

 [すべてのバージョンを表示](#)

MicroBlaze プロセッサ リファレンス ガイド

UG984 (v2018.3) 2018 年 11 月 14 日

この資料は表記のバージョンの英語版を翻訳したもので、内容に相違が生じる場合には原文を優先します。資料によっては英語版の更新に対応していないものがあります。日本語版は参考用としてご使用の上、最新情報につきましては、必ず最新英語版をご参照ください。

改訂履歴

次の表に、この文書の改訂履歴を示します。

日付	バージョン	改訂内容
2018年11月14日	2018.3	Vivado 2018.3 リリース用に更新。 <ul style="list-style-type: none">v11.0 の新機能である MicroBlaze の 64 ビット インプリメンテーションの説明を追加。
2018年4月4日	2018.1	Vivado 2018.1 リリース用に更新。 <ul style="list-style-type: none">命令パイプラインのハザードおよびフォワードに関する情報を追加。ソフトウェアブレークで MSR の BIP ビットが設定されないことを明確に記述。メモリスクラップの動作を説明。スリープおよび一時停止の使用の詳細を追加。パラレルデバッグクロックとリセットの使用に関する説明をわかりやすく記述。
2017年10月4日	2017.3	Vivado 2017.3 リリース用に更新。 <ul style="list-style-type: none">オートモーティブ UltraScale+ Zynq および Spartan-7 デバイスを追加。デバッグトレースの説明をアップデートして、バージョン 10.0 で新規導入されたイベントトレースの情報を追加。4PB 拡張アドレスサイズを追加。キャッシュトレース信号の説明をわかりやすく記述。
2017年4月5日	2017.1	Vivado 2017.1 リリース用に更新。 <ul style="list-style-type: none">MMU 物理アドレス拡張 (PAE) の新しいバージョン 10.0 の記述を追加。特権命令リストを追加して、命令の説明をアップデート。デバッグプログラムトレースに関する情報をアップデート。Triple Modular Redundancy (TMR) サブシステムへの参照文書を追加。BSIFI 命令の説明を修正。MFSE 命令の説明を PAE 情報を含めてアップデート。PAE と使用される MTSE 命令の新しいバージョン 10.0 を追加。外部キャッシュの無効とフラッシュを実行する WDC 命令をアップデート。
2016年10月5日	2016.3	Vivado 2016.3 リリース用に更新。 <ul style="list-style-type: none">v10.0 の新機能である周波数最適化の 8 段パイプラインの説明を追加。v10.0 の新機能であるビットフィールド命令の説明を追加。v10.0 の新機能であるパラレルデバッグインターフェイスの情報を含める。PVR の MicroBlaze リリースバージョンコードに v10.0 を追加。PVR に Spartan-7 ターゲットアーキテクチャを含める。MSR リセット値の説明をアップデート。ザイリンクス自動車用のアプリケーションの免責条項をアップデート。

日付	バージョン	改訂内容
2016年4月6日	2016.1	<p>Vivado 2016.1 リリース用に更新。</p> <ul style="list-style-type: none"> v9.6 の新機能であるアドレス拡張の説明を含める。 v9.6 の新機能であるパイプライン一時停止機能の説明を含める。 v9.6 の新機能である非セキュア AXI アクセス サポートの説明を含める。 v9.6 の新機能であるハイバーネートおよびサスペンド命令の説明を含める。 PVR の MicroBlaze リリース バージョン コードに v9.6 を追加。 表 2-47 および 表 2-48 への参照を修正。 XMD (Xilinx Microprocessor Debugger) を XSDB (ザイリンクス システム デバッガー) に置き換え。 svc_handler および svc_table_handler の C コード 関数属性を削除。
2015年4月15日	2015.1	<p>Vivado 2015.1 リリース用に更新。</p> <ul style="list-style-type: none"> v9.5 の新機能である 16 ワード キャッシュの説明を含める。 PVR の MicroBlaze リリース バージョン コードに v9.5 を追加。 サポートされるエンディアンネスおよびパラメーター C_ENDIANNESS の説明を修正。 命令およびデータ キャッシュの未処理読み出しの説明を修正。 FPGA コンフィギュレーション メモリ保護の資料参照 [参照 5] をアップデート。 表 3-14 のロックステップ比較のバス インデックス範囲の定期を修正。 IDIV 命令で変更されるレジスタを明記。 MFS 命令の PVR アセンブラー モニックを修正。 2015.1 用にパフォーマンスおよびリソース使用率をアップデート。 トレーニング リソースに参考資料を追加。
2014年10月1日	2014.3	<p>Vivado 2014.3 リリース用に更新。</p> <ul style="list-style-type: none"> 表 2-1 の PCMP_EQ および PCMP_NE の説明を修正。 PVR の MicroBlaze リリース バージョン コードに v9.4 を追加。 v9.4 の新機能である外部プログラム トレースの説明を含める。
2014年4月2日	2014.1	<p>Vivado 2014.1 リリース用に更新。</p> <ul style="list-style-type: none"> PVR の MicroBlaze リリース バージョン コードに v9.3 を追加。 スタッカ保護レジスタの使用と動作について明記。 LMB 命令およびデータ バス例外の説明を修正。 v9.3 の新機能である拡張デバッグ機能の説明を含める (パフォーマンス監視、プログラム トレース、非侵入型プロファイリング)。 v9.3 の新機能であるリセット モード信号の定義を含める。 AXI4-Stream TLAST 信号の処理方法を明記。 UltraScale を追加、2014.1 用にパフォーマンスおよびリソース使用率をアップデート。
2013年12月18日	2013.4	Vivado 2013.4 リリース用に更新。
2013年10月2日	2013.3	Vivado 2013.3 リリース用に更新。
2013年6月19日	2013.2	Vivado 2013.2 リリース用に更新。
2013年3月20日	2013.1	初版。UG081 を基に作成。

目次

第 1 章: 概要

内容	6
----------	---

第 2 章: MicroBlaze アーキテクチャ

はじめに	7
概要	7
データ型とエンディアンネス	11
命令	13
レジスタ	27
パイプライン アーキテクチャ	54
メモリアーキテクチャ	59
特権命令	60
仮想メモリ管理	61
リセット、割り込み、例外、およびブレーク	74
命令キヤッショ	81
データ キヤッショ	84
浮動小数点ユニット (FPU)	89
ストリームリンクインターフェイス	94
デバッグおよびトレース	95
フォールトトレランス	117
ロックステップ操作	124
コヒーレンシ	127
データおよび命令のアドレス拡張	129

第 3 章: MicroBlaze 信号インターフェイスの説明

はじめに	131
概要	131
MicroBlaze の I/O 概要	132
AXI4 および ACE インターフェイスについて	145
ローカルメモリバス (LMB) インターフェイスについて	151
ロックステップインターフェイスについて	160
デバッグインターフェイスについて	165
トレースインターフェイスについて	167
MicroBlaze コアのコンフィギュレーション	170

第 4 章: MicroBlaze アプリケーションバイナリインターフェイス

概要	182
データ型	182
レジスタの使用規則	183
スタック規則	185
メモリモデル	187

割り込み、ブレーク、例外処理	188
第 5 章: MicroBlaze 命令セット アーキテクチャ	
概要	190
記号の説明	190
フォーマット	192
MicroBlaze 32 ビット命令	192
MicroBlaze 64 ビット命令	298
付録 A: パフォーマンスおよびリソース使用率	
パフォーマンス	352
リソース使用量	353
IP 特性化および f_{MAX} マージン システム手法	361
付録 B: その他のリソースおよび法的通知	
ザイリンクス リソース	362
ソリューションセンター	362
Documentation Navigator およびデザインハブ	362
参考資料	363
トレーニング リソース	363
お読みください: 重要な法的通知	364

概要

この資料は、Vivado に含まれる 32 ビットおよび 64 ビットのソフト プロセッサである MicroBlaze™ に関する情報を提供する MicroBlaze ハードウェア アーキテクチャのリファレンス ガイドです。

内容

このガイドは、次の各章で構成されています。

- 第 2 章 「MicroBlaze アーキテクチャ」では、MicroBlaze の機能の概要をはじめ、ビッグ エンディアンおよびリトル エンディアンのビット反転フォーマット、32 ビットまたは 64 ビットの汎用レジスタ、キャッシュ ソフトウェア サポート、および AXI4-Stream インターフェイスなどの機能も説明します。
- 第 3 章 「MicroBlaze 信号インターフェイスの説明」では、MicroBlaze を接続するために使用可能な信号インターフェイスの種類を説明します。
- 第 4 章 「MicroBlaze アプリケーションバイナリ インターフェイス」では、MicroBlaze 用にアセンブリ言語でソフトウェアを開発するのに重要な、アプリケーションバイナリ インターフェイスを説明します。
- 第 5 章 「MicroBlaze 命令セット アーキテクチャ」では、MicroBlaze の命令セットアーキテクチャ (ISA) のシンボル、フォーマット、および命令について説明します。
- 付録 A 「パフォーマンスおよびリソース使用率」では、各コンフィギュレーションおよびデバイスの最大周波数およびリソース使用率を説明します。
- 付録 B 「その他のリソースおよび法的通知」には、さまざまな資料およびその他のリソースへのリンクがあります。

MicroBlaze アーキテクチャ

はじめに

この章では、MicroBlaze™ の機能の概要と、ビッグ エンディアンまたはリトル エンディアンのビット反転フォーマット、32 ビットまたは 64 ビット汎用レジスタ、仮想メモリ管理、キャッシングソフトウェアサポート、AXI4-Stream インターフェイスなど、このアーキテクチャの詳細について説明します。

概要

MicroBlaze エンベデッド プロセッサは、ザイリンクス フィールド プログラマブルゲート アレイ (FPGA) のインプリメンテーション用に最適化された RISC (Reduced Instruction Set Computer) コアです。次の図に、MicroBlaze コアのブロック図を示します。

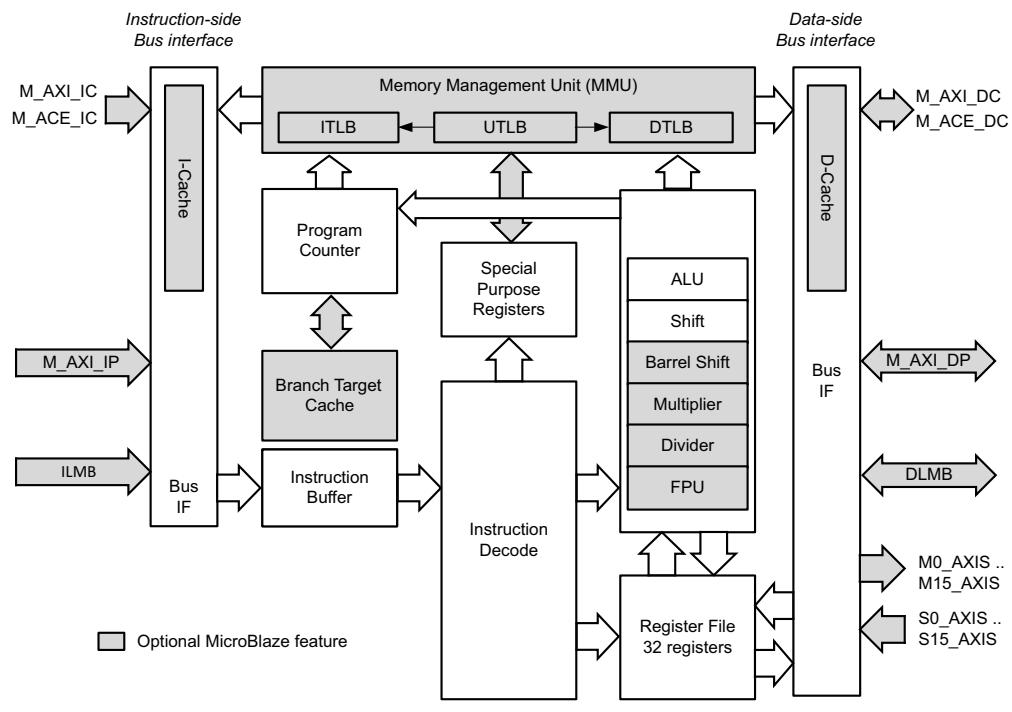


図 2-1: MicroBlaze コアのブロック図

機能

MicroBlaze ソフトコアプロセッサは細かく設定できるため、デザインに必要な特定の機能セットを選択できます。

プロセッサの固定機能セットには、次のものがあります。

- 32 個の 32 ビットまたは 64 ビット汎用レジスタ
- オペランド 3 つとアドレス指定モード 2 つを含む 32 ビット命令ワード
- デフォルトの 32 ビットアドレスバス (64 ビットに拡張可能)
- 単一発行のパイプライン

これらの固定機能のほか、MicroBlaze プロセッサには追加機能を選択して有効にできるパラメーター値があります。古いバージョン (サポートされなくなったもの) の MicroBlaze では、この資料で説明するオプションの機能の一部のみがサポートされます。最新バージョンの MicroBlaze (v11.0) では、すべてのオプションがサポートされます。



推奨: 新しいデザインには最新バージョンの MicroBlaze を使用してください。

次の表には、MicroBlaze のバージョン別に、コンフィギュレーション可能な機能がまとめられています。

表 2-1: コンフィギュレーション可能な機能 (MicroBlaze バージョン別)

機能	MicroBlaze バージョン					
	v9.3	v9.4	v9.5	v9.6	v10.0	v11.0
バージョンステータス	サポート終了	サポート終了	サポート終了	サポート終了	サポート終了	推奨
プロセッサのパイプラインの深さ	3/5	3/5	3/5	3/5	3/5/8	3/5/8
ローカルメモリバス (LMB) データ側インターフェイス	オプション	オプション	オプション	オプション	オプション	オプション
ローカルメモリバス (LMB) 命令側インターフェイス	オプション	オプション	オプション	オプション	オプション	オプション
ハードウェアバレルシフター	オプション	オプション	オプション	オプション	オプション	オプション
ハードウェア除算器	オプション	オプション	オプション	オプション	オプション	オプション
ハードウェアのデバッグロジック	オプション	オプション	オプション	オプション	オプション	オプション
ストリームリンクインターフェイス	0-16 AXI	0-16 AXI	0-16 AXI	0-16 AXI	0-16 AXI	0-16 AXI
マシンステータスセットおよびクリア命令	オプション	オプション	オプション	オプション	オプション	オプション
キヤッシュラインワード長	4、8	4、8	4、8、16	4、8、16	4、8、16	4、8、16
ハードウェア例外サポート	オプション	オプション	オプション	オプション	オプション	オプション
パターン比較命令	オプション	オプション	オプション	オプション	オプション	オプション
浮動小数点ユニット (FPU)	オプション	オプション	オプション	オプション	オプション	オプション
ハードウェア乗算器のディスクエーブル ¹	オプション	オプション	オプション	オプション	オプション	オプション
ハードウェアデバッグの読み出し可能な ESR および EAR	あり	あり	あり	あり	あり	あり

表 2-1: コンフィギュレーション可能な機能 (MicroBlaze バージョン別) (続き)

