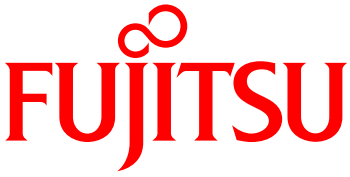


SPARC64™ VIIIfx Extensions

Fujitsu Limited

Ver 15, 26 Apr. 2010

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

変更履歴

日付	内容	ページ
2009/2/13	第 11 版発行。	
2009/02/25	準公開 PA イベント *single_sxar_commit* を追加した。 TABLE Q-1 と説明イベント名の大文字小文字を統一した。	314
2009/03/16	ASI_QUAD_LDD_PHYS{_L} でノンキャッシャブル空間にアクセスすると *data_access_exception* が通知されるという記述を削除した。	88
2009/03/27	RED_state 遷移時の動作仕様が実装と合っていなかったので、実装にあわせた。 • DCUCR, MCNTL, ERROR_CONTROL は RED_state 以外からに RED_state に遷移したときのみ更新される。	248
2009/03/27	error_state を経て WDR 発生時のレジスタ状態の変化について注釈をつけた。	249
2009/03/31	FTRIMADDd の係数テーブルを公開した。	125
2009/04/07	STICK_CNTL.stop = 1 でも STICK_CMPR = STICK に設定すればタイマ割り込みが起きることを追記した。	323
2009/4/8	第 12 版発行。	
2009/04/13	FTRIMADDd の *unfinished_FPop* 検出条件の 2 を削除した。	141
2009/04/24	ASI_UGESR.IUG_TSTATE=1 になる要因に TXAR を追加した。	276
2009/05/28	ASI_CACHE_INV の説明を、キャッシュラインの内容をメモリに書き込んでから無効化する、とした。	233
2009/07/07	2009/03/27 の修正 (RED_state 遷移時の仕様を実装にあわせる) は Errata に書くことにし、仕様書の記述は元に戻した。	248
2009/07/10	誤訳を修正した。RED_state だと常にノンキャッシャブルであるかのような記述になっていたので、MMU 無効時はノンキャッシャブルと直した。	43
2009/07/13	MOVr, MOVcc, FMOVr, FMOVcc 命令で条件不成立のとき、XASR のビットがセットされるかもしれないと注釈をつけた。	32

日付	内容	ページ
2009/07/13	第 13 版発行。	
2009/08/25	FMIN, FMAX の仕様を修正した。入力が SNaN なら NaN 伝播し、入力が QNaN または SNaN なら NV を検出する。	116
2009/09/08	第 14 版発行。	
2009/11/06	SXAR1 の s_ フィールドが 0 でない場合の Compatibility Note を追加した。	131
2009/11/06	FMADD の例外条件の説明の誤記を訂正した。cexc を aexc に訂正した。	72
2010/01/20	SIMD ロードの仕様誤記訂正。extend 側の *data_access_error* で basic の rd は更新されることになっていたが、basic, extend どちらで *data_access_error* しても basic, extend どちらの rd も更新されない。	81, 85, 147, 180
2010/04/13	XFILL は L1, L2 に UE があっても *data_access_error* を通知しないことを明記した。	133
2010/04/21	キャッシュサイズを 6M/12way にした。	12, 231, 328
2009/04/26	第 15 版発行。	

Contents

F.CHAPTER **1**

Overview

1.1 Navigating *the SPARC64™ VIIIfx Extensions*

SPARC64 VIIIfx は、JPS1 に準拠したアーキテクチャのプロセッサである。JPS1 では、仕様を共通仕様と実装依存仕様に分けて記述するようになっており、本仕様書は実装依存仕様を定義する。本仕様書では原則として、共通仕様で定義済の事項については再掲しない。

本書の章・節番号と見出し名は、基本的に JPS1 **Commonality** に合わせてあり、英語のままにしてある。各章・節では JPS1 **Commonality** の実装依存仕様や未定義仕様、あるいは SPARC64 VIIIfx で変更された仕様について説明している。JPS1 **Commonality** にない章・節には日本語の見出しがついており、SPARC64 VIIIfx 独自に追加された仕様を説明している。本書は JPS1 **Commonality** の定義を前提としているので、必要に応じて “SPARC Joint Programming Specification 1 (JPS1): Commonality” (JPS1 **Commonality**) を参照されたい。

1.2 Fonts and Notational Conventions

本仕様書の表記法は JPS1 **Commonality** に準ずる。

Reserved フィールドの扱い

命令語およびレジスタ中の使用されていないビットは、将来のために予約されている。そのようなフィールドを reserved フィールドと呼び、reserved または — と表記している。

JPS1 **Commonality** では Chapter 2 で、命令語およびレジスタの reserved フィールドの扱いを以下のように定義している。

- 命令語の reserved フィールドは 0 であるべき。0 以外の値が入っていた場合の動作は未定義 (Chapter 2)。
- レジスタの reserved フィールドにソフトウェアが書く値は、そのフィールドの読み出しで得られた値か 0。ハードウェアは 0 を出力すべき。

なお、命令語の reserved に関しては、JPS1 **Commonality** の Section 6.3.9 と Appendix I.2 にも記述がある。

SPARC64 VIIIfx では、reserved を以下のように扱う。

- 命令語の reserved フィールドは、仕様に動作と値が明記されているものはその通り。reserved フィールドが 0 以外の値のときの動作が明記されていない命令は、そのフィールドを無視して実行する。
- レジスタの reserved フィールドは、仕様に動作と値が明記されているものはその通り。動作と値が明記されていないフィールドは、書き込みは無視され、読み出しには不定値が返る。書き込んだ際の副作用が書かれていないものの動作は未定義である。

レジスタフィールドの読み書き属性

この論理仕様書では、レジスタのフィールドの読み書き属性を以下のように記述する。

TABLE 1-1 レジスタフィールドの読み書き属性

Type 属性	Description 意味
	読み出しには不定値が返され、書き込みは無視される。 値が明記されていない reserved はこれに相当する。
R	読み出しにはフィールドの値が返され、書き込みは無視される。
W	読み出しには不定値が返され、書き込みにはその値が書き込まれる。
RW	読み出しにはフィールドの値が返され、書き込みにはその値が書き込まれる。
RW1C	読み出しにはフィールドの値が返され、1 を書き込むとフィールドに 0 が書かれる。

F.CHAPTER **2**

Definitions

この章では SPARC64 VIIIfx に固有の概念・用語を定義する。JPS1 に共通の用語の定義は、JPS1 **Commonality** の Chapter 2 を参照。

basic floating-point register HPC-ACE で拡張されたレジスタのうち、SIMD の basic 側演算で使われる `f[0]` – `f[254]` レジスタ。

(SIMD の)basic 側演算 SIMD 動作時の 2 演算のうちのひとつ。命令で指定したレジスタ番号を使う。

extended floating-point register HPC-ACE で拡張されたレジスタのうち、SIMD の extended 側演算で使われる `f[256]` – `f[512]` レジスタ。

(SIMD の)extended 側演算 SIMD 動作時の 2 演算のうちのひとつ。命令で指定したレジスタ番号 +256 を使う。

HPC-ACE High Performance Computing - Arithmetic Computational Extensions の略で、SPARC64 VIIIfx 独自に拡張された論理仕様の総称。レジスタ本数の増加、HPC 用途の命令、浮動小数点演算の SIMD 拡張などを含む。

mTLB メイン TLB。命令用 (I) とデータアクセス用 (D) に分かれており、それぞれ ***mITLB***, ***mDTLB*** と呼ばれることもある。***uITLB***, ***uDTLB*** に載せるアドレス変換情報を保持している。***uITLB***, ***uDTLB*** がアドレス変換情報を持っていないとき、***mTLB*** が検索される。***mTLB*** にアドレス変換情報があれば、対応する ***uTLB*** に送られる。***mTLB*** にアドレス変換情報がないときは、例外が発生しソフトウェアにトラップが通知される。ソフトウェアによりアドレス変換情報を ***mTLB*** に載せた後、ハードウェアが命令を再実行する。

syncing 命令 ***マシン sync*** を起こす命令。syncing 命令の***発行***は、プログラム順で syncing 命令よりも前の命令がすべて***完了***してから行われ、syncing 命令が***完了***してから後続命令が***発行***される。つまり syncing 命令は単独で***発行***、***実行***、***完了***される命令である。

uTLB マイクロ TLB。命令用 (I) とデータアクセス用 (D) に分かれており、それぞれ ***uITLB***, ***uDTLB*** と呼ばれることもある。ハードウェアは ***uTLB*** に載っているアドレス変換情報にもとづいてアドレス変換を行う。***uTLB*** にアドレス変換情報がなければ、***mTLB*** から補充される。

XAR 対象命令 XAR レジスタで指定される、レジスタフィールドの上位ビット情報を結合して実行される命令。

アウトオブオーダ実行 命令の***実行***が，プログラム順序を追い越して行われるマイクロアーキテクチャ。入力ソースデータが揃っていない命令を，後続の入力ソースデータが揃った命令が追い抜いて***実行***する。

演算器 (functional unit) 演算命令を処理する資源。

(資源の) 解放 命令***実行***のために割り当てられた資源が、次の命令により使用可能になること。

コア コアとは、実行パイプラインと命令実行に関連する資源 (演算器や L1 キャッシュなど) を含んだ実体。一つまたはそれ以上の***スレッド***を持つことがあるが、SPARC64 VIIIfx では 1 コアは 1 スレッドからなる。

コミット (完了) ある命令について、その命令より前のすべての命令***実行***が正常に***完了***した後、当該命令の実行結果を確定すること。ある命令がコミットすると、その命令の実行結果はソフトウェアから見える資源に反映される。それ以前の状態は失われる。

コンプリート (終了) ある命令の***実行が終了***し、正常終了したことが命令発行ユニットに通知されること。命令の終了により実行結果が一時的に反映されるが、***完了***するまではまだ永続的な状態にはなっておらず、元の状態に復旧することが可能である。

実行 命令が実行ユニットに送られ、演算が行われること。命令が演算器にある間は実行中である。

実行終了 演算器での命令の実行が終了し、結果が出力バスに出力されること。出力バスの結果はレジスタファイルや他の演算器へと送られる。

サイクルアカウンティング 性能阻害要因を分析する手法。

サスペンド スレッドが命令実行を一時中止している状態。サスペンド中は命令は実行されないが、キャッシュ上のデータに矛盾が起きないようメンテナンスされている。***スリープ***と異なり、サスペンド中のスレッドを命令実行状態に復帰させるには、割り込みやタイマによるトラップ発生が必要である。

スーパースカラ 1 サイクルに複数の命令を***発行***、***実行***、***完了***する CPU の機構。

スキャン CPU チップ内のラッチやレジスタを読み書きする方法。スキャン可能なように設計されたラッチやレジスタはスキャンリングで読み出し・書き込みができる。

ストール ***命令発行***ができないこと。資源の空き状態やプログラム上の制約から、必ずしも毎サイクル命令が***発行***できるとは限らない。

ストロング プリフェッチ プロセッサ内部の資源が不足していても捨てられずに実行されることが保証されたデータプリフェッチ命令。

スレッド ソフトウェア命令列を実行するハードウェアの単位。ソフトウェアから見える資源 (PC、レジスタなど) や、直接見えないが命令実行に必要なマイクロアーキテクチャを含んでいる。

投機的命令実行 命令実行が投機的であるとは、条件分岐の分岐方向が未確定な時点で、または割り込みやトラップなどの発生有無が未確定な時点で、後続の命令を***実行***すること。また、投機的な実行により得られた結果を使い、さらに後続の命令を***実行***すること。

プロセッサモジュール 情報処理用の独立した実体で、外部との接続バスを共有するひとつまたはそれ以上の***コア***が入った部品。

命令ディスパッチ ある命令の実行に必要な資源が全部使用可能になり、演算器に送られること。

命令発行 命令が***リザベーションステーション***に送られること。

命令フェッチ 命令が命令キャッシュまたは***コア***内部のバッファから読み出され、命令発行ユニットに送られること。

メモリマネジメントユニット (MMU) ***コア***内にある、64 ビットの論理アドレスを物理アドレスに変換するハードウェア機構。MMU には ***mITLB***, ***mDTLB***, ***uITLB***, ***uDTLB*** およびアドレス変換を制御するための ASI レジスタがある。

リザベーション ステーション ***発行***された命令が、実行パイプラインに投入されるまで格納されるキュー (バッファ)。演算器が使用可能であった場合に、入力データが揃った命令は、演算器が使用可能になると、リザベーションステーションから取り出され演算器に投入される。***アウトオブオーダ実行***の制御はこのキューで行う。

リネーミングレジスタ 命令の実行結果を、コミットしてレジスタに格納されるまで一時的に保持するバッファ。ユーザが直接操作することはできない。

F.CHAPTER **3**

Architectural Overview

この章では、SPARC64 VIIIfx プロセッサの概要について説明する。この章は JPS1 **Commonality** の章立てには沿っていない。

3.1 SPARC64 VIIIfx プロセッサ

SPARC64 VIIIfx プロセッサは、1 つの CPU チップに 8 つのコアと L2 キャッシュ、さらにメモリコントローラ (MAC) を統合した、マルチコア・プロセッサである。アーキテクチャは SPARC V9 に準拠しながら独自の拡張を行い、サーバ用途の高性能・高信頼性に加え、HPC 分野でも高性能を追求したプロセッサである。

高性能を実現するマイクロアーキテクチャ

SPARC64 VIIIfx は、1 サイクルに最大 4 命令まで発行するアウトオブオーダ実行のスーパースカラプロセッサである。アウトオブオーダの処理では、命令フェッチ部は実行パスを予測しながら命令をフェッチし、その命令列をその順番に従い発行する。発行された命令は、実行可能になるまで一時的にリザベーションステーションに格納される。命令が実行可能になると、元の命令列の順序とは関係なく実行ユニットへ送られ、実行される。実行が終了 (complete) した命令は、元の命令列の順序に従い完了 (commit) 処理される。つまり、ある命令の完了処理は、その命令に先行する命令がすべて完了してから行われる。完了処理が終わると命令の実行結果がレジスタ・メモリ等に反映されプログラムから見えるようになる。アウトオブオーダ実行は SPARC64 VIIIfx の高性能に大きく貢献している。

SPARC64 VIIIfx は、分岐命令の実行方向を予測するためにブランチヒストリバッファを実装している。ブランチヒストリバッファは、DBMS のような大規模アプリケーションで高い予測的中率を維持するよう、また SPARC64 VIIIfx の高度な命令フェッチ機構をサポートするよう、十分な大きさを備えている。この命令フェッチ機

構は、分岐履歴に従い、複数の条件分岐を含む命令列の実行方向を予測し、命令をフェッチする。これにより、命令キャッシュミスによって生じる性能ペナルティの影響を可能な限り小さくできる。

SPARC64 VIIIfx はまた、HPC (High Performance Computing) 用途にふさわしい機能を備えている。それはたとえば、SPARC V9 の命令仕様を独自に拡張した HPC-ACE 拡張や、チップ内コア間で高速な同期を実現するハードウェアバリア機能などである。HPC-ACE 拡張では整数レジスタを 192 本、浮動小数点レジスタを 256 本に拡張し、浮動小数点演算命令を中心に 7 つの新規命令を追加したほか、2 並列の SIMD(Single Instruction Multiple Data) 演算も可能にした。SIMD の採用により、1 サイクルで最大 8 つの浮動小数点演算を実行することができ、HPC 分野の高性能を実現している。

チップ内に集約された高機能

SPARC64 VIIIfx は、CPU チップ上に L2 キャッシュを持っている。L2 キャッシュは、命令、データ共用で、8 コアに共有されている。SPARC64 VIIIfx では L2 キャッシュが最外層のキャッシュである。L2 キャッシュを CPU チップに統合したことで、SPARC64 VIIIfx の性能と信頼性は向上し、アクセス時間が短くウェイ数の多いキャッシュを実装することができている。また、L2 キャッシュに対する外部接続が不要なため、高信頼性にも貢献している。

さらに SPARC64 VIIIfx は、メモリコントローラも CPU チップに内蔵している。DIMM は CPU に直結され、メモリアクセスのレイテンシは劇的に短縮された。

HPC の高性能を約束する重要な機能の一つがハードウェアバリアである。マルチスレッドのジョブを高速に処理するためには、スレッド間の同期にかかる時間を可能な限り減らすことが重要である。この問題に対し SPARC64 VIIIfx は、ハードウェアによるバリア同期機構という解決策を提供する。SPARC64 VIIIfx のハードウェアバリアは、複数コアでのバリア同期用に加え、全コアによるマスター / ワーカーモデルを可能にする、post/wait 型のプリミティブも用意している。

高信頼性

SPARC64 VIIIfx は以下の高度な RAS 機能を実装している。

1. キャッシュの RAS 機能

- 強固なキャッシュエラー保護 :
  - D1(データレベル 1) キャッシュデータ、U2(ユニファイドレベル 2) キャッシュデータ、U2 キャッシュタグに対する ECC 保護。
  - I1(命令レベル 1) キャッシュデータに対するパリティ保護。
  - I1 キャッシュタグと D1 キャッシュタグに対するパリティ保護と二重化。
- すべての種類の 1 ビットエラーの自動訂正 :
  - ECC 保護されたデータに対する自動 1 ビット エラー訂正。
  - I1 キャッシュデータパリティエラーに対する I1 キャッシュデータの無効化と再読み込み。

- I1 キャッシュタグおよびD1 キャッシュタグのパリティエラーに対する二重化されたタグからのコピー。

- キャッシュの一貫性を維持したまま行われる動的なウェイ縮退。
- キャッシュデータの訂正不能エラーのマーキング :
  - 最初に訂正不能エラーを発見したモジュールが、特殊なパターンでマークする。
  - 一つの故障で何度もエラーが報告されないよう、エラーマークで故障モジュールを識別して隔離する。

2. 命令実行ユニットの RAS 機能

- 強固なエラー保護 :
  - 全てのデータパスのパリティ保護。
  - ほとんどのソフトウェア可視レジスタ、内部、一時レジスタに対するパリティ保護。
  - 演算結果のパリティまたは剰余チェック。
- ハードウェアによる命令再実行。
- (ハードウェア命令再実行の失敗の後の) ソフトウェア命令再実行のサポート。
- ソフトウェアリカバリのための エラー分離 :
  - どのレジスタがエラーしたか (被疑レジスタ) の明示。
  - エラーした命令が再実行可能かどうかを示す。
  - エラーの度合いによりトラップを使い分ける。

3. 充実したソフトウェアインターフェイス

- プログラム実行への影響の重大度によるエラー分類 :
  - 重大なエラー (マスク不可): OS の介入無しでの実行続行は不可能なエラー。例外で報告される。
  - 軽微なエラー(マスク可): エラーが例外で報告されるかどうかを OS が制御する。エラーは、プログラム実行には直接作用しない。
- ソフトウェアへの影響を判断するための、識別したエラーの表示。
- 付加的なエラーに対する非同期データエラー (ADE) 例外
  - 例外通知により命令実行が中断されるが、その状態は発見されたエラーによって異なるので、完了方法を表示する。
  - 遅延報告 (deferred) だがリトライ可能な ADE 例外。
  - エラー分離と命令再実行が正しく行えるよう、同時に起きた複数のエラーをすべて表示する。

3.1.1 コア内部のコンポーネント

FIGURE 3-1 は SPARC64 VIIIfx のブロック図である。SPARC64 VIIIfx は 8 つのコアとメモリコントローラ、バスインターフェースをチップ内に統合している。コア内には以下のコンポーネントがある。

- 命令制御ユニット (IU)
- 実行ユニット (EU)

- 記憶ユニット (SU)
- L2 キャッシュと外部アクセスユニット (SXU)

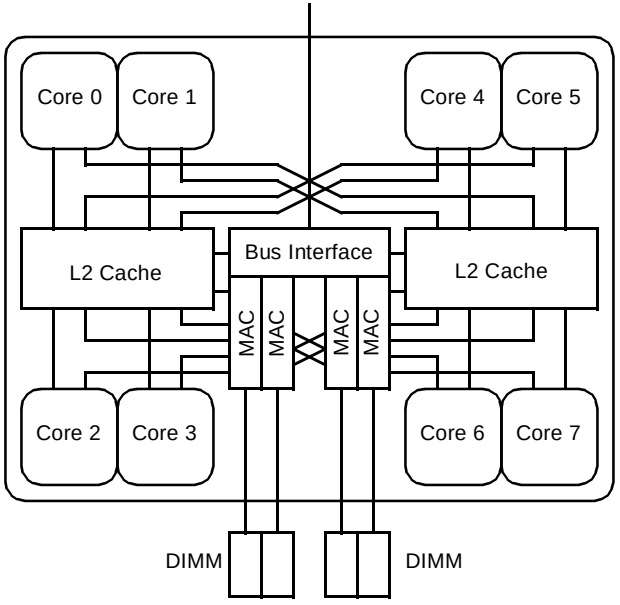


FIGURE 3-1 SPARC64 VIIIfx ブロック図

3.1.2 命令制御ユニット (IU)

IU (Instruction control Unit) は、命令実行パスを予測し、予測したパスの命令をフェッチする、その後、フェッチした命令を適切なリザベーションステーションに配送し、そして、実行パイプラインに命令を発送する。発送された命令は、アウトオブオーダで実行され、インオーダで完了される。主要なブロックを TABLE 3-1 に示す。

TABLE 3-1 命令制御ユニットの主要ブロック

ブロック	説明
命令フェッチパイプライン	5 ステージ : フェッチアドレス生成、iTLB と L1I キャッシュアクセス、iTLB と L1I キャッシュのタグ比較、命令バッファへの書き込み、結果の格納。
ブランチヒストリ	分岐先と分岐方向を予測するためのテーブル。
命令バッファ	フェッチした命令を保持するためのバッファ。

TABLE 3-1 命令制御ユニットの主要ブロック

ブロック	説明
リザベーションステーション	命令が実行可能になるまで保持するための 5 つのリザベーションステーション : 分岐とその他の制御転送命令用の RSBR、load/store 命令用の RSA、整数演算命令用の RSE、浮動小数点演算命令用の RSFA と RSFB。
コミットスタックエントリ (CSE)	実行中の (発行されてまだ完了していない) 命令の情報を保持するためのバッファ。
PC, nPC, CCR, FSR	命令実行制御のためのプログラム可視レジスタ。

3.1.3 命令実行ユニット (EU)

EU (Execution Unit) は全ての整数 (算術、論理、シフト) 命令と全ての浮動小数点及び VIS 命令の実行を行う。TABLE 3-2 に EU の主要なブロックを記述する。

TABLE 3-2 EU の主要ブロック

ブロック	説明
GUB	汎用整数レジスタ (gr) 用のリネーミングレジスタファイル。
GPR	汎用整数レジスタファイル。
FUB	浮動小数点レジスタ (fr) 用のリネーミングレジスタファイル。
FPR	浮動小数点レジスタファイル。
EU 制御論理	命令実行ステージを制御 : 実行する命令の選択、レジスタリード、命令実行。
インターフェースレジスタ	他ユニットへの入出力レジスタ。
2 つの整数実行パイプライン (EXA, EXB)	64 ビット ALU とシフタ。
2 つの浮動小数点実行パイプライン (FLA, FLB)	各浮動小数点実行パイプラインは、浮動小数点 (乗算、加算 / 減算、乗加算、除算 / 平方根、グラフィックス) 命令。
メモリアクセスパイプラインのための 2 つの仮想アドレス加算器 (EAGA, EAGB)	ロード / ストアのための 2 つの 64 ビット仮想アドレス生成。

3.1.4 ストレージ制御ユニット (SU)

SU (Storage Unit) は、ロード及びストア命令のために全てのデータの供給と受給を扱う。TABLE 3-3 に、SU の主要なブロックを記述する。

TABLE 3-3 SU の主要ブロック

ブロック	説明
命令レベル 1 キャッシュ	32KB, 2 ウェイセットアソシアティブ、1 ラインは 128 バイト。低レイテンシ、命令供給用。
データレベル 1 キャッシュ	32KB, 2 ウェイセットアソシアティブ、1 ラインは 128 バイト。低レイテンシ、ロード • ストアデータ供給用。
命令用 TLB	256 エントリ、2 ウェイセットアソシエイティブ TLB (sITLB)。 16 エントリ、フルアソシエイティブ TLB (fITLB)。
データ用 TLB	512 エントリ、2 ウェイセットアソシエイティブ TLB (sDTLB)。 16 エントリ、フルアソシエイティブ TLB (fDTLB)。
ストアバッファとライトバッファ	ストア操作のレイテンシからパイプラインを分離する。ストアがデータを待っている間、パイプラインが流れ続ける事を可能にする。最終的には、データレベル 1 キャッシュに書き込む。

3.1.5 二次キャッシュユニット (SXU)

SXU (Secondary Cache and External Access Unit) は、ユニファイドレベル 2 キャッシュの操作と外部データクセスインタフェースを制御する。TABLE 3-4 に、SXU の主要なブロックを記述する。

TABLE 3-4 SXU の主要ブロック

ブロック	説明
ユニファイドレベル 2 キャッシュ	サイズは 6M バイトであり、12 ウェイセットアソシエイティブ構成、 1 ラインは 128 バイト。ライトバック方式のキャッシュ。
ムーブインバッファ	キャッシュライン読み出しリクエストに応じてメモリシステムから返ってきたデータをキャッシュする。
ムーブアウトバッファ	メモリへのライトバックデータを保持する。

3.2 プロセッサパイプライン

SPARC64 VIIIfx のパイプラインは 16 ステージからなる。FIGURE 3-2 にパイプラインの各ステージを、FIGURE 3-3 にパイプライン図を示す。

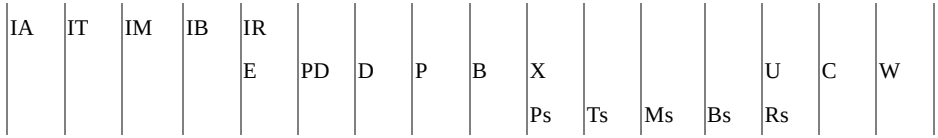


FIGURE 3-2 SPARC64 VIIIfx のパイプラインステージ

3.2.1 命令フェッチステージ

- IA: 命令フェッチアドレス生成
- IT: iTLB, 命令キャッシュタグアクセス
- IM: 命令キャッシュタグ比較
- IB: 命令キャッシュからバッファへ読み込み
- IR: 命令フェッチ最終ステージ

IA から IR までのステージでは、キャッシュアクセスユニット (SU) と連携して命令を読み込み、パイプラインの後続ステージに供給する。メモリまたはキャッシュから読み出された命令は、命令バッファ (Instruction Buffer) に蓄えられる。

SPARC64 VIIIfx は分岐予測用の BRHIS (BRanch HIStory) と RAS (Return Address Stack) 資源を備えている。命令フェッチアドレスステージでは、分岐予測を行いフェッチアドレスを決定する。

SPARC64 VIIIfx の命令フェッチステージは、実行ステージがストールしていても命令フェッチができるよう、後続ステージとは可能な限り独立したデザインになっている。命令フェッチ部は、命令バッファがフルになるまでフェッチを続け、その後もプリフェッチを出して命令列を L1 キャッシュに持ってくることが可能である。

3.2.2 命令発行ステージ

- E: エントリ
- PD: プリデコード
- D: デコード

SPARC64 VIIIfx はアウトオブオーダ実行のプロセッサである。SPARC64 VIIIfx には 6 個の演算器があり (整数用 2 つ、浮動小数点用 2 つ、メモリアクセス用 2 つ)、整数演算器とメモリアクセス演算器には演算器 2 つに対して 1 個の、浮動小数点演算器に

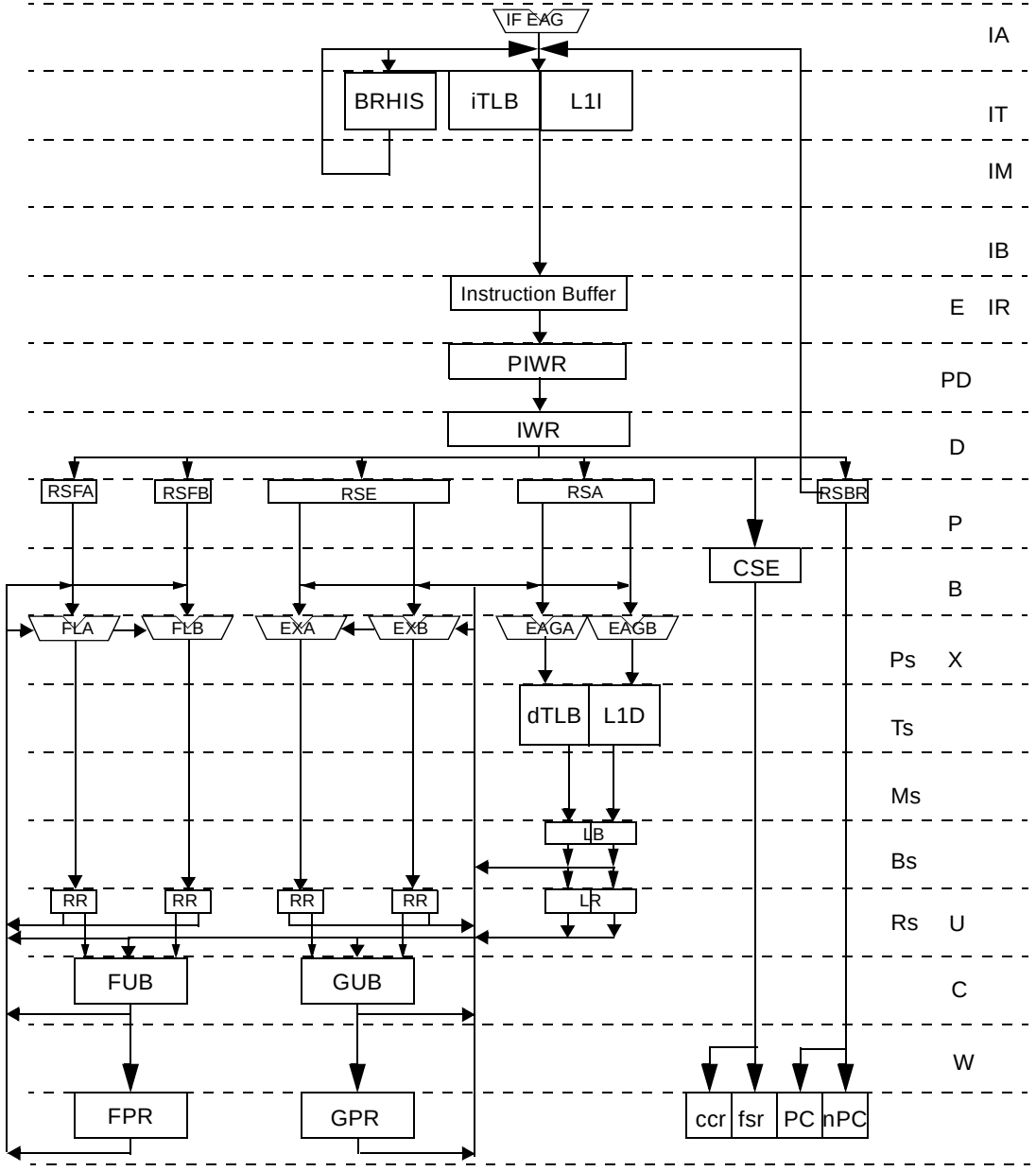


FIGURE 3-3 SPARC64 VIIIfx のパイプライン図

は独立した 2 個のリザベーション・ステーションがある。E, PD, D ステージでは命令をデコード・発行し、リザベーション・ステーションに登録する。SPARC64 VIIIfx は 1 サイクルで最大 4 命令まで発行できる。

命令実行に必要な以下の資源は、このステージで割り当てられる。

- コミットスタックエントリ (CSE)
- 整数用と浮動小数点用のリネーミングレジスタ (GUB, FUB)
- リザベーション・ステーションのエントリ
- メモリアクセス用のポート

命令実行に必要な資源は命令ごとに異なるが、どの命令でも、実行に必要なすべての資源の割り当てはこのステージで行われる。通常の実行では、リザベーション・ステーションの解放は X ステージで、その他の資源の解放はパイプラインの最後の W ステージで行われる。E ステージから W ステージまでにある命令が実行中 (in-flight) の命令となる。例外が通知されると、実行中の命令は中止され、それらの命令が使用していた資源は直ちに解放される。これにより、デコーダは新しい命令の発行がすぐに行える。

3.2.3 実行ステージ

- P: プライオリティ
- B: バッファ読み出し
- X: 実行
- U: 更新

リザベーション・ステーションに格納された命令は、各々の実行条件が満たされると演算器に送られる。実行条件とは、入力データが確定する、演算器が空く、などである。実行にかかるサイクル数は命令によって異なる。

キャッシュアクセスの実行ステージ

メモリアクセス要求は、アドレス計算が完了すると、キャッシュアクセスユニットに送られる。キャッシュアクセスのステージは、分岐予測を除けば命令フェッチの各ステージと同じである。詳細は Section 3.2.1 を参照。命令フェッチのステージ名とキャッシュアクセスのステージ名の対応を以下に示す。

命令フェッチ	データアクセス
IA	Ps
IT	Ts
IM	Ms
IB	Bs
IR	Rs

例外が通知されると、パイプラインにデータを送るための資源は解放されるが、キャッシュアクセスのパイプライン自体は、システムに要求したメモリアクセスを完了させるまでは動き続ける。データが返ってくると、キャッシュにストアされる。

3.2.4 命令完了ステージ

- W: 書き込み

アウトオブオーダで実行された命令は、最後に命令順序通りに完了処理が行われる。例外の処理はこのステージで行われる。実行ステージで起きた例外は、すぐに処理されるのではなく、先行するすべての命令の完了処理が終わった後で通知される[1]。

1. RAS 関連の例外は完了を待たずに報告される。

F.CHAPTER **4**

Data Formats

JPS1 **Commonality** の Chapter 4 を参照。

F.CHAPTER 5

Registers

JPS1 **Commonality** の Chapter 5 では、汎用レジスタ、ASR レジスタ、ASI レジスタの 3 種類を定義している。章立ては大きく、非特権レジスタ、特権レジスタの順番になっており、ASR, ASI レジスタは特権レジスタとして扱われているが、実際は非特権アクセス可能なものも混ざっているので適切さを欠いている。さらに、ASI レジスタについては、Chapter 5 の他、用途に応じていくつかの Appendix でも定義されており、この点でもまとまりを欠いている。

SPARC64™ VIIIfx Extensions の章立てはできる限り JPS1 **Commonality** に合わせることにしているので、JPS1 **Commonality** の Chapter 5 で定義されているレジスタに関する実装依存仕様はこの章に記述することにする。ASI, ASR レジスタは特権・非特権が混ざるが、便宜上 Section 5.2, *“Privileged Registers”* に配置した。

ASI レジスタについては、以下のセクションも参照されたい。

- Appendix F.10, *“Internal Registers and ASI Operations”*
- Appendix L.3.2, *“Special Memory Access ASIs”*
- Appendix L.4, *“ ハードウェアバリア ”*
- Appendix M.3, *“ キャッシュ制御 ASI”*
- Appendix N.4, *“Interrupt ASI Registers”*
- Appendix P.2.5, *“ASI_EIDR”*
- Appendix P.2.6, *“ エラー検出の制御 (ASI_ERROR_CONTROL)”*
- Appendix P.3.1, *“ASI_STCHG_ERROR_INFO”*
- Appendix P.4.1, *“ 緊急エラーステータス (ASI_UGESR)”*
- Appendix P.7.1, *“ASI_ASYNC_FAULT_STATUS (ASI_AFSR)”*
- Appendix R.1, *“System Config Register”*
- Appendix R.2, *“STICK Control Register”*

また、電源オン時やリセット時のレジスタ値については Appendix O.3, *“ リセット、RED_state 後のプロセッサ状態 ”* (page 249) に、レジスタでエラーが発生した際の通知方法やエラー修復方法については Appendix P.8, *“ レジスタで起きたエラーの処理方法 ”* (page 286) にまとめられている。

5.1 Nonprivileged Registers

5.1.1 General-Purpose r Registers

r[32] – r[63] (xg[0] – xg[31]) を追加する。

レジスタ番号は既存命令フィールドには入りきらないので、上位 1 ビットを XAR.urs1, XAR.urs2, XAR.urs3, XAR.urd で指示する。拡張されるレジスタ数は 32 本なので、各フィールドの bit<2:1> は 0 になっていなければならない。bit<2:1> が 0 でないときは、*illegal_action* 例外が通知される。

拡張整数レジスタは大部分の命令で使用可能。使用できない命令の実行時に XAR.v = 1 になっていると、*illegal_action* 例外が通知される。

xg[0] – xg[31] は PSTATE.AG, PSTATE.MG, PSTATE.IG の値によらず同じものが見える。

拡張レジスタに対する書込みを行うと、XASR.xgd = 1 になる。

Programming Note – Context switch 時に save する必要があるかどうかが判断できる。

5.1.4 Floating-Point Registers

レジスタを追加し、全部で 256 本の倍精度レジスタが使えるようにする。追加するのは倍精度レジスタで、f[64] – f[510] (偶数番号のみ) で指定する。レジスタの追加に伴い、レジスタの状態を表わす XASR も追加される。詳細は *"Extended Arithmetic Register Status Register (XASR) (ASR 30)"* (page 32) 参照。

f[0] – f[254] を Basic Floating-Point Registers、f[256] – f[510] を Extended Floating-Point Registers と呼ぶことにする。また f[0] – f[62] を V9 Floating-Point Registers とも呼ぶ。

Floating-Point Register Number Encoding

JPS1 **Commonality** の同名の節で、命令で倍精度レジスタを指示する際のエンコーディングが定義されている。

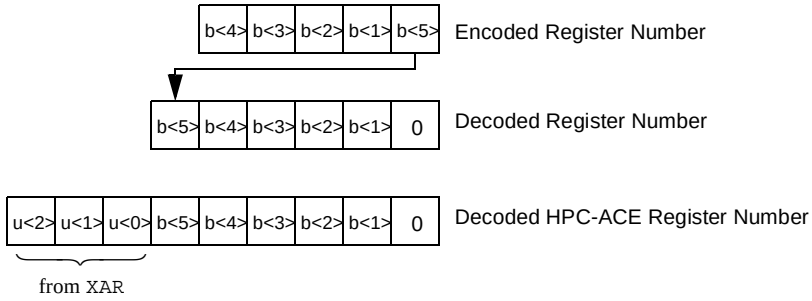


FIGURE 5-1 倍精度浮動小数点レジスタ番号のエンコーディング

拡張された 256 本の倍精度レジスタを指示するには、命令語のレジスタフィールド 5 ビットでは足りない。このため HPC-ACE では、レジスタフィールドの上位ビットを XAR レジスタに置き、命令実行時にそれらを結合してレジスタ番号を生成することにした。したがって、命令語だけではレジスタ番号は特定できない。

HPC-ACE 拡張レジスタ番号の生成は、XAR に置かれた上位 3 ビットと、FIGURE 5-1 に示したデコード後の 6 ビットを結合し 9 ビットの番号とする。最下位ビットは常に 0 になるので、`f[0]` – `f[510]` の偶数番号のみ、256 本のレジスタが指示できる。

倍精度レジスタの単精度利用

レジスタの追加に合わせて、倍精度レジスタを単精度浮動小数点演算でも使えるようにする。SPARC64 VIIIfx で追加されたレジスタだけでなく、SPARC V9 で定義された倍精度レジスタも単精度浮動小数点演算に使える。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する際は、以下の点が SPARC V9 仕様と異なる。

- 命令のレジスタフィールドの解釈は、JPS1 **Commonality** の TABLE 5-5 の倍精度と同じになる。したがって偶数番号レジスタのみが使用できる。`f[2n]` (n = 0–255)
- 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。

Programming Note – XAR.v = 1 かつ XAR.urs1 = 0 の場合、rs1 では SPARC V9 の倍精度レジスタを使って単精度演算を行う。rs2, rs3, rd も同様である。このとき bit<31:0> すなわち奇数番号レジスタの内容は破壊されることに注意すること。

エンディアン変換は単精度数、つまり 4 バイトを単位として行われる。

SIMD 拡張命令のレジスタ指定方法

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 2 演算を実行する。SIMD 演算で使われるレジスタの組は、f[2n] に対して f[2n+256] (n = 0–127) に固定されている。命令では f[2n] 側を指定する。使えないレジスタを指定した場合は *illegal_action* が通知される。

SIMD FMADD 命令だけは例外で、rs1, rs2 に f[2n+256] を指定できる。詳細は Appendix A.24.1, *"Floating-Point Multiply-Add/Subtract"* (page 70) を参照。

Programming Note – 単精度浮動小数点命令も SIMD 拡張できるが、その際は、倍精度レジスタを単精度で使う。詳細は *" 倍精度レジスタの単精度利用 "* (page 21) を参照。

既存浮動小数点命令のうち SIMD 拡張されない命令には以下の命令が含まれる。
SIMD 拡張可能な命令の一覧は TABLE A-2 を参照。

- FDIV(S,D), FSQRT(S,D)
- VIS のうち論理演算 (FOR, FAND, etc.) でないもの
- FBfcc, FBPfcc, FCMP, FCMPE, FMOVcc など、fcc, icc, xcc を参照、更新する命令
- FMOVr

なお、SIMD の 2 演算を区別するときは、f[2n] に結果が格納される側の演算を basic 側、f[2n+256] に結果が格納される側の演算を extended 側と呼ぶ。

エンディアン変換は basic, extended 毎に行われる。

5.1.7 Floating-Point State Register (FSR)

FSR_nonstandard_fp (NS)

SPARC V9 仕様では FSR.NS ビットを定義している。このビットを 1 にセットすることで、浮動小数点演算器が IEEE Std 754-1985 に準拠しない演算処理を行っても良いことになっている。SPARC64 VIIIfx は FSR.NS ビットをサポートしている。

FSR.NS に 1 がセットされていると、演算の入力が非正規化数の場合や結果が非正規化数の場合に *fp_exception_other* が ftt = *unfinished_FPop* 通知されるのではなく、同符号の 0 に置き換えられて、*fp_exception_ieee_754* が fsr.cexc.nxc = 1 で通知される (この例外は FSR.TEM.NXM でマスクすることができる)。詳細は Appendix B.6, *"Floating-Point Nonstandard Mode"* (page 139) を参照。

FSR.NS に 0 がセットされていると、IEEE Std 754-1985 に準拠して動作する。

FSR_version (*ver*)

個々の SPARC V9 IU 実装 (VER.impl で識別される) は、1 個以上の FPU を実装してもよいし実装しなくてもよい。このフィールドはこの CPU に実装されている FPU のバージョンを示す。SPARC64 VIIIfx の最初の版では FSR.ver = 0 である (impl.dep. #19)。しかし、将来の版では別の値になっているかもしれない。詳細はデータシートを参照。

FSR_floating-point_trap_type (*ftt*)

SPARC64 VIIIfx が *fp_exception_other* を ftt = *unfinished_FPop* で通知する条件は Appendix B.6.1, *"fp_exception_other Exception (ftt=unfinished_FPop)"* (page 140) を参照 (impl.dep. #248)。

FSR_current_exception (cexc)

ビット 4 から 0 は、IEEE 754 で定義された例外のうちどれが発生したかを示す。例外が発生していなければ、対応するビットには 0 がセットされる。

SPARC64 VIIIfx が cexc ビットをどのようにセットするかを擬似コードで示す。

```
if (<LDFSR or LDXFSR commits>)
   <update using data from LDFSR or LDXFSR>;
else if (<FPop commits with ftt = 0>)
   <update using value from FPU>
else if (<FPop commits with IEEE_754_exception>)
   <set one bit[1] in the CEXC field as supplied by FPU>;
else if (<FPop commits with unfinished_FPop error>)
   <no change>;
else if (<FPop commits with unimplemented_FPop error>)
   <no change>;
else
   <no change>;
```

1. non-SIMD 時は 1 ビットだが、SIMD では複数ビットセットされることがある。

FSR Conformance

SPARC V9 では TEM, cexc, aexc 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 VIIIfx は (1)、すなわちこれら 3 つを全部実装している。別の実装方法については JPS1 **Commonality** の Section 5.1.7 を参照。

SIMD 演算における cexc, aexc 更新

SIMD 演算では basic, extended 両方の演算が同時に行われる。入力データは basic, extended で異なるので、両方で例外が起きることも、片方だけ起きることもある。

片方で例外が起きた場合の動作は、non-SIMD 演算と同じである。両方で例外が起きた場合、SPARC64 VIIIfx の SIMD 演算での例外通知と ftt の更新は以下のようになる。

1. basic, extended 両方で *fp_exception_ieee_754* 例外が検出された場合

   説明のために、basic で起きた例外を保持する basic.cexc レジスタ、extended で起きた例外を保持する extend.cexc レジスタを仮定し、それぞれが、uf/of/dz/nx/nv のビットを持つものとする。

   a. 両方とも例外通知がマスクされており、例外を通知しない場合

      fsr.cexc には basic.cexc と extend.cexc の論理和が表示され、fsr.aexc には basic.cexc と extend.cexc の論理和が反映される。

      ```
      fsr.cexc  ← basic.cexc | extend.cexc
      fsr.aexc  ← fsr.aexc | basic.cexc | extend.cexc
      ```

   b. basic か extended のどちらかの例外が通知される場合

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

      ```
      fsr.cexc  ← basic.cexc | extend.cexc
      ```

   c. basic, extended 両方とも例外が通知される場合

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

      ```
      fsr.cexc  ← basic.cexc | extend.cexc
      ```

2. 一方で *fp_exception_ieee_754*, 他方で *fp_exception_other* が検出された場合

   例外の優先順位と異なり、*fp_exception_other* 例外が FSR.ftt = unfinished_FPop で通知される。FSR.aexc, FSR.cexc は更新されない。

Programming Note – *fp_exception_other* が通知された場合、同時に *fp_exception_ieee_754* 例外が起きたかどうかを判断する方法はないので、システム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も行うこと。

3. basic, extended 両方で *fp_exception_other* が検出された場合

   fp_exception_other 例外が `FSR.ftt = unfinished_FPop` で通知される。`FSR.aexc`, `FSR.cexc` は更新されない。

Note – non-SIMD 演算の場合、*fp_exception_ieee_754* が通知されるときは `fsr.cexc` には 1 要因のみが表示される。SIMD 演算の場合、basic と extended の要因の論理和が表示されるので、複数要因が表示されることがある。

5.1.9 Tick (TICK) Register

SPARC64 VIIIfx は `TICK.counter` を 63 ビット幅で実装している (Impl. Dep. #105).

Implementation Note – SPARC64 VIIIfx では、`TICK` の読み出し時に `counter` に表示される値は、`RDTICK` 命令が実行されたときのものであり、完了したときのものではない (SPARC64 VIIIfx はアウトオブオーダ実行なので両者は明らかに異なる)。`TICK`の読み出しを2回行ったとき、`counter`の差分は、2回の`RDTICK`命令の実行の間の CPU サイクル数である。間に挟まる命令列が十分長ければ、実行の間の CPU サイクル数と完了の間の CPU サイクル数の差は小さいだろう。

5.2 Privileged Registers

5.2.6 Trap State (TSTATE) Register

SPARC64 VIIIfx は `TSTATE.CWP` のビット <2:0> のみを実装している。ビット 3 とビット 4 の書き込みは無視され、読み出しには 0 が返る。

Note – ソフトウェアで `PSTATE.RED` に 1 を設定すべきではない。`RED_state` に遷移する際に必要な、ハードウェア内部の処理をせずに、遷移する可能性があるためである。

5.2.9 Version (VER) Register

TABLE 5-1 に SPARC64 VIIIfx の VER の各フィールドと値を示す。

TABLE 5-1 VER のエンコーディング

ビット	フィールド	説明
63:48	manuf	0004_{16} (Impl. Dep. #104)
47:32	impl	8
31:24	mask	*n* (*n* の値はプロセッサチップの版数による)
15:8	maxtl	5
4:0	maxwin	7

manuf フィールドには、JEDEC で定められた富士通を示す 8 ビットコードが表示され、上位 8 ビットは 0 となる。manuf, impl, mask フィールドの値はその性質上、将来のプロセッサでは変更されることがある。mask フィールドは論理回路が変更されるたびに大きい数字が入ることになるが、それが連続した数字である必要はない。

5.2.11 Ancillary State Registers (ASRs)

ASR の詳細については、JPS1 **Commonality** の Section 5.2.11 を参照。

Performance Control Register (PCR) (ASR 16)

SPARC64 VIIIfx の PCR 仕様は JPS1 **Commonality** 仕様と一部が異なる。FIGURE 5-2 と TABLE 5-2 で、JPS1 **Commonality** で定義された impl.dep. #207, #250 の SPARC64 VIIIfx での仕様と、PCR.SU, PCR.SL の JPS1 **Commonality** 仕様からの変更分を含む、PCR の仕様を説明する。PCR<2:1> は JPS1 **Commonality** に準拠している。

PA イベントカウンタの詳細は Appendix Q を参照。

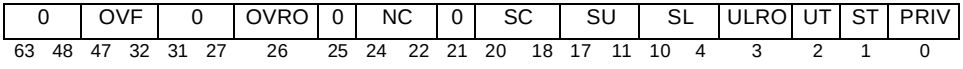


FIGURE 5-2 PA 設定レジスタ (Performance Control Register, PCR) (ASR 16)

TABLE 5-2 PCR のフィールド

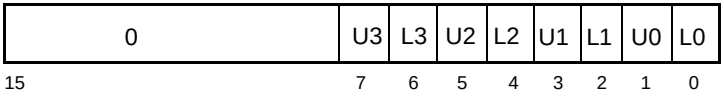
ビット	フィールド	説明
47:32	OVF	オーバーフロー。RDPCR でカウンタのオーバーフローの情報が読み出せ、WRPCR でオーバーフロー情報をセット・クリアすることができる。PCR.OVF は SPARC64 VIIIfx 独自のフィールドである (impl. dep. #207)。 OVF のビットとカウンタの対応は以下の通りである。OVF のビットに 0 を書き込むことで、対応するカウンタのオーバーフロー情報をクリアできる。  ソフトウェアが 1 を書き込んでも、オーバーフローの例外は通知されない。
26	OVRO	オーバーフロー情報の変更禁止。書き込みの際、書き込もうとするデータの OVRO が 0 だと、データの OVF で PCR.OVF が更新される。データの OVRO が 1 だと、データの OVF の値は無視され、PCR.OVF は書き込みによって更新されない。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。PCR.OVF はハードウェアが常に最新状態に保つので、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。PCR.OVRO は SPARC64 VIIIfx 独自のフィールドである (impl. dep. #207)。
24:22	NC	カウンタペアの数。このフィールドはリードオンリーで、SPARC64 VIIIfx では 3 (カウンタペア 4 つ) である。
20:18	SC	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。

TABLE 5-2 PCR のフィールド

ビット	フィールド	説明
17:11	SU	PIC<63:32> 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、読みだしでは現在の設定が返される。JPS1 **Commonality** の仕様を 1 ビット拡張し、7 ビットのフィールドとする。
10:4	SL	PIC<31:0> 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、読みだしでは現在の設定が返される。JPS1 **Commonality** の仕様を 1 ビット拡張し、7 ビットのフィールドとする。
3	ULRO	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 PCR.SU, PCR.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、PCR.SU, PCR.SL は更新されない。読み出しには 0 が返る。 ULRO は、SU, SL の設定を変更することなく PIC のマッピングを変更するためのものである。SPARC64 VIIIfx 独自のフィールドである (impl. dep. #207)。
2	UT	ユーザモード。PSTATE.PRIV = 0 で発生したイベントを計測する。
1	ST	システムモード。PSTATE.PRIV = 1 で発生したイベントを計測する。 PCR.UT と PCR.ST がともに 1 の場合、すべての場合で発生したイベントが計測される。PCR.UT と PCR.ST がともに 0 の場合、計測は行われない。 PCR.UT と PCR.ST はグローバルなフィールドで、すべての PIC に対して適用される。
0	PRIV	特権アクセス。PCR.PRIV = 1 のとき、非特権モード (PSTATE.PRIV = 0) で RDPCR, WRPCR, RDPIC, WRPIC を実行すると *privileged_action* 例外が通知される。 PCR.PRIV = 0 のときは、PSTATE.PRIV = 0 で RDPCR の実行は正常に完了し、WRPCR は PCR.PRIV を変更しようとすると (1 を書き込もうとすると) *privileged_action* が通知される (impl. dep. #250)。

Performance Instrumentation Counter (PIC) Register (ASR 17)

PIC は JPS1 **Commonality** に準拠する。

SPARC64 VIIIfx では PIC は 4 組実装される。PCR.SC で指定された PIC が ASR 17 でアクセスできる。PIC のアクセスは PICU, PICL カウンタのペアにアクセスすることになる。PICU, PICL のエンコーディングについては Appendix Q を参照。

オーバフロー発生時、カウンタは 0 にラップアラウンドし、SOFTINT レジスタのビット 15 に 1 がセットされ、interrupt level-15 例外が通知される。カウンタオーバフローのトラップは、カウンタ値が $\texttt{FFFF FFFF}_{16}$ から 0 に更新される際に通知される。同時に複数のオーバーラップが起きたときは、複数のオーバーラップビットが 1 にセットされる。すでに 1 にセットされているオーバーフロービットは 1 のまま変更されない。

オーバーフロービットは、ソフトウェアが対応する PCR.OVF に 0 を書き込むことでクリアされる。ソフトウェアは 1 を書き込むことができるが、この書き込みによるオーバーフロートラップは発生しない。

Dispatch Control Register (DCR) (ASR 18)

SPARC64 VIIIfx は DCR を実装しない。DCR の読み出しには 0 が返り、書き込みは無視される。DCR は特権レジスタである。非特権モードでのアクセスには *privileged_opcode* 例外が通知される。

Extended Arithmetic Register (XAR) (ASR 29)

新規に追加された非特権レジスタである。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 は all0 になる。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 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。

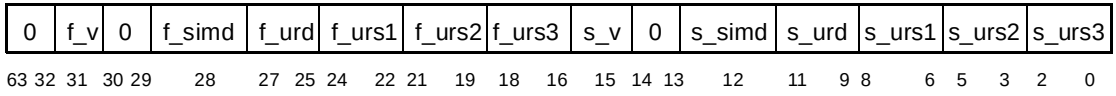


TABLE 5-3 XAR のフィールド

ビット	フィールド	説明
63:32	—	Reserved. このフィールドに 0 以外の値を書くと、*illegal_instruction* 例外が通知される。
31	f_v	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。
30:29	—	Reserved. このフィールドに 0 以外の値を書くと、*illegal_instruction* 例外が通知される。
28	f_simd	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。
27:25	f_urd	1 番目の命令の rd の拡張フィールド。
24:22	f_urs1	1 番目の命令の rs1 の拡張フィールド。

TABLE 5-3 XAR のフィールド

ビット	フィールド	説明
21:19	f_urs2	1 番目の命令の rs2 の拡張フィールド。
18:16	f_urs3	1 番目の命令の rs3 の拡張フィールド。
15	s_v	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。
14:13	—	Reserved. このフィールドに 0 以外の値を書くと、*illegal_instruction* 例外が通知される。
12	s_simd	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。
11:9	s_urd	2 番目の命令の rd の拡張フィールド。
8:6	s_urs1	2 番目の命令の rs1 の拡張フィールド。
5:3	s_urs2	2 番目の命令の rs2 の拡張フィールド。
2:0	s_urs3	2 番目の命令の rs3 の拡張フィールド。

本仕様書における XAR の記述について

Table 5-3 のフィールド名以外に以下の別名表記も用いる。

- メモリアクセス用

別名	実際のフィールド
XAR.f_dis_hw_pf	XAR.f_urs3<1>
XAR.s_dis_hw_pf	XAR.s_urs3<1>
XAR.f_sector	XAR.f_urs3<0>
XAR.s_sector	XAR.s_urs3<0>

- SIMD FMA 用

別名	実際のフィールド
XAR.f_negate_mul	XAR.f_urd<2>
XAR.s_negate_mul	XAR.s_urd<2>
XAR.f_rs1_copy	XAR.f_urs3<2>
XAR.s_rs1_copy	XAR.s_urs3<2>

- 総称
  1 命令目、2 命令目の区別なしに記述する場合は XAR.f_v, XAR.s_v の値により 1 命令目、2 命令目のどちらかを選択したものとする。

記述法	XAR.f_v = 1 のとき	XAR.f_v = 0 かつ XAR.s_v = 1 のとき
XAR.v	XAR.f_v	XAR.s_v
XAR.urd	XAR.f_urd	XAR.s_urd
XAR.urs1	XAR.f_urs1	XAR.s_urs1
XAR.urs2	XAR.f_urs2	XAR.s_urs2
XAR.urs3	XAR.f_urs3	XAR.s_urs3
XAR.dis_hw_pf	XAR.f_dis_hw_pf	XAR.s_dis_hw_pf
XAR.sector	XAR.f_sector	XAR.s_sector
XAR.negate_mul	XAR.f_negate_mul	XAR.s_negate_mul
XAR.rs1_copy	XAR.f_rs1_copy	XAR.s_rs1_copy

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、TABLE A-2 (page 59) 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* 例外が通知される。
- 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。

    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 Register Number Encoding”* (page 20) を参照。
  - 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 の関連フィールドは all0 になる。
- XAR 対象命令であっても、以下のどれかにあたる場合 *illegal_action* が通知される。
  - 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* が通知される。
  - FDIV(S,D), FSQRT(S,D) 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 FMADD は、rs1, rs2 に f[256] 以上を指定可能。
    XAR.urs3<2>, XAR.urd<2> も 1 になってもよいが、これは f[256] 以上の指定ではなく、別の意味に解釈される。詳細は *“FMADD の SIMD 拡張 ”* (page 73) を参照。
  - ld/st/atomic 命令で XAR.urs3<2> ≠ 0 のとき。

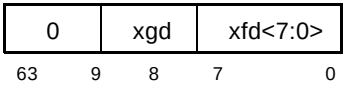
後続 1 命令のレジスタ番号を指示する場合、1st を使っても 2nd を使ってもよい。

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。

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

新規追加非特権レジスタ。



ビット	フィールド	属性	説明
63:9	—	R	Reserved.
8	xgd	RW	xg[0] – xg[31] が更新されると、xgd = 1 になる。
7:0	xfd<7:0>	RW	浮動小数点レジスタが更新されると、対応するビットに 1 がセットされる。

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] を更新すると XASR.xgd = 1 がセットされる。
- 浮動小数点レジスタを更新すると、対応する xfd<i> = 1 がセットされる。レジスタと xfd のビット対応は以下の通り。

浮動小数点レジスタ	対応する XASR のビット
f[0] – f[62]	xfd<0>
f[64] – f[126]	xfd<1>
f[128] – f[190]	xfd<2>
f[192] – f[254]	xfd<3>
f[256] – f[318]	xfd<4>
f[320] – f[382]	xfd<5>
f[384] – f[446]	xfd<6>
f[448] – f[510]	xfd<7>

Programming Note – xfd[0] は V9 FPR を更新したときにセットされる。このとき、V9 の 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。

Implementation Note – 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。

Trap XAR Registers (TXAR) (ASR 31)

新規追加特権レジスタ。フィールドは XAR と同じである。

TXAR はトラップ時に XAR が保存されるレジスタである。レジスタフィールドの定義は XAR と同じである。TXAR[1] – TXAR[MAXTL] までのレジスタが定義されている。TL > 0 のときは、TXAR[TL] が見える。TL を変更すると、変更直後の命令から新しい TL に対応する TXAR[TL] が read/write できる。

TL = 0 で read/write すると *illegal_instruction* が通知される。*reserved* フィールドに 0 でない値を書くと *illegal_instruction* が通知される。

5.2.12 Registers Referenced Through ASIs

本節では JPS1 **Commonality** の 5.2.12 で定義されている ASI レジスタについてのみ記述する。その他の ASI レジスタについては Appendix L を参照。

Data Cache Unit Control Register (DCUCR)

ASI 45_{16} (ASI_DCU_CONTROL_REGISTER), VA = 00_{16}.

