル遷移	priv

TADDccTV, TSUBccTV 命令で、入力オペランドの下位 2 ビットが 0 でないか、演算によりオーバーフローが発生したとき、この例外が発生する。

12.3.1.33. *trap_instruction*

TT	100_{16} – $17F_{16}$
優先順位	16.2
トラップ種別	precise
例外検出と特権レベル遷移	priv

トラップ番号が 00_{16} – $7F_{16}$ である Tcc 命令を実行し、条件が成立している場合、この例外が発生する。

12.3.1.34. *VA_watchpoint*

TT	062_{16}
優先順位	11.2
トラップ種別	precise
例外検出と特権レベル遷移	hpriv

メモリアクセス命令が、VA watchpoint で指定された VA にアクセスすると、この例外が発生する。

12.3.3. 優先順位の特例

複数の例外が発生しうる状況では、基本的には表 12-3 の優先順位の高い例外が選択されトラップが通知されるが、例外的に優先順位の低い例外が選択される場合がある。以下に、優先順位に関する特例を示す。

- *illegal_action* の優先順位は 8.5 だが、優先順位 6.2 の *illegal_instruction* より優先して通知される場合がある。詳細は WRASR (154 ページ) , FSHIFTORX(160 ページ)及び FEPERMD (144 ページ)を参照。
- 命令語として正しいが TL = 0 では実行できない命令を非特権モードで実行すると、*illegal_instruction* ではなく *privileged_opcode* 例外を検出する。

13. Memory Management Unit

13.1. アドレス体系

SPARC64™ XIfx では 2 種類のアドレスを定義する。

- VA (Virtual Address: 仮想アドレス) — ページ単位のアクセス保護が可能なアドレス体系。SPARC64™ XIfx の VA は 64 ビット幅であり、全 64 ビット有効（VA hole なし)。コンテキスト番号で識別する。
- RA (Real Address: 実アドレス) — VM 用に仮想化された物理アドレス。SPARC64™ XIfx では、RA はソフトウェアからは全 64 ビットが有効だが、real range. レジスタで RA を underlying hardware address に変換する際は、下位 41 ビットのみが有効である。

表 13-1　SPARC64™ XIfx のアドレス特性

	VA	RA
ビット幅	64 ビット	64 ビット
有効ビット	64 ビット（VA hole なし）	41 ビット

13.1.1. アドレス変換

アドレス変換には、VA-RA がある。CPU がアドレス変換を行うためには、変換の情報が TLB に登録されている必要がある。TLB への登録は、特権ソフトウェアが行う。

13.1.1.1. VA-RA変換

VA-RA 変換情報は特権ソフトウェアが管理し、TSB で提供する。

13.2. アドレス変換

MMU は、非特権モードおよび特権モードのときは常時使われる。

13.3. TSB (Translation Storage Buffer)

TSB はキャッシャブル空間のメモリ上にある TTE (Translation Table Entry) の配列である。

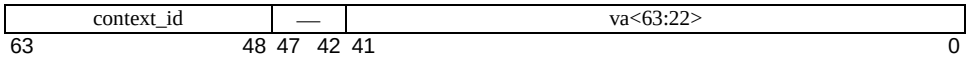
TSB の情報は、TLB ミス時に検索される。検索はソフトウェアが行う

VA-RA の TTE を格納する TSB は、特権ソフトウェアが直接参照・更新することができる。

13.4. TSB TTE (Translation Table Entry)

TSB TTE は VA-RA 変換情報を保持するデータ形式である。

TTE Tag



TTE Data

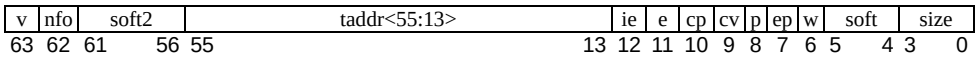


表 13-2　TSB TTE

ビット	フィールド	説明
Tag 63:48	context_id	コンテキスト番号
Tag 41:0	va<63:22>	VA。VA<21:13>の 9 ビットは TSB のインデクスで指示する（TSB のエントリ数は 2 のベキで、最小エントリ数は $2^9 = 512$）
Data 63	v	有効ビット。このビットが 0 のとき、タグとデータの他のビットは意味を持たない。
Data 62	nfo	non-fault only。このビットが 1 のページは、ノンフォールティングロードでのみアクセス可能。
Data 61:56	soft2	ハードウェアはこのビットを使用しない。
Data 55:13	taddr<55:13>	RA。 SPARC64™ XIfx では、 ・RA の場合、taddr<55:41>が 0 でない TTE があると 例外を発生させる。
Data 12	ie	ASI および PSTATE.cle で指定されているエンディアンを反転して使用する。IMMU ではこのフィールドは無視される。
Data 11	e	副作用ありページ。e に 1 がセットされていると、 • ノンフォールティングロードには例外が通知される。 • ノンキャッシャブル領域へのアクセスはストロングオーダで処理される。 • ノンキャッシャブル領域へのストアはマージされない。 このビットは、I/O デバイスの副作用があるレジスタをマッピングするときなどにセットする必要がある。 IMMU ではこのフィールドは無視される。 **Note**　e ビットはノンキャッシャブル領域へのアクセスを制御するために使用される。キャッシャブル領域の e ビットに 1 をセットされたときの動作は未定義である。
Data 10	cp	物理アドレスインデクスキャッシュにキャッシュするかどうか。 SPARC64™ XIfx はこの情報を使わない。
Data 9	cv	仮想アドレスインデクスキャッシュにキャッシュするかどうか。 SPARC64™ XIfx はこの情報を使わない。
Data 8	p	特権ページ。このビットが 1 のページは priv モードでのみアクセス可能。
Data 7	ep	実行可能ページ。このビットが 1 であるページのデータは、命令として実行可能。
Data 6	w	書き込み可能ページ。このビットが 1 であるページはストアセマンティクスのアクセスが可能。
Data 5:4	soft	ハードウェアはこのビットを使用しない。

Data 3:0	size	ページサイズ。SPARC64™ XIfx では以下の 4 種類が指定できる。 **Size<3:0>** / **ページサイズ** 0000_2 / 8KB 0001_2 / *reserved* 0010_2 / 512KB 0011_2 / 4MB 0100_2 / 32MB 0101_2 / *reserved* 0110_2-1111_2 / *reserved* サポートされているページの詳細については 13.7 参照。

Compatibility Note SPARC64™ VIIIfx, SPARC64™ IXfx では、8KB, 64KB, 512KB, 4MB, 32MB, 256MB, 2GB をサポートしていた。

13.5. コンテキスト

複数の仮想アドレス空間を識別するための識別子をコンテキスト番号と呼ぶ。SPARC64™ XIfx ではコンテキスト番号は 16 ビットの符号なし整数である。

Compatibility Note VA のページ番号が 51 ビットなので､JPS1 ではコンテキスト番号を 13 ビットとして 64 ビットに収めていた。UA2011 仕様では最大 16 ビットである。

コンテキストレジスタには primary, secondary, nucleus の 3 種類がある。命令フェッチ、データアクセス時にどのコンテキストレジスタを使うかは、特権レベル、TL の値、使用する ASI の組み合わせで決まる。

表 13-3 コンテキストレジスタの選択

特権レベル	TL	ASI	命令フェッチ	データアクセス	XASR.nf
user	0	—	primary	primary	有効
		`ASI_PRIMARY*`, `ASI_{PST*|FL*|XFILL|XFILL256}_PRIMARY*`, `ASI_{BLOCK|TWINX|STBI}_PRIMARY*`, `ASI_BLOCK_COMMIT_PRIMARY*`	指定できない	primary	有効
		`ASI_SECONDARY*`, `ASI_{PST*|FL*|XFILL|XFILL256}_SECONDARY*`, `ASI_{BLOCK|TWINX|STBI}_PRIMARY*`, `ASI_BLOCK_COMMIT_SECONDARY*`	指定できない	secondary	有効

primary, secondary コンテキストはそれぞれ 2 つずつ、primary 0, primary 1, secondary 0, secondary 1 がある。アドレス変換時は、primary 0 と primary 1 どちらかと一致すれば primary コンテキストは一致とみなす。secondary も同様。

Compatibility Note SPARC64™ IXfx の共有コンテキスト仕様は継承しない。

Non-Faulting Mode(XASR.nf = 1)は TL = 0 のとき特権レベルに関わらず有効である。

13.6. パーティション番号

複数の RA 空間（=複数の VM）を識別するための識別子をパーティション番号と呼ぶ。識別子のビット長はサポート可能最大 VM 数ではなく、TLB を共有しているコア内の VCPU 間で識別が可能なだけのビット幅とする。

Programming Note ひとつの VCPU で複数の VM を時分割で実行するときは、パーティション番号を指定してデマップし、再利用する。

13.7. ページ

仮想アドレスから RA への変換はページ単位で行われる。VA の上位 m ビットをページ番号、下位 n ビットをページ内オフセットとし(m + n = 64)、上位 m ビットを仮想ページ番号、下位 n ビットをページ内オフセットと呼ぶ。ページの種類(m と n の組合わせ)は複数考えられる。ページ種類を区別するために、ページ内オフセットで表現できるバイト量が使われる。
SPARC64™ XIfx で使えるページの種類は、8KB, 512KB, 4MB, 32MB の 4 種類である。

表 13-4 SPARC64™ XIfx がサポートするページ種類

ページ種類	仮想ページ番号(m)	ページ内オフセット(n)	エンコード
8KB ページ	51 ビット	13 ビット	000_2
512KB ページ	45 ビット	19 ビット	010_2
4MB ページ	42 ビット	22 ビット	011_2
32MB ページ	39 ビット	25 ビット	100_2

13.8. ソフトウェアでの TSB, TLB 処理

ソフトウェアは TSB を更新する際、更新しようとする TSB TTE と同じ VA を変換するようなエントリが、TLB に載っていないことを保証すること。TLB に載っているかどうか保証されていない場合は VA Demap を行うこと。

14. Hardware Barrier

複数の VCPU が協調してひとつのジョブを処理するとき、高性能を実現する上で重要な要素のひとつは、VCPU 間の同期を高速に取ることである。SPARC64™ XIfx は、ハードウェアによるバリアアシスト機構を提供しており高速な同期を実現することができる。

14.1. バリアの種類

SPARC64™ XIfx は 2 種類のバリアを提供する。ひとつは複数の VCPU 間で同期を取るための同期用バリアである。もうひとつは 2 つの VCPU 間の同期に特化した post-wait 用バリアである。この節では、どのようなソフトウェアに SPARC64™ XIfx のハードウェアバリア機構が有効であるかを解説する。

14.1.1. 同期用バリア

同期用バリアは、プログラム内の並列実行部の同期を取る際に使用されるバリアである。プログラム内に並列に実行できる部分が複数あり、ある並列実行部の実行が確実に終了してから次の並列実行部の処理を開始しなくてはいけないような場合、その 2 つの並列実行部の間で全 VCPU を同期させる必要が出てくる。このような場合に、並列実行部の間で全 VCPU の実行を待ち合わせるために同期用バリアが使われる。

同期用バリアは、並列実行部と 1 VCPU で実行する部分（逐次実行部）の間のバリアにも使われる。並列実行部が終了すると、並列実行を担当していた VCPU の役割は終了するが、次の並列実行部の実行を迅速に開始するために、同期成立待ちにしておく。こうすることで、逐次実行部の終了を同期の成立という形で他の VCPU に伝えることができ、次の並列実行部の処理を開始することができる。

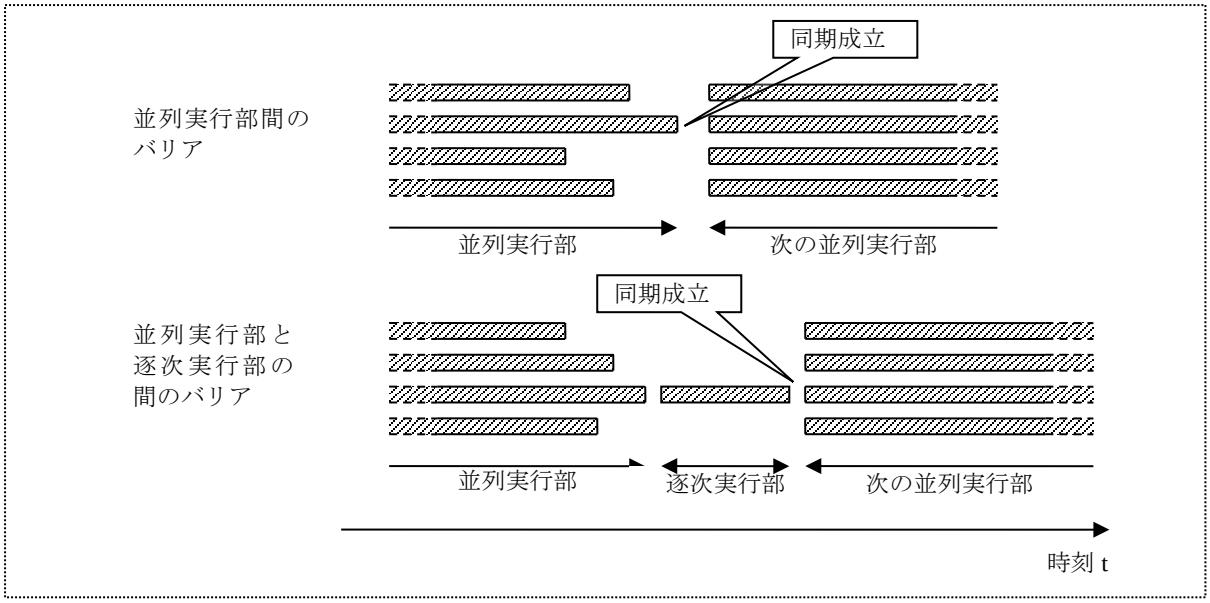


図 14-1　同期用バリア

同期の成立後、次の同期に備えて同期機構を再初期化する必要がある。ソフトウェアによる同期バリアでは、特に並列実行部間の同期に使われる場合、再初期化が並列実行部の実行開始よりも時間的に早く行われることが保証されるよう、慎重に設計しなければならない。ハードウェアによる同期機構では、同期成立と再初期化を高速かつ不可分に行うよう設計することができる。

14.1.2. post-wait 用バリア

post-wait とは、2 スレッド間で実行の同期を取るための機構である。ある処理を行っているスレッドと、その処理が終わるのを待っている別のスレッドがあるとき、待ち側のスレッドは post-wait 用バリアの wait 操作で待っておき、実行側のスレッドは処理が終わると post-wait 用バリアの post 操作を行うことで、待ち側のスレッドに実行が終了したことを伝える。実行側のスレッドは待ち合わせ側のスレッドとは独立して実行を継続する。

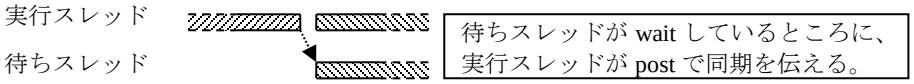
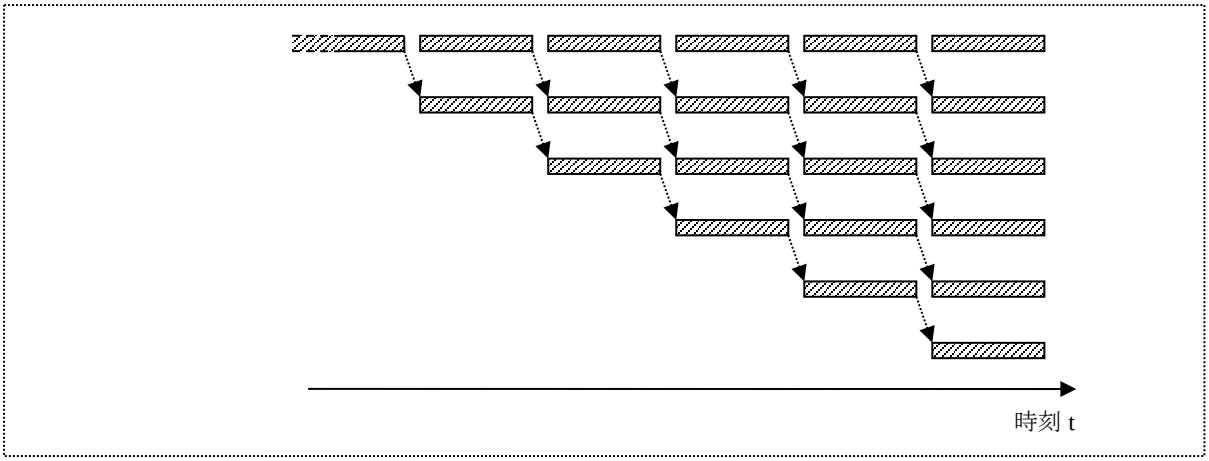


図 14-2　post-wait 同期

図 14-2 のような一回限りの同期通知では、post するスレッドは待ち側のスレッドが wait しているかどうかを気にする必要はないが、ループ内に部分的に依存がある場合の並列処理に使う場合は、post 側と wait 側のハンドシェークが必要になる。



14.2. SPARC64™ XIfx の同期機構

ハードウェアバリアの実体は Barrier Blade (BB) と呼ばれる資源である。BB は複数の VCPU で共有されており、BST（Barrier Status bit）や BST のマスクビット、さらに前回同期したときの値を記憶しておく LBSY（Last Barrier Synchronization status）といったフィールドを保持している。BB を共有している VCPU の同期は、BB ひとつだけで実現できる。BB を共有していない異なるバリアバンク間、すなわち異なるコアメモリグループ(CMG)間で同期を取るためには、複数の BB による木構造を構成する必要がある。この節では、SPARC64™ XIfx の同期機構の概念と木構造を構成する際に必要な概念を説明する。