機能	MicroBlaze バージョン					
	v9.3	v9.4	v9.5	v9.6	v10.0	v11.0
プロセッサ バージョン レジスタ (PVR)	オプション	オプション	オプション	オプション	オプション	オプション
エリアまたはスピードの 最適化	オプション	オプション	オプション	オプション	オプション	オプション
ハードウェア乗算器 64 ビット 結果	オプション	オプション	オプション	オプション	オプション	オプション
LUT キャッシュ メモリ	オプション	オプション	オプション	オプション	オプション	オプション
浮動小数点変換および平方根 命令	オプション	オプション	オプション	オプション	オプション	オプション
メモリ管理ユニット (MMU)	オプション	オプション	オプション	オプション	オプション	オプション
拡張ストリーム命令	オプション	オプション	オプション	オプション	オプション	オプション
すべての I キャッシュ メモリ アクセスにキャッシュ インターフェイスを使用	オプション	オプション	オプション	オプション	オプション	オプション
すべての D キャッシュ メモリ アクセスにキャッシュ インターフェイスを使用	オプション	オプション	オプション	オプション	オプション	オプション
D キャッシュのライトバック キャッシング ポリシーを使用	オプション	オプション	オプション	オプション	オプション	オプション
分岐先キャッシュ (BTC)	オプション	オプション	オプション	オプション	オプション	オプション
I キャッシュ用ストリーム	オプション	オプション	オプション	オプション	オプション	オプション
I キャッシュ用ビクティム処理	オプション	オプション	オプション	オプション	オプション	オプション
D キャッシュ用ビクティム処理	オプション	オプション	オプション	オプション	オプション	オプション
AXI4 (M_AXI_DP) データ側 インターフェイス	オプション	オプション	オプション	オプション	オプション	オプション
AXI4 (M_AXI_IP) 命令側 インターフェイス	オプション	オプション	オプション	オプション	オプション	オプション
D キャッシュ用 AXI4 (M_AXI_DC) プロトコル	オプション	オプション	オプション	オプション	オプション	オプション
I キャッシュ用 AXI4 (M_AXI_IC) プロトコル	オプション	オプション	オプション	オプション	オプション	オプション
ストリーム アクセス用 AXI4 プロトコル	オプション	オプション	オプション	オプション	オプション	オプション
フォールト トレランス	オプション	オプション	オプション	オプション	オプション	オプション
キャッシュ タグ用に分散 RAM を使用	オプション	オプション	オプション	オプション	オプション	オプション
コンフィギュラブル キャッシュ データ幅	オプション	オプション	オプション	オプション	オプション	オプション
前ゼロのカウント命令	オプション	オプション	オプション	オプション	オプション	オプション
メモリ バリア命令	あり	あり	あり	あり	あり	あり

表 2-1: コンフィギュレーション可能な機能 (MicroBlaze バージョン別) (続き)

機能	MicroBlaze バージョン					
	v9.3	v9.4	v9.5	v9.6	v10.0	v11.0
スタック オーバーフロー およびアンダーフローの検出	オプション	オプション	オプション	オプション	オプション	オプション
ユーザー モードのストリーム 命令	オプション	オプション	オプション	オプション	オプション	オプション
ロックステップ サポート	オプション	オプション	オプション	オプション	オプション	オプション
FPGA プリミティブの コンフィギュレーション設定	オプション	オプション	オプション	オプション	オプション	オプション
低レイテンシ割り込みモード	オプション	オプション	オプション	オプション	オプション	オプション
スワップ命令	オプション	オプション	オプション	オプション	オプション	オプション
スリープ モードおよび スリープ命令	あり	あり	あり	あり	あり	あり
配置換え可能なベース ベクター	オプション	オプション	オプション	オプション	オプション	オプション
D キャッシュ用 ACE (M_ACE_DC) プロトコル	オプション	オプション	オプション	オプション	オプション	オプション
I キャッシュ用 ACE (M_ACE_IC) プロトコル	オプション	オプション	オプション	オプション	オプション	オプション
拡張デバッグ: パフォーマンス 監視、プログラムトレース、 非侵入型プロファイリング	オプション	オプション	オプション	オプション	オプション	オプション
リセット モード: スリープに 入る、またはリセットで デバッグ停止	オプション	オプション	オプション	オプション	オプション	オプション
拡張デバッグ: 外部プログラム トレース		オプション	オプション	オプション	オプション	オプション
拡張データ アドレス指定				オプション	オプション	オプション
パイプライン一時停止機能				あり	あり	あり
ハイバーネートおよび サスペンド命令				あり	あり	あり
非セキュア モード				あり	あり	あり
ビット フィールド命令 ²					オプション	オプション
パラレルデバッグ インターフェイス					オプション	オプション
MMU 物理アドレス拡張					オプション	オプション
64 ビット モード						オプション

1. DSP48E プリミティブを節約するために使用。

2. C_USE_BARREL=1 のときビット フィールド命令は使用可能。

データ型とエンディアンネス

MicroBlaze プロセッサでは、エンディアンネスの設定により、データ表現にビッグ エンディアンまたはリトル エンディアンが使用されます。パラメーター C_ENDIANNESS はデフォルトで 1(リトル エンディアン)です。

32 ビット MicroBlaze では、ハードウェア サポートされているデータ型はワード、ハーフ ワード、およびバイトです。64 ビット MicroBlaze では、ハードウェアで long および double データ型もサポートされています。

反転ロードおよび反転ストア命令の LHUR、LWR、LLR、SHR、SWR、および SLR を使用すると、表のバイト反転順に示すように、データ内のバイトが反転されます。

次の表に、各データ型のビットおよびバイト構成を示します。

表 2-2: long データ型 (64 ビット MicroBlaze のみ)

	n	n+1	n+2	n+3	n+4	n+5	n+6	n+7
ビッグ エンディアン バイト アドレス								
ビッグ エンディアン上位/下位バイト	最上位 バイト							最下位 バイト
ビッグ エンディアン バイト順序	n	n+1	n+2	n+3	n+4	n+5	n+6	n+7
ビッグ エンディアン バイト反転順序	n+7	n+6	n+5	n+4	n+3	n+2	n+1	n
リトル エンディアン バイト アドレス	n+7	n+6	n+5	n+4	n+3	n+2	n+1	n
リトル エンディアン上位/下位バイト	最上位 バイト							最下位 バイト
リトル エンディアン バイト順序	n+7	n+6	n+5	n+4	n+3	n+2	n+1	n
リトル エンディアン バイト反転順序	n	n+1	n+2	n+3	n+4	n+5	n+6	n+7
ビット ラベル	0							63
上位ビット/下位ビット	最上位 ビット							最下位 ビット

表 2-3: ワード データ型

	n	n+1	n+2	n+3
ビッグ エンディアン バイト アドレス				
ビッグ エンディアン上位/下位バイト	最上位 バイト			最下位 バイト
ビッグ エンディアン バイト順序	n	n+1	n+2	n+3
ビッグ エンディアン バイト反転順序	n+3	n+2	n+1	n
リトル エンディアン バイト アドレス	n+3	n+2	n+1	n
リトル エンディアン上位/下位バイト	最上位 バイト			最下位 バイト
リトル エンディアン バイト順序	n+3	n+2	n+1	n
リトル エンディアン バイト反転順序	n	n+1	n+2	n+3
ビット ラベル	0			31
上位ビット/下位ビット	最上位 ビット			最下位 ビット

表 2-4: ハーフワードデータ型

ビッグエンディアンバイトアドレス	n	n+1
ビッグエンディアン上位/下位バイト	最上位 バイト	最下位 バイト
ビッグエンディアンバイト順序	n	n+1
ビッグエンディアンバイト反転順序	n+1	n
リトルエンディアンバイトアドレス	n+1	n
リトルエンディアン上位/下位バイト	最上位 バイト	最下位 バイト
リトルエンディアンバイト順序	n+1	n
リトルエンディアンバイト反転順序	n	n+1
ビットラベル	0	15
上位ビット/下位ビット	最上位 ビット	最下位 ビット

表 2-5: バイトデータ型

バイトアドレス	n
ビットラベル	0 7
上位ビット/下位ビット	最上位 ビット
	最下位 ビット

命令

命令のまとめ

MicroBlaze 命令はすべて 32 ビットで、タイプ A またはタイプ B として定義されています。タイプ A 命令には、ソースレジスタオペランドが最高で 2 つ、デスティネーションレジスタオペランドが 1 つあります。タイプ B 命令にはソースレジスタおよび 16 ビットの即値オペランドが 1 つずつあります (タイプ B 命令の前に imm 命令を使用することにより 32 ビットに拡張可能)。

タイプ B 命令にはシングルデスティネーションレジスタオペランドが 1 つあります。命令には、演算、論理、分岐、ロード/ストア、および特殊、というカテゴリがあります。次の表には、各命令のセマンティクスで使用される命令セット用語が説明されています。[表 2-6](#) には MicroBlaze 命令セットがリストされています。これらの命令の詳細は、[第 5 章「MicroBlaze 命令セット アーキテクチャ」](#) を参照してください。

表 2-6: 命令セットの用語

記号	説明
Ra	R0 ~ R31、汎用レジスタ、ソースオペランド a <ul style="list-style-type: none"> 32 ビット MicroBlaze では、32 ビット レジスタ全体を表します。 64 ビット MicroBlaze および L=0 では、下位 32 ビットを表します。 64 ビット MicroBlaze および L=1 では、64 ビット レジスタ全体を表します。 命令ビット L の定義は 表 2-7 を参照してください。
Rb	R0 ~ R31、汎用レジスタ、ソースオペランド b <ul style="list-style-type: none"> 32 ビット MicroBlaze では、32 ビット レジスタ全体を表します。 64 ビット MicroBlaze および L=0 では、下位 32 ビットを表します。 64 ビット MicroBlaze および L=1 では、64 ビット レジスタ全体を表します。 命令ビット L の定義は 表 2-7 を参照してください。
Rd	R0 ~ R31、汎用レジスタ、デスティネーションオペランド <ul style="list-style-type: none"> 32 ビット MicroBlaze では、32 ビット レジスタ全体が結果に割り当てられます。 64 ビット MicroBlaze および L=0 では、下位 32 ビットが結果に割り当てられます。 64 ビット MicroBlaze および L=1 では、64 ビット レジスタ全体が結果に割り当てられます。 命令ビット L の定義は 表 2-7 を参照してください。
SPR[x]	特殊用途レジスタ番号 x
MSR	マシンステータスレジスタ = SPR[1]
ESR	例外ステータスレジスタ = SPR[5]
EAR	例外アドレスレジスタ = SPR[3]
FSR	浮動小数点ユニットステータスレジスタ = SPR[7]
PVRx	プロセッサバージョンレジスタ、x はレジスタ番号 = SPR[8192 + x]
BTR	分岐先レジスタ = SPR[11]
PC	実行段プログラムカウンター = SPR[0]
x[y]	レジスタ x のビット y

表 2-6: 命令セットの用語(続き)

記号	説明
$x[y:z]$	レジスタ x のビット範囲 $y \sim z$
\bar{x}	レジスタ x のビット反転値
Imm	16 ビットの即値
Imm x	x ビットの即値
FSL x	4 ビット AXI4-Stream ポートの指定 (x はポート番号)
C	キャリーフラグ、MSR[29]
Sa	特殊用途レジスタ、ソースオペランド
Sd	特殊用途レジスタ、デスティネーションオペランド
s(x)	引数 x を 32 ビットまたは 64 ビット値に符号拡張
*Addr	ロケーション Addr でのメモリ内容(データサイズは揃っている)
:=	割り当て演算子
$=$	同等比較
\neq	不等比較
$>$	大なり
\geq	それ以上
$<$	小なり
\leq	それ以下
$+$	加算
$*$	乗算
$/$	除算
$>> x$	x ビット右シフト
$<< x$	x ビット左シフト
and	論理 AND
or	論理 OR
xor	論理排他的 OR
op1 if cond else op2	条件 $cond$ が真の場合は op1 を実行し、それ以外は op2 を実行
$\&$	連結。たとえば、「0000100 & Imm7」は固定フィールド「0000100」と 7 ビット即値の連結です。
signed	符号付きの整数データ型で実行する演算。すべての演算は、指定がない限り、符号付きワードオペランドで実行されます。
unsigned	符号なしの整数データ型で実行する演算
float	浮動小数点のデータ型で実行する演算
clz(r)	前ゼロのカウント

表 2-7: MicroBlaze 命令セットのまとめ

タイプ A	0 ~ 5	6 ~ 10	11 ~ 15	16 ~ 20	21 ~ 31	セマンティクス
タイプ B	0 ~ 5	6 ~ 10	11 ~ 15	16 ~ 31		
ADD Rd,Ra,Rb	000000	Rd	Ra	Rb	00L00000000	$Rd := Rb + Ra$
RSUB Rd,Ra,Rb	000001	Rd	Ra	Rb	00L00000000	$Rd := Rb + \overline{Ra} + 1$
ADDC Rd,Ra,Rb	000010	Rd	Ra	Rb	00L00000000	$Rd := Rb + Ra + C$
RSUBC Rd,Ra,Rb	000011	Rd	Ra	Rb	00L00000000	$Rd := Rb + \overline{Ra} + C$
ADDK Rd,Ra,Rb	000100	Rd	Ra	Rb	00L00000000	$Rd := Rb + Ra$
RSUBK Rd,Ra,Rb	000101	Rd	Ra	Rb	00L00000000	$Rd := Rb + \overline{Ra} + 1$
CMP Rd,Ra,Rb	000101	Rd	Ra	Rb	00L00000001	$Rd := Rb + \overline{Ra} + 1$ $Rd[0] := 0$ if ($Rb \geq Ra$) else $Rd[0] := 1$
CMPU Rd,Ra,Rb	000101	Rd	Ra	Rb	00L00000011	$Rd := Rb + \overline{Ra} + 1$ (unsigned) $Rd[0] := 0$ if ($Rb \geq Ra$, unsigned) else $Rd[0] := 1$
ADDKC Rd,Ra,Rb	000110	Rd	Ra	Rb	00L00000000	$Rd := Rb + Ra + C$
RSUBKC Rd,Ra,Rb	000111	Rd	Ra	Rb	00L00000000	$Rd := Rb + \overline{Ra} + C$
ADDI Rd,Ra,Imm	001000	Rd	Ra	Imm		$Rd := s(Imm) + Ra$
RSUBI Rd,Ra,Imm	001001	Rd	Ra	Imm		$Rd := s(Imm) + \overline{Ra} + 1$
ADDIC Rd,Ra,Imm	001010	Rd	Ra	Imm		$Rd := s(Imm) + Ra + C$
RSUBIC Rd,Ra,Imm	001011	Rd	Ra	Imm		$Rd := s(Imm) + \overline{Ra} + C$
ADDIK Rd,Ra,Imm	001100	Rd	Ra	Imm		$Rd := s(Imm) + Ra$
RSUBIK Rd,Ra,Imm	001101	Rd	Ra	Imm		$Rd := s(Imm) + \overline{Ra} + 1$
ADDIKC Rd,Ra,Imm	001110	Rd	Ra	Imm		$Rd := s(Imm) + Ra + C$
RSUBIKC Rd,Ra,Imm	001111	Rd	Ra	Imm		$Rd := s(Imm) + \overline{Ra} + C$
MUL Rd,Ra,Rb	010000	Rd	Ra	Rb	000000000000	$Rd := Ra * Rb$
MULH Rd,Ra,Rb	010000	Rd	Ra	Rb	000000000001	$Rd := (Ra * Rb) >> 32$ (signed)
MULHU Rd,Ra,Rb	010000	Rd	Ra	Rb	000000000011	$Rd := (Ra * Rb) >> 32$ (unsigned)
MULHSU Rd,Ra,Rb	010000	Rd	Ra	Rb	000000000010	$Rd := (Ra, signed * Rb, unsigned) >> 32$ (signed)
BSRL Rd,Ra,Rb	010001	Rd	Ra	Rb	00L00000000	$Rd := 0 \& (Ra >> Rb)$
BSRA Rd,Ra,Rb	010001	Rd	Ra	Rb	01L00000000	$Rd := s(Ra >> Rb)$
BSLL Rd,Ra,Rb	010001	Rd	Ra	Rb	10L00000000	$Rd := (Ra << Rb) \& 0$
IDIV Rd,Ra,Rb	010010	Rd	Ra	Rb	000000000000	$Rd := Rb/Ra$
IDIVU Rd,Ra,Rb	010010	Rd	Ra	Rb	000000000010	$Rd := Rb/Ra$, unsigned