DCUCR は、命令、プリフェッチ、書き込みとデータキャッシュ、MMU、ウォッチポイントなど、メモリアクセスに関連するハードウェアの機能を制御するためのレジスタである。SPARC64 VIIIfx の DCUCR は JPS1 **Commonality** で定義された仕様をほぼ実装している。

DCUCR のビット配置を FIGURE 5-3 に、ビットの説明を TABLE 5-4 に示す。

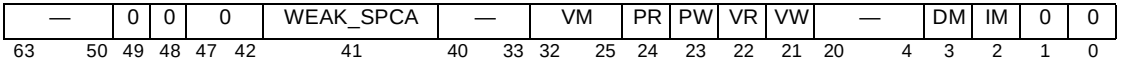


FIGURE 5-3 DCUCR (ASI 45_{16})

TABLE 5-4 DCUCR のフィールド

ビット	フィールド	属性	説明
63:50	—		Reserved
49:48	CP, CV	R	SPARC64 VIIIfx では実装されない。読み出しには 0 が返り、書き込みは無視される。
47:42	impl. dep.	R	読み出しには 0 が返り、書き込みは無視される。
41	WEAK_SPCA	RW	投機的なメモリアクセスの抑止 (impl. dep. #240)。WEAK_SPCA に 1 がセットされると、分岐予測が行われなくなる。命令のプリフェッチは常に分岐しない方向に行われる。分岐命令以降のロード、ストアは、分岐方向が確定するまで行われない。ハードウェアプリフェッチ機構は停止され、プリフェッチ命令はストロングプリフェッチも含めすべて無効となる。 命令プリフェッチのアクセス範囲は CPU 内部リソースによって決まる一定範囲内に留まるので、weak_spca = 1 ではメモリアクセスの範囲は推定可能である。
40:33	PM<7:0>		Reserved.
32:25	VM<7:0>	RW	Data Watchpoint Register のマスクを指定する。SPARC64 VIIIfx では Data Watchpoint Register は物理アドレス、論理アドレス共用。
24, 23	PR, PW	RW	Data Watchpoint Register の値を物理アドレスと解釈して VM で指定されたアドレスに read または write アクセスすると *PA_watchpoint* 例外が通知される。
22, 21	VR, VW	RW	Data Watchpoint Register の値を論理アドレスと解釈して VM で指定されたアドレスに read または write アクセスすると *VA_watchpoint* 例外が通知される。
20:4	—		Reserved.
3	DM	RW	Data MMU 有効化。DM を 0 にセットすると、データアクセス時にアドレス変換が行われない。このとき、論理アドレスはそのまま物理アドレスとして使われる。
2	IM	RW	命令 MMU 有効化。IM を 0 にセットすると、データアクセス時にアドレス変換が行われない。このとき、論理アドレスはそのまま物理アドレスとして使われる。
1	DC	R	SPARC64 VIIIfx では実装されない。読み出しには 0 が返り、書き込みは無視される。
0	IC	R	SPARC64 VIIIfx では実装されない。読み出しには 0 が返り、書き込みは無視される。

Implementation Note – DCUCR.WEAK_SPCA = 1 のとき、CTI 以降の命令プリフェッチは、最大でもその CTI から 1KB を越えることはない。

Programming Note – 投機メモリアクセス抑止を確実に行うために、システムソフトウェアは DCUCR.WEAK_SPCA = 1 に設定後直ちに membar #Sync を発行すること。

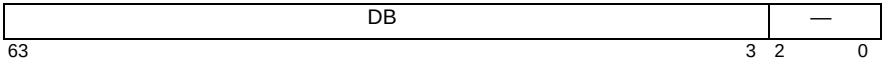
Programming Note – SPARC64 VIIIfx では DCUCR の IM (IMMU enable), DM (DMMU Enable) 変更は以下の命令列で行う必要がある。

```
# DCUCR.IM update
stxa DCUCR
flush

# DCUDR.DM update
stxa DCUCR
membar #sync
```

Data Watchpoint Registers

レジスタ名	ASI_WATCHPOINT
ASI	58_{16}
VA	38_{16}
アクセス種別	Supervisor Read/Write



ビット	フィールド	属性	説明
63:3	DB	RW	ウォッチポイントアドレス (VA, PA 共用)

JPS1 **Commonality** の TABLE 5-18 で、ウォッチポイントの対象は translating ASI, bypass ASI、であると定義している。しかしこの表の定義に従うと、実装依存 ASI や未定義 ASI もウォッチポイントの対象となってしまうので、SPARC64 VIIIfx では translating, bypass, nontranslating を定義し直した (TABLE L-1 (page 214) を参照)。この表で translating, bypass と定義された ASI がウォッチポイントの対象となる。

JPS1 **Commonality** ではウォッチポイントを、論理アドレス・物理アドレス独立に設定できるが、SPARC64 VIIIfx では、アドレスの指定は一つだけで、そのアドレスを論理アドレス・物理アドレスと解釈した場合にマッチするかどうかを監視するよう、

仕様を変更する。JPS1 **Commonality** の ASI_VA_WATCHPOINT (ASI = 58_{16}, VA = 38_{16}) を ASI_WATCHPOINT という名前にし、ASI_PA_WATCHPOINT (ASI = 58_{16}, VA = 40_{16}) は削除する。

Compatibility Note – この変更は SPARC JPS1 非互換である。

ウォッチポイントの有効・無効の指定方法は、SPARC JPS1 仕様に準拠し、DCUCR.VR, DCUCR.VW, DCUCR.PR, DCUCR.PW で指定する。DCUCR.VR または DCUCR.VW に 1 がセットされていると、DB を論理アドレスと解釈してマッチした場合に *VA_watchpoint* が通知され、DCUCR.PR または DCUCR.PW に 1 がセットされていると、DB を物理アドレスと解釈してマッチした場合に *PA_watchpoint* が通知される。論理アドレス・物理アドレスどちらでもマッチした場合は、*VA_watchpoint* が通知される。

ウォッチポイントが有効である ASI 空間の一覧が JPS1 **Commonality** の TABLE 5-18 で定義されているが、この表は SPARC64 VIIIfx で未実装である ASI も含まれている。SPARC64 VIIIfx は、未実装である translating ASI ではウォッチポイントの検出は行わず、アクセスすると *data_access_exception* が通知される。

物理アドレスとしてマッチしているかどうかを判定する際、DB<63:41> は無視される。

SIMD ロード、SIMD ストア命令では、basic 側、extended 側のどちらでもウォッチポイントを検出する。アドレスとマスクが basic 側のアドレスとアクセス長にマッチすれば、basic 側の *VA_watchpoint*, *PA_watchpoint* が通知され、アドレスとマスクが extended 側のアドレスとアクセス長にマッチすれば、extended 側の *VA_watchpoint*, *PA_watchpoint* が通知される。

ウォッチポイントの信頼性を下げる機能は SPARC64 VIIIfx には実装されていない (impl. dep. #244)。

以下の命令については、ウォッチポイントの設定と検出に関する特例がある。詳細はそれぞれの命令を参照。

- Appendix A.4, *"Block Load and Store Instructions (VIS I)"*
- Appendix A.30, *"Load Quadword, Atomic [Physical]"*
- Appendix A.42, *"Partial Store (VIS I)"*
- Appendix A.77, *"Store Floating-Point Register on Register Condition"*
- Appendix A.79, *"Cache Line Fill with Undetermined Values"*
- Appendix F.5.1, *"Trap Conditions for SIMD Load/Store"*

Instruction Trap Register

SPARC64 VIIIfx は命令トラップレジスタを実装する (impl. dep. #205)。

SPARC64 VIIIfx では、命令キャッシュに CALL, 条件分岐命令 (BPcc, FBPfcc, Bicc, BPr) が保持される際、下位 11 ビットは命令定義通りメモリ上と同じデータが保存される (impl. dep. #245)。

5.2.13 Floating-Point Deferred-Trap Queue (FQ)

SPARC64 VIIIfx は浮動小数点演算の遅延トラップキューを実装しない (impl. dep. #24). RDPR 命令で FQ を読み出そうとすると、*illegal_instruction* 例外を通知する (impl. dep. #25)。

5.2.14 IU Deferred-Trap Queue

SPARC64 VIIIfx は整数ユニットの遅延トラップキューを実装しない (impl. dep. #16)。

F.CHAPTER **6**

Instructions

6.1 Instruction Execution

SPARC64 VIIIfx は先進のスーパスケーラ機構を実装した SPARC V9 プロセッサである。一サイクルで複数の命令を発行・実行することができる。SPARC64 VIIIfx のプログラムセマンティクスは逐次命令実行モデルなので、この節の事項はソフトウェアには見えないが、効率と正確性を知る上で重要なのでここで説明する。

6.1.1 Data Prefetch

SPARC64 VIIIfx はプログラムの命令順序通りではなく、投機的に命令実行を行う (アウトオブオーダ)。投機実行された命令列が実行されるべき命令でないことがわかると、命令実行は取り消されるが、投機実行した命令列のメモリアクセスは取り消すことができない。SPARC64 VIIIfx の投機メモリアクセスの方針は以下の通りである。

1. あるメモリ操作 *x* のメモリアドレスが揮発性 (volatile) のとき (*location[x]*), SPARC64 VIIIfx は *location[x]* に対するプリフェッチを行わない。*location[x]* のフェッチは、*x* が確実に実行される (*committable*) と判明した後で行われる。
2. あるメモリ操作 *x* のメモリアドレスが不揮発性 (nonvolatile) のとき (*location[x]*), SPARC64 VIIIfx は *location[x]* に対するプリフェッチを行うかもしれない。その際は以下の方針に従う。
   a. メモリ操作 *x* がストアセマンティクスの操作で、キャッシャブル領域のアクセスの場合、データを排他的使用権を獲得するところまで行う。ストアセマンティクス以外の操作では、ノンキャッシャブルであってもプリフェッチを行う。
   b. アトミック操作 (`CAS(X)A`, `LDSTUB`, `SWAP`) のプリフェッチは行われない。

SPARC64 VIIIfx は投機的なロード命令の実行を防ぐための機構を 2 つ実装している。

1. ある種のページや I/O 空間に対しては投機アクセスを行わない。投機的なアクセスを禁止する場合は、PTE の E (side-effect) ビットに 1 をセットする。E ビットがセットされているページへのアクセスは、そのアクセスが投機的でなくなるまで待機させられる。詳細は Appendix F を参照。

2. ロードを強制的にプログラム順に行わせる ASI_PHYS_BYPASS_WITH_EBIT[_L] (ASI = 15_{16}, $1D_{16}$) は投機的には実行されない。

6.1.2 Instruction Prefetch

SPARC64 VIIIfx は、命令の供給が間に合わず実行が止まるということが極力起こらないよう、命令プリフェッチを行う。分岐予測の結果によっては、実際には実行しない命令列をプリフェッチすることがある。さらに、投機実行した命令がメモリアクセスをすることもある。命令プリフェッチや投機メモリアクセスによる例外を検出してもすぐには通知せず、その命令の直前の命令が完了してから通知される。[1]

6.1.3 Syncing Instructions

SPARC64 VIIIfx はある種の命令を *sync* 命令として扱う。sync 命令は実行するとパイプラインを数サイクルの間停止させるような命令である。sync 命令には *pre sync* と *post sync* の 2 種類がある。pre sync 命令は、プログラム順でその命令より前にある命令の実行が完了してから、その命令だけを実行完了し、その後で、後続命令を実行するような副次的効果を持つ命令である。post sync 命令は、プログラム順でその命令より後にある命令の発行を、その命令が実行完了するまで停止する副次的効果を持つ命令である。pre sync かつ post sync 属性を持つ命令も存在する。

SPARC64 VIIIfx では、ストアを除くすべての命令がプログラム順に完了する。ストアはその結果が global visible になる前に完了する (ストアの突き放し完了)。

6.2 Instruction Formats and Fields

SPARC64 VIIIfx の命令フォーマットは 5 種類に大別される。そのうち 4 種類の命令フォーマットは JPS1 **Commonality** の Section 6.2 を参照。5 番目の命令フォーマットは SPARC64 VIIIfx 独自のフォーマットである。

1. ハードウェアのエラーやその他命令実行とは非同期に起きるエラーは、そのエラーを起こした命令がの完了を待たずに通知される。

Format 5 (op = 2, op3 = 37_{16}): FMADD, FPMADDX, FSELMOV, and FTRIMADD (in place of IMPDEP2A and IMPDEP2B)

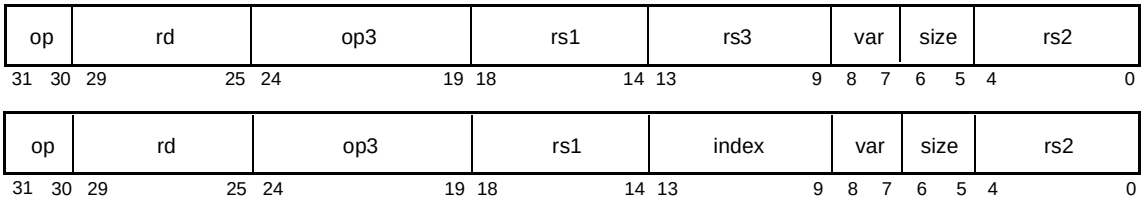


FIGURE 6-1 Summary of Instruction Formats: Format 5

命令フィールドの説明は JPS1 **Commonality** の TABLE 6-1 を参照。Format 5 で追加された 4 つのフィールドの説明を下表に示す。

TABLE 6-1 Format 5 の命令フィールドの解釈

フィールド	説明
rs3	この 5 ビットフィールドは、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。
var	この 2 ビットフィールドは、浮動小数点積和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。
size	この 2 ビットフィールドは、浮動小数点積和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。
index	FTRIMADDd で係数テーブルを指示するために使われる。

6.3 Instruction Categories

6.3.3 Control-Transfer Instructions (CTIs)

実行制御命令には以下の種類がある。

- 条件分岐 (Bicc, BPcc, BPr, FBfcc, FBPfcc)
- 無条件分岐
- Call and link (CALL)
- Jump and link (JMPL, RETURN)

- Return from trap (DONE, RETRY)
- Trap (Tcc)

この仕様書では SPARC64 VIIIfx の CALL と JMPL 命令について説明する。それ以外の実行制御命令は JPS1 **Commonality** を参照。

CALL and JMPL Instructions

SPARC64 VIIIfx は PSTATE.AM = 0 のとき、PC の 64 ビットすべてをデスティネーションレジスタに書き込む。PSTATE.AM = 1 のときは上位 32 ビットをゼロにした値をデスティネーションレジスタに書き込む。

6.3.7 Floating-Point Operate (FPop) Instructions

FPop が *fp_exception_other* 例外を FSR.ftt = *unfinished_FPop* で通知する正確な条件は、Appendix B.6, *"Floating-Point Nonstandard Mode"* (page 139) で定義する。

6.3.8 Implementation-Dependent Instructions

SPARC64 VIIIfx は IMPDEP1, IMPDEP2 に、浮動小数点演算命令を定義している。JPS1 **Commonality** では FPop を " オペコードが FPop1 または FPop2 である命令 " と定義しているので、IMPDEP の命令は FPop ではないことになる。

IMPDEP2 で定義された浮動小数点積和演算命令のうち FMADD, FMSUB, FNMSUB には 4 倍精度演算が定義されているが、SPARC64 VIIIfx では 4 倍精度演算は実装されないので、これらの命令を実行すると *illegal_instruction* 例外が通知される。FNMADD だけは 4 倍精度演算が定義されていない。4 倍精度浮動小数点積和演算は SPARC V9 で必須の命令ではないので、システムソフトウェアはエミュレーションしなくてもよい。

SPARC64 VIIIfx で IMPDEP1, IMPDEP2 に追加された命令のうち、浮動小数点レジスタを使う以下の命令は、PSTATE.PEF = 0 または FPRS.FEF = 0 のときに実行すると、*fp_disabled* 例外が通知される。

FCMP(GT,LE,EQ,NE,GE,LE)E(s,d), FCMP(EQ,NE)(s,d), FMAX(s,d), FMIN(s,d), FRCPA(s,d), FRSQRTA(s,d), FTRISSELd, FTRISMULd, FTRIMADDd, FSELMOV(s,d), F{N}M(ADD,SUB)(s,d), FPMADDX{HI}, ST{D}FR

これらは FPop ではないので、reserved であるオペコードを実行しようとすると、JPS1 **Commonality** 6.3.9 で定義されている通り、*illegal_instruction* 例外が通知される。また、これらの命令のうち FPMADDX{HI}, ST{D}FR 以外は、JPS1 **Commonality** 6.3.7 の FPop と同様に FSR を更新する。FTRISSELd, FSELMOV(s,d) は *fp_exception_ieee_754* 例外を通知しない命令だが、実行が完了すると FSR.cexc の全フィールドに 0 をセットし、FSR.aexc は変更しない。

F.CHAPTER **7**

Traps

7.1 Processor States, Normal and Special Traps

Appendix O.1, *“ リセット種類 ”* (page 245)JPS1 **Commonality** ではこの節で CPU のステートと遷移を定義しているが、SPARC64™ VIIIfx Extensions では Appendix O.1, *“ リセット種類 ”* (page 245) で定義しているので、そちらを参照されたい。

7.1.1 RED_state

Appendix O.2.1, *“RED_state”* (page 248) も参照。

RED_state Trap Table

`RED_state` 用のトラップテーブルは実装依存のアドレス `RSTVaddr` に配置される。`RSTVaddr` の値はある実装に対しては固定となっており、SPARC64 VIIIfx では論理アドレス FFFF FFFF F000 0000_{16}、物理アドレス 0000 01FF F000 0000_{16} である (impl. dep. #114)。

RED_state Execution Environment

`RED_state` では CPU はある制約下で動作するが、このためにいくつかの CPU 制御用レジスタの値は固定値で上書きされる。

Note – 遷移をアトミックに行うため、値は上書きであって代入ではない。

`RED_state` での SPARC64 VIIIfx の動作は以下の通りである (impl.dep. #115)

- RED_state である間は、ITLB によるアドレス変換機能は無効となる。DTLB によるアドレス変換は RED_state 遷移直後は無効だが、RED_state 中にソフトウェアによって有効にすることができる。RED_state 中でも、ASI レジスタ経由で TLB を参照・変更する機能は有効である。
- TLB によるアドレス変換が無効である間、すべてのメモリアクセスはノンキャッシャブル、ストロングオーダとして扱われる。
- XIR はマスクされないので、XIR を受けると例外を通知する。

Note – CPU 内のエラーにより RED_state に入った場合、ソフトウェアは致命的なエラーからの復旧やエラーを起こしたコンポーネントを無効にするなどの処理を行うべきである。リセット後に RED_state に入った場合、ソフトウェアはシステムを実行状態にするためのセットアップを行うこと。

7.1.2 error_state

プロセッサは、トラップレベルが最大 (TL = MAXTL) のときにトラップが起きると error_state に遷移する (impl. dep. #39)。

CPU は、CPU 内部で生成される *watchdog_reset* (WDR) によって error_state に遷移するが、WDR を抑止し、error_state で留まるように設定することもできる (impl. dep #40, #254)。

7.2 Trap Categories

7.2.2 Deferred Traps

SPARC64 VIIIfx ではエラー処理の例外が deferred で通知されることがある。詳細は Appendix P.2.2, *“ エラー検出時の動作 ”* (page 261) または Appendix P.4.3, *“ADE トラップ発生した時の命令の実行状況 ”* (page 281) を参照。

7.2.4 Reset Traps

SPARC64 VIIIfx は命令完了が約 6.7 秒の間行われない場合、watchdog reset (WDR) 例外を通知する。

7.2.5 Uses of the Trap Categories

SPARC64 VIIIfx では、命令実行に同期して起きる例外はすべて precise である (impl. dep. #33)。

複数のメモリアクセスとして実行される命令 (LDD(A), STD(A), LDSTUB, CASA, CASXA, SWAP など) が、最初のアクセスの後で致命的なエラーを起こした場合、precise で通知される。

7.3 Trap Control

7.3.1 PIL Control

SPARC64 VIIIfx はシステムからのインタラプト要求を受信すると、*interrupt_vector_trap* (TT = 60_{16}) を通知する。トラップハンドラはインタラプト情報を読んだ後、SPARC V9 準拠のインタラプトで通知しなおす。この通知はソフトウェアが SOFTINT を書き込むことで行われる。詳細は JPS1 **Commonality** の Section 5.2.11 を参照。

SPARC64 VIIIfx は SPARC V9 準拠のインタラプトを受けると、PIL レジスタの値をチェックする。優先順位の条件を満たすインタラプトがある場合、SPARC64 VIIIfx は新しい命令の発行をやめ、未完了の命令をすべてキャンセルし、トラップハンドラに処理を移す。ただし、実行中の命令列が、より優先順位が高いトラップハンドラのものである場合はこの限りではない。

SPARC64 VIIIfx はインタラプト要求を disrupting トラップとして処理する。

7.4 Trap-Table Entry Addresses

7.4.2 Trap Type (TT)

SPARC64 VIIIfx では以下のトラップを定義している (impl. dep. #35; impl. dep. #36)。

- *async_data_error*
- *illegal_action*
- *SIMD_load_across_pages*

JPS1 **Commonality** のトラップ一覧にこれらを含めたものを TABLE 7-1, TABLE 7-2 に示す。TABLE 7-1 の網掛け部分は、SPARC64 VIIIfx では起こらないトラップである。

TABLE 7-1 SPARC64 VIIIfx のトラップ一覧 (TT 番号順) *(1 of 2)*

SPARC V9 M/O	JPS1 M/O	トラップ名	TT	%g 種類	優先順位
●	●	*Reserved*	000_{16}	*-NA-*	*-NA-*
●	●	*power_on_reset*	001_{16}	AG	0
❍	●	*watchdog_reset*	002_{16}	AG	1
❍	●	*externally_initiated_reset*	003_{16}	AG	1
●	●	*software_initiated_reset*	004_{16}	AG	1
●	●	*RED_state_exception*	005_{16}	AG	1
●	●	*Reserved*	006_{16}–007_{16}	*-NA-*	*-NA-*
●	●	*instruction_access_exception*	008_{16}	MG	5
❍	❍	*instruction_access_MMU_miss*	009_{16}	MG (*impl. dep.)*	2
❍	●	*instruction_access_error*	$00A_{16}$	AG	3
●	●	*Reserved*	$00B_{16}$–$00F_{16}$	*-NA-*	*-NA-*
●	●	*illegal_instruction*	010_{16}	AG	7
●	●	*privileged_opcode*	011_{16}	AG	6
❍	❍	*unimplemented_LDD*	012_{16}	AG	6
❍	❍	*unimplemented_STD*	013_{16}	AG	6
●	●	*Reserved*	014_{16}–$01F_{16}$	*-NA-*	*-NA-*
●	●	*fp_disabled*	020_{16}	AG	8
❍	●	*fp_exception_ieee_754*	021_{16}	AG	11
❍	●	*fp_exception_other*	022_{16}	AG	11
		(ftt = *unimplemented_FPop* のみ)	022_{16}	AG	8.2
●	●	*tag_overflow*	023_{16}	AG	14
❍	●	*clean_window*	024_{16}–027_{16}	AG	10
●	●	*division_by_zero*	028_{16}	AG	15
❍	❍	*internal_processor_error*	029_{16}	*impl. dep.*	*impl. dep*
●	●	*Reserved*	$02A_{16}$–$02F_{16}$	*-NA-*	*-NA-*
●	●	*data_access_exception*	030_{16}	MG	12
❍	❍	*data_access_MMU_miss*	031_{16}	MG (*impl. dep.)*	12
❍	●	*data_access_error*	032_{16}	AG	12
❍	❍	*data_access_protection*	033_{16}	MG (*impl. dep.)*	12
●	●	*mem_address_not_aligned*	034_{16}	AG	10

TABLE 7-1 SPARC64 VIIIfx のトラップ一覧 (TT 番号順) *(2 of 2)*

SPARC V9 M/O	JPS1 M/O	トラップ名	TT	%g 種類	優先順位
❍	●	*LDDF_mem_address_not_aligned* (impl. dep. #109)	035_{16}	AG	10
❍	●	*STDF_mem_address_not_aligned* (impl. dep. #110)	036_{16}	AG	10
●	●	*privileged_action*	037_{16}	AG	11
❍	❍	*LDQF_mem_address_not_aligned* (impl. dep. #111)	038_{16}	AG	10
❍	❍	*STQF_mem_address_not_aligned* (impl. dep. #112)	039_{16}	AG	10
●	●	*Reserved*	$03A_{16}$–$03F_{16}$	*-NA-*	*-NA-*
❍	❍	*async_data_error*	040_{16}	AG	2
●	●	*interrupt_level_n* (*n* = 1–15)	041_{16}–$04F_{16}$	AG	32-*n*
●	●	*Reserved*	050_{16}–$05F_{16}$	*-NA-*	*-NA-*
❍	●	*interrupt_vector*	060_{16}	IG	16
❍	●	*PA_watchpoint*	061_{16}	AG	12
❍	●	*VA_watchpoint*	062_{16}	AG	11
❍	●	*ECC_error*	063_{16}	AG	33
❍	●	*fast_instruction_access_MMU_miss*	064_{16}–067_{16}	MG	2
❍	●	*fast_data_access_MMU_miss*	068_{16}–$06B_{16}$	MG	12
❍	●	*fast_data_access_protection*	$06C_{16}$–$06F_{16}$	MG	12
❍	❍	*implementation_dependent_exception_n* (impl. dep. #35)	070_{16}–072	*impl. dep.*	*impl. dep.*
❍	❍	*illegal_action*	073_{16}	AG	8.5
❍	❍	*implementation_dependent_exception_n* (impl. dep. #35)	074_{16}–076	*impl. dep.*	*impl. dep.*
❍	❍	*SIMD_load_across_pages*	077_{16}	AG	12
❍	❍	*implementation_dependent_exception_n* (impl. dep. #35)	078_{16}–07F	*impl. dep.*	*impl. dep.*
●	●	*spill_n_normal* (*n* = 0–7)	080_{16}–$09F_{16}$	AG	9
●	●	*spill_n_other* (*n* = 0–7)	$0A0_{16}$–$0BF_{16}$	AG	9
●	●	*fill_n_normal* (*n* = 0–7)	$0C0_{16}$–$0DF_{16}$	AG	9
●	●	*fill_n_other* (*n* = 0–7)	$0E0_{16}$–$0FF_{16}$	AG	9
●	●	*trap_instruction*	100_{16}–$17F_{16}$	AG	16
●	●	*Reserved*	180_{16}–$1FF_{16}$	*-NA-*	*-NA-*

TABLE 7-2 SPARC64 VIIIfx のトラップ一覧 (優先順位順、0 が高く番号が大きいほど低い) *(1 of 2)*

SPARC V9 M/O	JPS1 M/O	トラップ名	TT	%g 種類	優先順位
●	●	*power_on_reset* (POR)	001_{16}	AG	0
❍	●	*externally_initiated_reset* (XIR)	003_{16}	AG	1
❍	●	*watchdog_reset* (WDR)	002_{16}	AG	1
●	●	*software_initiated_reset* (SIR)	004_{16}	AG	1
●	●	*RED_state_exception*	005_{16}	AG	1
❍	❍	*async_data_error*	040_{16}	AG.	2
❍	●	*fast_instruction_access_MMU_miss*	064_{16}–067_{16}	MG	2
❍	●	*instruction_access_error*	$00A_{16}$	AG	3
●	●	*instruction_access_exception*	008_{16}	MG	5
●	●	*privileged_opcode*	011_{16}	AG	6
●	●	*illegal_instruction*	010_{16}	AG	7
●	●	*fp_disabled*	020_{16}	AG	8
❍	●	*fp_exception_other* (ftt = *unimplemented_FPop* のみ)	022_{16}	AG	8.2
❍	❍	*illegal_action*	073_{16}	AG	8.5
●	●	*spill_n_normal* (*n* = 0–7)	080_{16}–$09F_{16}$	AG	9
●	●	*spill_n_other* (*n* = 0–7)	$0A0_{16}$–$0BF_{16}$	AG	9
●	●	*fill_n_normal* (*n* = 0–7)	$0C0_{16}$–$0DF_{16}$	AG	9
●	●	*fill_n_other* (*n* = 0–7)	$0E0_{16}$–$0FF_{16}$	AG	9
❍	●	*clean_window*	024_{16}–027_{16}	AG	10
❍	●	*LDDF_mem_address_not_aligned* (impl. dep. #109)	035_{16}	AG	10
❍	●	*STDF_mem_address_not_aligned* (impl. dep. #110)	036_{16}	AG	10
●	●	*mem_address_not_aligned*	034_{16}	AG	10
❍	●	*fp_exception_ieee_754*	021_{16}	AG	11
❍	●	*fp_exception_other* (ftt = *unimplemented_FPop* 以外)	022_{16}	AG	11
●	●	*privileged_action*	037_{16}	AG	11
❍	●	*VA_watchpoint*	062_{16}	AG	11
●	●	*data_access_exception*	030_{16}	MG	12
❍	●	*fast_data_access_MMU_miss*	068_{16}–$06B_{16}$	MG	12
❍	●	*data_access_error*	032_{16}	AG	12
❍	●	*PA_watchpoint*	061_{16}	AG	12
❍	●	*fast_data_access_protection*	$06C_{16}$–$06F_{16}$	MG	12
❍	❍	*SIMD_load_across_pages*	077_{16}	AG	12

TABLE 7-2 SPARC64 VIIIfx のトラップ一覧 (優先順位順、0 が高く番号が大きいほど低い) *(2 of 2)*

SPARC V9 M/O	JPS1 M/O	トラップ名	TT	%g 種類	優先順位
●	●	*tag_overflow*	023_{16}	AG	14
●	●	*division_by_zero*	028_{16}	AG	15
●	●	*trap_instruction*	100_{16}–$17F_{16}$	AG	16
❍	●	*interrupt_vector*	060_{16}	IG	16
●	●	*interrupt_level_n* (n = 1–15)	041_{16}–$04F_{16}$	AG	32-*n*
❍	●	*ECC_error*	063_{16}	AG	33

7.4.3 Trap Priorities

SPARC64 VIIIfx では一部のトラップの優先順位を JPS1 **Commonality** 定義から変更している。

- *fp_exception_other* の優先順位は JPS1 **Commonality** では 11 だが、SPARC64 VIIIfx では `FSR.ftt` = 3 (*unimplemented_FPop*) のときに限り 8.2 である。
- *VA_watchpoint* は優先順位は 11 だが、SIMD ロード、SIMD ストア命令では優先順位 12 の例外とどちらが通知されるかは状況に依存する。詳細は Appendix F.5.1, *"Trap Conditions for SIMD Load/Store"* (page 180) を参照。
- *illegal_action* は SPARC64 VIIIfx 新規トラップで優先順位は 8.5 だが、優先順位 7 の *illegal_instruction* より優先して通知される場合がある。詳細は Chapter 7.6.1 を参照。
- TLB 多重ヒットが検出されると、TTE の内容に依存する例外は検出されない。詳細は Appendix F.5.2, *"Behavior on TLB Error"* (page 180) を参照。
- バスエラー、バスタイムアウトによる *data_access_error* は、優先順位 12 のトラップの中では一番優先順位が低い。詳細は Appendix F.5, *"Faults and Traps"* (page 178) を参照。

7.5 Trap Processing

JPS1 **Commonality** ではトラップ時のレジスタの変化を、いくつかの場合に分けて説明してあるが、SPARC64 VIIIfx で追加されたレジスタについてはどの場合でも同じなので、場合分けせず説明する。

トラップ時には、以下のレジスタの値が更新される。

- HPC-ACE 機能の使用状況を保存し、トラップハンドラの先頭命令からは HPC-ACE 機能を使わずに実行できるようにする。

```
TXAR[TL]    ← XAR
XAR         ← 0
```

XAR 対象命令で例外が発生した場合、0 になる前の値が TXAR[TL] にセーブされる。Tcc の場合、taken のとき TXAR[TL] には Tcc 実行前の XAR がセーブされる。

DONE, RETRY を実行したときのレジスタの変化は以下のようになる。

```
XAR         ← TXAR[TL]
TXAR[TL]    変化しない
```

Programming Note – エミュレーションルーチンが HPC-ACE 拡張命令をエミュレーションする場合は、XAR が消費されたように見えるように TXAR[TL] を調整してから DONE すること。

7.6 Exception and Interrupt Descriptions

7.6.1 Traps Defined by SPARC V9 As Mandatory

- *illegal_instruction* [tt = 010_{16}] (Precise) — *illegal_action* よりも優先順位が高いが、WRXAR, WRTXAR, WRPR %pstate では *illegal_action* が通知されることがある。詳細は各命令の説明を参照。

7.6.2 SPARC V9 Optional Traps That Are Mandatory in SPARC JPS1

- *fp_exception_other* [tt = 022_{16}] (Precise) — SPARC64 VIIIfx では、未実装の FPop を実行したときに通知される例外 (FSR.ftt = 3, *unimplemented_FPop*) のみ、優先順位が 8.5 となる。

7.6.4 SPARC V9 Implementation-Dependent, Optional Traps That Are Mandatory in SPARC JPS1

SPARC64 VIIIfx は SPARC V9 では実装依存、JPS1 で必須とされている 6 つのトラップをすべて実装する。

7.6.5 SPARC JPS1 Implementation-Dependent Traps

SPARC64 VIIIfx 固有のトラップは以下の通りである (impl. dep. #35)。

- *async_data_error* [tt = 040_{16}] (Preemptive or disrupting) (impl. dep. #218) — SPARC64 VIIIfx では緊急エラー (Urgent Error) を報告するために使われる。詳細は Appendix P.4, "*緊急エラー*" (page 274) を参照。
- *illegal_action* [tt = 073_{16}] (Precise) — 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。WRXAR, WRTXAR, WRPR %pstate 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を参照。
- *SIMD_load_across_pages* [tt = 077_{16}] (Precise) — SIMD ロードが複数ページにまたがり、extended 側が TLB ミスした場合に通知される。
  システムソフトウェアはこの例外が通知された場合、エミュレーションにより basic, extended のロードを個別に処理すること。

Note – *SIMD_load_across_pages* で TLB の更新処理を行うと、basic と extended が交互に TLB から落ちる無限ループに陥る可能性がある。

F.CHAPTER **8**

Memory Models

SPARC V9 アーキテクチャ仕様は、SPARC V9 システム上で動作するソフトウェアが観測可能な振る舞いを記述したモデルである。したがって、メモリアクセスの方法も、それが JPS1 **Commonality** の Chapter 8 および Appendix D で定義されるメモリモデルに準拠している限りはどのように実装されていてもよい。

SPARC V9 では Total Store Order (TSO), Partial Store Order (PSO), Relaxed Memory Order (RMO) の 3 種類のメモリモデルを定義している。SPARC V9 に準拠するプロセッサは、SPARC V8 との互換性を保証するため、TSO またはそれ以上に制約の強いメモリモデル (たとえば逐次一貫性 Sequential Consistency モデルなど) を実装する必要がある。

これに対し、PSO, RMO モデルをサポートするかどうかは SPARC V9 システムでは実装依存である。SPARC64 VIIIfx はどのメモリモデルでも、その定義通りに動作する。

8.1 Overview

Note – 以下の文で“ハードウェアメモリモデル”は“SPARC V9メモリモデル”との対比で使われる。SPARC V9 メモリモデルは PSTATE.MM で選択されるメモリモデルを意味する。

SPARC64 VIIIfx は SPARC V9 で定義された 3 種類のメモリモデルのどの定義にも準拠するメモリモデル一種類だけを実装する (impl. dep. #113)。

- **Total Store Order** — すべてのロードは、先行するロードとの順序を守り、すべてのストアは、先行するロードとストアとの順序を守る。この動作は SPARC V9 のメモリモデル TSO, PSO, RMO の仕様に準拠する。PSTATE.MM で PSO または RMO が指示された場合でも、SPARC64 VIIIfx はこのメモリモデルで動作する。定義から、PSO または RMO で動作するよう書かれたプログラムは TSO でも動作するので、SPARC64 VIIIfx のこの動作は PSO, RMO の弱い制約のメリットを享受することはできないが、安全である。

8.4 SPARC V9 Memory Model

8.4.5 Mode Control

SPARC64 VIIIfx は PSTATE.MM のすべての設定において TSO で動作する。PSTATE.MM に 11_2 をセットした場合も TSO で動作する (impl. dep. #119) が、将来のバージョンの SPARC64 VIIIfx では 11_2 を別のメモリモデルに割り当てるかもしれないので、使うべきではない。

8.4.7 Synchronizing Instruction and Data Memory

SPARC64 VIIIfx ではハードウェアがすべてのキャッシュのデータの同一性を保証する。データキャッシュの書き込みにより、対応するデータが命令キャッシュに載っていればそれを無効化し、命令フェッチによる命令キャッシュの読み出しでは、データキャッシュに変更済みデータがあればそれが充当される。

SPARC64 VIIIfx のこの仕様は FLUSH 命令が不要であることを意味しない。FLUSH はキャッシュとパイプライン内のデータの同一性を保証したい場合には実行する必要がある。

SPARC64 VIIIfx はマルチプロセッサをサポートしないので、マルチプロセッサでの FLUSH 命令のレイテンシは定義されない。CPU チップ内のコア間での FLUSH 命令のレイテンシは、CPU 内部の状態に依存するが、最短で 30 サイクルである (impl. dep. #122)。

F.APPENDIX **A**

Instruction Definitions

この章では、JPS1 **Commonality** で実装依存仕様が定義された命令の SPARC64 VIIIfx での仕様と、SPARC64 VIIIfx 独自に拡張された命令について記述する。JPS1 **Commonality** 仕様に準拠している命令については記述していないので、JPS1 **Commonality** を参照のこと。なお、セクション番号は JPS1 **Commonality** に一致させてある。

SPARC64 VIIIfx 固有に拡張された命令は、Section A.24 のサブセクションと、Section A.72 以降に記載されている。それ以外のセクションは、JPS1 **Commonality** の仕様通りである。

実装依存仕様を記述している命令は、必要な情報のみを記述している。
SPARC64 VIIIfx 固有命令の仕様には以下の記述が含まれている。

1. オペコード表。そのオペコードに固有のフィールド値と、HPC-ACE 拡張機能に対応しているかどうかの情報を含む。
2. 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 VIIIfx 用のプログラムでは 0 が入っているべきである。SPARC V9 のプロセッサでは、*reserved* が 0 でない命令を実行したときの挙動は未定義である。SPARC64 VIIIfx の動作は Section 1.2, *"Fonts and Notational Conventions"* (page 1) を参照。
3. アセンブリ言語での表記法。詳細は Appendix G を参照。
4. 命令の詳細、制限事項、例外発生条件などの説明。
5. その命令を実行したときに起こりうる例外の一覧を、優先順位に従って記載している。

   ただし以下の例外は記載していない。

   a. すべての命令で起こり得る *instruction_access_error, instruction_access_exception, fast_instruction_access_MMU_miss, async_data_error, ECC_error*, およびインタラプト。

   b. 実装されていない命令で起こる *illegal_instruction* (浮動小数点演算では *fp_exception_other* で `ftt` = *unimplemented_FPop*)。

c. IIU_INST_TRAP (ASI = 60_{16}, VA = 0) を指定すればどの命令でも発生する *illegal_instruction*。

なお、*illegal_action* 例外の発生条件を記述する際は、XAR のフィールド名には f_, s_ をつけず表記する。

以下の例外は SPARC64 VIIIfx では起こらない。

- *instruction_access_MMU_miss*
- *data_access_MMU_miss*
- *data_access_protection*
- *unimplemented_LDD*
- *unimplemented_STD*
- *LDQF_mem_address_not_aligned*
- *STQF_mem_address_not_aligned*
- *internal_processor_error*
- *fp_exception_other* (ftt = *invalid_fp_register*)

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。

JPS1 **Commonality** の命令と SPARC64 VIIIfx で拡張された命令の一覧を TABLE A-2 に示す。この表、およびこの章と Appendix E 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。.

TABLE A-1 オペコードの右肩の文字の意味

文字	意味
D	非推奨命令
P	特権命令
P_{ASI}	ASI の 7 ビット目が 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

TABLE A-2 および各命令のオペコード表にある HPC-ACE 拡張の列は、その命令で SPARC64 VIIIfx のどの拡張機能が有効かを示している。

- **Inst.** JPS1 **Commonality** では定義されていない、SPARC64 VIIIfx 固有の命令。
- **Regs.** XAR 対象命令。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。
  この列に ☆ がついている命令は、rd に指定できるのは basic FPR のみ。
- **SIMD** SIMD 演算可能。
  この列に ✝ がついている命令は、4 倍精度命令では SIMD 演算ができない。

この 3 つの欄のどれにも ✓ がついていない命令は XAR 非対象命令である。XAR 非対象命令の詳細は *"XAR 動作"* (page 31) を参照。

TABLE A-2 SPARC64 VIIIfx の命令セット *(1 of 7)*

命令	処理内容	HPC-ACE 拡張 Inst.	Regs.	SIMD	ページ
ADD (ADDcc)	Add (and modify condition codes)		✓		—
ADDC (ADDCcc)	Add with carry (and modify condition codes)		✓		—
ALIGNADDRESS{_LITTLE}	Calculate address for misaligned data				—
AND (ANDcc)	And (and modify condition codes)		✓		—
ANDN (ANDNcc)	And not (and modify condition codes)		✓		—
ARRAY(8,16,32)	3-D array addressing instructions				—
BPcc	Branch on integer condition codes with prediction				—
Bicc[D]	Branch on integer condition codes				—
BMASK	Set the GSR.MASK field				—
BPr	Branch on contents of integer register with prediction				—
BSHUFFLE	Permute bytes as specified by GSR.MASK				—
CALL	Call and link				68
CASA[PASI]	Compare and swap word in alternate space		✓		—
CASXA[PASI]	Compare and swap doubleword in alternate space		✓		—
DONE[P]	Return from trap				—
EDGE(8,16,32){L}	Edge handling instructions				—
FABS(s,d,q)	Floating-point absolute value		✓	✝	—
FADD(s,d,q)	Floating-point add		✓	✝	—
FALIGNDATA	Perform data alignment for misaligned data				—
FAND{S}	Logical AND operation		✓	✓	—
FANDNOT(1,2){S}	Logical AND operation with one inverted source		✓	✓	—
FBfcc[D]	Branch on floating-point condition codes				—
FBPfcc	Branch on floating-point condition codes with prediction				—
FCMP(s,d,q)	Floating-point compare		✓		—
FCMPE(s,d,q)	Floating-point compare (exception if unordered)		✓		—
FCMP(GT,LE,NE,EQ)(16,32)	Pixel compare operations				—
FCMP(EQ,NE)(s,d)	Floating-point conditional compare to register	✓	✓	✓	114
FCMP(GT,LT,EQ,NE,GE,LE)E(s,d)	Floating-point conditional compare (exception if unordered)	✓	✓	✓	114
FDIV(s,d,q)	Floating-point divide		☆		—
FdMULq	Floating-point multiply double to quad		✓		—
FEXPAND	Pixel expansion				—
FiTO(s,d,q)	Convert integer to floating-point		✓	✝	—
FLUSH	Flush instruction memory		✓		—
FLUSHW	Flush register windows				—
FMADD(s,d)	Foating-point Multiply-and-Add	✓	✓	✓	70

TABLE A-2 SPARC64 VIIIfx の命令セット *(2 of 7)*

		HPC-ACE 拡張			
命令	**処理内容**	**Inst.**	**Regs.**	**SIMD**	**ページ**
FMAX(s,d)	Floating-point maximum	✓	✓	✓	116
FMIN(s,d)	Floating-point minimum	✓	✓	✓	116
FMSUB(s,d)	Foating-point Multiply-and-Subtract	✓	✓	✓	70
FMOV(s,d,q)	Floating-point move		✓	†	—
FMOV(s,d,q)cc	Move floating-point register if condition is satisfied				—
FMOV(s,d,q)r	Move f-p reg. if integer reg. contents satisfy condition				—
FMUL(s,d,q)	Floating-point multiply		✓	†	—
FMUL8x16	8x16 partitioned product				—
FMUL8x16(AU,AL)	8x16 upper/lower α partitioned product				—
FMUL8(SU,UL)x16	8x16 upper/lower partitioned product				—
FMULD8(SU,UL)x16	8x16 upper/lower partitioned product				—
FNAND{S}	Logical NAND operation		✓	✓	—
FNEG(s,d,q)	Floating-point negate		✓	†	—
FNMADD(s,d)	Foating-point Multiply-and-Add and negate	✓	✓	✓	70
FNMSUB(s,d)	Foating-point Multiply-and-Subtract and negate	✓	✓	✓	70
FNOR{S}	Logical NOR operation		✓	✓	—
FNOT(1,2){S}	Copy negated source		✓	✓	—
FPACK(16,32, FIX)	Pixel packing				—
FPADD(16,32){S}	Pixel add (single) 16- or 32-bit				—
FPMADDX{HI}	Integer Multiply-and-Add	✓	✓	✓	78
FPMERGE	Pixel merge				—
FRCPA(s,d)	Floating-point reciprocal approximation	✓	✓	✓	118
FRSQRTA(s,d)	Floating-point reciprocal square root approximation	✓	✓	✓	118
FONE{S}	One fill		✓	✓	—
FOR{S}	Logical OR operation		✓	✓	—
FORNOT(1,2){S}	Logical OR operation with one inverted source		✓	✓	—
FPSUB(16,32){S}	Pixel subtract (single) 16- or 32-bit				—
FsMULd	Floating-point multiply single to double		✓	✓	—
FSQRT(s,d,q)	Floating-point square root		☆		—
FSRC(1,2){S}	Copy source		✓	✓	—
FSELMOV(s,d)	Move selected floating-point register	✓	✓	✓	122
F(s,d,q)TOi	Convert floating point to integer		✓	†	—
F(s,d,q)TO(s,d,q)	Convert between floating-point formats		✓	†	—
F(s,d,q)TOx	Convert floating point to 64-bit integer		✓	†	—
FSUB(s,d,q)	Floating-point subtract		✓	†	—
FTRIMADDd	Floating-point trigonometric function	✓	✓	✓	123

TABLE A-2 SPARC64 VIIIfx の命令セット *(3 of 7)*

命令	処理内容	HPC-ACE 拡張 Inst.	Regs.	SIMD	ページ
FTRIS(MUL,SEL)d	Floating-point trigonometric functions	✓	✓	✓	123
FXNOR{S}	Logical XNOR operation		✓	✓	—
FXOR{S}	Logical XOR operation		✓	✓	—
FxTO(s,d,q)	Convert 64-bit integer to floating-point		✓	✝	—
FZERO{S}	Zero fill		✓	✓	—
ILLTRAP	Illegal instruction				—
JMPL	Jump and link				80
LDD^{D}	Load integer doubleword		✓		—
LDDA$^{D,\ P_{ASI}}$	Load integer doubleword from alternate space		✓		—
LDDA ASI_NUCLEUS_QUAD*	Load integer quadword, atomic		✓		—
LDDA ASI_QUAD_PHYS*	Load integer quadword, atomic (physical address)		✓		88
LDDF	Load double floating-point		✓	✓	81
LDDFA$^{P_{ASI}}$	Load double floating-point from alternate space		✓	✓	85
LDDFA ASI_BLK*	Block loads		✓		66
LDDFA ASI_FL*	Short floating point loads				—
LDF	Load floating-point		✓	✓	81
LDFA$^{P_{ASI}}$	Load floating-point from alternate space		✓	✓	85
LDFSR^{D}	Load floating-point state register lower		✓		81
LDQF	Load quad floating-point		✓		81
LDQFA$^{P_{ASI}}$	Load quad floating-point from alternate space		✓		85
LDSB	Load signed byte		✓		—
LDSBA$^{P_{ASI}}$	Load signed byte from alternate space		✓		—
LDSH	Load signed halfword		✓		—
LDSHA$^{P_{ASI}}$	Load signed halfword from alternate space		✓		—
LDSTUB	Load-store unsigned byte		✓		—
LDSTUBA$^{P_{ASI}}$	Load-store unsigned byte in alternate space		✓		—
LDSW	Load signed word		✓		—
LDSWA$^{P_{ASI}}$	Load signed word from alternate space		✓		—
LDUB	Load unsigned byte		✓		—
LDUBA$^{P_{ASI}}$	Load unsigned byte from alternate space		✓		—
LDUH	Load unsigned halfword		✓		—
LDUHA$^{P_{ASI}}$	Load unsigned halfword from alternate space		✓		—
LDUW	Load unsigned word		✓		—
LDUWA$^{P_{ASI}}$	Load unsigned word from alternate space		✓		—
LDX	Load extended		✓		—
LDXA$^{P_{ASI}}$	Load extended from alternate space		✓		—

TABLE A-2 SPARC64 VIIIfx の命令セット *(4 of 7)*

命令	処理内容	HPC-ACE 拡張 Inst.	HPC-ACE 拡張 Regs.	HPC-ACE 拡張 SIMD	ページ
LDXFSR	Load floating-point state register		✓		81
MEMBAR	Memory barrier				90
MOVcc	Move integer register if condition is satisfied		✓		—
MOVr	Move integer register on contents of integer register		✓		—
MULScc[D]	Multiply step (and modify condition codes)		✓		—
MULX	Multiply 64-bit integers		✓		—
NOP	No operation		✓		92
OR (ORcc)	Inclusive-or (and modify condition codes)		✓		—
ORN (ORNcc)	Inclusive-or not (and modify condition codes)		✓		—
PDIST	Pixel component distance				—
POPC	Population count		✓		94
PREFETCH	Prefetch data		✓		95
PREFETCHA[P$_{ASI}$]	Prefetch data from alternate space		✓		95
RDASI	Read ASI register		✓		97
RDASR[P$_{ASR}$]	Read ancillary state register		✓		97
RDCCR	Read condition codes register		✓		97
RDDCR[P]	Read dispatch control register		✓		97
RDFPRS	Read floating-point registers state register		✓		97
RDGSR	Read graphic status register		✓		97
RDPC	Read program counter		✓		97
RDPCR[P$_{PCR}$]	Read performance control register		✓		97
RDPIC[P$_{PIC}$]	Read performance instrumentation counters		✓		97
RDPR[P]	Read privileged register		✓		—
RDSOFTINT[P]	Read per-processor soft interrupt register		✓		97
RDSTICK[P$_{NPT}$]	Read system TICK register		✓		97
RDSTICK_CMPR[P]	Read system TICK compare register		✓		97
RDTICK[P$_{NPT}$]	Read TICK register		✓		97
RDTICK_CMPR[P]	Read TICK compare register		✓		97
RDTXAR[P]	Read TXAR register	✓	✓		97
RDXASR	Read XASR register	✓	✓		97
RDY[D]	Read Y register		✓		97
RESTORE	Restore caller's window		✓		—
RESTORED[P]	Window has been restored				—
RETRY[P]	Return from trap and retry				—
RETURN	Return				—
SAVE	Save caller's window		✓		—

TABLE A-2 SPARC64 VIIIfx の命令セット *(5 of 7)*

命令	処理内容	HPC-ACE 拡張 Inst.	Regs.	SIMD	ページ
SAVED[P]	Window has been saved				—
SDIV[D] (SDIVcc[D])	32-bit signed integer divide (and modify condition codes)		✓		—
SDIVX	64-bit signed integer divide		✓		—
SETHI	Set high 22 bits of low word of integer register		✓		—
SHUTDOWN	Shut down the processor				99
SIAM	Set Interval Arithmetic Mode				—
SIR	Software-initiated reset				—
SLEEP	Sleep this thread				77
SLL	Shift left logical		✓		—
SLLX	Shift left logical, extended		✓		—
SMUL[D] (SMULcc[D])	Signed integer multiply (and modify condition codes)		✓		—
SRA	Shift right arithmetic		✓		—
SRAX	Shift right arithmetic, extended		✓		—
SRL	Shift right logical		✓		—
SRLX	Shift right logical, extended		✓		—
STB	Store byte		✓		—
STBA[P, ASI]	Store byte into alternate space		✓		—
STBAR[D]	Store barrier				113
STD[D]	Store doubleword		✓		—
STDA[D, P, ASI]	Store doubleword into alternate space		✓		—
ST(D,DF,X)A ASI_XFILL*	Cache line fill	✓	✓		133
STDF	Store double floating-point		✓	✓	100
STDFA[P, ASI]	Store double floating-point into alternate space		✓	✓	104
STDFA ASI_BLK*	Block stores		✓		66
STDFA ASI_FL*	Short floating point stores				—
STDFA ASI_PST*	Partial Store instructions				93
STDFR	Store double floating-point on register's condition	✓	✓	✓	128
STF	Store floating-point		✓	✓	100
STFA[P, ASI]	Store floating-point into alternate space		✓	✓	104
STFR	Store floating-point on register condition	✓	✓	✓	128
STFSR[D]	Store floating-point state register		✓		100
STH	Store halfword		✓		—
STHA[P, ASI]	Store halfword into alternate space		✓		—
STQF	Store quad floating-point		✓		100
STQFA[P, ASI]	Store quad floating-point into alternate space		✓		104
STW	Store word		✓		—

TABLE A-2 SPARC64 VIIIfx の命令セット *(6 of 7)*

命令	処理内容	HPC-ACE 拡張 Inst.	Regs.	SIMD	ページ
STWA[P, ASI]	Store word into alternate space		✓		—
STX	Store extended		✓		—
STXA[P, ASI]	Store extended into alternate space		✓		—
STXFSR	Store extended floating-point state register		✓		100
SUB (SUBcc)	Subtract (and modify condition codes)		✓		—
SUBC (SUBCcc)	Subtract with carry (and modify condition codes)		✓		—
SUSPEND[P]	Suspend this thread				76
SWAP[D]	Swap integer register with memory		✓		—
SWAPA[D, P, ASI]	Swap integer register with memory in alternate space		✓		—
SXAR(1,2)	Set XAR	✓			131
TADDcc (TADDccTV[D])	Tagged add and modify condition codes (trap on overflow)		✓		—
Tcc	Trap on integer condition codes				107
TSUBcc (TSUBccTV[D])	Tagged subtract and modify condition codes (trap on overflow)		✓		—
UDIV[D] (UDIVcc[D])	Unsigned integer divide (and modify condition codes)		✓		—
UDIVX	64-bit unsigned integer divide		✓		—
UMUL[D] (UMULcc[D])	Unsigned integer multiply (and modify condition codes)		✓		—
WRASI	Write ASI register		✓		111
WRASR[P, ASR]	Write ancillary state register		✓		111
WRCCR	Write condition codes register		✓		111
WRDCR[P]	Write dispatch control register		✓		111
WRFPRS	Write floating-point registers state register		✓		111
WRGSR	Write graphic status register		✓		111
WRPCR[P, PCR]	Write performance control register		✓		111
WRPIC[P, PIC]	Write performance instrumentation counters register		✓		111
WRPR[P]	Write privileged register		✓		108
WRSOFTINT[P]	Write per-processor soft interrupt register		✓		111
WRSOFTINT_CLR[P]	Clear bits of per-processor soft interrupt register		✓		111
WRSOFTINT_SET[P]	Set bits of per-processor soft interrupt register		✓		111
WRTICK_CMPR[P]	Write TICK compare register		✓		111
WRSTICK[P]	Write System TICK register		✓		111
WRSTICK_CMPR[P]	Write System TICK compare register		✓		111

TABLE A-2 SPARC64 VIIIfx の命令セット *(7 of 7)*

命令	処理内容	HPC-ACE 拡張 Inst.	Regs.	SIMD	ページ
WRTXAR[P]	Write TXAR register	✓	✓		111
WRXAR	Write XAR register	✓	✓		111
WRXASR	Write XASR register	✓	✓		111
WRY[D]	Write Y register		✓		111
XNOR (XNORcc)	Exclusive-nor (and modify condition codes)		✓		—
XOR (XORcc)	Exclusive-or (and modify condition codes)		✓		—

A.4 Block Load and Store Instructions (VIS I)

Deprecated – block load/store は SPARC64 VIIIfx では過去の互換性のために定義されており、新規プログラムでの使用は推奨されない。高速なメモリコピーには Section A.79, *"Cache Line Fill with Undetermined Values"* を用いること。

SPARC64 VIIIfx の block load/store 仕様は、SPARC64 V…SPARC64 VII とは異なる。新仕様は従来の仕様よりも制約が強いが、一部非互換なところがある。以下に SPARC64 VIIIfx の block load/store の動作と、従来仕様との相違点を示す。

1. block load/store はアトミックなメモリ操作ではなく、内部的に 8 バイトの load/store に分割して実行される。block load/store の前後にある `MEMBAR` またはアトミック命令との間でオーダリングを遵守する。

2. block load/store 命令は TSO を遵守する。前後の load/store/atomic 命令、および block load/store 内の 8 バイトに分割された各 load/store 間でも TSO に準拠して動作する。

Compatibility Note – 従来仕様では、命令間では SPARC V9 メモリモデルに準拠せず、命令内の 8 バイト単位では RMO で動作することになっていた。

3. レジスタ間のアクセス順序は、他の命令と同様に保証される。つまり、block load/store と他の命令の間でのレジスタの read-after-write, write-after-write はプログラム順序になる。

4. block load/store のキャッシュ上の動作は、通常の load/store と同じである。block load は L1 キャッシュ上にデータがあれば L1 キャッシュから読み、L1 キャッシュ上になければメモリから L1 キャッシュに読み込まれる。block store は L1 キャッシュ上にデータがあれば L1 キャッシュに書き、L1 キャッシュ上になければ L1 キャッシュに読み込んだ上で当該データを更新する。

Compatibility Note – block load/store 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、block load はキャッシュにデータがあれば読み、キャッシュにない場合の動作は未定義。block store は、ダーティデータを持つキャッシュがあれば、そのキャッシュに書き込むとともにそれより上位 (パイプラインに近い) キャッシュからは消し、ダーティーデータを持つキャッシュがないかまたはキャッシュに乗っていないときは、メモリに書く仕様だった。

5. SPARC64 VIIIfxでは block store と block store with commit は完全に同じ動作をする。

Compatibility Note – block store with commit 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き込む仕様だった。

6. TTE.E = 0 であるページに対する block load/store は、64 バイト中の任意の 8 バイトの処理中に *fast_data_access_MMU_miss* 例外が通知されることがある。block load の途中で例外が通知されたときは、レジスタには block load 実行前、実行後どちらの値が見えることもありうる。block store の途中で例外が通知されたときは、メモリの状態は block store 実行前のままである。

Programming Note – ノンキャッシャブル空間の一部に、block store は正常に終了したように見えるが実際に書き込みが行われない領域がある。詳細はシステム仕様書を参照。

Note – JPS1 **Commonality** で定義されている通り、block load/store 命令では、*LDDF_mem_address_not_aligned*, *STDF_mem_address_not_aligned* 例外は通知されない (Appendix L.3.3, *"ASI と命令の組み合わせと例外 "* (page 221) も参照)。LDDFA 命令で ASI_BLK_COMMIT_{P,S} を指定した場合は、block load/store 命令ではないので、4 バイト境界にアクセスすると *LDDF_mem_address_not_aligned* 例外が通知される (*"Block Load and Store ASIs"* (page 220) も参照)。

Exceptions

illegal_instruction (misaligned rd)
fp_disabled
illegal_action (XAR.v = 1 and (XAR.urs1 > 1 or
(i = 0 and XAR.urs2 > 1) or
(i = 1 and XAR.urs2 ≠ 0) or
XAR.urs3<2> ≠ 0);
XAR.v = 1 and XAR.simd = 1)
mem_address_not_aligned (see *"Block Load and Store ASIs"* (page 220))
LDDF_mem_address_not_aligned (see *"Block Load and Store ASIs"* (page 220))
VA_watchpoint (only detected on the first 8 bytes of a transfer)
fast_data_access_MMU_miss
data_access_exception (see *"Block Load and Store ASIs"* (page 220))
fast_data_access_protection
PA_watchpoint (only detected on the first 8 bytes of a transfer)
data_access_error

A.9 Call and Link

PSTATE.AM がセットされているとき、r[15] には PC の上位 32 ビットを 0 にした値が入る (impl. dep. #125)。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。

Exceptions *illegal_action* (XAR.v = 1)

A.24 Implementation-Dependent Instructions

オペコード	op3	動作
IMPDEP1	11 0110	実装依存命令 1
IMPDEP2	11 0111	実装依存命令 2

IMPDEP1 と IMPDEP2 は実装依存命令である。実装依存の意味は、命令の動作、bit<29:25>, bit<18:0> の解釈、通知される例外の種類などである。

SPARC64 VIIIfx は IMPDEP1 に VIS, SUSPEND, SLEEP, FCMP*cond*{d,s}, FMIN{d,s}, FMAX{d,s}, FRCPA{d,s}, FRSQRTA{d,s}, FTRISSELd, FTRISMULd 命令を実装している (impl. dep. #106)。IMPDEP2A には FPMADDX, FPMADDXHI, FTRIMADDd, FSELMOV{d,s} を、IMPDEP2B には浮動小数点積和演算命令を実装している (impl. dep. #106)。

SPARC V9 命令の実装依存な命令の拡張方法については、JPS1 **Commonality** の I.1.2, "*Implementation-Dependent and Reserved Opcodes*" を参照。

Compatibility Note – SPARC V8 ではこの命令は CPop*n* 命令だった。

なお、SPARC64 VIIIfx で新規追加された IMPDEP1, IMPDEP2 命令については、Section A.24 のサブセクションではなく、他の新規追加命令と同様 Section A.71 より後に配置してある。

Exceptions 実装に依存する

A.24.1 Floating-Point Multiply-Add/Subtract

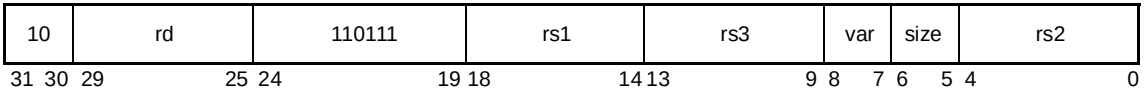
SPARC64 VIIIfx は `IMPDEP2B` に浮動小数点積和演算 (Floating-Point Multiply and Add: FMA) を実装している。FMA は HPC-ACE で SIMD 拡張されているが、他の SIMD 拡張よりも自由度が高いものとなっている。この節ではまず non-SIMD の動作を説明し、次に HPC-ACE の拡張を説明する。

HPC-ACE Ext. Regs.	SIMD	オペコード	Var	Size[1] [2]	処理
✓	✓	`FMADDs`	00	01	単精度の乗算と加算
✓	✓	`FMADDd`	00	10	倍精度の乗算と加算
✓	✓	`FMSUBs`	01	01	単精度の乗算と減算
✓	✓	`FMSUBd`	01	10	倍精度の乗算と減算
✓	✓	`FNMSUBs`	10	01	単精度の乗算と減算後符号反転
✓	✓	`FNMSUBd`	10	10	倍精度の乗算と減算後符号反転
✓	✓	`FNMADDs`	11	01	単精度の乗算と加算後符号反転
✓	✓	`FNMADDd`	11	10	倍精度の乗算と加算後符号反転

1.size = 00 については Section A.24.4, Section A.75, Section A.76 を参照。

2.size = 11 は 4 倍精度に予約されているが、一部は Section A.75, *"Move Selected Floating-Point Register on Floating-Point Register's Condition"* で使われている。

Format (5)



処理	演算
乗算と加算	$rd \leftarrow rs1 \times rs2 + rs3$
乗算と減算	$rd \leftarrow rs1 \times rs2 - rs3$
乗算と減算後符号反転	$rd \leftarrow -rs1 \times rs2 + rs3$
乗算と加算後符号反転	$rd \leftarrow -rs1 \times rs2 - rs3$

アセンブリ言語表記	
fmadds	$freg_{rs1}$, $freg_{rs2}$, $freg_{rs3}$, $freg_{rd}$
fmaddd	$freg_{rs1}$, $freg_{rs2}$, $freg_{rs3}$, $freg_{rd}$
fmsubs	$freg_{rs1}$, $freg_{rs2}$, $freg_{rs3}$, $freg_{rd}$
fmsubd	$freg_{rs1}$, $freg_{rs2}$, $freg_{rs3}$, $freg_{rd}$
fnmadds	$freg_{rs1}$, $freg_{rs2}$, $freg_{rs3}$, $freg_{rd}$
fnmaddd	$freg_{rs1}$, $freg_{rs2}$, $freg_{rs3}$, $freg_{rd}$
fnmsubs	$freg_{rs1}$, $freg_{rs2}$, $freg_{rs3}$, $freg_{rd}$
fnmsubd	$freg_{rs1}$, $freg_{rs2}$, $freg_{rs3}$, $freg_{rd}$

Description

FMADD は、rs1 で指定される浮動小数点レジスタに rs2 で指定される浮動小数点レジスタを乗じ、その結果に rs3 で指定される浮動小数点レジスタの値を加え、rd で指定される浮動小数点レジスタに格納する。

FMSUB は、rs1 で指定される浮動小数点レジスタに rs2 で指定される浮動小数点レジスタを乗じ、その結果から rs3 で指定される浮動小数点レジスタの値を引き、rd で指定される浮動小数点レジスタに格納する。

FNMADD は、rs1 で指定される浮動小数点レジスタに rs2 で指定される浮動小数点レジスタを乗じ、*その結果の符号を反転させた値*から rs3 で指定される浮動小数点レジスタの値を*引き*、rd で指定される浮動小数点レジスタに格納する。

FNMSUB は、rs1 で指定される浮動小数点レジスタに rs2 で指定される浮動小数点レジスタを乗じ、*その結果の符号を反転させた値*に rs3 で指定される浮動小数点レジスタの値を*加え*、rd で指定される浮動小数点レジスタに格納する。

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、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。

Programming Note – SPARC64 V では浮動小数点積和演算は、乗算と加算を独立して処理していた。つまり、乗算の結果は、単独の乗算命令と同じように丸められ、その後に加減算が行われ、その結果がまた丸められていた。したがって、丸め処理による誤差が入り込む機会が 2 回あった。
また、FNMADD と FNMSUB の rs1 または rs2 が NaN のときの動作も SPARC64 V と SPARC64 VIIIfx で異なる。SPARC64 VIIIfx は結果としてどちらかの NaN をそのまま出力するが、SPARC64 V は符号を反転させた NaN を結果として出力する。

TABLE A-3 に SPARC64 VIIIfx 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。乗算部分でトラップが生じる無効処理例外 (NV) が生じるか、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 IEEE754 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 IEEE754 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 と同じある。

TABLE A-3 浮動小数点積和演算命令の IEEE754 例外発生条件

FMUL	IEEE754 トラップ (NV または NX のみ)	トラップを生じない	トラップを生じない
FADD	—	IEEE754 トラップ	トラップを生じない
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 (トラップしない) 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

aexc に表示される値は各種条件に依存する。詳細を TABLE A-4, TABLE A-5 にまとめた。この表で uf, of, nv, nx はそれぞれ IEEE754 で定義された例外 (uf: アンダーフロー , of: オーバーフロー , nv: 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。

TABLE A-4 トラップしない場合の aexc の値 (FSR.NS = 0 の時)

		FADD			
		none	nx	of nx	nv
FMUL	none	none	nx	of nx	nv
	nv	nv	—	—	nv

TABLE A-5 トラップしない場合の aexc の値 (FSR.NS = 1 の時)

		FADD				
		none	nx	of nx	uf nx	nv
FMUL	none	none	nx	of nx	uf nx	nv
	nv	nv	—	—	—	nv
	nx	nx	nx	of nx	uf nx	nv nx

表中 “—” のケースは起こりえない。

Programming Note – 浮動小数点積和演算は SPARC V9 の IMPDEP2 に定義されている。この命令は SPARC64 VIIIfx 固有の命令であり、他の SPARC V9 プロセッサで実行されるかもしれないプログラムでは使うことができない。

FMADD の SIMD 拡張

SPARC64 VIIIfx の SIMD 拡張では、basic 側の演算と extended 側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 basic 側のレジスタ f[0] – f[254] なので、rs1, rs2, rs3, rd の最上位ビットは必ず 0 になる (page 22)。これに対し、FMADD の SIMD 拡張では、制限つきではあるが basic と extended の間での演算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。

FMADD の SIMD 拡張では、rs1, rs2 については basic 側、extended 側どちらにも全てのレジスタ f[2n] (n=0…255) を指定することができる。basic 側演算に extended 側のレジスタを指定すると、extended 側にはペアになる basic 側のレジスタが使われる。つまり、basic 側では f[2n] (n=0…255) が、extended 側では f[(2n+256) mod 512] (n=0…255) が使われる。

これに対し rs3 と rd は他の SIMD 拡張と同じで、basic 側演算では f[0] – f[254]、extended 側演算では f[256] – f[510] 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、extended 側演算の rs1 には basic 側と同じものが使われる。urd<2> = 1 のとき、extended 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>　basic 側演算の rs1<8>, extended 側演算の ¬rs1<8>
- XAR.urs2<2>　basic 側演算の rs2<8>, extended 側演算の ¬rs2<8>
- XAR.urs3<2>　extended 側演算に rs1<8> と ¬rs1<8> のどちらを使うか
- XAR.urd<2>　extended 側演算の積演算の符号を反転させるかどうか

なお、上の rs1<8> は、倍精度レジスタのビットデコード後のビットを意味する。詳細は FIGURE 5-1 (page 21) 参照。

	frs1: urs1<2:0>, rs1<5:0>	$frs1_i$: ¬urs1<2>, urs1<1:0>, rs1<5:0>
	frs2: urs2<2:0>, rs2<5:0>	$frs2_i$: ¬urs2<2>, urs2<1:0>, rs2<5:0>
	$frs3_b$: 1'b0, urs3<1:0>, rs3<5:0>	$frs3_e$: 1'b1, urs3<1:0>, rs3<5:0>
	frd_b: 1'b0, urd<1:0>, rd<5:0>	$frs1_i$: 1'b1, urd<1:0>, rd<5:0>
	c: urs3<2>	
	n: urd<2>	

命令	basic 側演算	extend 側演算
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	$frd_b \leftarrow -frs1 \times frs2 - frs3_b$	$frd_e \leftarrow -(-1)^n \times (c\ ?\ frs1 : frs1_i) \times frs2_i - frs3_e$

例 1: 複素数の乗算

$(a_1 + ib_1)\ (a_2 + ib_2) = (a_1a_2 - b_1b_2) + i(a_1b_2 + a_2b_1)$

```
        /*
         * X: 入力の複素数のアドレス
         * Y: 入力の複素数のアドレス
         * Z: 出力の複素数のアドレス
         */

        /* 前準備 */
        sxar2
        ldd,s           [X], %f0  /* %f0: a1, %f256: b1 */
        ldd,s           [Y], %f2  /* %f2: a2, %f258: b2 */
        sxar1
        fzero,s         %f4       /* 結果レジスタをクリアしておく */

        /* 計算 */
        sxar2
        fnmaddd,snc     %f256, %f258, %f4, %f4
                             /* %f4    := -%f256 * %f258 - %f4 */
                             /* %f260  := %f256 * %f2 - %f260 */
        fmaddd,sc       %f0, %f2, %f4, %f4
                             /* %f4    := %f0 * %f2 + %f4 */
                             /* %f260  := %f0 * %f258 + %f260 */
        /* 結果を格納する */
        sxar1
        std,s           %f4, [Z]
```

例 2: 2x2 行列の乗算

```
/*
 * A: 入力の行列のアドレス : a11, a12, a21, a22
 * B: 入力の行列のアドレス : b11, b12, b21, b22
 * C: 出力の行列のアドレス : c11, c12, c21, c22
 */

/* 前準備 */
sxar2
ldd,s          [A], %f0  /* %f0: a11, %f256: a12 */
ldd,s          [A+16], %f2/* %f2: a21, %f258: a22 */
sxar2
ldd,s          [B], %f4  /* %f4: b11, %f260: b12 */
ldd,s          [B+16], %f6/* %f6: b21, %f262: b22 */
sxar2
fzero,s        %f8       /* %f8: c11, %f264: c12 */
fzero,s        %f10      /* %f10: c21, %f266: c22 */

/* 計算 */
sxar2
fmaddd,sc      %f0, %f4, %f8, %f8
                    /* %f8    := %f0 * %f4 + %f8 */
                    /* %f264  := %f0 * %f260 + %f264 */
fmaddd,sc      %f256, %f6, %f8, %f8
                    /* %f8    := %f256 * %f6 + %f8 */
                    /* %f264  := %f256 * %f262 + %f264 */
sxar2
fmaddd,sc      %f2, %f4, %f10, %f10
                    /* %f10   := %f2 * %f4 + %f10 */
                    /* %f266  := %f2 * %f260 + %f266 */
fmaddd,sc      %f258, %f6, %f10, %f10
                    /* %f10   := %f258 * %f6 + %f10 */
                    /* %f266  := %f258 * %f262 + %f266 */
/* 結果を格納する */
sxar2
std,s          %f8, [Z]
std,s          %f10, [Z+16]
```

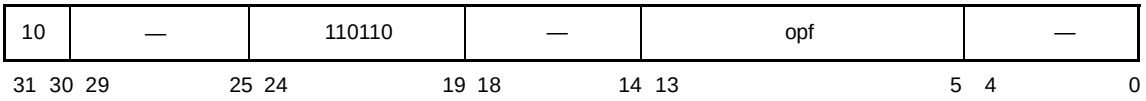
Exceptions

illegal_instruction (size = 11_2 and var ≠ 11_2)
(このとき *fp_disabled* かどうかのチェックは行わない)
fp_disabled
fp_exception_ieee_754 (NV, NX, OF, UF)
fp_exception_other (`FSR.ftt` = *unfinished_FPop*)

A.24.2 Suspend

HPC-ACE Ext. Regs.	SIMD	オペコード	opf	処理
		SUSPEND[P]	0 1000 0010	スレッドのサスペンド

Format (3)



アセンブリ言語表記
suspend

Description SUSPEND 命令は、PSTATE.IE = 1 にセットし、実行スレッドを SUSPENDED ステートに遷移させる。以下の条件により、SUSPENDED ステートから実行ステートに復帰する。

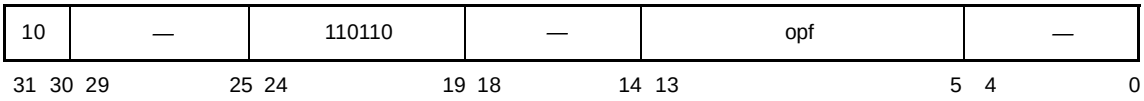
- POR, WDR, XIR
- *interrupt_vector*
- *interrupt_level_n*

Exceptions *privileged_opcode*
illegal_action (XAR.v = 1)

A.24.3 Sleep

HPC-ACE Ext.				
Regs.	**SIMD**	**オペコード**	**opf**	**処理**
		SLEEP	0 1000 0011	スレッドを一定時間停止する

Format (3)



アセンブリ言語表記
sleep

Description SLEEP は実行スレッドをスリープさせる。実行状態に復帰する条件は

- POR, WDR, XIR
- *interrupt_vector* 例外
- *interrupt_level_n* 例外
- 実装により決まる一定時間経過後
  SPARC64 VIIIfx では 1.6 マイクロ秒。STICK で計測される。
- バリアの窓 ASI に割り当てられている LBSY が更新されたとき。
  窓 ASI が割り当てられていなければ LBSY が更新されても実行状態に復帰しない。

Note – PSTATE.IE = 0 で SLEEP 命令を実行すると、インタラプトがあっても実行状態に復帰しない。

Programming Note – スレッドがスリープしていない状態で LBSY が更新されると、次の SLEEP 命令の実行でスリープしないことがある。

Exceptions *illegal_action* (XAR.v = 1)

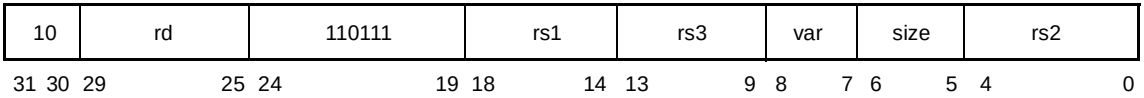
A.24.4 Integer Multiply-Add

SPARC64 VIIIfx は IMPDEP2A に整数積和演算を定義している。

HPC-ACE Ext. Regs.	SIMD	オペコード	Var[1]	Size	処理
✓	✓	FPMADDX	00	00	符号なし整数の積和演算の下位 8 バイト
✓	✓	FPMADDXHI	01	00	符号なし整数の積和演算の上位 8 バイト

1.var = 10 は Section A.76, var = 11 は Section A.75 を参照。

Format (5)



アセンブリ言語表記	
fpmaddx	*freg*$_{rs1}$, *freg*$_{rs2}$, *freg*$_{rs3}$, *freg*$_{rd}$
fpmaddxhi	*freg*$_{rs1}$, *freg*$_{rs2}$, *freg*$_{rs3}$, *freg*$_{rd}$

Description

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 rs1 で指定される倍精度レジスタと rs2 で指定される倍精度レジスタを乗じ、その結果に rs3 で指定される倍精度レジスタを加算し、結果の下位 8 バイトを rd で指定される倍精度レジスタに格納する。rs1, rs2, rs3 はすべて符号なし 8 バイト整数として扱われる。

FPMADDXHI は rs1 で指定される倍精度レジスタと rs2 で指定される倍精度レジスタを乗じ、その結果に rs3 で指定される倍精度レジスタを加算し、結果の上位 8 バイトを rd で指定される倍精度レジスタに格納する。rs1, rs2, rs3 はすべて符号なし 8 バイト整数として扱われる。

FPMADDX, FPMADDXHI は FSR のどのフィールドも更新しない。

FPMADDX, FPMADDXHI は IMPDEP2 で定義された命令だが、PA の *Impdep2_instruction* イベントでは計測されない。詳細は Appendix Q.2.1, *“ 命令種類、トラップ種類毎の統計情報 ”* (page 306) を参照。

Exceptions *fp_disabled*
illegal_action (`XAR.v` = 1 and `XAR.simd` = 1 and
(`XAR.urs1`<2> ≠ 0 or `XAR.urs2`<2> ≠ 0
or `XAR.urs3`<2> ≠ 0 or `XAR.urd`<2> ≠ 0))

A.25 Jump and Link

PSTATE.AM がセットされているとき、r[rd] には PC の上位 32 ビットを 0 にした値が入る (impl. dep. #125)。r[rd] 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。

飛び先アドレスの下位 2 ビットが 0 でない場合、*mem_address_not_aligned* 例外が通知される。このとき、DSFSR および DSFAR は更新されない (impl. dep. #237).