14.2.1. バリア資源

ハードウェアバリアの設定や実際の同期機構などをバリア資源と呼ぶ。バリア資源は VCPU で独立しているものと CMG 内で共有しているものがある。

SPARC64™ XIfx において、バリア資源に直接アクセスすることができない。バリア資源にアクセスするためには、バリア資源の使用権が与えられた VCPU の特権アクセスレジスタや窓レジスタを通して行う。バリア資源の使用権は VCPU 単位で制御され、使用権がない VCPU においては特権アクセスレジスタや窓レジスタを通してバリアへの各種操作を行うことはできない。

Programming Note　SPARC64™ XIfx においてバリア同期に参加できない VCPU（アシスタントコア）が存在するが、使用権はそれに関係なく設定される。バリア資源の使用権があるがバリア同期に参加できない VCPU において、バリアの初期化などの操作は可能である。

14.2.2. バリア同期の考え方

ハードウェアは、BB で同期を制御する。同期の成立と再初期化は不可分に行われる。

ユーザプログラムは、同期を 1 ビットの情報で扱う。同期が成立するのは、BB を使う全スレッドの 1 ビット情報が同じ値になったときである。

同期の情報も BB から取得することができる。BB は前回同期が成立したときの 1 ビット情報を持っている。ユーザプログラムが BB を操作する際は、前回の同期情報を読み出し、それを反転させた値を BB に書き込み、前回の同期情報が更新するのを待てばよい。

14.2.3. バリアバンク

バリアバンクとは、BB を共有する VCPU の集合である。SPARC64™ XIfx のバリアバンクは CMG 内の 16 個の VCPU と、8 個のバリア同期用 BB、および 17 個の post-wait 用 BB で構成される。一つの CMG は一つのバリアバンクに対応しており、バリアバンク内の VCPU は、共有するひとつの BB でバリア同期、post-wait 同期を取ることができる。

一方､異なるバリアバンクに属する VCPU は BB を共有していないが､SPARC64™ XIfx では、BB で同期用の木（同期木）を構成することで、バリア同期、post-wait 同期を取ることができる。

14.2.4. 同期木

SPARC64™ XIfx では同期木を構成するということは、2 つの CMG 間で同期を行うということと同義である。同期木を構成する場合、他方の CMG に同期木識別子が配送される

同期木は異なるバリアバンクに所属する複数の BB で構成される。一般に木構造と言うと最下層のノードで情報を集約し上層のノードに伝達する構造だが、SPARC64™ XIfx の同期木は、下層用 BB、上層用 BB と用途を分けた BB を用意するのではなく、各 BB の内部に下層用と上層用の情報を全て持たせている。BB はシステムで一意の同期木識別子を割り当てられ、同一の識別子を持つ BB 同士の各層が組み合わされ、全体としてひとつの同期木を構成する。

SPARC64™ XIfx では同期木の情報は他のバリアバンクを含めた各 BB に分散されている。BB 間で情報を共有するためバリアバンク間で同期木識別子を配送し､同じ同期木識別子を持つ BB がそれを受け取る。

14.2.5. Barrier Blade (BB)

BB には同期用 BB と post-wait 用 BB の 2 種類がある。同期用 BB は Share, Bottom, Top の 3 つの部分からなる。post-wait 用 BB は Share と Bottom の 2 つの部分からなる。

14.2.5.1. Share部

Share 部は、BB 全体を制御する情報を保持している。同期用 BB、post-wait 用 BB のどちらも、Share 部に持っている情報は同じである。

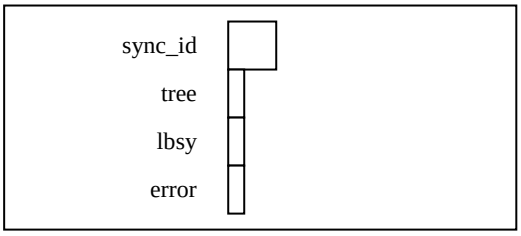


図 14-3 BB (Share)

lbsy は、BB が最後に同期したときの同期情報を保持しているフィールドで、非特権ソフトウェアが窓 ASI を通して読み出すことができる。

tree は、BB が同期木を構成するかどうかを表わす。tree が 1 のときは、sync_id は同期木の識別子を表わす。

error は、ハードウェアが異常動作を検出したことを表わす。同期木の情報は各 BB に分散して存在しているので、全体で一意性を持った状態にするのはソフトウェアの責任である。ハード

ウェアは、正しく木が構成されていれば起こらないはずの事象を検出した場合に、error に 1 をセットする。error = 1 である BB は、その後同期動作を行わなくなる。

14.2.5.2. Bottom部

Bottom 部には、同期木を構成する際に最下層となる情報が保持される。BB をバリアバンク内の同期用に使う場合は、Bottom 部と Share 部だけが使われる。

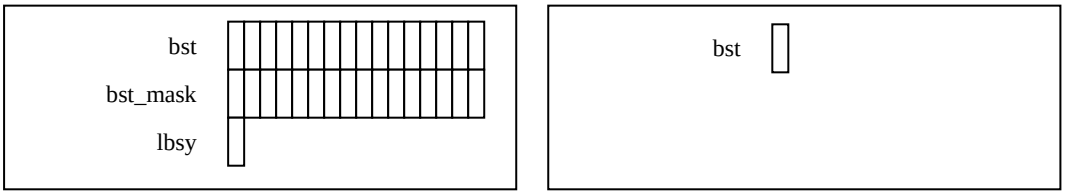


図 14-4 BB (Bottom) (左はバリア用、右は post-wait 用)

バリア用 BB

バリア用 BB では、bst, bst_mask は、バリアバンク内の各 VCPU に対応しているビットである。bst と bst_mask の同一ビットが対になって、対応する VCPU の同期状態を表わす。i 番の VCPU について、bst_mask<i>が 1 ならば、その VCPU は同期に参加しており、bst<i>が同期状態を表わす。bst_mask<i>が 0 ならば同期には参加しておらず、bst<i>は意味を持たない。同期状態は 1 ビットの情報である。

lbsy は最後に同期が成立したときの同期状態を記憶するフィールドである。同期状態は 1 ビットの情報なので lbsy も 1 ビットであり、同期が成立する度に、0 から 1 へ、1 から 0 へと変化する。同期情報を 1 ビットにすることで、カウンタ方式ならば必要な同期成立によるカウンタ再設定が不要となっている。

バリアバンク内でバリア同期を取るためには、bst, bst_mask, および lbsy があればよい。Share 部の tree ビットが 0 の場合、その BB はバリアバンク内の同期用に使われることになる。バリアバンク内の同期用の BB では、Bottom 部の lbsy と Share 部の lbsy は常に一致していることが保証される。

Share 部の tree ビットが 1 の場合、その BB は同期木の一部になっていることを意味する。この場合、Bottom 部の lbsy が更新されると、同期木の sync_id が他のバリアバンクに配送される。

post-wait 用 BB

post-wait 用 BB では、bst は 1 ビットで bst_mask, lbsy はない。バリアバンク内の全 VCPU が bst<0>に対応している。bst が変化すると、その値が Share 部の lbsy にコピーされる。Share 部の tree ビットが 1 の場合、その BB は同期木の一部なので、同期木の sync_id が他のバリアバンクに配送される。

14.2.5.3. Top部

Top 部は、同期木を構成する際の最上位層の情報が保持される。post-wait 同期では最上位層は必要ないので、post-wait 用 BB には Top 部はない。また、Share 部の tree が 0 の場合は、Top 部の情報に意味はない。

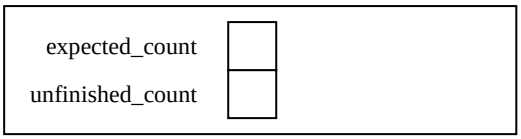


図 14-5　BB (Top)

expected_count は、この BB の Top 部が受信するブロードキャストの期待値を、unfinished_count はあといくつブロードキャストを受け取ると Top 部の同期が成立するかを表わす。unfinished_count が 0 になると、Top 部の同期が成立し、すなわち同期木全体の同期が成立したことになる。Top 部の同期が成立すると、Share 部の lbsy が反転し、unfinished_count には expected_count の値が設定される。同期成立と lbsy の反転、および unfinished_count の再設定はアトミックに行われる。

Programming Note　同期の成立時、Share 部の lbsy の値には Bottom 部の lbsy がコピーされるのではなく、Share 部の lbsy を反転させることに注意。ソフトウェアは、Share 部と Bottom 部で矛盾が起きないようにする必要がある。

Programming Note　SPARC64™ XIfx では、expected_count は 2 以外の値を取りえない。

14.2.6. バリア資源の操作

バリア資源の使用権が与えられた VCPU では、特権アクセスレジスタ及び窓レジスタを介してバリア資源へのアクセスが可能となる。特権アクセスレジスタと窓レジスタは ASI に定義されており、非特権ソフトウェアおよび特権ソフトウェアは、これらの ASI を通してバリアを操作する。

特権ソフトウェア

使用権が与えられた VCPU において特権ソフトウェアは、特権アクセスレジスタ及び窓レジスタを介し、同一 CMG 内のすべてのバリア資源の操作が可能である。特権ソフトウェアは、非特権ソフトウェアに対して窓レジスタを介した操作を許可するかの制御も行うことができる。

非特権ソフトウェア

特権ソフトウェアからは特権アクセスレジスタを介してすべてのバリア資源の操作が可能でるが、非特権ソフトウェアが操作できるのは、1 ビットの同期情報の読み出しと書き込みのみである。

ユーザプログラムから見ると、窓の VA と 1 ビットの同期情報がバリアのすべてである。BST のビット位置やどのバリアバンクに属しているか、同期木が使われているかどうか、などの情報はすべて隠蔽されており、これらの関係を意識する必要はない。

14.3. バリア構成方法

14.3.1. バリアバンク内同期

バリアバンク内の同期は、BB1 つで実現できる。同期木を構成する必要はないので、share.tree を 0 に設定する。Top 部は使われない。share.tree = 0 なので、share.sync_id も使われない。これらの使われないフィールドには、どのような値が設定されていても、動作には影響しない。

BB の同期情報は 1 ビットの情報である。BB は、Share 部に前回バリア同期が成立した際の同期情報 share.lbsy を、Bottom 部に各 VCPU の同期情報 bst を持っている。同期を取るためには、share.lbsy を読み出し、それを反転させた値を bst にセットすればよい。

bst, bst_mask は、バリアバンク内の VCPU と 1 対 1 に対応したビットマップになっている。ある VCPU がバリア同期用 BB を使用する場合、bst_mask の対応するビットを 1 に設定しておく。bst の対応するビットは、VCPU の同期状態を表している。バリア同期に参加している全 VCPU の同期情報が揃うと、BB 全体の同期が成立し、その値で bottom.lbsy および share.lbsy が更新される。つまり

- (bst **and** bst_mask) = 0 ならば、bottom.lbsy, share.lbsy に 0 がセットされる。
- (bst **and** bst_mask) = bst_mask ならば、bottom.lbsy, share.lbsy に 1 がセットされる。

となる（ただし bst_mask の全ビットが 0 の場合を除く）。post-wait 用 BB では便宜上 bst_mask<0>があると考え、bottom.lbsy を無視すれば、やはり上の式が適用できる。

ユーザプログラムがバリア同期を取るときは、窓を通して share.lbsy を読み出し、窓を通して bst を更新する。ユーザプログラムは bst のビット位置を知る必要はない。

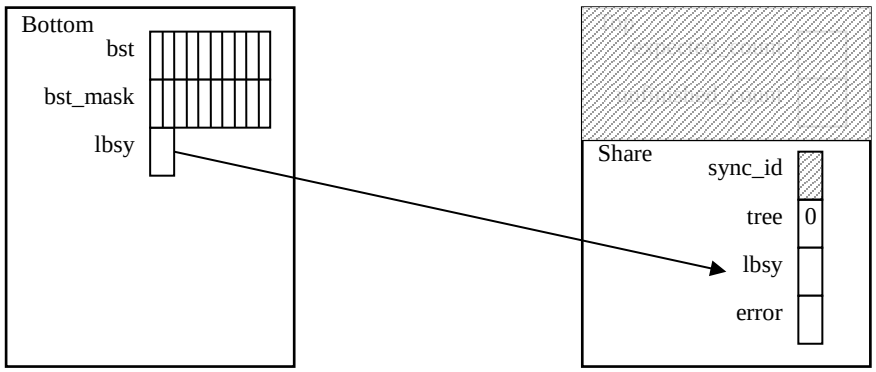


図 14-6 バリアバンク内の同期（バリア用 BB）

post-wait 用 BB の場合、BB は最後に同期した値を記憶していると考えるのではなく、相手側スレッドの状態を表していると考えるとよい。2 スレッド間でのシェークハンドを行うには、post 側の VCPU は lbsy が 0 になるのを待って bst に 1 を書き、wait 側の VCPU は lbsy が 1 になるのを待って bst に 0 を書く。

14.3.2. バリアバンク間同期

14.3.2.1. CMG間の同期木

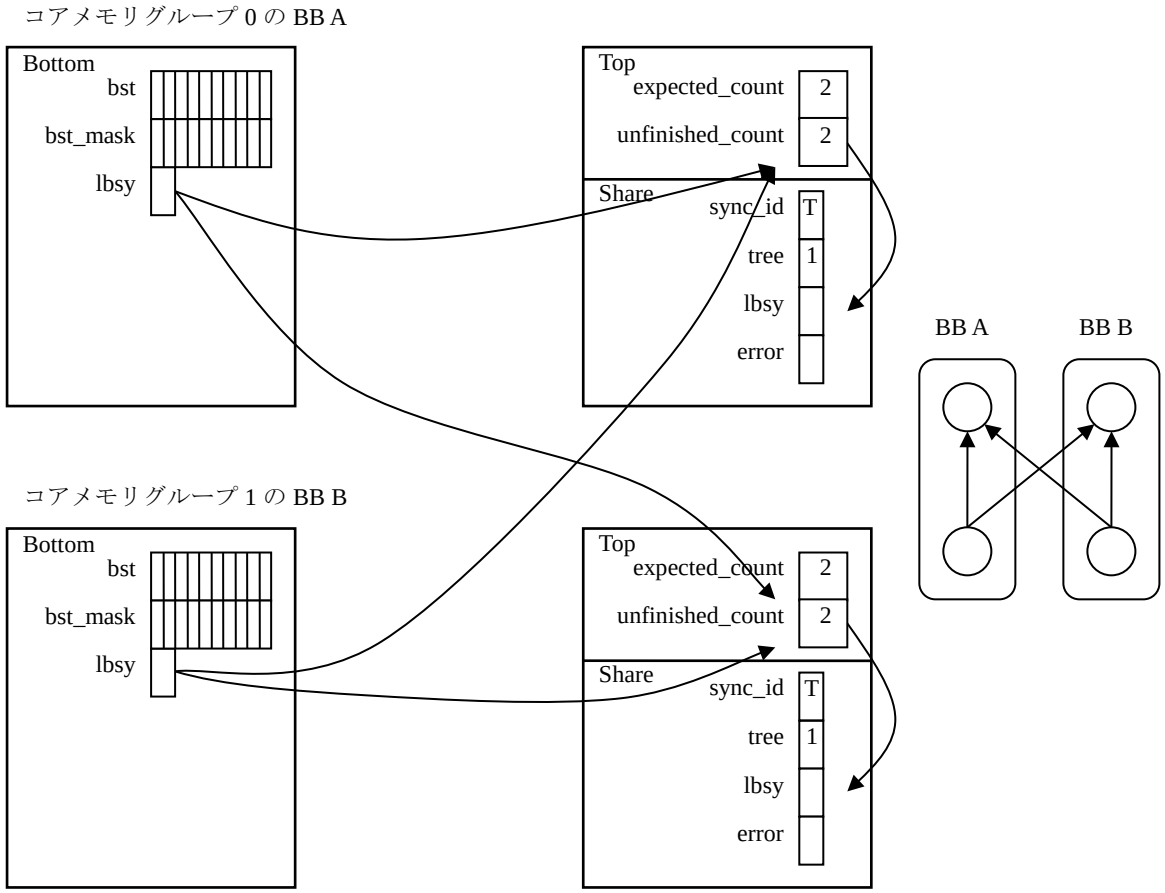


図 14-7 CMG 間の同期木（バリア BB）

図 14-7 は CMG 間のバリア同期木の動作を説明する図である。

同期木によるバリア同期では、Bottom 部の同期成立により lbsy が更新（bst の揃った値がコピー）され、各 CMG のバリアバンクへ情報が配送される。配送されるのは同期木の sync_id (図では T)であり、これを受け取った CMG は、sync_id が一致していることを確認すると、Top 部の unfinished_count をデクリメントする。unfinished_count が 0 になると同期木全体での同期が成立し、Share 部の lbsy が更新（反転）される。

Note Bottom 部の lbsy と Share 部の lbsy で、更新される値が異なることに注意。初期状態を正しく設定しないと、同期したことが木全体に伝わらない。

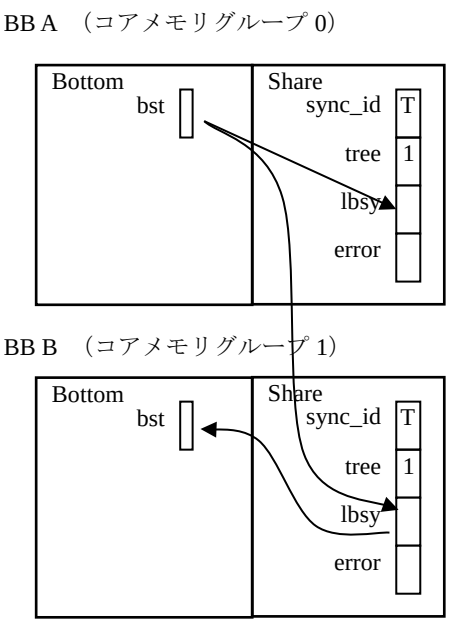


図 14-8　2BB の同期木（post-wait BB）

図 14-8 は、post-wait 用の同期木の動作を示している。tree = 1 のとき、post-wait 用 BB は bst<0>の値を share.lbsy に書き込み、他の CMG へ配送する。これを受け取った BB は、sync_id が一致すれば share.lbsy を更新（反転）し、更新後の値を bst<0>にも反映する。