表 2-7: MicroBlaze 命令セットのまとめ(続き)

タイプ A	0 ~ 5	6 ~ 10	11 ~ 15	16 ~ 20	21 ~ 31	セマンティクス
タイプ B	0 ~ 5	6 ~ 10	11 ~ 15	16 ~ 31		
TNEAGETD Rd,Rb	010011	Rd	00000	Rb	0N0TAE 00000	Rd := FSL Rb[28:31] (data read) MSR[FSL] := 1 if (FSL_S_Control = 1) MSR[C] := not FSL_S_Exists if N = 1
TNAPUTD Ra,Rb	010011	00000	Ra	Rb	0N0TA0 00000	FSL Rb[28:31] := Ra (data write) MSR[C] := FSL_M_Full if N = 1
TNECAGETD Rd,Rb	010011	Rd	00000	Rb	0N1TAE 00000	Rd := FSL Rb[28:31] (control read) MSR[FSL] := 1 if (FSL_S_Control = 0) MSR[C] := not FSL_S_Exists if N = 1
TNCAPUTD Ra,Rb	010011	00000	Ra	Rb	0N1TA0 00000	FSL Rb[28:31] := Ra (control write) MSR[C] := FSL_M_Full if N = 1
FADD Rd,Ra,Rb	010110	Rd	Ra	Rb	000000000000	Rd := Rb+Ra, float ¹
FRSUB Rd,Ra,Rb	010110	Rd	Ra	Rb	000100000000	Rd := Rb-Ra, float ¹
FMUL Rd,Ra,Rb	010110	Rd	Ra	Rb	001000000000	Rd := Rb*Ra, float ¹
FDIV Rd,Ra,Rb	010110	Rd	Ra	Rb	001100000000	Rd := Rb/Ra, float ¹
FCMP.UN Rd,Ra,Rb	010110	Rd	Ra	Rb	010000000000	Rd := 1 if (Rb = NaN or Ra = NaN, float ¹) else Rd := 0
FCMP.LT Rd,Ra,Rb	010110	Rd	Ra	Rb	01000010000	Rd := 1 if (Rb < Ra, float ¹) else Rd := 0
FCMP.EQ Rd,Ra,Rb	010110	Rd	Ra	Rb	01000100000	Rd := 1 if (Rb = Ra, float ¹) else Rd := 0
FCMP.LE Rd,Ra,Rb	010110	Rd	Ra	Rb	01000110000	Rd := 1 if (Rb <= Ra, float ¹) else Rd := 0
FCMP.GT Rd,Ra,Rb	010110	Rd	Ra	Rb	01001000000	Rd := 1 if (Rb > Ra, float ¹) else Rd := 0
FCMP.NE Rd,Ra,Rb	010110	Rd	Ra	Rb	01001010000	Rd := 1 if (Rb != Ra, float ¹) else Rd := 0
FCMP.GE Rd,Ra,Rb	010110	Rd	Ra	Rb	01001100000	Rd := 1 if (Rb >= Ra, float ¹) else Rd := 0
FLT Rd,Ra	010110	Rd	Ra	0	01010000000	Rd := float(Ra) ¹
FINT Rd,Ra	010110	Rd	Ra	0	01100000000	Rd := int(Ra) ¹
FSQRT Rd,Ra	010110	Rd	Ra	0	01110000000	Rd := sqrt(Ra) ¹
DADD Rd,Ra,Rb ²	010110	Rd	Ra	Rb	100000000000	Rd := Rb+Ra, double ¹
DRSUB Rd,Ra,Rb ²	010110	Rd	Ra	Rb	10010000000	Rd := Rb-Ra, double ¹

表 2-7: MicroBlaze 命令セットのまとめ(続き)

タイプ A	0 ~ 5	6 ~ 10	11 ~ 15	16 ~ 20	21 ~ 31	セマンティクス
タイプ B	0 ~ 5	6 ~ 10	11 ~ 15	16 ~ 31		
DMUL Rd,Ra,Rb ²	010110	Rd	Ra	Rb	10100000000	Rd := Rb*Ra, double ¹
DDIV Rd,Ra,Rb ²	010110	Rd	Ra	Rb	10110000000	Rd := Rb/Ra, double ¹
DCMP.UN Rd,Ra,Rb ²	010110	Rd	Ra	Rb	11000000000	Rd := 1 if (Rb = NaN or Ra = NaN, double ¹) else Rd := 0
DCMP.LT Rd,Ra,Rb ²	010110	Rd	Ra	Rb	11000010000	Rd := 1 if (Rb < Ra, double ¹) else Rd := 0
DCMP.EQ Rd,Ra,Rb ²	010110	Rd	Ra	Rb	11000100000	Rd := 1 if (Rb = Ra, double ¹) else Rd := 0
DCMP.LE Rd,Ra,Rb ²	010110	Rd	Ra	Rb	11000110000	Rd := 1 if (Rb <= Ra, double ¹) else Rd := 0
DCMP.GT Rd,Ra,Rb ²	010110	Rd	Ra	Rb	11001000000	Rd := 1 if (Rb > Ra, double ¹) else Rd := 0
DCMP.NE Rd,Ra,Rb ²	010110	Rd	Ra	Rb	11001010000	Rd := 1 if (Rb != Ra, double ¹) else Rd := 0
DCMP.GE Rd,Ra,Rb ²	010110	Rd	Ra	Rb	11001100000	Rd := 1 if (Rb >= Ra, double ¹) else Rd := 0
DBL Rd,Ra ²	010110	Rd	Ra	0	11010000000	Rd := double (Ra) ¹
DLONG Rd,Ra ²	010110	Rd	Ra	0	11100000000	Rd := long (Ra) ¹
DSQRT Rd,Ra ²	010110	Rd	Ra	0	11110000000	Rd := dsqrt (Ra) ¹
MULI Rd,Ra,Imm	011000	Rd	Ra	Imm		Rd := Ra * s(Imm)
BSRLI Rd,Ra,Imm	011001	Rd	Ra	00L0000000 & Imm5		Rd := 0 & (Ra >> Imm5)
BSRAI Rd,Ra,Imm	011001	Rd	Ra	00L00010000 & Imm5		Rd := s(Ra >> Imm5)
BSLLI Rd,Ra,Imm	011001	Rd	Ra	00L00100000 & Imm5		Rd := (Ra << Imm5) & 0
BSEFI Rd,Ra, Imm _W ,Imm _S	011001	Rd	Ra	01L00 & Imm _W & 0 & Imm _S		Rd[0:31-Imm _W] := 0 Rd[32-Imm _W :31] := (Ra >> Imm _S)
BSIFI Rd,Ra, Width,Imm _S	011001	Rd	Ra	10L00 & Imm _W & 0 & Imm _S		M := (0xffffffff << (Imm _W + 1)) xor (0xffffffff << Imm _S) Rd := ((Ra << Imm _S) and M) xor (Rd and \overline{M}) Imm _W := Imm _S + Width - 1
TNEAGET Rd,FSLx	011011	Rd	00000	0N0TAE000000 & FSLx		Rd := FSLx (data read, blocking if $N = 0$) MSR[FSL] := 1 if (FSLx_S_Control = 1) MSR[C] := not FSLx_S_Exists if $N = 1$

表 2-7: MicroBlaze 命令セットのまとめ(続き)

タイプ A	0 ~ 5	6 ~ 10	11 ~ 15	16 ~ 20	21 ~ 31	セマンティクス
タイプ B	0 ~ 5	6 ~ 10	11 ~ 15	16 ~ 31		
TNAPUT Ra,FSLx	011011	00000	Ra	1N0TA0000000 & FSLx		FSLx := Ra (data write, block if N=0) MSR[C] := FSLx_M_Full if N=1
TNECAGET Rd,FSLx	011011	Rd	00000	0N1TAE0000000 & FSLx		Rd := FSLx (control read, block if N=0) MSR[FSL] := 1 if (FSLx_S_Control = 0) MSR[C] := not FSLx_S_Exists if N=1
TNCAPUT Ra,FSLx	011011	00000	Ra	1N1TA0000000 & FSLx		FSLx := Ra (control write, block if N=0) MSR[C] := FSLx_M_Full if N=1
OR Rd,Ra,Rb	100000	Rd	Ra	Rb	000000000000	Rd := Ra or Rb
PCMPBF Rd,Ra,Rb	100000	Rd	Ra	Rb	100000000000	Rd := 1 if (Rb[0:7] = Ra[0:7]) else Rd := 2 if (Rb[8:15] = Ra[8:15]) else Rd := 3 if (Rb[16:23] = Ra[16:23]) else Rd := 4 if (Rb[24:31] = Ra[24:31]) else Rd := 0
AND Rd,Ra,Rb	100001	Rd	Ra	Rb	000000000000	Rd := Ra and Rb
XOR Rd,Ra,Rb	100010	Rd	Ra	Rb	000000000000	Rd := Ra xor Rb
PCMPEQ Rd,Ra,Rb	100010	Rd	Ra	Rb	100000000000	Rd := 1 if (Rb = Ra) else Rd := 0
ANDN Rd,Ra,Rb	100011	Rd	Ra	Rb	000000000000	Rd := Ra and \overline{Rb}
PCMPNE Rd,Ra,Rb	100011	Rd	Ra	Rb	100000000000	Rd := 1 if (Rb != Ra) else Rd := 0
SRA Rd,Ra	100100	Rd	Ra	000000000000000001		Rd := s(Ra >> 1) C := Ra[31]
SRC Rd,Ra	100100	Rd	Ra	0000000000100001		Rd := C & (Ra >> 1) C := Ra[31]
SRL Rd,Ra	100100	Rd	Ra	0000000001000001		Rd := 0 & (Ra >> 1) C := Ra[31]
SEXT8 Rd,Ra	100100	Rd	Ra	0000000001100000		Rd := s(Ra[24:31])
SEXT16 Rd,Ra	100100	Rd	Ra	0000000001100001		Rd := s(Ra[16:31])
SEXTL32 Rd,Ra ²	100100	Rd	Ra	0000000001100010		Rd := s(Ra[32:63])
CLZ Rd, Ra	100100	Rd	Ra	0000000011100000		Rd = clz(Ra)
SWAPB Rd, Ra	100100	Rd	Ra	0000000111100000		Rd = (Ra)[24:31, 16:23, 8:15, 0:7]
SWAPH Rd, Ra	100100	Rd	Ra	0000000111100010		Rd = (Ra)[16:31, 0:15]

表 2-7: MicroBlaze 命令セットのまとめ(続き)

タイプ A	0 ~ 5	6 ~ 10	11 ~ 15	16 ~ 20	21 ~ 31	セマンティクス
タイプ B	0 ~ 5	6 ~ 10	11 ~ 15	16 ~ 31		
WIC Ra,Rb	100100	00000	Ra	Rb	00001101000	ICache_Line[Ra >> 4].Tag := 0 if (C_ICACHE_LINE_LEN = 4) ICache_Line[Ra >> 5].Tag := 0 if (C_ICACHE_LINE_LEN = 8) ICache_Line[Ra >> 6].Tag := 0 if (C_ICACHE_LINE_LEN = 16)
WDC Ra,Rb	100100	00000	Ra	Rb	00001100100	キャッシュ ラインはクリア、格納されたデータを破棄。 DCache_Line[Ra >> 4].Tag := 0 if (C_DCACHE_LINE_LEN = 4) DCache_Line[Ra >> 5].Tag := 0 if (C_DCACHE_LINE_LEN = 8) DCache_Line[Ra >> 6].Tag := 0 if (C_DCACHE_LINE_LEN = 16)
WDC.FLUSH Ra,Rb	100100	00000	Ra	Rb	00001110100	キャッシュ ラインはフラッシュ、格納されたデータをメモリに書き込んでからクリア。C_DCACHE_USE_WRITEBACK = 1 の場合にのみ使用。
WDC.CLEAR Ra,Rb	100100	00000	Ra	Rb	00001100110	一致したアドレスのキャッシュ ラインはクリア、格納されたデータを破棄。 C_DCACHE_USE_WRITEBACK = 1 の場合にのみ使用。
WDC.CLEAR.EA Ra,Rb	100100	00000	Ra	Rb	00011100110	一致した拡張アドレス Ra & Rb のキャッシュ ラインはクリア。 C_DCACHE_USE_WRITEBACK = 1 の場合にのみ使用。
MTS Sd,Ra	100101	00000	Ra	11 & Sd		SPR[Sd] := Ra · SPR[0x0001] は MSR · SPR[0x0007] は FSR · SPR[0x0800] は SLR · SPR[0x0802] は SHR · SPR[0x1000] は PID · SPR[0x1001] は ZPR · SPR[0x1002] は TLBX · SPR[0x1003] は TLBLO[LSH] · SPR[0x1004] は TLBHI · SPR[0x1005] は TLBSX
MTSE Sd,Ra	100101	01000	Ra	11 & Sd		SPR[Sd] := Ra: · SPR[0x1003] は TLBLO[MSH]

表2-7: MicroBlaze 命令セットのまとめ(続き)