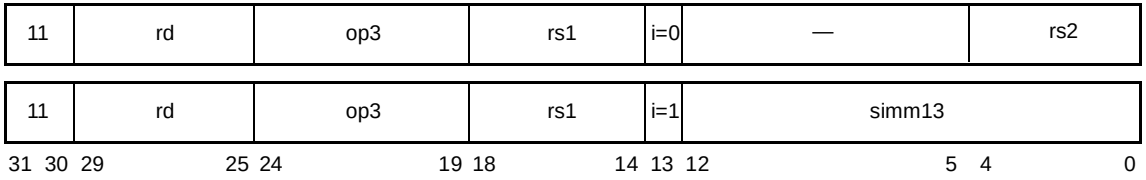
Exceptions *illegal_action* (XAR.v = 1)

A.26 Load Floating-Point

HPC-ACE Ext. Regs.	SIMD	オペコード	op3	rd	urd	処理
		LDF	10 0000	0–31	—[¶]	メモリから単精度浮動小数点レジスタへの読みだし
✓	✓	LDF	10 0000	[†]	0-7	メモリから単精度浮動小数点レジスタへの読みだし
✓	✓	LDDF	10 0011	[†]	0-7	メモリから倍精度浮動小数点レジスタへの読みだし
✓		LDQF	10 0010	[†]	0-7	メモリから4倍精度浮動小数点レジスタへの読みだし
✓		LDFSR[D]	10 0001	0	—	(A.71.4 of JPS1 **Commonality** 参照)
✓		LDXFSR	10 0001	1	—	メモリから FSR レジスタへの読みだし
		—	10 0001	2–31	—	*Reserved*

[†] JPS1 **Commonality** の Section 5.1.4 で定義される浮動小数点レジスタエンコードに従う。
[¶] `XAR.v` = 0 のとき

Format (3)



アセンブリ言語表記	
`ld`	[*address*], $freg_{rd}$
`ldd`	[*address*], $freg_{rd}$
`ldq`	[*address*], $freg_{rd}$
`ldx`	[*address*], `%fsr`

Description まず non-SIMD での動作仕様を説明する。

単精度浮動小数点ロード命令 (`LDF`) は、メモリの 4 バイト境界にある 4 バイトデータを `f[rd]` にコピーする。

倍精度浮動小数点ロード命令 (LDDF) は、メモリの 4 バイト境界にある 8 バイトデータを倍精度浮動小数点レジスタにコピーする。

4 倍精度浮動小数点ロード命令 (LDQF) は、メモリの 4 バイト境界にある 16 バイトデータを 4 倍精度浮動小数点レジスタにコピーする。

浮動小数点ステータスレジスタロード命令 (LDXFSR) 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 バイトデータを FSR にコピーする。

これらの浮動小数点ロード命令は、プライマリアドレス空間 (ASI = 80_{16}) にアクセスする。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。

LDF は、アクセスするアドレスが 4 バイト境界にない場合、*mem_address_not_aligned* を通知する。LDXFSR は、アクセスするアドレスが 8 バイト境界にない場合、*mem_address_not_aligned* を通知する。浮動小数点演算器が無効なとき (FPRS.FEF , PSTATE.PEF の値による) 、浮動小数点ロード命令は *fp_disabled* を通知する。

SPARC64 VIIIfx では、SIMD でない LDDF は、アクセスするアドレスが 8 バイト境界でない 4 バイト境界の場合、*LDDF_mem_address_not_aligned* 例外を通知する。システムソフトウェアはエミュレーションすること (impl.dep. #109(1))。

SPARC64 VIIIfx は LDQF を実装していないので、*illegal_instruction* 例外を通知する。*fp_disabled* は検出しない。システムソフトウェアは LDQF をエミュレーションすること (impl.dep. #111(1))。

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、複数の単精度ロード命令で処理するコードを生成するコンパイラがあった。SPARC V9 ではアラインされていない命令のエミューレーションが高速になると期待されるので、複数の単精度ロード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。

SPARC64 VIIIfx では、SIMD ではない浮動小数点ロードがアクセスエラーを起こした時、デスティネーションレジスタは無変更のままである (impl.dep. #44(1))。SIMD のときは下記を参照。

Programming Note – 単精度浮動小数点ロード命令 LDF 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、ロード先のレジスタは倍精度レジスタとなる。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 (page 21)、rs1, rs2 に拡張整数レジスタ、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。

SIMD 拡張

SPARC64 VIIIfx では浮動小数点ロード命令は SIMD 拡張される。SIMD 拡張されたロード命令は、basic 側の単精度・倍精度ロードと extended 側の単精度・倍精度ロードを同時に実行する。レジスタの使用方法は *“SIMD 拡張命令のレジスタ指定方法”* (page 22) を参照。

単精度 SIMD ロードは 4 バイト境界にある 2 つの単精度数データをロードする。アクセス境界違反には *mem_address_not_aligned* が通知される。

倍精度 SIMD ロードは 8 バイト境界にある 2 つの倍精度数データをロードする。アクセス境界違反には *mem_address_not_aligned* が通知される。

Note – 倍精度 SIMD ロードでは、8 バイト境界ではない 4 バイト境界に対するアクセスでも *LDDF_mem_address_not_aligned* は通知されない。

SIMD ロードは、単精度・倍精度とも、basic 側でロードされるデータと extended 側でロードされるデータが、異なるページに属することがあり得る。このようなアクセスに対し SPARC64 VIIIfx は、*SIMD_load_across_pages* 例外を通知する。例外を通知する条件は、basic 側の TLB 検索が成功し、extended 側の TLB 検索が失敗したときである。

SIMD ロード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および nontranslating ASI に対して SIMD ロードを実行しようとすると、*data_access_exception* が通知される。bypass ASI に対する SIMD ロードは、`ASI_PHYS_USE_EC{_LITTTLE}` については可能で、その場合ページサイズは 8KB と解釈される (Appendix F.11, *“MMU Bypass”* (page 202) 参照)。

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 ロードは 1 命令で basic と extended の両方を同時にロードするが、basic と extended の間でも TSO が遵守されるものとする。

SPARC64 VIIIfx では、SIMD 浮動小数点ロードがアクセスエラーを起こした時、デスティネーションレジスタは無変更のままである (impl.dep. #44(1))。

SIMD ロードにおけるエンディアン変換は、basic, extended 個別に行われる。basic, extended 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、一方のみエンディアン変換が行われる。

SIMD ロードは basic, extended どちらでもウォッチポイントを検出する。

Note – `PSTATE.AM` = 1 のとき、論理アドレス FFFF FFFF FFFF $FFFC_{16}$ に対する単精度 SIMD ロード、および論理アドレス FFFF FFFF FFFF $FFF8_{16}$ に対する倍精度 SIMD ロードの extended 側は、論理アドレス 0 番地にアクセスする。

SIMD ロードの例外条件と優先順位に関しては Appendix F.5.1, *“Trap Conditions for SIMD Load/Store”* (page 180) を参照。

Exceptions

illegal_instruction (LDQF;
LDXFSR with rd = 2–31)

fp_disabled

illegal_action (LDF, LDDF with XAR.v = 1 and (XAR.urs1 > 1 or
(i = 0 and XAR.urs2 > 1) or
(i = 1 and XAR.urs2 ≠ 0) or
XAR.urs3<2> ≠ 0);
LDF, LDDF with XAR.v = 1 and XAR.simd = 1 and XAR.urd<2> ≠ 0;
LDXFSR with XAR.v = 1 and (XAR.urs1 > 1 or
(i = 0 and XAR.urs2 > 1) or
(i = 1 and XAR.urs2 ≠ 0) or
XAR.urs3<2> ≠ 0 or
XAR.urd ≠ 0 or
XAR.simd = 1))

LDDF_mem_address_not_aligned (LDDF and (XAR.v = 0 or XAR.simd = 0))

mem_address_not_aligned

VA_watchpoint

fast_data_access_MMU_miss

SIMD_load_across_pages

data_access_exception

PA_watchpoint

data_access_error

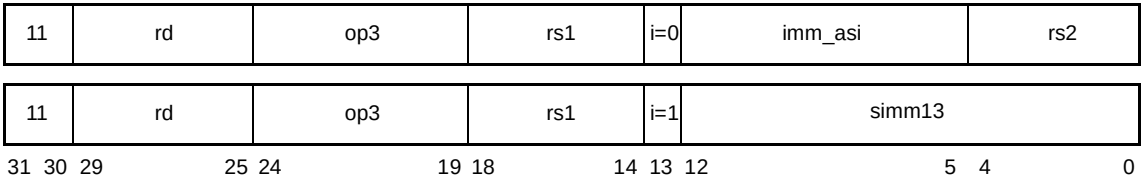
fast_data_access_protection

A.27 Load Floating-Point from Alternate Space

HPC-ACE Ext. Regs.	SIMD	オペコード	op3	rd	urd	処理
		LDFA$^{P_{ASI}}$	11 0000	0–31	—[¶]	別空間から単精度浮動小数点レジスタへの読みだし
✓	✓	LDFA	11 0000	[†]	0-7	別空間から単精度浮動小数点レジスタへの読みだし
✓	✓	LDDFA$^{P_{ASI}}$	11 0011	[†]	0-7	別空間から倍精度浮動小数点レジスタへの読みだし
✓		LDQFA$^{P_{ASI}}$	11 0010	[†]	0-7	別空間から4倍精度浮動小数点レジスタへの読みだし

[†] JPS1 **Commonality** の Section 5.1.4 で定義される浮動小数点レジスタエンコードに従う。
[¶] XAR.v = 0 のとき

Format (3)



アセンブリ言語表記	
lda	[*regaddr*] *imm_asi*, $freg_{rd}$
lda	[*reg_plus_imm*] %asi, $freg_{rd}$
ldda	[*regaddr*] *imm_asi*, $freg_{rd}$
ldda	[*reg_plus_imm*] %asi, $freg_{rd}$
ldqa	[*regaddr*] *imm_asi*, $freg_{rd}$
ldqa	[*reg_plus_imm*] %asi, $freg_{rd}$

Description まず non-SIMD での動作仕様を説明する。

別空間からの単精度浮動小数点ロード命令 (LDFA) は、メモリの 4 バイト境界にある 4 バイトデータを f[rd] にコピーする。

別空間からの倍精度浮動小数点ロード命令 (LDDFA) は、メモリの 4 バイト境界にある 8 バイトデータを倍精度浮動小数点レジスタにコピーする。

別空間からの 4 倍精度浮動小数点ロード命令 (LDQFA) は、メモリの 4 バイト境界にある 16 バイトデータを 4 倍精度浮動小数点レジスタにコピーする。

これら別空間からの浮動小数点ロード命令は、空間を示す識別子 (ASI) を必要とする。ASI は、i = 0 のとき imm_asi フィールドで指示され、i = 1 のとき ASI レジスタの値が使われる。ASI のビット 7 が 0 のときは特権アクセス、1 のときは非特権アクセスである。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。

LDFA は、アクセスするアドレスが 4 バイト境界でない場合、*mem_address_not_aligned* を通知する。浮動小数点演算器が無効なとき (FPRS.FEF , PSTATE.PEF の値による) 、別空間からの浮動小数点ロード命令は *fp_disabled* を通知する。

SPARC64 VIIIfx では、SIMD でない LDDFA は、アクセスするアドレスが 8 バイト境界でない 4 バイト境界の場合、*LDDF_mem_address_not_aligned* 例外を通知する。システムソフトウェアはエミュレーションすること (impl.dep. #109(2))。

SPARC64 VIIIfx は LDQFA を実装していないので、*illegal_instruction* 例外を通知する。*fp_disabled* は検出しない。システムソフトウェアは LDQFA をエミュレーションすること (impl.dep. #111(2))。

ASI 番号によっては、8 バイト以外のメモリアクセスを定義しているものがある。詳細は Appendix A の他の節を参照。

Implementation Note – LDFA, LDDFA は PSTATE.PRIV = 0 で ASI のビット 7 が 0 のとき *privileged_action* を通知する。

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、複数の単精度ロード命令で処理するコードを生成するコンパイラがあった。SPARC V9 ではアラインされていない命令のエミューレーションが高速になると期待されるので、コンパイラは倍精度または 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、複数の単精度ロード命令を生成すること。

SPARC64 VIIIfx では、SIMD ではない浮動小数点ロードがアクセスエラーを起こした時、デスティネーションレジスタは無変更のままである (impl.dep. #44(2))。

Programming Note – 単精度浮動小数点ロード命令 LDFA 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、ロード先のレジスタは倍精度レジスタとなる。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 (page 21)、rs1, rs2 に拡張整数レジスタ、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。

SIMD 拡張 Section A.26, *“Load Floating-Point”* の SIMD 拡張の項を参照。

Exceptions

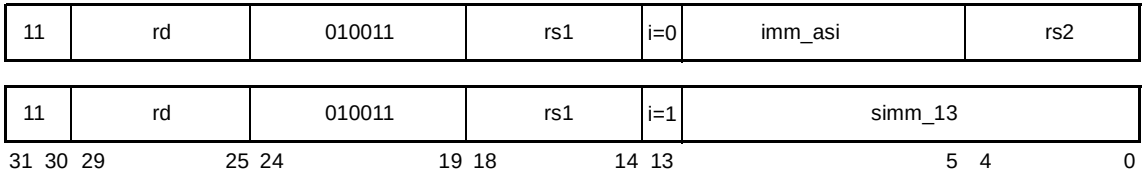
illegal_instruction (LDQFA only)
fp_disabled
illegal_action (LDFA, LDDFA with XAR.v = 1 and (XAR.urs1 > 1 or
(i = 0 and XAR.urs2 > 1) or
(i = 1 and XAR.urs2 ≠ 0) or
XAR.urs3<2> ≠ 0);
LDFA, LDDFA with XAR.v = 1 and XAR.simd = 1 and XAR.urd<2> ≠ 0)
LDDF_mem_address_not_aligned (LDDFA and (XAR.v = 0 or XAR.simd = 0))
mem_address_not_aligned
privileged_action
VA_watchpoint
fast_data_access_MMU_miss
SIMD_load_across_pages
data_access_exception
fast_data_access_protection
PA_watchpoint
data_access_error

A.30 Load Quadword, Atomic [Physical]

16 バイトロード ASI は SPARC64 VIIIfx 固有の ASI である。

HPC-ACE Ext. Regs.	SIMD	オペコード	imm_asi	ASI value	処理
✓		LDDA	ASI_QUAD_LDD_PHYS	34_{16}	物理アドレス指定で 128 ビット一括読みだし
✓		LDDA	ASI_QUAD_LDD_PHYS_L	$3C_{16}$	物理アドレス指定で 128 ビット一括読みだし、エンディアン変換あり

Format (3) LDDA



アセンブリ言語表記	
ldda	[*reg_addr*] *imm_asi*, reg_{rd}
ldda	[*reg_plus_imm*] %asi, reg_{rd}

Description

ASI 34_{16}, $3C_{16}$ は LDDA 命令で使われ、物理アドレスで指定された領域から 128 ビット長のデータを一度に (atomically) コピーする。データは偶数・奇数の 64 ビットレジスタの組にロードされる。アドレスの低位にある 64 ビットが偶数番号のレジスタに、高位にある 64 ビットが奇数番号のレジスタに格納される。

ASI 34_{16}, $3C_{16}$ は論理アドレスの 16 バイトロード (ASI 24_{16}, $2C_{16}$) の物理アドレス版で、SPARC64 VIIIfx 固有の ASI である。16 バイト境界にないアドレスにアクセスすると *mem_address_not_aligned* が通知される。

ASI_QUAD_LDD_PHYS{_L} を使ったアクセスは、TTE の設定が以下であるかのように振舞う。

- TTE.NFO = 0
- TTE.CP = 1
- TTE.CV = 0
- TTE.E = 0
- TTE.P = 1

■ TTE.W = 0

Note – TTE.IE の値はASIによって異なる。034_{16} の場合はTTE.IE = 0 で、$03C_{16}$ の場合は TTE.IE = 1 である。

このため、この ASI はキャッシャブル空間にのみ使用できる。意味的には、ASI_QUAD_LDD_PHYS{_L} はASI_NUCLEUS_QUAD_LDD とASI_PHYS_USE_ECを組み合わせたものである。

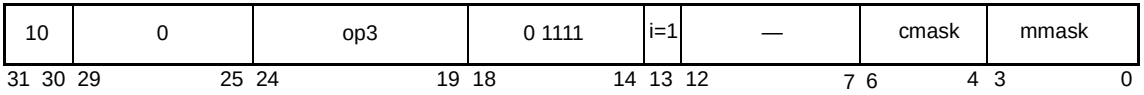
エンディアンの扱いは、64 ビット毎に行われる。各 64 ビットがバイト単位でエンディアン変換され、結果レジスタに書き込まれる。

Exceptions

illegal_instruction (misaligned rd)
illegal_action (XAR.v = 1 and (XAR.urs1 > 1 or
(i = 0 and XAR.urs2 > 1) or
(i = 1 and XAR.urs2 ≠ 0) or
XAR.urs3<2> ≠ 0
XAR.urd > 1);
XAR.v = 1 and XAR.simd = 1)
privileged_action
mem_address_not_aligned
fast_data_access_MMU_miss
data_access_exception
fast_data_access_protection
PA_watchpoint (recognized on only the first 8 bytes of a transfer)
data_access_error

A.35 Memory Barrier

Format (3)



アセンブリ言語表記	
membar	*membar_mask*

Description

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。TABLE A-6 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。

TABLE A-6 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 VIIIfx 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 VIIIfx 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の結果が、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 VIIIfx 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。TABLE A-7 (page 91) 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。

TABLE A-7 cmask フィールドの意味

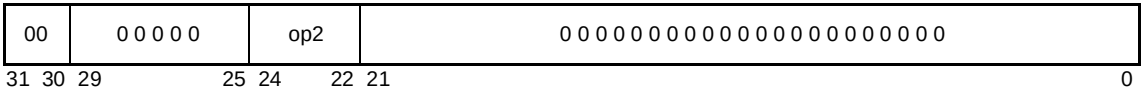
マスクビット	機能	名前	説明
cmask<2>	同期バリア	#Sync	MEMBAR より前に実行されるべきすべての命令 (メモリアクセス以外の命令も) が実行され、例外があれば、MEMBAR の後の命令の実行が開始される前に通知される。
cmask<1>	メモリ参照バリア	#MemIssue	MEMBAR の前に現れるすべてのメモリ参照命令が、MEMBAR の後に現れるどのメモリ参照命令の実行開始よりも前に実行されることを保証する。SPARC64 VIIIfx では #Sync と同じ。
cmask<0>	ルックアサイドバリア	#Lookaside	MEMBAR の前のストアが MEMBAR 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 VIIIfx では #Sync と同じ。

Exceptions *illegal_action* (XAR.v = 1)

A.41 No Operation

HPC-ACE Ext. Regs.	SIMD	オペコード	op2	処理
✓		NOP	100	何もしない

Format (2)



アセンブリ言語表記
nop

Description NOP 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 は PC と nPC 以外のレジスタ・メモリの状態を変更しない。ただし、NOP 命令を実行する時点で xar.urd = 1 の場合、SETHI 命令で rd に r[32] が指示されたものと解釈されるので、r[32] が更新される。

Exceptions *illegal_action* (XAR.v = 1 and
(XAR.simd = 1 or XAR.urs1 ≠ 0 or XAR.urs2 ≠ 0 or
XAR.urs3 ≠ 0 or XAR.urd > 1))

A.42 Partial Store (VIS I)

SPARC64 VIIIfx ではパーシャルストアのウォッチポイント検出は保守的に行われる。DCUCR のマスクビットはゼロかゼロでないかのみがチェックされる。パーシャルストア命令のバイトストアマスク (`r[rs2]`) は無視され、バイトストアマスクがゼロ (つまり何もストアされない) 場合でもウォッチポイント例外が通知される (impl. dep. #249)。

Implementation Note – ノンキャッシャブル領域に対するパーシャルストアでストアマスクが 0 のとき、SPARC64 VIIIfx はマスク 0 のバストランザクションを生成する。

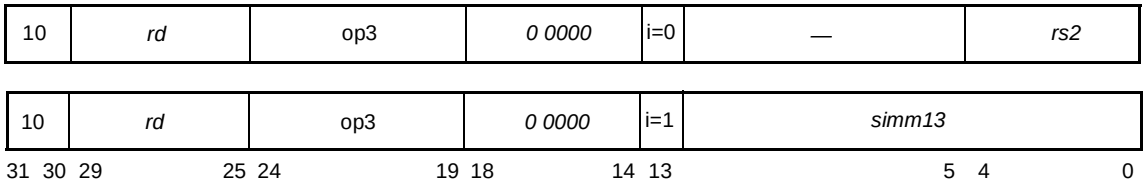
Exceptions

illegal_instruction (`i` = 1)
fp_disabled
illegal_action (`XAR.v` = 1)
LDDF_mem_address_not_aligned (see *"Partial Store ASIs"* (page 221))
mem_address_not_aligned (see *"Partial Store ASIs"* (page 221))
VA_watchpoint
fast_data_access_MMU_miss
data_access_exception (see *"Partial Store ASIs"* (page 221))
fast_data_access_protection
PA_watchpoint
data_access_error

A.48 Population Count

HPC-ACE Ext. Regs.	SIMD	オペコード	op3	処理
✓		POPC	10 1110	1 であるビット数を数える

Format (3)



アセンブリ言語表記	
popc	*reg_or_imm*, *reg*$_{rd}$

Description POPC は *i* = 0 のとき *r*[*rs2*] のビットが 1 である数を、*i* = 1 のとき sign_ext(*simm13*) のビットが 1 である数を返す。この命令は CCR を更新しない。

Note – SPARC64 V と異なり、SPARC64 VIIIfx はこの命令を実装している。

Exceptions *illegal_instruction* (instruction<18:14> ≠ 0)
illegal_action (XAR.v = 1 and (XAR.urs1 ≠ 0 or
(i = 0 and XAR.urs2 > 1) or
(i = 1 and XAR.urs2 ≠ 0) or
XAR.urs3 ≠ 0 or
XAR.urd > 1 or
XAR.simd = 1)

A.49 Prefetch Data

SPARC64 VIIIfx では PREFETCHA は以下の ASI でのみ有効である。

- ASI_PRIMARY (080_{16}), ASI_PRIMARY_LITTLE (088_{16})
- ASI_SECONDARY (081_{16}), ASI_SECONDARY_LITTLE (089_{16})
- ASI_NUCLEUS (04_{16}), ASI_NUCLEUS_LITTLE ($0C_{16}$)
- ASI_PRIMARY_AS_IF_USER (010_{16}), ASI_PRIMARY_AS_IF_USER_LITTLE (018_{16})
- ASI_SECONDARY_AS_IF_USER (011_{16}), ASI_SECONDARY_AS_IF_USER_LITTLE (019_{16})

その他の ASI については PREFETCHA は NOP として扱われる。

SPARC64 VIIIfx では、データブロックのサイズは 128 バイトで、アラインメントも 128 バイト境界である (impl. dep. #103(3))。PREFETCH/PREFETCHA 命令はアラインメント制約はなく、データブロック内の任意のアドレスが指定でき、1 データブロックをプリフェッチする。

SPARC64 VIIIfx は、TTE.CP = 0 である領域をプリフェッチできない領域と認識し、TTE.CP = 0 に対するプリフェッチは NOP として扱われる。

TABLE A-8 に SPARC64 VIIIfx のプリフェッチの動作を示す。

TABLE A-8 プリフェッチの種類

fcn	どのキャッシュにデータを載せるか	キャッシュ状態	説明
0	L1D	S,E	
1	L2	S,E	
2	L1D	M,E	
3	L2	M,E	
4	—	—	NOP
5-15	reserved (SPARC V9)		*illegal_instruction* 例外が通知される
16-19	実装依存		NOP
20	L1D	S,E	ストロングプリフェッチ
21	L2	S,E	ストロングプリフェッチ
22	L1D	M,E	ストロングプリフェッチ
23	L2	M,E	ストロングプリフェッチ
24-31	実装依存		NOP

ストロングプリフェッチ

fcn が 20, 21, 22, 23 のいずれかであるプリフェッチ命令はストロングプリフェッチである。SPARC64 VIIIfx ではストロングプリフェッチは、TLB ミスと DCUCR.weak_spca = 1 の場合を除いてロストしないことを保証されたプリフェッチである。

Programming Note – ストロングでないプリフェッチは、CPU の内部資源が枯渇している場合などは実行されない (ロスト) ことがあるが、ストロングプリフェッチはそのような状況でも実行される。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、濫用は避けるべきである。

SPARC64 VIIIfx は fcn が 20, 21, 22, 23 のときに *fast_data_access_MMU_miss* を通知しない (impl. dep. #103(2))。

ハードウェアプリフェッチ

PREFETCH, PREFETCHA 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。

Exceptions

illegal_instruction (fcn = 5–15)
illegal_action (XAR.v = 1 and (XAR.simd = 1 or
XAR.urs1 > 1 or
(i = 0 and XAR.urs2 > 1) or
(i = 1 and XAR.urs2 ≠ 0) or
XAR.urs3<2> ≠ 0 or
XAR.urd ≠ 0))

A.51 Read State Register

HPC-ACE Ext. Regs.	SIMD	オペコード	op3	rs1	処理
✓		RDY[D]	10 1000	0	Y レジスタを読み出す。使用を推奨しない命令 (JPS1 **Commonality** の A.71.9 参照)。
		—	10 1000	1	*Reserved*
✓		RDCCR	10 1000	2	CCR レジスタを読み出す。
✓		RDASI	10 1000	3	ASI レジスタを読み出す。
✓		RDTICK[PNPT]	10 1000	4	TICK レジスタを読み出す。
✓		RDPC	10 1000	5	PC レジスタを読み出す。
✓		RDFPRS	10 1000	6	FPRS レジスタを読み出す。
		—	10 1000	7–14	*Reserved*
		See text	10 1000	15	STBAR, MEMBAR または *Reserved* (JPS1 **Commonality** の A.51 参照)。
		RDASR	10 1000	16-31	SPARC V9 で定義されていない ASR を読み出す。
✓		RDPCR[PPCR]		16	PCR レジスタを読み出す。
✓		RDPIC[PPIC]		17	PIC レジスタを読み出す。
✓		RDDCR[P]		18	DCR レジスタを読み出す。
✓		RDGSR		19	GSR レジスタを読み出す。
		—		20–21	実装依存 (impl. dep. #8, 9)
✓		RDSOFTINT[P]		22	SOFTINT レジスタを読み出す。
✓		RDTICK_CMPR[P]		23	TICK_CMPR レジスタを読み出す。
✓		RDSTICK[PNPT]		24	STCIK レジスタを読み出す。
✓		RDSTICK_CMPR[P]		25	STICK_CMPR レジスタを読み出す。
		—		26-29	Reserved
✓		RDXASR		30	XASR レジスタを読み出す。
✓		RDTXAR[P]		31	TXAR レジスタを読み出す。

上表の網かけ部分については、JPS1 **Commonality** の Section A.51, *"Read State Register"* を参照。

SPARC64 VIIIfx では、RDPCR は PSTATE.PRIV = 0 かつ PCR.PRIV = 1 のとき *privileged_action* 例外を通知する。PSTATE.PRIV = 0 かつ PCR.PRIV = 0 ならば、RDPCR は例外を通知しない。(impl. dep. #250)

RDTXAR は、PSTATE.PRIV = 0 のとき *privileged_opcode* 例外を通知する。

Exceptions

privileged_opcode (RDDCR, RDSOFTINT, RDTICK_CMPR, RDSTICK, RDSTICK_CMPR, and RDTXAR)

illegal_instruction (RDASR with rs1 = 1 or 7–14;
RDASR with rs1 = 15 and rd ≠ 0;
RDASR with rs1 = 20–21, 26–29;
RDTXAR with TL = 0)

fp_disabled (RDGSR with PSTATE.PEF = 0 or FPRS.FEF = 0)

illegal_action (XAR.v = 1 and
(XAR.simd = 1 or XAR.urs1 ≠ 0 or XAR.urs2 ≠ 0 or XAR.urs3 ≠ 0 or XAR.urd > 1))

privileged_action (RDTICK with PSTATE.PRIV = 0 and TICK.NPT = 1;
RDPIC with PSTATE.PRIV = 0 and PCR.PRIV = 1;
RDSTICK with PSTATE.PRIV = 0 and STICK.NPT = 1;
RDPCR with PSTATE.PRIV = 0 and PCR.PRIV = 1)

A.59 SHUTDOWN (VIS I)

SPARC64 VIIIfx では SHUTDOWN は特権モードで NOP として動作する (impl. dep. #206)。

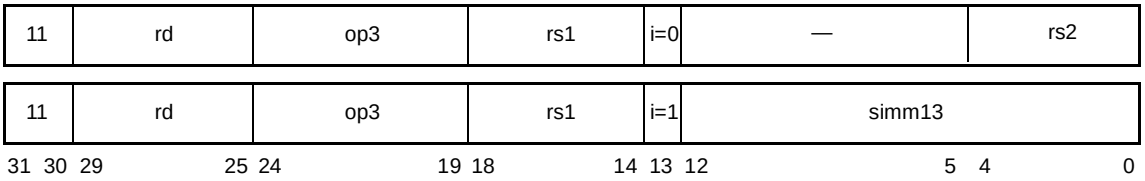
Exceptions *privileged_opcode*
illegal_action (XAR.v = 1)

A.61 Store Floating-Point

HPC-ACE Ext. Regs.	SIMD	オペコード	op3	rd	urd	処理
		STF	10 0100	0–31	—¶	単精度浮動小数点レジスタのメモリ書き込み
✓	✓	STF	10 0100	†	0-7	単精度浮動小数点レジスタのメモリ書き込み
✓	✓	STDF	10 0111	†	0-7	倍精度浮動小数点レジスタのメモリ書き込み
✓		STQF	10 0110	†	0-7	4 倍精度浮動小数点レジスタのメモリ書き込み
✓		STFSR^{D}	10 0101	0	—	(A.71.11 of JPS1 **Commonality** 参照)
✓		STXFSR	10 0101	1	—	FSR レジスタのメモリ書き込み
		–	10 0101	2–31	0	*Reserved*

† JPS1 **Commonality** の Section 5.1.4 で定義される浮動小数点レジスタエンコードに従う。
¶ XAR.v = 0 のとき

Format (3)



アセンブリ言語表記	
st	$freg_{rd}$, [*address*]
std	$freg_{rd}$, [*address*]
stq	$freg_{rd}$, [*address*]
stx	%fsr, [*address*]

Description まず non-SIMD での動作仕様を説明する。

単精度浮動小数点ストア命令 (STF) は、f[rd] の内容を 4 バイト境界にある 4 バイト領域にコピーする。

倍精度浮動小数点ストア命令 (STDF) は、倍精度浮動小数点レジスタの内容を 4 バイト境界にある 8 バイト領域にコピーする。

4 倍精度浮動小数点ストア命令 (STQF) は、4 倍精度浮動小数点レジスタの内容を 4 バイト境界にある 16 バイト領域にコピーする。

浮動小数点ステータスレジスタストア命令 (STXFSR) 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 ビットをメモリにコピーする。書き込み後 FSR.ftt はゼロクリアされる。

Implementation Note – ストア命令がトラップしないことが確実になるまで、FSR.ftt をゼロクリアしてはいけない。

これらの浮動小数点ストア命令は、プライマリアドレス空間 (ASI = 80_{16}) にアクセスする。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。

STF は、アクセスするアドレスが 4 バイト境界にないとき、*mem_address_not_aligned* を通知する。STXFSR は、アクセスするアドレスが 8 バイト境界にないとき、*mem_address_not_aligned* を通知する。浮動小数点演算器が無効なとき (FPRS.FEF , PSTATE.PEF の値による) 、浮動小数点ストア命令は *fp_disabled* を通知する。

SPARC64 VIIIfx では、SIMD でない STDF は、アクセスするアドレスが 8 バイト境界でない 4 バイト境界の場合、*STDF_mem_address_not_aligned* 例外を通知する。システムソフトウェアはエミュレーションすること (impl.dep. #110(1))。

SPARC64 VIIIfx は STQF を実装していないので、*illegal_instruction* 例外を通知する。*fp_disabled* は検出しない。システムソフトウェアは STQF をエミュレーションすること (impl.dep. #112(1))。

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。SPARC V9 ではアラインされていない命令のエミューレーションが高速になると期待されるので、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。

Programming Note – 単精度浮動小数点ストア命令 STF 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、ストア元のレジスタは倍精度レジスタとなる。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 (page 21)、rs1, rs2 に拡張整数レジスタ、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。

SIMD 拡張

SPARC64 VIIIfx では浮動小数点ストア命令は SIMD 拡張される。SIMD 拡張されたストア命令は、basic 側の単精度・倍精度ストアと extended 側の単精度・倍精度ストアを同時に実行する。レジスタの使用方法は *"SIMD 拡張命令のレジスタ指定方法 "* (page 22) を参照。

単精度 SIMD ストアは 8 バイト境界に 2 つの単精度数データをストアする。アクセス境界違反には *mem_address_not_aligned* が通知される。

倍精度 SIMD ストアは 16 バイト境界に 2 つの倍精度数データをストアする。アクセス境界違反には *mem_address_not_aligned* が通知される。

Note – 倍精度 SIMD ストアでは、8 バイト境界ではない 4 バイト境界に対するアクセスでも *STDF_mem_address_not_aligned* は通知されない。さらに、倍精度 SIMD ストアは、倍精度 SIMD ロードと異なり 8 バイト境界に対するアクセスでも *mem_address_not_aligned* が通知されることに注意。

SIMD ストアはキャッシャブル領域に対してのみストアできる。ノンキャッシャブル領域および nontranslating ASI に対して SIMD ストアを実行しようとすると、*data_access_exception* が通知される。bypass ASI に対する SIMD ストアは、`ASI_PHYS_USE_EC{_LITTTLE}` については可能である。

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 ストアは 1 命令で basic と extended の両方を同時にストアするが、basic と extended の間でも TSO が遵守される。

SIMD ストアは basic, extende どちらでもウォッチポイントを検出する。

SIMD ストアの例外条件と優先順位に関しては Appendix F.5.1, *"Trap Conditions for SIMD Load/Store"* (page 180) を参照。

Exceptions

illegal_instruction (`STXFSR` with `rd` = 2–31)
fp_disabled
illegal_action (`STF`, `STDF` with `XAR.v` = 1 and (`XAR.urs1` > 1 or
(`i` = 0 and `XAR.urs2` > 1) or
(`i` = 1 and `XAR.urs2` ≠ 0) or
`XAR.urs3`<2> ≠ 0);
`STF`, `STDF` with `XAR.v` = 1 and `XAR.simd` = 1 and `XAR.urd`<2> ≠ 0;
`STXFSR` with `XAR.v` = 1 and (`XAR.urs1` > 1 or
(`i` = 0 and `XAR.urs2` > 1) or
(`i` = 1 and `XAR.urs2` ≠ 0) or
`XAR.urs3`<2> ≠ 0 or
`XAR.urd` ≠ 0 or
`XAR.simd` = 1))
mem_address_not_aligned
STDF_mem_address_not_aligned (`STDF` and (`XAR.v` = 0 or `XAR.simd` = 0))
VA_watchpoint

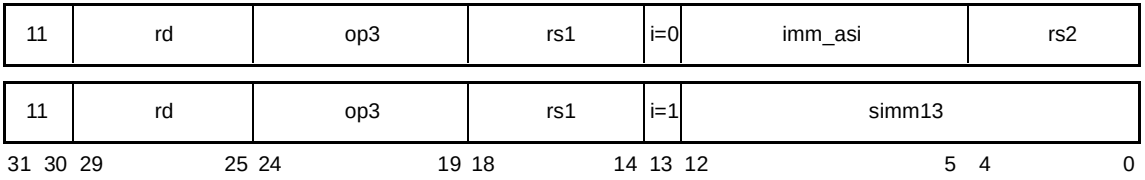
fast_data_access_MMU_miss
data_access_exception
fast_data_access_protection
PA_watchpoint
data_access_error

A.62 Store Floating-Point into Alternate Space

HPC-ACE Ext. Regs.	SIMD	オペコード	op3	rd	urd	処理
		STFA^PASI	11 0100	0–31	—¶	単精度浮動小数点レジスタの別空間への書き込み
✓	✓	STFA^PASI	11 0100	†	0-7	単精度浮動小数点レジスタの別空間への書き込み
✓	✓	STDFA^PASI	11 0111	†	0-7	倍精度浮動小数点レジスタの別空間への書き込み
✓		STQFA^PASI	11 0110	†	—	4 倍精度浮動小数点レジスタの別空間への書き込み

† JPS1 **Commonality** の Section 5.1.4 で定義される浮動小数点レジスタエンコードに従う。
¶ XAR.v = 0 のとき

Format (3)



アセンブリ言語表記	
sta	*freg$_{rd}$, [regaddr] imm_asi*
sta	*freg$_{rd}$, [reg_plus_imm]* %asi
stda	*freg$_{rd}$, [regaddr] imm_asi*
stda	*freg$_{rd}$, [reg_plus_imm]* %asi
stqa	*freg$_{rd}$, [regaddr] imm_asi*
stqa	*freg$_{rd}$, [reg_plus_imm]* %asi

Description まず non-SIMD での動作仕様を説明する。

別空間への単精度浮動小数点ストア命令 (STFA) は、f[rd] の内容を 4 バイト境界にある 4 バイト領域にコピーする。

別空間への倍精度浮動小数点ストア命令 (STDFA) は、倍精度浮動小数点レジスタの内容を 4 バイト境界にある 8 バイト領域にコピーする。

別空間への 4 倍精度浮動小数点ストア命令 (STQFA) は、4 倍精度浮動小数点レジスタの内容を 4 バイト境界にある 16 バイト領域にコピーする。

これら別空間への浮動小数点ストア命令は、空間を示す識別子 (ASI) を必要とする。ASI は、i = 0 のとき imm_asi フィールドで指示され、i = 1 のとき ASI レジスタの値が使われる。ASI のビット 7 が 0 のときは特権アクセス、1 のときは非特権アクセスである。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。

STFA は、アクセスするアドレスが 4 バイト境界にない場合、*mem_address_not_aligned* を通知する。浮動小数点演算器が無効なとき (FPRS.FEF , PSTATE.PEF の値による) 、別空間への浮動小数点ストア命令は *fp_disabled* を通知する。

Implementation Note – STQFA ではこの検査は行われない。STFA と STDFA は PSTATE.PRIV = 0 で ASI のビット 7 が 0 のとき *privileged_action* を通知する。

ASI 番号によっては、8 バイト以外のメモリアクセスを定義しているものがある。詳細は Appendix A の他の節を参照。

SPARC64 VIIIfx では、SIMD でない STDFA は、アクセスするアドレスが 8 バイト境界でない 4 バイト境界の場合、*STDF_mem_address_not_aligned* 例外を通知する。システムソフトウェアはエミュレーションすること (impl.dep. #110(2))。

SPARC64 VIIIfx は STQFA を実装していないので、*illegal_instruction* 例外を通知する。*fp_disabled* は検出しない。システムソフトウェアは STQFA をエミュレーションすること (impl.dep. #112(2))。

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。SPARC V9 ではアラインされていない命令のエミューレーションが高速になると期待されるので、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。

Programming Note – 単精度浮動小数点ストア命令 STFA 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、ストア元のレジスタは倍精度レジスタとなる。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 (page 21)、rs1, rs2 に拡張整数レジスタ、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。

SIMD 拡張 Section A.61, *“Store Floating-Point”* の SIMD 拡張の項を参照。

Exceptions

fp_disabled

illegal_action (STFA, STDFA with XAR.v = 1 and (XAR.urs1 > 1 or
(i = 0 and XAR.urs2 > 1) or
(i = 1 and XAR.urs2 ≠ 0) or
XAR.urs3<2> ≠ 0);
STFA, STDFA with XAR.v = 1 and XAR.simd = 1 and XAR.urd<2> ≠ 0)

mem_address_not_aligned

STDF_mem_address_not_aligned (STDFA and (XAR.v = 0 or XAR.simd = 0))

privileged_action

VA_watchpoint

fast_data_access_MMU_miss

data_access_exception

fast_data_access_protection

PA_watchpoint

data_access_error

A.68 Trap on Integer Condition Codes (Tcc)

Tcc 命令は XAR の値によらず、JPS1 **Commonality** で定義された通りに動作する。*illegal_action* 例外は起きない。

例外条件が成立して *trap_instruction* が通知されるときは、Tcc 命令実行直前の XAR の内容が TXAR にコピーされる。条件不成立のときは、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。詳細は *"XAR 動作"* (page 31) 参照。

Programming Note – Tcc はデバッガのブレークポイント用に使われるため、任意の位置に挿入できるよう XAR を無視する。

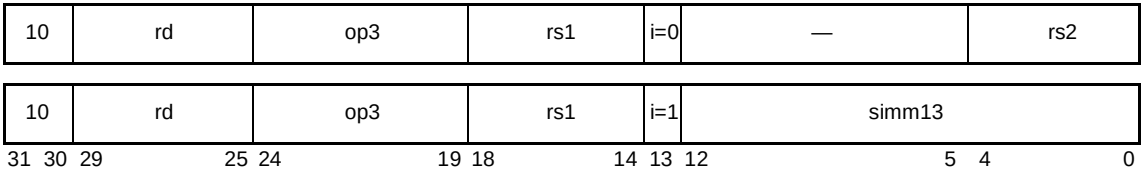
Exceptions

illegal_instruction (cc1 ▯ cc0 = 01_2 or 11_2, or reserved fields nonzero)
trap_instruction

A.69 Write Privileged Register

HPC-ACE Ext. Regs.	SIMD	オペコード	op3	処理
✓		WRPR[P]	11 0010	特権レジスタの書き込み

Format (3)



rd	特権レジスタ
0	TPC
1	TNPC
2	TSTATE
3	TT
4	TICK
5	TBA
6	PSTATE
7	TL
8	PIL
9	CWP
10	CANSAVE
11	CANRESTORE
12	CLEANWIN
13	OTHERWIN
14	WSTATE
15–31	*Reserved*

アセンブリ言語表記	
wrpr	*reg*$_{rs1}$, *reg_or_imm*, %tpc
wrpr	*reg*$_{rs1}$, *reg_or_imm*, %tnpc
wrpr	*reg*$_{rs1}$, *reg_or_imm*, %tstate
wrpr	*reg*$_{rs1}$, *reg_or_imm*, %tt
wrpr	*reg*$_{rs1}$, *reg_or_imm*, %tick
wrpr	*reg*$_{rs1}$, *reg_or_imm*, %tba
wrpr	*reg*$_{rs1}$, *reg_or_imm*, %pstate
wrpr	*reg*$_{rs1}$, *reg_or_imm*, %tl
wrpr	*reg*$_{rs1}$, *reg_or_imm*, %pil
wrpr	*reg*$_{rs1}$, *reg_or_imm*, %cwp
wrpr	*reg*$_{rs1}$, *reg_or_imm*, %cansave
wrpr	*reg*$_{rs1}$, *reg_or_imm*, %canrestore
wrpr	*reg*$_{rs1}$, *reg_or_imm*, %cleanwin
wrpr	*reg*$_{rs1}$, *reg_or_imm*, %otherwin
wrpr	*reg*$_{rs1}$, *reg_or_imm*, %wstate

Description

この命令は、i = 0 のとき “r[rs1] **xor** r[rs2]” を、i = 1 のとき “r[rs1] **xor** sign_ext(simm13)” を、特権レジスタの書き込み可能フィールドに書き込む。排他論理和であることに注意。

命令の rd フィールドは書き込む特権レジスタを指定するために使われる。TPC, TNPC, TT, TSTATE レジスタはトラップレベル毎にレジスタがあるが、現在の TL に対応したレジスタに書き込まれる。TL = 0 で TPC, TNPC, TT, TSTATE に書き込むと *illegal_instruction* が通知される。

TL に対する書き込みは、トラップを生成したりトラップから復帰したりはしない。TL 以外の CPU の状態は変更されない。

Programming Note – TL に対する書き込みは、任意のトラップレベルでの TPC, TNPC, TT, TSTATE を読み出すために使われる。TL が変更されている間にトラップが起きないよう注意すること。

WRPR は直ちに実行される。特権レジスタを更新することによるマシンステートの変更の影響は、WRPR の直後の命令から観測できる。

rd が 15–31 の範囲の WRPR は、将来のアーキテクチャのために予約されている。rd にこれらの値を指定して WRPR を実行すると、*illegal_instruction* が通知される。

WRPR で PSTATE レジスタを更新する際、AG, IG, MG の reserved な組合わせを指定すると *illegal_instruction* が通知されるが、この例外通知は *illegal_action* よりも優先順位が低い。

Exceptions

privileged_opcode

illegal_instruction ((rd = 15–31) or ((rd ≤ 3) and (TL = 0));
(rd = 6 and reserved combination of AG, IG, and MG))

illegal_action (XAR.v = 1 and (XAR.simd = 1 or
XAR.urs1 > 1 or
(i = 0 and XAR.urs2 > 1) or
(i = 1 and XAR.urs2 ≠ 0) or
XAR.urs3 ≠ 0 or
XAR.urd ≠ 0))

A.70 Write State Register

HPC-ACE Ext. Regs.	SIMD	オペコード	op3	rd	処理
✓		WRY[D]	11 0000	0	Y レジスタに書き込む。使用を推奨しない命令 (JPS1 **Commonality** の A.71.18 参照)。
		—	11 0000	1	*Reserved*
✓		WRCCR	11 0000	2	CCR レジスタに書き込む。
✓		WRASI	11 0000	3	ASI レジスタに書き込む。
		—	11 0000	4, 5	*Reserved*
✓		WRFPRS	11 0000	6	FPRS レジスタに書き込む。
		—	11 0000	7–14	*Reserved*
		—	11 0000	15	SIR 命令。(JPS1 **Commonality** の A.60 参照)。
		WRASR	11 0000	16–31	SPARC V9 で定義されていない ASR に書き込む。
✓		WRPCR[P PCR]		16	PCR レジスタに書き込む。
✓		WRPIC[P PIC]		17	PIC レジスタに書き込む。
✓		WRDCR[P]		18	DCR レジスタに書き込む。
✓		WRGSR		19	GSR レジスタに書き込む。
✓		WRSOFTINT_SET[P]		20	SOFTINT レジスタの特定ビットを 1 にする。
✓		WRSOFTINT_CLR[P]		21	SOFTINT レジスタの特定ビットを 0 にする。
✓		WRSOFTINT[P]		22	SOFTINT レジスタに書き込む。
✓		WRTICK_CMPR[P]		23	TICK_CMPR レジスタに書き込む。
✓		WRSTICK[P]		24	STICK レジスタに書き込む。
✓		WRSTICK_CMPR[P]		25	STICK_CMPR レジスタに書き込む。
		—		26-28	Reserved
✓		WRXAR		29	XAR レジスタに書き込む。
✓		WRXASR		30	XASR レジスタに書き込む。
✓		WRTXAR[P]		31	TXAR レジスタに書き込む。

上表の網かけ部分については、JPS1 **Commonality** の Section A.70, *"Write State Register"* を参照。

SPARC64 VIIIfx では、WRPCR は PSTATE.PRIV = 0 かつ PCR.PRIV = 1 のとき *privileged_action* 例外が通知される。PSTATE.PRIV = 0 かつ PCR.PRIV = 0 のときは、PCR.PRIV を変更しようとすると (すなわち 1 を書こうとすると) *privileged_action* 例外が通知される (impl. dep. #250)。

WRXAR, WRTXAR で XAR の *reserved* フィールドに 0 以外の値を書くと、*illegal_instruction* 例外が通知される。ただしこの理由による *illegal_instruction* と *illegal_action* 例外では、*illegal_action* 例外が優先して通知される。

Note – TL = 0 で WRTXAR を実行すると、XAR の値によらず *illegal_instruction* 例外が通知される。

WRXAR で XAR.v = 0 の値を書くとき、また、WRTXAR で TXAR.v = 0 の値を書くとき、関連するフィールドの値は書かれる値によらず不定になる。すなわち、

- XAR.f_v = 0 の値を書くときは、XAR.f_urs1, XAR.f_urs2, XAR.f_urs3, XAR.f_urd, XAR.f_simd の値は何を書くかによらず不定となる。
- XAR.s_v = 0 の値を書くときは、XAR.s_urs1, XAR.s_urs2, XAR.s_urs3, XAR.s_urd, XAR.s_simd の値は何を書くかによらず不定となる。
- TXAR.f_v = 0 の値を書くときは、TXAR.f_urs1, TXAR.f_urs2, TXAR.f_urs3, TXAR.f_urd, TXAR.f_simd の値は何を書くかによらず不定となる。
- TXAR.s_v = 0 の値を書くときは、TXAR.s_urs1, TXAR.s_urs2, TXAR.s_urs3, TXAR.s_urd, TXAR.s_simd の値は何を書くかによらず不定となる。

となる。

Implementation Note – XAR.v = 0 な値を書くときは関連フィールドを強制的に 0 クリアするよう実装してもよい。

Exceptions

software_initiated_reset (rd = 15, rs1 = 0, and i = 1 only)

privileged_opcode (WRDCR, WRSOFTINT_SET, WRSOFTINT_CLR, WRSOFTINT, WRTICK_CMPR, WRSTICK, WRSTICK_CMPR, and WRTXAR)

illegal_instruction (WRASR with rd = 1, 4, 5, 7–14, 26-28;
WRASR with rd = 15 and rs1 ≠ 0 or i ≠ 1,
WRTXAR with TL = 0;
WRXAR with reserved fields to nonzero)

fp_disabled (WRGSR with PSTATE.PEF = 0 or FPRS.FEF = 0)

illegal_action (XAR.v = 1 and (XAR.simd = 1 or
XAR.urs1 > 1 or
(i = 0 and XAR.urs2 > 1) or
(i = 1 and XAR.urs2 ≠ 0) or
XAR.urs3 ≠ 0 or
XAR.urd ≠ 0))

privileged_action (WRPIC with PSTATE.PRIV = 0 and PCR.PRIV = 1,
WRPCR with PSTATE.PRIV = 0 and PCR.PRIV = 1;
WRPCR to modify PCR.PRIV
with PSTATE.PRIV = 0 and PCR.PRIV = 0)

A.71 Deprecated Instructions

JPS1 **Commonality** の Appendix A.71 で定義される非推奨命令は、過去のアーキテクチャとの互換性のために提供される。新しいソフトウェアでこれらの命令を使うことは推奨されない。

A.71.10 Store Barrier

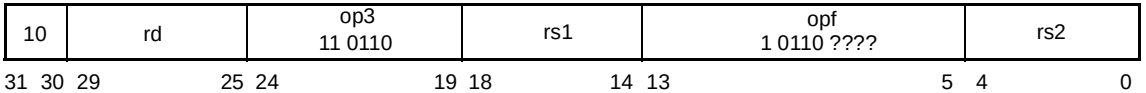
SPARC64 VIIIfx では `STBAR` は `NOP` として動作する。これは SPARC64 VIIIfx ハードウェアのメモリモデルが、すべてのメモリアクセスにこの命令を挟んでいるのと等価なためである。

Exceptions *illegal_action* (`XAR.v` = 1)

A.72 Floating-Point Conditional Compare to Register

HPC-ACE Ext. Regs.	SIMD	オペコード	op3	opf	処理	レジスタ比較
✓	✓	FCMPEQd	11 0110	1 0110 0000	倍精度レジスタ比較 (EQ)	f[rs1] = f[rs2]
✓	✓	FCMPEQEd	11 0110	1 0110 0010	倍精度レジスタ比較 (EQ)、比較不能なら例外生成	f[rs1] = f[rs2]
✓	✓	FCMPLEEd	11 0110	1 0110 0100	倍精度レジスタ比較 (LE)、比較不能なら例外生成	f[rs1] ≤ f[rs2]
✓	✓	FCMPLTEd	11 0110	1 0110 0110	倍精度レジスタ比較 (LT)、比較不能なら例外生成	f[rs1] < f[rs2]
✓	✓	FCMPNEd	11 0110	1 0110 1000	倍精度レジスタ比較 (NE)	f[rs1] ≠ f[rs2]
✓	✓	FCMPNEEd	11 0110	1 0110 1010	倍精度レジスタ比較 (NE)、比較不能なら例外生成	f[rs1] ≠ f[rs2]
✓	✓	FCMPGTEd	11 0110	1 0110 1100	倍精度レジスタ比較 (GT)、比較不能なら例外生成	f[rs1] > f[rs2]
✓	✓	FCMPGEEd	11 0110	1 0110 1110	倍精度レジスタ比較 (GE)、比較不能なら例外生成	f[rs1] ≥ f[rs2]
✓	✓	FCMPEQs	11 0110	1 0110 0001	単精度レジスタ比較 (EQ)	f[rs1] = f[rs2]
✓	✓	FCMPEQEs	11 0110	1 0110 0011	単精度レジスタ比較 (EQ)、比較不能なら例外生成	f[rs1] = f[rs2]
✓	✓	FCMPLEEs	11 0110	1 0110 0101	単精度レジスタ比較 (LE)、比較不能なら例外生成	f[rs1] ≤ f[rs2]
✓	✓	FCMPLTEs	11 0110	1 0110 0111	単精度レジスタ比較 (LT)、比較不能なら例外生成	f[rs1] < f[rs2]
✓	✓	FCMPNEs	11 0110	1 0110 1001	単精度レジスタ比較 (NE)	f[rs1] ≠ f[rs2]
✓	✓	FCMPNEEs	11 0110	1 0110 1011	単精度レジスタ比較 (NE)、比較不能なら例外生成	f[rs1] ≠ f[rs2]
✓	✓	FCMPGTEs	11 0110	1 0110 1101	単精度レジスタ比較 (GT)、比較不能なら例外生成	f[rs1] > f[rs2]
✓	✓	FCMPGEEs	11 0110	1 0110 1111	単精度レジスタ比較 (GE)、比較不能なら例外生成	f[rs1] ≥ f[rs2]

Format (3)



アセンブリ言語表記	
`fcmpgte{s,d}`	*fregrs1, fregrs2, fregrd*
`fcmplte{s,d}`	*fregrs1, fregrs2, fregrd*
`fcmpeqe{s,d}`	*fregrs1, fregrs2, fregrd*
`fcmpnee{s,d}`	*fregrs1, fregrs2, fregrd*
`fcmpgee{s,d}`	*fregrs1, fregrs2, fregrd*
`fcmplee{s,d}`	*fregrs1, fregrs2, fregrd*
`fcmpeq{s,d}`	*fregrs1, fregrs2, fregrd*
`fcmpne{s,d}`	*fregrs1, fregrs2, fregrd*

Description

これらの命令は、`rs1` で指定された浮動小数点レジスタの値と `rs2` で指定された浮動小数点レジスタの値を比較し、条件が成立していれば all1, 不成立なら all0 を、`rd` で指定された浮動小数点レジスタに格納する。

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* 例外通知時 `FSR.cexc` にセットされる値を、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。

	SNan		QNan	
命令	**例外**	**rd**	**例外**	**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}`, `STFR`, `STDFR`, 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。

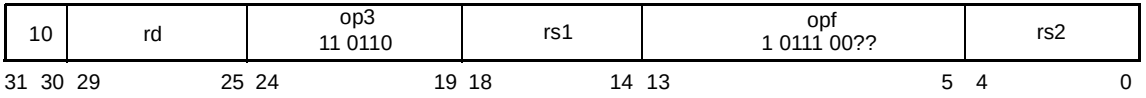
Exceptions

fp_disabled
illegal_action (`XAR.v` = 1 and `XAR.urs3` ≠ 0;
`XAR.v` = 1 and `XAR.simd` = 1 and
(`XAR.urs1`<2> ≠ 0 or `XAR.urs2`<2> ≠ 0 or `XAR.urd`<2> ≠ 0))
fp_exception_ieee_754 (`NV` if unordered)

A.73 Floating-Point Minimum and Maximum

HPC-ACE Ext. Regs.	SIMD	オペコード	op3	opf	処理
✓	✓	FMAXd	11 0110	1 0111 0000	倍精度最大値
✓	✓	FMAXs	11 0110	1 0111 0001	単精度最大値
✓	✓	FMINd	11 0110	1 0111 0010	倍精度最小値
✓	✓	FMINs	11 0110	1 0111 0011	単精度最小値

Format (3)



アセンブリ言語表記	
fmax{s,d}	*fregrs1*, *fregrs2*, *fregrd*
fmin{s,d}	*fregrs1*, *fregrs2*, *fregrd*

Description

FMAX{s, d} は、rs1 で指定された浮動小数点レジスタの値と rs2 で指定された浮動小数点レジスタの値を比較し、f[rs1] > f[rs2] なら f[rs1] を、そうでなければ f[rs2] を、rd で指定された浮動小数点レジスタに格納する。

FMIN{s, d} は、rs1 で指定された浮動小数点レジスタの値と rs2 で指定された浮動小数点レジスタの値を比較し、f[rs1] < f[rs2] なら f[rs1] を、そうでなければ f[rs2] を、rd で指定された浮動小数点レジスタに格納する。

FMIN, FMAX はゼロの符号は無視する。f[rs1], f[rs2] が +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、もう一方が QNaN, SNaN 以外の数ときは、rd には QNaN 以外の数が出力される。他の命令と異なり、NaN は伝播しない。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、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。

TABLE A-9 FMIN, FMAX の入力と出力

rs1	rs2	rd	例外
NaN 以外	NaN 以外	min(rs1, rs2), or max(rs1, rs2)	—
NaN 以外	QNaN	rs1	NV
NaN 以外	SNaN	QSNaN2	NV
QNaN	NaN 以外	rs2	NV
QNaN	QNaN	rs2 (QNaN)	NV
QNaN	SNaN	QSNaN2	NV
SNaN	NaN 以外	QSNaN1	NV
SNaN	QNaN	QSNaN1	NV
SNaN	SNaN	QSNaN2	NV

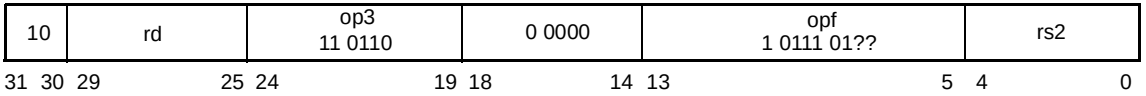
Exceptions

fp_disabled
illegal_action (XAR.v = 1 and XAR.urs3 ≠ 0;
XAR.v = 1 and XAR.simd = 1 and
(XAR.urs1<2> ≠ 0 or XAR.urs2<2> ≠ 0 or XAR.urd<2> ≠ 0))
fp_exception_ieee_754 (NV if unordered)

A.74 Floating-Point Reciprocal Approximation

HPC-ACE Ext.					
Regs.	**SIMD**	**オペコード**	**op3**	**opf**	**処理**
✓	✓	FRCPAd	11 0110	1 0111 0100	倍精度逆数の近似値
✓	✓	FRCPAs	11 0110	1 0111 0101	単精度逆数の近似値
✓	✓	FRSQRTAd	11 0110	1 0111 0110	倍精度平方根逆数の近似値
✓	✓	FRSQRTAs	11 0110	1 0111 0111	単精度平方根逆数の近似値

Format (3)



アセンブリ言語表記	
frcpa{s,d}	*fregrs2*, *fregrd*
frsqrta{s,d}	*fregrs2*, *fregrd*

Description FRCPA{s,d} は、rs2 で指定された浮動小数点レジスタの値の逆数近似値を求め、rd で指定された浮動小数点レジスタに格納する。得られる結果は近似値だが、FSR.RD 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 TABLE A-10 に示す。結果の欄の上段が例外、下段が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。

TABLE A-10 FRCPA{s,d} の演算結果

	例外と演算結果	
op2	**FSR.NS = 0**	**FSR.NS = 1**
+∞	— 0	— 0
+N ($N \geq 2^{126}$ for single, $N \geq 2^{1022}$ for double)	UF +1/N の近似値 (非正規化数)[1]	UF, NX +0
+N (+Nmin ≤ N < 2^{126} for single, +Nmin ≤ N < 2^{1022} for double)	— +1/N の近似値	— +1/N の近似値
+D	*unfinished_FPop* —	DZ +∞
+0	DZ +∞	DZ +∞
-0	DZ −∞	DZ −∞
-D	*unfinished_FPop* —	DZ −∞
-N (+Nmin ≤ N < 2^{126} for single, +Nmin ≤ N < 2^{1022} for double)	— -1/N の近似値	— -1/N の近似値
-N ($N \geq 2^{126}$ for single, $N \geq 2^{1022}$ for double)	UF -1/N の近似値 (非正規化数)[1]	UF, NX -0
−∞	— -0	— -0
SNaN	NV QSNaN2	NV QSNaN2
QNaN	— op2	— op2

1.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。

N	正の正規化数 (ゼロ , NaN, 無限大を除く)
D	正の非正規化数

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、rs2 で指定された浮動小数点レジスタの値の平方根の逆数近似値を求め、rd で指定された浮動小数点レジスタに格納する。得られる結果は近似値だが、FSR.RD 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

$$\left|\frac{frsqrta(x) - 1/(\sqrt{x})}{1/(\sqrt{x})}\right| < \frac{1}{256}$$

となる。

FRSQRTA{s, d} の演算結果と例外条件を TABLE A-11 に示す。結果の欄の上段が例外、下段が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。

TABLE A-11 FRSQRTA{s,d} の演算結果

	例外と演算結果	
op2	**FSR.NS = 0**	**FSR.NS = 1**
+∞	— 0	— 0
+N	— $+1/(\sqrt{N})$	— $+1/(\sqrt{N})$
+D	*unfinished_FPop* —	DZ +0
+0	DZ +0	DZ +0
-0	DZ +0	DZ +0
-D	NV dNaN	NV dNaN
-N	NV dNaN	NV dNaN
−∞	NV dNaN	NV dNaN
SNaN	NV QSNaN2	NV QSNaN2

TABLE A-11 FRSQRTA{s,d} の演算結果

	例外と演算結果	
op2	**FSR.NS = 0**	**FSR.NS = 1**
QNaN	— op2	— op2

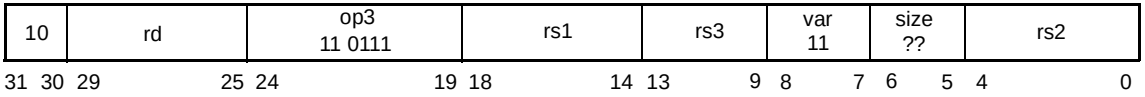
Exceptions

illegal_instruction (instruction<18:14> ≠ 0)
fp_disabled
illegal_action (XAR.v = 1 and (XAR.urs1 ≠ 0 or XAR.urs3 ≠ 0);
XAR.v = 1 and XAR.simd = 1 and
(XAR.urs2<2> ≠ 0 or XAR.urd<2> ≠ 0))
fp_exception_ieee_754 (NV, DZ, UF, NX for FRCPA{s, d};
NV, DZ for FRSQRTA{s, d})
fp_exception_other (ftt = *unfinished_FPop*)

A.75 Move Selected Floating-Point Register on Floating-Point Register's Condition

HPC-ACE Ext.						
Regs.	**SIMD**	**オペコード**	**op3**	**var**	**size**	**処理**
✓	✓	FSELMOVd	11 0111	11	00	倍精度レジスタを選択し移動
✓	✓	FSELMOVs	11 0111	11	11	単精度レジスタを選択し移動

Format (5)



アセンブリ言語表記	
fselmov{s,d}	*freg*$_{rs1}$, *freg*$_{rs2}$, *freg*$_{rs3}$, *freg*$_{rd}$

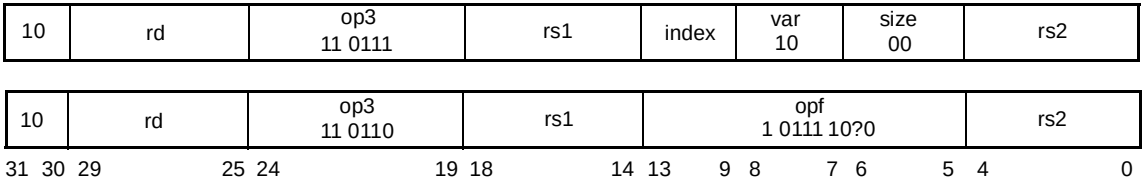
Description FSELMOV{s, d} は、rs3 で指定された浮動小数点レジスタの値の最上位ビットにより rs1 か rs2 を選択し、指定された浮動小数点レジスタの値を rd で指定された浮動小数点レジスタに格納する。rs3 で指定された浮動小数点レジスタの bit<63> が 1 なら rs1 を、0 なら rs2 を選択する。

Exceptions *fp_disabled*
illegal_action (XAR.v = 1 and XAR.simd = 1 and
(XAR.urs1<2> ≠ 0 or XAR.urs2<2> ≠ 0 or
XAR.urs3<2> ≠ 0 or XAR.urd<2> ≠ 0))

A.76 Floating-Point Trigonometric Functions

HPC-ACE Ext. Regs.	SIMD	オペコード	op3	opf	処理
✓	✓	FTRIMADDd	11 0111	—	三角関数補助演算
✓	✓	FTRISMULd	11 0110	1 0111 1010	FTRIMADDd の初期値算出
✓	✓	FTRISSELd	11 0110	1 0111 1000	係数テーブルの選択と最終段の係数算出

Format (5 and 3)



アセンブリ言語表記	
ftrimaddd	*fregrs1, fregrs2, index, fregrd*
ftrismuld	*fregrs1, fregrs2, fregrd*
ftrisseld	*fregrs1, fregrs2, fregrd*

処理	演算
FTRIMADDd	rd ← rs1 × abs(rs2) + T[rs2<63>][index]
FTRISMULd	rd ← (rs2<0> << 63) ^ (rs1 × rs1)
FTRISSELd	rd ← (rs2<1> << 63) ^ (rs2<0> ? 1.0 : rs1)

Description 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。FIGURE A-1 に、これら 3 命令が行う計算式を示す。

$$\sin x \cong x - \frac{1}{3!}x^3 + \frac{1}{5!}x^5 - \frac{1}{7!}x^7 + \frac{1}{9!}x^9 - \frac{1}{11!}x^{11} + \frac{1}{13!}x^{13} - \frac{1}{15!}x^{15}$$

$$= x\left(1 - \frac{1}{3!}x^2 + \frac{1}{5!}x^4 - \frac{1}{7!}x^6 + \frac{1}{9!}x^8 - \frac{1}{11!}x^{10} + \frac{1}{13!}x^{12} - \frac{1}{15!}x^{14}\right)$$

$$= x \cdot \underbrace{\left(\left(\left(\left(\left(\left(\left(0 \cdot x^2 - \frac{1}{15!}\right)x^2 + \frac{1}{13!}\right)x^2 - \frac{1}{11!}\right)x^2 + \frac{1}{9!}\right)x^2 - \frac{1}{7!}\right)x^2 + \frac{1}{5!}\right)x^2 - \frac{1}{3!}\right)x^2 + 1\right)}_{\texttt{FTRIMADDd}}$$

$$\cos x \cong 1 - \frac{1}{2!}x^2 + \frac{1}{4!}x^4 - \frac{1}{6!}x^6 + \frac{1}{8!}x^8 - \frac{1}{10!}x^{10} + \frac{1}{12!}x^{12} - \frac{1}{14!}x^{14}$$

$$= 1 \cdot \underbrace{\left(\left(\left(\left(\left(\left(\left(0 \cdot x^2 - \frac{1}{14!}\right)x^2 + \frac{1}{12!}\right)x^2 - \frac{1}{10!}\right)x^2 + \frac{1}{8!}\right)x^2 - \frac{1}{6!}\right)x^2 + \frac{1}{4!}\right)x^2 - \frac{1}{2!}\right)x^2 + 1\right)}_{\texttt{FTRIMADDd}}$$

FIGURE A-1 SPARC64 VIIIfx の三角関数補助演算

FTRIMADDd は、rs1 で指定される倍精度浮動小数点レジスタの値と rs2 で指定される倍精度浮動小数点レジスタの値の絶対値を乗じ、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 rd で指定される倍精度浮動小数点レジスタに格納する。FTRIMADDd は sin(x), cos(x) の級数部分の計算を行う。

FTRISMULd は、rs1 で指定される倍精度浮動小数点レジスタの値を二乗し、rs2 で指定される倍精度浮動小数点レジスタの値の最下位ビットを符号とする結果を、rd で指定される倍精度浮動小数点レジスタに格納する。FTRISMULd は FTRIMADDd の初期値を求めるために使われる。

FTRISSELd は、rs1 で指定される倍精度浮動小数点レジスタの値か 1.0 を、rs2 で指定される倍精度浮動小数点レジスタの値の最下位ビットにより選択し、その値に rs2 で指定される倍精度浮動小数点レジスタの値のビット 1 を符合として排他的論理和 (exclusive or) を取り、rd で指定される倍精度浮動小数点レジスタに格納する。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 f[rs1] にゼロ、f[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。FIGURE A-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。

- f[rs2]<63> = 0 のとき、sin(x) の係数テーブルを使う
- f[rs2]<63> = 1 のとき、cos(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。TABLE A-12, TABLE A-13 に SPARC64 VIIIfx の FTRIMADDd の係数テーブルの内容を示す。

TABLE A-12 sin(x) (f[rs2]<63> = 0) の係数テーブル

	演算に使われる係数		数学的に正しい係数
Index	**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!