Note　bst<0>の値が share.lbsy と異なる場合のみ他の CMG へ配送する。一致する場合、他の CMG への配送は行われない。

Note　post-wait 用 BB では、送信側の share.lbsy は bst<0>の値が書き込まれ、受信側の share.lbsy は反転する。ソフトウェアは、初期状態で送信側と受信側の bst<0>, share.lbsy が矛盾しないよう設定する必要がある。

14.3.3.　ソフトウェアで気をつけること

同期木が同一性を保ちながら正しく動作するためには、同期木を構成する BB が同じ情報を持ち、同じ動作をしなければならない。ハードウェアは、同期木を構成するすべての BB の設定に矛盾がないことをチェックすることはできないので、同期木を正しく構成することはソフトウェアの責任となる。

ハードウェアは、同期木が正しく構成されていれば起こらないはずの操作が起きたとき、share.error に 1 を設定する。しかし、ハードウェアでは、同期木の矛盾をすべて検出できるわけではないことに注意。

同期木を構成する際の注意事項には、例えば以下のものがある。

- Top 部の unfinished_count に expected_count と同じ値を設定する。ハードウェアは expected_count と異なる値を設定しても、error フィールドに 1 を設定することはない。
- unfinished_count に 0 を設定するとブロードキャストを受信しても unfinished_count は変化せず、同期できない。
- unfinished_count に expected_count より大きな値を設定すると、ブロードキャストが必要数受信できないので同期できない。
- unfinished_count に 0 より大きく expected_count より小さな値を設定すると、全体の同期を待たずにその BB だけ同期が成立したことになってしまう。同期途中の BB の状態を復元する場合は、このような値を設定することがありうる。

- bottom.lbsy と lbsy を異なる値で初期化すると、ASI_LBSY で読み出す値を反転させて ASI_BST に書くというハードウェアバリアの基本が成立しなくなるかもしれない。

> **Comment**　同期途中の BB の状態を復元する場合は、bottom.lbsy と lbsy に異なる値を設定することがありうるので、ハードウェアはこのような設定に対し error フィールドに 1 を設定することはない。

14.4. レジスタ

バリア資源は、特権アクセスレジスタ、窓レジスタを介してアクセスされる。特権アクセスレジスタ及び窓レジスタは仮想的なレジスタであり、バリア資源のすべてもしくは一部が割り当てられる。特権ソフトウェア及び非特権ソフトウェアは、この仮想的なレジスタを通しバリア資源に間接的なアクセスが可能である。

具体的にバリア資源の特権アクセスレジスタは、ASI_PRIV_BARRIER_CTRL, ASI_PRIV_BARRIER_INIT, ASI_PRIV_BST_BIT, ASI_PRIV_BARRIER_ASSIGN である。これらのうち、ASI_PRIV_BARRIER_INIT でアクセスされるバリア資源は CMG で共有されていることに注意が必要である。

また、窓レジスタは、ASI_LBSY, ASI_BST である。

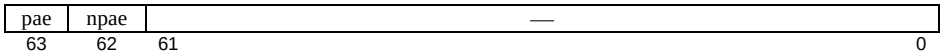
VCPU にバリア資源の使用権が与えられているかは、バリアアクセス制御レジスタに表示される。

14.4.1. バリアアクセス制御レジスタ

レジスタ名	ASI_PRIV_BARRIER_CTRL
ASI 番号	EF_{16}
VA	$3F8_{16}$
共有範囲	VCPU

アクセス

	read	write
user	*privileged_action*	*privileged_action*
priv	OK	*privileged_action* or OK



ビット	フィールド	アクセス	説明
63	pae	RO	privileged access enable バリアの特権アクセスレジスタ経由でのアクセスを許可する pae=0 のとき特権モードからのライトは *privileged_action* 例外を検出する。 pae=1 のとき特権モードからのライトは無視される。
62	npae	RW	nonprivileged access enable ASI_BST, ASI_LBSY について、非特権モードでのアクセスを許可する。

pae

pae = 1 のとき、特権モードにおいて、特権アクセスレジスタを介してバリア資源へのアクセスが可能となる。pae = 0 のとき、特権モードにより特権アクセスレジスタへアクセスした場合 *privileged_action* 例外を検出する。

ASI_PRIV_BARRIER_CTRL については、pae = 0 のときも特権モードから読み出しは可能であるが、書き込みには *privileged_action* 例外を検出する。pae = 1 のとき、特権モードから読み出しと書き込みが可能である。ただし、特権モードからの pae への書き込みは無視される。

ASI_PRIV_BARRIER_INIT, ASI_PRIV_BST_BIT, ASI_PRIV_BARRIER_ASSIGN は特権アクセスレジスタである。資源を CMG で共有している ASI_PRIV_BARRIER_INIT を更新した場合、CMG 全体に影響を与える。

npae

npae = 1 のとき、ASI_BST, ASI_LBSY に非特権モードでアクセス可能となる。npae = 0 のとき、ASI_BST, ASI_LBSY に非特権モードでアクセスすると、*privileged_action* 例外を検出する。

Programming Note　バリア窓への非特権モードからのアクセスを制御することでコンテキストスイッチ時に高速なバリア開閉処理が可能である。

pae, npae の値により、特権アクセスレジスタ及び窓レジスタは表 14-1 に示す特権モード及び非特権モードでアクセスが可能となる。

表 14-1　バリアアクセス制御レジスタ設定とバリアレジスタのアクセス権

ASI レジスタ名	pae = 0 npae = 0	pae = 0 npae = 1	pae = 1 npae = 0	pae = 1 npae = 1
ASI_PRIV_BARRIER_CTRL	priv(RO)	priv(RO)	priv	priv
ASI_PRIV_BARRIER_INIT			priv	priv
ASI_PRIV_BST_BIT			priv	priv
ASI_PRIV_BARRIER_ASSIGN			priv	priv
ASI_LBSY, ASI_BST	priv	priv, user	priv	priv, user

14.4.2.　BB の初期化

14.4.2.1.　バリア用BB

レジスタ名　ASI_PRIV_BARRIER_INIT（バリア用 BB）

ASI 番号　EF_{16}

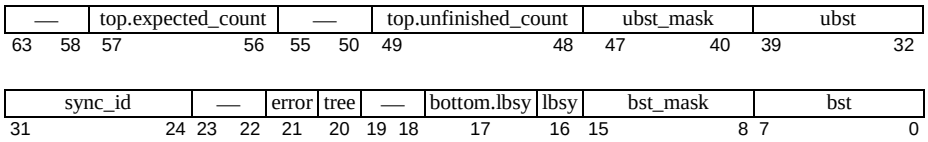
VA　200_{16}, 208_{16}, 210_{16}, 218_{16}, 280_{16}, 288_{16}, 290_{16}, 298_{16}

共有範囲　VCPU (実体は CMG)

アクセス

	read	write
user	*privileged_action*	*privileged_action*
priv	*privileged_action* or OK	*privileged_action* or OK

バリア用 BB



ビット	フィールド	アクセス	説明
57:56	top.expected_count	RW	Top 部が受信するブロードキャストの期待値。tree = 1 のときのみ有効。 tree の設定に関わらず、書き込んだ値が読み出せる。 また、SPARC64™ XIfx では tree = 1 のとき、2 以外の値を設定した場合動作は不定となる。
49:48	top.unfinished_count	RW	Top 部のブロードキャスト受信を制御するカウンタ。この値が 0 になると Share 部の lbsy が反転し、同時に top.expected_count の値が再設定される。tree = 1 のときのみ有効。 tree の設定に関わらず、書き込んだ値が読み出せる。
47:40	ubst_mask	RW	ubst_mask と bst_mask で BST のマスクを指示・読み出す。各ビットと VCPU の対応は、BST ビット位置取得の bstbit で取得される値 i に対し、bst_mask<i>がその VCPU のビット位置となる。
39:32	ubst	RW	ubst と bst で BST の値を指示・読み出す。各ビットと VCPU の対応は、BST ビット位置取得の bstbit で取得される値 i に対し、bst<i>がその VCPU のビット位置となる。
31:24	sync_id	RW	同期木の識別子。tree = 1 のときのみ有効。 tree の設定に関わらず、書き込んだ値が読み出せる。
21	error	RW	ハードウェアが異常を検出した場合に 1 がセットされる。 error = 1 のときは、 ソフトウェアによる書き込みでは、0 のみが書き込み可能。1 の書き込みは無視される。
20	tree	RW	同期木を構成するかどうか。 このフィールドが 0 のときは、Bottom 部だけでバリアバンク内の同期に使われる。1 のときは、同期木を構成する。
17	bottom.lbsy	RW	Bottom 部の lbsy。tree = 0 のときは Share 部の lbsy と常に一致する。tree = 1 のときは Bottom 部の同期が成立すると更新される。
16	lbsy	RW	Share 部の lbsy。窓 ASI (ASI_LBSY) で読み出すことができる。
15:8	bst_mask	RW	ubst_mask を参照。
7:0	bst	RW	ubst を参照。

Note ubst_mask と bst_mask は連結してひとつのビットマップフィールドを構成している。以下の説明では、曖昧さがない場合、ubst_mask と bst_mask を区別せず bst_mask と表記することがある。bst_mask<8>は ubst_mask<0>を意味する。ubst, bst も同様。

VA で指定される BB の値の取得および初期化を行う。読み出しで現在の設定が読み出され、書き込みで新しい設定を書くことができる。

BB は VA<7:3>で表わされる BB 番号で識別される。14.4.3 窓の割りつけ(P.392)により窓 ASI へ BB を割り付ける際には、この BB 番号で指示する。

読み出しに対しては、VA で指定される BB の状態が読み出される。bst の各ビットに読み出される値は、対応する bst_mask のビットが 1 のときは実際の値が読み出されるが、0 のときは不定値が返される。

書き込み時は、VA で指定される BB の設定が更新される。bst の各ビットに書き込む値は、対応する bst_mask のビットが 1 のときは正しく書き込まれるが、0 のときは書き込まれるかどうかは未定義である。

BB の設定を間違うとバリアが正しい動作をしなくなる。ハードウェアは、正しく設定していれば起こりえない値を設定した場合は error フィールドに 1 を設定するが、ハードウェアは異常設定のすべてを検出することはできない。同期木を構成する場合もしない場合も、BB を正しく設定するのはソフトウェアの責任である。ハードウェアの異常検出条件と書き込みに関する注意事項を以下に羅列する。また、14.3.3 も参照。

- error は以下の条件で 1 となる。
  - tree = 1 のとき、全体のバリア同期が成立しないうちに Bottom 部のバリア同期が複数回成立
- bottom.lbsy は Bottom 部のバリア同期が成立したときに更新されるが、バリア同期が成立しているかどうかの検査は書き込みの直後にも行われる。ただし、書き込みにより bottom.lbsy が更新された場合は、他のバリアバンクへの配送は発生しない。これは同期木が同期途中の状態でも BB の退避･復元ができるようにするためで、窓 ASI を通して bst を更新したときのみ他のバリアバンクへの配送が発生する。
- bst_mask がすべて 0 の場合、バリア同期が成立しているかどうかの検査は行われず、bottom.lbsy には書き込んだ値がそのまま保持される。

pae = 0 のとき特権モードにより `ASI_PRIV_BARRIER_INIT` にアクセスした場合、*privileged_action* 例外を検出する。

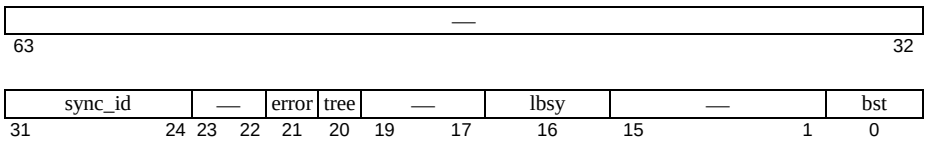
14.4.2.2. post/wait用BB

レジスタ名	`ASI_PRIV_BARRIER_INIT` (post/wait 用)
ASI 番号	EF_{16}
VA	220_{16}, 228_{16}, 230_{16}, 238_{16}, 240_{16}, 248_{16}, 250_{16}, 258_{16}, $2A0_{16}$, $2A8_{16}$, $2B0_{16}$, $2B8_{16}$, $2C0_{16}$, $2C8_{16}$, $2D0_{16}$, $2D8_{16}$, $2E0_{16}$
共有範囲	VCPU (実体は CMG)

アクセス

	read	**write**
user	*privileged_action*	*privileged_action*
priv	*privileged_action* or OK	*privileged_action* or OK

post/wait 用



ビット	フィールド	アクセス	説明
31:24	sync_id	RW	同期木の識別子。tree = 1 のときのみ有効。 tree の設定に関わらず、書き込んだ値が読み出せる。

21	error	RW	ハードウェアが異常を検出した場合に 1 がセットされる。 error = 1 のときは、 ソフトウェアによる書き込みでは、0 のみが書き込み可能。 1 の書き込みは無視される。
20	tree	RW	同期木を構成するかどうか。 このフィールドが 0 のときは、Bottom 部だけでバリアバンク内の同期に使われる。1 のときは、同期木を構成する。
16	lbsy	RW	Share 部の lbsy。窓 ASI (ASI_LBSY) で読み出すことができる。
0	bst	RW	BST の値を指示・読み出す。 バリア用 BB と異なり、BST ビット位置取得の bstbit で取得される値によらずすべての VCPU が bst<0>に対応する。

VA で指定される BB の値の取得および初期化を行う。読み出しで現在の設定が読み出され、書き込みで新しい設定を書くことができる。

BB は VA<7:3>で表わされる BB 番号で識別される。14.4.3 窓の割りつけ(P.392)により窓 ASI へ BB を割り付ける際には、この BB 番号で指示する。

読み出しに対しては、VA で指定される BB の状態が読み出される。bst<0>には常に実際の値が読み出される。

書き込み時は、VA で指定される BB の設定が更新される。bst<0>には常に指定した値が書き込まれる。

BB の設定を間違うとバリアが正しい動作をしなくなる。ハードウェアは、正しく設定していれば起こりえない値を設定した場合は error フィールドに 1 を設定するが、ハードウェアは異常設定のすべてを検出することはできない。同期木を構成する場合もしない場合も、BB を正しく設定するのはソフトウェアの責任である。ハードウェアの異常検出条件と書き込みに関する注意事項を以下に羅列する。また、14.3.3 も参照。

- post-wait 用 BB の error は以下のいずれかの条件で 1 となる。
  - tree = 1 のとき、ASI_BST に書き込み後の Bottom 部からのブロードキャスト発信待ちの状態で、さらに ASI_BST に lbsy と異なる値を書き込み
  - tree = 1 のとき、ASI_BST に書き込み後の Bottom 部からのブロードキャスト発信待ちの状態で、他バリアバンクからのブロードキャストを受信

pae = 0 のとき特権モードにより ASI_PRIV_BARRIER_INIT にアクセスした場合、*privileged_action* 例外を検出する。

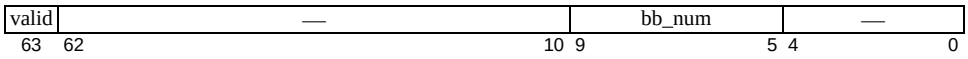
14.4.3. 窓の割りつけ

レジスタ名	ASI_PRIV_BARRIER_ASSIGN
ASI 番号	EF_{16}
VA	バリア用窓: 300_{16}, 308_{16}, 310_{16}, 318_{16} post/wait 用窓: 320_{16}, 328_{16}, 330_{16}, 338_{16}
共有範囲	VCPU

アクセス

演算コア		
	read	**write**
user	*privileged_action*	*privileged_action*
priv	*privileged_action* or OK	*privileged_action* or OK

アシスタントコア		
	read	**write**
user	*DAE_invalid_ASI*	*DAE_invalid_ASI*
priv	*DAE_invalid_ASI*	*DAE_invalid_ASI*



ビット	フィールド	アクセス	説明
63	valid	RW	窓を開ける場合は 1 を、閉める場合は 0 を指定する。
9:5	bb_num	RW	どの BB に対して窓を開けるかを指定する。

Compatibility Note SPARC64™ VIIIfx, SPARC64™ IXfx に比較し post-wait BB 用の窓を減らした。

窓 ASI (ASI_BST, ASI_LBSY) の割付け状態の取得および変更を行う ASI である。VA<7:0>が ASI_BST, ASI_LBSY の VA<7:0>と対応しており、VA で指定された窓に、bb_num で指定された BB を割りつける、あるいは VA で指定された窓の割りつけを解除することができる。

読み出しに対しては、どの BB が割付けられているかが返される。VA で指定された窓が BB に割り付けられているなら valid = 1 となり、bb_num には BB 番号が表示される。VA で指定された窓が BB に割りつけられていないときは、valid = 0 となり bb_num の値は不定である。

書き込みに対しては、

- valid = 1 の場合は、指定された bb_num の LBSY, BST を窓に割りつける。この書き込みの完了以降、ASI_BST への書き込みが BB の BST に反映されるようになり、ASI_LBSY の読み出しにより bb_num にある LBSY が読み出せるようになる。
- valid = 0 の場合は、指定された窓の割りつけを解除する。この書き込みの完了以降、ASI_BST への書き込みは無視され、ASI_LBSY の読み出しには不定値が返る。