タイプ A	0 ~ 5	6 ~ 10	11 ~ 15	16 ~ 20	21 ~ 31	セマンティクス
タイプ B	0 ~ 5	6 ~ 10	11 ~ 15	16 ~ 31		
MFS Rd,Sa	100101	Rd	00000	10 & Sa		Rd := SPR[Sa] · SPR[0x0000] は PC · SPR[0x0001] は MSR · SPR[0x0003] は EAR[LSH] · SPR[0x0005] は ESR · SPR[0x0007] は FSR · SPR[0x000B] は BTR · SPR[0x000D] は EDR · SPR[0x0800] は SLR · SPR[0x0802] は SHR · SPR[0x1000] は PID · SPR[0x1001] は ZPR · SPR[0x1002] は TLBX · SPR[0x1003] は TLBLO[LSH] · SPR[0x1004] は TLBHI · SPR[0x2000-200B] は PVR[0-12][LSH]
MFSE Rd,Sa	100101	Rd	01000	10 & Sa		Rd := SPR[Sa][MSH]: · SPR[0x0003] は EAR[MSH] · SPR[0x1003] は TLBLO[MSH] · SPR[0x2006-2009] は PVR[6-9][MSH]
MSRCLR Rd,Imm	100101	Rd	00001	00 & Imm14		Rd := MSR MSR := MSR and <u>Imm14</u>
MSRSET Rd,Imm	100101	Rd	00000	00 & Imm14		Rd := MSR MSR := MSR or Imm14
BR Rb	100110	00000	00000	Rb	000000000000	PC := PC + Rb
BRD Rb	100110	00000	10000	Rb	000000000000	PC := PC + Rb
BRLD Rd,Rb	100110	Rd	10100	Rb	000000000000	PC := PC + Rb Rd := PC
BRA Rb	100110	00000	01000	Rb	000000000000	PC := Rb
BRAD Rb	100110	00000	11000	Rb	000000000000	PC := Rb
BRALD Rd,Rb	100110	Rd	11100	Rb	000000000000	PC := Rb Rd := PC
BRK Rd,Rb	100110	Rd	01100	Rb	000000000000	PC := Rb Rd := PC MSR[BIP] := 1

表2-7: MicroBlaze 命令セットのまとめ(続き)

タイプ A	0 ~ 5	6 ~ 10	11 ~ 15	16 ~ 20	21 ~ 31	セマンティクス
タイプ B	0 ~ 5	6 ~ 10	11 ~ 15	16 ~ 31		
BEQ Ra,Rb	100111	0L000	Ra	Rb	000000000000	PC := PC + Rb if Ra = 0
BNE Ra,Rb	100111	0L001	Ra	Rb	000000000000	PC := PC + Rb if Ra != 0
BLT Ra,Rb	100111	0L010	Ra	Rb	000000000000	PC := PC + Rb if Ra < 0
BLE Ra,Rb	100111	0L011	Ra	Rb	000000000000	PC := PC + Rb if Ra <= 0
BGT Ra,Rb	100111	0L100	Ra	Rb	000000000000	PC := PC + Rb if Ra > 0
BGE Ra,Rb	100111	0L101	Ra	Rb	000000000000	PC := PC + Rb if Ra >= 0
BEQD Ra,Rb	100111	1L000	Ra	Rb	000000000000	PC := PC + Rb if Ra = 0
BNED Ra,Rb	100111	1L001	Ra	Rb	000000000000	PC := PC + Rb if Ra != 0
BLTD Ra,Rb	100111	1L010	Ra	Rb	000000000000	PC := PC + Rb if Ra < 0
BLED Ra,Rb	100111	1L011	Ra	Rb	000000000000	PC := PC + Rb if Ra <= 0
BGTD Ra,Rb	100111	1L100	Ra	Rb	000000000000	PC := PC + Rb if Ra > 0
BGED Ra,Rb	100111	1L101	Ra	Rb	000000000000	PC := PC + Rb if Ra >= 0
ORI Rd,Ra,Imm	101000	Rd	Ra	Imm		Rd := Ra or s(Imm)
ANDI Rd,Ra,Imm	101001	Rd	Ra	Imm		Rd := Ra and s(Imm)
XORI Rd,Ra,Imm	101010	Rd	Ra	Imm		Rd := Ra xor s(Imm)
ANDNI Rd,Ra,Imm	101011	Rd	Ra	Imm		Rd := Ra and $\overline{s(Imm)}$
IMM Imm	101100	00000	00000	Imm		Imm[0:15] := Imm
IMML Imm24 ²	101100	10	Imm24			Imm[24:47] := Imm24
RTSD Ra,Imm	101101	10000	Ra	Imm		PC := Ra + s(Imm)
RTID Ra,Imm	101101	10001	Ra	Imm		PC := Ra + s(Imm) MSR[IE] := 1
RTBD Ra,Imm	101101	10010	Ra	Imm		PC := Ra + s(Imm) MSR[BIP] := 0
RTED Ra,Imm	101101	10100	Ra	Imm		PC := Ra + s(Imm) MSR[EE] := 1、MSR[EIP] := 0 ESR := 0
BRI Imm	101110	00000	00000	Imm		PC := PC + s(Imm)
MBAR Imm	101110	Imm	00010	0000000000000100		PC := PC + 4、メモリ アクセスを待機。
BRID Imm	101110	00000	10000	Imm		PC := PC + s(Imm)
BRLID Rd,Imm	101110	Rd	10100	Imm		PC := PC + s(Imm) Rd := PC
BRAI Imm	101110	00000	01000	Imm		PC := s(Imm)

表 2-7: MicroBlaze 命令セットのまとめ(続き)

タイプ A	0 ~ 5	6 ~ 10	11 ~ 15	16 ~ 20	21 ~ 31	セマンティクス
タイプ B	0 ~ 5	6 ~ 10	11 ~ 15	16 ~ 31		
BRAID Imm	101110	00000	11000	Imm		PC := s(Imm) Rd := PC
BRALID Rd,Imm	101110	Rd	11100	Imm		PC := s(Imm) Rd := PC
BRKI Rd,Imm	101110	Rd	01100	Imm		PC := s(Imm) Rd := PC MSR[BIP] := 1
BEQI Ra,Imm	101111	0L000	Ra	Imm		PC := PC + s(Imm) if Ra = 0
BNEI Ra,Imm	101111	0L001	Ra	Imm		PC := PC + s(Imm) if Ra != 0
BLTI Ra,Imm	101111	0L010	Ra	Imm		PC := PC + s(Imm) if Ra < 0
BLEI Ra,Imm	101111	0L011	Ra	Imm		PC := PC + s(Imm) if Ra <= 0
BGTI Ra,Imm	101111	0L100	Ra	Imm		PC := PC + s(Imm) if Ra > 0
BGEI Ra,Imm	101111	0L101	Ra	Imm		PC := PC + s(Imm) if Ra >= 0
BEQID Ra,Imm	101111	1L000	Ra	Imm		PC := PC + s(Imm) if Ra = 0
BNEID Ra,Imm	101111	1L001	Ra	Imm		PC := PC + s(Imm) if Ra != 0
BLTID Ra,Imm	101111	1L010	Ra	Imm		PC := PC + s(Imm) if Ra < 0
BLEID Ra,Imm	101111	1L011	Ra	Imm		PC := PC + s(Imm) if Ra <= 0
BGTID Ra,Imm	101111	1L100	Ra	Imm		PC := PC + s(Imm) if Ra > 0
BGEID Ra,Imm	101111	1L101	Ra	Imm		PC := PC + s(Imm) if Ra >= 0
LBU Rd,Ra,Rb LBUR Rd,Ra,Rb	110000	Rd	Ra	Rb	000000000000 010000000000	Addr := Ra + Rb Rd[0:23] := 0 Rd[24:31] := *Addr[0:7]
LBUEA Rd,Ra,Rb	110000	Rd	Ra	Rb	000100000000	Addr := Ra & Rb Rd[0:23] := 0 Rd[24:31] := *Addr[0:7]
LHU Rd,Ra,Rb LHUR Rd,Ra,Rb	110001	Rd	Ra	Rb	000000000000 010000000000	Addr := Ra + Rb Rd[0:15] := 0 Rd[16:31] := *Addr[0:15]
LHUEA Rd,Ra,Rb	110001	Rd	Ra	Rb	000100000000	Addr := Ra & Rb Rd[0:15] := 0 Rd[16:31] := *Addr[0:15]
LW Rd,Ra,Rb LWR Rd,Ra,Rb	110010	Rd	Ra	Rb	000000000000 010000000000	Addr := Ra + Rb Rd := *Addr

表 2-7: MicroBlaze 命令セットのまとめ(続き)

タイプ A	0 ~ 5	6 ~ 10	11 ~ 15	16 ~ 20	21 ~ 31	セマンティクス
タイプ B	0 ~ 5	6 ~ 10	11 ~ 15	16 ~ 31		
LWX Rd,Ra,Rb	110010	Rd	Ra	Rb	10000000000	Addr := Ra + Rb Rd := *Addr Reservation := 1
LWEA Rd,Ra,Rb	110010	Rd	Ra	Rb	00010000000	Addr := Ra & Rb Rd := *Addr
LL Rd,Ra,Rb ² LLR Rd,Ra,Rb ²	110010	Rd	Ra	Rb	00100000000 01100000000	Addr := Ra[0:63] + Rb[0:63] Rd[0:63] := *Addr[0:63]
SB Rd,Ra,Rb SBR Rd,Ra,Rb	110100	Rd	Ra	Rb	00000000000 01000000000	Addr := Ra + Rb *Addr[0:8] := Rd[24:31]
SBEA Rd,Ra,Rb	110100	Rd	Ra	Rb	00010000000	Addr := Ra & Rb *Addr[0:8] := Rd[24:31]
SH Rd,Ra,Rb SHR Rd,Ra,Rb	110101	Rd	Ra	Rb	00000000000 01000000000	Addr := Ra + Rb *Addr[0:16] := Rd[16:31]
SHEA Rd,Ra,Rb	110101	Rd	Ra	Rb	00010000000	Addr := Ra & Rb *Addr[0:16] := Rd[16:31]
SW Rd,Ra,Rb SWR Rd,Ra,Rb	110110	Rd	Ra	Rb	00000000000 01000000000	Addr := Ra + Rb *Addr := Rd
SWX Rd,Ra,Rb	110110	Rd	Ra	Rb	10000000000	Addr := Ra + Rb *Addr := Rd if Reservation = 1 Reservation := 0
SWEA Rd,Ra,Rb	110110	Rd	Ra	Rb	00010000000	Addr := Ra & Rb *Addr := Rd
SL Rd,Ra,Rb ² SLR Rd,Ra,Rb ²	110110	Rd	Ra	Rb	00100000000 01100000000	Addr := Ra[0:63] + Rb[0:63] *Addr[0:63] := Rd[0:63]
LBUI Rd,Ra,Imm	111000	Rd	Ra	Imm		Addr := Ra + s(Imm) Rd[0:23] := 0 Rd[24:31] := *Addr[0:7]
LHUI Rd,Ra,Imm	111001	Rd	Ra	Imm		Addr := Ra + s(Imm) Rd[0:15] := 0 Rd[16:31] := *Addr[0:15]
LWI Rd,Ra,Imm	111010	Rd	Ra	Imm		Addr := Ra + s(Imm) Rd := *Addr
LLI Rd,Ra,Imm ²	111011	Rd	Ra	Imm		Addr := Ra[0:63] + s(Imm) Rd[0:63] := *Addr[0:63]

表 2-7: MicroBlaze 命令セットのまとめ(続き)

タイプ A	0 ~ 5	6 ~ 10	11 ~ 15	16 ~ 20	21 ~ 31	セマンティクス
タイプ B	0 ~ 5	6 ~ 10	11 ~ 15	16 ~ 31		
SBI Rd,Ra,Imm	111100	Rd	Ra	Imm		Addr := Ra + s(Imm) *Addr[0:7] := Rd[24:31]
SHI Rd,Ra,Imm	111101	Rd	Ra	Imm		Addr := Ra + s(Imm) *Addr[0:15] := Rd[16:31]
SWI Rd,Ra,Imm	111110	Rd	Ra	Imm		Addr := Ra + s(Imm) *Addr := Rd
SLI Rd,Ra,Imm ²	111111	Rd	Ra	Imm		Addr := Ra[0:63] + s(Imm) *Addr[0:63] := Rd[0:63]

1. 浮動小数点演算には多くのコーナー ケースが関与しているため、標準動作のみを説明しています。この動作の詳細は、[第5章の「MicroBlaze 命令セット アーキテクチャ」](#)を参照してください。

2. 64 ビット MicroBlaze でのみ使用可能。

セマフォ同期化

LWX および SWX 命令は、テストとセット、比較とスワップ、メモリ交換、フェッチと追加など、共通セマフォ操作をインプリメントするために使用されます。また、スピルロックをインプリメントするのにも使用されます。

これらの命令は、通常、システムプログラムによって使用され、適宜アプリケーションプログラムによって呼び出されます。

一般的には、プログラムは、メモリからセマフォをロードするのに LWX を使用し、予約が設定されます(プロセッサは予約を内部で維持)。プログラムは、セマフォ値に基づいて結果を計算し、SWX 命令を使用して、同じメモリロケーションにその結果を条件付きでストアします。この条件付きストアは、その前の LWX 命令によって設定された予約に基づいて、実行されます。ストア実行時に予約が存在する場合は、ストアが実行され、MSR[C] は 0 にクリアされます。ストア実行時に予約が存在しない場合は、ストアが実行され、MSR[C] は 1 にセットされます。

ストアが問題なく実行されると、セマフォのロードからセマフォのストアまで、順番に命令が、ほかの命令を差し挟むことなく、実行されます。ほかのデバイスが、読み出しからアップデートまでのセマフォ ロケーションを変更することはありません。この操作中、ほかのデバイスはこのセマフォ ロケーションから読み出しを実行できます。

セマフォ操作が正しく実行には、LWX 命令を SWX 命令とペアにし、両方が同じアドレスを指定する必要があります。

MicroBlaze での予約単位はワードです。両方の命令に対し、アドレスはワード アラインしている必要があります。アラインされていない例外は、これらの命令に対しては生成されません。

条件付きストアは、ストアアドレスが、予約を設定したロードアドレスに一致していないなくても、予約が存在していると常に実行されます。

1回につき、1 予約のみが維持されます。予約に関連付けられているアドレスは、後続の LWX 命令を実行することで、変更できます。

この条件付きストアは、その前の LWX 命令によって設定された予約に基づいて、実行されます。LWX で設定されたアドレスに一致するかどうかにかかわらず、プロセッサで保持されている予約は、SWX 命令を実行すると常にクリアになります。

リセット、割り込み、例外、ブレーク(BRK および BRKI 命令を含む)はすべて予約をクリアします。

LWX および SWX 命令の使用に関する一般ガイドラインは、次のようになっています。

- LWX および SWX 命令はペアにし、同じアドレスを使用するようにします。
- LWX とペアにならない SWX 命令で、任意アドレスに割り当てられているものは、プロセッサで保持されている予約をクリアするのに使用します。
- 条件付きシーケンスは LWX 命令から始まります。その後に、メモリアクセスが続くか、またはロードされた値に基づいた計算が続きます。シーケンスは SWX 命令で終わります。一般的には、SWX 命令にエラーがあると、分岐は LWX に戻って、繰り返し命令が実行されます。
- LWX によってロードされた値がゼロでなければ、一部の同期化プリミティブを実行するとき、その LWX は SWX とペアにしなくてもよい場合があります。次は、テスト アンド セットのインプリメンテーション例です。

```
loop: lwx    r5,r3,r0 ; load and reserve
      bnei   r5,next   ; branch if not equal to zero
      addik  r5,r5,1   ; increment value
      swx    r5,r3,r0 ; try to store non-zero value
      addic  r5,r0,0   ; check reservation
      bnei   r5,loop    ; loop if reservation lost
next:
```