TABLE A-13 cos(x) (f[rs2]<63> = 1) の係数テーブル

	演算に使われる係数		数学的に正しい係数
Index	**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。f[rs1] に x、f[rs2] に FIGURE A-2 の Q を与えると、x^2 を計算し、符号のところに係数テーブル選択するための情報を付加した値を返す。Q は浮動小数点数ではなく整数で与える。f[rs2]<63:1> は使われない。f[rs2] が NaN であっても例外を検出しない。

FTRISSELd は FTRIMADDd による級数計算の後、最後に乗ずる値を計算する。
f[rs1] に x、f[rs2] に FIGURE A-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。f[rs2]<63:2> は使われない。f[rs2] が NaN であっても例外を検出しない。

q: $(2q-1)\cdot\frac{\pi}{4} < x \leq (2q+1)\cdot\frac{\pi}{4}$

Q: q **mod** 4

R: $x - q\cdot\frac{\pi}{2}$ $\left(-\frac{\pi}{4} < R \leq \frac{\pi}{4}\right)$

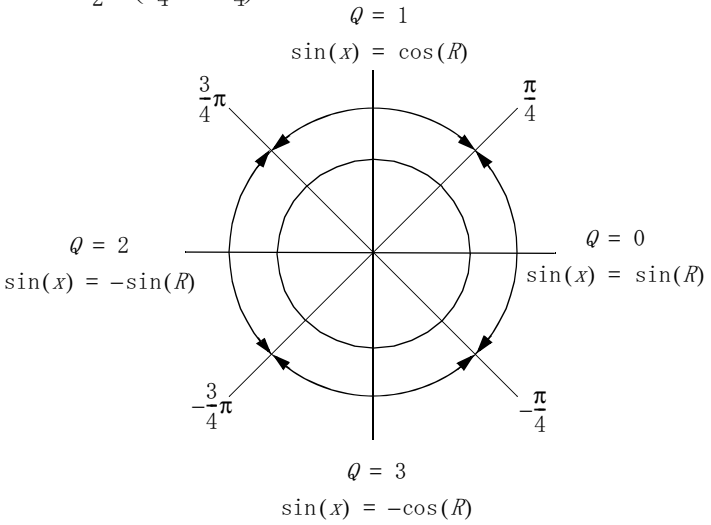


FIGURE A-2 三角関数補助演算を用いた sin(x) の計算

例 : 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
 */
```

Exceptions

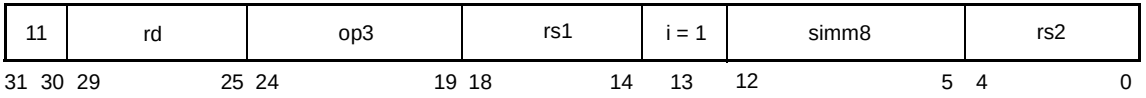
illegal_instruction (FTRIMADDd with index > 7)
fp_disabled
illegal_action (XAR.v = 1 and XAR.urs3 ≠ 0;
XAR.v = 1 and XAR.simd = 1 and
(XAR.urs1<2> ≠ 0 or XAR.urs2<2> ≠ 0 or XAR.urd<2> ≠ 0))
fp_exception_ieee_754 (FTRIMADDd with NV, NX, OF, UF;
FTRISMULd with NX, OF, UF;
FTRISMULd with NV (rs1 only))
fp_exception_other (FTRIMADDd, FTRISMULd with ftt = *unfinished_FPop*)

A.77 Store Floating-Point Register on Register Condition

HPC-ACE Ext. Regs.	SIMD	オペコード	op3	rd	urd	処理
		STFR	10 1100	0–31	¶	条件付き単精度ストア
✓	✓	STFR	10 1100	†	0-7	条件付き単精度ストア
✓	✓	STDFR	10 1111	†	0-7	条件付き倍精度ストア

† JPS1 **Commonality** の Section 5.1.4 で定義される浮動小数点レジスタエンコードに従う。
¶ XAR.v = 0 のとき

Format (3)



アセンブリ言語表記	
stfr	$freg_{rd}$, $freg_{rs2}$, [*address*]
stdfr	$freg_{rd}$, $freg_{rs2}$, [*address*]

Description

STFR は、f[rs2] の最上位ビットが 1 のとき、f[rd] で指示される単精度浮動小数点レジスタの内容を、4 バイト境界の 4 バイト領域に書き込む。

STDFR は、f[rs2] の最上位ビットが 1 のとき、f[rd] で指示される倍精度浮動小数点レジスタの内容を、8 バイト境界の 8 バイト領域に書き込む。

書き込みアドレスは “r[rs1] + sign_ext(simm8 << 2)” で計算される。

STFR 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* 例外を通知する。

STDFR 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* 例外を通知する。

SIMD でない 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* 例外を通知する。

浮動小数点演算器が使えないとき (FPRS.FEF , PSTATE.PEF の設定による)、STFR, STDFR は *fp_disabled* を通知する。

STFR, STDFR のウォッチポイント検出は、実際にストアをする / しないに係らず、アドレスがマッチしていれば例外を通知する。

Programming Note – 単精度浮動小数点ストア命令 STFR 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、ストア元のレジスタは倍精度レジスタとなる。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 (page 21)、rs1, rs2 に拡張整数レジスタ、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。

SIMD 拡張

SPARC64 VIIIfx では STFR, STDFR は SIMD 拡張される。SIMD 拡張された STFR, STDFR は、basic 側の単精度・倍精度ストアと extended 側の単精度・倍精度ストアを同時に実行する。レジスタの使用方法は *"SIMD 拡張命令のレジスタ指定方法 "* (page 22) を参照。

SIMD STFR は 8 バイト境界に 2 つの単精度数データをストアする。アクセス境界違反には *mem_address_not_aligned* が通知される。

SIMD STDFR は 16 バイト境界に 2 つの倍精度数データをストアする。アクセス境界違反には *mem_address_not_aligned* が通知される。SIMD STDFR では、4 バイト境界に対するアクセスでも *STDF_mem_address_not_aligned* は通知されない。

SIMD STFR, SIMD STDFR はキャッシャブル領域に対してのみストアできる。ノンキャッシャブル領域に対して SIMD STFR, SIMD STDFR を実行しようとすると、*data_access_exception* が通知される。bypass ASI に対する SIMD ストアは、ASI_PHYS_USE_EC{_LITTTLE} については可能である。

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, SIMD STDFR は 1 命令で basic と extended の両方を同時にストアするが、basic と extended の間でも TSO が遵守される。

SIMD STFR, SIMD STDFR は basic, extende どちらでもウォッチポイントを検出する。

SIMD STFR, SIMD STDFR の例外条件と優先順位に関しては Appendix F.5.1, *"Trap Conditions for SIMD Load/Store"* (page 180) を参照。

Exceptions

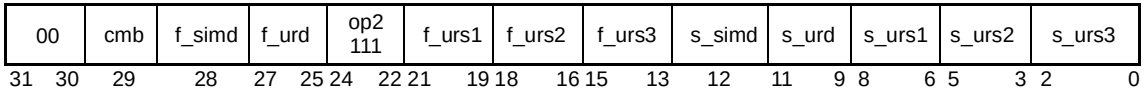
illegal_instruction (i = 0)
fp_disabled
illegal_action (XAR.v = 1 and (XAR.urs1 > 1 or
XAR.urs3<2> ≠ 0);
XAR.v = 1 and XAR.simd = 1 and
(XAR.urs2<2> ≠ 0 or XAR.urd<2> ≠ 0))
mem_address_not_aligned
STDF_mem_address_not_aligned (STDFR and (XAR.v = 0 or XAR.simd = 0))

VA_watchpoint
fast_data_access_MMU_miss
data_access_exception
fast_data_access_protection
PA_watchpoint
data_access_error

A.78 Set XAR (SXAR)

HPC-ACE Ext. Regs.	SIMD	オペコード	op2	cmb	処理
		SXAR1	111	0	後続 1 命令の XAR 指定
		SXAR2	111	1	後続 2 命令の XAR 指定

Format (2)



アセンブリ言語表記
sxar1
sxar2

Description

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 の直後に実行される命令に、s_ で始まるフィールドは 2 命令目に適用される。SXAR1 の s_* フィールドは無視される。

Compatibility Note – SXAR1 の s_ で始まるフィールドが 0 でも *illegal_instruction* 例外は通知されないが、将来の互換性を考慮したプログラムでは、SXAR1 の s_ で始まるフィールドに 0 以外の値を書くべきではない。

SXAR 命令は後続の最大 2 命令で、SPARC64 VIIIfx で追加された整数・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ために使われる。

Implementation Note – 命令列上連続する命令を高速に実行できるようにハードウェアを実装してよい。

SXAR 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 を挟んでいると、性能は低下する可能性がある。

SXAR の実行は IIU_INST_TRAP で検出できない場合がある。

SXAR 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* が通知される。

Compatibility Note – op = 00_2, op2 = 111_2 は SPARC V9 では reserved だが、SPARC V8 でははFBcc 命令が定義されていた。SPARC V8 のアプリケーションを SPARC64 VIIIfx で動かすと、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はない。HPC-ACE 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。

```
        sxar1
        faddd,s   %f0, %f2, %f4          /* SIMD */
```

Exceptions *illegal_action* (XAR.v = 1)

A.79 Cache Line Fill with Undetermined Values

HPC-ACE Ext. Regs.	SIMD	オペコード	imm_asi	ASI Value	処理
✓		STXA STDA[D] STDFA	ASI_XFILL_AIUP	72_{16}	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 STDA[D] STDFA	ASI_XFILL_AIUS	73_{16}	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 STDA[D] STDFA	ASI_XFILL_P	$f2_{16}$	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 STDA[D] STDFA	ASI_XFILL_S	$f3_{16}$	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。

Description

STXA, STDA, 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、8 バイト境界にある論理アドレスならば、キャッシュライン上の任意のアドレスを指定してよい。

STXA, STDA 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。

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。

XFILL_{AIUP,AIUS,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。

XFILL_{AIUP,AIUS,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。

TTE.CP = 0 であるページに対する XFILL_{AIUP,AIUS,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。

XFILL_{AIUP,AIUS,S,P} ではバスエラー、バスタイムアウトによる *ECC_error* 例外は通知されない。また、XFILL_{AIUP,AIUS,S,P} は、指定されたアドレスが L1 または L2 キャッシュにあり、そのキャッシュライン上に UE がある場合でも、*data_access_error* 例外は通知しない。

XFILL_{AIUP,AIUS,S,P} はキャッシュライン 128 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。

Programming Notes – XFILL 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は XFILL 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、XFILL 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。

本 ASI は memset(), memcpy() の高速化を意図して設けられた。以下に本 ASI を用いた memset()/memcpy() のサンプルコードを示す。命令ニーモニックは HPC-ACE 拡張サフィックスを使用している。詳細は Appendix G.4, *“HPC-ACE 拡張機能の表記法 ”* (page 206) を参照。

これらのサンプルコードはいずれも、セクタ 0 に再利用性の低いデータがキャッシュされる場合のコードであることに注意。セクタ 0 とセクタ 1 をどういう用途に使うかはアプリケーションが決めることなので、セクタ 1 に再利用性の低いデータをキャッシュするアプリケーションでこのサンプルコードを使うと、性能低下を引き起こす可能性がある。

[memset(0) の pseudo-code]

```
/*
 *  %i0: dst
 */
ahead = 4 * 128;!適時調整
for (i = 0 ; i < size; i += 128) {
   stxa      %g0, [%i0+ahead] #ASI_XFILL

   sxar2
   stx,d     %g0, [%i0]
   stx,d     %g0, [%i0+8]
   sxar2
   stx,d     %g0, [%i0+16]
   stx,d     %g0, [%i0+24]
   sxar2
   stx,d     %g0, [%i0+32]
   stx,d     %g0, [%i0+40]
   sxar2
   stx,d     %g0, [%i0+48]
   stx,d     %g0, [%i0+56]
   sxar2
```

```
   stx,d     %g0, [%i0+64]
   stx,d     %g0, [%i0+72]
   sxar2
   stx,d     %g0, [%i0+80]
   stx,d     %g0, [%i0+88]
   sxar2
   stx,d     %g0, [%i0+96]
   stx,d     %g0, [%i0+104]
   sxar2
   stx,d     %g0, [%i0+112]
   stx,d     %g0, [%i0+120]

   add       %i0, 128, %i0
}

[memcpy() の pseudo-code]
/*
 *  %i0: dst
 *  %i1: src
 */
ahead = 4 * 128;! 適時調整
for (i = 0 ; i < size; i += 128) {
   prefetch  [%i1+128], #n_reads
   ldx       [%i1], %l2
   ldx       [%i1+8], %l3
   ldx       [%i1+16], %l4
   ldx       [%i1+24], %l5
   ldx       [%i1+32], %l6
   ldx       [%i1+40], %l7
   ldx       [%i1+48], %o0
   ldx       [%i1+56], %o1
   ldx       [%i1+64], %o2
   ldx       [%i1+72], %o3
   ldx       [%i1+80], %o4
   ldx       [%i1+88], %o5
   ldx       [%i1+96], %o6
   ldx       [%i1+104], %o7
   ldx       [%i1+112], %i6
   ldx       [%i1+120], %i7

   stxa      %g0, [%i0+ahead] #ASI_XFILL

   prefetch  [%i0+128], #n_writes
   sxar2
   stx,d     %l2, [%i0]
   stx,d     %l3, [%i0+8]
   sxar2
```

```
        stx,d     %l4, [%i0+16]
        stx,d     %l5, [%i0+24]
        sxar2
        stx,d     %l6, [%i0+32]
        stx,d     %l7, [%i0+40]
        sxar2
        stx,d     %o0, [%i0+48]
        stx,d     %o1, [%i0+56]
        sxar2
        stx,d     %o2, [%i0+64]
        stx,d     %o3, [%i0+72]
        sxar2
        stx,d     %o4, [%i0+80]
        stx,d     %o5, [%i0+88]
        sxar2
        stx,d     %o6, [%i0+96]
        stx,d     %o7, [%i0+104]
        sxar2
        stx,d     %i6, [%i0+112]
        stx,d     %i7, [%i0+120]