窓 ASI はバリア用と post-wait 用に用途が分かれている。バリア用窓 ASI にはバリア用 BB (#0 – #3 および#16 – #19) が割り付け可能であり、post-wait 用窓 ASI には post-wait 用 BB (#4 – #11 および#20 – #28) が割り付け可能である。書き込む値の valid = 1 のとき、すなわちバリア用窓を開く書き込みでは、バリア用窓 ASI に post-wait 用 BB 番号を指定する、逆に post-wait 用窓 ASI にバリア用 BB 番号を指定すると、その書き込みは無視される。これに対し、書き込む値の valid = 0 のとき、すなわちバリア用窓を閉じる書き込みでは、bb_num によらず常に書き込みは有効である。

Compatibility Note SPARC64™ IXfx では、valid の値によらず間違った bb_num を指定した場合、書き込みは無視される仕様だった。

また、どちらの窓 ASI についても、存在しない BB の番号を指定した書き込みは無視される。

Programming Note 窓 ASI と BB の対応を間違えて指定しても例外は発生しない。正しく設定されたことを確認するためには、書き込み後に読み出しを行い、valid = 1 かつ bb_num が指定通りであることを確かめるとよい。

BB の初期化と窓の割りつけで矛盾するような設定をした場合の動作は不定である。ハードウェアでは矛盾を検出しないので、ソフトウェアは、バリア資源の初期化・割り当てに際し矛盾を起こさないようにすること。使用中の BB に BB の初期化処理を行う、bst_mask<i> = 0 である bst<i>を窓レジスタに割りつける、などの場合の同期処理は保証されない。

Programming Note システムソフトウェアは初期化済の BB を割り当てること。未初期化の BB を割り当て、その BB ユーザプログラムに操作されると、意図しない結果になるかもしれない。

窓の割りつけは、演算コアのみ有効である。アシスタントコアで読み出しもしくは書き込みを行った場合 *DAE_invalid_ASI* を検出する。

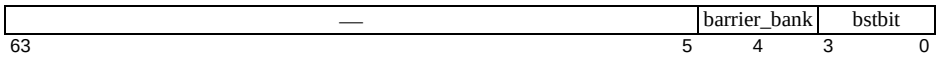
pae = 0 のとき特権モードにより ASI_PRIV_BARRIER_ASSIGN にアクセスした場合、*privileged_action* 例外を検出する。また、アシスタントコアよりアクセスした場合、*DAE_invalid_ASI* 例外を検出する。

14.4.4. BST ビット位置取得

レジスタ名	ASI_PRIV_BST_BIT
ASI 番号	EF_{16}
VA	$F8_{16}$
共有範囲	VCPU

アクセス

演算コア		
	read	**write**
user	*privileged_action*	*privileged_action*
priv	*privileged_action* or OK	*privileged_action* or *DAE_invalid_asi*
アシスタントコア		
	read	**write**
user	*DAE_invalid_asi*	*DAE_invalid_asi*
priv	*DAE_invalid_asi*	*DAE_invalid_asi*



ビット	フィールド	アクセス	説明
4	barrier_bank	RO	バリアバンク番号
3:0	bstbit	RO	BST のビット位置

bst_mask, bst のビット位置、およびバリアバンク番号を取得する ASI である。

barrier_bank には、VCPU の属するバリアバンク番号が読み出される。

bstbit には、VCPU ごとに 0 – 15 の値が読み出される。この値はバリアバンク内の VCPU でユニークである。

BST ビット位置取得は、演算コアのみ有効な ASI である。アシスタントコアで読み出しもしくは書き込みを行った場合 *DAE_invalid_ASI* を検出する。

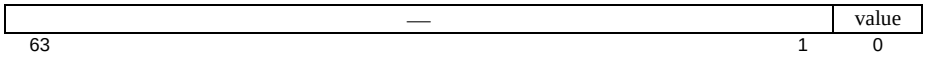
pae = 0 のとき特権モードにより ASI_PRIV_BST_BIT にアクセスした場合、*privileged_action* 例外を検出する。また、アシスタントコアよりアクセスした場合、*DAE_invalid_ASI* 例外を検出する。

14.4.5. バリア操作用 ASI

レジスタ名	ASI_LBSY (read), ASI_BST (write)
ASI 番号	EF_{16}
VA	バリア用: 00_{16}, 08_{16}, 10_{16}, 18_{16} post-wait 用: 20_{16}, 28_{16}, 30_{16}, 38_{16}
共有範囲	VCPU

アクセス

演算コア		
	read	**write**
user	*privileged_action* or OK	*privileged_action* or OK
priv	OK	OK
アシスタントコア		
	read	**write**
user	*DAE_invalid_asi*	*DAE_invalid_asi*
priv	*DAE_invalid_asi*	*DAE_invalid_asi*



ビット	フィールド	アクセス	説明
0	value	RW	読み出しには lbsy の値が返り、書き込むと bst が更新される。

ASI_LBSY, ASI_BST は BB にアクセスするための窓 ASI で、各 VCPU に 8 個用意されている。窓 ASI はバリア BB 用 4 個と post-wait BB 用 4 個に分かれており、窓の割りつけ(P.392)で窓と BB の対応を設定したうえで、非特権モードおよび特権モードのソフトウェアが使用する。割り付けられていない窓 ASI に対する読み出しは不定値が返り、書き込みは無視される（例外は発生しない)。

ASI_LBSY, ASI_BST は、演算コアのみ有効な ASI である。アシスタントコアで読み出しもしくは書き込みを行った場合 *DAE_invalid_ASI* を検出する。

npae = 0 のとき、非特権モードによるアクセスが制限される。ASI_LBSY, ASI_BST にアクセスしたとき、*privileged_action* 例外を検出する。npae = 1 のとき、非特権モードによりアクセスできる。

サンプルコード

```
/*
 * %r1: VA of a window ASI
 * %r2, %r3: work registers
 */

    ldxa [%r1]ASI_LBSY, %r2    ! 現在の LBSY を読み出す
    not %r2                    ! LBSY を反転させる
    and %r2, 1, %r2            ! reserved フィールドを捨てる
    stxa %r2, [%r1]ASI_BST     ! BST を更新する
    membar #storeload          ! stxa が完了するのを待つ

loop:
    ldxa [%r1]ASI_LBSY, %r3    ! LBSY を読み出す
    and %r3, 1, %r3            ! reserved フィールドを捨てる
    subcc %r3, %r2, %g0        ! 値が変化したか?
    bne,a loop
    sleep                      ! 変化していなければ sleep する
```

15. Sector Cache

SPARC64™ XIfx はキャッシュをセクタと呼ばれる部分に分けて更新管理をする仕組みを提供する。この機構をセクタキャッシュと呼ぶ。セクタの利用法としては、使用頻度が高いデータをキャッシュに残りやすくする、または、使用頻度の低いデータを大量に扱う際にキャッシュを汚さないようにする、などが考えられる。また、アシスタントコアが使用する領域を制限・隔離することで、演算コアとアシスタントコア双方の不必要なキャッシュのスラッシングを減少させることが考えられる。

15.1. 概要

セクタキャッシュとは、キャッシュ上にセクタと呼ばれる部分を作り、データを囲い込む機能である。各セクタにはそれぞれ最大許容量が指定されており、セクタの使用量が最大許容量より小さいときは、セクタ内のデータがキャッシュから落ちないよう、キャッシュの追い出し機構を制御する。キャッシュ上には複数のセクタを作成することができ、また各セクタの最大許容量は他のセクタの最大許容量とは独立に設定できるため、応用の自由度が高い。

SPARC64™ XIfx では L1 キャッシュ、L2 キャッシュともセクタキャッシュ機構を実装しており、セクタキャッシュ機能を有効にするかどうかは L1 キャッシュ、L2 キャッシュ個々に設定できる。SPARC64™ XIfx では、チップ内に 2 つの CMG を搭載しており、各々のグループ単位で個別にセクタ設定が可能である。

あるソフトウェアが使用できるのは L1 キャッシュでは一組 4 つ、L2 キャッシュでは一組 2 つのセクタである。L2 キャッシュは CMG ごとに多数の演算コアやアシスタントコアで共有されており、2 組のセクタ設定を VCPU ごとに切り替えることで最大 4 セクタを使用することが可能である。

CMG 間では設定はそれぞれ独立であり、他の CMG の設定にアクセスすることはできない。

Programming Note CMG をまたいで、全演算コア 32 コアを 1 プロセスで使用する場合、ソフトウェア側であらかじめ両 CMG の設定を共通にしておくことを推奨する。

Programming Note CMG 内で、16 コアを分割して複数プロセスで使用する場合、同一グループ内で設定される値はプロセス間で同一のものであることを推奨する。

Compatibility Note SPARC64™ VIIIfx, SPARC64™ IXfx では L1 キャッシュ、L2 キャッシュともセクタは一組 2 つであった。

セクタは番号で識別される。命令フェッチやロードストア命令など、すべてのメモリアクセスにはセクタ番号が付加される。セクタ番号はソフトウェアが明示的に指定することもできるし、指定しない場合は暗黙のセクタ番号が付加される。

使用したいメモリのデータがキャッシュ上になければ、メモリから読み込まれ、セクタ番号を付加してキャッシュに格納する。このとき、キャッシュ上のデータの追い出しが発生するが、セクタキャッシュが有効なときは、セクタの容量を考慮して追い出すデータが選択される。

一方、キャッシュ上にデータがあるときは、そのデータが読み出し、あるいは更新される。このときは、キャッシュ上のデータがどのセクタに属しているかや、キャッシュアクセスに付加されたセクタ番号には関係なく、すべてのセクタ上のデータが使用可能である。

15.2. セクタキャッシュの容量

セクタの最大許容量はウェイ数で指定する。最大許容量の指定方法は、キャッシュのあるウェイがどのセクタに属するかを設定するのではなく、どのセクタに何ウェイ分の容量を割り当てるかを指定する。セクタ容量の指定は全インデクスで共通であり、インデクス毎に個別の指定はできない。

セクタの最大許容量として指定できる一番小さな値は、1 ウェイである。セクタの最大許容量を 0 ウェイと指定したとき、キャッシュにデータを載せないという設定ではなく、そのセクタのセクタキャッシュ機能が無効となる。セクタキャッシュ機能が無効であるセクタが指定された場合、他のセクタの設定とは関係なく、通常のキャッシュリプレースが行われる。

セクタの最大許容量の最大値は、キャッシュの最大ウェイ数 MAXWAY である。MAXWAY を超える値はすべて、MAXWAY とみなされる。

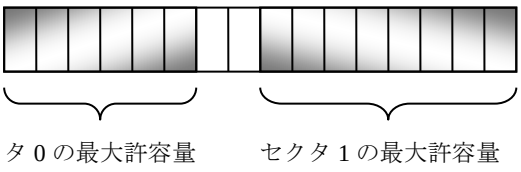
上記で説明したようにセクタキャッシュを有効にするためには、使用したいセクタの最大許容量を 1 ウェイ以上に指定すればよい。すべてのセクタの最大許容量が 0 の場合は、セクタキャッシュの機能は無効になる。

キャッシュの追い出し機構は、自セクタの容量が最大許容量より小さければ、他のセクタのデータを追い出して自セクタの容量を確保しようとする。最大許容量を超えているセクタを優先的に追い出すが、各セクタの最大許容量の合計が最大ウェイ数を超えていると、最大許容量より容量の小さいセクタが追い出されることもある。

セクタの最大許容量は、キャッシュからデータを追い出すときに参照される、セクタの容量の目標値である。セクタキャッシュを有効にした時点で、あるセクタの使用量が最大許容量以上だったとしても、あふれた分を強制的にキャッシュから追い出すことはない。

SPARC64™ XIfx では L1 キャッシュ、L2 キャッシュ共に 4 つのセクタを使用することができる。簡単化のために、以下では 2 つのセクタを使用する場合を例に挙動をまとめる。

- セクタ 0 とセクタ 1 の最大許容量の合計が MAXWAY より小さい場合

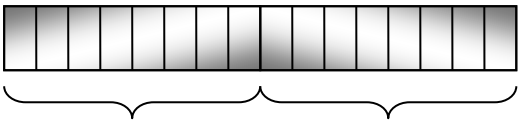


この場合、セクタ 0, セクタ 1 のどちらも最大許容量が保証される。

上図で空白のウェイは、どちらのセクタにも属さないウェイではないことに注意。キャッシュ上のデータは、セクタキャッシュ機能が有効か無効かに関わらずセクタ番号情報を持っているので、セクタキャッシュを有効にした時点で、保持しているセクタ情報にもとづいてどちらのセクタに属するかが決まる。このような場合には、セクタの使用量が最大許容量を超えている状態になる。

セクタキャッシュ機能を有効にした時点で空のウェイがある場合は、そのウェイは次のキャッシュミスで使われるため、この場合もセクタの使用量が最大許容量を超えている状態になる。

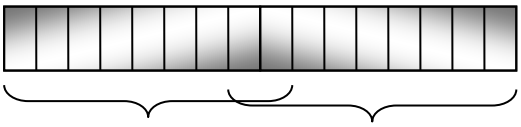
- セクタ 0 とセクタ 1 の最大許容量の合計が MAXWAY に等しい場合



セクタ 0 の最大許容量　　セクタ 1 の最大許容量

この場合は、セクタキャッシュを有効にした時点でどちらかのセクタの使用量が最大許容量を超えていたとすると、もう一方のセクタの容量は最大許容量未満なので、キャッシュミス時に反対側のセクタのデータを追い出すため、最終的に両方のセクタが最大許容量を使う状態に落ち着く。

- セクタ 0 とセクタ 1 の最大許容量の合計が MAXWAY より大きい場合



セクタ 0 の最大許容量　　セクタ 1 の最大許容量

この場合は、両方のセクタが同時に最大許容量まで使用することはできない。使用量が最大許容量以下のセクタでキャッシュミスが起こると、最大許容量を満たすまでは自セクタのデータを落とすことはせず、空のウェイがあればそれを、なければ相手側セクタのウェイを奪ってデータを格納する。

セクタの動作ではセクタ番号での優劣はなく、すべてのセクタが平等に扱われる。

あるセクタの使用量が最大許容量に満たない場合かつ、他のセクタのうち最大許容量を超えているものがあれば、最大許容量を超えているセクタを対象とした LRU によりデータが追い出される。また、無効であるセクタは最大許容量が 0 として扱われる。つまり、無効であるセクタのデータがある場合、そのセクタは常に最大許容量を超えている状態として扱われる。最大許容量を超えているセクタがなければ、他のすべてのセクタから LRU によりデータが追い出される。詳細は 15.5 節を参照。

15.3. セクタの指定方法

ソフトウェアの命令フェッチ、データアクセスにおけるセクタ番号は、0, 1, 2 または 3 の番号で指定する。ソフトウェアが明示的に指定するには、XAR レジスタまたは SXAR 命令で XAR.sector_h (XAR.urs3<2>)及び XAR.sector (XAR.urs3<0>) の 2 ビットを使用する。

このセクタ番号は 0,1,2,3 が指定可能であるが、L1 キャッシュと L2 キャッシュで扱いが異なることに注意が必要である。詳細を以下に述べる。

L1 キャッシュでは、XAR.v = 1 のメモリアクセス命令の場合、XAR.sector_h, XAR.sector により表 15-1 で示されるセクタ番号を指定する。XAR.v = 0 のメモリアクセスや命令フェッチでは暗黙のセクタ指示が付加される。暗黙のセクタは、セクタキャッシュ割り当て設定レジスタの default_sector_h, default_sector で指定する。

表 15-1　L1 キャッシュにおけるソフトウェアによるセクタ指示

XAR.v	XAR.sector_h	XAR.sector	指示されるセクタ番号
0	—	—	暗黙のセクタ (default_sector_h, default_sector)
1	0	0	セクタ 0
1	0	1	セクタ 1
1	1	0	セクタ 2
1	1	1	セクタ 3

一方、L2 キャッシュでは XAR.v = 1 のメモリアクセス命令の場合、XAR.sector のみにより表 15-1 で示されるセクタ番号を指定する。L1 キャッシュと異なり、XAR.sector_h で指定された値は無視される。つまり、セクタ 2,3 を指示した場合 L2 キャッシュではセクタ 0,1 に読み替えられる。XAR.v = 0 のメモリアクセスや命令フェッチでは暗黙のセクタ指示が付加される。暗黙のセクタは、セクタキャッシュ割り当て設定レジスタの default_sector で指定する。

表 15-2　L2 キャッシュにおけるソフトウェアによるセクタ指示

XAR.v	XAR.sector_h	XAR.sector	指示されるセクタ番号
0	—	—	暗黙のセクタ (default_sector)
1	—	0	セクタ 0
1	—	1	セクタ 1

XAR.sector_h, XAR.sector または暗黙のセクタ指示によって、すべてのメモリアクセス命令にセクタ指示が付加されるが、意味があるのはキャッシャブル空間に対するアクセスのみである。ノンキャッシャブル空間や non-translating ASI に対するセクタ指示は無視され、例外は発生しない。

セクタキャッシュの機能が無効な場合、セクタ指示はキャッシュの追い出し機構にとって意味がないが、指示された値はキャッシュ上に記録されるため、セクタキャッシュ機能が有効になった際や、データが L1 キャッシュから追い出され L2 キャッシュに書き込まれる場合に、指示したセクタ値に応じて追い出し機構が適切に働く。