- パフォーマンスは、目的の値を返すことができない LWX 命令でのループを最小限にすることで、改善できます。また、初期値チェックをする普通のロード命令を使用することで、パフォーマンスを改善することもできます。次は、スピルロックのインプリメンテーション例です。

```
loop: lw     r5,r3,r0 ; load the word
      bnei   r5,loop    ; loop back if word not equal to 0
      lwx    r5,r3,r0 ; try reserving again
      bnei   r5,loop    ; likely that no branch is needed
      addik  r5,r5,1   ; increment value
      swx    r5,r3,r0 ; try to store non-zero value
      addic  r5,r0,0   ; check reservation
      bnei   r5,loop    ; loop if reservation lost
```

- LWX/SWX 命令ペアでのループを最小限にすると、前進する可能性が高くなります。古い値は、ストアを実行する前にテストする必要があります。順番が逆の場合(ストアの後にロードを実行)、実行される SWX 命令の数が多くなり、LWX と SWX との間で、予約が失われる可能性が高くなります。

自己変更コード

自己変更コードを使用する場合は、変更された命令を実行する前にフェッチする前に、メモリにその命令が書き込まれていることをソフトウェアで確認する必要があります。次の点に注意する必要があります。

- 変更される命令が、変更前に既に、次の場所にフェッチされている可能性がある。
 - 命令プリフェッчバッファー
 - 命令キャッシュ(イネーブルになっている場合)
 - ストリームバッファー(命令キャッシュストリームバッファーが使用されている場合)
 - 命令キャッシュ。その後、ビクティムバッファーが使用されている場合はそこに保存される。

古い未変更のコードではなく、変更されたコードが常に実行されていることを確認するには、ソフトウェアはこれらのすべてのケースを処理する必要があります。

- 変更する命令が1つ以上分岐で、分岐先キャッシュが使用されている場合は、分岐先アドレスがキャッシュされている可能性があります。

キャッシュされた分岐先アドレスの使用を避けるには、変更されたコードを実行する前に、分岐先キャッシュがクリアになっていることをソフトウェアで確認する必要があります。

- 変更された命令は、実行前にメモリに書き込まれていない可能性があります。
 - 変更された命令はメモリに書き込まれる手前で、インターフェクトまたはメモリコントローラーの一時的なストレージにある可能性があります。
 - ライトバックキャッシュが使用されている場合は、変更された命令はデータキャッシュに格納されている可能性があります。
 - ライトバックキャッシュおよびビクティムバッファーが使用されている場合は、変更された命令はビクティムバッファーに格納されている可能性があります。

プロセッサがフェッチする前に、変更された命令がメモリに書き込まれていることをソフトウェアで確認する必要があります。

次の注釈付きコードは、上記の問題のそれぞれがどのように解決されるかを示しています。このコードは、命令キャッシュおよびライトバックデータキャッシュの両方が使用されていることを前提にしています。使用されていない場合は、その部分の命令は省くことができます。

次のコード例は、変更された命令を格納します。

```
swi      r5,r6,0 ; r5 = new instruction
              ; r6 = physical instruction address
wdc.flush r6,r0 ; flush write-back data cache line
mbar     1       ; ensure new instruction is written to memory
wic      r7,r0   ; invalidate line, empty stream & victim buffers
              ; r7 = virtual instruction address
mbar     2       ; empty prefetch buffer, clear branch target cache
```

MMU仮想モードが使用されていない限り、上記の物理および仮想アドレスは同じです。MMUがイネーブルになっている場合は、WICおよびWDCは特権命令であるため、コードシーケンスをリアルモードで実行する必要があります。上記のコードシーケンスの後の最初の命令はプリフェッчされている可能性があるため、変更しないでください。

レジスタ

MicroBlaze は直交命令セットアーキテクチャを持ちます。設定オプションによって、32 個の 32 ビットまたは 64 ビット汎用レジスタと、16 個までの特殊レジスタがあります。

汎用レジスタ

32 個の 32 ビットまたは 64 ビットの汎用レジスタには、R0 ~ R31 の番号が付いています。レジスタ ファイルは、ビットストリームをダウンロードするとリセットされます(リセット値は 0x00000000)。次の図に汎用レジスタを図示し、表 2-8 に各レジスタの説明とリセット値を示します。

汎用レジスタは、64 ビット MicroBlaze がイネーブルの場合 (`C_DATA_SIZE = 64`) は 64 ビットで、それ以外の場合は 32 ビットです。

注記: レジスタ ファイルは、外部リセット入力 (Reset および Debug_Rst) ではリセットされません。

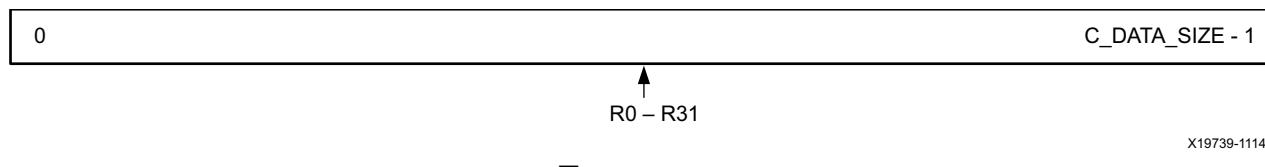


図 2-2: R0 ~ R31

X19739-111417

表 2-8: 汎用レジスタ (R0 ~ R31)

ビット ¹	名前	説明	リセット値
0:31	R0	値は常に 0 です。R0 に書き込まれた値は破棄されます。	0x0
0:63	R1 ~ R13	汎用レジスタ	-
	R14	割り込み用に戻りアドレスを格納するために使用されるレジスタ。	-
	R15	汎用レジスタ。ユーザー ベクター用の戻りアドレスを格納するために推奨。	-
	R16	ブレーク用に戻りアドレスを格納するために使用されるレジスタ。	-
	R17	MicroBlaze がハードウェア例外をサポートするように設定されている場合は、このレジスタに命令のアドレスがロードされ、命令の後にハードウェア例外が発生します。ただし、BTR を使用する遅延スロットの例外を除きます(「 分岐先レジスタ (BTR) 」を参照)。ハードウェア例外をサポートするように設定されていない場合、このレジスタは汎用レジスタです。	-
	R18 ~ R31	汎用レジスタ。	-

1. 64 ビット MicroBlaze 32 ビット (`C_DATA_SIZE = 64`) では 64 ビット、それ以外の場合は 32 ビット

汎用レジスタのソフトウェア使用規則は、表 4-2 を参照してください。

特殊用途レジスタ

プログラム カウンター (PC)

プログラム カウンター (PC) は実行命令のアドレスです。MFS 命令で読み出すことができますが、MTS 命令で書き込むことはできません。MFS 命令を使用する際、PC レジスタは $Sa = 0x0000$ で指定します。次の図に PC を図示し、[表 2-9](#) にその説明とリセット値を示します。

プログラム カウンターは、64 ビット MicroBlaze がイネーブルの場合 ($C_DATA_SIZE = 64$) は C_ADDR_SIZE パラメーターの設定によって 64 ビットまで、それ以外の場合は 32 ビットです。

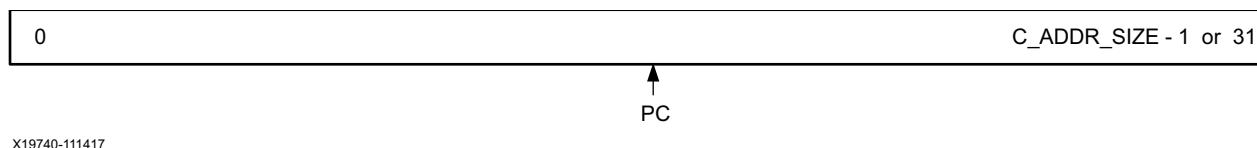


図 2-3: PC

表 2-9: プログラム カウンター (PC)

ビット ¹	名前	説明	リセット値
0:31	PC	プログラム カウンター	$C_BASE_VECTORS$
0: C_ADDR_SIZE-1		実行命令のアドレス。mfs r20 は MFS 命令のアドレスを R2 に格納します。	

1. 64 ビット MicroBlaze ($C_DATA_SIZE = 64$) では C_ADDR_SIZE ビット、それ以外の場合は 32 ビット。

マシンステータス レジスタ (MSR)

マシンステータス レジスタには、プロセッサの制御ビットおよびステータスビットが含まれており、MFS 命令で読み出すことができます。MSR を読み出すと、キャリービットはキャリーコピー ビットに複製されます。MSR への書き込みには、MTS 命令を使用するか、または専用 MSRSET および MSRCLR 命令を使用します。

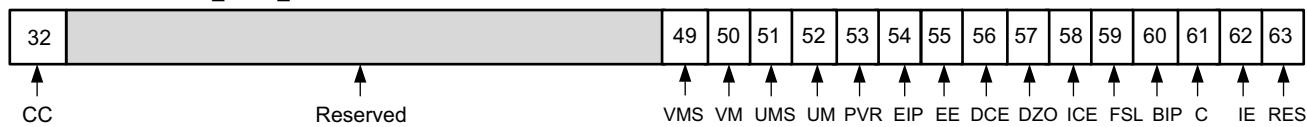
MSRSET または MSRCLR を使用して MSR に書き込む場合、キャリービットはすぐに適用され、残りのビットは 1 クロックサイクル後に適用されます。MTS を使用して書き込む場合は、すべてのビットが 1 クロックサイクル後に適用されます。キャリーコピー ビットに書き込まれた値は破棄されます。

MTS または MFS 命令を使用する場合は、MSR は $Sx = 0x0001$ で指定します。次の図に MSR レジスタを図示し、[表 2-10](#) にビットの説明とリセット値を示します。

32-bit MicroBlaze: C_DATA_SIZE = 32

0		17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
---	--	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----

64-bit MicroBlaze: C_DATA_SIZE = 64



X19741-111517

図 2-4: MSR

表 2-10: マシンステータスレジスタ (MSR)

ビット ¹	名前	説明	リセット値
0、32	CC	演算キャリー コピー 演算キャリーのコピー。CC はビット C と常に同じです。	0
1:16 2:48	予約		
17、49	VMS	仮想保護モード保存 MMU を設定した場合にのみ使用可能 (C_USE_MMU > 1 および C_AREA_OPTIMIZED = 0 または 2 の場合) 読み出し/書き込み	0
18、50	VM	仮想保護モード 0: C_USE_MMU = 3 (仮想) の場合は MMU アドレス変換およびアクセス保護がディスエーブル。C_USE_MMU = 2 (保護) の場合はアクセス保護がディスエーブル。 1: C_USE_MMU = 3 (仮想) の場合は MMU アドレス変換およびアクセス保護がイネーブル。C_USE_MMU = 2 (保護) の場合はアクセス保護がイネーブル。 MMU を設定した場合にのみ使用可能 (C_USE_MMU > 1 および C_AREA_OPTIMIZED = 0 または 2 の場合) 読み出し/書き込み	0
19、51	UMS	ユーザー モード保存 MMU を設定した場合にのみ使用可能 (C_USE_MMU > 0 および C_AREA_OPTIMIZED = 0 または 2 の場合) 読み出し/書き込み	0
20、52	UM	ユーザー モード 0: 特権モード、すべての命令を使用可能 1: ユーザー モード、一部の命令は使用不可 MMU を設定した場合にのみ使用可能 (C_USE_MMU > 0 および C_AREA_OPTIMIZED = 0 または 2 の場合) 読み出し/書き込み	0

表 2-10: マシンステータスレジスタ (MSR) (続き)

ビット ¹	名前	説明	リセット値
21、53	PVR	プロセッサバージョンレジスタの有無 0: プロセッサバージョンレジスタなし 1: プロセッサバージョンレジスタあり 読み出し専用	パラメータ C_PVR に基づく
22、54	EIP	処理中例外 0: 処理中ハードウェア例外なし 1: 処理中ハードウェア例外あり 例外サポートを有効にして設定している場合 (C_*_EXCEPTION または C_USE_MMU > 0) にのみ使用可能 読み出し/書き込み	0
23、55	EE	例外イネーブル 0: ハードウェア例外をディスエーブル ² 1: ハードウェア例外をイネーブル 例外サポートを有効にして設定している場合 (C_*_EXCEPTION または C_USE_MMU > 0) にのみ使用可能 読み出し/書き込み	0
24、56	DCE	データキャッシュイネーブル 0: データキャッシュをディスエーブル 1: データキャッシュをイネーブル データキャッシュを使用するように設定している場合 (C_USE_DCACHE = 1) にのみ使用可能 読み出し/書き込み	0
25、57	DZO	ゼロ除算、または除算オーバーフロー ³ 0: ゼロ除算または除算オーバーフローなし 1: ゼロ除算、または除算オーバーフローが発生 ハードウェア除算器を使用するように設定している場合 (C_USE_DIV = 1) にのみ使用可能 読み出し/書き込み	0
26、58	ICE	命令キャッシュイネーブル 0: 命令キャッシュをディスエーブル 1: 命令キャッシュをイネーブル 命令キャッシュを使用するように設定している場合 (C_USE_DCACHE = 1) にのみ使用可能 読み出し/書き込み	0

表 2-10: マシンステータスレジスタ (MSR) (続き)

ビット ¹	名前	説明	リセット値
27、59	FSL	AXI4-Stream エラー 0: get または getd にエラーなし 1: get または getd の制御タイプが一致しない このビットはステイッキービットで、制御ビットが一致しない場合に get または getd 命令で設定されます。これをクリアにするには、MTS または MSRCLR 命令を使用する必要があります。 ストリームリンクを使用するように設定している場合 (C_FSL_LINKS > 0) にのみ使用可能 読み出し/書き込み	0
28、60	BIP	処理中ブレーク 0: 処理中ブレークなし 1: 処理中ブレークあり ブレークソースはソフトウェアブレーク命令、あるいは Ext_Brk または Ext_NM_Brk ピンからのハードウェアブレーク。 読み出し/書き込み	0
29、61	C	演算キャリー 0: キャリーなし (繰り下げ) 1: キャリーあり (繰り上げ) 読み出し/書き込み	0
30、62	IE	割り込みイネーブル 0: 割り込みをディスエーブル 1: 割り込みをイネーブル 読み出し/書き込み	0
31、63	-	予約	0

- ビット番号は、64ビットMicroBlaze (C_DATA_SIZE = 64) がイネーブルかどうかによります。
- MMU例外(データストレージ例外、命令ストレージ例外、データTLBミス例外、命令TLBミス例外)はディスエーブルにできず、このビットの影響を受けません。
- このビットは、整数のゼロ除算または除算オーバーフローにのみ使用されます。FSRに浮動小数点用のものがあります。例外処理ができるように設定されているかどうかにかかわらず、DZOビットはゼロ除算または除算オーバーフロー コンディションをフラグします。