        add       %i1, 128, %i1
        add       %i0, 128, %i0
    }
```

Exceptions

*fp_disabled (*STDFA*)*
*illegal_action (*STXA, STDA *with* XAR.v *= 1 and (*XAR.urs1 *> 1 or*
*(*i *= 0 and* XAR.urs2 *> 1) or*
*(*i *= 1 and* XAR.urs2 *≠ 0) or*
XAR.urs3<*2*> *≠ 0 or*
XAR.urd *> 1);*
STDFA *with* XAR.v *= 1 and(*XAR.urs1 *> 1 or*
*(*i *= 0 and* XAR.urs2 *> 1) or*
*(*i *= 1 and* XAR.urs2 *≠ 0) or*
XAR.urs3<*2*> *≠ 0);*
XAR.v *= 1 and* XAR.simd *= 1)*
mem_address_not_aligned
STDF_mem_address_not_aligned
*privileged_action (*ASI_XFILL_AIUP, ASI_XFILL_AIUS*)*
VA_watchpoint
fast_data_access_MMU_miss
data_access_exception
fast_data_access_protection
PA_watchpoint
data_access_error

F.APPENDIX **B**

IEEE Std. 754-1985 Requirements for SPARC-V9

IEEE Std. 754-1985 浮動小数点数規格は実装依存仕様を許容している。JPS1 **Commonality** の Appendix B では SPARC V9 に準拠したプロセッサ間での処理が統一されるよう、IEEE 754-1985 実装依存仕様の扱いを定義している。その詳細については JPS1 **Commonality** を参照。

この章で規定するのは、

- unfinished_FPop が起きる条件
- IEEE 非準拠モードの動作仕様

である。なお、前者は JPS1 **Commonality** の Section 5.1.7, *"Floating-Point State Register (FSR)"* (page 22)内の *"FSR_floating-point_trap_type (ftt)"* の実装依存部を明確に再定義したもので、便宜上この章に配置してある。

B.1 Traps Inhibiting Results

詳細は JPS1 **Commonality** の Section B.1 を参照。SPARC64 VIIIfx はハードウェアとシステムソフトウェアが協調して、この節で定義されている状態を実現する。

B.6 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 VIIIfx 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 VIIIfx では IEEE 準拠モード (`FSR.NS` = 0) のときのみ起きる例外だが、読者の便宜を考えこの章に配置してある。

SPARC64 VIIIfx 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 VIIIfx 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 VIIIfx 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 (*"FSR_nonstandard_fp (NS)"* (page 22) を参照)。FSR.NS の値により SPARC64 VIIIfx の動作は変わってくる。

B.6.1 *fp_exception_other* Exception (ftt=*unfinished_FPop)*

SPARC64 VIIIfx 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、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。

Implementation Note – 境界条件を精確に検出するためには大量の論理回路が必要になる。このようなハードウェアのコストを避けるため、SPARC64 VIIIfx は中間結果の指数部 (丸める前の値) を計算することで境界条件にあてはまるかどうかを大まかに判定する。このような判定方法のため、精確にはゼロやオーバフローにならない値でも、そうであると判定し、*unfinished_FPop* 例外を通知することがある。

境界条件検出のための、結果の指数部を求める式を TABLE B-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。

TABLE B-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`	*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。

TABLE B-2 に、全浮動小数点演算命令の *unfinished_FPop* 検出条件をまとめた。

TABLE B-2 *unfinished_FPop* 検出条件

処理	検出条件
`FdTOs`	–25 < *eres* < 1 かつ `TEM.UFM` = 0 のとき。
`FsTOd`	第二オペランド (`rs2`) が非正規化数のとき。
`FADDs`, `FSUBs`, `FADDd`, `FSUBd`	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)[1] 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 `TEM.UFM` = 0 のとき。
`FMULs`, `FMULd`	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 `TEM.UFM` = 0 のとき、 単精度演算 : –25 < eres < 1 倍精度演算 : –54 < eres < 1

TABLE B-2 *unfinished_FPop* 検出条件 *(Continued)*

処理	検出条件
FsMULd	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。
FDIVs, FDIVd	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 TEM.UFM = 0 のとき、 単精度演算 : –25 < eres < 1 倍精度演算 : –54 < eres < 1
FSQRTs, FSQRTd	入力 (rs2) が正のゼロでない非正規化数のとき。
FMADD{s,d}, FMSUB{s,d}, FNMADD{s,d}, FNMSUB{s,d}	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	1. rs1 がゼロ以外の正規化数 (無限大と NaN を除く) で TEM.UFM = 0 のとき、 倍精度演算 : –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。

1.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。

Conditions for a Zero Result

TABLE B-3 の条件が成立するとき、SPARC64 VIIIfx 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。

TABLE B-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[1]		両方のオペランドが非正規化数	両方が正規化数[2]
FdTOs	常に		—	*eres* ≤ -25
FMULs, FMULd	単精度 倍精度	*Er* ≤ –25 *Er* ≤ –54	常に	単精度 倍精度	*eres* ≤ –25 *eres* ≤ –54
FDIVs, FDIVd	単精度 倍精度	*Er* ≤ –25 *Er* ≤ –54	起こらない	単精度 倍精度	*eres* ≤ –25 *eres* ≤ –54

1.両方ともゼロ、NaN、無限大以外。

2.両方がゼロでもよいが、どちらも無限大、NaN ではない。

Conditions for an Overflow Result

TABLE B-4 の条件が成立するとき、SPARC64 VIIIfx は演算でオーバーフローが起きたものとみなす。

TABLE B-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。

B.6.2 Behavior when FSR.NS = 1

FSR.NS = 1 (IEEE 非準拠モード) の場合、SPARC64 VIIIfx 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、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 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

  cexc.nxc = 1 かつ FSR の 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 の TEM.UFM = 1 ならば cexc.ufc = 1 が、TEM.UFM = 0 で TEM.NXM = 1 なら cexc.nxc = 1 がセットされ、*fp_exception_ieee_754* 例外が通知される。TEM.UFM = 0 かつ TEM.NXM = 0 のときは、cexc.ufc = 1, 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 VIIIfx 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。

TABLE B-5 は、TABLE B-2 であげた浮動小数点演算命令[1] 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とたどると、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 TABLE B-6 を参照。TABLE B-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 – TABLE B-5, TABLE B-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。

TABLE B-5 浮動小数点例外の発生条件と結果 *(1 of 2)*

FSR.NS	入力が非正規化数[1]	結果が非正規化数[2]	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[3]
			No	—	1	—	—	UF
					0	—	—	*unfinished_FPop*[4]
		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[5]
				No	—	—	—	*unfinished_FPop*

1. FTRISmuld の rs2 は浮動小数点数ではないので、Source Denormal の対象外。

TABLE B-5 浮動小数点例外の発生条件と結果 *(2 of 2)*

FSR.NS	入力が非正規化数[1]	結果が非正規化数[2]	Zero Result	Overflow Result	UFM	OFM	NXM	結果
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	—	—	—	—	—	—	TABLE B-6

1.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。

2.丸め前の演算結果が非正規化数になった場合。

3.Dmin = 最小の非正規化数。

4.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。

5.Nmax = 最大の正規化数。

TABLE B-6 は、TABLE B-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 VIIIfx の動作をまとめたものである。TABLE B-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。

TABLE B-6 FSR.NS = 1 における非正規化数の演算 *(1 of 2)*

命令	入力値			FSR.TEM				結果
	op1	op2	op3	UFM	NXM	DVM	NVM	
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} FSUB{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} FsMULd	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

TABLE B-6 FSR.NS = 1 における非正規化数の演算 *(2 of 2)*

命令	入力値			FSR.TEM				結果
	op1	op2	op3	UFM	NXM	DVM	NVM	
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[1]
FSQRT{s,d}	—	Denorm and op2 > 0	—	—	1	—	—	NX
					0	—	—	nx, ゼロ
		Denorm and op2 < 0	—		—	—	1	NV
							0	nv, dNaN[1]
FMADD{s,d} FMSUB{s,d} FNMADD{s,d} FNMSUB{s,d} FTRIMADDd[2]	Denorm	—	Normal	—	1	—	—	NX
					0	—	—	nx, op3
			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[3]
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

1.単精度の dNaN は $7FFF.FFFF_{16}$、倍精度の dNaN は $7FFF.FFFF.FFFF.FFFF_{16}$.

2.op3 は演算器内のテーブルから引いてくる。常に正規化数。

3.op1 × op2 が Denorm のときは、op1 × op2 と同符号のゼロ。

F.APPENDIX **C**

Implementation Dependencies

この章では、JPS1 **Commonality** で実装依存仕様とされたものについて、SPARC64 VIIIfx がどのように実装しているかをまとめている。SPARC V9 と SPARC JPS1 で、実装依存仕様は **“IMPL. DEP. *#nn*:”** という形式で記述されており、この仕様書で実装依存仕様の実装を定義した部分には “(impl. dep. *#nn*)” という形式で、対応する実装依存仕様が示されている。この実装依存仕様の一覧を TABLE C-1 にまとめた。

Note – SPARC International は、SPARC V9 に準拠したすべてのプロセッサの実装依存について書かれたドキュメント *Implementation Characteristics of Current SPARC-V9-based Products, Revision 9.x* を提供している。このドキュメントに関する問い合わせは下記へ。
home page: `www.sparc.org`
email: info@sparc.org

C.4 List of Implementation Dependencies

TABLE C-1 は、JPS1 実装依存仕様が SPARC64 VIIIfx でどのように実装されているかをまとめた表である。

TABLE C-1 JPS1 実装依存仕様の SPARC64 VIIIfx での実装 *(1 of 12)*

番号	SPARC64 VIIIfx での実装	ページ
1	**命令のソフトウェアエミュレーション** ソフトウェアは *illegal_instruction* または *unimplemented_FPop* 例外を通知するすべての 4 倍精度命令をエミュレーションすること。	—

TABLE C-1 JPS1 実装依存仕様の SPARC64 VIIIfx での実装 *(2 of 12)*

番号	SPARC64 VIIIfx での実装	ページ
2	**汎用整数レジスタの本数** SPARC64 VIIIfx は 8 レジスタウインドウを実装する (NWINDOWS = 8)。 SPARC64 VIIIfx はさらに 2 セットのグローバルレジスタ (インタラプト用と MMU 用) を実装するほか、HPC-ACE でも拡張され、合計 192 本の汎用整数レジスタを実装する。	—
3	**IEEE 754-1985 仕様に準拠しない演算結果** Appendix B.6, *"Floating-Point Nonstandard Mode"* を参照。	139
4–5	Reserved.	
6	**I/O レジスタのアクセス権限** この実装依存項目は本仕様書の規定範囲外である。SPARC64 VIIIfx を搭載するシステムの仕様書を参照。	—
7	**I/O レジスタ定義** この実装依存項目は本仕様書の規定範囲外である。SPARC64 VIIIfx を搭載するシステムの仕様書を参照。	—
8	**RDASR/WRASR 対象レジスタ** SPARC64 VIIIfx では RDASR で XAR, XASR, TXAR の読み出しができ、WRASR で XASR, TXAR の書き込みができる。	97, 111
9	**RDASR/WRASR のアクセス権限** SPARC64 VIIIfx では TXAR は特権レジスタである。	97, 111
10–12	Reserved.	
13	**VER.impl** SPARC64 VIIIfx では VER.impl = 8 である。	26
14–15	Reserved.	—
16	**IU deferred トラップキュー** SPARC64 VIIIfx では IU deferred トラップキューは不要なので実装していない。	38
17	Reserved.	—
18	**IEEE 754-1985 非準拠時の演算結果** SPARC64 VIIIfx は FSR.NS = 1 のとき、演算結果が非正規化数になるとゼロで置き換える。詳細は Appendix B.6, *"Floating-Point Nonstandard Mode"* を参照。	139
19	**FPU バージョン、FSR.ver** SPARC64 VIIIfx では FSR.ver = 0。	23
20–21	Reserved.	
22	**FPU の TEM, cexc, aexc** SPARC64 VIIIfx は TEM, cexc, and aexc のすべてのビットを実装している。	23
23	**浮動小数点演算のトラップ** SPARC64 VIIIfx では、浮動小数点演算で検出される例外のトラップはすべて precise である。よって FQ は必要ない。	38

TABLE C-1 JPS1 実装依存仕様の SPARC64 VIIIfx での実装 *(3 of 12)*

番号	SPARC64 VIIIfx での実装	ページ
24	**FPU deferred トラップキュー (FQ)** SPARC64 VIIIfx では FQ deferred トラップキューは不要なので実装していない。	38
25	**FQ が存在しない場合の `RDPR` での読みだし** FQ を `RDPR` で読み出そうとすると、*illegal_instruction* が通知される。	38
26–28	Reserved.	—
29	**空間識別子 (ASI) の定義** SPARC64 VIIIfx の定義は Appendix L を参照。	213
30	**ASI 番号の有効ビット数** SPARC64 VIIIfx は 8 ビットすべてをデコードする。	—
31	**致命的なエラーのトラップ** SPARC64 VIIIfx は、命令コミットを監視するウォッチドッグタイマーを実装しており、ある CPU サイクル以上命令がコミットされないと、*async_data_error* 例外を通知しようとする。約 6.7 秒経過すると、プロセッサは `error_state` に遷移する。`error_state` に入る際、`error_state` からの復旧のために WDR リセットを発行するように設定することもできる。	246
32	**Deferred トラップ** SPARC64 VIIIfx は深刻なエラーを報告するトラップを deferred で通知することがある。SPARC64 VIIIfx は deferred トラップキューを実装しない。	44, 255
33	**トラップの精度** 深刻なエラーを報告するトラップ以外で deferred なトラップはない。 SPARC64 VIIIfx では、命令実行の結果発生するトラップはすべて precise である。	44
34	**インタラプトのクリア** インタラプトの詳細は Appendix N を参照。	239
35	**実装依存なトラップ** SPARC64 VIIIfx は実装依存と定義された以下のトラップを実装している。 • *interrupt_vector_trap* (`tt` = 060_{16}) • *PA_watchpoint* (`tt` = 061_{16}) • *VA_watchpoint* (`tt` = 062_{16}) • *ECC_error* (`tt` = 063_{16}) • *fast_instruction_access_MMU_miss* (`tt` = 064_{16} から 067_{16}) • *fast_data_access_MMU_miss* (`tt` = 068_{16} から $06B_{16}$) • *fast_data_access_protection* (`tt` = $06C_{16}$ から $06F_{16}$) • *async_data_error* (`tt` = 040_{16})	51

TABLE C-1 JPS1 実装依存仕様の SPARC64 VIIIfx での実装 *(4 of 12)*

番号	SPARC64 VIIIfx での実装	ページ
36	**トラップの優先順位** SPARC64 VIIIfx の実装依存トラップの優先順位は以下の通りである。 • *interrupt_vector_trap* (priority=16) • *PA_watchpoint* (priority=12) • *VA_watchpoint* (priority=1) • *ECC_error* (priority=33) • *fast_instruction_access_MMU_miss* (priority = 2) • *fast_data_access_MMU_miss* (priority = 12) • *fast_data_access_protection* (priority = 12) • *async_data_error* (priority = 2)	49
37	**リセットトラップ** SPARC64 VIIIfx はパワーオンリセット (POR) とウォッチドッグリセットを実装している。	44
38	**リセットトラップ時の実装依存レジスタへの作用** Appendix O.2, *"RED_state と error_state"* を参照。	247
39	**`error_state` に遷移する固有の条件** ウォッチドッグタイムアウトで約 6.7 秒経過すると、または `TL = MAXTL` で通常のトラップか SIR が発生すると、`error_state`. に遷移する。	44
40	`error_state` SPARC64 VIIIfx は `error_state` に遷移後、ウォッチドッグリセットを発行することもできる。ほとんどのエラーログ用レジスタの状態は保存される (impl. dep. #254 も参照)。	44
41	Reserved.	
42	`FLUSH` **命令** SPARC64 VIIIfx は `FLUSH` 命令をハードウェアで実行する。	—
43	Reserved.	
44	**データアクセスの FPU トラップ** アクセスエラーを起こしたとき、レジスタの値は更新されない。	81
45–46	Reserved.	
47	**`RDASR`** SPARC64 VIIIfx では `rd` = 29–31 でそれぞれ `XAR`, `XASR`, `TXAR` が読み出せる。このとき、 • 命令フィールドの bits 18:0 の扱いは他の `RDASR` と同じである。すなわち、bit<18:14> は `rs1`, bit<13> は `i`。`i=0` のとき、bit<12:5> は *reserved*, bit<4:0> は `rs2` である。`i=1` のとき、bit<12:0> は `simm13` である。 • 特権レジスタは `TXAR` のみ。 reserved が 0 以外でも *illegal_instruction* 例外を検出しない。	97

TABLE C-1 JPS1 実装依存仕様の SPARC64 VIIIfx での実装 *(5 of 12)*

番号	SPARC64 VIIIfx での実装	ページ
48	**WRASR** SPARC64 VIIIfx では rd = 29–31 でそれぞれ XAR, XASR, TXAR に書き込むことができる。このとき、 • 命令フィールドの bits 18:0 の扱いは他の RDASR と同じである。すなわち、bit<18:14> は rs1, bit<13> は i。i=0 のとき、bit<12:5> は *reserved*, bit<4:0> は rs2 である。i=1 のとき、bit<12:0> は simm13 である。 • rs1 と rs2 または simm13 の間の演算は **xor** である。 • 特権レジスタは TXAR のみ。 reserved が 0 以外でも *illegal_instruction* 例外を検出しない。	111
49–54	Reserved.	
55	**浮動小数点演算のアンダーフロー検出** SPARC64 VIIIfx は JPS1 の規定通り、丸め前にアンダーフローを検出する。	—
56–100	Reserved.	
101	**最大トラップレベル** MAXTL = 5.	26
102	**クリーンウィンドウトラップ** SPARC64 VIIIfx は *clean_window* トラップを通知する。レジスタウィンドウはソフトウェアがクリアする。	—
103	**プリフェッチ命令** SPARC64 VIIIfx では PREFETCH 命令の fcn 0–3 と 20–23 は以下のように動作する。 • 特権モードでも PREFETCH 命令は有効に作用する。 • どの fcn も *fast_data_access_MMU_miss* トラップを発生しない。 • プリフェッチはすべてキャッシュライン単位で行われるので、128 バイトアラインされた 128 バイトがプリフェッチされる。 • fcn の特性の説明は Appendix A.49, *"Prefetch Data"* を参照。 • プリフェッチが作用するのは ASI_PRIMARY, ASI_SECONDARY, ASI_NUCLEUS, ASI_PRIMARY_AS_IF_USER, ASI_SECONDARY_AS_IF_USER およびこれらのリトルエンディアン用の ASI に対してである。	95
104	**VER.manuf** VER.manuf = 0004_{16} である。最下位 8 ビットは富士通を示す JEDEC の製造者コードである。	26
105	**TICK レジスタ** SPARC64 VIIIfx は TICK レジスタを 63 ビット幅で実装しており、毎 CPU サイクルカウントアップする。	25
106	**IMPDEP*n* 命令** SPARC64 VIIIfx は VIS1, VIS2 命令を実装するほか、独自の拡張命令を多数実装している。	69
107	**Unimplemented LDD trap** SPARC64 VIIIfx は LDD をハードウェアで実装する。	—

TABLE C-1 JPS1 実装依存仕様の SPARC64 VIIIfx での実装 *(6 of 12)*

番号	SPARC64 VIIIfx での実装	ページ
108	**Unimplemented `STD` trap** SPARC64 VIIIfx は `STD` をハードウェアで実装する。	—
109	***LDDF_mem_address_not_aligned*** SPARC64 VIIIfx では、SIMD でない `LDDF` は、アクセスするアドレスが 8 バイト境界でない 4 バイト境界の場合、*LDDF_mem_address_not_aligned* 例外を通知する。システムソフトウェアはエミュレーションすること。SIMD の `LDDF` では *LDDF_mem_address_not_aligned* ではなく *mem_address_not_aligned* が通知される。	81, 85
110	***STDF_mem_address_not_aligned*** SPARC64 VIIIfx では、SIMD でない `STDF` は、アクセスするアドレスが 8 バイト境界でない 4 バイト境界の場合、*STDF_mem_address_not_aligned* 例外を通知する。システムソフトウェアはエミュレーションすること。SIMD の `STDF` では *STDF_mem_address_not_aligned* ではなく *mem_address_not_aligned* が通知される。	100, 104
111	***LDQF_mem_address_not_aligned*** SPARC64 VIIIfx は `LDQF` を実装していないので、*illegal_instruction* 例外を通知する。*fp_disabled* は検出しない。システムソフトウェアは `LDQF` をエミュレーションすること。	81, 85
112	***STQF_mem_address_not_aligned*** SPARC64 VIIIfx は `STQF` を実装していないので、*illegal_instruction* 例外を通知する。*fp_disabled* は検出しない。システムソフトウェアは `STQF` をエミュレーションすること。	100, 104
113	**実装しているメモリモデル** SPARC64 VIIIfx は `PSTATE.MM` のすべての値に対し Total Store Order (TSO) で動作する。	53
114	**`RED_state` トラップベクタアドレス (`RSTVaddr`)** SPARC64 VIIIfx では `RSTVaddr` は固定であり、 `VA=FFFF FFFF F000 0000`$_{16}$ `PA=01FF F000 0000`$_{16}$ である。	43
115	**`RED_state` プロセッサステート** `RED_state` 時の動作は Section 7.1.1, *"RED_state"* を参照。	43
116	**`SIR_enable` 制御フラグ** JPS1 の規定通り、SPARC64 VIIIfx では `SIR_enable` は存在せず、非特権モードでは `NOP` となる。	—
117	**MMU 無効時のプリフェッチの動作** SPARC64 VIIIfx では、DMMU が無効のとき `PREFETCH` はメモリアクセスをせずに実行完了する。ノンフォールティングロードは、JPS1 **Commonality** の Section F.5 の定義通り *data_access_exception* が通知される。	182
118	**I/O 領域の識別方法** この項目は本仕様書の記載範囲外である。SPARC64 VIIIfx を搭載するシステムの仕様書を参照。	—

TABLE C-1 JPS1 実装依存仕様の SPARC64 VIIIfx での実装 *(7 of 12)*

番号	SPARC64 VIIIfx での実装	ページ
119	**PSTATE.MM の未定義値** PSTATE.MM に 11_2 を指定すると、lTSO で動作する。しかしながら、11_2 は SPARC64 VIIIfx の将来のバージョンでは別の意味になるかもしれないので、使うべきではない。	54
120	**プロセッサと I/O DMA 間のメモリ操作の同一性・原子性** この項目は本仕様書の記載範囲外である。SPARC64 VIIIfx を搭載するシステムの仕様書を参照。	—
121	**実装依存メモリモデル** SPARC64 VIIIfx は E ビット (Volatile) がセットされているページに対するアクセスをプログラムオーダで処理する。	—
122	**FLUSH レイテンシ** FLUSH 命令はプロセッサ内のキャッシュの状態を同期させるので、処理にかかる時間はプロセッサの状態に依存する。先行する命令がすべて完了しているとき、FLUSH 命令のレイテンシは 30CPU サイクルである。	54
123	**I/O のセマンティクス** この項目は本仕様書の記載範囲外である。SPARC64 VIIIfx を搭載するシステムの仕様書を参照。	—
124	**TL > 0 のとき暗黙に使用される ASI** JPS1 の規定通り、TL > 0 では PSTATE.CLE に応じて ASI_NUCLEUS または ASI_NUCLEUS_LITTLE が使われる。	—
125	**アドレスマスク** PSTATE.AM = 1 のとき、SPARC64 VIIIfx は PC の上位 32 ビットをマスクしてデスティネーションレジスタに転送する。	42, 68, 80
126	**レジスタウィンドウ関連レジスタのビット幅** SPARC64 VIIIfx では NWINDOWS は 8 である。よって、CWP, CANSAVE, CANRESTORE, OTHERWIN の有効ビットは 3 ビットである。これらのレジスタに NWINDOWS – 1 より大きな値を設定しようとすると、下 3 ビット以外を無視した値がセットされる。CLEANWIN レジスタは 3 ビットを保持する。	—
127–201	Reserved.	
202	***fast_ECC_error* トラップ** SPARC64 VIIIfx は *fast_ECC_error* トラップを生成しない。	—
203	**DCR のビット 13:6 とビット 1** SPARC64 VIIIfx は DCR を実装しない。	29
204	**DCR のビット 5:3 とビット 0** SPARC64 VIIIfx は DCR を実装しない。	29
205	**命令トラップレジスタ** SPARC64 VIIIfx は命令トラップレジスタを JPS1 の定義通りに実装する。	37
206	**SHUTDOWN 命令** SPARC64 VIIIfx は特権モードで SHUTDOWN 命令を NOP として実行する。	99

TABLE C-1 JPS1 実装依存仕様の SPARC64 VIIIfx での実装 *(8 of 12)*

番号	SPARC64 VIIIfx での実装	ページ
207	**PCR のビット 47:32、ビット 26:17、ビット 3** SPARC64 VIIIfx ではこれらのビットには以下の機能が実装されている。 • ビット 47:32 は、オーバーフローの表示と設定 (OVF)。 • ビット 26 は OVF の書き込み制御 (OVRO)。 • ビット 24:22 はカウンタ数 (NC)。 • ビット 20:18 はカウンタ選択 (SC)。 • ビット 3 は SU/SL の書き込み制御 (ULRO)。 他の実装依存ビットは読み出しには 0 が返り、書き込みは無視される。	26
208	**命令実行とエラーの通知順序** SPARC64 VIIIfx ではプログラム順にエラーが通知される。	255
209	**ソフトウェアが介入する必要があるエラーのトラップ** SPARC64 VIIIfx では命令実行に同期して起こるエラーは precise で通知される。	—
210	**ERROR 出力信号** この項目は本仕様書の記載範囲外である。SPARC64 VIIIfx システム制御仕様書を参照。	—
211	**エラー情報表示レジスタの値** SPARC64 VIIIfx では致命的エラーの要因は ASI_STCHG_ERR_INFO レジスタに表示されない。	272
212	**致命的エラーのトラップ** SPARC64 VIIIfx では致命的エラーはトラップを発生しない。	272
213	**AFSR.PRIV** SPARC64 VIIIfx は AFSR.PRIV ビットを実装しない。	285
214	**deferred トラップの有効・無効制御** SPARC64 VIIIfx には deferred トラップを制御する機能はない。	—
215	**エラーバリア** —	—
216	***data_access_error* トラップの精度** SPARC64 VIIIfx では *data_access_error* は常に precise である。	—
217	***instruction_access_error* トラップの精度** SPARC64 VIIIfx では *instruction_access_error* は常に precise である。	—
218	***async_data_error*** SPARC64 VIIIfx は *async_data_error* トラップを TT = 40_{16} で生成する。	45, 255
219	**AFSR** SPARC64 VIIIfx は AFAR を実装しない。	—
220	**その他のエラー表示・制御レジスタ** SPARC64 VIIIfx は RAS 機能を充実させ高い信頼性を保証している。詳細は Appendix P を参照。	255
221	**Special/signalling ECCs** —	—

TABLE C-1 JPS1 実装依存仕様の SPARC64 VIIIfx での実装 *(9 of 12)*

番号	SPARC64 VIIIfx での実装	ページ
222	**TLB 構成** SPARC64 VIIIfx の TLB 構成は以下のようになっている。 • レベル 1 マイクロ iTLB (uITLB)、フルアソシエイティブ • レベル 1 マイクロ dTLB (uDTLB)、フルアソシエイティブ • レベル 2 IMMU TLB—sITLB (セットアソシエイティブ) と fITLB (フルアソシエイティブ) からなる • レベル 2 DMMU TLB—sDTLB (セットアソシエイティブ) と fDTLB (フルアソシエイティブ) からなる	173
223	**TLB マルチヒット検出** SPARC64 VIIIfx では、マイクロ TLB をミスしメイン fTLB にアクセスした場合のみマルチヒットを検出する。	174
224	**MMU 物理アドレスビット幅** SPARC64 VIIIfx では物理メモリのアドレス空間は 41 ビットである。TTE の PA フィールドは 41 ビットのみを格納し、PA<42:41> は読み出しは 0 が読み出され、書き込みは無視される。	176
225	**TLB ロックエントリ** ロックビットがセットされた TTE が TLB Data In で書かれると、SPARC64 VIIIfx はそのエントリを fTLB に書き込みロックする。それ以外では、ページサイズに応じて sTLB か fTLB に書かれる。	176
226	**TTE.CV ビット** SPARC64 VIIIfx ではどの TLB も CV を実装していない。SPARC64 VIIIfx はハードウェアでキャッシュエイリアスを解消する機構を備えている。#232 も参照。	176
227	**TSB エントリ数** SPARC64 VIIIfx 仕様では TSB をサポートしないので、この実装依存仕様は意味を持たない。	—
228	**TSB、コンテキスト ID どちらが TSB_Hash に使われるか** SPARC64 VIIIfx 仕様では TSB をサポートしないので、この実装依存仕様は意味を持たない。	—
229	**TSB_Base アドレス生成** SPARC64 VIIIfx 仕様では TSB をサポートしないので、この実装依存仕様は意味を持たない。	—
230	***data_access_exception* トラップ** SPARC64 VIIIfx は JPS1 **Commonality** の Appendix F.5 で挙げられた要因で *data_access_exception* 例外を通知する。	178
231	**MMU 物理アドレスビット幅の変動** SPARC64 VIIIfx では物理アドレスのビット幅は 41 ビットで一定である。	182
232	**DCUCR の CP, CV ビット** SPARC64 VIIIfx は DCUCR の CP, CV ビットを実装しない。impl. dep. #226 も参照。	34, 182

TABLE C-1 JPS1 実装依存仕様の SPARC64 VIIIfx での実装 *(10 of 12)*

番号	SPARC64 VIIIfx での実装	ページ
233	**TSB_Hash フィールド** SPARC64 VIIIfx では TSB Extension レジスタを定義していないので、この実装依存仕様は意味を持たない。	183
234	**TLB 入れ替えアルゴリズム** fTLB は擬似 LRU、sTLB は LRU である。	192
235	**TLB データアクセスのアドレス割り当て** Appendix F.10.4 参照。	192
236	**TSB_Size フィールド幅** SPARC64 VIIIfx では TSB_Size は 4 ビットで、bit<3:0> を占める。 TSB_Size に書いた値は、読みだしによりその値のまま読みだされる。 SPARC64 VIIIfx は値を保持するだけで、この値を何の計算にも使わない。	193
237	**JMPL/RETURN で *mem_address_not_aligned* が起きたときの DSFAR/DSFSR** JMPL または RETURN で *mem_address_not_aligned* が起きた場合、D-SFAR, D-SFSR は更新されない。	80, 178, 194
238	**TLB はラージページのページ内オフセットを保存するかどうか** SPARC64 VIIIfx ではページオフセットは書き込み時に捨てられ、読み出しでは不定データが読み出される。	176
239	**ASI レジスタ 55_{16} または $5D_{16}$ のアクセス** SPARC64 VIIIfx では、IMMU ASI 55_{16}, DMMU ASI $5D_{16}$ の VA<63:18> は無視される。	183
240	**DCUCR のビット 47:41** SPARC64 VIIIfx はビット 41 を、メモリの投機アクセスを制御する WEAK_SPCA として実装する。	34
241	**アドレスマスクと DSFAR** PSTATE.AM = 1 のとき、SPARC64 VIIIfx は DSFAR の上位 32 ビットに 0 を書く。	?
242	**TLB ロックビット** fITLB と fDTLB はロックビットをサポートする。sITLB と sDTLB では 0 が読み出される。	176
243	**インタラプト送信ステータスレジスタの BUSY/NACK の組** SPARC64 VIIIfx では 8 組の BUSY/NACK が実装される。	242
244	**データウォッチポイントの信頼性** データウォッチポイントの信頼性を下げるような機構は実装されていない。	36
245	**命令キャッシュ内での CALL、分岐命令の飛び先の保存形式** SPARC64 VIIIfx では CALL、分岐 (BPcc, FBPfcc, Bicc, BPr) 命令の低位 11 ビットはメモリ内の形式通りに保存される。	?
246	**インタラプト送信レジスタアクセス時の VA<38:29>** インタラプト送信レジスタにアクセスする際、SPARC64 VIIIfx は VA<38:29> の全 10 ビットを無視する。	242
247	**インタラプト受信レジスタの SID フィールド** SID_H と SID_L の値は未定義である。	243

TABLE C-1 JPS1 実装依存仕様の SPARC64 VIIIfx での実装 *(11 of 12)*

番号	SPARC64 VIIIfx での実装	ページ
248	***fp_exception_other* が *unfinished_FPop* で通知される条件** SPARC64 VIIIfx は JPS1 **Commonality** Section 5.1.7 で既定された条件で *unfinished_FPop* の *fp_exception_other* を通知する。	23
249	**パーシャルストアのデータウォッチポイント検出** SPARC64 VIIIfx ではウォッチポイントの検出は保守的に行われる。DCUCR のデータマスクは 0 かどうかだけが検査される。バイトマスク (r[rs2]) は無視され、マスクが 0 であっても (ストアが行われない場合でも) ウォッチポイントを検出する。	93
250	**PSTATE.PRIV = 0 で PCR のアクセスが可能かどうか** SPARC64 VIIIfx では、PSTATE.PRIV = 0 のとき PCR にアクセスできるかどうかは PCR.PRIV によって決まる。PSTATE.PRIV = 0 かつ PCR.PRIV = 1 のとき、RDPCR や WRPCR 命令を実行しようとすると、*privileged_action* が通知される。PSTATE.PRIV = 0 かつ PCR.PRIV = 0 のときは、RDPCR は正常に実行され、WRPCR は PCR.PRIV を変更しようとする (つまり 1 を書き込もうとする) 場合のみ、*privileged_action* を通知する。	26, 28, 97
251	Reserved.	—
252	**DCUCR.DC(データキャッシュ有効 / 無効設定)** SPARC64 VIIIfx は DCUCR.DC を実装していない。	34
253	**DCUCR.IC 命令キャッシュ有効 / 無効設定)** SPARC64 VIIIfx は DCUCR.IC を実装していない。	34
254	**error_state から復旧する手段** 通常の SPARC64 VIIIfx の動作では、error_state に遷移後 *watchdog_reset* (WDR) を発生しリセットする。しかし、OPSR による設定で、error_state に遷移後その状態にとどまるようにすることもできる。	44, 253
255	**LDDFA で ASI $E0_{16}$ または $E1_{16}$ を指定し、レジスタ番号の条件が間違っている場合** デスティネーションレジスタ番号を間違っていてもトラップは通知されない。	220
256	**LDDFA で ASI $E0_{16}$ または $E1_{16}$ を指定し、メモリアドレスのアラインメントが間違っている場合** SPARC64 VIIIfx は以下のように動作する。 • 8 バイト境界なら、アラインメント例外は通知されず、*data_access_exception* が通知される。 • 4 バイト境界なら、*LDDF_mem_address_not_aligned* が通知される。 • それ以外なら、*mem_address_not_aligned* が通知される。	220
257	**LDDFA で ASI $C0_{16}$–$C5_{16}$ または $C8_{16}$–CD_{16} を指定し、メモリアドレスのアラインメントが間違っている場合** SPARC64 VIIIfx は以下のように動作する。 • 8 バイト境界なら、アラインメント例外は通知されず、*data_access_exception* が通知される。 • 4 バイト境界なら、*LDDF_mem_address_not_aligned* が通知される。 • それ以外なら、*mem_address_not_aligned* が通知される。	220

TABLE C-1 JPS1 実装依存仕様の SPARC64 VIIIfx での実装 *(12 of 12)*

番号	SPARC64 VIIIfx での実装	ページ
258	**`ASI_SERIAL_ID`** SPARC64 VIIIfx では個々のプロセッサの識別コードを返す。	220

F.APPENDIX **D**

Formal Specification of the Memory Models

JPS1 **Commonality** の Appendix D を参照。

F.APPENDIX **E**

Opcode Maps

Appendix E は、JPS1 **Commonality** と HPC-ACE 拡張命令の定義を含む完全な情報を提供する。

表中、横棒 (—) になっているオペコードは予約 *reserved* 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、例外が通知される。詳細は JPS1 **Commonality** の Section 6.3.9, *Reserved Opcodes and Instruction Fields* を参照。

この章および Appendix A のオペコードには、右肩に記号が付されたものがある。この記号の意味は TABLE A-1 (58 ページ) に示してある。非推奨命令 (deprecated) については JPS1 **Commonality** の Section A.71, *"Deprecated Instructions"* を参照。

この章の表において、予約オペコード (—) と網掛けのエントリは SPARC64 VIIIfx では実装されていない。

TABLE E-1 op<1:0>

op <1:0>			
0	**1**	**2**	**3**
分岐と SETHI TABLE E-2 参照。	CALL	算術演算その他 TABLE E-3 参照。	メモリアクセス TABLE E-4 参照。

TABLE E-2 op2<2:0> (op = 0)

op2 <2:0>							
0	**1**	**2**	**3**	**4**	**5**	**6**	**7**
ILLTRAP	BPcc – TABLE E-7 参照。	Bicc[D]– TABLE E-7 参照。	BPr – TABLE E-8 参照。	SETHI NOP[†]	FBPfcc – TABLE E-7 参 照。	FBfcc[D]– TABLE E-7 参 照。	SXAR

†rd = 0, imm22 = 0

ILLTRAP を実行すると *illegal_instruction* 例外が通知される。

TABLE E-3 op3<5:0> (op = 2)

op3<3:0>	op3 <5:4>			
	0	**1**	**2**	**3**
0	ADD	ADDcc	TADDcc	WRY[D] (rd = 0) — (rd = 1) WRCCR (rd = 2) WRASI (rd = 3) — (rd = 4, 5) WRFPRS (rd = 6) WRPCR[P$_{PCR}$] (rd = 16) WRPIC[P$_{PIC}$] (rd = 17) WRDCR[P] (rd = 18) WRGSR (rd = 19) WRSOFTINT_SET[P] (rd = 20) WRSOFTINT_CLR[P] (rd = 21) WRSOFTINT[$\overline{P}$] (rd = 22) WRTICK_CMPR[P] (rd = 23) WRSTICK[P] (rd = 24) WRSTICK_CMPR[P] (rd = 25) WRXAR (rd = 29) WRXASR (rd = 30) WRTXAR[P] (rd = 31) SIR (rd = 15, rs1 = 1, i = 1)
1	AND	ANDcc	TSUBcc	SAVED[P] (fcn = 0) RESTORED[P] (fcn = 1)
2	OR	ORcc	TADDccTV[D]	WRPR[P]
3	XOR	XORcc	TSUBccTV[D]	—
4	SUB	SUBcc	MULScc[D]	FPop1 – TABLE E-5 参照。
5	ANDN	ANDNcc	SLL (x = 0), SLLX (x = 1)	FPop2 – TABLE E-6 参照。
6	ORN	ORNcc	SRL (x = 0), SRLX (x = 1)	IMPDEP1 (VIS) – TABLE E-12 および TABLE E-13 参照。
7	XNOR	XNORcc	SRA (x = 0), SRAX (x = 1)	IMPDEP2 (FMADD/SUB, etc.) – TABLE E-14 参照。

TABLE E-3 op3<5:0> (op = 2)

op3<3:0>	op3 <5:4>			
	0	**1**	**2**	**3**
8	ADDC	ADDCcc	RDY[D] (rs1 = 0) — (rs1 = 1) RDCCR (rs1 = 2) RDASI (rs1 = 3) RDTICKP[NPT] (rs1 = 4) RDPC (rs1 = 5) RDFPRS (rs1 = 6) RDPCR[PPCR] (rs1 = 16) RDPIC[PPIC] (rs1 = 17) RDDCR[P] (rs1 = 18) RDGSR (rs1 = 19) RDSOFTINT[P] (rs1 = 22) RDTICK_CMPR[P] (rs1 = 23) RDSTICK[PNPT] (rs1 = 24) RDSTICK_CMPR[P] (rs1 = 25) RDXASR (rs1 = 30) RDTXAR[P] (rs1 = 31) MEMBAR (rs1 = 15, rd = 0, i = 1) STBAR[D] (rs1 = 15, rd = 0, i = 0)	JMPL
9	MULX	—	—	RETURN
A	UMUL[D]	UMULcc[D]	RDPR[P]	Tcc – TABLE E-7 参照。
B	SMUL[D]	SMULcc[D]	FLUSHW	FLUSH
C	SUBC	SUBCcc	MOVcc	SAVE
D	UDIVX	—	SDIVX	RESTORE
E	UDIV[D]	UDIVcc[D]	POPC (rs1 = 0) — (rs1 > 0)	DONE[P] (fcn = 0) RETRY[P] (fcn = 1)
F	SDIV[D]	SDIVcc[D]	MOVr TABLE E-8 参照。	—

TABLE E-4 op3<5:0> (op = 3)

op3<3:0>	op3 <5:4>			
	0	**1**	**2**	**3**
0	LDUW	LDUWA[PASI]	LDF	LDFA[PASI]
1	LDUB	LDUBA[PASI]	LDFSR[D], LDXFSR	—
2	LDUH	LDUHA[PASI]	LDQF	LDQFA[PASI]
3	LDD[D]	LDDA[D, PASI]	LDDF	LDDFA[PASI]

TABLE E-4 op3<5:0> (op = 3) *(Continued)*

op3<3:0>	op3 <5:4>			
	0	**1**	**2**	**3**
4	STW	STWA$^{P_{ASI}}$	STF	STFA$^{P_{ASI}}$
5	STB	STBA$^{P_{ASI}}$	STFSR^{D}, STXFSR	—
6	STH	STHA$^{P_{ASI}}$	STQF	STQFA$^{P_{ASI}}$
7	STD^{D}	STDA$^{P_{ASI}}$	STDF	STDFA$^{P_{ASI}}$
8	LDSW	LDSWA$^{P_{ASI}}$	—	—
9	LDSB	LDSBA$^{P_{ASI}}$	—	—
A	LDSH	LDSHA$^{P_{ASI}}$	—	—
B	LDX	LDXA$^{P_{ASI}}$	—	—
C	—	—	STFR	CASA$^{P_{ASI}}$
D	LDSTUB	LDSTUBA$^{P_{ASI}}$	PREFETCH	PREFETCHA$^{P_{ASI}}$
E	STX	STXA$^{P_{ASI}}$	—	CASXA$^{P_{ASI}}$
F	SWAP^{D}	SWAPA$^{D, P_{ASI}}$	STDFR	—

SPARC64 VIIIfx では LDQF, LDQFA, STQF, STQFA と *reserved* (—) オペコードを実行すると *illegal_instruction* 例外を通知する。

TABLE E-5 opf<8:0> (op = 2, op3 = 34_{16} = FPop1)

opf<8:3>	opf<2:0>							
	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**
00_{16}	—	FMOVs	FMOVd	FMOVq	—	FNEGs	FNEGd	FNEGq
01_{16}	—	FABSs	FABSd	FABSq	—	—	—	—
02_{16}	—	—	—	—	—	—	—	—
03_{16}	—	—	—	—	—	—	—	—
04_{16}	—	—	—	—	—	—	—	—
05_{16}	—	FSQRTs	FSQRTd	FSQRTq	—	—	—	—
06_{16}	—	—	—	—	—	—	—	—
07_{16}	—	—	—	—	—	—	—	—
08_{16}	—	FADDs	FADDd	FADDq	—	FSUBs	FSUBd	FSUBq
09_{16}	—	FMULs	FMULd	FMULq	—	FDIVs	FDIVd	FDIVq

TABLE E-5 opf<8:0> (op = 2, op3 = 34_{16} = FPop1) *(Continued)*

opf<8:3>	opf<2:0>							
	0	1	2	3	4	5	6	7
$0A_{16}$	—	—	—	—	—	—	—	—
$0B_{16}$	—	—	—	—	—	—	—	—
$0C_{16}$	—	—	—	—	—	—	—	—
$0D_{16}$	—	FsMULd	—	—	—	—	FdMULq	—
$0E_{16}$	—	—	—	—	—	—	—	—
$0F_{16}$	—	—	—	—	—	—	—	—
10_{16}	—	FsTOx	FdTOx	FqTOx	FxTOs	—	—	—
11_{16}	FxTOd	—	—	—	FxTOq	—	—	—
12_{16}	—	—	—	—	—	—	—	—
13_{16}	—	—	—	—	—	—	—	—
14_{16}	—	—	—	—	—	—	—	—
15_{16}	—	—	—	—	—	—	—	—
16_{16}	—	—	—	—	—	—	—	—
17_{16}	—	—	—	—	—	—	—	—
18_{16}	—	—	—	—	FiTOs	—	FdTOs	FqTOs
19_{16}	FiTOd	FsTOd	—	FqTOd	FiTOq	FsTOq	FdTOq	—
$1A_{16}$	—	FsTOi	FdTOi	FqTOi	—	—	—	—
$1B_{16}$–$3F_{16}$	—	—	—	—	—	—	—	—

網掛けの部分と *reserved* (—) は、SPARC64 VIIIfx では *fp_exception_other* 例外を ftt = *unimplemented_FPop* で通知する。

TABLE E-6 opf<8:0> (op = 2, op3 = 35_{16} = FPop2)

opf<8:4>	opf<3:0>								
	0	1	2	3	4	5	6	7	8–F
00_{16}	—	FMOVs (fcc0)	FMOVd (fcc0)	FMOVq (fcc0)	—	†	†	†	—
01_{16}	—	—	—	—	—	—	—	—	—
02_{16}	—	—	—	—	—	FMOVsZ	FMOVdZ	FMOVqZ	—
03_{16}	—	—	—	—	—	—	—	—	—
04_{16}	—	FMOVs (fcc1)	FMOVd (fcc1)	FMOVq (fcc1)	—	FMOVsLEZ	FMOVdLEZ	FMOVqLEZ	—

TABLE E-6 opf<8:0> (op = 2, op3 = 35_{16} = FPop2) *(Continued)*

opf<8:4>	opf<3:0>								
	0	1	2	3	4	5	6	7	8–F
05_{16}	—	FCMPs	FCMPd	FCMPq	—	FCMPEs	FCMPEd	FCMPEq	—
06_{16}	—	—	—	—	—	FMOVsLZ	FMOVdLZ	FMOVqLZ	—
07_{16}	—	—	—	—	—	—	—	—	—
08_{16}	—	FMOVs (fcc2)	FMOVd (fcc2)	FMOVq (fcc2)	—	†	†	†	—
09_{16}	—	—	—	—	—	—	—	—	—
$0A_{16}$	—	—	—	—	—	FMOVsNZ	FMOVdNZ	FMOVqNZ	—
$0B_{16}$	—	—	—	—	—	—	—	—	—
$0C_{16}$	—	FMOVs (fcc3)	FMOVd (fcc3)	FMOVq (fcc3)	—	FMOVsGZ	FMOVdGZ	FMOVqGZ	—
$0D_{16}$	—	—	—	—	—	—	—	—	—
$0E_{16}$	—	—	—	—	—	FMOVsGEZ	FMOVdGEZ	FMOVqGEZ	—
$0F_{16}$	—	—	—	—	—	—	—	—	—
10_{16}	—	FMOVs (icc)	FMOVd (icc)	FMOVq (icc)	—	—	—	—	—
11_{16}–17_{16}	—	—	—	—	—	—	—	—	—
18_{16}	—	FMOVs (xcc)	FMOVd (xcc)	FMOVq (xcc)	—	—	—	—	—
19_{16}–$1F_{16}$	—	—	—	—	—	—	—	—	—

†FMOVR に予約されたオペコード

網掛けの部分と *reserved* (—) は、SPARC64 VIIIfx では *fp_exception_other* 例外を ftt = *unimplemented_FPop* で通知する。

TABLE E-7 cond<3:0>

cond<3:0>	BPcc	Bicc[D]	FBPfcc	FBfcc[D]	Tcc
	op = 0 op2 = 1	op = 0 op2 = 2	op = 0 op2 = 5	op = 0 op2 = 6	op = 2 op3 = $3A_{16}$
0	BPN	BN[D]	FBPN	FBN[D]	TN
1	BPE	BE[D]	FBPNE	FBNE[D]	TE
2	BPLE	BLE[D]	FBPLG	FBLG[D]	TLE
3	BPL	BL[D]	FBPUL	FBUL[D]	TL

TABLE E-7 cond<3:0>

cond<3:0>	BPcc	Bicc[D]	FBPfcc	FBfcc[D]	Tcc
	op = 0 op2 = 1	op = 0 op2 = 2	op = 0 op2 = 5	op = 0 op2 = 6	op = 2 op3 = $3A_{16}$
4	BPLEU	BLEU[D]	FBPL	FBL[D]	TLEU
5	BPCS	BCS[D]	FBPUG	FBUG[D]	TCS
6	BPNEG	BNEG[D]	FBPG	FBG[D]	TNEG
7	BPVS	BVS[D]	FBPU	FBU[D]	TVS
8	BPA	BA[D]	FBPA	FBA[D]	TA
9	BPNE	BNE[D]	FBPE	FBE[D]	TNE
A	BPG	BG[D]	FBPUE	FBUE[D]	TG
B	BPGE	BGE[D]	FBPGE	FBGE[D]	TGE
C	BPGU	BGU[D]	FBPUGE	FBUGE[D]	TGU
D	BPCC	BCC[D]	FBPLE	FBLE[D]	TCC
E	BPPOS	BPOS[D]	FBPULE	FBULE[D]	TPOS
F	BPVC	BVC[D]	FBPO	FBO[D]	TVC

TABLE E-8 命令ワードの rcond<2:0> のエンコーディング

		BPr	MOVr	FMOVr
		op = 0 op2 = 3	op = 2 op3 = $2F_{16}$	op = 2 op3 = 35_{16}
rcond <2:0>	0	—	—	—
	1	BRZ	MOVRZ	FMOVRZ
	2	BRLEZ	MOVRLEZ	FMOVRLEZ
	3	BRLZ	MOVRLZ	FMOVRLZ
	4	—	—	—
	5	BRNZ	MOVRNZ	FMOVRNZ
	6	BRGZ	MOVRGZ	FMOVRGZ
	7	BRGEZ	MOVRGEZ	FMOVRGEZ

TABLE E-9 cc / opf_cc フィールド (MOVcc および FMOVcc)

opf_cc			選択される条件フィールド
cc2	cc1	cc0	
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	—

TABLE E-10 cc フィールド (FBPfcc, FCMP および FCMPE)

cc1	cc0	選択される条件フィールド
0	0	fcc0
0	1	fcc1
1	0	fcc2
1	1	fcc3

TABLE E-11 cc フィールド (BPcc および Tcc)

cc1	cc0	選択される条件フィールド
0	0	icc
0	1	—
1	0	xcc
1	1	—

TABLE E-12 IMPDEP1: VIS 命令（op = 2, op3 = 36_{16}) の opf<8:0>, ただし 0 ≤ opf<8:4> ≤ 7

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	—	—	FMUL 8x16	—	FPADD16S	FZEROS	FANDS
2_{16}	EDGE8L	ARRAY16	FCMPNE16	—	—	FPADD32	FNOR	FXNOR
3_{16}	EDGE8LN	—	—	FMUL 8x16AU	—	FPADD32S	FNORS	FXNORS
4_{16}	EDGE16	ARRAY32	FCMPLE32	—	—	FPSUB16	FANDNOT2	FSRC1
5_{16}	EDGE16N	—	—	FMUL 8x16AL	—	FPSUB16S	FANDNOT2S	FSRC1S
6_{16}	EDGE16L	—	FCMPNE32	FMUL 8SUx16	—	FPSUB32	FNOT2	FORNOT2
7_{16}	EDGE16LN	—	—	FMUL 8ULx16	—	FPSUB32S	FNOT2S	FORNOT2S

TABLE E-12 IMPDEP1: VIS 命令（op = 2, op3 = 36_{16}) の opf<8:0>, ただし 0 ≤ opf<8:4> ≤ 7

opf<3:0>	opf<8:4>							
	00_{16}	01_{16}	02_{16}	03_{16}	04_{16}	05_{16}	06_{16}	07_{16}
8_{16}	EDGE32	ALIGN ADDRESS	FCMPGT16	FMULD 8SUx16	FALIGNDATA	—	FANDNOT1	FSRC2
9_{16}	EDGE32N	BMASK	—	FMULD 8ULx16	—	—	FANDNOT1S	FSRC2S
A_{16}	EDGE32L	ALIGN ADDRESS _LITTLE	FCMPEQ16	FPACK32	—	—	FNOT1	FORNOT1
B_{16}	EDGE32LN	—	—	FPACK16	FPMERGE	—	FNOT1S	FORNOR1S
C_{16}	—	—	FCMPGT32	—	BSHUFFLE	—	FXOR	FOR
D_{16}	—	—	—	FPACKFIX	FEXPAND	—	FXORS	FORS
E_{16}	—	—	FCMPEQ32	PDIST	—	—	FNAND	FONE
F_{16}	—	—	—	—	—	—	FNANDS	FONES

TABLE E-13 IMPDEP1: VIS 命令（op = 2, op3 = 36_{16}) の opf<8:0>, ただし 08_{16} ≤ opf<8:4> ≤ $1F_{16}$

opf<3:0>	opf<8:4>				
	08_{16}	09_{16}–15_{16}	16_{16}	17_{16}	18_{16}–$1F_{16}$
0_{16}	SHUTDOWN	—	FCMPEQd	FMAXd	—
1_{16}	SIAM	—	FCMPEQs	FMAXs	—
2_{16}	SUSPEND^{P}	—	FCMPEQEd	FMINd	—
3_{16}	SLEEP	—	FCMPEQEs	FMINs	—
4_{16}	—	—	FCMPLEEd	FRCPAd	—
5_{16}	—	—	FCMPLEEs	FRCPAs	—
6_{16}	—	—	FCMPLTEd	FRSQRTAd	—
7_{16}	—	—	FCMPLTEs	FRSQRTAs	—
8_{16}	—	—	FCMPNEd	FTRISSELd	—
9_{16}	—	—	FCMPNEs	—	—
A_{16}	—	—	FCMPNEEd	FTRISMULd	—
B_{16}	—	—	FCMPNEEs	—	—
C_{16}	—	—	FCMPGTEd	—	—
D_{16}	—	—	FCMPGTEs	—	—

TABLE E-13 IMPDEP1: VIS 命令（op = 2, op3 = 36_{16}) の opf<8:0>, ただし $08_{16} \le$ opf<8:4> $\le 1F_{16}$

opf<3:0>	opf<8:4>				
	08_{16}	09_{16}–15_{16}	16_{16}	17_{16}	18_{16}–$1F_{16}$
E_{16}	—	—	FCMPGEEd	—	—
F_{16}	—	—	FCMPGEEs	—	—

TABLE E-14 IMPDEP2 (op = 2, op3 = 37_{16})

size	var			
	00_{02}	01_{02}	10_{02}	11_{02}
00_{02}	FPMADDX	FPMADDXHI	FTRIMADDd	FSELMOVd
01_{02}	FMADDs	FMSUBs	FNMSUBs	FNMADDs
10_{02}	FMADDd	FMSUBd	FNMSUBd	FNMADDd
11_{02}	(4 倍精度演算に予約されている)			FSELMOVs

F.APPENDIX **F**

Memory Management Unit

本章では、SPARC64 VIIIfx の MMU の実装依存部の定義と、SPARC64 VIIIfx で拡張された機能について説明する。SPARC64 VIIIfx の MMU には JPS1 仕様と非互換な部分がある。詳細は以下を参照。

- Section F.4, *"Hardware Support for TSB Access"*
- Section F.10, *"Internal Registers and ASI Operations"*

F.1 Virtual Address Translation

IMPL. DEP. #222: TLB organization is JPS1 implementation dependent.

SPARC64 VIIIfx の TLB は 2 階層構成となっている。

- レベル 1 マイクロ ITLB (uITLB)、フルアソシエイティブ
- レベル 1 マイクロ DTLB (uDTLB)、フルアソシエイティブ
- レベル 2 IMMU-TLB、これはセットウェイアソシエイティブの sITLB と、フルアソシエイティブの fITLB からなる。
- レベル 2 DMMU-TLB、これはセットウェイアソシエイティブの sDTLB と、フルアソシエイティブの fDTLB からなる。

TABLE F-1 に SPARC64 VIIIfx の各 TLB の構成を示す。

uITLB と uDTLB は、それぞれ IMMU-TLB、DMMU-TLB の一時的な記憶領域として使われる。マイクロ TLB に対し、IMMU-TLB、DMMU-TLB をメイン TLB と呼ぶこともある。マイクロ TLB の内容はメイン TLB のサブセットとなっており、ハードウェアが同一性を保証する。

マイクロ TLB はソフトウェアから直接操作することはできず、また、ソフトウェアの動作が変わるような影響を与えることも、TLB の複数エントリがヒットした場合を除いて、ない。本仕様書ではマイクロ TLB についてはこれ以上触れない。

TABLE F-1 SPARC64 VIIIfx の TLB 構成

諸元	sITLB と sDTLB	fITLB と fDTLB
エントリ数	256 (sITLB), 512 (sDTLB)	16
構成	2 ウェイセットアソシエイティブ	フルアソシエイティブ
エントリをロックできるか	できない	できる
ページサイズ	二種類	すべてのページサイズ

IMPL. DEP. #223: Whether TLB multiple-hit detection is supported in a JPS1 processor is implementation dependent.

SPARC64 VIIIfx の MMU は多重ヒットを検出するが、検出できるのはメイン TLB でも fTLB 内で起きた多重ヒットのみである。fTLB に多重ヒットエントリがあるときでも、マイクロ TLB にヒットしている間は検出できない。詳細は Appendix F.5.2, *"Behavior on TLB Error"* (page 180) を参照。

F.2 Translation Table Entry (TTE)

変換テーブルエントリ (Translation Table Entry:TTE) は、論理アドレスである連続した領域 (ページ) から物理アドレスへの対応と、そのページの属性を保持するエントリである。TTE は、タグ部とデータ部という 2 つの 64 ビットのデータからなる。タグ部は検索用の情報が入っており、一致している場合はデータ部にあるページ情報がアドレス変換に使われる。

SPARC JPS1 では、TTE は TSB のエントリであり、TLB のエントリの操作も同じフォーマットで操作するという定義になっている。SPARC64 VIIIfx は TSB をハードウェアでサポートしていないが、TLB の操作はこのフォーマットで行うので、JPS1 **Commonality** の定義を FIGURE F-1, TABLE F-2 に示す。

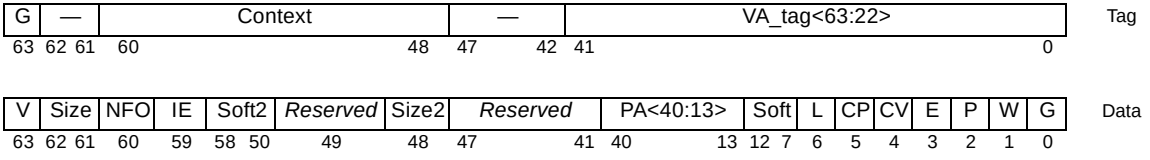


FIGURE F-1 変換テーブルエントリ (TTE)

TABLE F-2 TTE のビット説明 *(1 of 3)*

ビット位置	フィールド名	説明
Tag – 63	G	グローバル。G に 1 がセットされていると、TLB 検索時に Context フィールドは使われない。ある CPU 上で動作している、ユーザ・スーパバイザを問わずすべてのコンテキストが、G = 1 のエントリを共有することになる。ソフトウェアが TLB ミスを効率的に処理できるよう、G ビットは TTE のタグとデータの両方にある。
Tag – 60:48	Context	この TTE のコンテキスト番号。
Tag – 41:0	VA_tag	論理アドレスタグ。論理アドレスのページ番号。
Data – 63	V	バリッド。V に 1 がセットされていると、TTE の他のフィールドの値が意味を持つ。**Note:** V ビットがなくても、ソフトウェアのコンベンションとして使わない TTE を設定することで、無効なエントリを作ることができる。しかし、V ビットを持たせるほうが TLB ミスハンドラの処理が簡単になる。
Data – 62:61	Size	size2 と size をビット結合した値でページサイズを指示する。 **Size2 Size<1:0> Page Size** 000_2 8 Kbyte 001_2 64 Kbyte 010_2 512 Kbyte 011_2 4 Mbyte 100_2 32 Mbyte 101_2 256 Mbyte 110_2 2 Gbyte
Data – 60	NFO	NFO (No Fault Only)。NFO に 1 がセットされているエントリは、ASI_PRIMARY_NO_FAULT, ASI_SECONDARY_NO_FAULT およびそれに *_LITTLE がついた ASI を使ったときのみ変換される。それ以外のアクセスでは *data_access_exception* 例外が通知され、DSFSR.FT = 10_{16} がセットされる。IMMU の NFO ビットは、読み出しには 0 が返り、書き込みは無視される。iTLB のミスハンドラは、TLB にエントリを書き込む際、NFO = 1 ならばエラーを報告すべきである。
Data – 59	IE	エンディアン反転。IE に 1 がセットされていると、そのページに対するアクセスでは、命令で指定したのとは逆のエンディアン (big には little、little には big) が使われる。詳細は JPS1 **Commonality** の Section F.7 を参照。IMMU の IE ビットは、読み出しには 0 が返り、書き込みは無視される。**Note:** このビットは主に、ノンキャッシャブル領域に使われる。キャッシャブルで IE = 1 であるページへのアクセスが D1$ ミスすると、性能に悪影響を与える可能性がある。
Data - 58:50	Soft2	ソフトウェアが使用するフィールド。ハードウェアはこのフィールドを TLB 内で保持する必要はなく、TLB から読み出すと 0 が読めるかもしれない。
Data – 49	*Reserved*	*Reserved*. 書き込みは無視され、0 が読み出される。
Data – 48	Size2	size の項参照。

TABLE F-2 TTE のビット説明 *(2 of 3)*

ビット位置	フィールド名	説明
Data – 47:41	*Reserved*	*Reserved*. 書き込みは無視され、0 が読み出される。
Data – 40:13	PA	物理ページ番号。8KB より大きいページサイズのページ内オフセットにあたるビット (64KB ページなら PA<15:13>, 512KB ページなら PA<18:13> など) は通常の変換では無視される。 SPARC64 VIIIfx は、JPS1 **Commonality** の定義と異なり、物理アドレスのビット数は 41 ビットまでをサポートする。(*impl.dep.#224*) TLB からエントリを読み出した際、ページ内オフセットにあたるビットに読み出される値は不定である。PA だけでなく VA についても、8KB より大きいページでページ内オフセットに当たるビットに読み出される値は不定である。(*impl.dep.#238*)
Data – 12:7	Soft	ソフトウェアが使用するフィールド。ハードウェアはこのフィールドを TLB 内で保持する必要はなく、TLB から読み出すと 0 が読めるかもしれない。
Data – 6	L	ロック。L に 1 がセットされているエントリが TLB に書き込まれると、そのエントリは TLB 上にロックされる。エントリが有効ならば、Data In レジスタによる TLB 書き込みによるリプレース対象とならない。無効なエントリでは L ビットは意味を持たない。ソフトウェアはロックしていないエントリを最低 1 つ残すようにしなければならない。 SPARC64 VIIIfx は TLB Data In を通じた書き込みではロックするかどうかを自動で判断する。書き込もうとしているエントリが TTE.L = 1 であるならば fTLB に、そうでなければページサイズに応じて fTLB か sTLB に書き込まれる。(*impl.dep.#225*) SPARC64 VIIIfx では、fITLB と fDTLB が lock ビットをサポートする。sITLB, sDTLB では lock ビットは実装されておらず、lock ビットの書き込みは無視され、読み出しには 0 が返る。(*impl.dep.#242*)
Data – 5 Data – 4	CP, CV	物理ページ・インデクスキャッシュと、論理ページ・インデクスキャッシュでキャッシャブルかどうかを指示する。CP に 1 がセットされていると、そのページのデータは I1, D1, U2 キャッシュにキャッシュされる。 SPARC64 VIIIfx ではどの TLB も CV を実装していない。SPARC64 VIIIfx はハードウェアでキャッシュエイリアスを解消する機構を備えている。CV ビットの書き込みは無視され、読み出しには 0 が返る。(*impl.dep.#226*)

TABLE F-2 TTE のビット説明 *(3 of 3)*

ビット位置	フィールド名	説明
Data – 3	E	TTE. 副作用があることを示す。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はセットする必要がある。 IMMU ではこのビットは、書き込みは無視され、読み出しには 0 が返る。 **Note:** E ビットはノンキャッシャブルを意味しない。E に 1 がセットされているときは、CP には通常 0 がセットされているはずだが、強制はされていない。CP にも E にも 1 がセットされているときの動作は未定義である。 **Note:** E ビットと NFO ビットは同時に 1 に設定してはいけない。
Data – 2	P	特権アクセスビット。P に 1 がセットされていると、その TTE でマップされたページには特権モードでのみアクセス可能である。PSTATE.PRIV = 0 でアクセスすると、*instruction_access_exception* か *data_access_exception* 例外を通知し、そのときの ISFSR.FT または DSFSR.FT には 1_{16} がセットされる。
Data – 1	W	書き込み可能。W に 1 がセットされていると、その TTE でマップされたページのデータは書き換え可能である。0 がセットされている場合、書き込もうとすると *fast_data_access_protection* 例外が通知される。IMMU ではこのビットは、書き込みは無視され、読み出しには 0 が返る。
Data – 0	G	グローバル。このビットの値は TTE タグの G ビットと一致していなければならない。

F.4 Hardware Support for TSB Access

JPS1 **Commonality** では TSB を検索するのはソフトウェアであり、ハードウェアは TLB ミス時に、ミスを起こした VA があると思われる TSB 上のポインタを計算するのみである。しかし、このポインタ計算は簡単な整数演算数命令で行える簡単なものであるし、また、JPS1 **Commonality** の仕様では TSB のハードウェアサポートは、8KB ページと 64KB ページに対してのみであり、それより大きなページサイズの TLB ミスに対するサポートは一切ないことから、SPARC64 VIIIfx では TSB のハードウェアサポートは実装していない。

しかしながら、SPARC64 VIIIfx では TSB Base レジスタは仕様に残してある。TLB ミス発生時、システムソフトウェアは TSB の先頭アドレスをメモリから読むのではなく、TSB Base レジスタから取得することができるので、TLB ミス時の処理オーバーヘッドはポインタ計算の数命令程度にとどまり、従来仕様の CPU とほとんど変わらない。TSB Base レジスタの詳細は Section F.10.6 を参照。

F.5 Faults and Traps

IMPL. DEP. #230: The cause of a *data_access_exception* trap is implementation dependent in JPS1, but there are several mandatory causes of a *data_access_exception* trap.

SPARC64 VIIIfx は *data_access_exception* を JPS1 **Commonality** の Section F.5 で定義された条件で通知する。ただし invalid ASI の場合は注意が必要である。詳細は Section F.10.9, *"I/D Synchronous Fault Status Registers (I-SFSR, D-SFSR)"* を参照。

IMPL. DEP. #237: Whether the fault status and/or address (DSFSR/DSFAR) are captured when *mem_address_not_aligned* is generated during a JMPL or RETURN instruction is implementation dependent.

SPARC64 VIIIfx では、JMPL または RETURN 命令で *mem_address_not_aligned* が通知される際、DSFSR/DSFAR は更新されない。

SPARC64 VIIIfx では、JPS1 **Commonality** の定義に加え、*instruction_access_error* と *data_access_error, SIMD_load_across_pages* が MMU により記録される。JPS1 **Commonality** の TABLE F-2 にこれらを加えたものを TABLE F-3 に示す。

TABLE F-3 MMU 例外と ASI レジスタ更新

			更新されるレジスタ				
番号	**トラップ名**	**トラップ要因**	**I-SFSR**	**I-MMU Tag Access**	**D-SFSR, SFAR**	**D-MMU Tag Access[1]**	**トラップ番号**
1.	*fast_instruction_access_MMU_miss*	I-TLB ミス	X[2]	X			64_{16}–67_{16}
2.	*instruction_access_exception*	下記参照	X[2]	X			08_{16}
3.	*fast_data_access_MMU_miss*	D-TLB ミス			X[3]	X	68_{16}–$6B_{16}$
4.	*data_access_exception*	下記参照			X[3]	X[4]	30_{16}
5.	*fast_data_access_protection*	書き込み保護違反			X[3]	X	$6C_{16}$-$6F_{16}$
6.	*privileged_action*	特権 ASI 使用			X[3]		37_{16}
7.	watchpoint	ウォッチポイント検出			X[3]		61_{16}–62_{16}
8.	*mem_address_not_aligned*, **_mem_address_not_aligned*	アラインメント違反			(impl. dep #237)		35_{16}, 36_{16}, 38_{16}, 39_{16}

TABLE F-3 MMU 例外と ASI レジスタ更新

番号	トラップ名	トラップ要因	更新されるレジスタ I-SFSR	I-MMU Tag Access	D-SFSR, SFAR	D-MMU Tag Access[1]	トラップ番号
9.	*instruction_access_error*	下記参照	X[2]				$0A_{16}$
10	*data_access_error*	下記参照			X[3]		32_{16}
11	*SIMD_load_across_pages*	SIMD ロードの extend 側で D-TLB ミス			X[3]		77_{16}

1.TAG_ACCESS_EXT_REG を含む。

2. I-SFSR の詳細は Section F.10.9 を参照。

3.. D-SFSR, D-SFAR の詳細は Section F.10.9 を参照。

4.*data_access_exception* 通知後、D-MMU Tag Access Register の context フィールドの内容は未定義。

バスエラー、バスタイムアウトによる *data_access_error* は、優先順位 12 のトラップの中では一番優先順位が低い。

TABLE F-3 の Ref #1~8 は JPS1 **Commonality** の Section F.5 の定義に準拠している。Ref #9, #10, #11 は以下の通り。

9. *instruction_access_error* — 以下の条件のうちどれか一つでも成立していれば例外が通知される。
   - 命令フェッチで訂正不可能なエラーが発見された場合。
   - 命令フェッチのメモリ参照にバスエラーが通知された場合。
   - fITLB で多重ヒットが検出された場合。

10. *data_access_error* — 以下の条件のうちどれか一つでも成立していれば例外が通知される。
   - データアクセスで訂正不可能なエラーが発見された場合。
   - データアクセスのメモリ参照にバスエラー、バスタイムアウトが通知された場合。
   - fDTLB で多重ヒットが検出された場合。

Note – SPARC64 VIIIfx はストアバッファを実装しているので、あるアドレスの読み出しに対して *data_access_error* が通知されない場合がある。詳細は Appendix P.7.1, *“ASI_ASYNC_FAULT_STATUS (ASI_AFSR)”* (page 285) を参照。

---

11. *SIMD_load_across_pages* — SIMD ロードの extended 側で TLB ミスが起きた場合、この例外が通知される。DSFAR には extended 側のアドレスが表示される。

Programming Note – *SIMD_load_across_pages* が通知された場合、システムソフトウェアは TLB を更新するのではなくエミュレーションすべきである。TLB 更新処理は行う必要はないので、TAG_ACCESS_REG は更新されない。Section 7.6.5, *"SPARC JPS1 Implementation-Dependent Traps"* (page 51) も参照。

F.5.1 Trap Conditions for SIMD Load/Store

SIMD ロード / ストアの例外は、basic, extended を個々にこの順序で処理した場合に起こりうる例外が、TABLE 7-2 (page 48) の優先順位に従って通知される。DSFSR, DSFAR に表示される値は、basic 側で起きた例外では basic 側の情報が、extended 側で起きた例外では extended 側の情報が表示される。

Note – *SIMD_load_across_pages* は extended 側で起きる例外である。

ただし *VA_watchpoint* については、basic 側の優先順位 12 の例外 (*fast_data_MMU_miss*, *data_access_exception*, *fast_data_access_protection*, *data_access_error*, *data_access_protection*) より、extended 側の *VA_watchpoint* が優先して通知される場合がある。

F.5.2 Behavior on TLB Error

SPARC64 VIIIfx は、fTLB の多重ヒットを検出すると *data_access_error* で通知するが、sTLB の多重ヒットは消去されソフトウェアには通知されない。TLB 検索時にパリティエラーが発見されると、そのエントリは消去 (sTLB) または自動訂正 (fTLB) され、ソフトウェアには通知されない。トラップはかならずプログラム順序で起こるが、消去または自動訂正は検出された時点で行われるので、投機実行されたメモリアクセスでも行われる。

SPARC64 VIIIfx で TLB でパリティエラー、多重ヒットが起きた場合の動作を TABLE F-4 に示す。

TABLE F-4 パリティエラー、多重ヒット検出時の動作

パリティエラー		多重ヒット		
sTLB	fTLB	sTLB	fTLB	動作
✓				エントリを消去し、*fast_instruction_access_MMU_miss* または *fast_data_access_MMU_miss* を通知する。
	✓			自動訂正する[1]。ソフトウェアには見えない。
✓	✓			fTLB のエントリを自動訂正し[1]、sTLB のエントリを消去する。
		✓		エントリを消去し、*fast_instruction_access_MMU_miss* または *fast_data_access_MMU_miss* を通知する。
			✓	*instruction_access_error* または *data_access_error* を通知する[2]。
		✓	✓	多重ヒットは検出せず、sTLB の内容で動作する[3]。
✓		✓		パリティエラー、多重ヒットを起こした全エントリを消去する。
	✓	✓		fTLB のエントリを自動訂正し[1]、sTLB のエントリを消去する。
✓			✓	sTLB のエントリを消去し、fTLB の多重ヒット[2] で *instruction_access_error* または *data_access_error* を通知する。
	✓		✓	パリティエラーしているエントリを自動訂正し、多重ヒット[2] で *instruction_access_error* または *data_access_error* を通知する。

1.fTLB は二重化されているので訂正可能。訂正できない場合は致命的エラー。
2.fTLB の多重ヒットは、常に検出できるとは限らない。
3. sTLB と fTLB の間で多重ヒットしている場合。

sTLB のパリティエラー、多重ヒットでは、エラーを起こしたエントリが消去される。この消去はソフトウェアには通知されない。ただし、SIMD ロードにおいては、basic 側の TLB 検索により extended 側で必要な sTLB エントリのパリティエラー、多重ヒットが消去された場合、*SIMD_load_across_pages* 例外という形でソフトウェアに見える。

パリティエラー、多重ヒットの検出は、TABLE F-3 に示すすべての例外検出と同時に起こりうる。TLB エントリの消去は、他の例外を検出したかどうかとは無関係に行われる。つまり、投機実行によりパリティエラーや、多重ヒットが検出された場合でも、そのエントリは消去される。

Note – 多重ヒット検出時は、どの TTE が正しいかを決定できないので、TTE の内容に依存する例外 (*data_access_exception*, *PA_watchpoint*, *fast_data_access_protection*, *SIMD_load_across_pages*) は検出されない。

F.8 Reset, Disable, and RED_state Behavior

IMPL. DEP. #231: The variability of the width of physical address is implementation dependent in JPS1, and if variable, the initial width of the physical address after reset is also implementation dependent in JPS1.

TABLE F-2 の Data 部の `PA` の項を参照。SPARC64 VIIIfx の物理アドレスは 41 ビットである。

IMPL. DEP. #232: Whether `CP` and `CV` bits exist in the DCU Control Register is implementation dependent in JPS1.

SPARC64 VIIIfx は DCU を実装していない (page 29)。`CP`, `CV` ビットは存在しない。

DMMU が無効なとき、MMU は TTE が以下の設定であるとみなして動作する。

- `TTE.IE` ← 0
- `TTE.P` ← 0
- `TTE.W` ← 1
- `TTE.NFO` ← 0
- `TTE.CV` ← 0
- `TTE.CP` ← 0
- `TTE.E` ← 1

IMPL. DEP. #117: Whether prefetch and nonfaulting loads always succeed when the MMU is disabled is implementation dependent.

SPARC64 VIIIfx では、DMMU が無効のとき `PREFETCH` はメモリアクセスをせずに実行完了する。ノンフォールティングロードは、JPS1 **Commonality** の Section F.5 の定義通り *data_access_exception* が通知される。

F.10 Internal Registers and ASI Operations

SPARC64 VIIIfx 仕様では TSB のハードウェアサポートは実装しない。このため JPS1 **Commonality** で定義されているレジスタのうち以下のものは実装されない。

TABLE F-5 SPARC64 VIIIfx 仕様で無効な MMU 関連レジスタ

IMMU ASI	DMMU ASI	VA	レジスタ
50_{16}	58_{16}	48_{16}	Instruction/Data TSB Primary Extension Registers
—	58_{16}	50_{16}	DATA TSB Secondary Extension Register
50_{16}	58_{16}	58_{16}	I/D TSB Nucleus Extension Registers
51_{16}	59_{16}	00_{16}	I/D TSB 8KB Pointer Registers
52_{16}	$5A_{16}$	00_{16}	I/D TSB 64KB Pointer Registers
—	$5B_{16}$	00_{16}	DATA TSB Direct Pointer Register

これらの ASI, VA にアクセスすると、*data_access_exception* 例外が通知される。

F.10.1 Accessing MMU Registers

IMPL. DEP. #233: Whether the `TSB_Hash` field is implemented in I/D Primary/Secondary/Nucleus TSB Extension Register is implementation dependent in JPS1.

SPARC64 VIIIfx では TSB Extension レジスタを定義していないので、この実装依存仕様は意味を持たない。

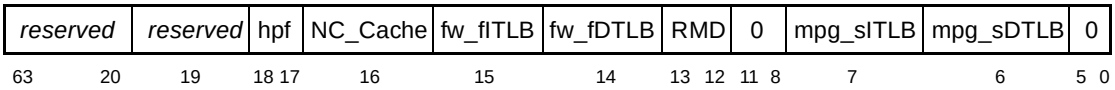
IMPL. DEP. #239: The register(s) accessed by IMMU ASI 55_{16} and DMMU ASI $5D_{16}$ at virtual addresses 40000_{16} to $60FF8_{16}$ are implementation dependent.

“I/D TLB Data In, Data Access, and Tag Read Registers” (page 192)*“I/D TLB Data In, Data Access, and Tag Read Registers”* (page 192) を参照。

SPARC64 VIIIfx では JPS1 **Commonality** の TABLE F-9 記載のレジスタ以外にも、`ASI_DCUCR` (page 34) と `ASI_MCNTL` (page 183) に MMU の制御機能が割り当てられている。

ASI_MCNTL (Memory Control Register)

レジスタ名	`ASI_MCNTL`
ASI	45_{16}
VA	08_{16}
アクセス種別	Supervisor read/write



ビット	フィールド名	アクセス	説明
63:20	Reserved		
19	*Reserved*	RW	ソフトウェアはこのフィールドには 0 を設定すること。1 を設定した場合の CPU の動作は不定。
18:17	hpf	RW	ハードウェアプリフェッチモードを指定する。 00_2: ハードウェアプリフェッチを strong pf で生成する。 01_2: ハードウェアプリフェッチを生成しない。 10_2: ハードウェアプリフェッチを weak pf で生成する。 11_2: reserved 11_2 を指定したときのハードウェアプリフェッチ動作は不定。
16	NC_Cache	RW	ノンキャッシャブル領域にある命令列を強制的にキャッシュする。NC_Cache が 1 にセットされていると、CPU からは 16 バイトのノンキャッシャブルアクセスが 8 回行われ、全 128 バイトが I1$ キャッシュに書き込まれる。データアクセスの動作には影響を与えない。 NC_Cache は、OBP の実行速度向上のための機能である。OBP は OS に制御を移す際に NC_Cache に 0 をセットすること。さもないと、I1 キャッシュにノンキャッシャブル領域の命令が残ってしまう。
15	fw_fITLB	RW	ITLB 更新時、必ず fITLB に書き込む。fw_fITLB に 1 がセットされていると、ITLB Data In レジスタ経由の TLB 書き込みは、必ず fITLB に書かれる。fw_fITLB は OBP 向けの機能である。

ビット	フィールド名	アクセス	説明
14	fw_fDTLB	RW	DTLB 更新時、必ず fDTLB に書き込む。fw_fDTLB に 1 がセットされていると、DTLB Data In レジスタ経由の TLB 書き込みは、必ず fDTLB に書かれる。fw_fDTLB は OBP 向けの機能である。
13:12	RMD	R	このフィールドは 2 に固定されており、書き込みは無視される。
11:8	Reserved		
7	mpg_sITLB[1]	RW	sITLB でマルチプルページサイズ機能を有効にする。mpg_sITLB に 1 がセットされていると、sITLB にはコンテキストごとに異なるページサイズの TTE を登録できる。mpg_sITLB に 0 がセットされていると、コンテキストレジスタと IMMU_TAG_ACCESS_EXT のページサイズ情報は意味を持たず、規定のページサイズ (1st sITLB は 8KB, 2nd sITLB は 4MB）が使われる。
6	mpg_sDTLB[1]	RW	sDTLB でマルチプルページサイズ機能を有効にする。mpg_sDTLB に 1 がセットされていると、sDTLB にはコンテキストごとに異なるページサイズの TTE を登録できる。mpg_sDTLB に 0 がセットされていると、コンテキストレジスタと DMMU_TAG_ACCESS_EXT のページサイズ情報は意味を持たず、規定のページサイズ (1st sDTLB は 8KB, 2nd sDTLB は 4MB）が使われる。
5:0	Reserved		

1.mpg_sITLB = 1 かつ mpg_sDTLB = 0 という設定をしてはいけない。この設定のときの動作は未定義。

F.10.2 Context Registers

sTLB は 2 つのセットアソシエイティブ TLB で構成されている。1st TLB, 2nd TLB とも、ITLB は 128 エントリ、DTLB は 256 エントリである。デフォルトでは 1st sTLB には 8KB ページの TTE のみ、2nd sTLB には 4MB ページの TTE のみが登録されるが、MCNTL.mpg_sITLB, MCNTL.mpg_sDTLB を 1 にセットすることで、8 KB, 64 KB, 512 KB, 4 MB, 32MB, 256MB, 2GB のうち任意の一つのページサイズの TTE が登録できる。ページサイズの設定は 1st sTLB, 2nd sTLB で独自に設定でき、両方を同じページサイズに設定することもできる。

ページサイズはコンテキストレジスタのフィールドで指定する。
ASI_PRIMARY_CONTEXT_REG では sITLB, sDTLB のページ設定が、ASI_SECONDARY_CONTEXT_REG では sDTLB の設定ができる。1st sTLB, 2nd sTLB のページサイズを同じものに設定した場合、sTLB は一つの 4 ウェイセットアソシエイティブ TLB と同じように動作する。

ページサイズ指定は以下のエンコードデータで行う。

- 000_{02} = 8 KB
- 001_{02} = 64 KB
- 010_{02} = 512 KB
- 011_{02} = 4 MB
- 100_{02} = 32 MB
- 101_{02} = 256 MB
- 110_{02} = 2 GB

Note – 111_{02} を指定したときの挙動は未定義。

JPS1 **Commonality** で定義されたコンテキストレジスタに加えて、SPARC64 VIIIfx では共有コンテキスト (Shared Context) が定義されている。共有コンテキストは、ある論理空間を複数のプロセスで共有するための仕組みで、命令列や共有データを置くのに使われる。あるコンテキストから別のコンテキストをアクセスするという点ではセカンダリコンテキストと似ているが、以下の点が異なる。

- セカンダリコンテキスト空間にアクセスするためには、ASI 付きロード / ストア命令を使う必要があるが、共有コンテキスト空間はプライマリコンテキストと同じく、暗黙のうちにアクセス可能である。
- セカンダリコンテキストがデータアクセスのみなのに対し、共有コンテキスト空間は命令フェッチとデータアクセス両方で使われる。

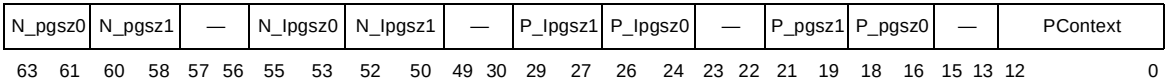
以下の説明では実効コンテキストという用語が使われる。複数あるコンテキストレジスタのどの値を使うかは命令種類やプロセッサの状態により決まるので、実際に使われるコンテキスト番号を指す用語として用いる。

- TL = 0における命令フェッチやASI付きではないロード/ストア命令によるアクセスでは、ASI_PRIMARY_CONTEXT の値を指す。
- TL > 0における命令フェッチやASI付きではないロード/ストア命令によるアクセスでは、ASI_NUCLEUS_CONTEXT の値を指す。

■ ASI 付きのロード / ストア命令では、その ASI によって決まるコンテキスト。

ASI_PRIMARY_CONTEXT

レジスタ名	ASI_PRIMARY_CONTEXT
ASI	58_{16}
VA	08_{16}
アクセス種別	Supervisor read/write

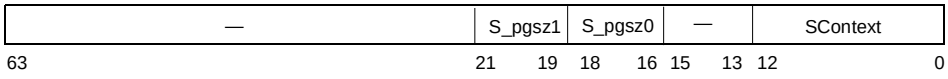


ビット	フィールド名	アクセス	説明
63:61	N_pgsz0	RW	Nucleus コンテキストが 1st sDTLB で使うページサイズ。
60:58	N_pgsz1	RW	Nucleus コンテキストが 2nd sDTLB で使うページサイズ。
55:53	N_Ipgsz0	RW	Nucleus コンテキストが 1st sITLB で使うページサイズ。
52:50	N_Ipgsz1	RW	Nucleus コンテキストが 2nd sITLB で使うページサイズ。
29:27	P_Ipgsz1	RW	プライマリコンテキストが 2nd sITLB で使うページサイズ。
26:24	P_Ipgsz0	RW	プライマリコンテキストが 1st sITLB で使うページサイズ。
21:19	P_pgsz1	RW	プライマリコンテキストが 2nd sDTLB で使うページサイズ。
18:16	P_pgsz0	RW	プライマリコンテキストが 1st sDTLB で使うページサイズ。
12:0	PContext	RW	プライマリコンテキスト番号。

ASI_PRIMARY_CONTEXT の各ページサイズフィールドは、ASI_MCNTL.mpg_sITLB, ASI_MCNTL.mpg_sDTLB の設定に係わらず読み出すことができる。

ASI_SECONDARY_CONTEXT

レジスタ名	ASI_SECONDARY_CONTEXT
ASI	58_{16}
VA	10_{16}
アクセス種別	Supervisor read/write

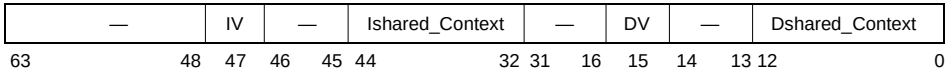


ビット	フィールド名	アクセス	説明
21:19	S_pgsz1	RW	セカンダリコンテキストが 2nd sDTLB で使うページサイズ。
18:16	S_pgsz0	RW	セカンダリコンテキストが 1st sDTLB で使うページサイズ。
12:0	SContext	RW	セカンダリコンテキスト番号。

ASI_SECONDARY_CONTEXT の各ページサイズフィールドは、ASI_MCNTL.mpg_sITLB, ASI_MCNTL.mpg_sDTLB の設定に係わらず読み出すことができる。

ASI_SHARED_CONTEXT

レジスタ名	ASI_SHARED_CONTEXT
ASI	58_{16}
VA	68_{16}
アクセス種別	Supervisor read/write



ビット	フィールド名	アクセス	説明
47	IV	RW	Ishared_Context の有効ビット。IV が 1 かつ Ishared_Context が 0 以外の値のとき、命令フェッチにおける MMU 変換では、実効コンテキストと Ishared_Context の値の両方が使われる。IV が 0 または実効コンテキストが 0 のときは、MMU 変換には実効コンテキストのみが使われる。
44:32	Ishared_Context	RW	共有コンテキストの命令フェッチで使われるコンテキスト番号。
15	DV	RW	Dshared_Context の有効ビット。DV が 1 かつ Dshared_Context が 0 以外の値のとき、データアクセスにおける MMU 変換では、実効コンテキストと Dshared_Context の値の両方が使われる。DV が 0 または実効コンテキストが 0 のときは、MMU 変換には実効コンテキストのみが使われる。
12:0	Dshared_Context	RW	共有コンテキストのデータアクセスで使われるコンテキスト番号。

ASI_SHARED_CONTEXT は、実効コンテキストによる MMU 変換時に、共有コンテキストによる変換を行うかどうか、またそのときのコンテキスト番号を指示するレジスタである。共有コンテキスト番号として 0 以外の値を指定し、IV または DV を 1 にセットしたときに有効になる。実効コンテキストが 0 のときは、IV または DV の値によらず共有コンテキストは使われない。例えば、TL > 0 で ASI_AS_IF_USER_SECONDARY を指定したロード命令を実行すると、SContext が実効コンテキストになるので、共有コンテキストが使われるかどうかは SContext が 0 かどうかで決まる。

共有コンテキストの機能は、ページサイズ指定を除き実効コンテキストと同じである。SPARC64 VIIIfx には 2 つの sITLB と 2 つの sDTLB があり、格納される TTE のページサイズを個々に設定することができる。しかし、共有コンテキストは独自のページサイズを持たず、実効コンテキストのページサイズが使われる。このため、ASI_MCNTL.mpg_sI/DTLB が 0 のときは、1st sTLB は 8KB ページ、2nd sTLB は 4 MB ページのエントリが書き込まれ、ASI_MCNTL.mpg_sI/DTLB が 1 のときは、1st sTLB には p_pgsz0/s_pgsz0/p_Ipgsz0 で指定されたページサイズが、2nd sTLB には p_pgsz0/s_pgsz0/p_Ipgsz0 で指定されたページサイズが使われる。

Note – 実効コンテキストが 0 のときは共有コンテキストは無効なので、n_pgsz0/1 は使われない。

Programming Note – 共有コンテキストで sTLB を効率よく使うためには、ある共有コンテキストを使う全てのコンテキストで p_pgsz(0,1)/p_Ipgsz(0,1)/s_pgsz(0,1) に同じページサイズを指定するとよい。

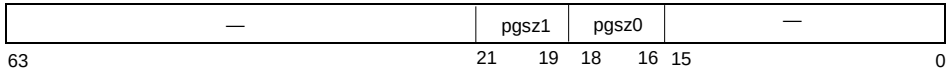
F.10.3 Instruction/Data MMU TLB Tag Access Registers

MMU ミスやアクセス権違反による例外が通知されるとき、共有コンテキストが有効な場合、I/D TLB Tag Access レジスタには実効コンテキストのコンテキスト番号が表示される。

Programming Note – TLB に共有コンテキストの TTE を書き込む場合、I/D TLB Tag Access レジスタに共有コンテキスト番号を設定してから、I/D TLB Data In/Data Access レジスタによる書き込みを行うこと。

ASI_I/DMMU_TAG_ACCESS_EXT

レジスタ名	ASI_IMMU_TAG_ACCESS_EXT, ASI_DMMU_TAG_ACCESS_EXT
ASI	50_{16} (IMMU), 58_{16} (IMMU)
VA	60_{16}
アクセス種別	Supervisor read/write



MMU の例外によるトラップが発生する際、トラップの種類によっては、ハードウェアは例外を起こした論理アドレスとコンテキスト番号を Tag Access レジスタ (ASI_I/DMMU_TAG_ACCESS) にセーブする。詳細は TABLE F-3 (page 178) を参照。I/DTLB Data In レジスタで TLB に TTE を書き込む際の sTLB のインデクス計算を簡単にするため、SPARC64 VIIIfx は Tag Access レジスタで欠けている実効コンテキストのページサイズ情報を、ASI_I/DMMU_TAG_ACCESS_EXT レジスタにセーブする。

Note – 書き込まれる TTE のページサイズと ASI_I/DMMU_TAG_EXT.pgsz0/1 が異なる場合、その TTE は sTLB ではなく fTLB に書き込まれる。

---

instruction_access_exception, *data_access_exception* が通知された場合、ASI_I/DMMU_TAG_ACCESS_EXT レジスタは無効で、値は未定義である。また、ASI_MCNTL.mpg_sITLB が 0 のときの ASI_IMMU_TAG_ACCESS_EXT レジスタ、ASI_MCNTL.mpg_sDTLB が 0 のときの ASI_DMMU_TAG_ACCESS_EXT レジスタは無効で、値は未定義である。

F.10.4 I/D TLB Data In, Data Access, and Tag Read Registers

IMPL. DEP. #234: The replacement algorithm of a TLB entry is implementation dependent in JPS1.

fTLB は pseudo-LRU, sTLB は LRU により追い出すエントリが決定される。

IMPL. DEP. #235: The MMU TLB data access address assignment and the purpose of the address are implementation dependent in JPS1.

SPARC64 VIIIfx の I/D TLB Data Access レジスタのアドレス割り当てを TABLE F-6 に示す。

TABLE F-6 TLB Data Access レジスタのアドレス割り当て

ビット	フィールド名	アクセス	説明
17:16	TLB#	RW	アクセスする TLB を指定する。 00_{02}: fTLB (16 エントリ) 01_{02}: reserved 10_{02}: sTLB(IMMU では 256 エントリ、DMMU では 512 エントリ) 11_{02}: reserved
15	Reserved		
13:3	TLB index	RW	TLB のインデクス番号。 • fTLBの場合は、下位 4 ビットがインデクス番号で、上位 7 ビットは無視される。下位 4 ビットの値と TLB インデクスの関係は : 0-15: fTLB のインデクス番号 • sITLBの場合は、bit<13:12> でウェイ、bit<8:3> でインデクスを表わし、bit<11:9> は無視される。値と TLB インデクスの関係は : 0-63: 1st sITLB のウェイ 0 のインデクス番号 512-575: 1st sITLB のウェイ 1 のインデクス番号 1024-1087: 2nd sITLB のウェイ 0 のインデクス番号 1536-1599: 2nd sITLB のウェイ 1 のインデクス番号 • sDTLBの場合は、bit<13:12> でウェイ、bit<9:3> でインデクスを表わし、bit<11:10> は無視される。値と TLB インデクスの関係は : 0-127:1st sDTLB のウェイ 0 のインデクス番号 512-639:1st sDTLB のウェイ 1 のインデクス番号 1024-1151:2nd sDTLB のウェイ 0 のインデクス番号 1536-1663:2nd sDTLB のウェイ 1 のインデクス番号

Note – I/D TLB Data Inによる TLB 書き込みでは、TTE.G = 1のエントリは fTLB に書かれる。

I/D MMU TLB Tag Read Register

IMPL. DEP. #238: When read, an implementation will return either 0 or the value previously written to them.

TABLE F-2 (page 175) の PA の項参照。

TLB Tag Read レジスタの VA フォーマットは、TLB Data Access レジスタと同じである。詳細は TABLE F-6 を参照。

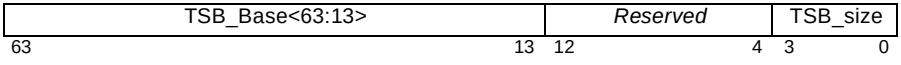
I/D MMU TLB Tag Access Register

しかし、TTE.V = 0 であるエントリを I/D TLB Data Access で書き込むときは、整合性の検査は行われず書き込まれる。これにより TLB の特定エントリだけを削除することができる。この機能は、ソフトウェアによりエラーを起こしている TLB エントリのエラーを消去をする際に使うことができる。

I/D TLB Data Access レジスタ、I/D TLB Data In レジスタで TLB に TTE を書き込む際、ハードウェアは I/D TLB Tag Access レジスタの内容との整合性を検査し、不整合な場合は TLB は更新されない。

Implementation Note – TTE.V = 0 であるエントリを書き込み、それを読み出すと、全ビット 0 のデータが返る。

F.10.6 I/D TSB Base Registers



SPARC64 VIIIfx は TSB のハードウェアサポートはないが、システムソフトウェアが TSB を扱えるよう TSB Base レジスタ仕様は残してある。JPS1 **Commonality** では TSB Base レジスタには以下のフィールドがある。

- TSB_Base
- Split
- TSB_Size

SPARC64 VIIIfx の TSB Base レジスタはこのうち `TSB_Base` と `TSB_Size` を実装し、`Split` は *reserved* とする。

TSB_size は bit<3:0> の 4 ビットとする (impl.dep. #236)。TSB_Size に書いた値は、読みだしによりその値のまま読みだされる。ハードウェアは値を保持するだけで、この値を何の計算にも使わない。

F.10.7 I/D TSB Extension Registers

SPARC64 VIIIfx は TSB Extension レジスタをサポートしない。読み出し、書き込みをしようとすると *data_access_exception* が通知される。

F.10.8 I/D TSB 8-Kbyte and 64-Kbyte Pointer and Direct Pointer Registers

SPARC64 VIIIfx はこれらのレジスタをサポートしない。読み出し、書き込みをしようとすると *data_access_exception* が通知される。

F.10.9 I/D Synchronous Fault Status Registers (I-SFSR, D-SFSR)

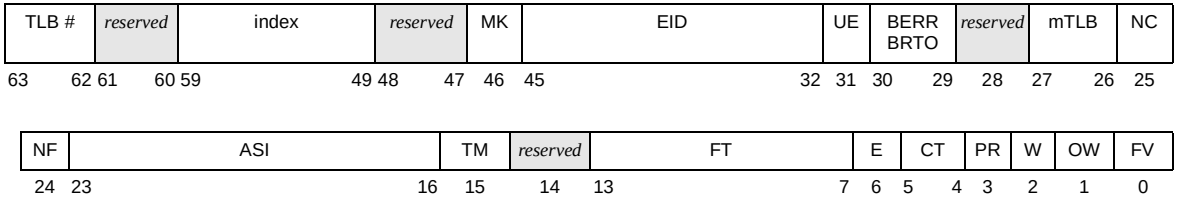


FIGURE F-2 MMU I/D Synchronous Fault Status Registers (`I-SFSR`, `D-SFSR`)

JPS1 **Commonality** では、I/D-SFSR のビット 63–25 は実装依存となっている。SPARC64 VIIIfx の I-SFSR の実装依存ビットの定義を TABLE F-7 に D-SFSR の実装依存ビットの定義を TABLE F-10 に示す。ビット 24–0 は JPS1 **Commonality** に準拠する。

TABLE F-7 I-SFSR のフィールドの説明 *(1 of 2)*

ビット	フィールド名	アクセス	説明
63:62	TLB#	RW	mITLB でエラーが起こったことを示す。SPARC64 VIIIfx では 00_{02} が表示される。
59:49	index	RW	mITLB でエラーが起きた場合のインデクス番号を示す。 複数のエラーが起きたとき、任意の一つのインデクスが表示される。
46	MK	RW	マーク済の訂正不能エラー。SPARC64 VIIIfx では、すべての訂正不能エラーはエラーマークして報告される。I-SFSR.UE が 1 のとき、MK には常に 1 がセットされる。詳細は Appendix P.2.4, “ *キャッシャブルデータのエラーマーキング* ” (page 267) 参照。
45:32	EID	RW	エラーマーク ID。このフィールドは MK が 1 のとき有効。詳細は Appendix P.2.4, “ *キャッシャブルデータのエラーマーキング* ” (page 267) 参照。
31	UE	RW	訂正不能エラー (Uncorrectable Error:UE)。UE に 1 がセットされていると、フェッチした命令列の中に訂正不能エラーがあったことを示す。このフィールドは *instruction_access_error* が通知されたときのみ有効。
30	BERR	RW	命令フェッチにメモリバスエラーが返されたことを示す。このフィールドは *instruction_access_error* が通知されたときのみ有効。
29	BRTO	RW	命令フェッチにバスタイムアウトが返されたことを示す。このフィールドは *instruction_access_error* が通知されたときのみ有効。
27:26	mITLB<1:0>	RW	mITLB のエラーステータス。mITLB の検索時に多重ヒットが起きたときは mITLB<1> に 1 がセットされる。mITLB<0> は常に 0。このフィールドは *instruction_access_error* が通知されたときのみ有効。
25	NC	RW	ノンキャッシャブル領域を参照したことを示す。このフィールドは訂正不能エラー、バスエラー、バスタイムアウトのいずれかによる *instruction_access_error* が通知されたときのみ有効。それ以外の例外通知時は、このフィールドの値は不定である。
23:16	ASI<7:0>	RW	例外が起きた際のアクセスに使われた ASI 番号が表示される。このフィールドは ISFSR.FV に 1 がセットされているときのみ有効。 表示される ASI 番号は、TL = 0 のときは 80_{16}(ASI_PRIMARY), TL > 0 のとき 04_{16} (ASI_NUCLEUS) である。
15	TM	RW	命令フェッチ中に TLB ミスが起きたことを示す。

TABLE F-7 I-SFSR のフィールドの説明 *(2 of 2)*

ビット	フィールド名	アクセス	説明
13:7	FT<6:0>	RW	例外発生の原因をエンコード情報で示す。エンコードの意味は TABLE F-8 を参照。 このフィールドは *instruction_access_exception* が通知されたときのみ有効。*fast_instruction_access_MMU_miss* では常に 0 が読み出され、*instruction_access_exception* では常に 01_{16} が読み出される。
5:4	CT<1:0>	RW	例外を起こした命令フェッチのコンテキストに関する情報が表示される。 00_{02}: Primary 01_{02}: Reserved 10_{02}: Nucleus 11_{02}: Reserved Translating ASI でないか、または無効な ASI の場合、11_{02} が表示される。 **Note:** 共有コンテキストで起きたことを示す手段は定義されていない。共有コンテキストが関係する多重ヒットのときは、実効コンテキストの情報が表示される。
3	PR	RW	特権モードで命令フェッチ中に例外が通知されたことを示す。このフィールドは FV が 1 のときのみ有効。
1	OW	RW	ISFSR.FV = 1 のときに例外が通知されたことを示す。例外通知時点で ISFSR.FV = 1 だと 1 がセットされ、ISFSR.FV = 0 だと 0 がセットされる。
0	FV	RW	IMMU で TLB ミス以外の例外が発生したときに 1 がセットされる。このフィールドが 0 のとき、他のフィールドは意味を持たない (ただし MMU ミスの場合を除く)。

ISFSR.FT のエンコーディングを TABLE F-8 に示す。

TABLE F-8 I-SFSR.FT のエンコーディング

FT<6:0>	例外発生理由
01_{16}	特権アクセス違反。命令フェッチ時、TTE.P = 1 かつ PSTATE.PRIV = 0 だったことを示す。特権アクセス違反は *instruction_access_exception* 例外で通知される。
02_{16}	Reserved
04_{16}	Reserved
08_{16}	Reserved
10_{16}	Reserved
20_{16}	Reserved
40_{16}	Reserved

I-SFSR は *fast_instruction_access_MMU_miss*, *instruction_access_exception*, *instruction_access_error* のいずれかが通知された際に更新される。TABLE F-9 に各例外によりどのフィールドが更新されるかを示す。

TABLE F-9 I-SFSR 更新方針のまとめ

フィールド		TLB#, index	FV	OW	PR, CT[1]	FT	TM	ASI	UE, BERR, BRTO, mITLB, NC[2]
I-SFSR.OW = 0 のとき 0: 0 がセットされる。 1: 1 がセットされる。 V: 有効な値がセットされる。 —: フィールドは無効。									
Miss:	*fast_instruction_access_MMU_miss*	—	0	0	V	—	1	—	—
Exception:	*instruction_access_exception*	—	1	0	V	V	0	V	—
Error:	*instruction_access_error*	V[3]	1	0	V	—	0	V	V
I-SFSR.OW = 1 のとき 0: 0 がセットされる。 1: 1 がセットされる。 K: 元の値が保存される。 U: 更新される。									
Exception の後の Error		U[3]	1	1	U	K	K	U	U
Error の後の Exception		K	1	1	U	U	K	U	K
Miss の後の Error		U[3]	1	K	U	K	1	U	U
Miss の後の Exception		K	1	K	U	U	1	U	K
Exception か Error の後の Miss		K	1	K	K	K	1	K	K
Miss の後の Miss		K	K	K	U	K	1	K	K

1. Translating ASI でないか、無効な ASI では ISFSR.CT に 11_{02} がセットされる。

2.訂正不能エラー、バスエラー、バスタイムアウトのいずれかによる *instruction_access_error* が通知されたときのみ有効。

3.TLB の多重ヒット時のみ。

TABLE F-10 D-SFSR フィールドの説明 *(1 of 3)*

ビット	フィールド名	アクセス	説明
63:62	TLB#	RW	mDTLB でエラーが起こったことを示す。SPARC64 VIIIfx では 00_{02} が表示される。
59:49	index	RW	mDTLB でエラーが起きた場合のインデクス番号を示す。 複数のエラーが起きたとき、任意の一つのインデクスが表示される。
46	MK	RW	マーク済の訂正不能エラー。SPARC64 VIIIfx では、すべての訂正不能エラーはエラーマークして報告される。DSFSR.UE が 1 のとき、MK には常に 1 がセットされる。詳細は Appendix P.2.4, “ キャッシャブルデータのエラーマーキング ” (page 267) 参照。

TABLE F-10 D-SFSR フィールドの説明 *(2 of 3)*

ビット	フィールド名	アクセス	説明
45:32	EID	RW	エラーマーク ID。このフィールドは MK が 1 のとき有効。詳細は Appendix P.2.4, *“ キャッシャブルデータのエラーマーキング ”* (page 267) 参照。
31	UE	RW	訂正不能エラー (Uncorrectable Error:UE)。UE に 1 がセットされていると、アクセスデータの中に訂正不能エラーがあったことを示す。このフィールドは *data_access_error* が通知されたときのみ有効。
30	BERR	RW	データアクセスにメモリバスエラーが返されたことを示す。このフィールドは *data_access_error* が通知されたときのみ有効。
29	BRTO	RW	データアクセスにバスタイムアウトが返されたことを示す。このフィールドは *data_access_error* が通知されたときのみ有効。
27:26	mDTLB<1:0>	RW	mDTLB のエラーステータス。mDTLB の検索時に多重ヒットが起きたときは mDTLB<1>1 がセットされる。mDTLB<0> は常に 0。このフィールドは *data_access_error* が通知されたときのみ有効。
25	NC	RW	ノンキャッシャブル領域を参照したことを示す。このフィールドは訂正不能エラー、バスエラー、バスタイムアウトのいずれかによる *data_access_error* が通知されたときのみ有効。それ以外の例外通知時は、このフィールドの値は不定である。
24	NF	RW	ノンフォールティングロード命令で例外が起きたことを示す。
23:16	ASI<7:0>	RW	例外が起きた際のアクセスに使われた ASI 番号が表示される。このフィールドは DSFSR.FV に 1 がセットされているときのみ有効。データアクセス時に明示的に ASI が使われていないければ、暗黙の ASI が使われているので、以下の値がセットされる。 TL = 0, PSTATE.CLE = 0　80_{16} (ASI_PRIMARY) TL = 0, PSTATE.CLE = 1　88_{16} (ASI_PRIMARY_LITTLE) TL > 0, PSTATE.CLE = 0　04_{16} (ASI_NUCLEUS) TL > 0, PSTATE.CLE = 1　$0C_{16}$ (ASI_NUCLEUS_LITTLE)
15	TM	RW	データアクセス中に TLB ミスが起きたことを示す。
13:7	FT<6:0>	RW	例外発生の原因をエンコード情報で示す。エンコードの意味は TABLE F-11 を参照。
6	E	RW	副作用があるページにアクセスしたことを示す。TTE.E = 1 のページか、ASI 15_{16} か $1D_{16}$ でアクセスして例外が通知されたときに、E に 1 がセットされる。このフィールドは訂正不能エラー、バスエラー、バスタイムアウトのいずれかによる *data_access_error* が通知されたときのみ有効。それ以外の例外通知時は、このフィールドの値は不定である。

TABLE F-10 D-SFSR フィールドの説明 *(3 of 3)*

ビット	フィールド名	アクセス	説明
5:4	CT<1:0>	RW	例外を起こしたデータアクセスのコンテキストに関する情報が表示される。 00_{02}: Primary 01_{02}: Secondary 10_{02}: Nucleus 11_{02}: Reserved Translating ASI でないか、または無効な ASI の場合、11_{02} が表示される。 無効な組み合わせの ASI と命令により *data_access_exception* が通知された場合 (atomic quad load, block load/store, block commit store, partial store, short floating-point load/store, xfill の各 ASI は、特定のメモリアクセス命令でのみ使える)、CT には命令で指定された ASI が表示される。 **Note:** 共有コンテキストで起きたことを示す手段は定義されていない。共有コンテキストが関係する多重ヒットのときは、実効コンテキストの情報が表示される。
3	PR	RW	特権モードでデータアクセス中に例外が通知されたことを示す。このフィールドは FV が 1 のときのみ有効。
2	W	RW	書き込み命令 (ストア命令かアトミック命令) で例外が発生したことを示す。
1	OW	RW	DSFSR.FV = 1 のときに例外が通知されたことを示す。例外通知時点で DSFSR.FV = 1 だと 1 がセットされ、DSFSR.FV = 0 だと 0 がセットされる。
0	FV	RW	DMMU で TLB ミス以外の例外が発生したときに 1 がセットされる。このフィールドが 0 のとき、他のフィールドは意味を持たない (ただし MMU ミスの場合を除く)。

DSFSR.FT のエンコーディングを TABLE F-11 に示す。

TABLE F-11 D-SFSR.FT のエンコーディング

FT<6:0>	例外発生理由
01_{16}	特権アクセス違反。TTE.P = 1 のページにアクセスした時、PSTATE.PRIV = 0 か ASI_PRIMARY/SECONDARY_AS_IF_USER{_LITTLE} が使われたことを示す。特権アクセス違反は *data_access_exception* 例外で通知される。
02_{16}	ノンフォールティングロードで TTE.E = 1 であるページにアクセスしたとき、1 がセットされる。
04_{16}	TTE.CP = 0 であるページに、アトミック命令 (CASA, CASXA, SWAP, SWAPA, LDSTUB, LDSTUBA), atomic quad load 命令 (LDDA with ASI = 24_{16}, $2C_{16}$, 34_{16}, $3C_{16}$) または SIMD ロード / ストアでアクセスしたとき、1 がセットされる。

TABLE F-11 D-SFSR.FT のエンコーディング

FT<6:0>	例外発生理由
08_{16}	ASI 空間に対し、無効な ASI 番号、無効な VA、間違った操作 (読み出し / 書き込み) でアクセスしたとき、1 がセットされる。 ASI 番号が有効かどうかの検査は TTE の検索より先に行われるので、上記の条件にあてはまるときは *data_access_exception* が通知される。FT の他のビットは、TTE を検査して発見される要因なので、FT<3> = 1 のときは不定となる。 間違ったデータ長の命令でアクセスした場合は、より優先順位の高い *mem_address_not_aligned*, **_mem_address_not_aligned* が通知される。このとき FT の値は不定である。詳細は Appendix L.3.3, *"ASI と命令の組み合わせと例外 "* (page 221) を参照。
10_{16}	TTE.NFO = 1 のページにノンフォールティングロード以外でアクセスしたとき、1 がセットされる。
20_{16}	Reserved.
40_{16}	Reserved.

複数の要因であるひとつの例外が通知されると、DSFSR.FT の複数ビットがセットされうる。

D-SFSR は *fast_data_access_MMU_miss*, *data_access_exception*, *fast_data_access_protection*, *PA_watchpoint*, *VA_watchpoint*, *privileged_action*, *mem_address_not_aligned*, *data_access_error* のいずれかが通知された際に更新される。TABLE F-12 に各例外によりどのフィールドが更新されるかを示す。

TABLE F-12 D-SFSR 更新方針のまとめ

フィールド		TLB#, index	FV	OW	W, PR, NF, CT[1]	FT	TM	ASI	UE, BERR, BRTO, mDTLB, NC[2] , E[2]	DSFAR
D-SFSR.OW = 0 のとき 0: 0 がセットされる。 1: 1 がセットされる。 V: 有効な値がセットされる。 —: フィールドは無効。										
Miss:	*fast_data_access_MMU_miss*	—	0	0	V	—	1	—	—	V
Exception:	*data_access_exception*	—	1	0	V	V	0	V	—	V
Faults:	*fast_data_access_protection*	—	1	0	V	—	0	V	—	V
	PA_watchpoint	—	1	0	V	—	0	V	—	V
	VA watchpoint	—	1	0	V	—	0	V	—	V
	privileged_action[3]	—	1	0	V	—	0	V	—	V
	*mem_address_not_aligned, *_mem_address_not_aligned*	—	1	0	V	—	0	V	—	V
	data_access_error	V[4]	1	0	V	—	0	V	V	V
	SIMD_load_across_pages	—	1	0	V	—	0	V	—	V

TABLE F-12 D-SFSR 更新方針のまとめ

フィールド	TLB#, index	FV	OW	W, PR, NF, CT[1]	FT	TM	ASI	UE, BERR, BRTO, mDTLB, NC[2] , E[2]	DSFAR
D-SFSR.OW = 1 のとき 0: 0 がセットされる。 1: 1 がセットされる。 K: 元の値が保存される。 U: 更新される。									
Exception の後の Fault	U[4]	1	1	U	K	K	U	U	U
Fault の後の Exception	K	1	1	U	U	K	U	K	U
Miss の後の Fault[5]	U[4]	1	K	U	K	1	U	U	U
Miss の後の Exception[5]	K	1	K	U	U	1	U	K	U
Fault/Exception の後の Miss	K	1	K	K	K	1	K	K	K
Miss の後の Miss	K	K	K	U	K	1	K	K	K

1.Translating ASI ではないか、または無効な ASI の場合、DSFSR.CT に 11_{02} がセットされる。

2.訂正不能エラー、バスエラー、バスタイムアウトのいずれかによる *data_access_error* が通知されたときのみ有効。

3.メモリアクセス命令のみ。

4. TLB の多重ヒット時のみ。

5.Miss の後の Fault/Exception は、まず Miss が起き、その後ソフトウェアが DSFSR をクリアする前に Fault/Exception が起きる状況。

F.10.10 Synchronous Fault Addresses

VA_watchpoint, *PA_watchpoint* 例外が通知されたとき、D-SFAR には命令で指示されたアドレスが表示される。

ただし、SIMD 拡張されたロード・ストア命令の extend 側だけでウォッチポイント例外を検出したときは、extend 側のアドレス、つまり、命令で指示されたアドレスに、単精度ならば 4、倍精度ならば 8 を加算したアドレスが表示される。

F.10.11 I/D MMU Demap

sTLB 上のページデマップ時、インデクス計算に使われるページサイズは、ASI_I/DMMU_DEMAP アクセスアドレスの context フィールドの情報から、TLB 検索と同じ手順で求められる。すなわち、ASI_MCNTL.mpg_sI/DTLB が 0 のときは、1st sTLB は 8KB ページで 2nd sTLB は 4MB ページ、ASI_MCNTL.mpg_sI/DTLB が 1 のときは、context フィールドで指示されたコンテキストのページサイズ情報が使われる。

ページサイズ情報はまた、ページデマップやコンテキストデマップにおける TTE の選択にも使われる。すなわち、ページサイズが TLB エントリのそれと一致しない場合、そのエントリはデマップされない。

Note – ページデマップ、コンテキストデマップでは、有効なコンテキスト番号を指定すること。IMMU で 01_2 または 11_2, DMMU で 11_2 を指定した場合は、無関係な sTLB のエントリをデマップするかもしれない。

---

sTLB の全デマップでは、ページ情報に係わらずすべてのエントリがデマップされる。

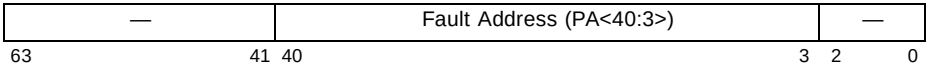
共有コンテキストを直接指示してデマップする方法はない。

Programming Note – 共有コンテキストの TTE のデマップは、セカンダリコンテキストを一時的に変更して行うことができる。

F.10.12 Synchronous Fault Physical Addresses

JPS1 **Commonality** では、IMMU, DMMU で例外が発生した際に論理アドレス情報を記録するレジスタを定義している。SPARC64 VIIIfx ではこれらに加え、物理アドレスを記録するレジスタを定義する。

レジスタ名	`ASI_IMMU_SFPAR`, `ASI_DMMU_SFPAR`
ASI	50_{16} (IMMU), 58_{16} (DMMU)
VA	78_{16}
アクセス種別	Supervisor read/write



`I/D-SFPAR` は、例外を起こしたメモリアクセスの物理アドレス情報を表示するレジスタである。*instruction/data_access_error* が通知され、`I/D-SFSR` の `MK`, `UE`, `BERR`, `BRTO` のどれかに 1 がセットされるときに更新される。

F.11 MMU Bypass

SPARC64 VIIIfx では、以下の 2 つの MMU バイパス ASI が定義されている。

- ASI_ATOMIC_QUAD_LDD_PHYS (ASI 34_{16})
- ASI_ATOMIC_QUAD_LDD_PHYS_LITTLE (ASI $3C_{16}$)

各バイパス ASI 使用時に適用されるページ属性を TABLE F-13 に示す。表の上 4 列の属性は JPS1 **Commonality** の TABLE F-15 で定義された属性に一致する。

TABLE F-13 バイパス ASI のページ属性

ASI 名	ASI 値	属性ビット							
		CP	IE	CV	E	P	W	NFO	Size
ASI_PHYS_USE_EC	14_{16}	1	0	0	0	0	1	0	8 Kbytes
ASI_PHYS_USE_EC_LITTLE	$1C_{16}$								
ASI_PHYS_BYPASS_EC_WITH_EBIT	15_{16}	0	0	0	1	0	1	0	8 Kbytes
ASI_PHYS_BYPASS_EC_WITH_EBIT_LITTLE	$1D_{16}$								
ASI_ATOMIC_QUAD_LDD_PHYS	34_{16}	1	0	0	0	0	0	0	8 Kbytes
ASI_ATOMIC_QUAD_LDD_PHYS_LITTLE	$3C_{16}$								

F.12 Translation Lookaside Buffer Hardware

F.12.2 TLB Replacement Policy

Automatic TLB Replacement

I/D MMU Data In レジスタによる TLB 書き込みでは、ハードウェアが入れ替える TLB 種類とエントリを選択する。その選択基準は以下の通りである。

1. 以下のすべての条件に当てはまる場合は sTLB が、そうでなければ fTLB が選択される。
   - 書き込むエントリは TTE.L = 0 かつ TTE.G = 0
   - ページサイズが、
     ASI_MCNTL.mpg_sITLB/mpg_sDTLB = 0 なら、8KB か 4MB
     ASI_MCNTL.mpg_sITLB/mpg_sDTLB = 1 なら、I/DTLB_TAG_ACCESS_EXT コンテキストレジスタのページサイズに一致するとき
   - ASI_MCNTRL.fw_fITLB/fDTLB = 0
2. sTLBが選択された場合、TLB Tag Access の VA からページサイズに応じたビット位置を切り出して、インデクス番号とする。そのインデクスのエントリの LRU により追い出すエントリを決定する。

3. fTLB が選択された場合、以下の順で決定する。

   a. エントリ 0 から探索して最初に見つかった空きエントリが使われる。空きエントリがなければ、

   b. エントリ 0 から探索して最初に見つかった、ロックされておらず、used ビット[1] が 0 のエントリが使われる。

   c. そのようなエントリが見つからない場合、全エントリの used ビットを 0 クリアして、b をやりなおす。

   全エントリがロックされている場合、TLB の書き込みは行われず、例外も通知されない。

4. fTLB に書き込む際は、多重ヒットの確認が行われる。すでに fTLB にある TTE と書き込もうとする TTE が比較され、多重ヒットになる場合、新規 TTE の書き込みは行われない。

Restrictions on Direct Replacement of sTLB Entries

I/D MMU Data Access レジスタによる sTLB 書き込みでは、書き込む TTE に制約はない。TTE のページサイズと sTLB のページサイズが一致するかどうかの検査は行われない。

1. TLB 内部にあるフラグ。ソフトウェアからは見えない。

F.APPENDIX **G**

Assembly Language Syntax

G.1 Notation Used

G.1.5 Other Operand Syntax

JPS1 **Commonality** のソフトウェアトラップの定義を以下のように修正する。

software_trap_number

以下のいずれかの形式である

reg_{rs1}	(reg_{rs1} + `%g0` と等価)
reg_{rs1} + *imm7*	
reg_{rs1} – *imm7*	
imm7	(`%g0` + *imm7* と等価)
imm7 + reg_{rs1}	(reg_{rs1} + *imm7* と等価)
reg_{rs1} + reg_{rs2}	

ここで、*imm7* は 7 ビットで表現可能な符号なし整数である。オペランドの値 (ソフトウェアトラップ番号) は 0–127 の間の値でなければならない。

G.4 HPC-ACE 拡張機能の表記法

いくつかの命令については HPC-ACE で機能が拡張されているが、この拡張が使われるかどうかは、命令実行時点の XAR の値により決まる。一般的には SXAR と演算命令を組み合わせて指示する。この節ではアセンブリ言語で HPC-ACE 拡張機能を指示するための表記法を定義する。

命令仕様の HPC-ACE 拡張には、拡張されたレジスタの使用、SIMD 演算、セクタキャッシュ指定、ハードウェアプリフェッチのオン / オフなどがある。仕様上はこれらはすべて SXAR で指示するが、アセンブリ言語上の表記は見やすさを考慮して以下のようにする。

1. SXAR は sxar1 または sxar2 と表記する。引数はなし。
2. 拡張レジスタは対象命令の中で直接指定する。
3. それ以外の拡張機能は、対象命令のニーモニックにサフィックスをつけて指示する。
4. ニーモニックで指示された機能は、その命令から遡って最初に出現する SXAR で指示されているものとする。対象命令と SXAR の間にラベルがあったとしても、そのラベルへと分岐してくる命令列の方向へは遡らない。

2 または 3 で表記された命令は、その 1 命令または 2 命令前に先行する SXAR がなくてはならない。先行する SXAR をアセンブラが推測して自動挿入することは、不可能なケースがあるので、SXAR は省略できないものとする。

一方、SXAR と対象命令の間にラベルを許すかどうかは規定しない。4 によりどの SXAR で HPC-ACE 拡張機能を指示するかは明確なためである。

G.4.1 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 1 文字で指定する。TABLE G-1 に拡張指示サフィックスを示す。

TABLE G-1 HPC-ACE 拡張命令で使用できるサフィックス

XAR 表記	サフィックス	備考
XAR.simd	s	
XAR.dis_hw_pf	d	

TABLE G-1 HPC-ACE 拡張命令で使用できるサフィックス

XAR 表記	サフィックス	備考
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。

例 1: 拡張レジスタ使用と SIMD 演算

```
sxar2
fmaddd    %f0, %f2, %f510    /* 拡張レジスタ使用、non-SIMD */
fmaddd,s  %f0, %f2, %f4      /*SIMD, 拡張側で拡張レジスタ使用 */
```

例 2: sector 1 に SIMD load