Compatibility Note　SPARC64™ VIIIfx, SPARC64™ IXfx では、セクタ番号は 0,1 のみで指定できた。

15.4. セクタキャッシュ制御レジスタ

セクタキャッシュを制御するレジスタの使用権が与えられた VCPU では、特権アクセスレジスタ及び仮想化されたセクタキャッシュ制御レジスタを介し、セクタキャッシュの設定が可能となる。特権アクセスレジスタ及び仮想化されたセクタキャッシュ制御レジスタは ASI に定義されており、特権ソフトウェア及び非特権ソフトウェアはこれらの ASI を通してセクタキャッシュの設定を行う。

セクタキャッシュを制御するレジスタには、2 組のセクタ設定用のセクタキャッシュ制御レジスタ(SCCR0, SCCR1)、仮想化されたセクタ設定用の仮想セクタキャッシュ制御レジスタ(VSCCR)、仮想化レジスタと実体レジスタを結びつけるセクタキャッシュ割り当て設定レジスタ(SCCR_ASSIGN)がある。また、特権アクセスレジスタへのアクセスは、特権アクセスレジスタ制御レジスタ（SCCR_CTRL）によって制御される。セクタ制御レジスタ(SCCR0, SCCR1)は CMG 内で共有されており、設定は CMG 内のすべての VCPU に影響を与えることに注意が必要である。図 15-1 にレジスタの関係を示す。

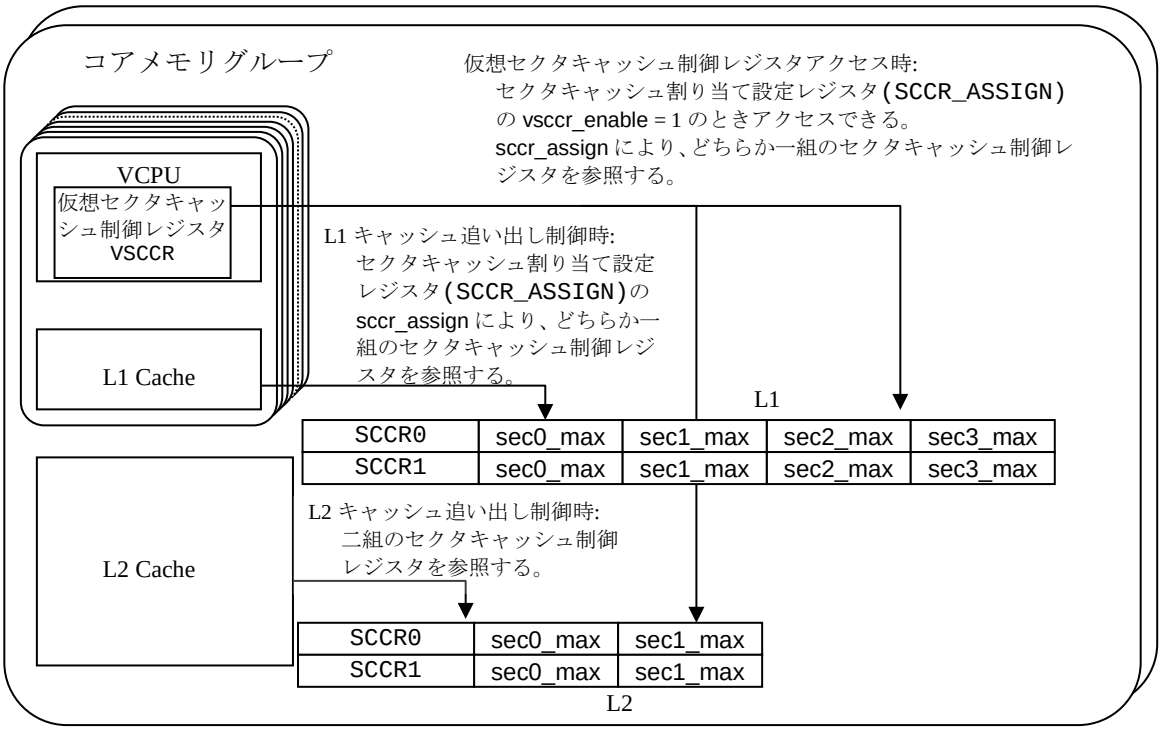


図 15-1　セクタキャッシュ制御レジスタの関係

Programming Note　仮想レジスタをマッピングせず、最大許容量を適切に設定することで、L2 キャッシュを 4 つに分割して、それぞれを VCPU に割り当てることができる。

SCCR0, SCCR1 の設定変更にあたっては、以下の 2 つの方法がある。

1) SCCR0, SCCR1 へ直接アクセス（特権ソフトウェアのみ）

- SCCR_CTRL の pae が 1 の場合、特権ソフトウェアによって SCCR0, SCCR1 の設定を変更することができる。

2) VSCCR を通して SCCR0, SCCR1 へアクセス（特権ソフトウェア、非特権ソフトウェア）

- 特権ソフトウェアでは、SCCR_CTRL の pae が 1 または、SCCR_ASSIGN の vsccr_enable が 1 の場合、VSCCR を通して SCCR0 もしくは SCCR1 の設定を変更することができる。SCCR_CTRL の pae が 0 かつ SCCR_ASSIGN の vsccr_enable が 0 の場合、VSCCR にアクセスすると *privileged_action* 例外が発生する。
- 非特権ソフトウェアでは、SCCR_ASSIGN の vsccr_enable が 1 の場合、VSCCR を通して SCCR0 もしくは SCCR1 の設定を変更することができる。SCCR_ASSIGN の vsccr_enable が 0 の場合、VSCCR にアクセスすると *privileged_action* 例外が発生する。
- VSCCR を通してアクセスされるのは SCCR0, SCCR1 のどちらかのレジスタで、割り当ては SCCR_ASSIGN の sccr_assign により決まる。VSCCR を通して設定が変更された場合も CMG 内すべての VCPU に影響を与える点に注意が必要である。

15.5. キャッシュ追い出し機構のアルゴリズム

セクタキャッシュ機能が有効かどうかによらず、すべての命令フェッチ、メモリアクセスにはセクタ番号が付加される。セクタ番号の情報は、データがキャッシュに格納されるときに一緒に保存される。一方、キャッシュの追い出し機構は、セクタキャッシュ機能が有効なときは、キャッシュ上に保存されたセクタ情報と最大許容量から追い出すデータを決定する。つまりセクタ情報は、そのメモリアクセスによるデータがキャッシュに載る時点でセクタキャッシュ機能が有効だったかどうかとは関係なく、キャッシュの追い出し機構が追い出すデータを選出する時点で作用する。セクタキャッシュ機能が無効なときは、キャッシュの追い出し機構はセクタ情報を使わずに追い出すデータを選出する。

セクタ情報は、データをキャッシュに格納する際だけでなく、キャッシュ上のデータにアクセスする際にも更新される。最初のアクセスをセクタ番号 0 で最初のアクセスが起きてキャッシュに載ったデータを、その後セクタ番号 1 でアクセスすると、セクタ番号は 1 に更新される。

Programming Note　データ読み出しやプリフェッチでも、セクタ情報は変更される。セクタ情報はキャッシュライン単位で管理されるので、ライン内のデータ毎に異なるセクタを指定すると、最後にアクセスされたときのセクタが指定されたことになる。

Programming Note　SPARC64™ XIfx はメモリアクセスをアウトオブオーダで処理するので、ユーザプログラムの意図通りにセクタ情報が更新されない可能性がある。

15.5.1. 表記規則

この節で使われる表記規則を説明する。

- L1 キャッシュと L2 キャッシュでセクタキャッシュの構成が異なっているため、L1 キャッシュ、L2 キャッシュの実際のセクタ番号を区別して表記する。
  - L1 キャッシュのセクタは一組 4 つである。セクタ 0 から順に、S_{F0} ,S_{F1} ,S_{F2} ,S_{F3} と表記する。
  - L2 キャッシュではソフトウェアからの指示はセクタ番号 0, 1 に読み替えられ、二組 4 つのセクタを指定する。セクタ 0 から順に、S_{S0}, S_{S1}, S_{S2}, S_{S3} と表記する。
- ソフトウェアが指定したセクタ番号と実際のセクタ番号の対応は表 15-3 を参照。
- キャッシュの最大ウェイ数を MAXWAY と表記する。
- セクタ S_{F0}, S_{F1}, S_{F2}, S_{F3}, S_{S0}, S_{S1}, S_{S2}, S_{S3} の最大許容量を max(S_{Xi})と表記する。これはセクタキャッシュ制御レジスタのフィールドそのものではなく、説明用の仮想的なフィールドである。両者の関係は 15.5.3 を参照。
- セクタ S_{Xi} が使用しているウェイ数を use(S_{Xi})で表わす。
- キャッシュから追い出すウェイを選択する操作を replace(S_{Xi})で表わす。

15.5.2. セクタ番号

L1 キャッシュではソフトウェアが指定する 0, 1, 2, 3 のセクタ番号がそれぞれ S_{F0}, S_{F1} ,S_{F2}, S_{F3} に対応する。

L2 キャッシュではソフトウェアが指定する 2,3 のセクタ番号は 0,1 に読み替えられる。このセクタ番号が 2 組 S_{S0}, S_{S1} と S_{S2}, S_{S3} のどちらに対応するかは、仮想セクタキャッシュ制御レジスタと実体セクタキャッシュ制御レジスタの対応づけにより決定される。仮想セクタキャッシュ制御レジスタ ASI_VSCCR にセクタキャッシュ制御レジスタ 0 を対応させている場合は S_{S0},

S_{S1}が選択され、セクタキャッシュ制御レジスタ 1 を対応させている場合は、S_{S2}, S_{S3}が選択される。

表 15-3 にソフトウェアのセクタ指定と設定値に対応する各キャッシュのセクタ番号を示す。表中の default_sector は default_sector_h と default_sector を結合して 2 ビットとして扱った(default_sector_h::default_sector)値である。

表 15-3　SCCR_ASSIGN の設定とセクタ番号の対応

sccr_assign	ソフトウェアのセクタ指定	vsccr_enable = 0	vsccr_enable = 0	vsccr_enable = 0	vsccr_enable = 0
		default_sector = 00	default_sector = 01	default_sector = 10	default_sector = 11
0	0 指定	S_{F0}, S_{S0}	S_{F1}, S_{S1}	S_{F2}, S_{S0}	S_{F3}, S_{S1}
	1 指定	S_{F0}, S_{S0}	S_{F1}, S_{S1}	S_{F2}, S_{S0}	S_{F3}, S_{S1}
	2 指定	S_{F0}, S_{S0}	S_{F1}, S_{S1}	S_{F2}, S_{S0}	S_{F3}, S_{S1}
	3 指定	S_{F0}, S_{S0}	S_{F1}, S_{S1}	S_{F2}, S_{S0}	S_{F3}, S_{S1}
	指定なし	S_{F0}, S_{S0}	S_{F1}, S_{S1}	S_{F2}, S_{S0}	S_{F3}, S_{S1}
1	0 指定	S_{F0}, S_{S2}	S_{F1}, S_{S3}	S_{F2}, S_{S2}	S_{F3}, S_{S3}
	1 指定	S_{F0}, S_{S2}	S_{F1}, S_{S3}	S_{F2}, S_{S2}	S_{F3}, S_{S3}
	2 指定	S_{F0}, S_{S2}	S_{F1}, S_{S3}	S_{F2}, S_{S2}	S_{F3}, S_{S3}
	3 指定	S_{F0}, S_{S2}	S_{F1}, S_{S3}	S_{F2}, S_{S2}	S_{F3}, S_{S3}
	指定なし	S_{F0}, S_{S2}	S_{F1}, S_{S3}	S_{F2}, S_{S2}	S_{F3}, S_{S3}

sccr_assign	ソフトウェアのセクタ指定	vsccr_enable = 1	vsccr_enable = 1	vsccr_enable = 1	vsccr_enable = 1
		default_sector = 00	default_sector = 01	default_sector = 10	default_sector = 11
0	0 指定	S_{F0}, S_{S0}	S_{F0}, S_{S0}	S_{F0}, S_{S0}	S_{F0}, S_{S0}
	1 指定	S_{F1}, S_{S1}	S_{F1}, S_{S1}	S_{F1}, S_{S1}	S_{F1}, S_{S1}
	2 指定	S_{F2}, S_{S0}	S_{F2}, S_{S0}	S_{F2}, S_{S0}	S_{F2}, S_{S0}
	3 指定	S_{F3}, S_{S1}	S_{F3}, S_{S1}	S_{F3}, S_{S1}	S_{F3}, S_{S1}
	指定なし	S_{F0}, S_{S0}	S_{F1}, S_{S1}	S_{F2}, S_{S0}	S_{F3}, S_{S1}
1	0 指定	S_{F0}, S_{S2}	S_{F0}, S_{S2}	S_{F0}, S_{S2}	S_{F0}, S_{S2}
	1 指定	S_{F1}, S_{S3}	S_{F1}, S_{S3}	S_{F1}, S_{S3}	S_{F1}, S_{S3}
	2 指定	S_{F2}, S_{S2}	S_{F2}, S_{S2}	S_{F2}, S_{S2}	S_{F2}, S_{S2}
	3 指定	S_{F3}, S_{S3}	S_{F3}, S_{S3}	S_{F3}, S_{S3}	S_{F3}, S_{S3}
	指定なし	S_{F0}, S_{S2}	S_{F1}, S_{S3}	S_{F2}, S_{S2}	S_{F3}, S_{S3}

15.5.3. セクタの最大許容量の算出

L1 キャッシュのセクタの最大許容量の算出を表 15-4 に示す。l1_sec0_max, l1_sec1_max, l1_sec2_max, l1_sec3_max はセクタキャッシュ制御レジスタ 0 のフィールドを意味する。l1_sec4_max, l1_sec5_max, l1_sec6_max, l1_sec7_max はそれぞれセクタキャッシュ制御レジスタ 1 の l1_sec0_max, l1_sec1_max, l1_sec2_max, l1_sec3_max を意味する。

表 15-4　L1 キャッシュのセクタの最大許容量

sccr_assign	最大許容量			
	max(S_{F0})	max(S_{F1})	max(S_{F2})	max(S_{F3})
0	l1_sec0_max	l1_sec1_max	l1_sec2_max	l1_sec3_max
1	l1_sec4_max	l1_sec5_max	l1_sec6_max	l1_sec7_max

L2 キャッシュのセクタの最大許容量の算出を表 15-4 に示す。l2_sec0_max, l2_sec1_max はセクタキャッシュ制御レジスタ 0 のフィールドを意味する。l2_sec2_max, l2_sec3_max はそれぞれセクタキャッシュ制御レジスタ 1 の l2_sec0_max, l2_sec1_max を意味する。

表 15-5　L2 キャッシュのセクタの最大許容量

最大許容量			
$\max(S_{S0})$	$\max(S_{S1})$	$\max(S_{S2})$	$\max(S_{S3})$
l2_sec0_max	l2_sec1_max	l2_sec2_max	l2_sec3_max

15.5.4. セクタキャッシュ機能の有効・無効

セクタキャッシュ機能が有効かどうかは、セクタキャッシュ制御レジスタにより設定されるセクタの最大許容量によって決まる。アクセスするセクタの最大許容量に 1 以上が設定された場合、セクタキャッシュ機能は有効となる。最大許容量が 0 であった場合は、セクタは無効である。アクセスするセクタが無効であった場合は、セクタを指定していないかのように振舞う。

- L1 キャッシュでは n=0,1,2,3 で
  - $\max(S_{Fn}) = 0$ のとき、セクタ n のセクタキャッシュ機能は無効である
  - $\max(S_{Fn}) > 0$ のとき、セクタ n のセクタキャッシュ機能は有効である
- L2 キャッシュでは n=0,1,2,3 で
  - $\max(S_{Sn}) = 0$ のとき、セクタ n のセクタキャッシュ機能は無効である
  - $\max(S_{Sn}) > 0$ のとき、セクタ n のセクタキャッシュ機能は有効である

15.5.5. セクタキャッシュ管理動作

【L1 キャッシュ】

セクタ ID=0 のリクエストがコアから発行された場合, $\max(S_{F0}) >= 1$ ならばセクタキャッシュ機能は有効. そうでなければ通常の LRU 処理.

セクタ ID=1 のリクエストがコアから発行された場合, $\max(S_{F1}) >= 1$ ならばセクタキャッシュ機能は有効. そうでなければ通常の LRU 処理.

セクタ ID=2 のリクエストがコアから発行された場合, $\max(S_{F2}) >= 1$ ならばセクタキャッシュ機能は有効. そうでなければ通常の LRU 処理.

セクタ ID=3 のリクエストがコアから発行された場合, $\max(S_{F3}) >= 1$ ならばセクタキャッシュ機能は有効. そうでなければ通常の LRU 処理.

【L2 キャッシュ】

SCCR0 かつセクタ ID=0 もしくは ID=2 のリクエストがストレージ制御ユニットから発行された場合, $\max(S_{S0}) >= 1$ ならばセクタキャッシュ機能は有効. そうでなければ通常の LRU 処理.

SCCR0 かつセクタ ID=1 もしくは ID=3 のリクエストがストレージ制御ユニットから発行された場合, $\max(S_{S1}) >= 1$ ならばセクタキャッシュ機能は有効. そうでなければ通常の LRU 処理.

SCCR1 かつセクタ ID=0 もしくは ID=2 のリクエストがストレージ制御ユニットから発行された場合, $\max(S_{S2}) >= 1$ ならばセクタキャッシュ機能は有効. そうでなければ通常の LRU 処理.

SCCR1 かつセクタ ID=1 もしくは ID=3 のリクエストがストレージ制御ユニットから発行された場合, max(S_{S3}) >= 1 ならばセクタキャッシュ機能は有効. そうでなければ通常の LRU 処理.