例外アドレス レジスタ (EAR)

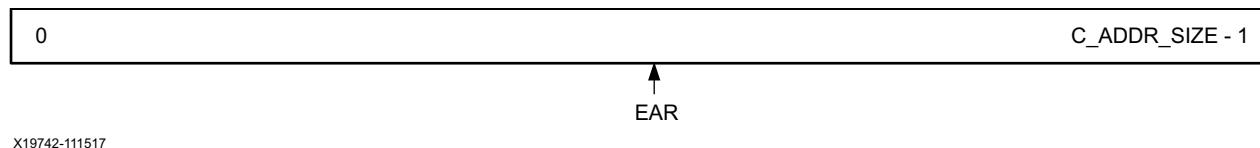
例外アドレス レジスタは、次の例外を発生させたロード/ストアのフルアドレスを格納します。

- 非境界整列アクセス データ アドレスを指定する非境界整列アクセス例外
- エラーの発生している AXI4 データ アクセス アドレスを指定する M_AXI_DP 例外
- アクセスされた (仮想) 有効アドレスを指定するデータ ストレージ例外
- 読み出された (仮想) 有効アドレスを指定する命令ストレージ例外
- アクセスされた (仮想) 有効アドレスを指定するデータ TLB ミス例外
- 読み出された (仮想) 有効アドレスを指定する命令 TLB ミス例外

上記以外の例外に対しては、このレジスタの内容は定義されていません。MFS または MFSE 命令で読み出す際、EAR は Sa = 0x0003 で指定します。次の図に EAR レジスタを図示し、表 2-11 にビットの説明とリセット値を示します。

32 ビット MicroBlaze (C_DATA_SIZE = 32) では、拡張データ アドレス指定がイネーブル (C_ADDR_SIZE > 32) の場合、レジスタの下位 32 ビットは MFS 命令で読み出し、上位ビットは MFSE 命令で読み出します。

64 ビット MicroBlaze (C_DATA_SIZE = 64) では、レジスタ全体を MFS 命令で読み出します。



X19742-111517

図 2-5: EAR

表 2-11: 例外アドレス レジスタ (EAR)

ビット	名前	説明	リセット値
0:C_ADDR_SIZE-1	EAR	例外アドレス レジスタ	0

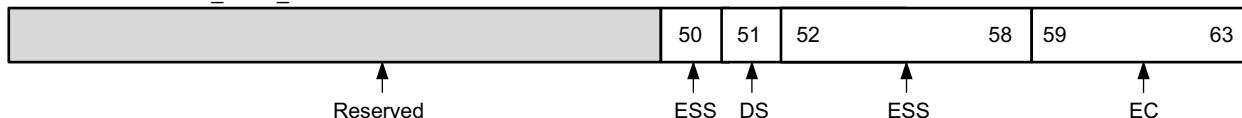
例外ステータス レジスタ (ESR)

例外ステータス レジスタには、プロセッサのステータス ビットが含まれています。MFS 命令で読み出す際、ESR は Sa = 0x0005 で指定します。次の図に ESR レジスタを図示し、表 2-12 にビットの説明とリセット値、表 2-13 に例外別ステータス (ESS) を示します。

32-bit MicroBlaze: C_DATA_SIZE = 32



64-bit MicroBlaze: C_DATA_SIZE = 64



X19743-111517

図 2-6: ESR

表 2-12: 例外ステータス レジスタ (ESR)

ビット ¹	名前	説明	リセット値
0:17 0:49	予約		
-、50	ESS	例外別ステータス。64 ビット MicroBlaze (C_DATA_SIZE = 64) でのみ使用可能、それ以外の場合は予約。 詳細は表 2-13 を参照。 読み出し専用	表 2-13 を参照
19、51	DS	遅延スロット例外。 0: 遅延スロット命令により発生していない 1: 遅延スロット命令により発生 読み出し専用	0

表 2-12: 例外ステータス レジスタ (ESR) (続き)

ビット ¹	名前	説明	リセット値
20:26 52:58	ESS	例外別ステータス 詳細は表 2-13 を参照。 読み出し専用	表 2-13 を参照
27:31 59:63	EC	例外の原因 00000: ストリーム例外 00001: 非境界整列データ アクセス例外 00010: 無効なオペコード例外 00011: 命令バス エラー例外 00100: データ バス エラー例外 00101: 除算例外 00110: 浮動小数点ユニット例外 00111: 特権命令例外 00111: スタック保護違反例外 10000: データ ストレージ例外 10001: 命令ストレージ例外 10010: データ TLB ミス例外 10011: 命令 TLB ミス例外 読み出し専用	0

1. ビット番号は、64 ビット MicroBlaze (C_DATA_SIZE = 64) がイネーブルかどうかによります。

表 2-13: 例外別ステータス (ESS)

例外の原因	ビット ¹	名前	説明	リセット値
非境界整列 データ アクセス	-、50	L	long アクセス例外 0: 非境界整列ワードまたはハーフワード アクセス 1: 非境界整列 long アクセス	0
	20、52	W	ワード アクセス例外 0: 非境界整列ハーフワード アクセス 1: 非境界整列ワード アクセス	0
	21、53	S	ストア アクセス例外 0: 非境界整列ロード アクセス 1: 非境界整列ストア アクセス	0
	22:26 54:58	Rx	ソース/デスティネーション レジスタ 非境界整列アクセスでソース(ストア) またはデスティネーション(ロード) として使用される汎用レジスタ	0
無効な命令	20:26 52:58	予約		0
命令バス エラー	20、52	ECC	ILMB の訂正可能エラーまたは訂正不可能なエラーによる例外	0
	21:26 53:58	予約		0

表 2-13: 例外別ステータス (ESS) (続き)

例外の原因	ビット ¹	名前	説明	リセット値
データバスエラー	20、52	ECC	DLMB の訂正可能エラーまたは訂正不可能なエラーによる例外	0
	21:26 53:58	予約		0
除算	20、52	DEC	除算 - 除算例外の原因 0: ゼロ除算 1: 除算オーバーフロー	0
	21:26 53:58	予約		0
浮動小数点ユニット	20:26 52:58	予約		0
特権命令	20:26 52:58	予約		0
スタック保護違反	20:26 52:58	予約		0
ストリーム	20:22 52:54	予約		0
	23:26 55:58	FSL	例外を発生させた AXI4-Stream のインデックス	0
データストレージ	20、52	DIZ	データストレージ - ゾーン保護 0: 発生していない 1: 発生	0
	21、53	S	データストレージ - ストア命令 0: 発生していない 1: 発生	0
	22:26 54:58	予約		0
命令ストレージ	20、52	DIZ	命令ストレージ - ゾーン保護 0: 発生していない 1: 発生	0
	21:26 53:58	予約		0
データ TLB ミス	20、52	予約		0
	21、53	S	データ TLB ミス - ストア命令 0: 発生していない 1: 発生	0
	22:26 54:58	予約		0
命令 TLB ミス	20:26 52:58	予約		0

1. ビット番号は、64 ビット MicroBlaze (C_DATA_SIZE = 64) がイネーブルかどうかによります。

分岐先レジスタ (BTR)

分岐先レジスタは、MicroBlaze プロセッサが例外を使用するように設定されている場合にのみ存在します。このレジスタは、MSR[EIP] = 0 のときに実行されたすべての遅延スロット分岐命令の分岐先アドレスを格納します。例外が遅延スロットの命令によって発生した場合 (ESR[DS]=1)、例外ハンドラーは、R17 に格納されている標準例外戻りアドレスではなく、BTR に格納されているアドレスに実行を戻す必要があります。MFS 命令で読み出す際、BTR は Sa = 0x000B で指定します。次の図に BTR レジスタを図示し、表 2-14 にビットの説明とリセット値を示します。

分岐先レジスタは、64 ビット MicroBlaze がイネーブルの場合 (C_DATA_SIZE = 64) は C_ADDR_SIZE パラメーターの設定によって 64 ビットまで、それ以外の場合は 32 ビットです。

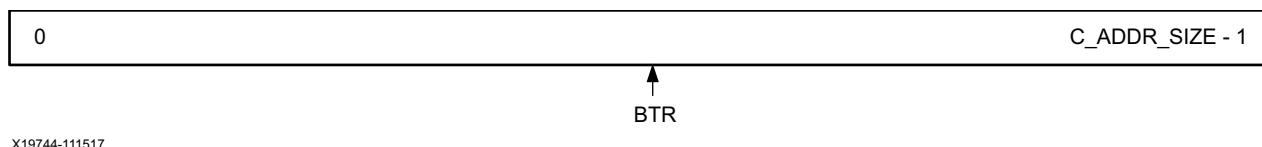


図 2-7: BTR

表 2-14: 分岐先レジスタ (BTR)

ビット ¹	名前	説明	リセット値
0:31 0:C_ADDR_SIZE-1	BTR	遅延スロットの例外によって発生した例外から戻るときにハンドラーが使用する分岐先アドレス。 読み出し専用	0x0

1. 64 ビット MicroBlaze (C_DATA_SIZE = 64) では C_ADDR_SIZE ビット、それ以外の場合は 32 ビット。

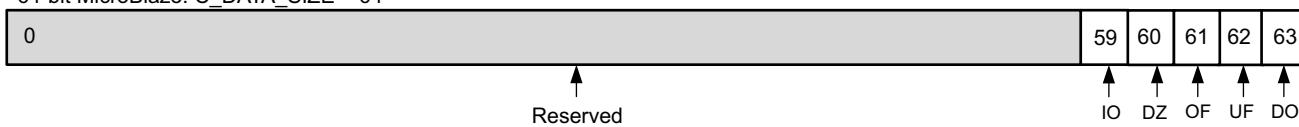
浮動小数点ステータス レジスタ (FSR)

浮動小数点ステータス レジスタには、浮動小数点ユニットのステータス ビットが含まれています。MFS 命令で読み出し、MTS 命令で書き込みます。読み出しちゃは書き込みを実行する際、このレジスタは Sa = 0x0007 で指定します。このレジスタのビットはステッキで、浮動小数点命令でのみ設定可能です。レジスタをクリアするには MTS 命令を使用する必要があります。次の図に FSR レジスタを図示し、表 2-15 にビットの説明とリセット値を示します。

32-bit MicroBlaze: C_DATA_SIZE = 32



64-bit MicroBlaze: C_DATA_SIZE = 64



X19745-111517

図 2-8: FSR

表 2-15: 浮動小数点ステータス レジスタ (FSR)

ビット ¹	名前	説明	リセット値
0:26 0:58	予約		未定義
27、59	IO	無効な操作	0
28、60	DZ	ゼロ除算	0
29、61	OF	オーバーフロー	0
30、62	UF	アンダーフロー	0
31、63	DO	非正規オペランド エラー	0

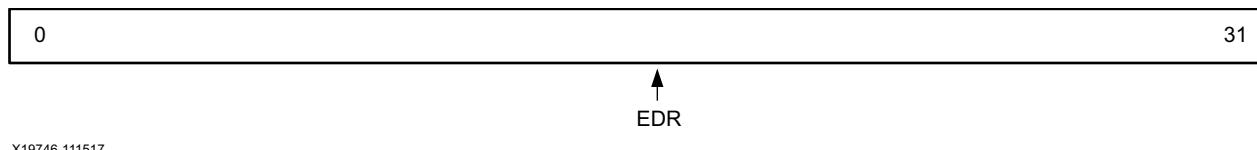
1. ビット番号は、64 ビット MicroBlaze (C_DATA_SIZE = 64) がイネーブルかどうかによります。

例外データ レジスタ (EDR)

例外データ レジスタは、ストリーム例外を発生させた AXI4-Stream リンクで読み出されたデータを格納します。

上記以外の例外に対しては、このレジスタの内容は定義されていません。MFS 命令で読み出す際、EDR は Sa = 0x000D で指定します。次の図に EDR レジスタを図示し、表 2-16 にビットの説明とリセット値を示します。

注記: このレジスタは、C_FSL_LINKS が 0 より大きく、C_FSL_EXCEPTION が 1 に設定されている場合にのみインプリメントされます。



X19746-111517

図 2-9: EDR

表 2-16: 例外データ レジスタ (EDR)

ビット	名前	説明	リセット値
0:31	EDR	例外データ レジスタ	0x00000000

スタック ロー レジスタ (SLR)

スタック ロー レジスタは、スタック オーバーフローを検出するためのスタックの最小値を格納します。rA としてスタック ポインター(レジスタ R1)を使用するロードまたはストア命令のアドレスがスタック ロー レジスタより小さい場合、スタック オーバーフローが発生し、MSR でスタック保護違反例外がイネーブルになっている場合はスタック保護違反例外が発生します。

MFS 命令で読み出す際、SLR は Sa = 0x0800 で指定します。図 2-10 に SLR レジスタを図示し、表 2-17 にビットの説明およびリセット値を示します。

スタック ロー レジスタは、64 ビット MicroBlaze がイネーブルの場合 (C_DATA_SIZE = 64) は C_ADDR_SIZE パラメーターの設定によって 64 ビットまで、それ以外の場合は 32 ビットです。

注記: このレジスタは、パラメーター C_USE_STACK_PROTECTION が 1 に設定されていてスタック保護がイネーブルになっている場合にのみインプリメントされます。スタック保護がインプリメントされていない場合は、このレジスタに書き込みを実行しても無視されます。

注記: MMU がイネーブルの場合 ($C_USE_MMU > 0$) はスタック保護は使用できません。MMU ページベースのメモリは UTLB を使用して保護されます。

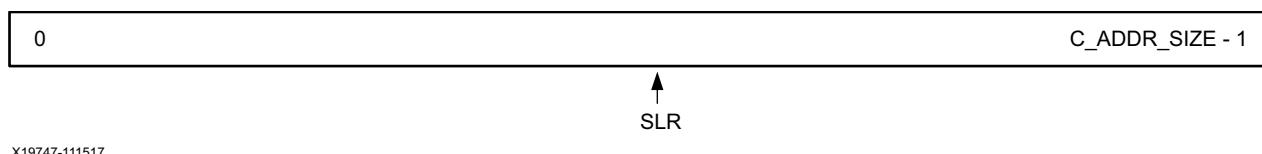


図 2-10: SLR

表 2-17: スタック ローレジスタ (SLR)

ビット ¹	名前	説明	リセット値
0:31	SLR	スタック ローレジスタ	0x0
0:C_ADDR_SIZE-1			

1. 64 ビット MicroBlaze ($C_DATA_SIZE = 64$) では C_ADDR_SIZE ビット、それ以外の場合は 32 ビット。

スタック ハイ レジスタ (SHR)