```
sxar1
ldd,s1    [%xg24], %f0       /* サフィックスは 1s でも可 */
```

F.APPENDIX **H**

Software Considerations

JPS1 **Commonality** の Appendix H を参照。

F.APPENDIX **I**

Extending the SPARC V9 Architecture

JPS1 **Commonality** の Appendix I を参照。

F.APPENDIX **J**

Changes from SPARC V8 to SPARC V9

JPS1 **Commonality** の Appendix J を参照。

F.APPENDIX **K**

Programming with the Memory Models

JPS1 **Commonality** の Appendix K を参照。

F.APPENDIX **L**

Address Space Identifiers

この章では SPARC64 VIIIfx でサポートされている ASI の一覧を示し、特殊な ASI について解説する。

L.2 ASI Values

SPARC V9 の空間識別子 (Address Space Identifier:ASI) は、制限ありのものと制限なしの 2 つにわけられる。00_{16}–$7F_{16}$ は制限ありで、80_{16}–FF_{16} は制限なしである。制限ありの ASI を非特権ソフトウェアが使おうとすると、*privileged_action* 例外が通知される。

さらに ASI は、MMU により変換されるもの (translating)、MMU をバイパスするもの (bypass)、CPU 内部の資源にアクセスするもの (nontranslating) の 3 つに分けられる。SPARC64 VIIIfx の定義を TABLE L-1 に示す。

Compatibility Note – JPS1 **Commonality** ではこの三種類に、実装依存や未定義 ASI が含まれていたが、SPARC64 VIIIfx 仕様では定義された ASI のみを含むように分類されている。

TABLE L-1 SPARC64 VIIIfx の ASI

種類	該当する ASI
Translating ASIs	制限あり 04_{16}, $0C_{16}$, 10_{16}, 11_{16}, 18_{16}, 19_{16}, 24_{16}, $2C_{16}$, 70_{16}–73_{16}, 78_{16}, 79_{16} 制限なし 80_{16}–83_{16}, 88_{16}–$8B_{16}$, $C0_{16}$–$C5_{16}$, $C8_{16}$–CD_{16}, $D0_{16}$–$D3_{16}$, $D8_{16}$–DB_{16}, $E0_{16}$, $E1_{16}$, $F0_{16}$–$F3_{16}$, $F8_{16}$, $F9_{16}$
Bypass ASIs	制限あり 14_{16}, 15_{16}, $1C_{16}$, $1D_{16}$, 34_{16}, $3C_{16}$
Nontranslating ASIs	制限あり 45_{16}, 48_{16}–$4C_{16}$, $4F_{16}$, 50_{16}, 53_{16}–58_{16}, $5C_{16}$–60_{16}, 67_{16}, $6D_{16}$–$6F_{16}$, 74_{16}, 77_{16}, $7F_{16}$ 制限なし $E7_{16}$, EF_{16}

ASI の種類は Data Watchpoint とも関連している。詳細は、JPS1 **Commonality** および本論理仕様書の *“Data Watchpoint Registers”* (page 36) を参照。

L.3 SPARC64 VIIIfx ASI Assignments

SPARC V9 プロセッサでは、メモリアクセス命令で指示するアドレスは、8 ビットの空間識別子 (Address Space Identifier) と論理アドレス (VA) の組で一意に定まるアドレスを使用する。命令フェッチや ASI を明示しないメモリアクセス命令は、暗黙の ASI がハードウェアによって付加される。load from alternate, store from alternate 命令等の ASI を明示する命令の場合、`%asi` レジスタまたは命令により直に指定される ASI が使われる。その他、メモリ空間をアクセスするのではなく、MMU やハードウェアバリアなど CPU レジスタ内のレジスタにアクセスするために使われる ASI もある。

Section L.3.1 の情報は JPS1 **Commonality** と SPARC64 VIIIfx 拡張の両方を網羅している。

L.3.1 Supported ASIs

TABLE L-2 には、SPARC V9 で定義された ASI、SPARC V9 では定義されていないが JPS1 プロセッサで必須の ASI、および SPARC64 VIIIfx で定義された ASI のリストである。網掛けされた部分は、SPARC V9 または JPS1 では定義されていたが SPARC64 VIIIfx では定義されていない ASI である。

黒丸 (●) がついているのは SPARC V9 定義の ASI である。任意のサイズのメモリアクセスで使用できる。

白丸 (❍) がついているのは SPARC V9 では定義されていないが JPS1 プロセッサで必須の ASI である。これらは特に記述がない限り LDXA, STXA, LDDFA, STDFA 命令でのみ使用できる。

星印 (★) がついているのは SPARC64 VIIIfx で定義された ASI である。これらは特に記述がない限り LDXA, STXA, LDDFA, STDFA 命令でのみ使用できる。

ASI の有効な論理アドレスは、TABLE L-2 の VA, Effective Bits, Alignment の列により規定される。

- VA の列は論理アドレスを示す。“—” と表示されている場合は、任意のアドレスが指定できる。“encode” と表示されている場合は各 ASI の説明を参照。
- 有効ビットで、VA のどのビットが有効かを示す。有効でないビットは無視される。
  - “full” は 64 ビットすべてが有効。
  - “physical” は物理アドレスのビット幅までが有効。
  - bit<a:b> はビット位置 a から b までが有効。
- アラインの列には、アラインメントに制約がある場合はその条件が示され、アラインメントに制約がない場合は “—” となっている。アラインメント違反に対して通知される例外の種類は、各命令の説明を参照。

未定義の ASI および許されていない命令との組み合わせで起こる例外については、Appendix L.3.3 を参照。

TABLE L-2 SPARC64 VIIIfx ASIs *(1 of 5)*

ASI	VA	有効ビット	アライン	ASI 名 (と省略表記)	アクセス	ページ
● 04_{16}	—	full	—	ASI_NUCLEUS (ASI_N)	RW	
● $0C_{16}$	—	full	—	ASI_NUCLEUS_LITTLE (ASI_NL)	RW	
● 10_{16}	—	full	—	ASI_AS_IF_USER_PRIMARY (ASI_AIUP)	RW	
● 11_{16}	—	full	—	ASI_AS_IF_USER_SECONDARY (ASI_AIUS)	RW	
❍ 14_{16}	—	physical	—	ASI_PHYS_USE_EC	RW	
❍ 15_{16}	—	physical	—	ASI_PHYS_BYPASS_EC_WITH_EBIT	RW	
● 18_{16}	—	full	—	ASI_AS_IF_USER_PRIMARY_LITTLE (ASI_AIUPL)	RW	
● 19_{16}	—	full	—	ASI_AS_IF_USER_SECONDARY_LITTLE (ASI_AIUSL)	RW	
❍ $1C_{16}$	—	physical	—	ASI_PHYS_USE_EC_LITTLE (ASI_PHYS_USE_EC_L)	RW	
❍ $1D_{16}$	—	physical	—	ASI_PHYS_BYPASS_EC_WITH_EBIT_LITTLE (ASI_PHYS_BYPASS_EC_WITH_EBIT_L)	RW	
❍ 24_{16}	—	full	16byte	ASI_NUCLEUS_QUAD_LDD	R	

TABLE L-2 SPARC64 VIIIfx ASIs *(2 of 5)*

	ASI	VA	有効ビット	アライン	ASI 名 (と省略表記)	アクセス	ページ
❍	$2C_{16}$	—	full	16byte	ASI_NUCLEUS_QUAD_LDD_LITTLE (ASI_NUCLEUS_QUAD_LDD_L)	R	
★	34_{16}	—	physical	16byte	ASI_ATOMIC_QUAD_LDD_PHYS	R	88
★	$3C_{16}$	—	physical	16byte	ASI_ATOMIC_QUAD_LDD_PHYS_LITTLE	R	88
❍	45_{16}	00_{16}	bit<7:0>	8byte	ASI_DCU_CONTROL_REGISTER (ASI_DCUCR)	RW	34
❍	45_{16}	08_{16}	bit<7:0>	8byte	ASI_MEMORY_CONTROL_REG (ASI_MCNTL)	RW	184
★	46_{16}	00_{16}	bit<7:0>	8byte	reserved	R	
★	47_{16}	00_{16}	bit<7:0>	8byte	reserved	R	
❍	48_{16}	00_{16}	bit<7:0>	8byte	ASI_INTR_DISPATCH_STATUS (ASI_MONDO_SEND_CTRL)	R	242
❍	49_{16}	00_{16}	bit<7:0>	8byte	ASI_INTR_RECEIVE (ASI_MONDO_RECEIVE_CTRL)	RW	243
★	$4A_{16}$	—	bit<7:0>	8byte	ASI_SYS_CONFIG	R	323
★	$4B_{16}$	00_{16}	bit<7:0>	8byte	ASI_STICK_CNTL	RW	324
❍	$4C_{16}$	00_{16}	bit<7:0>	8byte	ASI_ASYNC_FAULT_STATUS (ASI_AFSR)	RW	285
★	$4C_{16}$	08_{16}	bit<7:0>	8byte	ASI_URGENT_ERROR_STATUS (ASI_UGESR)	R	275
★	$4C_{16}$	10_{16}	bit<7:0>	8byte	ASI_ERROR_CONTROL (ASI_ECR)	RW	270
★	$4C_{16}$	18_{16}	bit<7:0>	8byte	ASI_STATE_CHANGE_ERROR_INFO (ASI_STCHG_ERR_INFO)	RW	272
❍	$4D_{16}$	00_{16}			ASI_ASYNC_FAULT_ADDR (ASI_AFAR)	R	
★	$4F_{16}$	00_{16}	bit<7:0>	8byte	ASI_SCRATCH_REG0	RW	220
★	$4F_{16}$	08_{16}	bit<7:0>	8byte	ASI_SCRATCH_REG1	RW	220
★	$4F_{16}$	10_{16}	bit<7:0>	8byte	ASI_SCRATCH_REG2	RW	220
★	$4F_{16}$	18_{16}	bit<7:0>	8byte	ASI_SCRATCH_REG3	RW	220
★	$4F_{16}$	20_{16}	bit<7:0>	8byte	ASI_SCRATCH_REG4	RW	220
★	$4F_{16}$	28_{16}	bit<7:0>	8byte	ASI_SCRATCH_REG5	RW	220
★	$4F_{16}$	30_{16}	bit<7:0>	8byte	ASI_SCRATCH_REG6	RW	220
★	$4F_{16}$	38_{16}	bit<7:0>	8byte	ASI_SCRATCH_REG7	RW	220
❍	50_{16}	00_{16}	bit<7:0>	8byte	ASI_IMMU_TAG_TARGET	R	
❍	50_{16}	18_{16}	bit<7:0>	8byte	ASI_IMMU_SFSR	RW	194
❍	50_{16}	28_{16}	bit<7:0>	8byte	ASI_IMMU_TSB_BASE	RW	193
❍	50_{16}	30_{16}	bit<7:0>	8byte	ASI_IMMU_TAG_ACCESS	RW	193
❍	50_{16}	48_{16}			ASI_IMMU_TSB_PEXT_REG	RW	
❍	50_{16}	58_{16}			ASI_IMMU_TSB_NEXT_REG	RW	
★	50_{16}	60_{16}	bit<7:0>	8byte	ASI_IMMU_TAG_ACCESS_EXT	RW	191
★	50_{16}	78_{16}	bit<7:0>	8byte	ASI_IMMU_SFPAR	RW	202
❍	51_{16}	00_{16}			ASI_IMMU_TSB_8KB_PTR_REG	R	
❍	52_{16}	00_{16}			ASI_IMMU_TSB_64KB_PTR_REG	R	

TABLE L-2 SPARC64 VIIIfx ASIs *(3 of 5)*

	ASI	VA	有効ビット	アライン	ASI 名 (と省略表記)	アクセス	ページ
★	53_{16}	—	bit<7:0>	8byte	ASI_SERIAL_ID	R	220
❍	54_{16}	—	bit<7:0>	8byte	ASI_ITLB_DATA_IN_REG	W	192
❍	55_{16}	encode	bit<17:0>	8byte	ASI_ITLB_DATA_ACCESS_REG	RW	192
❍	56_{16}	encode	bit<17:0>	8byte	ASI_ITLB_TAG_READ_REG	R	193
❍	57_{16}	encode	full	8byte	ASI_IMMU_DEMAP	W	201
❍	58_{16}	00_{16}	bit<7:0>	8byte	ASI_DMMU_TAG_TARGET_REG	R	
❍	58_{16}	08_{16}	bit<7:0>	8byte	ASI_PRIMARY_CONTEXT_REG	RW	188
❍	58_{16}	10_{16}	bit<7:0>	8byte	ASI_SECONDARY_CONTEXT_REG	RW	188
❍	58_{16}	18_{16}	bit<7:0>	8byte	ASI_DMMU_SFSR	RW	194
❍	58_{16}	20_{16}	bit<7:0>	8byte	ASI_DMMU_SFAR	RW	
❍	58_{16}	28_{16}	bit<7:0>	8byte	ASI_DMMU_TSB_BASE	RW	193
❍	58_{16}	30_{16}	bit<7:0>	8byte	ASI_DMMU_TAG_ACCESS	RW	193
❍	58_{16}	38_{16}	bit<7:0>	8byte	ASI_DMMU_WATCHPOINT_REG	RW	36
❍	58_{16}	40_{16}			ASI_DMMU_PA_WATCHPOINT_REG	RW	
❍	58_{16}	48_{16}			ASI_DMMU_TSB_PEXT_REG	RW	
❍	58_{16}	50_{16}			ASI_DMMU_TSB_SEXT_REG	RW	
❍	58_{16}	58_{16}			ASI_DMMU_TSB_NEXT_REG	RW	
★	58_{16}	60_{16}	bit<7:0>	8byte	ASI_DMMU_TAG_ACCESS_EXT	RW	191
★	58_{16}	68_{16}	bit<7:0>	8byte	ASI_SHARED_CONTEXT_REG	RW	189
★	58_{16}	78_{16}	bit<7:0>	8byte	ASI_DMMU_SFPAR	RW	202
❍	59_{16}	00_{16}			ASI_DMMU_TSB_8KB_PTR_REG	R	
❍	$5A_{16}$	00_{16}			ASI_DMMU_TSB_64KB_PTR_REG	R	
❍	$5B_{16}$	00_{16}			ASI_DMMU_TSB_DIRECT_PTR_REG	R	
❍	$5C_{16}$	—	bit<7:0>	8byte	ASI_DTLB_DATA_IN_REG	W	192
❍	$5D_{16}$	encode	bit<17:0>	8byte	ASI_DTLB_DATA_ACCESS_REG	RW	192
❍	$5E_{16}$	encode	bit<17:0>	8byte	ASI_DTLB_TAG_READ_REG	R	193
❍	$5F_{16}$	encode	full	8byte	ASI_DMMU_DEMAP	W	201
❍	60_{16}	—	bit<7:0>	8byte	ASI_IIU_INST_TRAP	RW	37
★	67_{16}	—	bit<7:0>	8byte	ASI_FLUSH_L1I	W	233
★	$6D_{16}$	00_{16}–58_{16}	bit<7:0>	8byte	ASI_BARRIER_INIT	RW	224
★	$6E_{16}$	00_{16}	bit<7:0>	8byte	ASI_ERROR_IDENT (ASI_EIDR)	RW	270
★	$6F_{16}$	00_{16}–58_{16}	bit<7:0>	8byte	ASI_BARRIER_ASSIGN	RW	226
❍	70_{16}	—	full	64byte	ASI_BLOCK_AS_IF_USER_PRIMARY (ASI_BLK_AIUP)	RW	
❍	71_{16}	—	full	64byte	ASI_BLOCK_AS_IF_USER_SECONDARY (ASI_BLK_AIUS)	RW	
★	72_{16}	—	full	8byte	ASI_XFILL_AIUP	W	133

TABLE L-2 SPARC64 VIIIfx ASIs *(4 of 5)*

	ASI	VA	有効ビット	アライン	ASI 名 (と省略表記)	アクセス	ページ
★	73_{16}	—	full	8byte	ASI_XFILL_AIUS	W	133
★	74_{16}	—	physical	8byte	ASI_CACHE_INV	W	233
❍	77_{16}	40_{16}	bit<7:0>	8byte	ASI_INTR_DATA0_W	W	242
❍	77_{16}	48_{16}	bit<7:0>	8byte	ASI_INTR_DATA1_W	W	242
❍	77_{16}	50_{16}	bit<7:0>	8byte	ASI_INTR_DATA2_W	W	242
❍	77_{16}	58_{16}			ASI_INTR_DATA3_W	W	
❍	77_{16}	60_{16}			ASI_INTR_DATA4_W	W	
❍	77_{16}	68_{16}			ASI_INTR_DATA5_W	W	
❍	77_{16}	80_{16}			ASI_INTR_DATA6_W	W	
❍	77_{16}	88_{16}			ASI_INTR_DATA7_W	W	
❍	77_{16}	encode|70_{16}	bit<26:24>, bit<16:14>, bit<13:0>	8byte	ASI_INTR_DISPATCH_W	W	242
❍	78_{16}	—	full	64byte	ASI_BLOCK_AS_IF_USER_PRIMARY_LITTLE (ASI_BLK_AIUPL)	RW	
❍	79_{16}	—	full	64byte	ASI_BLOCK_AS_IF_USER_SECONDARY_LITTLE (ASI_BLK_AIUSL)	RW	
❍	$7F_{16}$	40_{16}	bit<7:0>	8byte	ASI_INTR_DATA0_R	R	242
❍	$7F_{16}$	48_{16}	bit<7:0>	8byte	ASI_INTR_DATA1_R	R	242
❍	$7F_{16}$	50_{16}	bit<7:0>	8byte	ASI_INTR_DATA2_R	R	242
❍	$7F_{16}$	58_{16}			ASI_INTR_DATA3_R	R	
❍	$7F_{16}$	60_{16}			ASI_INTR_DATA4_R	R	
❍	$7F_{16}$	68_{16}			ASI_INTR_DATA5_R	R	
❍	$7F_{16}$	80_{16}			ASI_INTR_DATA6_R	R	
❍	$7F_{16}$	88_{16}			ASI_INTR_DATA7_R	R	
●	80_{16}	—	full	—	ASI_PRIMARY (ASI_P)	RW	
●	81_{16}	—	full	—	ASI_SECONDARY (ASI_S)	RW	
●	82_{16}	—	full	—	ASI_PRIMARY_NO_FAULT (ASI_PNF)	R	
●	83_{16}	—	full	—	ASI_SECONDARY_NO_FAULT (ASI_SNF)	R	
●	88_{16}	—	full	—	ASI_PRIMARY_LITTLE (ASI_PL)	RW	
●	89_{16}	—	full	—	ASI_SECONDARY_LITTLE (ASI_SL)	RW	
●	$8A_{16}$	—	full	—	ASI_PRIMARY_NO_FAULT_LITTLE (ASI_PNFL)	R	
●	$8B_{16}$	—	full	—	ASI_SECONDARY_NO_FAULT_LITTLE (ASI_SNFL)	R	
❍	$C0_{16}$	—	full	—	ASI_PST8_PRIMARY (ASI_PST8_P)	W	221
❍	$C1_{16}$	—	full	—	ASI_PST8_SECONDARY (ASI_PST8_S)	W	221
❍	$C2_{16}$	—	full	—	ASI_PST16_PRIMARY (ASI_PST16_P)	W	221

TABLE L-2 SPARC64 VIIIfx ASIs *(5 of 5)*

	ASI	VA	有効ビット	アライン	ASI 名 (と省略表記)	アクセス	ページ
❍	$C3_{16}$	—	full	—	ASI_PST16_SECONDARY (ASI_PST16_S)	W	221
❍	$C4_{16}$	—	full	—	ASI_PST32_PRIMARY (ASI_PST32_P)	W	221
❍	$C5_{16}$	—	full	—	ASI_PST32_SECONDARY (ASI_PST32_S)	W	221
❍	$C8_{16}$	—	full	—	ASI_PST8_PRIMARY_LITTLE (ASI_PST8_PL)	W	221
❍	$C9_{16}$	—	full	—	ASI_PST8_SECONDARY_LITTLE (ASI_PST8_SL)	W	221
❍	CA_{16}	—	full	—	ASI_PST16_PRIMARY_LITTLE (ASI_PST16_PL)	W	221
❍	CB_{16}	—	full	—	ASI_PST16_SECONDARY_LITTLE (ASI_PST16_SL)	W	221
❍	CC_{16}	—	full	—	ASI_PST32_PRIMARY_LITTLE (ASI_PST32_PL)	W	221
❍	CD_{16}	—	full	—	ASI_PST32_SECONDARY_LITTLE (ASI_PST32_SL)	W	221
❍	$D0_{16}$	—	full	—	ASI_FL8_PRIMARY (ASI_FL8_P)	RW	
❍	$D1_{16}$	—	full	—	ASI_FL8_SECONDARY (ASI_FL8_S)	RW	
❍	$D2_{16}$	—	full	—	ASI_FL16_PRIMARY (ASI_FL16_P)	RW	
❍	$D3_{16}$	—	full	—	ASI_FL16_SECONDARY (ASI_FL16_S)	RW	
❍	$D8_{16}$	—	full	—	ASI_FL8_PRIMARY_LITTLE (ASI_FL8_PL)	RW	
❍	$D9_{16}$	—	full	—	ASI_FL8_SECONDARY_LITTLE (ASI_FL8_SL)	RW	
❍	DA_{16}	—	full	—	ASI_FL16_PRIMARY_LITTLE (ASI_FL16_PL)	RW	
❍	DB_{16}	—	full	—	ASI_FL16_SECONDARY_LITTLE (ASI_FL16_SL)	RW	
❍	$E0_{16}$	—	full	—	ASI_BLOCK_COMMIT_PRIMARY (ASI_BLK_COMMIT_P)	W	220
❍	$E1_{16}$	—	full	—	ASI_BLOCK_COMMIT_SECONDARY (ASI_BLK_COMMIT_S)	W	220
★	$E7_{16}$	00_{16}	bit<7:0>	8byte	ASI_SCCR	RW	234
★	EF_{16}	00_{16}–58_{16}	bit<7:0>	8byte	ASI_LBSY, ASI_BST	RW	227
❍	$F0_{16}$	—	full	64byte	ASI_BLOCK_PRIMARY (ASI_BLK_P)	RW	
❍	$F1_{16}$	—	full	64byte	ASI_BLOCK_SECONDARY (ASI_BLK_S)	RW	
★	$F2_{16}$	—	full	8byte	ASI_XFILL_P	W	133
★	$F3_{16}$	—	full	8byte	ASI_XFILL_S	W	133
❍	$F8_{16}$	—	full	64byte	ASI_BLOCK_PRIMARY_LITTLE (ASI_BLK_PL)	RW	
❍	$F9_{16}$	—	full	64byte	ASI_BLOCK_SECONDARY_LITTLE (ASI_BLK_SL)	RW	

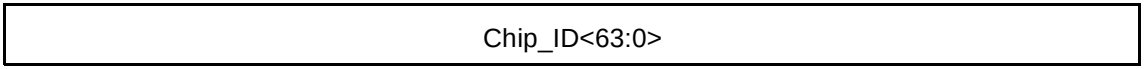
L.3.2 Special Memory Access ASIs

この節では SPARC64 VIIIfx で定義されている ASI について説明する。JPS1 **Commonality** の Section L.3.2 で定義されている ASI はここでは説明しないので、対応する節を参照。

ASI 53_{16} (ASI_SERIAL_ID)

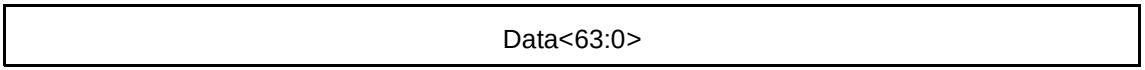
SPARC64 VIIIfx は個々の CPU チップ毎にユニークな ID コードを持っている。この ID コードと VER の情報から、完全に一意な CPU 識別を生成することができる。

このレジスタはリードオンリーで、書き込もうとすると *data_access_exception* 例外が通知される。



ASI $4F_{16}$ (ASI_SCRATCH_REGx)

SPARC64 VIIIfx はシステムソフトウェア用に 8 本の 64 ビットレジスタを用意している。



レジスタ名	ASI_SCRATCH_REG*x* (*x* = 0–7)
ASI	$4F_{16}$
VA	VA<5:3> = レジスタ番号 他のビットは 0 でなければならない。
アクセス種別	Supervisor read/write

Block Load and Store ASIs

ASI 番号 $E0_{16}$ と $E1_{16}$ は Block Store with Commit (page 66) と定義されており、STDFA 命令でのみ使える ASI である。この ASI を LDDFA 命令で指定することはできない。指定した場合の動作は以下の通りである。

- デスティネーションレジスタ rd に関する例外は検出されない (impl. dep. #255)。
- メモリアドレスの境界条件によって、以下の例外が通知される (impl. dep. #256)。

- 8 バイト境界ならば、*data_access_exception* で、DSFSR.FTYPE = 08_{16} (invalid ASI)
- 4 バイト境界ならば、*LDDF_mem_address_not_aligned*
- それ以外なら *mem_address_not_aligned*

Partial Store ASIs

"Partial Store (VIS I)" (page 93)ASI 番号 $C0_{16}$–$C5_{16}$, $C8_{16}$–CD_{16} は Partial Store (page 93) と定義されており、STDFA 命令でのみ使える ASI である。この ASI を LDDFA 命令で指定することはできない。指定した場合の動作は以下の通りである。

- メモリアドレスの境界条件によって、以下の例外が通知される (impl. dep. #257)。
  - 8 バイト境界ならば、*data_access_exception* で、DSFSR.FTYPE = 08_{16} (invalid ASI)
  - 4 バイト境界ならば、*LDDF_mem_address_not_aligned*
  - それ以外なら *mem_address_not_aligned*

L.3.3 ASI と命令の組み合わせと例外

SPARC64 VIIIfx 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 VIIIfx 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。

Block Load/Store, Partial Store 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。LDDA, STDA 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。

5. 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. block load, block store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。

   LDDFA 命令で block store with commit ASI を指定した場合は、block load, block store 命令ではないので、この項には当てはまらない。

   b. 16 ビット short load, short store 命令は 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 short load, short store 命令は 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. partial store 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 を指定した場合は、partial store 命令ではないので、この項には当てはまらない。

e. LDDFA, STDFA で上記以外の ASI を指定し、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。

上記 e および f は、ASI が定義済みか未定義か、ASI と命令の組み合わせが正しいかどうかより優先されるので、*data_access_exception* (FT = 08_{16}) は通知されない。

6. ASI と命令の組み合わせが正しくない場合、*data_access_exception* が通知される。

   ただし PREFETCHA では *data_access_exception* が通知されず、nop として処理される。

L.3.4 内部レジスタの更新タイミング

SPARC64 VIIIfx では、nontranslating ASI のほとんどが、CPU 内部レジスタにマップされている。CPU 内部レジスタは MMU 関連やハードウェアバリアなど、副作用を伴うものがほとんどだが、nontranslating ASI のアクセスがプログラム順序になることを保証しない。CPU 内部レジスタの更新による効果 (副作用) を、レジスタ変更直後の命令に見せるためには、ソフトウェアによる明示的な membar #Sync が必要である。

L.4 ハードウェアバリア

SPARC64 VIIIfx は CPU チップ内で高速に同期を取るためのハードウェアバリア機構を提供する。バリア機構は CPU チップ内にあり、全コアで共有されている。
FIGURE L-1 は SPARC64 VIIIfx のバリア資源の構成図である

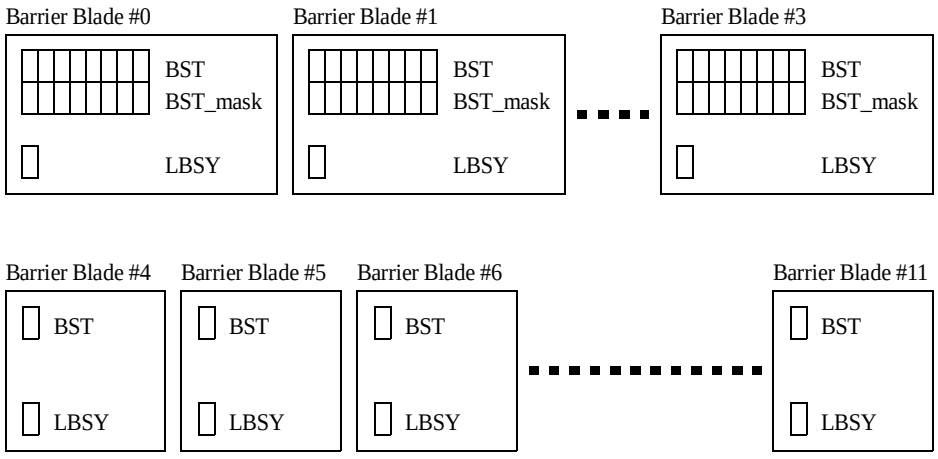


FIGURE L-1 SPARC64 VIIIfx のバリア資源

主要な資源は Barrier Blade (BB) と呼ばれるもので、SPARC64 VIIIfx には 12 個ある。各 BB は BST (Barrier STatus bit) と BST のマスクビット、それに前回同期したときの値を記憶しておく LBSY (Last Barrier SYnchronization status) を持っている。12 個ある BB のうち 4 個は、BST, BST_mask は 8 ビット長で、それぞれがチップ内のコアに固定的に対応している。残り 8 個では BST が 1 ビット長で BST_mask は存在しない。前者は複数スレッド間のバリア同期用、後者は 2 スレッド間での post-wait 同期用を意図している。

同期が成立するのは、BST の選択されたビット (BST_mask で選択する) がすべて 0 または 1 のどちらかに揃ったときである。同期が成立すると、その値 (0 か 1) が LBSY にコピーされる。同期成立と LBSY へのコピーは単一の操作として行われるので、同期成立前に LBSY を読み出すと必ず古い値が読め、同期成立後は必ず新しい値が読める。したがって、ソフトウェアが同期を取る手順は、まず LBSY を読み出し、BST を更新した後で LBSY が変化するのを待つという順序になる。LBSY の変化を監視するためには普通スピンループが使われるが、複数コア・スレッドで資源を共有する CPU では、スピンループは CPU 資源を浪費し他コア・スレッドの実行を阻害するおそれがある。SPARC64 VIIIfx では LBSY の値が変化したとき、SLEEP 命令による休止状態にあるコア・スレッドを実行状態に復帰させる。これにより、高速な同期と CPU 資源の効率的な利用の両立を可能にしている。

LBSY は最後に同期したときの値を覚えているので、ソフトウェアは、次の同期で BST にセットする値を容易に決定することができる。すなわち、LBSY から読み出した値が 0 なら 1 を、1 なら 0 を BST に書き込めばよい。

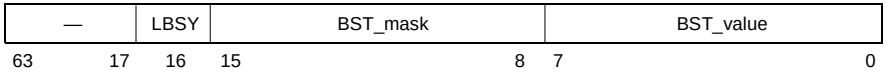
各コア・スレッドは 12 個の BB に対応した 12 個の窓 ASI を持っている。ユーザプログラムはバリア資源に直接アクセスするのではなく、窓を通じてアクセスすることになる。窓を設けることで、BST ビットマップを隠蔽し単一の操作を可能にし、他のユーザプログラムから同期を破壊するような操作を防ぐことができる。

バリア資源のメモリモデルは JPS1 **Commonality** の Chapter 8 で定義されている TSO に準拠する。これは BB 同士、BB とメモリ間いずれにおいても成立する。つまり、store の後の load 以外は命令列に現れた順序に実行される。窓 ASI に対する store の後でメモリまたは `LBSY` を読み出す場合には、間に `membar #storeload` を挟む必要がある。

Note – SPARC64 VIIIfx は CPU チップ間のバリア同期はサポートしない。

L.4.1 バリア資源の初期化と状態獲得

レジスタ名	`ASI_BARRIER_INIT`
ASI	$6D_{16}$
VA	00_{16}, 08_{16}, 10_{16}, 18_{16}, 20_{16}, 28_{16}, 30_{16}, 38_{16}, 40_{16}, 48_{16}, 50_{16}, 58_{16}
アクセス種別	Supervisor read/write



VA で指定される Barrier Blade の値の取得および初期化を行う。read で現在の設定が読み出され、write で新しい設定を書くことができる。

BST_mask, BST_value がバリアのグループ構成とバリアの状態を表わす。各ビットは各コアに対応している。BST_mask には、Barrier Blade を使うコアのビットを 1、使わないコアのビットを 0 に設定する。

ビット	フィールド名	アクセス	説明
63:17	reserved		
16	LBSY	RW	最後に同期したときの BST の値。
15:8	BST_mask	RW	BST のマスク。各ビットとコアの対応は、 • BB#0–BB#3では、 BST_mask<0> コア 0 BST_mask<1> コア 1 BST_mask<2> コア 2 BST_mask<3> コア 3 BST_mask<4> コア 4 BST_mask<5> コア 5 BST_mask<6> コア 6 BST_mask<7> コア 7 • BB#4–BB#11には BST_mask は存在しない。
7:0	BST_value	RW	BST の値。各ビットとコアの対応は、 • BB#0–BB#3では、 BST_value<0> コア 0 BST_value<1> コア 1 BST_value<2> コア 2 BST_value<3> コア 3 BST_value<4> コア 4 BST_value<5> コア 5 BST_value<6> コア 6 BST_value<7> コア 7 • BB#4–BB#11では、 BST_value<0> コア 0–7

読み出し時は、VA で指定される Barrier Blade の BST_value, BST_mask, LBSY が読み出される。

BB#0-#3 では、BST_mask と BST_value の各ビットはそれぞれが固有のコアに対応している。BST_mask のあるビットが 0 のとき、BST_value の対応するビットに読み出される値は不定である。

post/wait 用 BB では、LBSY, BST_value<0> のみが意味を持ち、他のビットは 0 が読み出される。

■ 書き込み時は、VA で指定される Barrier Blade の BST_value, BST_mask, LBSY が更新される。

BB#0-#3 では、BST_mask と BST_value の各ビットはそれぞれが固有のコアに対応している。BST_mask のあるビットが 0 で、BST_value の対応するビットが 1 であるような値を書こうとした場合、その値が書かれる書かれるかどうかは未定義である。

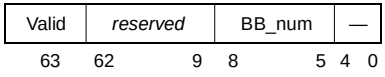
post/wait 用 BB では、LBSY, BST_value<0> のみが意味を持ち、他のビットの書き込みは無視される。

書き込み後、ハードウェアはバリア同期が成立しているかどうかを確認し、それに応じて LBSY を更新する。BST_value, BST_mask にすべて 1 を、LBSY に 0 を書き込むと、LBSY は直ちに 1 に更新される。

bst_mask = 0 の場合バリア同期が成立しているかどうかの確認は行われず、LBSY に書き込んだ値がそのまま保持される。

L.4.2 バリア資源の割り付け

レジスタ名	ASI_BARRIER_ASSIGN
ASI	$6F_{16}$
VA	00_{16}, 08_{16}, 10_{16}, 18_{16}, 20_{16}, 28_{16}, 30_{16}, 38_{16}, 40_{16}, 48_{16}, 50_{16}, 58_{16}
アクセス種別	Supervisor read/write



ASI_BARRIER_ASSIGN は、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 は ASI_BST, ASI_LBSY のそれと対応しており、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。

ビット	フィールド名	アクセス	説明
63	Valid	RW	
62:9	reserved		
8:5	BB_num	RW	窓と BB の対応
4:0	reserved		

- 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、

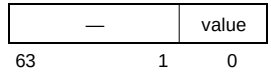
- valid = 1 の場合は、指定された BB_num の LBSY, BST を窓に割りつける。この書き込みの完了以降、ASI_BST への書き込みが BB の BST に反映されるようになり、ASI_LBSY の読み出しにより BB_num にある LBSY が読み出せるようになる。
- valid = 0 の場合は、指定された窓の割りつけを解除する。この書き込みの完了以降、ASI_BST への書き込みは無視され、ASI_LBSY の読み出しには不定値が返る。
- BB_num の値は 0 から 11 までが有効である。12 以上を指定して書き込むと、その書き込みは無視される。

ASI_BARRIER_INIT と ASI_BARRIER_ASSIGN 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・割当に際し矛盾を起こさないようにすること。使用中の Barrier Blade に ASI_BARRIER_INIT を発行する、BST_mask<i> = 0 である BST<i> を ASI_BARRIER_ASSIGN で割りつける、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 を割り当てると意図しない結果が起きることになる。

L.4.3 バリア操作用 ASI

レジスタ名	ASI_LBSY (read), ASI_BST (write)
ASI	EF_{16}
VA	00_{16}, 08_{16}, 10_{16}, 18_{16}, 20_{16}, 28_{16}, 30_{16}, 38_{16}, 40_{16}, 48_{16}, 50_{16}, 58_{16}
アクセス種別	Read/Write



ASI_LBSY, ASI_BST はバリア資源にアクセスするための窓 ASI で、各コア・スレッドに 12 個用意されている。ユーザモードのプログラムが直接アクセスできる。

ビット	フィールド名	アクセス	説明
63:1	reserved		
0	Value	RW	読み出すと LBSY の値が読め、書き込むと BST が更新される。

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 (例外は通知されない)。

Sample Code for Barrier Synchronization