15.6. レジスタ

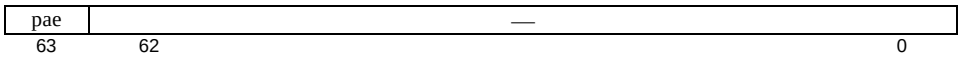
15.6.1. 特権アクセスレジスタ制御レジスタ

15.6.1.1. ASI_PRIV_SCCR_CTRL

レジスタ名	ASI_PRIV_SCCR_CTRL
ASI 番号	$E7_{16}$
VA	$3F8_{16}$
共有範囲	VCPU

アクセス

	read	write
user	*privileged_action*	*privileged_action*
priv	OK	*privileged_action*



ビット	フィールド	アクセス	説明
63	pae	RO	1: 特権アクセスレジスタへのアクセス可能 0: 特権アクセスレジスタへのアクセス不可

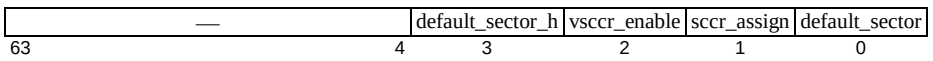
15.6.2. セクタキャッシュ割り当て設定レジスタ

15.6.2.1. ASI_PRIV_SCCR_ASSIGN

レジスタ名	ASI_PRIV_SCCR_ASSIGN
ASI 番号	$E7_{16}$
VA	300_{16}
共有範囲	VCPU

アクセス

	read	write
user	*privileged_action*	*privileged_action*
priv	*privileged_action* or OK	*privileged_action* or OK



ビット	フィールド	アクセス	説明
3	default_sector_h	RW	命令フェッチおよびセクタ番号を指定しないメモリアクセス命令で使われる、暗黙のセクタ番号を指

			定する。 L1 キャッシュにアクセスする際のセクタ番号の bit<1>に使われる。 L2 キャッシュにアクセスする際はこの設定は無視される。
2	vsccr_enable	RW	非特権モードおよび特権モードの ASI_VSCCR へのアクセスを許可し、XAR.sector 及び XAR.sector_h によるセクタ番号指示を有効にする。 • vsccr_enable = 1 のとき、ASI_VSCCR に特権または非特権モードでアクセスできる。XAR.sector 及び XAR.sector_h でセクタ番号指示ができる。 vsccr_enable = 0 のとき、ASI_VSCCR に非特権モードでアクセスすると *privileged_action* 例外が発生する。vsccr_enable = 0 かつ ASI_PRIV_SCCR_CTRL.pae = 0 のとき、ASI_VSCCR に特権モードでアクセスすると *privileged_action* 例外が発生する。*privileged_action* 例外発生時には、XAR.sector 及び XAR.sector_h のセクタ番号指示は無効。
1	sccr_assign	RW	この VCPU のメモリアクセスによるセクタキャッシュ制御を、2 つあるセクタキャッシュ設定レジスタ ASI_PRIV_SCCR0, ASI_PRIV_SCCR1 のどちらに基づいて行なうかを指定する。このフィールドが 0 なら ASI_PRIV_SCCR0 が、1 なら ASI_PRIV_SCCR1 が使われる。 vsccr_enable = 1 のときは、ASI_VSCCR でアクセスできるレジスタの実体もこのフィールドにより決まる。
0	default_sector	RW	命令フェッチおよびセクタ番号を指定しないメモリアクセス命令で使われる、暗黙のセクタ番号を指定する。 L1 キャッシュにアクセスする際のセクタ番号の bit<0>に使われる。 L2 キャッシュにアクセスする際のセクタ番号の bit<0>に使われる。

ASI_PRIV_SCCR_ASSIGN は、セクタキャッシュ設定レジスタの割り当てを指示するレジスタである。sccr_assign で二組あるセクタキャッシュ設定レジスタのどちらを使うかを指定し、vsccr_enable で非特権および特権モードのソフトウェアがセクタキャッシュを操作するための窓を設定する。vsccr_enable はさらに XAR.sector 及び XAR.sector_h によるセクタ指示の有効無効を切り替える。

特権モードおよび非特権モードのソフトウェアによるセクタキャッシュ機能の使用を許可するには、sccr_assign でセクタキャッシュ制御レジスタを選択し、vsccr_enable に 1 を設定する。このとき、特権モードおよび非特権モードの ASI_VSCCR へのアクセスが許可され、XAR.sector 及び XAR.sector_h によるセクタ指示が有効となる。VCPU からのメモリアクセスには選択されたセクタキャッシュ制御レジスタに対応する L1 キャッシュで 4 種, L2 キャッシュで 2 種のセクタ番号のいずれかが付加される。

特権モードおよび非特権モードのソフトウェアによるセクタキャッシュ機能使用を禁止するには、sccr_assign で使用するセクタキャッシュ制御レジスタを選択し、vsccr_enable に 0 を設定する。このとき、特権モードおよび非特権モードの ASI_VSCCR へのアクセスが禁止され、XAR.sector 及び XAR.sector_h によるセクタ指示が無効となる。VCPU からのメモリアクセスに付加されるセクタ指示は、選択されたセクタキャッシュ制御レジスタと default_sector 及び default_sector_h で指定される 1 種のみとなる。

表 15-3 に vsccr_enable, sccr_assign, default_sector, default_sector_h の設定とセクタ番号の関係を示す。

Note vsccr_enable の設定とは関係なく、すべての特権レベルのメモリアクセスに必ずセクタ情報が付随することに注意。

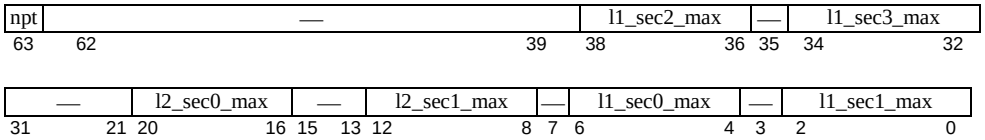
ASI_PRIV_SCCR_CTRL.pae = 0 のとき特権モードにより ASI_PRIV_SCCR_ASSIGN にアクセスした場合、*privileged_action* 例外を検出する。

15.6.3. セクタキャッシュ制御レジスタ

15.6.3.1. セクタキャッシュ制御レジスタ（ASI_PRIV_SCCR0, ASI_PRIV_SCCR1）

レジスタ名	ASI_PRIV_SCCR0, ASI_PRIV_SCCR1
ASI 番号	$E7_{16}$
VA	310_{16}, 318_{16}
共有範囲	VCPU (実体は CMG)
アクセス	

	read	write
user	*privileged_action*	*privileged_action*
priv	*privileged_action* or OK	*privileged_action* or OK



ビット	フィールド	アクセス	説明
63	npt	RW	ASI_VSCCR の説明を参照。
38:36	l1_sec2_max	RW	L1 キャッシュのセクタ 2 の最大ウェイ数。
34:32	l1_sec3_max	RW	L1 キャッシュのセクタ 3 の最大ウェイ数。
20:16	l2_sec0_max	RW	L2 キャッシュのセクタ 0 の最大ウェイ数。
12:8	l2_sec1_max	RW	L2 キャッシュのセクタ 1 の最大ウェイ数。
6:4	l1_sec0_max	RW	L1 キャッシュのセクタ 0 の最大ウェイ数。
2:0	l1_sec1_max	RW	L1 キャッシュのセクタ 1 の最大ウェイ数。

ASI_PRIV_SCCR{0|1}はセクタキャッシュの設定を行うレジスタである。特権モードからは ASI_PRIV_SCCR{0|1}に直接アクセスするか、もしくは ASI_VSCCR を通してアクセスする。非特権モードからは、ASI_VSCCR を通してアクセスする。

ASI_PRIV_SCCR_CTRL.pae = 0 のとき特権モードにより、ASI_PRIV_SCCR{0|1}にアクセスした場合、*privileged_action* 例外を検出する。

15.6.4. 仮想セクタキャッシュ制御レジスタ（窓レジスタ）

レジスタ名　ASI_VSCCR

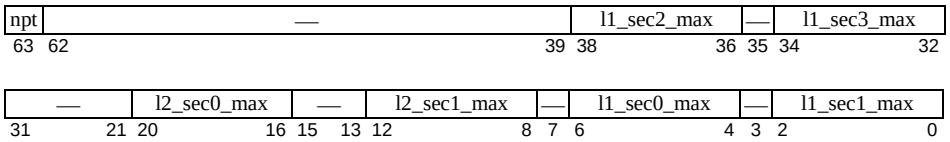
ASI 番号　$E7_{16}$

VA　00_{16}

共有範囲　VCPU (実体は CMG)

アクセス

	read	write
user	OK or *privileged_action*	OK or *privileged_action*
priv	OK or *privileged_action*	OK or *privileged_action*



ビット	フィールド	アクセス	説明
63	npt	RW (priv) RO (user)	非特権モードのアクセスを許すかどうか。 • npt = 0 のとき、ASI_VSCCR に非特権モードでアクセスできる。 • npt = 1 のとき、ASI_VSCCR に非特権モードでアクセスすると *privileged_action* 例外が発生する。 非特権モードのソフトウェアが npt を変更することはできない。
38:36	l1_sec2_max	RW	ASI_PRIV_SCCR{0|1}の説明参照。
34:32	l1_sec3_max	RW	ASI_PRIV_SCCR{0|1}の説明参照。
20:16	l2_sec0_max	RW	ASI_PRIV_SCCR{0|1}の説明参照。
12:8	l2_sec1_max	RW	ASI_PRIV_SCCR{0|1}の説明参照。
6:4	l1_sec0_max	RW	ASI_PRIV_SCCR{0|1}の説明参照。
2:0	l1_sec1_max	RW	ASI_PRIV_SCCR{0|1}の説明参照。

ASI_VSCCR は特権および非特権モードのソフトウェアがセクタキャッシュの設定を参照・変更するための窓 ASI である。セクタキャッシュ制御レジスタ 0 またはセクタキャッシュ制御レジスタ 1 の全ビットが一対一にマップされて見える。セクタキャッシュ制御レジスタ 0, セクタキャッシュ制御レジスタ 1 のどちらにマップされるかはセクタキャッシュ割り当て設定レジスタの sccr_assign によって決まる。

セクタキャッシュ割り当て設定レジスタの vsccr_enable が 0 かつ、非特権レジスタアクセス制御レジスタの pae が 0 のとき、特権モードでアクセスすると *privileged_action* 例外が発生する。これ以外の時、特権モードで ASI_VSCCR からセクタキャッシュ割り当て設定レジスタの sccr_assign で選択されたセクタキャッシュ制御レジスタ 0 と 1 の全てのフィールドを参照・更新することができる。

また、セクタキャッシュ割り当て設定レジスタの vsccr_enable が 0 もしくは npt が 1 のとき非特権モードでアクセスすると *privileged_action* 例外が発生する。これ以外のとき、非特権モードのソフトウェアは、ASI_VSCCR からセクタキャッシュ割り当て設定レジスタの sccr_assign で選択されたセクタキャッシュ制御レジスタ 0 と 1 の全てのフィールドを参照でき、npt 以外のフィールドを更新できる。npt を 1 に更新しようとしても npt には 0 が保持される。

ASI_VSCCR への各モードからのアクセスの可否について表 15-6 にまとめた。

表 15-6　設定値と ASI_VSCCR アクセスの可否

設定			アクセス(read/write)	
vsccr_enable	pae	npt	user	priv
0	0	0 or 1	*privileged_action*	*privileged_action*
0	1	0 or 1	*privileged_action*	OK
1	0 or 1	0	OK	OK
1	0 or 1	1	*privileged_action*	OK

15.7. 使用例

本節では、L2 キャッシュメモリでのセクタキャッシュの使用例を 3 種類示す。

使用例 1

アシスタントコアが使用するウェイ数を制限しながら、演算コアで 2 セクタのセクタキャッシュを使用するときの使用例を示す。

演算コアは SCCR0, アシスタントコアは SCCR1 を使用する。図 15-2 のように各コアの SCCR を割り当て、最大許容量を設定することで、24 ウェイの L2 キャッシュを 3 分割して利用可能である。このとき、アシスタントコアは S2(SCCR1 の SEC0)のみを使用する。もし、S3（SCCR1 の SEC1）を指定した場合、そのアクセスは全ウェイがリプレース対象となり他の領域を侵食する可能性がある。

CMG ごとに異なる設定が可能であるため、図 15-2 では、異なる割り当てを実施している。

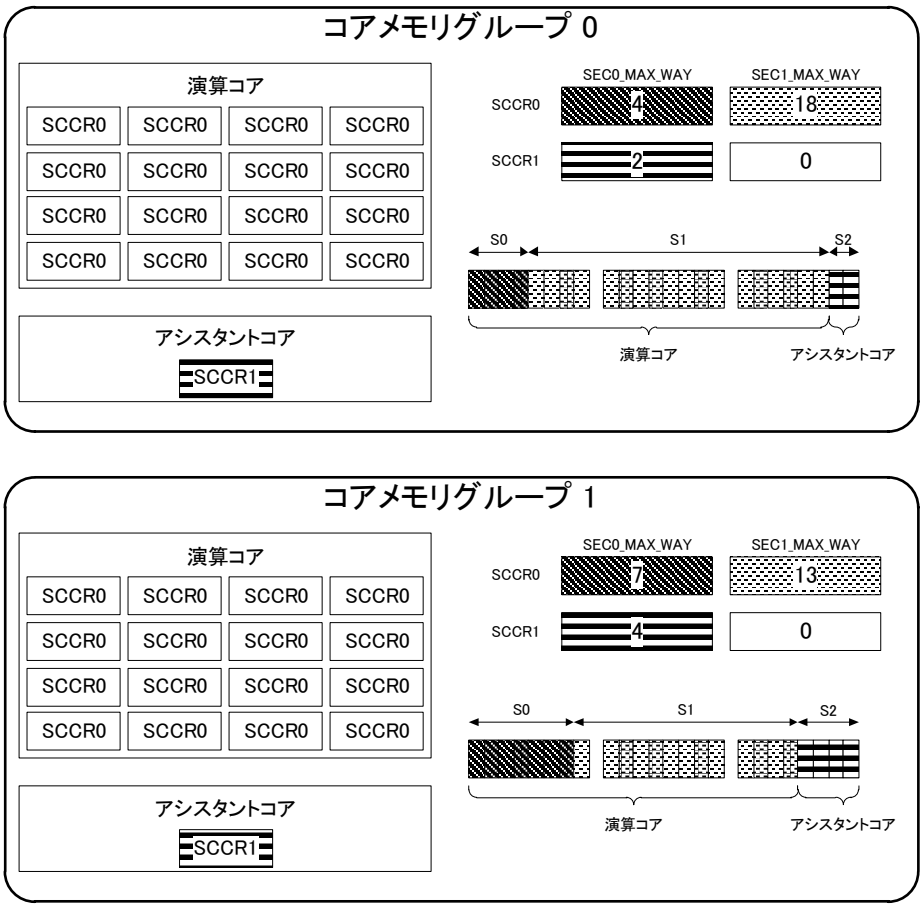


図 15-2　使用例 1

使用例 2

演算コアはセクタを使用せず、アシスタントコアのみ制限するときの使用例を示す。

演算コアは SCCR0, アシスタントコアは SCCR1 を使用する。図 15-3 のように、CMG0 ではアシスタントコアに 2 ウェイを設定している。演算コアのセクタ機能は無効となるので、リプ

レースが発生するとアシスタントコアの領域を侵食することができる。これにより、演算コアに 24 ウェイが設定される。

CMG ごとに異なる設定が可能なので、CMG1 ではアシスタントコアに 4 ウェイを設定している。

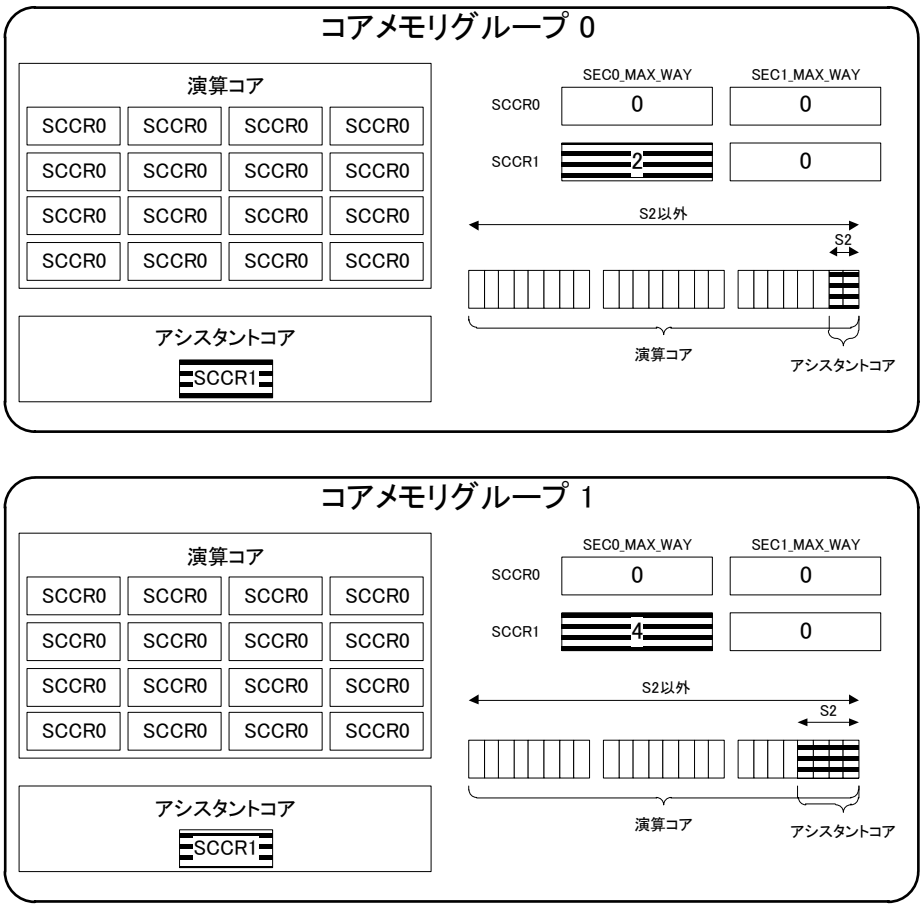


図 15-3　使用例 2

使用例 3

演算コアとアシスタントコアの使用可能なウェイを完全に分割するときの設定を示す。

演算コアは SCCR0, アシスタントコアは SCCR1 を使用する。図 15-4 のように、両 CMG ともに演算コアに 20 ウェイを、アシスタントコアに 4 ウェイを設定している。S0, S2 のみを使用することで、完全分割を実現する。