スタック ハイ レジスタは、スタック アンダーフローを検出するためのスタックの最大値を格納します。rA としてスタック ポインター(レジスタ R1)を使用するロードまたはストア命令のアドレスが、スタック ハイ レジスタより大きい場合、スタック アンダーフローが発生し、MSR でスタック 保護違反例外がイネーブルになっている場合はスタック 保護違反例外が発生します。

MFS 命令で読み出す際、SHR は Sa = 0x0802 で指定します。次の図に SHR レジスタを図示し、表 2-18 にビットの説明とリセット値を示します。

スタック ハイ レジスタは、64 ビット MicroBlaze がイネーブルの場合 ($C_DATA_SIZE = 64$) は C_ADDR_SIZE パラメーターの設定によって 64 ビットまで、それ以外の場合は 32 ビットです。

注記: このレジスタは、パラメーター $C_USE_STACK_PROTECTION$ が 1 に設定されていてスタック 保護がイネーブルになっている場合にのみインプリメントされます。スタック 保護がインプリメントされていない場合は、このレジスタに書き込みを実行しても無視されます。

注記: MMU がイネーブルの場合 ($C_USE_MMU > 0$) はスタック 保護は使用できません。MMU ページベースのメモリは UTLB を使用して保護されます。

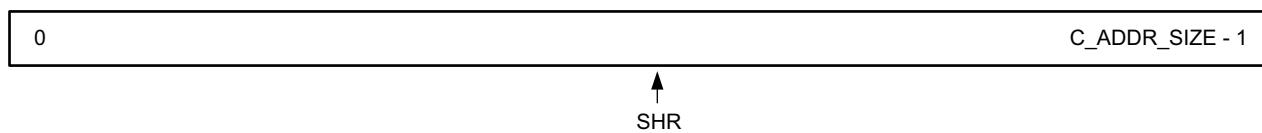


図 2-11: SHR

表 2-18: スタック ハイ レジスタ (SHR)

ビット ¹	名前	説明	リセット値
0:31	SHR	スタック ハイ レジスタ	すべてのビットを 0 に設定
0:C_ADDR_SIZE-1			

1. 64 ビット MicroBlaze ($C_DATA_SIZE = 64$) では C_ADDR_SIZE ビット、それ以外の場合は 32 ビット。

プロセス ID レジスタ (PID)

プロセス ID レジスタは、MMU アドレス変換中のソフトウェアプロセスを特定します。MicroBlaze の C_USE_MMU コンフィギュレーションオプションで設定されます。このレジスタは、C_USE_MMU が 1 より大きく(ユーザー モード)、C_AREA_OPTIMIZED が 0(パフォーマンス) または 2(周波数) に設定されている場合にのみインプリメントされます。

MFS および MTS 命令でアクセスする際、PID は $Sa = 0x1000$ で指定します。このレジスタには、メモリ管理特殊レジスタのパラメーター C_MMU_TLB_ACCESS の設定に従ってアクセスできます。

PID は TLB エントリにアクセスするときにも使用されます。

- TLBHI(変換ルックアサイド バッファーハイ)を書き込むと、PID の値が TLB エントリの TID フィールドに格納されます。
- TLBHI を読み出すときに MSR[UM] がセットされていない場合は、TID フィールドの値が PID に格納されます。

次の図に PID レジスタを図示し、表 2-19 にビットの説明とリセット値を示します。

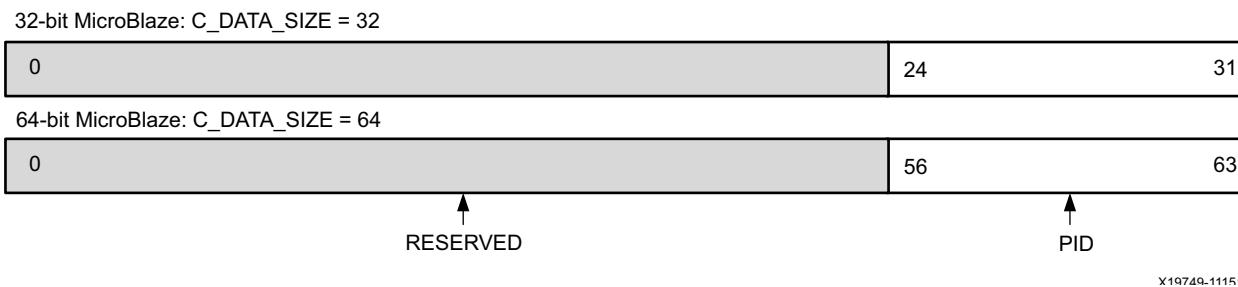


図 2-12: PID

X1974Q-111517

表 2-19: プロセス ID レジスタ (PID)

ビット ¹	名前	説明	リセット値
0:23 0:55	予約		
24:31 56:63	PID	MMU アドレス変換中のソフトウェアプロセスを特定するために使用されます。 読み出し/書き込み	0x00

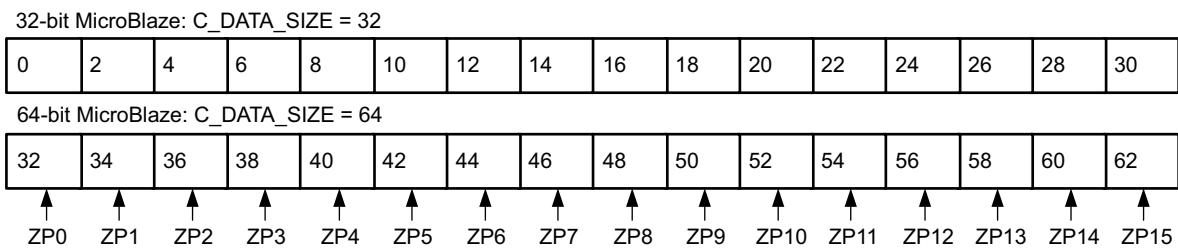
- ビット番号は、64 ビット MicroBlaze ($C_DATA_SIZE = 64$) がイネーブルかどうかによります。

ゾーン保護レジスタ (ZPR)

ゾーン保護レジスタは、TLB エントリで定義されている MMU メモリ保護を上書きするために使用されます。

MicroBlaze の C_USE_MMU コンフィギュレーションオプションで設定されます。このレジスタは、C_USE_MMU が 1 よりも大きく(ユーザー モード)、C_AREA_OPTIMIZED が 0(パフォーマンス)または 2(周波数)に設定されていて、さらに指定されているメモリ保護ゾーンの数が 0 よりも大きい(C_MMU_ZONES > 0)場合にのみインプリメントされます。インプリメントされるレジスタビット数は、指定されているメモリ保護ゾーンの数(C_MMU_ZONES)によって異なります。MFS および MTS 命令でアクセスする際、ZPR は Sa = 0x1001 で指定します。このレジスタには、メモリ管理特殊レジスタのパラメーター C_MMU_TLB_ACCESS の設定に従ってアクセスできます。

次の図に ZPR レジスタを図示し、表 2-20 にビットの説明とリセット値を示します。



X19750-111517

図 2-13: ZPR

表 2-20: ゾーン保護レジスタ (ZPR)

ビット ¹	名前	説明	リセット値
0:1	ZP0	ゾーン保護	0x0
2:3	ZP1	<u>ユーザー モード (MSR[UM] = 1):</u>	
...	...	00: TLB エントリで V を上書き。ページへのアクセスは不可。	
30:31	ZP15	01: 上書きなし。TLB エントリの V、WR、EX を使用。 10: 上書きなし。TLB エントリの V、WR、EX を使用。 11: TLB エントリで WR および EX を上書き。書き込み可能および実行可能な権限でページにアクセス。	
32:33	ZP0	<u>特権モード (MSR[UM] = 0):</u>	
34:35	ZP1	00: 上書きなし。TLB エントリの V、WR、EX を使用。 01: 上書きなし。TLB エントリの V、WR、EX を使用。	
...	...		
62:63	ZP15	10: TLB エントリで WR および EX を上書き。書き込み可能および実行可能な権限でページにアクセス。 11: TLB エントリで WR および EX を上書き。書き込み可能および実行可能な権限でページにアクセス。 読み出し/書き込み	

1. ビット番号は、64 ビット MicroBlaze (C_DATA_SIZE = 64) がイネーブルかどうかによります。

変換ルックアサイド バッファー ロー レジスタ (TLBLO)

変換ルックアサイド バッファー ロー レジスタは、MMU の統合変換ルックアサイド バッファー (UTLB) エントリにアクセスするために使用されます。MicroBlaze の C_USE_MMU コンフィギュレーションオプションで設定されます。このレジスタは、C_USE_MMU が 1 より大きく (ユーザー モード)、C_AREA_OPTIMIZED が 0 (パフォーマンス) または 2 (周波数) に設定されている場合にのみインプリメントされます。MFS および MTS 命令でアクセスする際、TLBLO は Sa = 0x1003 で指定します。

TLBLO に対して読み出しありは書き込みを実行するときは、TLBX レジスタでインデックス化された UTLB エントリにアクセスします。このレジスタは、メモリ管理特殊レジスタのパラメーター C_MMU_TLB_ACCESS の設定に従って読み出し可能です。

MMU 物理アドレス拡張 (PAE) がイネーブルの場合 (C_DATA_SIZE = 32、C_USE_MMU = 3、および C_ADDR_SIZE > 32)、TLBLO の下位 32 ビットにアクセスするには MFS および MTS 命令を使用し、上位ビットにアクセスするには MFSE および MTSE 命令を使用します。PAE をイネーブルにしてレジスタに書き込む場合は、最上位ビットを最初に書き込む必要があります。

64 ビット MicroBlaze (C_DATA_SIZE = 64) では、レジスタ全体を MFS 命令で読み出します。

UTLB はビットストリームをダウンロードするとリセットされます (すべての TLBLO エントリのリセット値は 0x00000000)。

注記: UTLB は、外部リセット入力 (Reset および Debug_Rst) ではリセットされません。つまり、データの陳腐化を避けるため、リセット後に UTLB 全体を初期化する必要があります。

次の図に TLBLO レジスタを図示し、表 2-21 にビットの説明とリセット値を示します。PAE がイネーブルの場合、C_ADDR_SIZE パラメーターに基づいてレジスタの RPN フィールドが最大 54 ビットまで拡張され、64 ビットまでの物理アドレスを保持できるようになります。

32-bit MicroBlaze: (C_ADDR_SIZE = 32 or C_USE_MMU ≠ 3) and (C_DATA_SIZE = 32):

0	22	23	24	28	29	30	31
---	----	----	----	----	----	----	----

PAE or 64-bit MicroBlaze: (C_ADDR_SIZE > 32 and C_USE_MMU = 3) or (C_DATA_SIZE = 64) (n = C_ADDR_SIZE):

0	n-10	n-9	n-8	n-4	n-3	n-2	n-1
↑	EX	WR	ZSEL	W	I	M	G

X19751-111517

図 2-14: TLBLO

表 2-21: 変換ルックアサイド バッファーローレジスタ (TLBLO)

ビット ¹	名前	説明	リセット値
0:21 0:n-11	RPN	実ページ番号または物理ページ番号 TLB ヒットが発生すると、このフィールドは TLB エントリから読み出され、物理アドレスを作成するために使用されます。SIZE フィールドの値により、RPN の一部のビットは物理アドレスでは使用されません。このフィールドの未使用ビットは、ソフトウェアでゼロにクリアにする必要があります。 C_USE_MMU=3 (仮想) の場合のみ定義されます。 読み出し/書き込み	0x0000000
22 n-10	EX	実行ファイル 1 の場合、ページに実行可能コードが含まれており、命令をフェッチできます。0 の場合は、ページから命令はフェッチできません。EX ビットが 0 のページから命令をフェッチしようとすると、命令ストレージ例外が発生します。 読み出し/書き込み	0
23 n-9	WR	書き込み可能 1 の場合、ページは書き込み可能で、ストア命令を使用してページ内のアドレスにデータを格納できます。 0 の場合は、ページは読み出し専用(書き込み不可)です。WR ビットが 0 のページにデータを格納しようとすると、データストレージ例外が発生します。 読み出し/書き込み	0
24:27 n-8:n-5	ZSEL	ゾーン選択 ゾーン保護レジスタ (ZPR) の 16 個のゾーン (Z0 ~ Z15) から 1 つを選択します。 たとえば、ZSEL が 0x5 の場合はゾーンフィールド Z5 が選択されます。選択された ZPR フィールドは、TLB エントリの EX および WR フィールドにより指定されているアクセス保護を変更するのに使用されます。また、TLB V(有効) フィールドを無効にしてページへのアクセスを防止するためにも使用されます。 読み出し/書き込み	0x0
28 n-4	W	ライトスルー パラメーター C_DCACHE_USE_WRITEBACK が 1 に設定されている場合、このビットがキャッシングポリシーを制御します。このビットが 1 の場合はライトスルーポリシーが選択され、それ以外の場合はライトバックポリシーが選択されます。 C_DCACHE_USE_WRITEBACK が 0 の場合、このビットは 1 に固定され、常にライトスルーポリシーが使用されます。 読み出し/書き込み	0/1
29 n-3	I	キャッシングの禁止 1 の場合、ページへのアクセスはキャッシングされません(キャッシングの禁止)。 0 の場合、ページへのアクセスはキャッシング可能です。 読み出し/書き込み	0

表 2-21: 変換ルックアサイド バッファーローレジスタ (TLBLO) (続き)

ビット ¹	名前	説明	リセット値
30 n-2	M	メモリコヒーレント メモリコヒーレンシは MicroBlaze にはインプリメントされていないため、このビットは 0 に固定されています。 読み出し専用	0
31 n-1	G	ガード 1 の場合、投機的ページアクセスは許可されません (メモリがガードされる)。 0 の場合、投機的ページアクセスは許可されます。 G 属性は、メモリマップされた I/O を不適切な命令アクセスから保護するために使用できます。 読み出し/書き込み	0

1. PAE または 64 ビット MicroBlaze がイネーブルの場合は、ビット インデックス n=C_ADDR_SIZE が適用されます。

変換ルックアサイド バッファーハイ レジスタ (TLBHI)

変換ルックアサイド バッファーハイ レジスタは、MMU の統合変換ルックアサイド バッファー (UTLB) エントリにアクセスするために使用されます。MicroBlaze の C_USE_MMU コンフィギュレーション オプションで設定されます。このレジスタは、C_USE_MMU が 1 より大きく (ユーザー モード)、C_AREA_OPTIMIZED が 0 (パフォーマンス) または 2 (周波数) に設定されている場合にのみインプリメントされます。MFS および MTS 命令でアクセスする際、TLBHI は Sa = 0x1004 で指定します。TLBHI を読み出しありまたは書き込みするときは、TLBX レジスタでインデックス化された UTLB エントリにアクセスします。

このレジスタは、メモリ管理特殊レジスタのパラメーター C_MMU_TLB_ACCESS の設定に従って読み出し可能です。

PID は TLB エントリにアクセスするときにも使用されます。

- TLBHI を書き込むと、PID の値は TLB エントリの TID フィールドに格納されます。
- TLBHI を読み出すときに MSR[UM] がセットされていない場合は、TID フィールドの値が PID に格納されます。