```
    /*
     * %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
```

F.APPENDIX **M**

Cache Organization

M.1 キャッシュタイプ

SPARC64 VIIIfx はチップ上に 2 階層のキャッシュを持つ。

- L1 キャッシュは命令とデータに分かれており、L2 キャッシュはユニファイドキャッシュである。
- L1 キャッシュは論理アドレスインデクス・物理アドレスタグ (VIPT) で、L2 キャッシュは物理アドレスインデクス・物理アドレスタグ (PIPT) である。
- L1 キャッシュ、L2 キャッシュともラインサイズは 128 バイトである。
- L1 キャッシュと L2 キャッシュは包含関係にあり、L1 キャッシュのすべてのデータは L2 キャッシュ上にも載っている。
- L1 命令キャッシュと L1 データキャッシュ間、L1 キャッシュと L2 キャッシュ間のデータの同一性はハードウェアによって維持される。すなわち、
  - L2 キャッシュのあるキャッシュライン上のデータを無効化する場合、L1 キャッシュ上にデータがあればそれも無効化する。
  - 命令列を変更すると、L1 データキャッシュ上のデータが更新され、L1 命令キャッシュの命令列は無効化される。
- L2 キャッシュはプロセッサモジュール上のすべてのコアで共有される。

M.1.1 L1 命令キャッシュ (L1I キャッシュ)

L1 命令キャッシュの諸元を以下に示す。

諸元	値
総容量	32 Kbytes
ウェイ数	2 ウェイ
ラインサイズ	128 バイト
インデクス方式	論理アドレスインデクス・物理アドレスタグ (VIPT)
エラー保護方式 (タグ)	二重化とパリティ
エラー保護方式 (データ)	パリティ
その他	—

SPARC64 VIIIfx はハードウェアがエイリアス処理を行うため、L1 命令キャッシュは VIPT であるが TTE.CV は意味を持たない。

ノンキャッシャブル領域から命令をフェッチする場合、L1 命令キャッシュには命令列は載らない。ノンキャッシャブルアクセスが起きるのは、以下の 3 つの場合である。

- PSTATE.RED = 1
- DCUCR.IM = 0
- TTE.CP = 0

MCNTL.NC_CACHE = 1 の場合、上の条件に係わらず SPARC64 VIIIfx はすべての命令列をキャッシャブル領域にあるものとして扱う。詳細は *"ASI_MCNTL (Memory Control Register)"* (page 184) を参照。

Programming Note – この機能は OBP のために用意されている。OBP がこの機能を使うときは、OBP から抜ける前に MCNTL.NC_CACHE に 0 をセットし、ASI_FLUSH_L1I で L1 命令キャッシュの内容を無効にすること。

M.1.2 L1 データキャッシュ (L1D キャッシュ)

L1 データキャッシュはライトバックキャッシュである。以下に諸元を示す。

諸元	値
総容量	32 Kbytes
ウェイ数	2 ウェイ
ラインサイズ	128 バイト
インデクス方式	論理アドレスインデクス・物理アドレスタグ (VIPT)
エラー保護方式 (タグ)	二重化とパリティ
エラー保護方式 (データ)	ECC
その他	セクタキャッシュ

SPARC64 VIIIfx はハードウェアがエイリアス処理を行うため、L1 データキャッシュは VIPT であるが TTE.CV は意味を持たない。

ノンキャッシャブル領域上のデータにアクセスする場合、そのデータは L1 データキャッシュには載らない。ノンキャッシャブルアクセスが起きるのは、以下の 3 つの場合である。

- ASI_PHYS_BYPASS_EC_WITH_E_BIT (15_{16}) または ASI_PHYS_BYPASS_EC_WITH_E_BIT_LITTLE ($1D_{16}$) 経由のアクセス。
- DCUCR.DM = 0
- TTE.CP = 0

L1 データキャッシュは、MCNTL.NC_CACHE の値に係わらずノンキャッシャブル領域のデータはキャッシュに載せない。

M.1.3 L2 キャッシュ

L2 キャッシュはライトバックキャッシュである。以下に諸元を示す。

諸元	値
総容量	6Mbytes
ウェイ数	12 ウェイ
ラインサイズ	128 バイト
インデクス方式	物理アドレスインデクス・物理アドレスタグ (PIPT)
エラー保護方式 (タグ)	ECC
エラー保護方式 (データ)	ECC
その他	インデクスハッシュ、セクタキャッシュ

L2 キャッシュは、MCNTL.NC_CACHE の値に係わらずノンキャッシャブル領域のデータはキャッシュに載せない。

インデクスハッシュ

SPARC64 VIIIfx は L2 キャッシュのインデックスにハッシュをかける。ハッシュの計算式は以下の通り。

- index<11:9> = PA<33:31> **xor** PA<30:28> **xor** PA<27:25> **xor** PA<24:22> **xor** PA<21:19> **xor** PA<18:16>
- index<8:0> = PA<15:7>

M.2 キャッシュコヒーレンシプロトコル

Note – SPARC64 VIIIfx はマルチプロセッサ構成をサポートしないので、この節は削除した。

M.3 キャッシュ制御 ASI

M.3.1 命令キャッシュフラッシュ (ASI_FLUSH_L1I)

レジスタ名	ASI_FLUSH_L1I
ASI	67_{16}
VA	任意の 8 バイトアライン VA
アクセス種別	Supervisor write のみ

ASI_FLUSH_L1I は、それを実行したコアの L1 命令キャッシュの内容を全部無効にする。起動するには 8 バイト境界にある任意の VA に任意の値を書き込む。

ASI_FLUSH_L1I は書き込みだけが可能であり、読み出しには *data_access_exception* 例外が通知される。

M.3.2 キャッシュデータの無効化 (ASI_CACHE_INV)

レジスタ名	ASI_CACHE_INV
ASI	74_{16}
VA	物理アドレスを指定する
アクセス種別	Supervisor write のみ

ASI_CACHE_INV は、CPU チップ内の全コアの L1 キャッシュおよび L2 キャッシュの特定のキャッシュラインについて、必要ならばその内容をメモリに書き出した上で無効にする。キャッシュラインは VA に物理アドレスを指示して指定する。

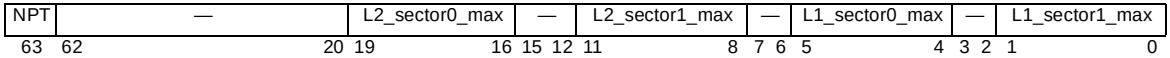
ASI_CACHE_INV は書き込みのみ可能であり、読み出しには *data_access_exception* 例外が通知される。

Note – DCUCR.WEAK_SPCA = 0 だと、ASI_CACHE_INV を発行した時点では無効化されるが、その後の投機実行やハードウェアプリフェッチによりキャッシュに載るかもしれないので、キャッシュ上にデータが残らないことを保証したい場合は、この命令を実行する前に DCUCR.WEAK_SPCA を 1 にセットすること。

M.3.3 セクタキャッシュ設定 (SCCR)

レジスタ名	ASI_SCCR
ASI	$E7_{16}$
VA	00_{16}
アクセス種別	User read/write (制限あり)

ASI_SCCR はセクタキャッシュの設定を制御するレジスタである。SCCR は CPU チップでひとつの資源であり、全コアで共有されている。



ビット	フィールド名	アクセス	説明
63	NPT	RW	特権アクセス。NPT = 1 のとき、PSTATE.priv = 0 で SCCR にアクセスすると *privileged_action* 例外が通知される。NPT = 0 のとき、ユーザ権限で NPT に 1 をセットすることは可能である。
62:20	—		reserved.
19:16	L2_sector0_max	RW	L2 キャッシュのセクタ 0 で使用する最大ウェイ数
15:12	—		reserved.
11:8	L2_sector1_max	RW	L2 キャッシュのセクタ 1 で使用する最大ウェイ数
7:6	—		reserved.
5:4	L1_sector0_max	RW	L1 キャッシュのセクタ 0 で使用する最大ウェイ数。あるコアで設定を変更すると、全コアの L1 キャッシュの設定が変更される。
3:2	—		reserved.
1:0	L1_sector1_max	RW	L1 キャッシュのセクタ 1 で使用する最大ウェイ数。あるコアで設定を変更すると、全コアの L1 キャッシュの設定が変更される。

Warning – SCCR は全コアで共有されているので、あるプロセスがあるコア上でセクタキャッシュを使っているとき、別のコアから SCCR.NPT に 1 をセットすると、セクタキャッシュを使っているプロセスが SCCR にアクセスできなくなる。

SPARC64 VIIIfx はキャッシュを 2 つの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導入する。この機構をセクタキャッシュと呼ぶ。セクタの指示はメモリアクセス命令単位で指定することができ、アクセスされたデータがキャッシュの指定され

たセクタに格納される。SPARC64 VIIIfx 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, L2 独立に設定できる。

セクタの容量はウェイ数で指定する。セットアソシエイティブキャッシュでは、あるインデクスには複数のウェイが存在するが、このうちセクタ 0 に属する最大ウェイ数と、セクタ 1 に属する最大ウェイ数を指定する。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。

セクタキャッシュ機能が有効になるのは、セクタ 0 の容量、セクタ 1 の容量とも 1 以上の値を指定したときである。キャッシュウェイ数より大きな値を指定した場合は、キャッシュウェイ数が指定されたものとみなす。セクタ 0 の最大ウェイ数とセクタ 1 の最大ウェイ数の和は、キャッシュのウェイ数と等しくなくてもよい。どちらかのセクタのウェイ数が 0 の場合はセクタキャッシュ機能は無効となる。

セクタキャッシュ機能は、キャッシュデータの入れ替え時の動作の違いとしてだけ見える。セクタキャッシュ機能が無効のときは、追い出すエントリは全ウェイから選択される。セクタキャッシュ機能が有効のときは、各セクタ毎に設定された最大値を超えないように、追い出すエントリを選ぶ。すなわち、あるインデクスにおいて当該セクタのデータ数が最大値よりも小さければ、当該セクタでない側から追い出すエントリを選び、あるインデクスにおいて当該セクタのデータ数が最大値以上ならば、当該セクタから追い出すエントリを選ぶ。

セクタキャッシュ機能が有効であるかどうかと、キャッシュ上に載っているデータへのアクセスも無関係で、ソフトウェアは常に全ウェイのデータにアクセスできる。そのデータのセクタと異なるセクタを指定してアクセスした場合、そのデータが所属するセクタが変更される。

Notes – 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なので、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。

メモリアクセス命令 (load/store/atomic/prefetch) でキャッシュセクタを指定するには、`XAR.sector` (`XAR.urs3`<0>) を使用する。`XAR.sector` = 0 ならセクタ 0 が、`XAR.sector` = 1 ならセクタ 1 が指定される。

セクタ指定情報とセクタキャッシュ機能は独立した概念である。セクタ指定情報はデータの属性であり、セクタキャッシュ機能はキャッシュ入れ替え時の動作である。セクタキャッシュ機能が無効であっても、セクタ指定情報は常に保存されている。たとえば L1 のセクタキャッシュ機能が無効、L2 のセクタキャッシュ機能が有効であっても、L1 のデータが L2 にライトバックされる際は、L2 のセクタ情報は適切に更新される。

Implementation Note – L1キャッシュ上のセクタ情報の変化をL2に伝達する手段とタイミングは実装依存とする。

各セクタの最大ウェイ数は、キャッシュ上のデータ更新時に参照される情報であり、ウェイ数設定時点で最大ウェイ数を越えるウェイが当該セクタに割り当てられていたとしても、強制的に無効にはしない。例えばあるインデクスでセクタ 0 が 5 ウェイ使用しているときに、セクタ 0 の最大ウェイ数に 2 が指定されたとしても、最大数を越えた 3 ウェイはすぐに無効にされるわけではない。この意味で最大ウェイ数は制御の目標値の役割をしているといえる。

各セクタの用途は本仕様では規定しない。

セクタキャッシュの動作アルゴリズムを説明する。このアルゴリズムは L1, L2 キャッシュとも同じである。なお、以下の説明ではフィールド名の `L1_`, `L2_` は省略し、キャッシュのウェイ数を *nway* と表記する。

SCCR 設定値

- `sector0_max` > 0 かつ `sector1_max` > 0 のときセクタキャッシュ有効
- `sector0_max` = 0 または `sector1_max` = 0 のときはセクタキャッシュ無効
- `sector0_max` + `sector1_max` = nway は成立する必要なし。

セクタキャッシュ管理動作

セクタ 0 として使用されているウェイ数を `sector0_use`、セクタ 1 として使用されているウェイ数を `sector1_use` と表す。常に以下が成立する。

`sector0_use` + `sector1_use` ≤ *nway*
0 ≤ `sector0_use` ≤ *nway*, 0 ≤ `sector1_use` ≤ *nway*

セクタ番号 *S* に対するメモリアクセスが要求された場合の動作

- キャッシュにヒットした場合
  ヒットしたウェイのセクタが *S* と異なる場合はウェイ数管理を調整する。

  `sector`*S*`_use++`, `sector`*T*`_use--` (セクタ *T*: セクタ *S* ではないセクタ)

  `sector`*S*`_use` > `sector`*S*`_max` になり得る (ただし `sector`*S*`_max` < *nway* のとき)。
- キャッシュミスした場合
  - 空のウェイがある場合、そのウェイをセクタ *S* とする。

    `sector`*S*`_use++`

    空のウェイがある場合、`sector`*S*`_use` は `sector`*S*`_max` より大きな値になり得る。
  - `sector`*S*`_use` < min(*nway*, `sector`*S*`_max`) の場合、セクタ *T* の最も古いウェイを置き換え、セクタ *S* とする。

    `sector`*S*`_use++`, `sector`*T*`_use--`

- sector*S*_use ≥ min(*nway*, sector*S*_max) の場合、セクタ *S* の最も古いウェイを置き換え、セクタ *S* とする。

  sector*S*_use, sector*T*_use は変化しない

  sector*S*_use > min(*nway*, sector*S*_max) でも sector*S*_use は減少しない。sector*S*_use が min(*nway*, sector*S*_max) に近づくには、セクタ *T* へのアクセスが必要。

セクタキャッシュ無効時の動作

- キャッシュミスして空ウェイがない場合、リプレースウェイの選択に sector*S*_use, sector*T*_use を使わず、全ウェイから最も古いウェイを選択する。

  セクタキャッシュ機能が無効でも、セクタ指定情報は保存される。

Note – SPARC64 VIIIfx 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。

XAR.sector はすべてのメモリアクセス命令で指定可能であるが、意味があるのは TTE.CP = 1 である空間に対するアクセスのみである。TTE.CP = 0 である空間や nontranslating ASI に対するアクセスでは XAR.sccs の値は無視され、例外は通知されない。

M.4 ハードウェアプリフェッチ

SPARC64 VIIIfx では、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するとハードウェアがプリフェッチを生成する機構が実装されている[1]。対象はキャッシャブル空間へのロード命令、ストア命令であり、PREFETCH, PREFETCHA, LDSTUB, LDSTUBA, SWAP, SWAPA, CASA, CASXA、block load/store, partial store, short load/store, xfill は対象外。

ハードウェアプリフェッチ機構の動作概要は以下の通り。

1. ld/st 命令が L1 キャッシュをミスすると (アドレス A)、そのアドレスの隣接ライン (A+128,A-128) へのアクセス監視を始める。
2. 監視しているアドレスへのアクセスがあると (たとえば A+128)、さらに隣接するライン (A+256) のプリフェッチを生成し、同時にそのライン (A+256) への ld/st アクセスを監視する。

1. ここで言う連続アドレスは、キャッシュライン (128byte) 単位での連続を意味する。

3. A+256 への ld/st アクセスで、A+384 のプリフェッチを生成する。

アクセス監視はキャッシュミスを契機とし、連続アクセスかどうかはキャッシュアクセス (ヒットかミスかによらない) で判断する。

連続回数が増えてくると、より遠くをプリフェッチするようになり、また、L1 キャッシュ へのプリフェッチも併用するようになる。(最初は L2 キャッシュプリフェッチのみ)。

ソフトウェアからハードウェアプリフェッチ機構を制御する方法は 2 つある。

1. ASI_MCNTL.hpf でハードウェアプリフェッチ機構全体の on/off を制御できる。
   詳細は *“ASI_MCNTL (Memory Control Register)”* (page 184) 参照。

2. XAR.dis_hw_pf で命令単位での on/off を制御できる。
   XAR.dis_hw_pf = 1 のとき、ld/st 命令が L1 キャッシュミスしても、隣接アドレスのキャッシュミスを監視しない。XAR.dis_hw_pf = 0 のときは、ld/st 命令の L1 キャッシュミスで、隣接アドレスのキャッシュミス監視を始める (ASI_MCNTL.hpf = 1 のとき)。

Note – SPARC64 VIIIfx 仕様では、ハードウェアプリフェッチ機構が生成するプリフェッチの種類が何になるかは定義しない。

XAR.dis_hw_pf はすべてのメモリアクセス命令で指定可能であるが、意味があるのは TTE.CP = 1 である空間に対するロード、ストア命令のみである。TTE.CP = 0 である空間や nontranslating ASI に対するアクセスおよび、PREFETCH, PREFETCHA, LDSTUB, LDSTUBA, SWAP, SWAPA, CASA, CASXA, block load/store, short load/store, xfill では XAR.dis_hw_pf の値は無視され、例外は通知されない[1]。

1. partial store は XAR 非対象命令なので、ハードウェアプリフェッチの指定はできない。

F.APPENDIX **N**

Interrupt Handling

N.1 Interrupt Vector Dispatch

あるプロセッサ[1] から別のプロセッサに対してインタラプトを発行するとき、ソフトウェアはまずインタラプトのデータを ASI_INTR_DATA_[0-2]W にセットし、次に ASI_INTR_DISPATCH_W に書き込みを行いインタラプトを送出する。送り側のプロセッサは INTR_DISPATCH_STATUS の BUSY ビットをポーリングし、送信が成功したかどうかを確認する。FIGURE N-1 にインタラプト送信の手順を示す。

1. ここではハードウェアの命令実行の主体。SPARC64 VIIIfx ではコアと同義。

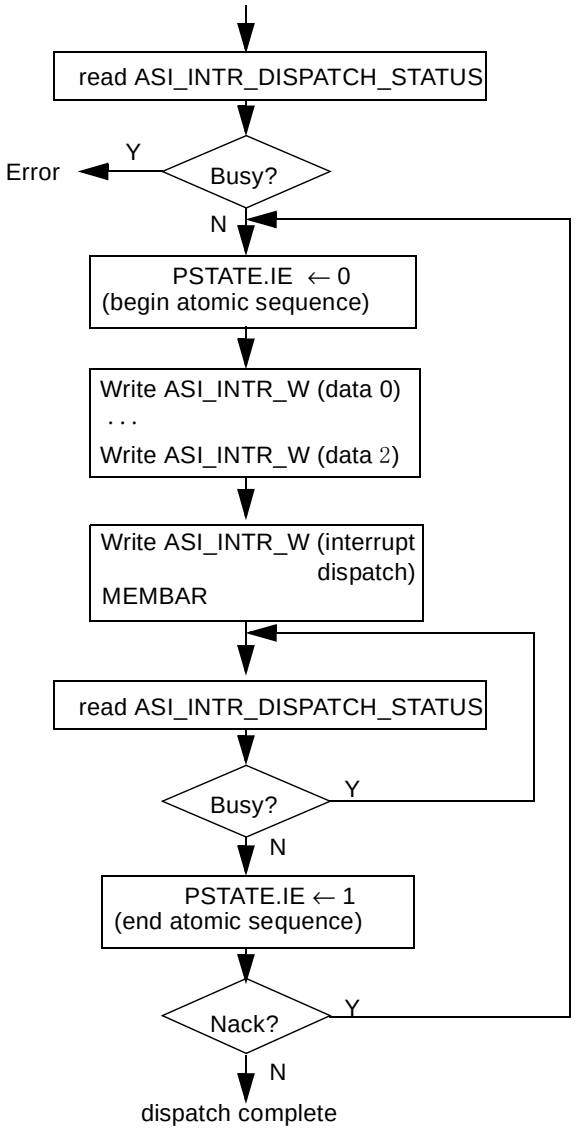


FIGURE N-1 Dispatching an Interrupt

N.2 Interrupt Vector Receive

インタラプトパケットを受信すると、受信データが ASI_INTR_DATA_[0-2]R に格納され、ASI_INTR_RECEIVE レジスタの BUSY ビットに 1 がセットされる。受信したプロセッサがインタラプトを許可している (PSTATE.IE = 1) とトラップが起きる。ソフトウェアはデータを読み出し、その内容に応じてハンドラを選択する。ハンドラによっては、処理を委譲するため、より優先度の低いトラップを起こすこともある。

受信パケットにエラーがあった場合、ASI_INTR_RECEIVE レジスタの BUSY ビットに 1 がセットされない。この場合、ASI_INTR_DATA_[0-2]R もエラーしているあるかも知れないので、読み出してはいけない。詳細は Appendix P.8.3, *"ASI レジスタの処理方法"* (page 289) を参照。

FIGURE N-2 にインタラプト受信の手順を示す。

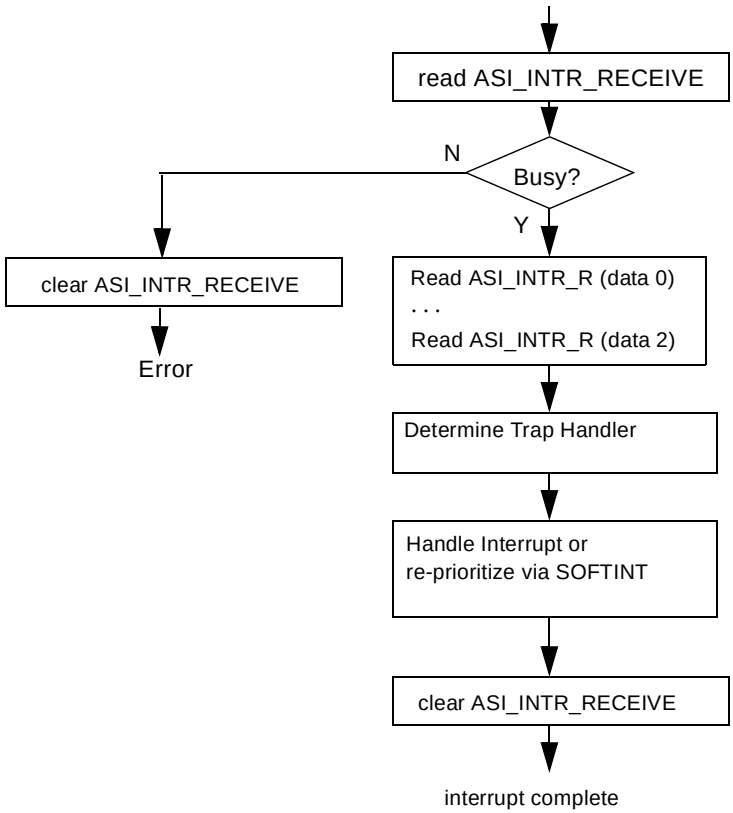


FIGURE N-2 Receiving an Interrupt

N.4 Interrupt ASI Registers

N.4.1 Outgoing Interrupt Vector Data<7:0> Register

JPS1 **Commonality** ではインタラプト送信レジスタを 8 つ定義しているが、SPARC64 VIIIfx ではこれを 3 つにする。`ASI_INTR_DATA_[3-7]W` に対する書き込みは未定義の ASI に準拠する。

Compatibility Note – この変更は SPARC JPS1 非互換である。

N.4.2 Interrupt Vector Dispatch Register

SPARC64 VIIIfx では、`ASI_INTR_DISPATCH_W` レジスタの書き込みの際、`SID`<9:0> (= `VA`<38:29>) の全 10 ビットが無視される (impl.dep.#246)。

SPARC64 VIIIfx は BUSY/NACK ビットは 8 組までを実装する。`ASI_INTR_DISPATCH_W` レジスタの書き込みの際、`BN`<4:3> (= `VA`<28:27>) は無視される。

SPARC64 VIIIfx では、ITID<9:3> (= `VA`<23:17>) は無視される。

N.4.3 Interrupt Vector Dispatch Status Register

SPARC64 VIIIfx では、BUSY/NACK ペアを 8 組実装する。同時に通信できるのは 8 箇所までとなる。

`bit`[63:16] は読み出しに対しては 0 が読みだされる。

N.4.4 Incoming Interrupt Vector Data Registers

JPS1 **Commonality** ではインタラプト受信レジスタを 8 つ定義しているが、SPARC64 VIIIfx ではこれを 3 つにする。`ASI_INTR_DATA_[3-7]R` に対する読み出しは未定義の ASI に準拠する。

Compatibility Note – この変更は SPARC JPS1 非互換である。

N.4.5 Interrupt Vector Receive Register

SPARC64 VIIIfx は 10 ビットの値を SID_H および SID_L フィールドに表示するが、その値は不定である (impl. dep. #247)。

N.6 インタラプト配送先の識別法

SPARC64 VIIIfx は CPU チップ内に複数のコアがある。よって、CPU チップが受信したインタラプトを適切なコアに配送するための判断基準が必要になる。CPU コアの識別子としては ASI_SYS_CONFIG.ITID と ASI_EIDR があり、ファームウェアによって正しく初期化された後は ASI_EIDR が識別の手段となる。

インタラプトパケットが正しく配送されるためには、すべてのコアの ASI_EIDR が初期化済で、ASI_EIDR<2:0> がユニークである必要がある。そのようになっていない状態では、インタラプトパケットが正しく配送されるかどうかは保証されない。

F.APPENDIX **O**

Reset, RED_state, and error_state

この章では、電源投入時およびリセット後の動作について説明する。JPS1 **Commonality** では Section 7.1, *"Processor States, Normal and Special Traps"* (page 43) でリセットの説明をしているが、リセット時の動作はハードウェア実装に強く依存するので、SPARC64™ VIIIfx Extensions ではこの章で説明する。`RED_state` 遷移時のレジスタ値や命令シーケンス開始位置など、ソフトウェアから観測可能な動作については、Section 7.1 を参照。

この章では以下の項目について説明する。

この章は JPS1 **Commonality** の章立てとは一致していない。

O.1 リセット種類

この節ではパワーオンリセット (POR)、ウォッチドッグリセット (WDR)、外部指示リセット (XIR)、ソフトウェア指示リセット (SIR) の 4 種類のリセットについて説明する。

POR と XIR は CPU チップ内の全コアに作用する。言い換えれば、全コアで同じトラップ処理を行う。一方 WDR と SIR はそれを起こしたコアにのみ作用する。他のコアはリセットの影響を受けず実行を継続する。

O.1.1 パワーオンリセット (POR)

SPARC64 VIIIfx で POR を起こすには、外部機構から JTAG で一連の操作を行う必要がある。

リセットピンがアサートされているか、パワーレディ信号がアサートされていないとき、プロセッサの動作は停止し JTAG コマンドのみが実行可能な状態になっている。プロセッサは、JTAG コマンドによる変更以外、ソフトウェアから見える状態を変更せず、メモリシステムの状態も変更しない。

POR を受けると、プロセッサは RED_state に遷移し、*power_on_reset* 例外 (TT = 1) が通知され、RSTVaddr + 20_{16} から命令実行を開始する。

O.1.2 ウォッチドッグリセット (WDR)

ウォッチドッグリセットは以下のときに生成される。

- TL < MAXTL で 2 回目のウォッチドッグタイムアウトを検出したとき。
- TL = MAXTL で 1 回目のウォッチドッグタイムアウトを検出したとき。
- TL = MAXTL でトラップが発生したとき。

ウォッチドッグタイムアウトが発生すると、*watchdog_reset* 例外 (TT = 2) が通知され、RSTVaddr + 40_{16} から命令実行を開始する。それ以外では TT は変更されず、CPU は error_state に遷移する。

O.1.3 外部指示リセット (XIR)

システムから XIR 要求を受けると、プロセッサは RED_state に遷移し、*externally_initiated_reset* 例外 (TT = 3) が通知され、RSTVaddr + 60_{16} から命令実行を開始する。

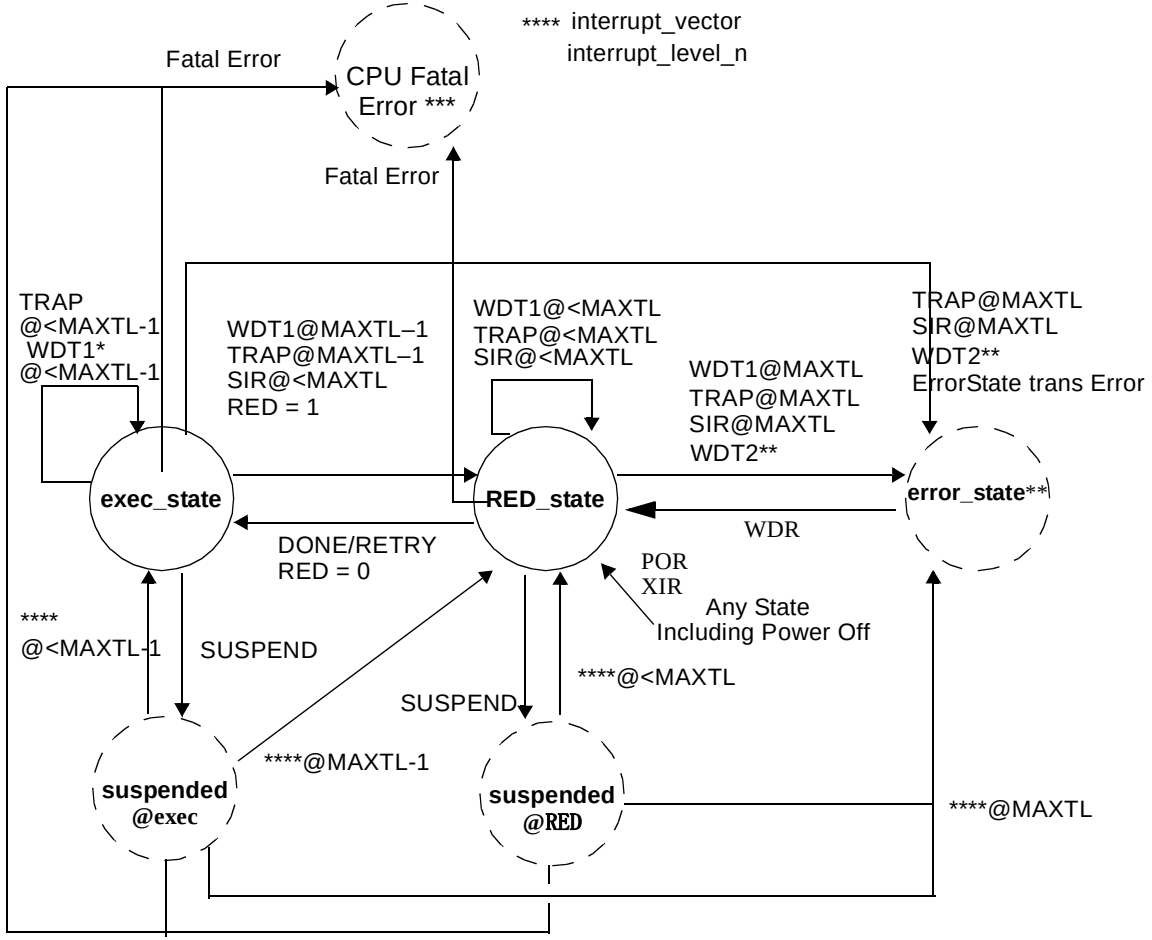
O.1.4 ソフトウェア指示リセット (SIR)

CPU チップ内のどのコアでも、SIR 命令によりソフトウェア指示リセットをかけることができる。

TL < MAXTL (5) で SIR 命令を実行すると、RED_state に遷移し、*software_initiated_reset* 例外 (TT = 4) が通知され、RSTVaddr + 80_{16} から命令実行を開始する。TL = 5 で SIR 命令を実行すると、error_state に遷移し、最終的にはウォッチドッグリセットによる例外が通知される。

O.2 RED_state と error_state

JPS1 **Commonality** の定義に、CPU Fatal Error ステートと suspended ステートが追加されている。



* WDT1 は 1 回目のウォッチドッグタイムアウト。

** WDT2 は 2 回目のウォッチドッグタイムアウト。WDT2 により CPU は error_state に遷移する。通常の設定では、error_state に入ると直ちにウォッチドッグリセットがかかり、CPU は RED_state に遷移するので、error_state にあるのは一時的である。OPSR (Operation Status Register) 設定により、error_state に遷移後ウォッチドッグリセットがかからず、CPU が error_state に留まるようにすることも可能である。

***`CPU_fatal_error_state` では、CPU で致命的エラーが検出されたことを示す `P_FERR` をシステムに通知し、CPU は `CPU_fatal_error_state` ステートに遷移して停止する。

FIGURE O-1 プロセッサの状態遷移図

O.2.1 RED_state

Section 7.1.1, *“RED_state”* (page 43) も参照。

プロセッサが POR 以外の要因で RED_state に遷移した場合、ソフトウェアは execute_state に遷移しようとしてはいけない。遷移した場合、プロセッサは予期せぬ状態になる。

プロセッサがリセットや RED_state に遷移する例外を通知されると、RED_state 用のトラップテーブル相対のトラップベクタ (RSTVaddr) から命令を実行する。SPARC64 VIIIfx では RSTVaddr は VA = FFFF FFFF F000 0000_{16}, PA = 0000 01FF F000 0000_{16} である。

プロセッサはまた、PSTATE.RED = 1 に設定することでも RED_state に遷移する。この場合、RED_state 用のトラップベクタトラップへの分岐は起こらない。

RED_state に遷移する際と、RED_state 中での動作は以下の通りである。

- トラップまたはリセットにより RED_state に遷移した場合、ハードウェアによりいくつかの機能が無効化され、ASI_DCUCR が更新される。ソフトウェアは必要ならこれらのレジスタの値を再設定すること。
- トラップまたはリセット以外の要因で RED_state に遷移した場合、(WRPR で PSTATE.RED ビットに 1 をセットするなど)、DCUCR のビットは更新されない。唯一の副作用は命令の MMU が無効化されることだけである。
- プロセッサが RED_state である間は、DCUCR.IM の値によらず IMMU は無効化される。
- RED_state でもキャッシュの同一性はハードウェアによって保たれる。

O.2.2 error_state

プロセッサは TL = MAXTL (5) でトラップが起きるか、2 回目のウォッチドッグタイムアウトが起きると、error_state に遷移する。

通常の設定では、error_state に入ると直ちにウォッチドッグリセットがかかり、CPU は RED_state に遷移する。OPSR (Operation Status Register) 設定により、error_state に遷移後ウォッチドッグリセットがかからず、CPU が error_state に留まるようにすることも可能である。

O.2.3 CPU Fatal Error

プロセッサは致命的なエラーを検出すると、CPU Fatal Error ステートに遷移する。プロセッサは致命的なエラーがおきたことをシステムに通知し、停止する。

O.3 リセット、RED_state 後のプロセッサ状態

TABLE O-1, TABLE O-2, TABLE O-3 にリセット、RED_state 時のプロセッサの状態を示す。

Programming Note – SPARC64 VIIIfx は、error_state から復帰するために WDR を発生させることができる。ソフトウェアからは、error_state への遷移要因により WDR が発生するように見えるが、ハードウェアの内部では 2 回の遷移が起きている。TABLE O-1, TABLE O-2, TABLE O-3 各表の WDR の列が表しているのは、WDR の前後でレジスタの状態であって、error_state に遷移する際のレジスタ状態の変化は含まれていないことに注意。

TABLE O-1 は、トラップまたはリセットにより RED_state に遷移する場合の特権・非特権レジスタの値である。WRPR 命令で PSTATE.RED ビットに 1 をセットした場合、PSTATE.RED ビット以外の特権・非特権レジスタは変化しない。

TABLE O-1 RED_state およびリセット後の特権・非特権レジスタの値 *(1 of 2)*

名前	POR[1]	WDR[2]	XIR	SIR	RED_state
整数レジスタ	不定 / 無変更	無変更			
浮動小数点レジスタ	不定 / 無変更	無変更			
RSTV 値	VA = FFFF FFFF F000 0000_{16} PA = 01FF F000 0000_{16}				
PC	RSTV | 20_{16}	RSTV |40_{16}	RSTV | 60_{16}	RSTV | 80_{16}	RSTV | $A0_{16}$
nPC	RSTV | 24_{16}	RSTV | 44_{16}	RSTV | 64_{16}	RSTV | 84_{16}	RSTV | $A4_{16}$
PSTATE					
AG	1 (AG が使われる)				
MG	0 (MG は使われない)				
IG	0 (IG は使われない)				
IE	0 (割り込み禁止)				
PRIV	1 (特権モード)				
AM	0 (64 ビットアドレスモード)				
PEF	1 (FPU 使用可能)				
RED	1 (Red_state)				
MM	00_2 (TSO)				
TLE	0	無変更			
CLE	0	TLE がコピーされる			
TBA<63:15>	不定 / 無変更	無変更			
Y	不定 / 無変更	無変更			
PIL	不定 / 無変更	無変更			

TABLE O-1 RED_state およびリセット後の特権・非特権レジスタの値 *(2 of 2)*

名前	POR[1]	WDR[2]	XIR	SIR	RED_state
CWP	不定 / 無変更	無変更 (レジスタウィンドウトラップ以外)	無変更	無変更	無変更 (レジスタウィンドウトラップ以外)
TT[TL]	1	トラップタイプか 2	3	4	トラップタイプ
CCR	不定 / 無変更	無変更			
ASI	不定 / 無変更	無変更			
TL	MAXTL	min (TL + 1, MAXTL)			
TPC[TL] TNPC[TL]	不定 / 無変更 不定 / 無変更	PC nPC			
TSTATE CCR ASI PSTATE CWP PC nPC	不定 / 無変更	CCR ASI PSTATE CWP PC nPC			
TICK NPT Counter	1 0 からカウント開始	無変更 カウント継続	無変更 0 からカウント開始	無変更 カウント継続	
CANSAVE	不定 / 無変更	無変更			
CANRESTORE	不定 / 無変更	無変更			
OTHERWIN	不定 / 無変更	無変更			
CLEARWIN	不定 / 無変更	無変更			
WSTATE OTHER NORMAL	不定 / 無変更 不定 / 無変更	無変更 無変更			
VER MANUF IMPL MASK MAXTL MAXWIN	0004_{16} 8 マスク値 (定数) で固定 5_{16} 7_{16}				
FSR	0	無変更			
FPRS	不定 / 無変更	無変更			

1.ハードパワーオンリセットは電源投入時。ソフトパワーオンリセットはリセット信号がアサートされた時。

2.1 回目のウォッチドッグタイムアウトリセットが execute_state (PSTATE.RED = 0) で起きると、次のウォッチドッグタイムアウトリセットと TL = MAXTL でのウォッチドッグトラップで RED_state に入る。詳細は Appendix O.1.2 参照。

TABLE O-2 は、トラップまたはリセットにより RED_state に遷移した際の ASR レジスタの値である。WRPR 命令で PSTATE.RED ビットに 1 をセットした場合、ASR レジスタの値は変化しない。

TABLE O-2 RED_state およびリセット後の ASR レジスタの値

ASR	名前	POR[1]	WDR[2]	XIR	SIR	RED_state
16	PCR UT ST Others	 0 0 不定 / 無変更	無変更			
17	PIC	不定 / 無変更	無変更			
18	DCR	常に 0				
19	GSR IM IRND その他	 0 0 不定 / 無変更	 無変更 無変更 無変更			
22	SOFTINT	不定 / 無変更	無変更			
23	TICK_COMPARE INT_DIS TICK_CMPR	 1 0	 無変更 無変更			
24	STICK NPT Counter	 1 0 からカウント開始	 無変更 カウント継続			
25	STICK_COMPARE INT_DIS TICK_CMPR	 1 0	 無変更 無変更			
29	XAR	0	0			
30	XASR	不定 / 無変更	無変更			
31	TXAR[TL]	不定 / 無変更	XAR			

1.ハードパワーオンリセットは電源投入時。ソフトパワーオンリセットはリセット信号がアサートされた時。

2.1 回目のウォッチドッグタイムアウトリセットが execute_state (PSTATE.RED = 0) で起きると、次のウォッチドッグタイムアウトリセットと TL = MAXTL でのウォッチドッグトラップで RED_state に入る。詳細は Appendix O.1.2 参照。

TABLE O-3 は、トラップまたはリセットにより RED_state に遷移した際の ASI レジスタの値である。WRPR 命令で PSTATE.RED ビットに 1 をセットした場合、ASI レジスタの値は変化しない。

TABLE O-3 RED_state およびリセット後の ASI レジスタの値 *(1 of 2)*

ASI	VA	名前	POR[1]	WDR[2]	XIR	SIR	RED_state
45_{16}	00_{16}	DCUCR	0	0			
45_{16}	08_{16}	MCNTL RMD その他	 2 0	 2 0			
48_{16}	00_{16}	INTR_DISPATCH_STATUS	0	無変更			
49_{16}	00_{16}	INTR_RECEIVE	0	無変更			
$4A_{16}$	—	SYS_CONFIG ITID	 システム規定値 / 無変更	 無変更			
$4B_{16}$	00_{16}	STICK_CNTL	0	無変更			
$4C_{16}$	00_{16}	AFSR	不定 / 無変更	無変更			
$4C_{16}$	08_{16}	UGESR	不定 / 無変更	無変更			
$4C_{16}$	10_{16}	ERROR_CONTROL WEAK_ED その他	 1 不定 / 無変更	 1 無変更			
$4C_{16}$	18_{16}	STCHG_ERR_INFO	不定 / 無変更	無変更			
$4F_{16}$	00_{16}–38_{16}	SCRATCH_REGs	不定 / 無変更	無変更			
50_{16}	00_{16}	IMMU_TAG_TARGET	不定 / 無変更	無変更			
50_{16}	18_{16}	IMMU_SFSR	不定 / 無変更	無変更			
50_{16}	28_{16}	IMMU_TSB_BASE	不定 / 無変更	無変更			
50_{16}	30_{16}	IMMU_TAG_ACCESS	不定 / 無変更	無変更			
50_{16}	60_{16}	IMMU_TAG_ACCESS_EXT	不定 / 無変更	無変更			
50_{16}	78_{16}	IMMU_SFPAR	不定 / 無変更	無変更			
53_{16}	—	SERIAL_ID	定数値	定数値			
54_{16}	—	ITLB_DATA_IN	不定 / 無変更	無変更			
55_{16}	—	ITLB_DATA_ACCESS	不定 / 無変更	無変更			
56_{16}	—	ITLB_TAG_READ	不定 / 無変更	無変更			
57_{16}	—	ITLB_DEMAP	不定 / 無変更	無変更			
58_{16}	00_{16}	DMMU_TAG_TARGET	不定 / 無変更	無変更			
58_{16}	08_{16}	PRIMARY_CONTEXT	不定 / 無変更	無変更			
58_{16}	10_{16}	SECONDARY_CONTEXT	不定 / 無変更	無変更			
58_{16}	18_{16}	DMMU_SFSR	不定 / 無変更	無変更			
58_{16}	20_{16}	DMMU_SFAR	不定 / 無変更	無変更			
58_{16}	28_{16}	DMMU_TSB_BASE	不定 / 無変更	無変更			
58_{16}	30_{16}	DMMU_TAG_ACCESS	不定 / 無変更	無変更			

TABLE O-3 RED_state およびリセット後の ASI レジスタの値 *(2 of 2)*

ASI	VA	名前	POR[1]	WDR[2]	XIR	SIR	RED_state
58_{16}	38_{16}	DMMU_WATCHPOINT	不定 / 無変更	無変更			
58_{16}	60_{16}	DMMU_TAG_ACCESS_EXT	不定 / 無変更	無変更			
58_{16}	68_{16}	SHARED_CONTEXT	不定 / 無変更	無変更			
58_{16}	78_{16}	DMMU_SFPAR	不定 / 無変更	無変更			
$5C_{16}$	—	DTLB_DATA_IN	不定 / 無変更	無変更			
$5D_{16}$	—	DTLB_DATA_ACCESS	不定 / 無変更	無変更			
$5E_{16}$	—	DTLB_TAG_READ	不定 / 無変更	無変更			
$5F_{16}$	—	DMMU_DEMAP	不定 / 無変更	無変更			
60_{16}	—	IIU_INST_TRAP	0	無変更			
$6D_{16}$	00_{16}–58_{16}	BARRIER_INIT	0	無変更			
$6E_{16}$	00_{16}	EIDR	0/ 無変更	無変更			
$6F_{16}$	00_{16}-58_{16}	BARRIER_ASSIGN	0	無変更			
77_{16}	40_{16}–50_{16}	INTR_DATA0:2_W	不定 / 無変更	無変更			
77_{16}	70_{16}	INTR_DISPATCH_W	不定 / 無変更	無変更			
$7F_{16}$	40_{16}–50_{16}	INTR_DATA0:2_R	不定 / 無変更	無変更			
$E7_{16}$	00_{16}	SCCR NPT その他	 1 0	無変更			
EF_{16}	00_{16}-58_{16}	LBSY, BST	0	無変更			

1.ハードパワーオンリセットは電源投入時。ソフトパワーオンリセットはリセット信号がアサートされた時。

2.1 回目のウォッチドッグタイムアウトリセットが execute_state (PSTATE.RED = 0) で起きると、次のウォッチドッグタイムアウトリセットと TL = MAXTL でのウォッチドッグトラップで RED_state に入る。詳細は Appendix O.1.2 参照。

O.3.1 Operating Status Register (OPSR)

OPSR は CPU チップの制御レジスタである。OPSR の値は CPU 外部からセットされ、ソフトウェアから変更することはできない。ハードウェアパワーオンリセット時 CPU が動き出す前にスキャンインされ、その後は JTAG コマンドで変更できる。

大部分の OPSR 設定はソフトウェアには見えない。

F.APPENDIX **P**

Error Handling

この章では SPARC64 VIIIfx のエラー時における SPARC64 VIIIfx の振舞いと、エラー復旧のために OS やファームウェアが行うべきことを説明する。

P.1 エラーの分類

SPARC64 VIIIfx ではエラーは以下の 4 つに分類される。

- 致命的エラー (Fatal Error)
- `error_state` 遷移エラー (Error State Transition error)
- 緊急エラー (Urgent Error)
- 抑止可能エラー (Restrainable Error)

SPARC64 VIIIfx は 8 コア / チッププロセッサである。どのコアでどのようなエラーが起きたかを識別する方法は、上記のエラー種類によって異なる。

命令実行に起因するエラー、あるいはスレッド固有のリソースに起因するエラーは、**命令実行に同期したエラー**である。この種のエラーは、エラーを起こしたスレッドだけに報告される。*instruction_access_error* や *data_access_error* は、この種のエラーを通知する例外である。

命令実行に起因しないエラー、あるいはチップ内で共有されるリソースに起因するエラーは、**命令実行に非同期のエラー**である。この種のエラーは、関連するすべてのスレッドに報告される。

エラーマーキングは基本的に命令実行とは非同期に行われる。L1$ や L2$ でマークされていないエラー (Raw UE) が発見された場合、生きている (degraded でない) コア内で最小の `EIDR` でマークされる。

サスペンド状態のスレッドで起きたエラーをどのようにログし報告するかも重要な問題である。致命的エラーを除き、スレッドがサスペンドから抜けだすまでエラー報告は延期される。

P.1.1 致命的エラー

致命的エラーはシステム全体に影響を及ぼすエラーである。

a. システム内のデーター貫性が保証できなくなるようなエラー

キャッシュコヒーレントを壊すようなエラーがこれにあたる。

b. CPU チップ内が制御不能になるようなエラー

致命的エラーを発見すると、CPU は CPU Fatal Error ステートに遷移し、システムに致命的エラーが起きたことを通知した後に停止する。システムは CPU からの致命的エラー通知を受けた後、そのシステムで規定された動作を行う。

すべての致命的エラーは命令実行に非同期である。あるスレッドが致命的エラーを発見すると、CPU チップ内のすべてのスレッドにリセット (Power On Reset: POR) が通知される。これはサスペンド中のスレッドがいるかどうかによらない。

P.1.2 error_state 遷移エラー

error_state 遷移エラー (EE) とは、トラップを上げることができないほどの深刻なエラーである。しかし、そのエラーの影響範囲は CPU 内に限定される。

CPU が error_state 遷移エラーを検出すると、error_state に遷移する。error_state からは、ウォッチドッグリセットで RED_state に遷移し、ウォッチドッグリセットトラップハンドラから命令実行を再開する。

命令実行に非同期な error_state 遷移エラー

命令実行に非同期の error_state 遷移エラーには以下のものがある。この種の EE がスレッドで起きると、コア内の全スレッドの ASI_STCHG_ERROR_INFO にエラー情報が記録され、ウォッチドッグリセット例外が通知される (ただしサスペンドしていないとき)。他のコアのスレッドは影響を受けない。

- EE_TRAP_ADR_UE
- EE_OTHER

命令実行に同期して起きる error_state 遷移エラー

命令実行に同期して起きる error_state 遷移エラーには以下のものがある。この種の EE がスレッドで起きると、当該スレッドの ASI_STCHG_ERROR_INFO にエラー情報が記録され、ウォッチドッグリセット例外が通知される。他のスレッドは影響を受けない。

- EE_SIR_IN_MAXTL
- EE_TRAP_IN_MAXTL
- EE_WDT_IN_MAXTL

- EE_SECOND_WDT

Note – SPARC64 VIIIfx はマルチスレッド CPU ではないので、命令実行に同期して起きる error_state 遷移エラーでも非同期に起きる error_state 遷移エラーでも、エラー情報はそのコアの ASI_STCHG_ERROR_INFO だけに記録される。

P.1.3 緊急エラー

緊急エラー (UGE) とは、直ちにシステムソフトウェアが介入する必要のあるエラーである。以下の種類がある。

- 命令実行を阻害するエラー
  - I_UGE: 命令の緊急エラー
  - IAE: 命令アクセスエラー
  - DAE: データアクセスエラー
- 命令実行とは無関係のエラー
  - A_UGE: 自律性緊急エラー

命令実行を阻害するエラー

命令実行阻害エラーとは、命令実行中に検出される、命令実行を不可能にするようなエラーである。

命令実行阻害エラーを検出したときに ASI_ERROR_CONTROL.WEAK_ED = 0 だと (これは通常の実行状態でシステムソフトウェアによって設定される)、例外が通知される。このエラーはマスクすることができない。ASI_ERROR_CONTROL.WEAK_ED = 1 のときは (多重エラー時や POST/OBP によるリセット処理中)、以下のどれかが起きる。

- 可能ならば、実行を阻害された命令のデスティネーションレジスタに、不定値を書きこみ、命令を完了させる。
- それができなければ、例外を通知する。エラーを起こした命令は、ASI_ERROR_CONTROL.WEAK_ED = 0 のときと同様に実行される。

この種のエラーは 3 種類ある。

- I_UGE **(Instruction Urgent Error: 命令緊急エラー)** —IAE (命令アクセスエラー), DAE (データアクセスエラー) 以外のエラー。I_UGE はさらに 2 種類に分けられる。
  - **プログラムから見えるレジスタに復旧不可能なエラーが起き、命令実行ができなくなった**

    PSTATE, PC, NPC, CCR, ASI, FSR, GSR レジスタで復旧不可能なエラーが起きたときがこれにあたる。一回目のウォッチドッグタイムアウトも I_UGE として扱われる。

- **命令の実行部でエラーが起きたとき**

  演算部や一時レジスタや内部のバスエラーがこれにあたる。

I_UGE は Appendix P.2.2 で言うところのプリエンプティブエラーに相当する。

- IAE **(Instruction Access Error: 命令アクセスエラー)** — JPS1 **Commonality** で定義されている *instruction_access_error* のこと。SPARC64 VIIIfx では、キャッシュやメモリ上の UE を命令フェッチ中に検出すると、IAE が通知される。

  IAE は precise な例外である。

- DAE **(Data Access Error: データアクセスエラー)** — JPS1 **Commonality** で定義されている *data_access_error* のこと。SPARC64 VIIIfx では、キャッシュやメモリ上の UE をデータアクセス中に検出すると、DAE が通知される。

  DAE は precise な例外である。

命令実行とは無関係に発生するエラー

- A_UGE **(Autonomous Urgent Error: 自律性緊急エラー)** — 命令実行とは無関係に発生するエラーで、直ちに処理する必要があるものである。

  通常の命令実行では、`ASI_ERROR_CONTROL.WEAK_ED` は 0 にセットされている。この場合、緊急エラー処理中 (つまり *async_data_error* トラップハンドラ内) では、A_UGE による例外は通知されない。

  そうでない場合、たとえばマルチエラーが起きた場合や POST/OBP のリセットルーチンでは、ソフトウェアによって `ASI_ERROR_CONTROL.WEAK_ED` に 1 がセットされる。この場合、A_UGE による例外は通知されない。

  A_UGEs には 2 種類ある。

  - 重要なリソースでエラーが起きていて、そのリソースを使うと致命的エラーや `error_state` 遷移エラーを起こすような場合。
  - 重要なリソースでエラーが起きていて、OS パニックさせたい場合。

    エラーが起きているリソースを使うとこれ以上実行が続けられないので、OS パニックさせたいとき。

  自律性緊急エラーは disrupting 例外で通知されるが、以下の点で V9 仕様と異なる。

  - `PSTATE.IE` = 0 でもマスクされず例外が通知される。
  - `TPC` が指すアドレスにある命令は終了させることができないかもしれない。終了方法はトラップステートレジスタに表示される。

緊急エラーによる例外通知

緊急エラーが起こり、そのエラーがマスクされていないとき、以下のいずれかの例外でシステムソフトウェアに通知される。

- I_UGE, A_UGE: *async_data_error* 例外
- IAE: *instruction_access_error* 例外

- DAE: *data_access_error* 例外

命令実行とは無関係に起きる緊急エラー

以下のエラーがこれに当たる。あるスレッドでこのエラーが起きると、コア内の全スレッドの ASI_UGESR にエラーが記録され、*async_data_error* 例外が通知される。ただしサスペンド状態を除く。他コアは影響を受けない。

- IAUG_CRE
- IAUG_TSBCTXT
- IUG_TSBP
- IUG_PSTATE
- IUG_TSTATE
- IUG_%F (ただし `f[n]` のパリティエラー以外)
- IUR_%R (ただし `r[n]` と `Y` のパリティエラー以外)
- IUG_WDT
- IUG_DTLB
- IUG_ITLB
- IUG_COREERR

命令実行に同期して起きる緊急エラー

以下のエラーがこれに当たる。あるスレッドでこのエラーが起きると、そのスレッドの ASI_UGESR だけにエラーが記録され、ADE 例外が通知される。ただしサスペンド状態を除く。他スレッドは影響を受けない。

- IUG_%F (`f[n]` のパリティエラーのみ)
- IUR_%R (`r[n]` と `Y` のパリティエラーのみ)

Note – SPARC64 VIIIfx はマルチスレッド CPU ではないので、命令実行に同期して起きる緊急エラーでも非同期に起きる緊急エラーでも、エラー情報はそのコアの `ASI_UGESR` だけに記録される。

P.1.4 抑止可能エラー

抑止可能エラー (Restrainable Error) とは、実行中のプログラムに深刻な影響を与えないため、システムソフトウェアが直ちに処理する必要のないエラーである。優先度の低い disrupting 例外で通知される。

抑止可能エラーには 2 種類ある。

- 訂正不可能だが現在の命令列の実行に影響を与えないエラー

  キャッシュのライトバックやコピーバック時に検出されたエラーがこれにあたる。
- 縮退 (Degradation)

頻繁にエラーを起こしているが、命令実行に深刻な影響を与えないリソースを隔離して使わないようにすることができる。しかしながら、性能は多少犠牲になる。

Compatibility Note – SPARC64 VIIIfx は訂正可能エラー (Correctable Error: CE) を検出した場合、自動で訂正し、ソフトウェアには通知しない。

抑止可能エラーは *ECC_error* で通知される。ただしこれは、PSTATE.IE = 1 で、抑止可能エラーの通知が許可されているときのみである。

DG_U2$, UE_RAW_L2$INSD

これらは命令実行に非同期なエラーである。これらのエラーが発見されると、CPU モジュール内の全スレッドの AFSR にエラーが記録され、*ECC_error* が通知される。ただしサスペンド中のスレッドには例外は通知されない。

DG_D1$sTLB, UE_RAW_D1$INSD

これらは命令実行に非同期なエラーである。これらのエラーが発見されると、コア内の全スレッドの AFSR にエラーが記録され、*ECC_error* が通知される。ただし サスペンド中のスレッドには例外は通知されない。

他のコアには影響を与えない。

UE_DST_BETO

これは命令実行に同期したエラーである。このエラーが発見されると、エラーを起こしたスレッドの AFSR にエラーが記録され、*ECC_error* が通知される。ただしサスペンド中のスレッドには例外は通知されない。他のスレッドには影響を与えない。

P.1.5 instruction_access_error

これは命令実行に同期したエラーである。このエラーが検出されると、エラーを起こしたスレッドの ASI_ISFSR, TPC, ASI_ISFPAR にエラーが記録され、*instruction_access_error* が通知される。他のスレッドには影響を与えない。

P.1.6 data_access_error

これは命令実行に同期したエラーである。このエラーが発見されると、エラーを起こしたスレッドの ASI_DSFSR, ASI_DSFAR, ASI_DSFPAR にエラーが記録され、*data_access_error* が通知される。他のスレッドには影響を与えない。

P.2 エラー処理とエラー制御

P.2.1 エラー処理に必要なレジスタ

TABLE P-1 はエラー処理に必要なレジスタの一覧である。これらのレジスタのうち、ASI_ERROR_CONTROL は、エラーを検出した際に例外として通知するかどうかを制御するレジスタで、ASI_EIDR はエラーマーキング用の識別 ID である。その他のレジスタにはエラーの詳細情報が表示される。

TABLE P-1 エラー処理に必要なレジスタ

ASI	VA	名前	説明されている章
$4C_{16}$	00_{16}	ASI_ASYNC_FAULT_STATUS	P.7.1
$4C_{16}$	08_{16}	ASI_URGENT_ERROR_STATUS	P.4.1
$4C_{16}$	10_{16}	ASI_ERROR_CONTROL	P.2.6
$4C_{16}$	18_{16}	ASI_STCHG_ERROR_INFO	P.3.1
50_{16}	18_{16}	ASI_IMMU_SFSR	F.10.9
50_{16}	78_{16}	ASI_IMMU_SFPAR	F.10.12
58_{16}	18_{16}	ASI_DMMU_SFSR	F.10.9
58_{16}	20_{16}	ASI_DMMU_SFAR	F.10.10 of JPS1 **Commonality**
58_{16}	78_{16}	ASI_DMMU_SFPAR	F.10.12
$6E_{16}$	00_{16}	ASI_EIDR	P.2.5

P.2.2 エラー検出時の動作

ここではエラー検出時の挙動を説明する。

エラー検出を抑止する条件

エラー種類	検出抑止条件
致命的エラー	なし (すべて検出する)。
error_state 遷移エラー	ASI_ECR.WEAK_ED = 1 で大部分のエラーを検出しなくなるが、一部は検出する。
緊急エラー	**I_UGE, IAE, DAE:** • ASI_ECR.WEAK_ED = 1 または SUSPENDED ステートのとき、大部分のエラーを検出しなくなるが、一部は検出する。 **A_UGE:** • SUSPENDEDステートのとき、大部分のエラーを検出しなくなるが、一部は検出する。 • レジスタ使用以外のエラーは、ASI_ECR.WEAK_ED = 1 または個々のエラー独自の条件のときに抑止される。 レジスタ使用のエラーは、個々のエラー独自の条件のときに抑止される。個々のエラー独自の条件があるのはごく一部。
抑止可能エラー	なし。

エラー検出時、例外の通知を抑止する条件

エラー種類	通知抑止条件
致命的エラー	なし (すべて検出する)。
error_state 遷移エラー	なし (すべて検出する)。
緊急エラー	**I_UGE, IAE, DAE:** • SUSPENDED ステートのとき。 **A_UGE:** • ASI_ECR.UGE_HANLDER = 1 のとき。 • ASI_ECR.WEAK_ED = 1 のとき。 トラップマスク中に例外が検出されると、通知は延期される。マスクが開くと *async_data_error* が通知される。 • SUSPENDED ステートのとき。
抑止可能エラー	• ASI_ECR.UGE_HANLDER = 1 のとき。 • ASI_ECR.WEAK_ED = 1 のとき。 • PSTATE.IE = 1 のとき。 • エラーをマスクする設定になっているとき。 マスクはエラー種類に応じて ASI_ECR.RTE_DG と ASI_ECR.RTE_UE がある。 • SUSPENDED ステートのとき。

エラー検出時の動作

エラー種類	動作
致命的エラー	1. CPU はフェイタルステートに遷移する。 2. CPUはシステムにフェイタルエラー検出を通知する。 3. システムが停止する。
error_state 遷移エラー	1. CPU は error_state に遷移する。 2. WDR が CPU に通知される。

エラー種類	動作
緊急エラー	**I_UGE:** • ASI_ECR.UGE_HANLDER = 0 のとき、単独 ADE 例外が通知される。 • ASI_ECR.UGE_HANLDER = 1 のとき、多重 ADE 例外が通知される。 **A_UGE:** • 例外通知がマスクされていないときは、単独 ADE 例外が通知される。 • 例外通知がマスクされているときは、例外通知はペンディングされる。 **IAE:** • ASI_ECR.UGE_HANLDER = 0 のとき、IAE 例外が通知される。 • ASI_ECR.UGE_HANLDER = 1 のとき、多重 ADE 例外が通知される。 **DAE:** • ASI_ECR.UGE_HANLDER = 0 のとき、DAE 例外が通知される。 • ASI_ECR.UGE_HANLDER = 1 のとき、多重 ADE 例外が通知される。
抑止可能エラー	例外通知がマスクされておらず、ASI_AFSR にもエラー情報が表示されていないにも係らず、*ECC_error* が通知されることがある。 1. エラー通知がペンディングされている状態で、ASI_AFSR への書き込みを行い、エラー情報が消されたとき。 2. UEを検出して *ECC_error* が通知されたとき、ペンディングしている DG の情報が、ASI_AFSR への書き込みにより消されたとき。 3. DG を検出して *ECC_error* が通知されたとき、ペンディングしている UE の情報が、ASI_AFSR への書き込みにより消されたとき。 このような例外が通知された場合、システムソフトウェアは例外を無視して処理を続行すべきである。

TPC とエラーを起こした命令の関係

エラー種類	動作
致命的エラー	無関係。
error_state 遷移エラー	無関係。
緊急エラー	**I_UGE:** • TLB書き込みエラーでは、TLB を更新しようとした命令は TPC が指す命令か、命令フロー上それより前の命令のこともある。TLB 書き込みエラーは、書き込み後 DONE/RETRY が実行されるか例外が通知された時点検出される。 • TLB書き込み以外のエラーでは、TPC が指す命令か、命令フロー上それより後の命令でエラーが起きている。 **A_UGE:** • 無関係。 **IAE, DAE** • TPC が指す命令がエラーを起こした命令である。
抑止可能エラー	無関係。

その他

複数種類のエラーが同時に検出された場合の優先順位

致命的エラー	error_state 遷移エラー	緊急エラー	抑止可能エラー
1. フェイタルステートに遷移 (TT = 1)	2. error_state に遷移 (TT = 2)	3. ADE (TT = 40_{16}) 4. DAE (TT = 32_{16}) 5. IAE (TT = $0A_{16}$)	6. ECC_error_trap (TT = 63_{16})

割り込まれた命令の完了方法

致命的エラー	error_state 遷移エラー	緊急エラー	抑止可能エラー
完了できない	完了できない	**ADE:** • P.4.3 参照。 **IAE, DAE:** • JPS1の Precise 定義通り。	JPS1 の Precise 定義通り。

エラー表示レジスタ

致命的エラー	error_state 遷移エラー	緊急エラー	抑止可能エラー

	ASI_STCHG_ ERROR_INFO	**I_UGE, A_UGE:** • ASI_UGESR **IAE:** • ASI_ISFSR **DAE:** • ASI_DSFSR	ASI_AFSR

一回の例外で通知されるエラー数

致命的エラー	error_state 遷移エラー	緊急エラー	抑止可能エラー
すべての致命的エラーが検出される。	すべての error_state 遷移エラーが検出され、ASI_STCHG_ ERROR_INFO に表示される。	**単独 ADE:** • すべての I_UGE, A_UGE が検出される。 **多重 ADE:** • 多重発生したことと、最初の ADE の UGE が表示される。 **IAE:** • 1 つだけ表示される。 **DAE:** • 1 つだけ表示される。	すべての抑止可能エラーが検出され、ASI_AFSR に表示される。

P.2.3 CE 発見時に元データを自動で訂正できる限界

訂正可能エラー (CE) が発見されると、CPU は入ってきたデータを訂正し、演算を続ける。しかし、元のデータを自動で訂正するのは限界がある。以下の箇所のデータは自動で訂正できない。

- メモリの CE
- 外部からの非同期割込みに付属するデータ (INTR_DATA_R)

その他の CE エラーについては元のデータを自動訂正することができる。

INTR_DATA_R の CE については、次回の外部割込みによりエラーデータは上書きされ消えるので、OS によるエラー処理は不要である。メモリの CE は OS による訂正処理が必要である。

P.2.4 キャッシャブルデータのエラーマーキング

キャッシャブルデータのエラーマーキング

最初にキャッシャブルデータに訂正不能エラー (UE) を見つけたハードウェア内のユニットは、そのデータと ECC を特別な値にする。これにより、エラーが認識されていることと、エラー発生元が特定できる。これをエラーマーキングと言う。エラーマーキングはエラー発生源を特定し、一つのエラーにより何度もエラー報告が上がるのを防ぐ。

システム内で ECC で保護されている箇所には以下のものがある。

- メインメモリ
- メモリと ICC からのデータパス
- U2 キャッシュデータ
- D1 キャッシュデータ

CPU がまだマークされていない UE を見つけると、エラーマーキングを行う。

エラーがマークされたかどうかは 8 バイト毎につけられるシンドロームで識別できる。

TABLE P-2 エラーマークに使われるシンドローム

シンドローム	エラーマークの状態	訂正不能エラ (UE)ーの種類
$7F_{16}$	マーク済	マーク済 UE
$7F_{16}$ 以外の複数ビットエラーパターン	まだマークされていない	マークされていない UE (Raw UE)

シンドローム $7F_{16}$ は、3 ビットエラーが起きていることを表わす。エラーマーキングでは、元のデータと ECC を次節で説明するものに置換える。エラー以外でシンドローム $7F_{16}$ が起きる確率はほぼ 0 とみなす。

エラーマーキングデータのフォーマット

キャッシャブルデータで UE が発見されると、エラーデータと ECC はエラーマーキングデータで置換えられる。

TABLE P-3 エラーマーキングデータのフォーマット

Data/ECC	ビット	値
data	63	エラービット。値は不定。
	62:56	0 (7 ビット).
	55:42	ERROR_MARK_ID (14 ビット).
	41:36	0 (6 ビット).
	35	エラービット。値は不定。
	34:23	0 (12 ビット).
	22	エラービット。値は不定。
	21:14	0 (8 ビット).
	13:0	ERROR_MARK_ID (14 ビット).
ECC		ビット 63, 35, 22 に 3 ビットエラーが存在することを示すパターン。シンドロームが $7F_{16}$ になるパターンが設定される。

ERROR_MARK_ID (14bit) は、エラー発生元を示す。エラーを発見したハードウェアユニットが値をセットする。

ERROR_MARK_ID のフォーマットを TABLE P-4 に示す。

TABLE P-4 ERROR_MARK_ID の各ビットの説明

ビット	値
13:12	モジュール ID。エラーが起きたハードウェアを示す。 00_2: メモリシステム (DIMM を含む) 01_2: チャネル 10_2: CPU 11_2: Reserved
11:0	ソース ID。モジュール ID = 00 のとき、ソース ID は常に 0。それ以外では、エラーを発見したハードウェアの ID が入る。

CPU が付加する ERROR_MARK_ID

CPU が付加する ERROR_MARK_ID を TABLE P-5 に示す。

TABLE P-5 CPU が付加する ERROR_MARK_ID

マークされていないエラー (Raw UE) の種類	Module_ID の値	Source_ID の値
メモリバスからの入力データ	00_2 (メモリシステム)	0
メモリバスへの出力データ	10_2 (CPU)	1 0000 0000_2 ▯ 000_2
U2 キャッシュデータ	10_2 (CPU)	1 0000 0000_2 ▯ 000_2
D1 キャッシュデータ	10_2 (CPU)	0 0000 0000_2 ▯ ASI_EIDR<2:0>

P.2.5 ASI_EIDR

ASI_EIDR レジスタは ERROR_MARK_ID の Source_ID に記録するための情報を保持する。ASI_EIDR はまた、インタラプトを受信する CPU の識別にも使われる (Appendix N.6, “ *インタラプト配送先の識別法* ” (page 243) 参照)。

レジスタ名	ASI_EIDR
ASI	$6E_{16}$
VA	00_{16}
エラー検出	パリティ
フォーマット	TABLE P-6 参照。

TABLE P-6 ASI_EIDR のフィールドの説明

ビット	フィールド名	アクセス	説明
63:3	Reserved	R	常に 0。
2:0	ERROR_MARK_ID	RW	CPU でエラーが発生したとき、エラーデータの ERROR_MARK_ID にコピーする。

Compatibility Note – SPARC64 VII 以前の仕様では、ソフトウェアが ASI_EIDR<13:12> に 10_2 を設定し、その値が ERROR_MARK_ID に反映されることになっていた。SPARC64 VIIIfx では Module_ID_Value はハードウェアが固定で持っているため、ソフトウェアによる設定は不要となった。

P.2.6 エラー検出の制御 (ASI_ERROR_CONTROL)

ASI_ERROR_CONTROL は、エラー検出のマスクおよび検出後の動作を設定するレジスタである。

レジスタ名	ASI_ERROR_CONTROL (ASI_ECR)
ASI	$4C_{16}$
VA	10_{16}
エラー検出	なし
フォーマット	TABLE P-7 参照。
リセット後の初期値	ハード POR 時は、WEAK_ED は 1、それ以外は 0 に設定される。 それ以外のリセットでは、UGE_HANDLER と WEAK_ED の値が ASI_STCHG_ERROR_INFO にコピーされ、全フィールドが 0 に設定される。

ASI_ERROR_CONTROL レジスタはエラーの検出時、例外通知時、多重エラー発生時の処理を制御する。レジスタフィールドは以下の通り。

TABLE P-7 ASI_ERROR_CONTROL のフィールドの説明

ビット	フィールド名	アクセス	説明
9	RTE_UE	RW	抑止可能エラーのうち UE, Raw UE を例外で通知するかどうかを指定する。Appendix P.2.2 に記述された処理を行う。
8	RTE_DG	RW	抑止可能エラーのうちデグレードエラーを例外で通知するかどうかを指定する。Appendix P.2.2 に記述された処理を行う。
1	WEAK_ED	RW	エラー検出を弱める。I_UGE, DAE の検出を抑止するかどうかを決める。 0 エラー検出は行われる。 1 CPU が実行を続けられるならば、エラー検出は行われない。 WEAK_ED = 1 で命令実行中に I_UGE, DAE が検出されると、結果出力 (レジスタやメモリ) には不定値が書き込まれる。 WEAK_ED = 1 であっても、I_UGE, DAE エラーを無視して実行を継続できないときは、エラーが通知される。 WEAK_ED は、P.2.2 に記述された、A_UGE と抑止可能エラーにおける例外通知マスクである。 多重 ADE が起きたとき、ハードウェアによって WEAK_ED = 1 がセットされる。
0	UGE_HANDLER	RW	UGE が起きた際に、OS が UGE 処理中かどうかをハードウェアが識別するために使われる。 0 ハードウェアは、OS が UGE 処理中ではない、と認識する。 1 ハードウェアは、OS が UGE 処理中である、と認識する。 UGE_HANDLER は、P.2.2 で説明した、A_UGE と抑止可能エラーにおける例外通知マスクである。 I_UGE,IAE,DAE 発生時に多重エラーが起きたかどうかを識別するために使われる。 ADE が起きると、UGE_HANDLER = 1 にセットされる。RETRY/DONE で 0 にリセットされる。
その他	Reserved	R	常に 0。

P.3 致命的エラーと error_state 遷移エラー

P.3.1 ASI_STCHG_ERROR_INFO

ASI_STCHG_ERROR_INFO には、検出された error_state 遷移エラーの情報が表示される。これらの情報は主に OBP が利用する。

Compatibility Note – SPARC64 VIIIfx では致命的エラーの情報は ASI_STCHG_ERROR_INFO には表示されないので、システムソフトウェアが致命的エラーの詳細情報を知ることはできない。

レジスタ名	ASI_STCHG_ERROR_INFO
ASI	$4C_{16}$
VA	18_{16}
エラー検出	なし
フォーマット	TABLE P-8 参照
リセット後の初期値	ハード POR 時は、全てのフィールドがで初期化される。 それ以外のリセットでは、値は変更されない。
更新ポリシー	エラーを検出した際、関連するフィールドを 1 に設定する。 ビット 0 に 1 を書くと、全ビットが 0 にリセットされる。

TABLE P-8 に ASI_STCHG_ERROR_INFO のフィールドの説明を示す。表中 sticky とは、いったんハードウェアがそのビットに 1 をセットすると、ソフトウェアが 0 を書くまでその値を保持し続けることを意味する。

TABLE P-8 ASI_STCHG_ERROR_INFO のフィールドの説明 *(1 of 2)*

ビット	フィールド名	アクセス	説明
63:34	Reserved	R	常に 0。
33	ECR_WEAK_ED	R	POR またはウォッチドッグリセット時に ASI_ERROR_CONTROL.WEAK_ED がコピーされる。
32	ECR_UGE_HANDLER	R	POR またはウォッチドッグリセット時に ASI_ERROR_CONTROL.UGE_HANDLER がコピーされる。
31:24	Reserved	R	常に 0。
23	EE_MODULE	RW	error_state 遷移エラーにより、CPU モジュールの縮退が要求されたことを示す。sticky.
22	EE_CORE	RW	error_state 遷移エラーにより、コアの縮退が要求されたことを示す。sticky.
21	EE_THREAD	RW	error_state 遷移エラーにより、スレッドの縮退が要求されたことを示す。sticky. ハードウェアがこのビットを 1 にセットすることはない。
20	UGE_MODULE	RW	緊急エラーにより、CPU モジュールの縮退が要求されたことを示す。sticky.
19	UGE_CORE	RW	緊急エラーにより、コアの縮退が要求されたことを示す。sticky.
18	UGE_THREAD	RW	緊急エラーにより、スレッドの縮退が要求されたことを示す。sticky. ハードウェアがこのビットを 1 にセットすることはない。
17	rawUE_MODULE	RW	L2$ で rawUE が検出されたことを示す。sticky.
16	rawUE_CORE	RW	L1$ で rawUE が検出されたことを示す。sticky.
15	EE_DCUCR_MCNTL_ECR	R	以下のレジスタで UE が検出されたことを示す。 (A) ASI_DCUCR (A) ASI_MCNTL (A) ASI_ECR
14	EE_OTHER	R	この表に記載されていない箇所でのエラー発生で 1 がセットされる。SPARC64 VIIIfx では常に 0。
13	EE_TRAP_ADR_UE	R	例外発生時にアドレスを計算しようとした際、TBA, TT あるいはアドレス計算回路で UE が発生しアドレス計算ができなかったことを示す。
12	Reserved	R	常に 0。

TABLE P-8 ASI_STCHG_ERROR_INFO のフィールドの説明 *(2 of 2)*

ビット	フィールド名	アクセス	説明
11	EE_WDT_IN_MAXTL	R	TL = MAXTL でウォッチドッグタイムアウトが発生したことを示す。
10	EE_SECOND_WDT	R	*async_data_error* 発生後、2 度目のウォッチドッグタイムアウトが発生したことを示す。(*async_data_error* が最初のウォッチドッグタイムアウト)。
9	EE_SIR_IN_MAXTL	R	TL = MAXTL で SIR が発生したことを示す。
8	EE_TRAP_IN_MAXTL	R	TL = MAXTL で例外が発生したことを示す。
7:1	Reserved	R	常に 0。
0	clear_all	W	このビットに 1 を書くと、すべてのフィールドが 0 クリアされる。

P.3.2 サスペンド中のスレッドでの error_state 遷移エラー

SPARC64 VIIIfx は SUSPEND 命令により suspended 状態に遷移し、POR, WDR, XDR, *interrupt_vector*, *interrupt_level_n* により復旧する。例外処理に関連する回路でエラーが発生すると、suspend 状態から復旧できなくなる。このような状態に陥ることを防ぐため、以下のレジスタの緊急エラーは error_state 遷移エラーとしてサスペンド中でも報告される。

- ASI_EIDR
- STICK, STICK_CMPR
- TICK, TICK_CMPR

このような場合、UGESR の対応するビットと STCHG_ERROR_INFO.UGE_CORE に 1 がセットされる。

P.4 緊急エラー

この章では緊急エラーの詳細、モニタ方法や命令終了方法について説明する。

P.4.1 緊急エラーステータス (ASI_UGESR)

レジスタ名	ASI_URGENT_ERROR_STATUS
ASI	$4C_{16}$
VA	08_{16}
エラー検出	なし
フォーマット	TABLE P-9 参照
リセット後の初期値	ハード POR 時は、全てのフィールドがで初期化される。 それ以外のリセットでは、値は変更されない。

UGESR は *async_data_error* 発生時のエラー詳細と多重 *async_data_error* 時の 2 番目のエラーの詳細を表示する。

TABLE P-9 に UGESR のフィールドの意味を示す。フィールド名にはプリフィクスがついているが、その意味は以下の通りである。

- IUG_ 命令の緊急エラー
- IAG_ 自律発生の緊急エラー
- IAUG_ I_UGE,A_UGE の両方のエラー

TABLE P-9 ASI_UGESR のフィールドの説明 *(1 of 3)*

ビット	フィールド名	アクセス	説明
bit<22:8> の各ビットは単独 ADE 例外発生の要因を表わす。1 のとき、例外が発生していることを示す。bit<22:16> は CPU 内レジスタでのエラーを表わす。これらエラーの検出条件は Appendix P.8, *“ レジスタで起きたエラーの処理方法 ”* を参照。			
22	IAUG_CRE	R	以下のレジスタの UE。 (IA) ASI_EIDR (IA) ASI_WATCHPOINT (有効な場合) (I) ASI_INTR_R (A) ASI_INTR_DISPATCH_W (書き込み時の UE) (IA) STICK (IA) STICK_CMPR
21	IAUG_TSBCTXT	R	以下のレジスタの UE。 (IA) ASI_DMMU_TSB_BASE (IA) ASI_PRIMARY_CONTEXT (IA) ASI_SECONDARY_CONTEXT (IA) ASI_SHARED_CONTEXT (IA) ASI_IMMU_TSB_BASE
20	IUG_TSBP	R	以下のレジスタの UE。 (I) ASI_DMMU_TAG_TARGET (I) ASI_DMMU_TAG_ACCESS (I) ASI_IMMU_TAG_TARGET (I) ASI_IMMU_TAG_ACCESS
19	IUG_PSTATE	R	以下のレジスタの UE。 PSTATE, PC, NPC, CWP, CANSAVE, CANRESTORE, OTHERWIN, CLEANWIN, PIL, WSTATE
18	IUG_TSTATE	R	以下のレジスタの UE。 TSTATE, TPC, TNPC, TXAR
17	IUG_%F	R	浮動小数点レジスタ (拡張レジスタを含む)、または FPRS, FSR, GSR の UE。
16	IUG_%R	R	整数レジスタ (拡張レジスタを含む)、または Y, CCR, ASI の UE。
14	IUG_WDT	R	一番目のウォッチドッグタイムアウト。単独 ADE 通知時に IUG_WDT = 1 がセットされていると、TPC が指す命令の実行は中断され、結果は不定である。
10	IUG_DTLB	R	• load/store, demap時に DTLB で UE が発生した場合、1 がセットされる。DTLB で以下の事象が起きていることを示す。 • DTLB_DATA_ACCESS, DTLB_TAG_ACCESS で DTLB を読み出そうとした際、DTLB の data または tag で UE が発生した場合。 • DTLBへの書き込み、demap が失敗した場合。TPC はエラーを起こした命令かその次の命令を指す。

TABLE P-9 ASI_UGESR のフィールドの説明 *(2 of 3)*

ビット	フィールド名	アクセス	説明
9	IUG_ITLB	R	• load/store, demap時に ITLB で UE が発生した場合、1 がセットされる。ITLB で以下の事象が起きていることを示す。 • ITLB_DATA_ACCESS, ITLB_TAG_ACCESS で ITLB を読み出そうとした際、ITLB の data または tag で UE が発生した場合。 • ITLBへの書き込み、demap が失敗した場合。TPC はエラーを起こした命令かその次の命令を指す。
8	IUG_COREERR	R	CPU core でエラーが起きたことを示す。命令実行に関するリソースで、ソフトウェアに見えないものでエラーが起きた場合に 1 がセットされる。 ソフトウェアから見えるレジスタでエラーが起き、そのレジスタを読み出す命令が実行されたとき、そのレジスタのエラーを示すビットがセットされるが、IUG_COREERR はセットされる場合もされない場合もある。

TABLE P-9 ASI_UGESR のフィールドの説明 *(3 of 3)*

ビット	フィールド名	アクセス	説明
5:4	INSTEND	R	割込まれた命令の終了方法を示す。単独 ADE でウォッチドッグタイムアウトが検出されていない場合、INSTEND は TPC が指す命令の終了方法を示している。 00_2: Precise 01_2: Retryable but not precise 10_2: *Reserved* 11_2: Not retryable 詳細は P.4.3 を参照。ウォッチドッグリセットが起きた場合は、命令の終了方法は未定義である。
3	PRIV	R	特権モードフラグ。単独 ADE 発生直前の PSTATE.PRIV の値がコピーされる。 PSTATE に UE が発生したため、単独 ADE 発生直前の PSTATE の値が不明の場合は、ASI_UGESR.PRIV が 1 にセットされる。
2	MUGE_DAE	R	DAE により多重 UGE が起きたことを示す。単独 ADE の場合、MUGE_DAE が 0 にセットされる。DAE により多重 ADE が起きると、MUGE_DAE が 1 にセットされる。DAE 以外の要因による多重 ADE の場合は MUGE_DAE は変化しない。
1	MUGE_IAE	R	IAE により多重 UGE が起きたことを示す。単独 ADE の場合、MUGE_IAE が 0 にセットされる。IAE により多重 ADE が起きると、MUGE_IAE が 1 にセットされる。IAE 以外の要因による多重 ADE の場合は MUGE_IAE は変化しない。
0	MUGE_IUGE	R	I_UGE により多重 UGE が起きたことを示す。単独 ADE の場合、MUGE_IUGE が 0 にセットされる。I_UGE により多重 ADE が起きると、MUGE_IUGE が 1 にセットされる。I_UGE 以外の要因による多重 ADE の場合は MUGE_IUGE は変化しない。
Other	*Reserved*	R	常に 0。

P.4.2 async_data_error (ADE) トラップ発生時の処理

単独または多重 ADE が起きる条件は、P.2.2 で定義された通りである。この節では発生した際の処理方法を説明する。

1. ADEが発生する条件は、以下のいずれかである。
   - ASI_ERROR_CONTROL.UGE_HANDLER = 0 で、I_UGEs と / または A_UGEs が検出された場合、単独 ADE 例外が通知される。
   - ASI_ERROR_CONTROL.UGE_HANDLER = 1 で、I_UGEs, IAE, DAE のひとつまたは複数が検出された場合、多重 ADE 例外が通知される。
2. 状態遷移、トラップハンドラのアドレス計算、TL の処理は以下の順で行われる。

a. 状態遷移

TL = MAXTL のときは、ADE 例外の動作を中断し、error_state に遷移する。

CPU が実行状態で TL = MAXTL – 1 のときは、RED_state に遷移する。

b. トラップハンドラのアドレス計算

CPU が実行ステートのときは、TBA, TT, TL からアドレスを計算する。

それ以外、つまり RED_state では、RSTVaddr + $A0_{16}$ がセットされる。

c. TLに 1 を加算する。

3. TSTATE, TPC, TNPC, TXAR の更新。

ADE 例外が通知される直前の PSTATE, PC, NPC, XAR が TSTATE, TPC, TNPC, TXAR にコピーされる。元のレジスタに UE が含まれていても、そのままコピーされる。

4. その他のレジスタ値の更新。

以下の 3 種類のレジスタが更新される。

a. レジスタの自動検証。

ハードウェアにより以下のレジスタが更新される。

レジスタ	更新する条件	更新値
PSTATE	常に	AG = 1, MG = 0, IG = 0, IE = 0, PRIV = 1, AM = 0, PEF = 1, RED = 0 (CPU の状態によっては 1), MM = 00, TLE = 0, CLE = 0.
PC	常に	ADE トラップのアドレス。
nPC	常に	ADE トラップのアドレス +4。
CCR	レジスタのデータが訂正不能エラーを起こしているとき	0.

レジスタ	更新する条件	更新値
FSR, GSR	レジスタのデータが訂正不能エラーを起こしているとき	UE を保持するレジスタに 0 が書かれる。単独 ADE の場合、ASI_UGESR.IUG_%F に 1 がセットされる。
CWP, CANSAVE, CANRESTORE, OTHERWIN, CLEANWIN	レジスタのデータが訂正不能エラーを起こしているとき	UE を保持するレジスタに 0 が書かれる。単独 ADE の場合、ASI_UGESR.IUG_PSTATE に 1 がセットされる。
TICK	レジスタのデータが訂正不能エラーを起こしているとき	NPT = 1, Counter = 0.
TICK_COMPARE	レジスタのデータが訂正不能エラーを起こしているとき	INT_DIS = 1, TICK_CMPR = 0.
XAR	常に	0
XASR	レジスタのデータが訂正不能エラーを起こしているとき	0

書き込まれたレジスタに存在していたエラーは消去される。

上記以外のレジスタおよび TLB エントリに存在するエラーは保持されたままとなる。

b. ASI_UGESR の更新。

ビット	フィールド	単独 ADE トラップでの更新	多重 ADE トラップでの更新
63:6	エラー表示	すべてのビットが更新される。 検出されたすべての I_UGE, A_UGE が同時に表示される。	変更されない。
5:4	INSTEND	TPC が指す命令の終了方法が表示される	変更されない。
2	MUGE_DAE	0 がセットされる。	DAE で多重 ADE 例外が起きた場合、1 がセットされる。それ以外では変更されない。
1	MUGE_IAE	0 がセットされる。	IAE で多重 ADE 例外が起きた場合、1 がセットされる。それ以外では変更されない。
0	MUGE_IUGE	0 がセットされる。	I_UGE で多重 ADE 例外が起きた場合、1 がセットされる。それ以外では変更されない。

c. ASI_ERROR_CONTROL の更新。

単独 ADE 例外時、ASI_ERROR_CONTROL.UGE_HANDLER に 1 がセットされる。RETRY または DONE が実行されるまでは UGE_HANDLER は 1 のままで、ハードウェアにエラー処理中であることを伝える。

多重 ADE が起きると、ASI_ERROR_CONTROL.WEAK_ED が 1 にセットされ、CPU はエラー検出を弱めて動作する。

5. ASI_ERROR_CONTROL.UGE_HANDLER に 0 がセットされる。

   RETRY または DONE 命令が実行されると、完了のときに UGE_HANDLER に 0 がセットされる。

P.4.3 ADE トラップ発生した時の命令の実行状況

SPARC64 VIIIfx では *async_data_error* 通知により終了させられた命令、つまり TPC が指す命令の終了状態は以下の 3 種類のいずれかになる。

- Precise
- Retryable but not precise (JPS1 定義外)
- Not retryable (JPS1 定義外)

単独 ADE 発生時は、TPC が指す命令の終了方法は、ASI_UGESR.INSTEND に表示される。

各終了方法の相違点を TABLE P-10 に示す。

TABLE P-10 *async_data_error* 通知時点の命令の実行状況

	Precise	Retryable But Not Precise	Not Retryable
前回の ADE,IAE,DAE 発生から TPC が指す命令の直前までの命令	完了。 UGE が起きていない命令は、仕様通りに完了する。UGE を起こした命令の出力は不定となり、結果レジスタやメモリに不定値が書きこまれる。		
TPC が指す命令	未実行と同じ結果。	不完全な結果が出力される。 結果の一部だけを書き出したり、結果が壊れている場合 がある。 命令とは無関係なレジスタ・メモリが壊れることはない。 以下の動作は行われない。 • キャッシャブル領域へのストア (メモリ、キャッシュどちらも) • ノンキャッシャブル領域へのストア • 命令の入力レジスタと出力レジスタが同じ場合の、出力レジスタ変更	不完全な結果が出力される。 結果の一部だけを書き出したり、結果が壊れている場合 がある。 命令とは無関係なレジスタ・メモリが壊れることはない。 無効なアドレスに対するストアは行われない (有効なアドレスの場合は実行されるかもしれない)。
TPC の次以降の命令の実行	未実行と同じ結果。	未実行と同じ結果。	未実行と同じ結果。

TABLE P-10 *async_data_error* 通知時点の命令の実行状況

	Precise	Retryable But Not Precise	Not Retryable
例外を通知されたプログラムが、単独 ADE で報告されたエラーでダメージを受けていない場合、実行を継続できるか。	できる。	できる。	できない。

P.4.4 ADE トラップハンドラの処理例

ADE トラップハンドラの例を擬似 C コードで示す。このコードでは以下のエラーの復旧を目的としている。

- CPU 内部の RAM やレジスタ
- 加算機
- CPU 内の一時レジスタやデータバス

```
void
expected_software_handling_of_ADE_trap()
{
    /*
     * ここから #1 地点までは、レジスタウィンドウを制御するレジスタ
     * が壊れているかもしれないので、%r0-%r7 レジスタのみを使う。
     * 可能ならば、単独 ADE のハンドラ全体で %r0-%r7 だけを使うのが
     * 望ましい。
     */

    ASI_SCRATCH_REGp ← %rX;      /* working register 1 */
    ASI_SCRATCH_REGq ← %rY;      /* working register 2 */
    %rX ← ASI_UGESR;

    if ((%rX && 0x07) ≠ 0) {
         /* 多重 ADE 例外が起きている */
         パニックルーチンに入り、ASI_ERROR_CONTROL.WEAK_ED == 1
         で可能な範囲でシステムダンプを採取 ;
    }

    if (%rX.IUG_%R == 1) {
         %rX と %rY 以外の r1-%r63 ← %r0;
         %y ← %r0;
         %tstate.pstate ← %r0;
         /* %tstate.pstate の asi がエラーしているかもしれないので */
}
    else {
```

```
        %rX, %rY, ASI_SCRATCH_REGp, ASI_SCRATCH_REGq を使い
        作業に必要なレジスタを確保するため、%rX, %rY,
        ASI_SCRATCH_REGp, ASI_SCRATCH_REGq を使い %r1-%r7 を
        退避する ;

        /*
         * PSTATE.AG == 1 であるコンテキストでエラーが起き、
         * その処理に復旧するのならば、全 %r レジスタの退避と
         * 復元が必要である。
         */
    }

    if (ASI_UGESR.IUG_PSTATE == 1) {
        %tstate.pstate ← %r0;
        %tpc ← %r0;
        %pil ← %r0;
        %wstate ← %r0;
        レジスタウィンドウ中の全レジスタ ← %r0;
        レジスタウィンドウ制御レジスタ (CWP, CANSAVE,
          CANRESTORE, OTHERWIN, CLEANWIN) には適切な値を設定する ;
    }

    /*
     * Point#1
     * ここまででレジスタウィンドウ制御レジスタのエラー検証が完了して
     * いるので、これ以降 %r0-%r7 以外のウィンドウレジスタを使用可。
     */

    if (ASI_UGESR.IAUG_CRE == 1
          || ASI_UGESR.IAUG_TSBCTXT == 1
          || ASI_UGESR.IUG_TSBP == 1
          || ASI_UGESR.IUG_TSTATE == 1
          || ASI_UGESR.IUG_%F==1) {

        これらのエラーを起こす可能性のあるレジスタをすべて検証する ;
    }

    if (ASI_UGESR.IUG_DTLB == 1) {
        DTLB に対し demap_all を実行する ;
        /*
         * fDTLB のロックされた TTE は demap_all では消去されない。
         */
    }

    if (ASI_UGESR.IUG_ITLB == 1) {
        ITLB に対し demap_all を実行する ;
        /*
```

```
            * fITLB のロックされた TTE は demap_all では消去されない。
            */
        }