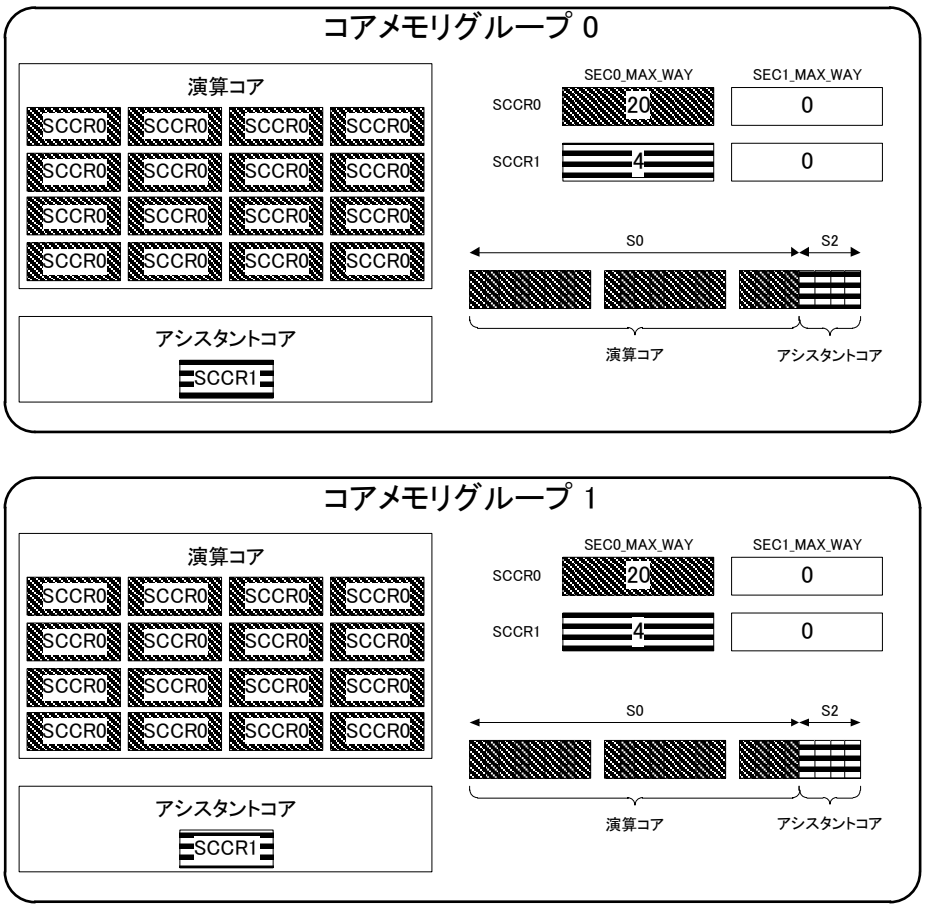

コアメモリグループ 0
演算コア
SCCR0
アシスタントコア
SCCR1
SEC0_MAX_WAY
SEC1_MAX_WAY
SCCR0
20
0
SCCR1
4
0
S0
S2
演算コア
アシスタントコア
コアメモリグループ 1
演算コア
SCCR0
アシスタントコア
SCCR1
SEC0_MAX_WAY
SEC1_MAX_WAY
SCCR0
20
0
SCCR1
4
0
S0
S2
演算コア
アシスタントコア

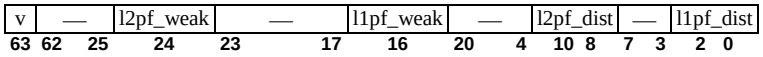
図 15-4　使用例 3

16. Configuration and Diagnostics Support

16.1. Hardware Prefetch Control Register

レジスタ名　ASI_HWPF_CTRL
ASI 番号　$E7_{16}$
VA　200_{16}
共有範囲　VCPU
アクセス

	read	write
user	OK	OK
priv	OK	OK



ビット	フィールド	アクセス	説明
63	v	RW	ASI_HWPF_CTRL レジスタの設定値を有効化 0_2: 無効（デフォルトのハードウェアプリフェッチ） 1_2: 有効
24	l2pf_weak	RW	L2 キャッシュのハードウェアプリフェッチモードを指定する 0_2: “strong” プリフェッチで生成する 1_2: “weak” プリフェッチで生成する
16	l1pf_weak	RW	L1 キャッシュのハードウェアプリフェッチモードを指定する 0_2: “strong” プリフェッチで生成する 1_2: “weak” プリフェッチで生成する
10:8	l2pf_dist	RW	L2 キャッシュのハードウェアプリフェッチの距離を指定する。 プリフェッチ距離は l2pf_dist * 1024B で指定される。 l2pf_dist = 0 と指定したとき、L2 キャッシュのハードウェアプリフェッチは抑止される。
2:0	l1pf_dist	RW	L1 キャッシュのハードウェアプリフェッチの距離を指定する。 プリフェッチ距離は l1pf_dist * 256B で指定される。 l1pf_dist = 0 と指定したとき、L1 キャッシュのハードウェアプリフェッチは抑止される。

Programming Note　レジスタ共有範囲が VCPU であり、コンテキストスイッチ時の復元退避対象レジスタとする。

SPARC64™ XIfx にはハードウェアプリフェッチ機能が備わっている。本機能は、連続するキャッシャブルアドレスに対するアクセスを検出し、それらがキャッシュミスした場合にハードウェアがプリフェッチを生成する。連続するキャッシャブルアドレスとはキャッシュライン単位（256Byte）であることに注意が必要である。昇順・降順ともに検出可能だが、アドレスがキャッシュライン単位で連続しないケースでは動作しない。

本レジスタを設定することで、ハードウェアプリフェッチの動作を制御できる。設定可能な項目は、L1 キャッシュと L2 キャッシュ独立に、ハードが生成するプリフェッチモード及び、プリフェッチの距離を指定することが可能である。

本レジスタのフィールドにおいて、v = 1, l1pf_dist = 0 及び l2pf_dist = 0 を設定することで、ハードウェアプリフェッチ機能の動作を抑止することが可能である。

命令単位でハードウェアプリフェッチを無効化する場合は、XAR.dis_hw_pf を設定する（XAR の章を参照）。

本レジスタのフィールドにおいて v = 0 と設定すると、ハードウェアプリフェッチ機能はハードの Default 値で動作する。Default 値は、l1pf_dist = 0x3 (768B), l2pf_dist = 0x4 (4KB), l1pf_weak = 0 (Strong), l2pf_weak = 0 (Strong)である。

17. Opcode Maps

この章では、SPARC64™ XIfx で使用可能な命令セットのオペコード表を提供する。

表中、横棒 (—) になっているオペコードは予約されたオペコードである。予約されたオペコードを実行しようとすると、*illegal_instruction* 例外が発生する。また表中、アスタリスク (*) になっているオペコードは XAR レジスタとの組み合わせで発行できないオペコードである。このオペコードを実行しようとすると、*illegal_action* 例外が発生する。

表 17-1　op<1:0>

op<1:0>			
0	1	2	3
分岐命令、SETHI と SXAR 表 17-2 参照	CALL	演算命令その他 表 17-3 参照	メモリアクセス命令 表 17-4 参照

表 17-2　分岐命令, SETHI, SXAR (op<1:0> = 0)

op2<2:0>							
0	1	2	3	4	5	6	7
ILLTRAP	BPcc 表 17-16 参照	Bicc[D] 表 17-16 参照	BPr 表 17-17 参照	SETHI, NOP	FBPfcc 表 17-16 参照	FBfcc[D] 表 17-16 参照	SXAR1, SXAR2

表 17-3　演算命令その他 (op<1:0> = 2)

op3<3:0>	op3<5:4>			
	0	1	2	3
0	ADD	ADDcc	TADDcc	WRY[D] (rd = 0) WRCCR (rd = 2) WRASI (rd = 3) WRFPRS (rd = 6) WRPCR[PPCR] (rd = 16) WRPIC[PPIC] (rd = 17) WRGSR (rd = 19) WRXAR (rd = 29) WRXASR (rd = 30)
1	AND	ANDcc	TSUBcc	
2	OR	ORcc	TADDccTV[D]	
3	XOR	XORcc	TSUBccTV[D]	
4	SUB	SUBcc	MULScc[D]	FPop1 (表 17-13, 表 17-14 参照)
5	ANDN	ANDNcc	SLL (x = 0, r = 0), SLLX (x = 1, r = 0), ROLX (x = 1, r = 1)	FPop2 (表 17-15 参照)
6	ORN	ORNcc	SRL (x = 0), SRLX (x = 1)	IMPDEP1 (表 17-21, 表 17-22, 表 17-23 参照)
7	XNOR	XNORcc	SRA (x = 0), SRAX (x = 1)	IMPDEP2 (表 17-24 参照)
8	ADDC	ADDCcc	RDY[D] (rs1 = 0, i = 0) RDCCR (rs1 = 2, i = 0) RDASI (rs1 = 3, i = 0) RDTICK[PNPT] (rs1 = 4, i = 0) RDPC (rs1 = 5, i = 0) RDFPRS (rs1 = 6, i = 0) MEMBAR (rs1 = 15, rd = 0, i = 1) STBAR (rs1 = 15, rd = 0, i = 0) RDPCR[PPCR] (rs1 = 16, i = 0) RDPIC[PPIC] (rs1 = 17, i = 0) RDGSR (rs1 = 19, i = 0) RDSTICK (rs1 = 24, i = 0) RDXASR (rs1 = 30, i = 0)	JMPL
9	MULX	—		RETURN
A_{16}	UMUL[D]	UMULcc[D]		Tcc
B_{16}	SMUL[D]	SMULcc[D]	FLUSHW	FLUSH
C_{16}	SUBC	SUBCcc	MOVcc	SAVE
D_{16}	UDIVX	—	SDIVX	RESTORE
E_{16}	UDIV[D]	UDIVcc[D]	POPC (rs1 = 0)	
F_{16}	SDIV[D]	SDIVcc[D]	MOVR (rs1 = 0)	—

表 17-4　メモリアクセス命令（op<1:0> = 3）

op3 <3:0>	op3<5:4>			
	0	1	2	3
0	LDUW	LDUWA$^{P_{ASI}}$	LDF((i = 0 **and** id = 0) **or** (i = 1)) 表 17-5 参照 LDFID(i = 0 and id = 1)表 17-11 参照	LDFA$^{P_{ASI}}$
1	LDUB	LDUBA$^{P_{ASI}}$	LDFSR^{D} (rd = 0) LDXFSR (rd = 1) LDXEFSR (rd = 3)	—
2	LDUH	LDUHA$^{P_{ASI}}$	LDQF	LDQFA$^{P_{ASI}}$
3	LDTW^{D} (rd even)	LDTWA$^{D,P_{ASI}}$ (rd even) LDTXA (rd even)	LDDF ((i = 0 **and** id = 0) **or** (i = 1)) 表 17-6 参照 LDDFID (i = 0 and id = 1) 表 17-11 参照	LDDFA$^{P_{ASI}}$ LDBLOCKF LDSHORTF
4	STW	STWA$^{P_{ASI}}$ XFILL256 XFILL^{N}	STF ((i = 0 **and** id = 0) **or** (i = 1)) 表 17-7 参照 STFID (i = 0 and id = 1) 表 17-12 参照	STFA$^{P_{ASI}}$ XFILL256 XFILL^{N}
5	STB	STBA$^{P_{ASI}}$ XFILL256 XFILL^{N}	STFSR^{D} (rd = 0) STXFSR (rd = 1)	—
6	STH	STHA$^{P_{ASI}}$ XFILL256 XFILL^{N}	STQF	STQFA$^{P_{ASI}}$
7	STTW^{D} (rd even)	STTWA$^{D,P_{ASI}}$ (rd even) STBI^{N} XFILL256 XFILL^{N}	STDF ((i = 0 **and** id = 0) **or** (i = 1)) 表 17-8 参照 STDFID (i = 0 and id = 1) 表 17-12 参照	STDFA$^{P_{ASI}}$ STBLOCKF STPARTIALF STSHORTF XFILL256 XFILL^{N}
8	LDSW	LDSWA$^{P_{ASI}}$	—	—
9	LDSB	LDSBA$^{P_{ASI}}$	—	—
A_{16}	LDSH	LDSHA$^{P_{ASI}}$	—	—
B_{16}	LDX	LDXA$^{P_{ASI}}$	—	—
C_{16}	—	—	STFR ((i = 0 **and** id = 0) **or** (i = 1))表 17-9 参照 STFRID (i = 0 and id = 1) 表 17-12 参照	CASA$^{P_{ASI}}$
D_{16}	LDSTUB	LDSTUBA$^{P_{ASI}}$	PREFETCH((i = 0 **and** id = 0) **or** (i = 1)) PREFETCHID (i = 0 and id = 1)	PREFETCHA$^{P_{ASI}}$
E_{16}	STX	STXA$^{P_{ASI}}$ STBI^{N} XFILL256 XFILL^{N}	—	CASXA$^{P_{ASI}}$
F_{16}	SWAP^{D}	SWAPA$^{D,P_{ASI}}$	STDFR ((i = 0 **and** id = 0) **or** (i = 1)) 表 17-10 参照 STDFRID (i = 0 and id = 1) 表 17-12 参照	—

表 17-5　LDF 命令

XAR.simd	XAR.urd<2>	XAR.urs1<2:1>	XAR.urs2<2:1>			
			0	1	2	3
0	0	0	LDF	LDFUW	LDFIB	LDFSW
0	0	1	*	*	*	*
0	0	2	*	*	*	*
0	0	3	*	*	*	*
0	1	0	LDF	LDFUW	LDFIB	LDFSW
0	1	1	*	*	*	*
0	1	2	*	*	*	*
0	1	3	*	*	*	*
1	0	0	LDF	LDFUW	LDFIB	LDFSW
1	0	1	LDFBC	LDFBCUW	LDFBCIB	LDFBCSW
1	0	2	LDFST	LDFSTUW	LDFSTIB	LDFSTSW
1	0	3	LDFST	LDFSTUW	LDFSTIB	LDFSTSW
1	1	0	LDFST	LDFSTUW	LDFSTIB	LDFSTSW
1	1	1	LDFST	LDFSTUW	LDFSTIB	LDFSTSW
1	1	2	LDFST	LDFSTUW	LDFSTIB	LDFSTSW
1	1	3	LDFST	LDFSTUW	LDFSTIB	LDFSTSW

表 17-6　LDDF 命令

XAR.simd	XAR.urd<2>	XAR.urs1<2:1>	XAR.urs2<2:1>			
			0	1	2	3
0	0	0	LDDF	LDDFDS	*	*
0	0	1	*	*	*	*
0	0	2	*	*	*	*
0	0	3	*	*	*	*
0	1	0	LDDF	LDDFDS	*	*
0	1	1	*	*	*	*
0	1	2	*	*	*	*
0	1	3	*	*	*	*
1	0	0	LDDF	LDDFDS	*	*
1	0	1	LDDFBC	*	*	*
1	0	2	LDDFST	*	*	*
1	0	3	LDDFST	*	*	*
1	1	0	LDDFST	*	*	*
1	1	1	LDDFST	*	*	*
1	1	2	LDDFST	*	*	*
1	1	3	LDDFST	*	*	*

表 17-7　STF 命令

XAR.simd	urd<2>	urs1<2:1>	urs2<2:1>			
			0	1	2	3
0	0	0	STF	STFUW	*	*
0	0	1	*	*	*	*
0	0	2	*	*	*	*
0	0	3	*	*	*	*
0	1	0	STF	STFUW	*	*
0	1	1	*	*	*	*
0	1	2	*	*	*	*
0	1	3	*	*	*	*
1	0	0	STF	STFUW	*	*
1	0	1	*	*	*	*
1	0	2	STFST	STFSTUW	*	*
1	0	3	STFST	STFSTUW	*	*
1	1	0	STFST	STFSTUW	*	*
1	1	1	STFST	STFSTUW	*	*
1	1	2	STFST	STFSTUW	*	*
1	1	3	STFST	STFSTUW	*	*

表 17-8　STDF 命令

XAR.simd	urd<2>	urs1<2:1>	urs2<2:1>			
			0	1	2	3
0	0	0	STDF	STDFDS	*	*
0	0	1	*	*	*	*
0	0	2	*	*	*	*
0	0	3	*	*	*	*
0	1	0	STDF	STDFDS	*	*
0	1	1	*	*	*	*
0	1	2	*	*	*	*
0	1	3	*	*	*	*
1	0	0	STDF	STDFDS	*	*
1	0	1	*	*	*	*
1	0	2	STDFST	*	*	*
1	0	3	STDFST	*	*	*
1	1	0	STDFST	*	*	*
1	1	1	STDFST	*	*	*
1	1	2	STDFST	*	*	*
1	1	3	STDFST	*	*	*

表 17-9　STFR 命令(i = 0)