        if (ASI_UGESR.bits<22:14> == 0 &&
            ASI_UGESR.INSTEND == 0 || ASI_UGESR.INSTEND == 1) {
            ++ADE_trap_retry_per_unit_of_time;
            if (ADE_trap_retry_per_unit_of_time < threshold)
                RETRY で例外が起きる前のコンテキストに復帰する ;
            else
                ADE 例外発生回数が閾値を越えたので、OS を停止する ;
        } else if (ASI_UGESR.bits<22:18> == 0 &&
                   ASI_UGESR.bits<15:14> == 0 &&
                   ASI_UGESR.PRIV == 0) {
            ++ADE_trap_kill_user_per_unit_of_time;
            if (ADE_trap_kill_user_per_unit_of_time
                  < threshold) {
                ユーザプロセスの実行を停止し、OS は処理を継続する ;
            } else {
                ADE 例外発生によるユーザプロセス中断回数が閾値を越え
                たので、OS を停止する ;
            }
        } else {
            復旧不可能な緊急エラーが起きたので、OS を停止する。
        }
    }
```

P.5 Instruction Access Errors

Appendix F.5, *“Faults and Traps”* (page 178) を参照。

P.6 Data Access Errors

Appendix F.5, *“Faults and Traps”* (page 178) を参照。

P.7 抑止可能エラー

P.7.1 ASI_ASYNC_FAULT_STATUS (ASI_AFSR)

レジスタ名	ASI_ASYNC_FAULT_STATUS (ASI_AFSR)
ASI	$4C_{16}$
VA	00_{16}
エラー検出	なし
フォーマット	TABLE P-11 参照
リセット後の初期値	ハード POR 時は、全てのフィールドがで初期化される。 それ以外のリセットでは、値は変更されない。

ASI_ASYNC_FAULT_STATUS は、抑止可能エラーが発生した際その種類を表示するレジスタである。各ビットは 1 にセットされると、システムソフトウェアによって上書きされるまでその値を保持しつづける。TABLE P-11 に AFSR のフィールドを示す。各フィールドには抑止可能エラーの種類を表わす接頭語がついている。

- DG_ 縮退
- UE_ 訂正不能エラー

TABLE P-11 ASI_ASYNC_FAULT_STATUS のフィールドの説明

ビット	フィールド名	アクセス	説明
12	Reserved		
11	DG_U2$	RW1C	CPU の U2 キャッシュが縮退した場合、1 がセットされる。
10	DG_D1$sTLB	RW1C	I1 キャッシュ、D1 キャッシュ、sITLB, sDTLB が縮退した場合、1 がセットされる。
9	Reserved	R	読み出しには常に 0 が返り、書き込みは無視される。
3	UE_DST_BETO	RW1C	メモリ書き込みに対してバスエラーが返ってきた場合、1 がセットされる。
2	Reserved	R	読み出しには常に 0 が返り、書き込みは無視される。
1	UE_RAW_L2$INSD	RW1C	L2 キャッシュでマークされていない UE が発見された場合、1 がセットされる。
0	UE_RAW_D1$INSD	RW1C	D1 キャッシュでマークされていない UE が発見された場合、1 がセットされる。
その他	Reserved	R	読み出しには常に 0 が返り、書き込みは無視される。

Note – disrupting なバスエラーまたはタイムアウトは、AFSR.UE_DST_BETO, DSFSR.BERR, DSFSR.RTO のいずれか 1 つで報告される。

Note – 参照すると AFSR.UE_DST_BETO がセットされるような領域に対する書き込みの直後に同一アドレスを読み出すと、ストアバッファにあるデータが読み出され、*data_access_error* が発生しないことがある。AFSR.UE_DST_BETO は書き込みの実行後にセットされる。

P.7.2 抑止可能エラーに対するソフトウェア処理

すべての抑止可能エラーは、エラーを記録することが望ましい。この節では各抑止可能エラーに対して期待されるソフトウェアでの処理法を説明する。

- DG_L1$, DG_U2$ — 以下のような CPU の状態が報告される。
  - I1 キャッシュ, D1 キャッシュ, U2 キャッシュ, sITLB, sDTLB でウェイが縮退したので、性能低下の可能性があることを示す。
  - CPU の可用性が低下していることを示す。I1 キャッシュ , D1 キャッシュ , U2 キャッシュ , sITLB, sDTLB が 1 ウェイで動作している状態で、最後のウェイでもエラーが発生した場合は、error_state 遷移エラーとなる。

    必要ならば、ソフトウェアはエラーを起こしている CPU の使用を止める。
- UE_DST_BETO — このエラーが起きるのは以下のどちらかの場合である。
  - DTLB に間違った TTE が存在する。
  - 物理アドレスアクセス ASI で、無効な領域にアクセスした。

  どちらの場合も、原因はシステムソフトウェアのバグである。エラー情報を元にシステムソフトウェアを修正する。
- UE_RAW_L2$INSD, and UE_RAW_D1$INSD — このエラーの処理は、
  - 可能ならば、UE を含むキャッシュラインのエラーを消去する。これによりキャッシュラインに載っていたデータは失われることに注意。
- *ECC_error* 例外が通知されたが、ASI_AFSR にエラーが記録されていない場合 — *ECC_error* 例外を無視する。

  詳細は “ *エラー検出時の動作* ” (page 263) を参照。

P.8 レジスタで起きたエラーの処理方法

この節では、以下のレジスタで起きたエラーの処理方法について説明する。

- 特権、非特権レジスタ
- ASR レジスタ
- ASI レジスタ

P.8.1 特権・非特権レジスタの処理方法

TABLE P-12 内で使われる用語の定義は以下の通りである。

列	用語	意味
エラー検出条件	InstrAccess	命令実行によりレジスタがアクセスされたときにエラーが検出される。
エラー訂正	W	レジスタ全体への書き込みでエラーが訂正される。
	ADE trap	ハードウェアが、*async_data_error* 例外によるトラップ処理中にレジスタ全体への書き込みを行い、エラーが訂正される。

TABLE P-12 に、特権、非特権レジスタのエラー処理方法を示す。PC, nPC, PSTATE, CWP, ASI, XAR で緊急エラーが起きた場合、*async_data_error* トラップハンドラに処理が移った時点で、エラーも含めてそれぞれ TPC, TNPC, TSTATE, TXAR にコピーされていることに注意。

TABLE P-12 特権・非特権レジスタのエラーの扱い *(1 of 2)*

レジスタ	RW	エラー保護	エラー検出条件	エラー種類	エラー訂正
%rn[1]	RW	パリティ	InstrAccess	IUG_%R	W
%fn[1]	RW	パリティ	InstrAccess	IUG_%F	W
PC	R	パリティ	Always	IUG_PSTATE	ADE trap
nPC	R	パリティ	Always	IUG_PSTATE	ADE trap
PSTATE	RW	パリティ	Always	IUG_PSTATE	ADE trap, W
TBA	RW	パリティ	PSTATE.RED = 0	error_state	W(OBP が行う)
PIL	RW	パリティ	PSTATE.IE = 1 InstrAccess	IUG_PSTATE	W
CWP, CANSAVE, CANRESTORE, OTHERWIN, CLEANWIN	RW	パリティ	Always	IUG_PSTATE	ADE trap, W
TT	RW	なし	—	—	—
TL	RW	パリティ	PSTATE.RED = 0	error_state	W(OBP が行う)
TPC	RW	パリティ	InstrAccess	IUG_TSTATE	W
TNPC	RW	パリティ	InstrAccess	IUG_TSTATE	W
TSTATE	RW	パリティ	InstrAccess	IUG_TSTATE	W
WSTATE	RW	パリティ	Always	IUG_PSTATE	ADE trap, W
VER	R	なし	—	—	—

TABLE P-12 特権・非特権レジスタのエラーの扱い *(2 of 2)*

レジスタ	RW	エラー保護	エラー検出条件	エラー種類	エラー訂正
FSR	RW	パリティ	Always	IUG_%F	ADE trap, W
Y	RW	パリティ	InstrAccess	IUG_%R	W
CCR	RW	パリティ	Always	IUG_%R	ADE trap, W
ASI	RW	パリティ	Always	IUG_%R	ADE trap, W
TICK	RW	パリティ	AUG Always[2]	IUG_COREERR	ADE trap[3] , W
FPRS	RW	パリティ	Always	IUG_%F	ADE trap, W

1.拡張レジスタを含む。

2.サスペンド中のスレッドには error_state 遷移エラーとして通知される。

3.訂正のために書く値は 0x8000_0000_0000_0000

P.8.2 ASR レジスタの処理方法

TABLE P-13 内で使われる用語の定義は以下の通りである。

列	用語	意味
エラー検出条件	AUG always	エラーは、ASI_ERROR_CONTROL.UGE_HANDLER = 0 かつ ASI_ERROR_CONTROL.WEAK_ED = 0 のときに検出される。
	InstrAccess	命令実行によりレジスタがアクセスされたときに検出される。
エラー種類	(I)AUG_*xxx*	自律エラーが起き、ASI_UGESR.IAUG_*xxx* = 1 となった。
	I(A)UG_*xxx*	命令エラーが起き、ASI_UGESR.IAUG_*xxx* = 1 となった。
エラー訂正	W	レジスタ全体への書き込みでエラーが訂正される。
	ADE trap	ハードウェアが、*async_data_error* 例外によるトラップ処理中にレジスタ全体への書き込みを行い、エラーが訂正される。

TABLE P-13 に、ASR レジスタのエラー処理方法を示す。

TABLE P-13 ASR レジスタのエラーの扱い *(1 of 2)*

ASR Number	レジスタ	RW	エラー保護	エラー検出条件	エラー種類	エラー訂正
16	PCR	RW	なし	—	—	—
17	PIC	RW	なし	—	—	—
18	DCR	R	なし	—	—	—
19	GSR	RW	パリティ	Always	IUG_%F	ADE trap, W
20	SET_SOFTINT	W	なし	—	—	—

TABLE P-13 ASR レジスタのエラーの扱い *(2 of 2)*

ASR Number	レジスタ	RW	エラー保護	エラー検出条件	エラー種類	エラー訂正
21	CLEAR_SOFTINT	W	なし	—	—	—
22	SOFTINT	RW	なし	—	—	—
23	TICK_COMPARE	RW	パリティ	AUG always[1]	IUG_COREERR	ADE trap, W
24	STICK	RW	パリティ	AUG always[1]	(I)AUG_CRE	W
				InstrAccess	I(A)UG_CRE	W
25	STICK_COMPARE	RW	パリティ	AUG always[1]	(I)AUG_CRE	W
				InstrAccess	I(A)UG_CRE	W
29	XAR	RW	パリティ	Always	IUG_COREERR	ADE trap, W
30	XASR	RW	パリティ	Always	IUG_COREERR	ADE trap, W
29	TXAR	RW	パリティ	InstrAccess	IUG_TSTATE	W

1.サスペンド中のスレッドには error_state 遷移エラーとして通知される。

STICK Behavior on Error

STICK でエラーが起きた場合、TABLE P-13 の検出条件とは関係なくカウントアップは停止する。

P.8.3 ASI レジスタの処理方法

TABLE P-14 内で使われる用語の定義は以下の通りである。

列	用語	意味
エラー保護	パリティ	パリティで保護されている。
	Triple	レジスタが三重化されている。
	ECC	ECC で保護されている。
	Gecc	生成された ECC で保護されている。
	なし	保護なし

列	用語	意味
エラー検出条件	Always	エラーは常に検出される。
	AUG always	エラーは、ASI_ERROR_CONTROL.UGE_HANDLER = 0 かつ ASI_ERROR_CONTROL.WEAK_ED = 0 のときに検出される。
	LDXA	エラーはレジスタの読み出し時に検出される。
	ITLB write	エラーは、ITLB 書き込みまたは demap による更新時に検出される。
	DTLB write	エラーは、DTLB 書き込みまたは demap による更新時に検出される。
	Used by TLB	エラーは、TLB 検索の際に参照されたときに検出される。
	Enabled	エラーは、その機能が有効なときに検出される。
	intr_receive	エラーは、割り込みパケットを受信したときに検出される。受信パケットの中に UE があると、vector_interrupt 例外が通知されるが ASI_INTR_RECEIVE.BUSY は 0 にセットされる。ASI_INTR_RECEIVE.BUSY に 0 を書き込むと、新たなパケットが受信可能となる。
エラー種類	error_state	error_state 遷移エラー。
	(I)AUG_*xxxx*	自律エラーが発生し、エラー情報が ASI_UGESR.IAUG_*xxxx* = 1 に表示される。
	I(A)UG_*xxxx*	命令エラーが発生し、エラー情報が ASI_UGESR.IAUG_*xxxx* = 1 に表示される。
	Other	ASI_UGESR の、エラーに対応するビットに 1 がセットされる。
エラー訂正	RED trap	RED_state トラップの発生時に値が更新されエラーが訂正される。
	W	ASI レジスタへの書き込みでエラーが訂正される。
	W_other_I	• 以下のレジスタ全てを更新することでエラーが訂正される。 • ASI_IMMU_TAG_ACCESS • ASI_UGESR.IAUG_TSBCTXT = 1 で単独 ADE のときは ASI_IMMU_TSB_BASE, ASI_PRIMARY_CONTEXT, ASI_SECONDARY_CONTEXT, ASI_SHARED_CONTEXT
	W_other_D	• 以下のレジスタ全てを更新することでエラーが訂正される。 • ASI_DMMU_TAG_ACCESS • ASI_UGESR.IAUG_TSBCTXT = 1 で単独 ADE のときは ASI_DMMU_TSB_BASE, ASI_PRIMARY_CONTEXT, ASI_SECONDARY_CONTEXT, ASI_SHARED_CONTEXT
	Interrupt receive	割り込みパケットの受信でエラーが訂正される。

TABLE P-14 に、ASI レジスタのエラー処理方法を示す。

TABLE P-14 ASI レジスタのエラーの扱い *(1 of 2)*

ASI	VA	レジスタ	RW	エラー保護	エラー検出条件	エラー種類	エラー訂正
45_{16}	00_{16}	DCU_CONTROL	RW	パリティ	Always	error_state	RED trap
	08_{16}	MEMORY_CONTROL	RW	パリティ	Always	error_state	RED trap
48_{16}	00_{16}	INTR_DISPATCH_STATUS	R	パリティ	LDXA またはレジスタ更新	I(A)UG_CRE (UE)	None
49_{16}	00_{16}	INTR_RECEIVE	RW	パリティ	LDXA	I(A)UG_CRE (UE)	None
$4A_{16}$	—	SYS_CONFIG	R	なし	—	—	—
$4B_{16}$	00_{16}	STICK_CNTL	RW	Triple	Always	—	Always
$4C_{16}$	00_{16}	ASYNC_FAULT_STATUS	RW1C	なし	—	—	—
$4C_{16}$	08_{16}	URGENT_ERROR_STATUS	R	なし	—	—	—
$4C_{16}$	10_{16}	ERROR_CONTROL	RW	パリティ	Always	error_state	RED trap
$4C_{16}$	18_{16}	STCHG_ERROR_INFO	R, W1AC	なし	—	—	—
$4F_{16}$	00_{16}–38_{16}	SCRATCH_REGs	RW	パリティ	LDXA	IUG_COREERR	W
50_{16}	00_{16}	IMMU_TAG_TARGET	R	パリティ	LDXA	IUG_TSBP	W_other_I
50_{16}	18_{16}	IMMU_SFSR	RW	なし	—	—	—
50_{16}	28_{16}	IMMU_TSB_BASE	RW	パリティ	LDXA	I(A)UG_TSBCTXT	W
50_{16}	30_{16}	IMMU_TAG_ACCESS	RW	パリティ	LDXA	IUG_TSBP	W (W_other_I)
50_{16}	60_{16}	IMMU_TAG_ACCESS_EXT	RW	パリティ	LDXA	IUG_TSBP	W
50_{16}	78_{16}	IMMU_SFPAR	RW	パリティ	LDXA	I(A)UG_CRE	W
53_{16}	—	SERIAL_ID	R	なし	—	—	—
54_{16}	—	ITLB_DATA_IN	W	パリティ	ITLB 書き込み	IUG_ITLB	DemapAll
55_{16}	—	ITLB_DATA_ACCESS	RW	パリティ	LDXA	IUG_ITLB	DemapAll
					ITLB 書き込み	IUG_ITLB	DemapAll
56_{16}	—	ITLB_TAG_READ	R	パリティ	LDXA	IUG_ITLB	DemapAll
57_{16}	—	IMMU_DEMAP	W	パリティ	ITLB 書き込み	IUG_ITLB	DemapAll
58_{16}	00_{16}	DMMU_TAG_TARGET	R	パリティ	LDXA	IUG_TSBP	W_other_D
58_{16}	08_{16}	PRIMARY_CONTEXT	RW	パリティ	LDXA	I(A)UG_TSBCTXT	W
					Used by TLB AUG always	I(A)UG_TSBCTXT	W
						(I)AUG_TSBCTXT	W
58_{16}	10_{16}	SECONDARY_CONTEXT	RW	パリティ	= P_CONTEXT	IAUG_TSBCTXT	W
58_{16}	18_{16}	DMMU_SFSR	RW	なし	—	—	—
58_{16}	20_{16}	DMMU_SFAR	RW	パリティ	LDXA	IAUG_CRE	W
58_{16}	28_{16}	DMMU_TSB_BASE	RW	パリティ	LDXA	I(A)UG_TSBCTXT	W
58_{16}	30_{16}	DMMU_TAG_ACCESS	RW	パリティ	LDXA	IUG_TSBP	W (W_other_D)

TABLE P-14 ASI レジスタのエラーの扱い *(2 of 2)*

ASI	VA	レジスタ	RW	エラー保護	エラー検出条件	エラー種類	エラー訂正
58_{16}	38_{16}	DMMU_WATCHPOINT	RW	パリティ	Enabled	(I)AUG_CRE	W
					LDXA	I(A)UG_CRE	W
58_{16}	60_{16}	DMMU_TAG_ACCCESS_EXT	RW	パリティ	LDXA	IUG_TSBP	W
58_{16}	68_{16}	SHARED_CONTEXT	RW	パリティ	= P_CONTEXT	(I)AUG_TSBCTXT	W
58_{16}	78_{16}	DMMU_SFPAR	RW	パリティ	LDXA	I(A)UG_CRE	W
$5C_{16}$	—	DTLB_DATA_IN	W	パリティ	DTLB 書き込み	IUG_DTLB	DemapAll
$5D_{16}$	—	DTLB_DATA_ACCESS	RW	パリティ	LDXA	IUG_DTLB	DemapAll
					DTLB 書き込み	IUG_DTLB	DemapAll
$5E_{16}$	—	DTLB_TAG_READ	R	パリティ	LDXA	IUG_DTLB	DemapAll
$5F_{16}$	—	DMMU_DEMAP	W	パリティ	DTLB 書き込み	IUG_DTLB	DemapAll
60_{16}	—	IIU_INST_TRAP	RW	パリティ	LDXA	No match at error	W
67_{16}	—	FLUSH_L1I	W	なし	—	—	—
$6D_{16}$	00_{16}- 58_{16}	BARRIER_INIT	RW	パリティ	Always(割り当てられているとき) または LDXA	Fatal Error	—
$6E_{16}$	00_{16}	EIDR	RW	パリティ	Always[1]	IAUG_CRE	W
$6F_{16}$	00_{16}- 58_{16}	BARRIER_ASSIGN	RW	パリティ	Always(割り当てられているとき)	Fatal Error	—
74_{16}	addr	CACHE_INV	W	なし	—	—	—
77_{16}	40_{16}–50_{16}	INTR_DATA0:2_W	W	Gecc	None	—	W
77_{16}	70_{16}	INTR_DISPATCH_W	W	Gecc	store	(I)AUG_CRE	W
$7F_{16}$	40_{16}–50_{16}	INTR_DATA0:2_R	R	ECC	LDXA	IAUG_CRE	Interrupt Receive
					intr_receive	BUSY = 0	
$E7_{16}$	00_{16}	SCCR	RW	パリティ	Always	IUG_COREERR	W
FE_{16}	00_{16}- 58_{16}	LBSY, BST	RW	パリティ	Always(割り当てられているとき)	Fatal Error	—

1.サスペンドステートでは error_state で報告される。

P.9 キャッシュで起きたエラーの処理方法

この節では、キャッシュタグ、キャッシュデータのエラー対処について説明する。

P.9.1 キャッシュタグで起きたエラーの処理方法

I1 キャッシュタグ、D1 キャッシュタグのエラー

U2 キャッシュは、I1 キャッシュタグと D1 キャッシュタグのコピーを保持している。I1 キャッシュタグとそのコピー、D1 キャッシュタグとそのコピーはいずれもパリティで保護されている。

D1 キャッシュタグまたは D1 キャッシュタグのコピーでエラーが検出されると、ハードウェアは自動的にエラーが起きていない方から起きている方へコピーを行う。この処理でエラーが修復されたときは、プログラムの実行には影響を与えない。

同様に、I1 キャッシュタグまたは I1 キャッシュタグのコピーでエラーが検出されると、ハードウェアは自動的にエラーが起きていない方から起きている方へコピーを行う。この処理でエラーが修復されたときは、プログラムの実行には影響を与えない。

上記の動作でエラーが修復されないときは、タグコピーが繰り返される。固定故障によるエラーの場合、最終的にウォッチドッグタイムアウトかフェイタルエラーが検出される。

U2 キャッシュタグのエラー

U2 キャッシュタグは ECC により保護され、1 ビットエラーの訂正と 2 ビットエラーの検出ができる。

U2 キャッシュタグで訂正可能なエラーが発見された場合、ハードウェアは自動的に正しいデータを上書きしデータを修復する。エラーはシステムソフトウェアには報告されない。

U2 キャッシュタグで訂正不可能なエラーが発見された場合、致命的エラーとして通知され、CPU はフェイタルエラーステートに遷移する。

P.9.2 I1 キャッシュデータで起きたエラーの処理方法

I1 キャッシュデータは、8 バイト単位でパリティ保護されている。

命令フェッチ中、I1 キャッシュデータにパリティエラーが発見されると、ハードウェアは以下の順序で処理を行う。

1. パリティエラーを含むキャッシュラインを U2 キャッシュから読み出す。

   U2 キャッシュから読み出されたデータは、UE を含まないデータか、マーク済 UE を含むデータのどちらかである。なぜなら、エラーマークは U2 キャッシュから外に出て行く (outgoing) データにのみ行われるからである。

2. U2 キャッシュから読み出された 8 バイトごとに、

a. その 8 バイトに UE がなければ、I1 キャッシュに保存し、必要なら命令フェッチ部に供給する。

   I1 キャッシュ再コピーの際にエラー訂正が行われても、直接システムソフトウェアに報告は上がらない。

b. その 8 バイトにマーク済 UE があるときは、I1 キャッシュデータの当該 8 バイトに対するパリティビットを、パリティエラーを示すように設定する。必要なら命令フェッチ部にデータを供給する。

3. フェッチ部でのエラーのある命令の扱いは、

   パリティエラーしている命令をフェッチしたが、その命令は実行されず、ソフトウェアから見える状態が変わらないとき、その命令は単に捨てられる。

   フェッチした命令が実行され完了したときは、*instruction_access_error* 例外が通知され、ASI_ISFSR にマーク済 UE を検出したことと、その ERROR_MARK_ID が表示される。

P.9.3 D1 キャッシュデータで起きたエラーの処理方法

D1 キャッシュデータは 8 バイト単位で ECC により保護され、1 ビットエラーの訂正と 2 ビットエラーの検出ができる。

D1 キャッシュデータの訂正可能エラー

D1 キャッシュデータで訂正可能エラーが発見された場合、データはハードウェアにより自動的に訂正される。訂正可能エラーはシステムソフトウェアには報告しない。

D1 キャッシュデータのマーク済 UE

D1 キャッシュから U2 キャッシュへのライトバック中に、D1 キャッシュデータでマーク済 UE が発見されると、D1 キャッシュデータと ECC は変更なしで U2 キャッシュに書き戻される。つまり、D1 キャッシュのマーク済 UE は U2 キャッシュに引き継がれる。この書き戻しはシステムソフトウェアには報告されない。

D1 キャッシュのマーク済 UE に対するロード、ストア命令 (8 バイトストアを除く) には、*data_access_error* 例外が通知される。この例外通知は precise で、ASI_DSFSR にはマークの ERROR_MARK_ID が表示される。

D1 キャッシュ上のマークしていない UE の U2 への書き戻し

D1 キャッシュから U2 への書き戻し時にマークしていない UE が発見されると、エラーを含む 8 バイトにマーキングが行われる。このときの ERROR_MARK_ID には ASI_EIDR の値が使われる。U2 キャッシュに書き戻されるのは、訂正済みかマーク済のデータのみである。

訂正が行われると ASI_AFSR.UE_RAW_D1$INSD に 1 がセットされる。

D1 キャッシュ上のマークしていない UE の命令による読み出し

D1 キャッシュ上のマークしていない UE の命令に対する、メモリアクセス命令の読み出しに対し、ハードウェアは以下の順に処理を行う。

1. 一旦 D1 キャッシュから U2 に書き戻し、再度 U2 キャッシュから読み込む。

   D1 キャッシュのデータは、U2 のデータと同じであるか更新されているかに係わらず書き戻される。この書き戻しの際にエラーマーキングが行われる。このときの ERROR_MARK_ID には ASI_EIDR の値が使われる。D1 キャッシュラインは U2 キャッシュから読み出され、ASI_AFSR.UE_RAW_D1$INSD に 1 がセットされる。

2. 通常は 1 でマークされていないエラーはマーク済になるが、この操作の途中に同一の 8 バイトに新たな UE が起きることがある。この場合 1 の処理が繰り返される。この動作は D1 キャッシュウェイが縮退するまで行われる。

3. ここまでの操作で、ハードウェアは D1 キャッシュデータのマークされていない UE をマーク済にする。この後メモリアクセス命令はマーク済の UE をアクセスするが、このときの動作は *“D1 キャッシュデータのマーク済 UE”* (page 294) を参照。

P.9.4 U2 キャッシュデータで起きたエラーの処理方法

U2 キャッシュデータは 8 バイト単位で ECC により保護され、1 ビットエラーの訂正と 2 ビットエラーの検出ができる。

U2 キャッシュデータの訂正可能エラー

メモリから U2 キャッシュへ読み込まれたデータに訂正可能エラーが発見されると、ハードウェアが自動的にエラーを訂正する。例外は通知されない。

U2 キャッシュから I1 キャッシュ、D1 キャッシュへの読み込みや、メモリや他キャッシュへの書き出し時に、転送データや元データに訂正可能エラーが発見された場合、ハードウェアが自動的にエラーを訂正する。このエラーはシステムソフトウェアには報告されない。

U2 キャッシュデータのマーク済 UE

U2 キャッシュデータ上にある、マーク済 UE の 8 バイトデータは、訂正済みデータと同等に扱われる。マーク済 UE が U2 キャッシュデータ上で発見されても、エラー報告はあがらない。

メモリから U2 キャッシュへ読み込まれたデータにマーク済 UE が発見されても、変更されずそのまま U2 キャッシュに書き込まれる。

D1 キャッシュから U2 キャッシュへの書き戻しの際にマーク済 UE が発見されても、変更されずそのまま U2 キャッシュに書き込まれる。なお、マークしていない UE を書き戻すことはない。詳細は *“D1 キャッシュ上のマークしていない UE の U2 への書き戻し”* (page 295) を参照。

U2 キャッシュから I1 キャッシュ、D1 キャッシュへの読み込みや、メモリや他キャッシュへの書き出し時に、転送データや元データにマーク済 UE が発見されても、変更されずそのまま転送される。

U2 キャッシュデータのマークされていない UE

メモリから U2 キャッシュへ読み込まれたデータにマークされていない UE が発見されると、エラーを含む 8 バイト単位でエラーマーキングが行われる。このとき ERROR_MARK_ID は 0 でマーキングされる。当該 8 バイトと ECC がマーク済データに置き換えられ、U2 キャッシュに書き込まれる。例外は通知されない。

U2 キャッシュからの読み出し (I1 キャッシュや D1 キャッシュへの読み込み、メモリや他キャッシュへの書き出し）時に、マークされていない UE が発見されると、エラーを含む 8 バイト単位でエラーマーキングが行われる。このとき ERROR_MARK_ID には ASI_EIDR が使われる。ASI_AFSR.UE_RAW_L2$INSD に 1 がセットされる。

P.9.5 I1,D1,U2 キャッシュの自動ウェイ縮退

I1,D1,U2 キャッシュでエラーが頻発すると、ハードウェアはキャッシュデータの一貫性を保ちながら、ウェイを縮退させる。

ウェイ縮退の条件

ハードウェアは、各キャッシュのウェイ毎に、以下のエラー発生回数の合計を計測している。

- I1 キャッシュの各ウェイについて
  - I1 キャッシュタグと I1 キャッシュタグコピーのパリティエラー
  - I1 キャッシュデータのパリティエラー
- D1 キャッシュの各ウェイについて
  - D1 キャッシュタグと D1 キャッシュタグコピーのパリティエラー

- D1 キャッシュデータの訂正可能エラー
- D1 キャッシュデータのマークされていない UE

- U2 キャッシュの各ウェイについて
  - U2 キャッシュタグの訂正可能エラーと UE
  - U2 キャッシュデータの訂正可能エラー
  - U2 キャッシュデータのマークされていない UE

あるキャッシュウェイのカウンタが、一定時間内に定められた閾値を越えると、ハードウェアはそのキャッシュウェイを縮退させる。その手順は以下の通り。

I1 キャッシュのウェイ縮退

I1 キャッシュのウェイ *w* を縮退させる手順は、

1. すでに 1 ウェイ縮退している場合は、エラー箇所のみを無効化する。
2. それ以外では、
   - ウェイ *w* の全データが無効化され、これ以降ウェイ *w* への読み込みは行われない。
   - `ASI_AFSR.DG_D1$STLB` に 1 がセットされ、抑止可能エラーが通知される。

D1 キャッシュのウェイ縮退

D1 キャッシュのウェイ *w* を縮退させる手順は、

1. すでに 1 ウェイ縮退している場合は、エラー箇所のみ U2 にライトバックし、エラー箇所のみを無効化する。
2. それ以外では、
   - ウェイ *w* の全データが無効化され、これ以降ウェイ *w* への読み込みは行われない。U2 キャッシュデータから変更されているデータについては、U2 キャッシュへライトバックされる。
   - `ASI_AFSR.DG_D1$STLB` に 1 がセットされ、抑止可能エラーが通知される。

U2 キャッシュのウェイ縮退

U2 キャッシュの縮退は、`DCUCR.WEAK_SPCA` = 0 のときはすぐに行われるが、`DCUCR.WEAK_SPCA` = 1 のときはペンディングされ、`DCUCR.WEAK_SPCA` = 0 になったときに開始される。

U2 キャッシュのウェイ *w* を縮退させる手順は、

1. すでに他のウェイは縮退して、1 ウェイしか残っていない場合は、
   - U2 キャッシュのウェイ *w* の全データが一斉に無効化されるが、ウェイ *w* は引き続き使用される。システム全体のデータ一意性を保つため、U2 キャッシュ上のデータは破棄される。

- ASI_AFSR.DG_U2 に 1 がセットされ、抑止可能エラーが通知される。エラー通知は、キャッシュ構成の変化によらず行われる。

2. それ以外では、
   - システム全体のデータ一意性を保つため、ウェイ *w* を含む全ウェイの全データが無効化される。
   - これ以降ウェイ *w* は使用されない。
   - ASI_AFSR.DG_U2 に 1 がセットされ、抑止可能エラーが通知される。

P.10 TLB で発生したエラー

この節では、TLB エントリのエラー処理方法と sTLB のウェイ縮退について説明する。

P.10.1 TLB エラーの処理

SPARC64 VIIIfx の各 TLB のエラー保護を TABLE P-15 に示す。

TABLE P-15 TLB エントリのエラー保護と検出方法

TLB 種類	フィールド	エラー保護	検出できるエラー
sITLB, sDTLB	タグ	パリティ	パリティエラー (訂正不可能)
	データ	パリティ	パリティエラー (訂正不可能)
fITLB, fDTLB	ロックビット	三重化	なし。値は多数決で決定される。
	ロックビット以外	パリティ	パリティエラー (二重化しており訂正可能)
	データ	パリティ	パリティエラー (二重化しており訂正可能)

TLB のエラーは、メモリアクセス時のアドレス変換と ASI レジスタ経由で直接アクセスする際に発見される。

ASI レジスタ経由でアクセスする際に発見されたエラー

DTLB に ASI_DTLB_DATA_ACCESS または ASI_DTLB_TAG_ACCESS でエラーが発見されると、ASI_UGESR.IUG_DTLB に 1 をセットし、命令緊急エラーを通知する。

ITLB に ASI_ITLB_DATA_ACCESS または ASI_ITLB_TAG_ACCESS でエラーが発見されると、ASI_UGESR.IUG_ITLB に 1 をセットし、命令緊急エラーを通知する。

アドレス変換の際に sTLB で発見されたエラー

アドレス変換の際に sTLB でエラーが発見されると、そのエントリを無効にする。システムソフトウェアには報告されない。

アドレス変換の際に fTLB で発見されたエラー

fTLB はタグもデータも二重化されているので、アドレス変換の際に fTLB でパリティエラーが発見されると、正しいほうからコピーして自動訂正する。システムソフトウェアには報告されない。両方のエントリでパリティエラーが発見されると、致命的エラーとなる。

F.APPENDIX **Q**

Performance Instrumentation

この章では、SPARC64 VIIIfx の性能計測カウンタ (Performance Counter:PA) について説明する。

Q.1 PA 概要

PA カウンタの定義については、*“Performance Control Register (PCR) (ASR 16)” (page 26)* および *“Performance Instrumentation Counter (PIC) Register (ASR 17)”* (page 28) を参照。

Q.1.1 サンプル擬似コード

カウンタのセットとクリア

PIC は読み書き可能なレジスタである。0 を書くとカウンタがクリアされ、それ以外の値を書けばそれがカウンタにセットされる。以下の擬似コードは、すべての PIC カウンタをクリアする例である。

```
/* SU/SL の設定を変更せずに PIC をクリアする */
pic_init = 0x0;
pcr = rd_pcr();
pcr.ulro = 0x1;         /* 書き込み時に su/sl を変更しない */
pcr.ovf = 0x0;          /* オーバーフロービットをクリアする */
pcr.ut = 0x0;
pcr.st = 0x0;           /* 正確な計測のためにカウンタを停止する */
for (i=0; i<=pcr.nc; i++) {
   /* 書き込む PIC を選択する */
   pcr.sc = i;
   wr_pcr(pcr);
```

```
    wr_pic(pic_init);/* PIC[i] をクリア */
}
```

計測項目の選択と計測開始

計測項目は PCR.SC と PCR.SU/PCR.SL で選択する。以下のコードは項目の選択と計測開始のサンプルである (特権アクセス可能な場合)

```
pcr.ut = 0x0;          /* ユーザイベント計測を停止 */
pcr.st = 0x0;          /* システムイベント計測も停止 */
pcr.ulro = 0x0;        /* SU/SL を変更可能にする */
pcr.ovro = 0x1;        /* オーバーフロービットは変更しない */
/* 計測せずにイベントを選択する */
for(i=0; i<=pcr.nc; i++) {
   pcr.sc = i;
   pcr.sl = select an event;
   pcr.su = select an event;
   wr_pcr(pcr);
}
/* 計測開始 */
pcr.ut = 0x1;
pcr.st = 0x1;
pcr.ulro = 0x1;        /* SU/SL ビットを変更しない */
/* 必要ならオーバーフロービットをここでクリアする */
wr_pcr(pcr);
```

計測停止と読みだし

以下のコードは計測停止と読みだしのサンプルである (特権アクセス可能な場合)

```
pcr.ut = 0x0;          /* ユーザイベント計測を停止 */
pcr.st = 0x0;          /* システムイベント計測も停止 */
pcr.ulro = 0x1;        /* SU/SL をリードオンリーにする */
pcr.ovro = 0x1;        /* オーバーフロービットは変更しない */
for(i=0; i<=pcr.nc; i++) {
   pcr.sc = i;
   wr_pcr(pcr);
   pic = rd_pic();
   picl[i] = pic.picl;
   picu[i] = pic.picu;
}
```

Q.2 PA イベントの説明

PA カウンタは以下のグループに大別される。

1. 命令種類、トラップ種類毎の統計情報
2. MMU と L1 キャッシュ関連のイベント計測
3. L2 キャッシュ関連のイベント計測
4. バストランザクションの計測

PA カウンタで計測できるイベントには公開、準公開の 2 種類がある。

公開イベントは、正確に動作することが検証されており、SPARC64 VIIIfx の将来の版でも互換性を保証する[1] イベントである。

準公開イベントは、主としてハードウェアのデバッグ用のイベントである。

- 準公開イベント必ずしも検証されているとは限らないので、このドキュメントで書かれている通りに動かないかもしれない。
- 定義は予告なしに変更されることがある。SPARC64 VIIIfx の将来の版での互換性は保証しない。

SPARC64 VIIIfx で定義されている全 PA イベントを TABLE Q-1 に示す。網掛けのイベントは準公開イベントである。各イベントの計測内容は次節以降で解説する。特に断りない限り、投機命令による事象も PA イベントの計測対象となる。

1. デザイン変更により該当する機能がなくなった場合はこの限りではない。

TABLE Q-1 PA カウンタのイベント番号と PIC

Encoding	Counter							
	picu0	**picl0**	**picu1**	**picl1**	**picu2**	**picl2**	**picu3**	**picl3**
0000000	cycle_counts							
0000001	instruction_counts							
0000010	instruction_flow_counts	*Reserved*			instruction_flow_counts	*Reserved*		xma_inst
0000011	iwr_empty	*Reserved*			iwr_empty	*Reserved*		
0000100	*Reserved*							
0000101	op_stv_wait							
0000110	effective_instruction_counts							
0000111	SIMD_load_store_instructions	SIMD_floating_instructions	SIMD_fma_instructions	sxar1_instructions	sxar2_instructions	unpack_sxar1	unpack_sxar2	*Reserved*
0001000	load_store_instructions							
0001001	branch_instructions							
0001010	floating_instructions							
0001011	fma_instructions							
0001100	prefetch_instructions							
0001101	*Reserved*	ex_load_instructions	ex_store_instructions	fl_load_instructions	fl_store_instructions	SIMD_fl_load_instructions	SIMD_fl_store_instructions	*Reserved*
0001110	*Reserved*							
0001111	*Reserved*							
0010000	*Reserved*							
0010001	*Reserved*							
0010010	rs1	flush_rs	*Reserved*					
0010011	1iid_use	2iid_use	3iid_use	4iid_use	*Reserved*	*sync_intlk*	*regwin_intlk*	*Reserved*
0010100	*Reserved*							
0010101	*Reserved*	toq_rsbr_phantom	*Reserved*	flush_rs	*Reserved*		rs1	*Reserved*
0010110	trap_all	trap_int_vector	trap_int_level	trap_spill	trap_fill	trap_trap_inst	trap_IMMU_miss	trap_DMMU_miss
0010111	*Reserved*	trap_SIMD_load_across_pages	*Reserved*					
0011000	*Reserved*							
0011001	*Reserved*							

TABLE Q-1 PA カウンタのイベント番号と PIC *(Continued)*

Encoding	Counter							
	picu0	**picl0**	**picu1**	**picl1**	**picu2**	**picl2**	**picu3**	**picl3**
0011010	*Reserved*		single_sxar_commit	*Reserved*				suspend_cycle
0011011	rsf_pmmi	*Reserved*		0iid_use	flush_rs	*Reserved*		decall_intlk
0011100	*Reserved*							
0011101	op_stv_wait_pfp_busy_ex	op_stv_wait_sxmiss	op_stv_wait_sxmiss_ex	op_stv_wait_nc_pend	cse_window_empty_sp_full	op_stv_wait_pfp_busy	*Reserved*	
0011110	cse_window_empty	eu_comp_wait	branch_comp_wait	0endop	op_stv_wait_ex	fl_comp_wait	1endop	2endop
0011111	inh_cmit_gpr_2write	*Reserved*			3endop	*Reserved*	sleep_cycle	op_stv_wait_swpf
0100000	uITLB_miss2	uDTLB_miss2	uITLB_miss	uDTLB_miss	L1I_miss	L1D_miss	L1I_wait_all	L1D_wait_all
0100001	*Reserved*							
0100010	*Reserved*							
0100011	L1I_thrashing	L1D_thrashing	*Reserved*					
0100100	swpf_success_all	swpf_fail_all	*Reserved*		swpf_lbs_hit	*Reserved*		
0100101	*Reserved*							
0100110	*Reserved*							
0100111	*Reserved*							
0110000	*Reserved*		L2_miss_dm	L2_miss_pf	L2_read_dm	L2_read_pf	L2_wb_dm	L2_wb_pf
0110001	bi_count	*Reserved*		cpd_count	cpu_mem_read_count	cpu_mem_write_count	IO_mem_read_count	IO_mem_write_count
0110010	L2_miss_wait_dm_bank0	L2_miss_wait_pf_bank0	L2_miss_count_dm_bank0	L2_miss_count_pf_bank0	L2_miss_wait_dm_bank1	L2_miss_wait_pf_bank1	L2_miss_count_dm_bank1	L2_miss_count_pf_bank1
0110011	L2_miss_count_dm_bank2	L2_miss_count_pf_bank2	L2_miss_wait_dm_bank2	L2_miss_wait_pf_bank2	L2_miss_count_dm_bank3	L2_miss_count_pf_bank3	L2_miss_wait_dm_bank3	L2_miss_wait_pf_bank3
0110100	lost_pf_pfp_full	lost_pf_by_abort	IO_pst_count	*Reserved*				
0110101	*Reserved*							
0110110	*Reserved*							
0111111	*Disabled (No PIC is counted up)*							
1111111	*Disabled (No PIC is counted up)*							

Q.2.1 命令種類、トラップ種類毎の統計情報

公開 PA 項目

1 *cycle_counts*

CPU サイクル数を計測する。TICK レジスタと同じサイクルを計測するが、*cycle_counts* は PCR.UT, PCR.ST の設定によりユーザ、システム個別に計測することができる。

2 *instruction_counts* (Non-Speculative)

完了した命令数を計測する。この命令数には SXAR1, SXAR2 が含まれる。

SPARC64 VIIIfx は 1 サイクルで最大 4 命令ずつ実行完了することができるが、SXAR1, SXAR2 は通常、この完了命令数には含まれない。このため、*instruction_counts* / *cycle_counts* は 4 より大きな値になることがある。

3 *effective_instruction_counts* (Non-Speculative)

完了した実効命令数を計測する。この命令数には SXAR1, SXAR2 は含まれない。

cycle_counts と組み合わせることで、サイクルあたりの実行命令数 IPC が導出される。

IPC = *effective_instruction_counts* / *cycle_counts*

ユーザ、システムそれぞれの effective_*Instruction_counts* と *cycle_counts* からは、ユーザ、システム毎の IPC が得られる。

4 *load_store_instructions* (Non-Speculative)

完了した整数、浮動小数点数のロード、ストア命令数を計測する。アトミック命令もカウントされる。

SIMD load, SIMD store は計測対象ではなく、別のイベントで計測される。

5 *branch_instructions* (Non-Speculative)

完了した分岐命令数を計測する。CALL, JMPL, RETURN 命令もカウントされる。

6 *floating_instructions* (Non-Speculative)

完了した 1 演算の浮動小数点演算命令数を計測する。計測対象は FPop1 (TABLE E-5), FPop2 (TABLE E-6) のほか、IMPDEP1 の opf<8:4> = 16_{16} または 17_{16} の命令である。SIMD 演算は計測対象ではなく、別のイベントでカウントされる。

Compatibility Note – SPARC64 VII までの CPU では、このイベントの計測対象は `FPop1`, `FPop2` のみだった。

7 *fma_instructions* (Non-Speculative)

完了した浮動小数点積和演算命令の命令数を計測する。計測対象命令は `FM{ADD,SUB}{s,d}`, `FNM{ADD,SUB}{s,d}`, `FTRIMADDd` である。SIMD 演算は計測対象ではなく、別のイベントでカウントされる。

Compatibility Note – SPARC64 VII までの CPU では、このイベントは *impdep2_instructions* という名前で、計測対象は浮動小数点積和演算のみだった。

計測対象命令の命令当たりの演算数は 2 なので、演算数を求める際は 2 倍する。

8 *prefetch_instructions* (Non-Speculative)

完了したプリフェッチ命令数を計測する。

9 *SIMD_load_store_instructions* (Non-Speculative)

完了した浮動小数点数の SIMD ロード、SIMD ストア命令数を計測する。

10 *SIMD_floating_instructions* (Non-Speculative)

計測対象は *floating_instructions* と同じだが、SIMD 実行し完了した命令数を計測する。

計測対象命令の SIMD 実行時の命令当たりの演算数は 2 なので、演算数を求める際は 2 倍する。

11 *SIMD_fma_instructions* (Non-Speculative)

計測対象は *fma_instructions* と同じだが、SIMD 実行し完了した命令数を計測する。

計測対象命令の SIMD 実行時の命令当たりの演算数は 4 なので、演算数を求める際は 4 倍する。

12 *sxar1_instructions* (Non-Speculative)

完了した `SXAR1` 命令数を計測する。

13 *sxar2_instructions* (Non-Speculative)

完了した `SXAR2` 命令数を計測する。

14 *trap_all* (Non-Speculative)

すべてのトラップの回数を計測する。これは他のトラップイベントの合計に等しい。

15 *trap_int_vector* (Non-Speculative)

interrupt_vector_trap が通知された回数を計測する。

16 *trap_int_level* (Non-Speculative)

interrupt_level_n が通知された回数を計測する。

17 *trap_spill* (Non-Speculative)

spill_n_normal, spill_n_other が通知された回数を計測する。

18 *trap_fill* (Non-Speculative)

fill_n_normal, fill_n_other が通知された回数を計測する。

19 *trap_trap_inst* (Non-Speculative)

trap_instruction が通知された回数を計測する。

20 *trap_IMMU_miss* (Non-Speculative)

fast_instruction_access_MMU_miss が通知された回数を計測する。

21 *trap_DMMU_miss* (Non-Speculative)

fast_data_instruction_access_MMU_miss が通知された回数を計測する。

22 *trap_SIMD_load_across_pages* (Non-Speculative)

SIMD_load_across_pages が通知された回数を計測する。

準公開 PA 項目

23 *xma_inst* (Non-Speculative)

完了した `FPMADDX`, `FPMADDXHI` 命令数を計測する。

24 *unpack_sxar1* (Non-Speculative)

完了した `SXAR1` のうち、パックされなかった命令数を計測する。

25 *unpack_sxar2* (Non-Speculative)

完了した SXAR2 のうち、パックされなかった命令数を計測する。

26 *instruction_flow_counts* (Non-Speculative)

完了した命令数を計測する。SPARC64 VIIIfx では、いくつかの命令は内部的に複数の命令に分けて実行する。*instruction_flow_counts* が計測するのはこの内部的な命令数である。この命令数にはパックされた SXAR1, SXAR2 は含まれない。

27 *ex_load_instructions* (Non-Speculative)

完了した整数ロード命令の命令数を計測する。対象となる命令は *LD(S,U)B{A}, LD(S,U)H{A}, LD(S,U)W{A}, LDD{A}, LDX{A}* である。

28 *ex_store_instructions* (Non-Speculative)

完了した整数ストア命令およびアトミック命令の命令数を計測する。対象となる命令は *STB{A}, STH{A}, STW{A}, STD{A}, STX{A}, LDSTUB{A}, SWAP{A}, CAS{X}A* である。

29 *fl_load_instructions* (Non-Speculative)

完了した浮動小数点ロード命令の命令数を計測する。対象となる命令は *LDF{A}, LDDF{A}, LD{X}FSR* である。

SIMD ロード命令および *LDQF{A}* はこのイベントでは計測されない。

30 *fl_store_instructions* (Non-Speculative)

完了した浮動小数点ストア命令の命令数を計測する。対象となる命令は *STF{A}, STDF{A}, STFR, STDFR, ST{X}FSR* である。

SIMD ストア命令および *STQF{A}* はこのイベントでは計測されない。

31 *SIMD_fl_load_instructions* (Non-Speculative)

完了した SIMD 浮動小数点ロード命令の命令数を計測する。対象となる命令は SIMD で実行された *LDF{A}, LDDF{A}* である。

32 *SIMD_fl_store_instructions* (Non-Speculative)

完了した SIMD 浮動小数点ストア命令の命令数を計測する。対象となる命令は SIMD で実行された *STF{A}, STDF{A}, STFR, STDFR* である。

33 *iwr_empty*

IWR (Issue Word Register) が空であるサイクル数を計測する。IWR はデコード時に命令を保持する 4 エントリのバッファで、IWR が空という状態は、命令フェッチがキャッシュミスによりできないときなどに起こりうる。

34 *rs1* (Non-Speculative)

以下の要因で命令実行が停止したサイクル数を計測する。

- トラップ、インタラプト
- 特権レジスタの更新
- メモリオーダリング保証
- ハードウェア内部での再実行 (RAS 起因)

35 *flush_rs* (Non-Speculative)

分岐予測ミスによるパイプラインフラッシュが起きた回数を計測する。
SPARC64 VIIIfx は投機実行を行うので、分岐予測ミスにより実際には実行されない命令をパイプラインに投入することがある。このような命令は、分岐方向がはっきりし間違ったパスであることが確定した時点でキャンセルされる。パイプラインフラッシュはこのときに起きる。

misprediction rate = *flush_rs* / *branch_instructions*

36 *0iid_use*

ある CPU サイクルの命令発行数が 0 である回数を計測する。SPARC64 VIIIfx は最大 4 命令を発行することができるが、発行命令数が 0 のとき、*0iid_use* が 1 加算される。

SPARC64 VIIIfx では、いくつかの命令は内部的に複数の命令に分けて実行するが、*0iid_use* この個々の命令に対して計測される。`SXAR` も計測する。

37 *1iid_use*

ある CPU サイクルの命令発行数が 1 である回数を計測する。計測条件は 0iid_use を参照。

38 *2iid_use*

ある CPU サイクルの命令発行数が 2 である回数を計測する。計測条件は 0iid_use を参照。

39 *3iid_use*

ある CPU サイクルの命令発行数が 3 である回数を計測する。計測条件は 0iid_use を参照。

40 *4iid_use*

ある CPU サイクルの命令発行数が 4 である回数を計測する。計測条件は 0iid_use を参照。

41 *sync_intlk*

パイプラインの sync により命令発行ができなかったサイクル数を計測する。

42 *regwin_intlk*

レジスタウィンドウの切り替えにより命令発行ができなかったサイクル数を計測する。

43 *decall_intlk*

命令デコード時のインタロックにより命令発行ができなかったサイクル数を計測する。*decall_intlk* は *sync_intlk* と *regwin_intlk* を含むが、動的に変化する要因 (リザベーションステーションの枯渇など) によるインタロックは計測対象ではない。

44 *rsf_pmmi* (Non-Speculative)

単精度の浮動小数点演算と倍精度の浮動小数点演算が混ざったため、命令発行ができなかったサイクル数を計測する。

45 *toq_rsbr_phantom*

分岐すると予測した命令が実際には分岐命令ではなかった回数を計測する。SPARC64 VIIIfx の分岐予測機構は命令デコードより前に分岐予測を行うので、分岐命令かどうかとは無関係に分岐するかしないかを予測する。*toq_rsbr_phantom* は、このような間違った命令に対する予測のうち、分岐すると予測したものの回数を計測する。

46 *op_stv_wait* (Non-Speculative)

実行中の命令 の中で一番古い命令が、メモリアクセスによるデータ待ちのため、完了命令数が 0 であるサイクル数を計測する。*op_stv_wait* はストア命令によるデータ待ちは計測しない (アトミック命令は計測する)。

op_stv_wait はすべてのキャッシュミスレイテンシを計測しているのではないことに注意。先行の未完了命令がある場合、*op_stv_wait* は計測されない。

47 *op_stv_wait_nc_pend* (Non-Speculative)

ノンキャッシャブルアクセスによる *op_stv_wait* を計測する。

48 *op_stv_wait_ex* (Non-Speculative)

整数のメモリアクセス命令による *op_stv_wait* を計測する。L1 キャッシュか L2 キャッシュかは区別しない。

49 *op_stv_wait_sxmiss* (Non-Speculative)

L2 キャッシュミスによる *op_stv_wait* を計測する。整数ロードか浮動小数点ロードかは区別しない。

50 *op_stv_wait_sxmiss_ex* (Non-Speculative)

整数ロードの L2 キャッシュミスによる *op_stv_wait* を計測する。

51 *op_stv_wait_pfp_busy* (Non-Speculative)

プリフェッチ用の空きポートがないため、メモリアクセス命令が実行できないことによる *op_stv_wait* を計測する。

52 *op_stv_wait_pfp_busy_ex* (Non-Speculative)

プリフェッチ用の空きポートがないため、整数メモリアクセス命令が実行できないことによる *op_stv_wait* を計測する。

53 *op_stv_wait_swpf* (Non-Speculative)

プリフェッチ用の空きポートがないため、プリフェッチ命令が実行できないことによる *op_stv_wait* を計測する。

54 *cse_window_empty_sp_full* (Non-Speculative)

CSE が空でストアポートがフルのため、完了命令数が 0 であるサイクル数を計測する。

55 *cse_window_empty* (Non-Speculative)

CSE が空のため、完了命令数が 0 であるサイクル数を計測する。

56 *branch_comp_wait* (Non-Speculative)

実行中の命令 の中で一番古い命令が、実行中の分岐命令で、完了命令数が 0 であるサイクル数を計測する。*branch_comp_wait* の計測は *eu_comp_wait* より優先度が低い。

57 *eu_comp_wait* (Non-Speculative)

実行中の命令 の中で一番古い命令が、整数または浮動小数点数の演算実行中で、完了命令数が 0 であるサイクル数を計測する。*eu_comp_wait* の計測は *branch_comp_wait* より優先度が高い。

58 *fl_comp_wait* (Non-Speculative)

実行中の命令 の中で一番古い命令が、浮動小数点数の演算実行中で、完了命令数が 0 であるサイクル数を計測する。

59 *0endop* (Non-Speculative)

完了命令数が 0 であるサイクル数を計測する。SXAR 命令だけがコミットした場合も *0endop* で計測される。

60 *1endop* (Non-Speculative)

完了命令数が 1 であるサイクル数を計測する。

61 *2endop* (Non-Speculative)

完了命令数が 2 であるサイクル数を計測する。

62 *3endop* (Non-Speculative)

完了命令数が 3 であるサイクル数を計測する。

63 *inh_cmit_gpr_2write* (Non-Speculative)

整数レジスタを 2 個更新しているため、4 命令完了できなかったサイクル数を計測する。

64 *suspend_cycle* (Non-Speculative)

SUSPEND 命令、SLEEP 命令により命令制御部が停止しているサイクル数。

65 *sleep_cycle* (Non-Speculative)

SLEEP 命令により命令制御部が停止しているサイクル数。

66 *single_sxar_commit* (Non-Speculative)

パックされなかった SXAR 命令だけがコミットしたサイクル数を計測する。*0endop* にも計測されている。

Q.2.2 MMUと L1 キャッシュ関連のイベント計測

公開 PA 項目

1 *uITLB_miss*

命令フェッチの uTLB ミスが起きた回数を計測する。

2 *uDTLB_miss*

データアクセスの uTLB ミスが起きた回数を計測する。

Note – メイン TLB のミス回数は *trap_IMMU_miss*, *trap_DMMU_miss* で計測できる。

---

3 *L1I_miss*

L1 命令キャッシュのミス回数を計測する。

4 *L1D_miss*

L1 データキャッシュのミス回数を計測する。

5 *L1I_wait_all*

L1 命令キャッシュミスの処理にかかる時間 (ミスレイテンシ) を計測する。SPARC64 VIIIfx の L1 命令キャッシュはノンブロッキングキャッシュなので、同時に複数のキャッシュミスを処理することができるが、*L1I_wait_all* が計測するのは一つのミスレイテンシだけである。

6 *L1D_wait_all*

L1 データキャッシュミスの処理にかかる時間 (ミスレイテンシ) を計測する。SPARC64 VIIIfx の L1 データキャッシュはノンブロッキングキャッシュなので、同時に複数のキャッシュミスを処理することができるが、*L1D_wait_all* が計測するのは一つのミスレイテンシだけである。

準公開 PA 項目

7 *uITLB_miss2*

命令フェッチの uTLB ミスが起き、fITLB から読み出した回数を計測する。

8 *uDTLB_miss2*

データアクセスの uTLB ミスが起き、fDTLB から読み出した回数を計測する。

9 *swpf_success_all*

SU でロストせず SX に送られた PREFETCH 命令の数。

10 *swpf_fail_all*

SU でロストした PREFETCH 命令の数。

11 *swpf_lbs_hit*

L1 キャッシュにヒットした PREFETCH 命令の数。

SU に送られた PREFETCH 命令の総数

12 *L1I_thrashing*

リードポートが獲得されてから解放されるまでの間に、L2 の読み出し要求が 2 回発行された回数。命令フェッチが L1I キャッシュをミスし、そのデータが L1I キャッシュに書き込まれたが、読み出される前にキャッシュから追い出された場合、このカウンタで計測される。

13 *L1D_thrashing*

リードポートが獲得されてから解放されるまでの間に、L2 の読み出し要求が 2 回発行された回数。メモリアクセス命令が L1D キャッシュをミスし、そのデータが L1D キャッシュに書き込まれたが、読み出される前にキャッシュから追い出された場合、このカウンタで計測される。

Q.2.3 L2 キャッシュ関連のイベント計測

L2 キャッシュ関連のイベントには、CPU コアの動作により起きる事象と、CPU チップ外部からの要求に起きる事象がある。前者はコア毎に個別に計測され、後者はすべてのコアで計測される。

大部分の L2 キャッシュ関連のイベントには、dm (デマンド), pf (プリフェッチ) の区別がある。しかしこれは、それぞれロード・ストア・アトミック命令とプリフェッチ命令にに対応しているわけではない。その理由は、

- L1 キャッシュにデータを読みこむための資源が不足していてロード・ストア命令が実行できないとき、先行して L2 キャッシュにデータを読みこんでおき、資源が確保できたら元のロード・ストア命令を行う。この先行読みこみがプリフェッチとして処理されるため。
- ハードウェアプリフェッチ機構がプリフェッチを生成するため。
- L1 キャッシュに対するプリフェッチ命令はデマンド要求として処理しようとするため。

したがって、L2 キャッシュ関連の PA イベントのデマンド (dm)、プリフェッチ (pf) で計測される事象は以下のようになる。

- L2 キャッシュをデマンド (dm) でアクセスするのは、命令フェッチ、ロード・ストア・アトミック命令、L1 プリフェッチ命令のうち、メモリアクセスに必要な資源が獲得できたもの。
- L2 キャッシュをプリフェッチ (pf) でアクセスするのは、命令フェッチ、ロード・ストア命令、L1 プリフェッチ命令のうち、メモリアクセスに必要な資源が獲得できたなかったもの、およびハードウェアプリフェッチ。

公開 PA 項目

1 *L2_read_dm*

デマンド要求の L2 キャッシュ参照回数を計測する。ブロックロード、ブロックストア命令は一命令で 8 回の参照回数として計測される。

CPU チップ外部からのキャッシュ参照要求は計測されない。

2 *L2_read_pf*

プリフェッチ要求の L2 キャッシュ参照回数を計測する。ブロックロード、ブロックストア命令は一命令で 8 回の参照回数として計測される。

3 *L2_miss_dm*

デマンド要求の L2 キャッシュミス回数を計測する。

このカウンタは、*L2_miss_count_dm_bank{0,1,2,3}* の合計を計測する。

4 *L2_miss_pf*

プリフェッチ要求の L2 キャッシュミス回数を計測する。

このカウンタは、*L2_miss_count_pf_bank{0,1,2,3}* の合計を計測する。

5 *L2_miss_count_dm_bank{0,1,2,3}*

デマンド要求が L2 キャッシュをミスした回数を計測する。L2 キャッシュのバンク毎に個別に計測する。

Note – あるアドレスに対して先行してプリフェッチが出され、L2 キャッシュをミスしてメモリアクセス要求を発行し、そのデータが戻ってくる前にデマンド要求がきた場合、後続のデマンド要求はミス回数には計上されない。

---

6 *L2_miss_count_pf_bank{0,1,2,3}*

プリフェッチ要求が L2 キャッシュの各バンクをミスした回数をバンク毎に計測する。

7 *L2_miss_wait_dm_bank{0,1,2,3}*

デマンド要求が L2 キャッシュをミスしたときの処理にかかる時間 (ミスレイテンシ) を、バンク毎に計測する。計測は個々のメモリアクセス要求について行われる。

Note – あるアドレスに対して先行してプリフェッチが出され、L2 キャッシュをミスしてメモリアクセス要求を発行し、そのデータが戻ってくる前にデマンド要求がきた場合、それ以降データが戻ってくるまでのサイクル数は、*L2_miss_wait_dm_bank{0,1,2,3}* で計測される。

---

8 *L2_miss_wait_pf_bank{0,1,2,3}*

プリフェッチ要求が L2 キャッシュをミスしたときの処理にかかる時間 (ミスレイテンシ) を、バンク毎に計測する。計測は個々のメモリアクセス要求について行われる。

*L2_miss_wait_** を *L2_miss_count_** で割ることで、L2 キャッシュミスレイテンシを求めることができる。

Note – *L2_miss_count_** と *L2_miss_wait_** から L2 キャッシュミスレイテンシを求めることができる。プリフェッチ要求を処理中に後続のデマンド要求が到着した場合の PA 測定の特性から、プリフェッチとデマンドを分けて求めると、デマンド要求のレイテンシが大きく見えることがある。

---

9 *L2_wb_dm*

デマンド要求の L2 キャッシュミスによるライトバック回数を計測する。

10 *L2_wb_pf*

プリフェッチ要求の L2 キャッシュミスによるライトバック回数を計測する。

準公開 PA 項目

11 *lost_pf_pfp_full*

PF ポートフルによりプリフェッチ要求がロストした回数を計測する。

12 *lost_pf_by_abort*

SX パイプラインアボートによりプリフェッチ要求がロストした回数を計測する。

Q.2.4 バストランザクションの計測

公開 PA 項目

1 *cpu_mem_read_count*

CPU からの要求により、メモリ読み出し要求を発行した回数を計測する。

2 *cpu_mem_write_count*

CPU からの要求により、メモリ書き込み要求を発行した回数を計測する。

3 *IO_mem_read_count*

I/O からの要求により、メモリ読み出し要求を発行した回数を計測する。

4 *IO_mem_write_count*

I/O からの要求により、メモリ書き込み要求を発行した回数を計測する。

この項目で計測するのは、ICC-FST のみ。ICC-PST は *IO_pst_count* で計測できる。

5 *bi_count*

CPU チップが外部からキャッシュ無効化要求を受けた回数を計測する。計測対象は、キャッシュの状態によらず無効化する要求である。

この項目は、すべてのコアで同じ値が計測される。

6 *cpi_count*

CPU チップが外部からキャッシュ書き出しと無効化要求を受けた回数を計測する。計測対象は、キャッシュ上のデータが更新されていればメモリに書き出して無効化し、キャッシュ上のデータとメモリの内容が一致していれば無効化する要求である。

この項目は、すべてのコアで同じ値が計測される。

Implementation Note – SPARC64 VIIIfx ではこの PA イベントは存在しない。項目は互換性のために残してある。

---

7 *cpb_count*

CPU チップが外部からキャッシュ書き出しと要求を受けた回数を計測する。計測対象は、キャッシュ上の更新されているデータをメモリに書き出す要求である。

この項目は、すべてのコアで同じ値が計測される。

Implementation Note – SPARC64 VIIIfx ではこの PA イベントは存在しない。項目は互換性のために残してある。

---

8 *cpd_count*

CPU チップが外部からキャッシュ内容の読み出し要求を受けた回数を計測する。計測対象は、DMA 読み出しなど、キャッシュ上の更新されているデータを読み出し、メモリに書き出さない要求である。

この項目は、すべてのコアで同じ値が計測される。

準公開 PA 項目

9 *IO_pst_count*

I/O からの要求により、メモリ書き込み要求 (ICC-PST) を発行した回数を計測する。

Q.3 サイクルアカウンティング

サイクルアカウンティングとは、性能ボトルネック要因分析の手法である。ある命令列を実行するためにかかった総時間 (CPU サイクル数) を、CPU の動作状態 (命令実行中である、メモリアクセス待ちである、演算完了待ちである、など）で分類し、命令列を実行した時、CPU 内のどの部分にボトルネックがあるかを把握することで、性能分析や改善を行うことができる。SPARC64 VIIIfx は豊富な PA イベントを備えており、CPU の動作状態に関する詳細な情報が取得できるので、詳細なサイクルアカウンティングが作成でき、効率的なボトルネック分析や性能向上チューニングに役立てることができる。

SPARC64 VIIIfx は複数の演算器を持ち、アウトオブオーダで実行する CPU である。このため、ある命令がメモリからのデータ待ち状態であっても、別の命令は浮動小数点乗算の実行中で、さらにまた別の命令は分岐方向の確定待ちにあるなど、実行中と実行待ちの命令が複数入り乱れた状態にある。このような状態で個々の命令の待ち要因を分析するのは意味がないので、インオーダで行われる命令完了に着目し、計測期

間の総サイクル数を、1 サイクルあたりの命令完了数で大きく分類し、命令完了がないサイクルについてはさらにその要因により細分類したものをサイクルアカウンティングと呼んでいる。

SPARC64 VIIIfx は 1 サイクルに最大 4 命令まで完了することができるので、4 命令完了しているサイクル数の割合が多いほど実行効率が高いことになる。命令完了できなかったサイクルは、性能に与える悪影響が極めて大きいので、詳細に分析する必要がある。主な原因として、

- メモリアクセスのデータが帰ってくるのを待っている
- 演算の完了を待っている
- 命令フェッチが間に合わず、パイプラインに命令が供給されていない

が考えられる。サイクルアカウンティングに使える PA イベントの一覧とその計算方法を TABLE Q-2 に示す。

また、FIGURE Q-1 はメモリアクセス待ちの PA イベント群 *op_stv_wait_** 間の関係を図示したものである。なお、表や図中で † がついた PA イベントは、計算によって求められる架空の PA イベントである。

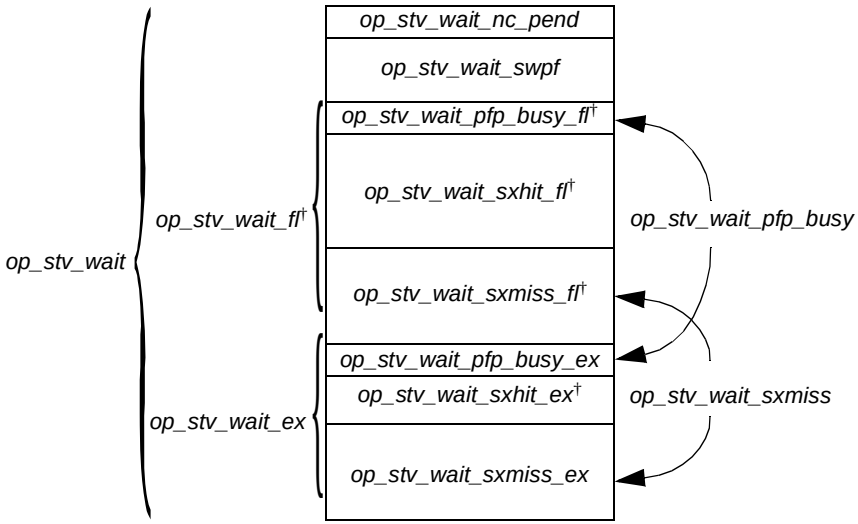


FIGURE Q-1 *op_stv_wait* の内訳

TABLE Q-2 サイクルアカウンティングに有用な PA イベント

CPU サイクルあたりのコミット数	総 CPU サイクル数	備考
4	*cycle_counts* *- 3endop - 2endop* *- 1endop - 0endop*	N/A (最大コミット数に達している)
3	*3endop*	次の命令をコミットできなかった要因のひとつは *inh_cmit_gpr_2write* で計測される。
2	*2endop*	
1	*1endop*	
0	命令実行待ち : *eu_comp_wait* *+ branch_comp_wait*	*eu_comp_wait* *= ex_comp_wait*[†]*+ fl_comp_wait*
	命令フェッチ待ち : *cse_window_empy*	*cse_window_empty* *= cse_window_empty_sp_full* *+ sleep_state* + misc.[†]
	L1D キャッシュアクセス待ち : *op_stv_wait* - L2 キャッシュミス (下記参照)	
	L2 キャッシュミス : *op_stv_wait_sxmiss* *+ op_stv_wait_nc_pend*	
	その他 : *0endop* *- op_stv_wait* *- cse_window_empy* *- eu_comp_wait* *- branch_comp_wait* *-(instruction_flow_counts* *- instruction_counts*)	

F.APPENDIX **R**

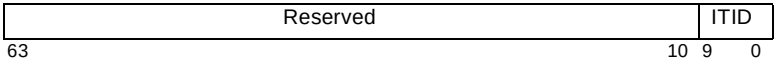
System Programmer’s Model

この章では、これまでの章で触れられていない CPU 内のコンポーネントにあるレジスタについて定義する。

なお、サービスプロセッサから CPU を制御するための仕様は本仕様書の範囲外である。

R.1 System Config Register

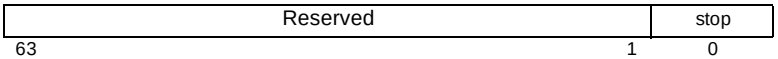
レジスタ名	ASI_SYS_CONFIG
ASI	$4A_{16}$
VA	—
アクセス種別	Supervisor read/write(書き込みは無視される)



ビット	フィールド名	アクセス	説明
63:10	TBD	TBD	TBD
9:0	ITID	R	スレッドの ITID (Interrupt Target ID)。

R.2 STICK Control Register

レジスタ名	ASI_STICK_CNTL
ASI	$4B_{16}$
VA	00_{16}
アクセス種別	Supervisor read/write



ビット	フィールド名	アクセス	説明
63:1	—		
0	stop	RW	stop に 1 を設定すると STICK のカウントアップは停止し、stop に 0 設定すると STICK のカウントアップが再開する。

STICK_CNTL は、STICK のカウントアップを有効 / 無効を制御するレジスタである。STICK_CNTL は全コアで共有されており、どのコアから操作しても、全コアの STICK に対して同時に作用する。

STICK.stop = 1 の間は STICK のカウントアップが停止している。その影響は以下の通り。

- STICK_CMPR をセットしていてもタイマ割り込みがかからない。

  ただし、STICK_CNTL.stop = 1 の状態で
  - STICK_CMPR.INT_DIS = 0
  - STICK_CMPR.STICK_CMPR = STICK.counter

  と設定した場合は、SOFTINT.SM = 1 がセットされる。このとき PSTATE.IE = 1 かつ PIL < 14 ならば、レベル 14 の割り込みがかかる。
- SLEEP 命令を実行してスリープ中のコアは、命令実行状態に復帰しない。

複数のコアがほぼ同時に STICK_CNTL を読み書きした場合、それらの要求はひとつずつ処理される。処理順序はハードウェアの実装に依存する。

Programming Note – STICK_CNTL の操作はあるひとつのコアから行うこと。

STICK_CNTL の書き込み後の STICK の書き込み / 読み出しは、STICK_CNTL の書き込み処理の完了を待って FLUSH 命令を実行してから行うこと。完了までの時間は未定義。完了確認は、STICK_CNTL への書き込みを行ったコアから STICK_CNTL を読み出すことで行える。STICK_CNTL の書き込みが完了する前に STICK を書き込み / 読み出した場合、STICK の値は保証されない。

F.APPENDIX **S**

Summary of Specification Differences

この章では、SPARC V9, SPARC JPS1, SPARC64 VII 仕様と SPARC64 VIIIfx 仕様の相違点を総括する。この章は読者の便宜を図る目的で書かれており、論理仕様の定義ではない。正確な定義は各項目を参照されたい。

TABLE S-1 に、SPARC V9, SPARC JPS1, SPARC64 VII 各仕様と SPARC64 VIIIfx 仕様との相違点をまとめた。前方互換性の欄は、SPARC V9, SPARC JPS1, SPARC64 VII 各仕様に準拠したソフトウェアが、SPARC64 VIIIfx 仕様の CPU で動作するかどうかを示している[1]。

1. SPARC V9, SPARC JPS1, SPARC64 VII 各仕様で *reserved* の部分を使っているソフトウェアは動作非互換になるが、TABLE S-1 は *reserved* については言及しない。

TABLE S-1 仕様差分サマリ *(1 of 4)*

項目	仕様				前方互換性			ページ
	V9	JPS1	SPARC64 VII	SPARC64 VIIIfx	V9	JPS1	SPARC64 VII	
アーキテクチャ								
コア、スレッド	未定義		4 コア、コア当たり 2 スレッド	8 コア、コア当たり 1 スレッド			非互換	10
整数レジスタ	160 本			192 本				20
浮動小数点レジスタ	単精度 32 本 倍精度 32 本			単精度 32 本 倍精度 256 本 倍精度レジスタを単精度演算でも利用できる。				20
ASR	未定義	%pcr, %pic, %dcr, %gsr, %softint, %tick_cmpr, %sys_tick, %sys_tick_cmpr		%pcr, %pic, %dcr, %gsr, %softint, %tick_cmpr, %sys_tick, %sys_tick_cmpr, %xar, %xasr, %txar				26
物理アドレス	未定義	43 ビット以上	47 ビット	41 ビット		非互換		176, 182
RSTVaddr	未定義	実装依存	PA = 7fff f000 0000_{16}	PA = 1ff f000 0000_{16}			非互換	43
キャッシュ	未定義		• L1: 64KB/2way(I), 64KB/2way(D), 64byte line • L2: 6MB/12way, 256byte line/4sublines	• L1: 32KB/2way(I), 32KB/2way(D), 128byte line セクタあり • L2: 6MB/12way, 128byte line インデクスハッシュ・セクタ機能あり			非互換 (インデクスハッシュ)	12, 12, 230, 231
SXflush	未定義		あり	なし			非互換	—
TLB	未定義		32(fTLB)+2048/4way(sTLB), I,D とも。 fTLB は sTLB の victim cache でもある。	16(fTLB)+256/4way(sITLB), 512/4way(sDTLB) victim cache の機能なし。 エラー注入機能削除。			非互換	173, 192
ページサイズ	未定義	8KB, 64KB, 512KB, 4MB	8KB, 64KB, 512KB, 4MB, 32MB, 256MB	8KB, 64KB, 512KB, 4MB, 32MB, 256MB, 2GB				175
TSB	未定義	TLBmiss 時、ハードウェアが TSBptr を計算する。		ハードウェアのサポートなし。 以下の ASI は削除 : • I/D TSB Primary Extension • D TSB Secondary Extension • I/D TSB Nuclues Extension • I/D TSB 8KB ptr • I/D TSB 64KB ptr • D TSB Direct ptr TSBbase は残してあるが split フィールドを削除。		非互換		177, 184, 193

TABLE S-1 仕様差分サマリ *(2 of 4)*

項目	仕様				前方互換性			ページ
	V9	JPS1	SPARC64 VII	SPARC64 VIIIfx	V9	JPS1	SPARC64 VII	
バリア	未定義		BPU 2, BB 12/BPU, BST 24bit/BPU	BPU なし、BB 12, BST 8bit/BB			非互換	222
ハードウェアプリフェッチ	未定義		あり。ソフトウェアで制御できないため、仕様書には記載されていない。	あり。ソフトウェアで制御できる。				237
インタラプトレジスタ	未定義	8 本		3 本		非互換		242
命令								
impdep1	未定義	VIS	VIS, SLEEP, SUSPEND	SLEEP, SUSPEND, FCMP(EQ,LE,LT,NE,GT,GE)E(s,d), FCMP(EQ,NE)(s,d), FMAX(s,d), FMIN(s,d), FRCPA(s,d), FRSQRTA(s,d), FTRISSELd, FTRISMULd				76, 77, 114, 116, 118, 123
impdep2	未定義	未定義	F{N}M(ADD,SUB)(s,d), FPMADDX{HI}	F{N}M(ADD,SUB)(s,d), FPMADDX{HI}, FTRIMADDd, FSELMOV(s,d)				70, 78, 122
load/store			QUAD_LDD_PHYS	QUAD_LDD_PHYS, ST{D}FR, XFILL				88, 122, 133
その他			POPC	POPC, SXAR	V8[1]			94, 131
SIMD	なし			あり				
block load, block store 動作	未定義	• bst commit はキャッシュ上のデータを無効化し、メモリに書く。 • レジスタ依存を無視する。	• bst commit はキャッシュ上のデータを無効化し、メモリに書く。 • レジスタ依存は検出する。 • メモリモデルは、bld/bst 内部は RMO, 前後命令とは V9 非準拠。 • bld/bst 中に TTE が無効化されると fast_data_access_MMU_miss 発生。	• bst commit はキャッシュにストアする。 • 内部も前後命令とも TSO 遵守。		非互換		66
load 例外時の rd 更新	impdep. #44		更新しない。	non-SIMD は更新しない。 SIMD では更新されることがある。				81, 85
LDDF/ STDF_mem_address_not_aligned	impdep. #109, #110		例外通知する。	non-SIMD では例外通知する。 SIMD では通知しない。				81, 85, 100, 104

TABLE S-1 仕様差分サマリ *(3 of 4)*

項目	仕様				前方互換性			ページ
	V9	JPS1	SPARC64 VII	SPARC64 VIIIfx	V9	JPS1	SPARC64 VII	
命令属性	なし			SIMD, セクタキャッシュおよびハードウェアプリフェッチ無効が指定できる。				29
トラップ								
種類			*async_data_error*	*async_data_error*, *illegal_action*, *SIMD_load_across_pages*				51
優先順位			*async_data_error* が 2。	*async_data_error* が 2, *illegal_action* が 8.5, *SIMD_load_across_pages* が 12, *fp_exception_other* (ftt = *unimplemented_FPop*) が 8.2。 SIMD で *fp_exception_ieee754* と *fp_exception_other* (ftt = unfinished_FPop) が同時に起きたときは、*fp_exception_other* が優先される。		*fp_exception_other* が異なるが互換性には問題ない。		48
レジスタセーブ				追加レジスタに関して、 • トラップ時 TXAR[TL] ← XAR XAR ← 0 • DONE/RETRY 時 XAR ← TXAR[TL] TXAR[TL] は変化しない				49
レジスタ機能								
%ver.impl			7	8			非互換	26
%fsr.cexc 更新	高々 1 ビットがセットされる。			SIMD 演算では 2 ビットセットされることがある。				24
PA イベント種類			6 ビット	7 ビット				26, 304
watchpoint		VA,PA 独立に指定可能。		VA,PA でレジスタ共有。		非互換		36
AFAR		オプショナル	0 固定で読み出し可。	削除。			非互換	—
EIDR			bit<13:0> が有効。 bit<13:12> にはソフトウェアが 10_{02} をセットする。エラーマーク ID に使われる。	bit<2:0> が有効。 bits <13:12> はハード固定で 10_{02}。				270
SYS_CONFIG			JB_CONFIG_REGISTER UC_S, UC_SW, CLK_MODE, ITID が定義されている。	SYS_CONFIG ITID のみ定義されている。			非互換	323
その他								

TABLE S-1 仕様差分サマリ *(4 of 4)*

項目	仕様				前方互換性			ページ
	V9	JPS1	SPARC64 VII	SPARC64 VIIIfx	V9	JPS1	SPARC64 VII	
致命的エラーの要因表示			STCHG_ERROR_INFO で要因識別可能。	致命的エラー要因は表示されない。			非互換	272
STICK start/stop			なし (SC で制御)。	あり。			非互換	324

1.SXAR は V8 仕様と非互換。

Index

B

C

D

E

F

G

H

I

J

L

M

N

S

T

U

サ行

タ行

ハ行

マ行

ヤ - ワ行