XAR.simd	urd<2>	urs1<2:1>	IW<11:10>			
			0	1	2	3
0	0	0	STFR	STFRUW	*	*
0	0	1	*	*	*	*
0	0	2	*	*	*	*
0	0	3	*	*	*	*
0	1	0	STFR	STFRUW	*	*
0	1	1	*	*	*	*
0	1	2	*	*	*	*
0	1	3	*	*	*	*
1	0	0	STFR	STFRUW	*	*
1	0	1	*	*	*	*
1	0	2	STFRST	STFRSTUW	*	*
1	0	3	STFRST	STFRSTUW	*	*
1	1	0	STFRST	STFRSTUW	*	*
1	1	1	STFRST	STFRSTUW	*	*
1	1	2	STFRST	STFRSTUW	*	*
1	1	3	STFRST	STFRSTUW	*	*

表 17-10 STDFR 命令(i = 0)

XAR.simd	urd<2>	urs1<2:1>	IW<11:10>			
			0	1	2	3
0	0	0	STDFR	*	*	*
0	0	1	*	*	*	*
0	0	2	*	*	*	*
0	0	3	*	*	*	*
0	1	0	STDFR	*	*	*
0	1	1	*	*	*	*
0	1	2	*	*	*	*
0	1	3	*	*	*	*
1	0	0	STDFR	*	*	*
1	0	1	*	*	*	*
1	0	2	STDFRST	*	*	*
1	0	3	STDFRST	*	*	*
1	1	0	STDFRST	*	*	*
1	1	1	STDFRST	*	*	*
1	1	2	STDFRST	*	*	*
1	1	3	STDFRST	*	*	*

表 17-11 LDFID/LDDFID 命令

XAR.simd	type	LDFID	LDDFID
0	0	LDFID	LDDFID
0	1	LDFIDUW	—
0	2	—	—
0	3	LDFIDSW	—
1	0	LDFID	LDDFID
1	1	LDFIDUW	—
1	2	—	—
1	3	LDFIDSW	—

表 17-12 STFID/STDFID/STFRID/STDFRID 命令

XAR.simd	type	STFID	STDFID	STFRID	STDFRID
0	0	STFID	STDFID	STFRID	STDFRID
0	1	STFIDUW	—	STFRIDUW	—
0	2	—	—	—	—
0	3	—	—	—	—
1	0	STFID	STDFID	STFRID	STDFRID
1	1	STFIDUW	—	STFRIDUW	—
1	2	—	—	—	—
1	3	—	—	—	—

表 17-13 FPop1 (op<1:0> = 2, op3 = 34_{16}) (1/2)

opf<8:4>	opf<3:0>							
	0	1	2	3	4	5	6	7
00_{16}	—	FMOVs	FMOVd	FMOVq	—	FNEGs	FNEGd	FNEGq
01_{16}	—	—	—	—	—	—	—	—
02_{16}	—	—	—	—	—	—	—	—
03_{16}	—	—	—	—	—	—	—	—
04_{16}	FADDds	FADDs	FADDd	FADDq	FSUBds	FSUBs	FSUBd	FSUBq
05_{16}	—	FNADDs	FNADDd	—	—	—	—	—
06_{16}	—	—	—	—	—	—	—	—
07_{16}	—	—	—	—	—	—	—	—
08_{16}	—	FsTOx	FdTOx	FqTOx	FxTOs	—	—	—
09_{16}	—	—	—	—	—	—	—	—
$0A_{16}$	—	—	—	—	—	—	—	—
$0B_{16}$	—	—	—	—	—	—	—	—
$0C_{16}$	—	—	—	—	FiTOs FwTOs	—	FdTOs	FqTOs
$0D_{16}$	—	FsTOi FsTOw	FdTOi FdTOw	FqTOi	—	—	—	—
$0E_{16}$ – $1F_{16}$	—	—	—	—	—	—	—	—

表 17-14 FPop1 (op<1:0> = 2, op3 = 34_{16}) (2/2)

opf<8:4>	opf<3:0>							
	8	9	A_{16}	B_{16}	C_{16}	D_{16}	E_{16}	F_{16}
00_{16}	—	FABSs	FABSd	FABSq	—	—	—	—
01_{16}	—	—	—	—	—	—	—	—
02_{16}	—	FSQRTs	FSQRTd	FSQRTq	—	—	—	—
03_{16}	—	—	—	—	—	—	—	—
04_{16}	FMULds	FMULs	FMULd	FMULq	—	FDIVs	FDIVd	FDIVq
05_{16}	—	FNMULs	FNMULd	—	—	—	—	—
06_{16}	—	FsMULd	—	—	—	—	FdMULq	—
07_{16}	—	FNsMULd	—	—	—	—	—	—
08_{16}	FxTOd	—	—	—	FxTOq	—	—	—
09_{16}	—	—	—	—	—	—	—	—
$0A_{16}$	—	—	—	—	—	—	—	—
$0B_{16}$	—	—	—	—	—	—	—	—
$0C_{16}$	FiTOd FwTOd	FsTOd	—	FqTOd	FiTOq	FsTOq	FdTOq	—
$0D_{16}$	—	—	—	—	—	—	—	—
$0E_{16}$ – $1F_{16}$	—	—	—	—	—	—	—	—

表 17-15 FPop2 (op<1:0> = 2, op3 = 35_{16})

opf<8:4>	opf<3:0>								
	0	1	2	3	4	5	6	7	8 – F_{16}
00_{16}	—	FMOVs (fcc0)	FMOVd (fcc0)	FMOVq (fcc0)	—	— (FMOVR 拡張用に予約)			—
01_{16}	—	—	—	—	—	—	—	—	—
02_{16}	—	—	—	—	—	FMOVRsZ[iii]	FMOVRdZ[iii]	FMOVRqZ[iii]	—
03_{16}	—	—	—	—	—	—	—	—	—
04_{16}	—	FMOVs (fcc1)	FMOVd (fcc1)	FMOVq (fcc1)	—	FMOVRsLEZ[iii]	FMOVRdLEZ[iii]	FMOVRqLEZ[iii]	—
05_{16}	—	FCMPs	FCMPd	FCMPq	—	FCMPEs[iii]	FCMPEd[iii]	FCMPEq[iii]	—
06_{16}	—	—	—	—	—	FMOVRsLZ[iii]	FMOVRdLZ[iii]	FMOVRqLZ[iii]	—
07_{16}	—	—	—	—	—	—	—	—	—
08_{16}	—	FMOVs (fcc2)	FMOVd (fcc2)	FMOVq (fcc2)	—	— (FMOVR 拡張用に予約)			—
09_{16}	—	—	—	—	—	—	—	—	—
$0A_{16}$	—	—	—	—	—	FMOVRsNZ[iii]	FMOVRdNZ[iii]	FMOVRqNZ[iii]	—
$0B_{16}$	—	—	—	—	—	—	—	—	—
$0C_{16}$	—	FMOVs (fcc3)	FMOVd (fcc3)	FMOVq (fcc3)	—	FMOVRsGZ[iii]	FMOVRdGZ[iii]	FMOVRqGZ[iii]	—
$0D_{16}$	—	—	—	—	—	—	—	—	—
$0E_{16}$	—	—	—	—	—	FMOVRsGEZ[iii]	FMOVRdGEZ[iii]	FMOVRqGEZ[iii]	—
$0F_{16}$	—	—	—	—	—	—	—	—	—
10_{16}	—	FMOVs (icc)	FMOVd (icc)	FMOVq (icc)	—	—	—	—	—
11_{16} – 17_{16}	—	—	—	—	—	—	—	—	—
18_{16}	—	FMOVs (xcc)	FMOVd (xcc)	FMOVd (xcc)	—	—	—	—	—
19_{16} – $1F_{16}$	—	—	—	—	—	—	—	—	—

[iii] iw<13> = 0

表 17-16 cond<3:0>

cond<3:0>	BPcc op = 0 op2 = 1	Bicc op = 0 op2 = 2	FBPfcc op = 0 op2 = 5	FBfcc op = 0 op2 = 6	Tcc op = 2 op3 = $3A_{16}$
0_{16}	BPN	BN^D	FBPN	FBN^D	TN
1_{16}	BPE	BE^D	FBPNE	$FBNE^D$	TE
2_{16}	BPLE	BLE^D	FBPLG	$FBLG^D$	TLE
3_{16}	BPL	BL^D	FBPUL	$FBUL^D$	TL
4_{16}	BPLEU	$BLEU^D$	FBPL	FBL^D	TLEU
5_{16}	BPCS	BCS^D	FBPUG	$FBUG^D$	TCS
6_{16}	BPNEG	$BNEG^D$	FBPG	FBG^D	TNEG
7_{16}	BPVS	BVS^D	FBPU	FBU^D	TVS
8_{16}	BPA	BA^D	FBPA	FBA^D	TA
9_{16}	BPNE	BNE^D	FBPE	FBE^D	TNE
A_{16}	BPG	BG^D	FBPUG	$FBUG^D$	TG
B_{16}	BPGE	BGE^D	FBPGE	$FBGE^D$	TGE
C_{16}	BPGU	BGU^D	FBPUGE	$FBUGE^D$	TGU
D_{16}	BPCC	BCC^D	FBPLE	$FBLE^D$	TCC
E_{16}	BPPOS	$BPOS^D$	FBPULE	$FBULE^D$	TPOS
F_{16}	BPVC	BVC^D	FBPO	FBO^D	TVC

表 17-17 rcond<2:0>

rcond<2:0>	BPr op = 0 op2 = 3 iw<28> = 0	MOVr op = 2 op2 = $2F_{16}$	FMOVr op = 2 op2 = 35_{16}
0	—	—	—
1	BRZ	MOVRZ	FMOVR{s|d|q}Z
2	BRLEZ	MOVRLEZ	FMOVR{s|d|q}LEZ
3	BRLZ	MOVRLZ	FMOVR{s|d|q}LZ
4	—	—	—
5	BRNZ	MOVRNZ	FMOVR{s|d|q}NZ
6	BRGZ	MOVRGZ	FMOVR{s|d|q}GZ
7	BRGEZ	MOVRGEZ	FMOVR{s|d|q}GEZ

表 17-18 cc, opf_cc (MOVcc, FMOVcc)

cc2	cc1	cc0	使われる条件コード
0	0	0	fcc0
0	0	1	fcc1
0	1	0	fcc2
0	1	1	fcc3
1	0	0	icc
1	0	1	—
1	1	0	xcc
1	1	1	—

表 17-19 cc フィールド (FBPfcc, FCMP*, FLCMP* 及び FCMPE*)

cc1	cc0	使われる条件コード
0	0	fcc0
0	1	fcc1
1	0	fcc2
1	1	fcc3

表 17-20 cc フィールド (BPcc, Tcc)

cc1	cc0	使われる条件コード
0	0	icc
0	1	—
1	0	xcc
1	1	—

表 17-21 IMPDEP1: VIS 命令 (op<1:0> = 2, op3 = 36_{16}) (1/3)

opf<3:0>	opf<8:4>							
	00_{16}	01_{16}	02_{16}	03_{16}	04_{16}	05_{16}	06_{16}	07_{16}
0_{16}	EDGE8	ARRAY8	FCMPLE16	—	—	FPADD16	FZERO	FAND
1_{16}	EDGE8N	—	—	FMUL8x16	—	FPADD16S	FZEROS	FANDS
2_{16}	EDGE8L	ARRAY16	FCMPNE16	—	FPADD64	FPADD32	FNOR	FXNOR
3_{16}	EDGE8LN	—	—	FMUL8x16AU	—	FPADD32S	FNORS	FXNORS
4_{16}	EDGE16	ARRAY32	FCMPLE32	—	—	FPSUB16	FANDNOT2	FSRC1
5_{16}	EDGE16N	—	—	FMUL8x16AL	—	FPSUB16S	FANDNOT2S	FSRC1S
6_{16}	EDGE16L	—	FCMPNE32	FMUL8sUx16	FPSUB64	FPSUB32	FNOT2	FORNOT2
7_{16}	EDGE16LN	LZD	—	FMUL8uLx16	—	FPSUB32S	FNOT2S	FORNOT2S
8_{16}	EDGE32	ALIGNADDRESS	FCMPGT16	FMULD8sUx16	FALIGNDATA	—	FANDNOT1	FSRC2
9_{16}	EDGE32N	BMASK	—	FMULD8uLx16	—	—	FANDNOT1S	FSRC2S
A_{16}	EDGE32L	ALIGNADDRESS_LITTLE	FCMPEQ16	FPACK32	—	—	FNOT1	FORNOT1
B_{16}	EDGE32LN	—	—	FPACK16	FPMERGE	—	FNOT1S	FORNOT1S
C_{16}	—	—	FCMPGT32	—	BSHUFFLE	—	FXOR	FOR
D_{16}	—	—	—	FPACKFIX	FEXPAND	—	FXORS	FORS
E_{16}	—	—	FCMPEQ32	PDIST	FPMUL64	—	FNAND	FONE
F_{16}	—	—	—	—	—	—	FNANDS	FONES

表 17-22 IMPDEP1: VIS 命令(op<1:0> = 2, op3 = 36_{16}) (2/3)

opf<3:0>	opf<8:4>							
	08_{16}	09_{16}	$0A_{16}$	$0B_{16}$	$0C_{16}$	$0D_{16}$	$0E_{16}$	$0F_{16}$
0_{16}		—	—	—	FPCMPLE16X FPCMPLEh	FPCMPLE8X FPCMPLEb	—	FMADD1ds
1_{16}	SIAM	—	—	—	FPCMPULE16X FPCMPULEh	FPCMPULE8X FPCMPULEb	—	FMSUB1ds
2_{16}		—	—	—	—	—	—	FNMSUB1ds
3_{16}	SLEEP	—	—	—	FPCMPUNE16X FPCMPUNEh	FPCMPUNE8X FPCMPUNEb	—	FNMADD1ds
4_{16}	—	—	—	—	FPCMPLE32X FPCMPLEw	FPCMPLE64X	FPMAX32X	FMADD2ds
5_{16}	—	FPSELMOV8X	—	—	FPCMPULE32X FPCMPULEw	FPCMPULE64X	FPMAXU32X	FMSUB2ds
6_{16}	—	FPSELMOV16X	—	—	—	—	FPMIN32X	FNMSUB2ds
7_{16}	—	FPSELMOV32X	—	—	FPCMPUNE32X FPCMPUNEw	FPCMPUNE64X	FPMINU32X	FNMADD2ds
8_{16}	—	—	—	—	FPCMPGT16X FPCMPGTh	FPCMPGT8X FPCMPGTb	—	—
9_{16}	PADD32	—	—	—	FPCMPUGT16X FPCMPUGTh	FPCMPUGT8X FPCMPUGTb	—	—
A_{16}	—	—	—	—	—	—	—	—
B_{16}	—	—	—	—	FPCMPUEQ16X FPCMPUEQh	FPCMPUEQ8X FPCMPUEQb	—	—
C_{16}	—	—	—	—	FPCMPGT32X FPCMPGTw	FPCMPGT64X	FPMAX64X	—
D_{16}	SSM SNF	—	—	—	FPCMPUGT32X FPCMPUGTw	FPCMPUGT64X	FPMAXU64X	—
E_{16}	—	—	—	—	—	—	FPMIN64X	—
F_{16}	—	—	—	—	FPCMPUEQ32X FPCMPUEQw	FPCMPUEQ64X	FPMINU64X	—

表 17-23 IMPDEP1: VIS 命令(op<1:0> = 2, op3 = 36_{16}) (3/3)

opf<3:0>	opf<8:4>									
	10_{16}	11_{16}	12_{16}	13_{16}	14_{16}	15_{16}	16_{16}	17_{16}	18_{16}	19_{16}-$1F_{16}$
0_{16}	FSEXTW	—	FPCMPULE8	—	—	—	FCMPEQd	FMAXd	FEPERMD	—
1_{16}	FZEXTW	—	—	—	—	FLCMPs	FCMPEQs	FMAXs	FECSLD	—
2_{16}	—	—	FPCMPUNE8	—	—	FLCMPd	FCMPEQEd	FMINd	FESUMMD	—
3_{16}	—	—	—	—	—	—	FCMPEQEs	FMINs	FECPD	—
4_{16}	FPCMP64x	—	—	—	—	—	FCMPLEEd	FRCPAd	—	—
5_{16}	FPCMPU64x	—	—	—	—	—	FCMPLEEs	FRCPAs	—	—
6_{16}	FPSLL64x	—	—	—	—	—	FCMPLTEd	FRSQRTAd	—	—
7_{16}	FPSRL64x	—	—	—	—	—	FCMPLTEs	FRSQRTAs	—	—
8_{16}	—	—	FPCMPUGT8	—	—	—	FCMPNEd	FTRISSELd	—	—
9_{16}	—	—	—	—	—	—	FCMPNEs	—	—	—
A_{16}	—	—	FPCMPUEQ8	—	—	—	FCMPNEEd	FTRISMULd	—	—
B_{16}	—	—	—	—	—	—	FCMPNEEs	—	—	—
C_{16}	—	—	—	—	—	—	FCMPGTEd	FEXPAd	—	—
D_{16}	—	—	—	—	—	—	FCMPGTEs	—	—	—
E_{16}	—	—	—	—	—	—	FCMPGEEd	FRDd	—	—
F_{16}	FPSRA64x	—	—	—	—	—	FCMPGEEs	FRDs	—	—

表 17-24 IMPDEP2: (op<1:0> = 2, op3 = 37_{16})

size	var			
	0	1	2	3
0_{16}	FPMADDX	FPMADDXHI	FTRIMADDd	FSELMOVd
1_{16}	FMADDs	FMSUBs	FNMSUBs	FNMADDs
2_{16}	FMADDd	FMSUBd	FNMSUBd	FNMADDd
3_{16}	—	—	FSHIFTORX	FSELMOVs