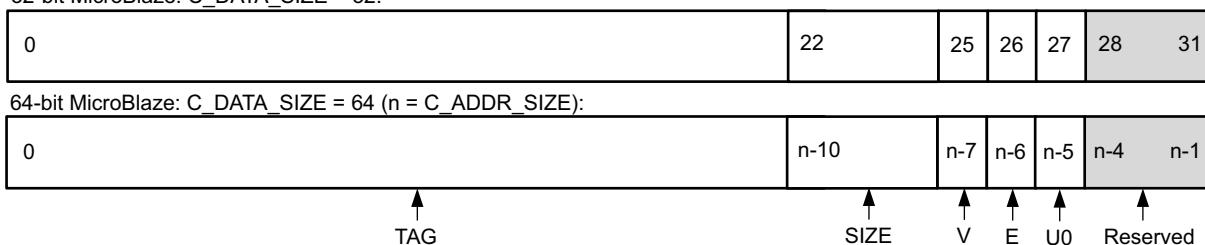
UTLB はビットストリームをダウンロードするとリセットされます (すべての TLBHI エントリのリセット値は 0x00000000)。

TLBHI は、64 ビット MicroBlaze がイネーブルの場合 (C_DATA_SIZE=64) は C_ADDR_SIZE パラメーターの設定によって 64 ビットまでで、それ以外の場合は 32 ビットです。

注記: UTLB は、外部リセット入力 (Reset および Debug_Rst) ではリセットされません。

次の図に TLBHI レジスタを図示し、表 2-22 にビットの説明とリセット値を示します。

32-bit MicroBlaze: C DATA SIZE = 32:



X19752-020618

図 2-15: TLBHI

表 2-22: 変換ルックアサイド バッファーハイレジスタ (TLBHI)

ビット ¹	名前	説明	リセット値
0:21 0:n-11	TAG	TLB エントリタグ SIZE フィールドの値に従って、仮想メモリ アドレスのページ番号の部分と比較されます。 読み出し/書き込み	0x0
22:24 n-10:n-8	SIZE	サイズ ページサイズを指定します。SIZE フィールドは、TAG フィールドを仮想メモリ アドレスのページ番号部分と比較するときに使用されるビット範囲を制御します。このフィールドで定義されるページサイズは表 2-39 にリストされています。 読み出し/書き込み	000
25 n-7	V	有効 1 の場合、TLB エントリは有効で、ページ変換エントリが含まれます。 0 の場合、TLB エントリは無効です。 読み出し/書き込み	0
26 n-6	E	エンディアン 1 の場合、ページはビッグ エンディアン ページとしてアクセスされます。 0 の場合、ページはリトル エンディアン ページとしてアクセスされます。 E ビットはデータ読み出しまたはデータ書き込みアクセスにのみ影響します。命令アクセスには影響しません。 E ビットは、C_USE_REORDER_INSTR が 1 に設定されている場合にのみインプリメントされ、それ以外の場合は 0 に固定されます。 読み出し/書き込み	0
27 n-5	U0	ユーザー定義 MicroBlaze にはユーザー定義のストレージ属性がないため、このビットは 0 に固定されています。 読み出し専用	0
28:31 n-4:n-1	予約		

1. 64 ビット MicroBlaze がイネーブルの場合は、ビット インデックス $n = \text{C_ADDR_SIZE}$ が適用されます。

変換ルックアサイド バッファー インデックスレジスタ (TLBX)

変換ルックアサイド バッファー インデックス レジスタは、TLBLO および TLBHI レジスタにアクセスするとき、統合変換ルックアサイド バッファー (UTLB) へのインデックスとして使用されます。MicroBlaze の C_USE_MMU コンフィギュレーション オプションで設定されます。このレジスタは、C_USE_MMU が 1 より大きく (ユーザー モード)、C_AREA_OPTIMIZED が 0 (パフォーマンス) または 2 (周波数) に設定されている場合にのみインプリメントされます。MFS および MTS 命令でアクセスする際、TLBX は Sa = 0x1002 で指定します。

次の図に TLBX レジスタを図示し、表 2-23 にビットの説明とリセット値を示します。

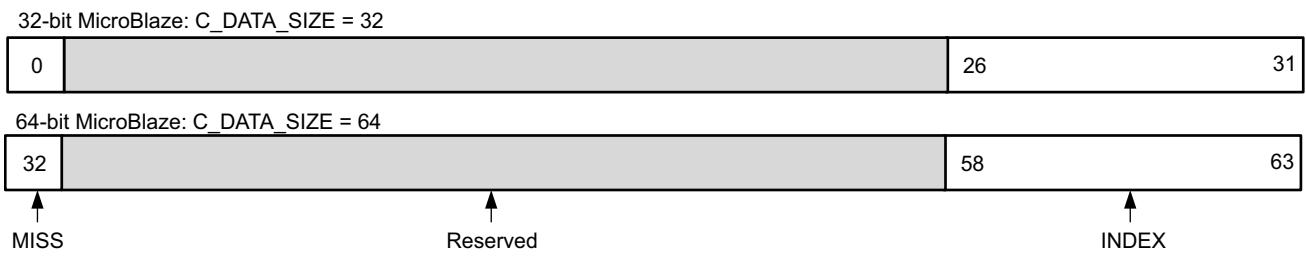


表 2-23: 変換ルックアップサイド バッファ=インデックスレジスタ (TIBX)

ビット ¹	名前	説明	リセット値
0、32	MISS	<p>TLB ミス</p> <p>TLBSX レジスタに仮想アドレスが書き込まれ、その仮想アドレスが TLB エントリにある場合、このビットは 0 になります。</p> <p>仮想アドレスが検出されない場合、1 になります。TLBX レジスタ自体が書き込まれた場合にも 0 になります。</p> <p>読み出し専用</p> <p>メモリ管理特殊レジスタのパラメーターが C_MMU_TLB_ACCESS > 0 (MINIMAL) の場合、読み出し可能です。</p>	0
1:25 33:57	予約		
26:31 58:63	INDEX	<p>TLB インデックス</p> <p>TLBLO および TLBHI レジスタでアクセスされる変換ルックアサイドバッファーのエントリをインデックス化するために使用されます。</p> <p>TLBSX レジスタに仮想アドレスが書き込まれ、その仮想アドレスが対応する TLB エントリにある場合、このフィールドは TLB インデックスでアップデートされます。</p> <p>読み出し/書き込み</p> <p>メモリ管理特殊レジスタのパラメーターが C_MMU_TLB_ACCESS > 0 (MINIMAL) の場合、読み出しおよび書き込み可能です。</p>	000000

1. ビット番号は、64ビット MicroBlaze (C DATA SIZE = 64) がバイナリブルかどうかによります。

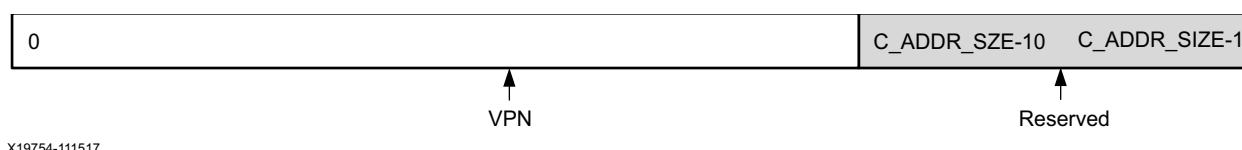
変換ルックアサイド バッファー検索インデックス レジスタ (TLBSX)

変換ルックアサイド バッファー検索インデックス レジスタ (TLBSX) は、統合変換ルックアサイド バッファー (UTLB) で仮想ページ番号を検索するために使用されます。これは、MicroBlaze プロセッサの C_USE_MMU コンフィギュレーション オプションで設定されます。

このレジスタは、C_USE_MMU が 1 より大きく (ユーザー モード)、C_AREA_OPTIMIZED が 0 (パフォーマンス) または 2 (周波数) に設定されている場合にのみインプリメントされます。

MTS 命令で書き込む際、TLBSX は Sa = 0x1005 で指定します。次の図に TLBSX レジスタを図示し、表 2-24 にビットの説明とリセット値を示します。

TLBSX は、64 ビット MicroBlaze がイネーブルの場合 (C_DATA_SIZE = 64) は C_ADDR_SIZE パラメーターの設定によって 64 ビットまでで、それ以外の場合は 32 ビットです。



X19754-111517

図 2-17: TLBSX

表 2-24: 変換ルックアサイド バッファー検索インデックス レジスタ (TLBSX)

ビット ¹	名前	説明	リセット値
0:21 0:n-9	VPN	仮想ページ番号 仮想メモリアドレスのページ番号部分を表します。V ビットが 1 にセットされている変換ルックアサイド バッファーの各エントリで、SIZE フィールドの値に従って、このフィールドが仮想メモリアドレスのページ番号部分と比較されます。 仮想ページ番号が検出されると、TLBX レジスタに TLB エントリのインデックスが書き込まれ、TLBX の MISS ビットが 0 になります。仮想ページ番号がどの TLB エントリでも検出されない場合は、TLBX レジスタの MISS ビットが 1 になります。 書き込み専用	
22:31 n-10:n-1	予約		

1. 64 ビット MicroBlaze がイネーブルの場合は、ビット インデックス n = C_ADDR_SIZE が適用されます。

プロセッサ バージョンレジスタ (PVR)

プロセッサ バージョンレジスタは、MicroBlaze の C_PVR コンフィギュレーションオプションで設定されます。

- C_PVR が 0(なし)に設定されている場合、プロセッサに PVR はインプリメントされず、MSR[PVR]=0 になります。
- C_PVR が 1(Basic)に設定されている場合は MicroBlaze に最初のレジスタ PVR0 のみがインプリメントされ、C_PVR が 2(Full)に設定されている場合は 13 個の PVR レジスタすべて(PVR0 ~ PVR12)がインプリメントされます。

MFS または MFSE 命令で読み出す際、PVR は Sa=0x200x で指定します。ここで、x はレジスタ番号(0x0 ~ 0xB)です。

拡張データアドレス指定がイネーブル(C_DATA_SIZE=32 および C_ADDR_SIZE > 32)の場合、PVR8 および PVR9 の下位 32 ビットは MFS 命令で読み出し、上位ビットは MFSE 命令で読み出します。

物理アドレス拡張(PAE)がイネーブル(C_DATA_SIZE=32、C_USE_MMU=3、および C_ADDR_SIZE > 32)の場合、PVR6 および PVR7 の下位 32 ビットは MFS 命令で読み出し、上位ビットは MFSE 命令で読み出します。

64 ビット MicroBlaze (C_DATA_SIZE=64)では、PVR6 ~ PVR9 および PVR12 レジスタ全体を MFS 命令で読み出します。

表 2-25 ~ 表 2-37 に、ビットの説明と値を示します。

表 2-25: プロセッサ バージョンレジスタ 0 (PVR0)

ビット ¹	名前	説明	値
0、32	CFG	PVR インプリメンテーション: 0: 基本 1: フル	パラメーター C_PVR に基づく
1、33	BS	バレルシフターを使用	C_USE_BARREL
2、34	DIV	除算器を使用	C_USE_DIV
3、35	MUL	ハードウェア乗算器を使用	C_USE_HW_MUL > 0(なし)
4、36	FPU	FPU を使用	C_USE_FPU > 0(なし)
5、37	EXC	任意タイプの例外を使用	C_*_EXCEPTION に基づく C_USE_MMU > 0(なし)の場合も設定される
6、38	ICU	命令キャッシュを使用	C_USE_ICACHE
7、39	DCU	データキャッシュを使用	C_USE_DCACHE
8、40	MMU	MMU を使用	C_USE_MMU > 0(なし)
9、41	BTC	分岐先キャッシュを使用	C_USE_BRANCH_TARGET_CACHE
10、42	ENDI	選択されているエンディアンネス 常に 1(リトルエンディアン)	C_ENDIANNESS
11、43	FT	フォールトトレランス機能をインプリメント	C_FAULT_TOLERANT
12、44	SPROT	スタック保護を使用	C_USE_STACK_PROTECTION
13、45	REORD	再順序付け命令をインプリメント	C_USE_REORDER_INSTR
14、46	64BIT	64 ビット MicroBlaze	C_DATA_SIZE = 64
15、47	予約		0

表 2-25: プロセッサ バージョン レジスタ 0 (PVR0) (続き)

ビット ¹	名前	説明	値
16:23	MBV	MicroBlaze リリース バージョン コード 0x19: v8.40.b 0x1B: v9.0 0x1D: v9.1 0x1F: v9.2 0x20: v9.3	リリース別 0x21: v9.4 0x22: v9.5 0x23: v9.6 0x24: v10.0 0x25: v11.0
24:31	USR1	ユーザー設定値 1	C_PVR_USER1
56:63			

1. ビット番号は、64 ビット MicroBlaze (C_DATA_SIZE = 64) がイネーブルかどうかによります。

表 2-26: プロセッサ バージョン レジスタ 1 (PVR1)

ビット ¹	名前	説明	値
0:31	USR2	ユーザー設定値 2	C_PVR_USER2
32:63			

1. ビット番号は、64 ビット MicroBlaze (C_DATA_SIZE = 64) がイネーブルかどうかによります。

表 2-27: プロセッサ バージョン レジスタ 2 (PVR2)

ビット ¹	名前	説明	値
0、32	DAXI	使用されているデータ側 AXI4 または ACE	C_D_AXI
1、33	DLMB	使用されているデータ側 LMB	C_D_LMB
2、34	IAXI	使用されている命令側 AXI4 または ACE	C_I_AXI
3、35	ILMB	使用されている命令側 LMB	C_I_LMB
4、36	IRQEDGE	割り込みをエッジでトリガー	C_INTERRUPT_IS_EDGE
5、37	IRQPOS	割り込みエッジは立ち上がり	C_EDGE_IS_POSITIVE
6、38	CEEXEC	LMB メモリの ECC 訂正可能エラーに対してバス例外を生成	C_ECC_USE_CE_EXCEPTION
7、39	FREQ	プロセッサ周波数を最適化するインプリメンテーションを選択	C_AREA_OPTIMIZED=2 (周波数)
8、40	予約		0
9、41	予約		1
10、42	ACE	ACE インターコネクトを使用	C_INTERCONNECT = 3 (ACE)
11、43	AXI4DP	データ ペリフェラル AXI インターフェイスに AXI4 プロトコルを使用 (排他的アクセスのサポートあり)	C_M_AXI_DP_EXCLUSIVE_ACCESS
12、44	FSL	拡張 AXI4-Stream 命令を使用	C_USE_EXTENDED_FSL_INSTR
13、45	FSLEXC	AXI4-Stream 制御ビット不一致に対して例外を生成	C_FSL_EXCEPTION
14、46	MSR	msrset および msrclr 命令を使用	C_USE_MSR_INSTR
15、47	PCMP	パターン比較および CLZ 命令を使用	C_USE_PCMP_INSTR

表 2-27: プロセッサバージョンレジスタ2(PVR2)(続き)

ビット ¹	名前	説明	値
16、48	AREA	低い命令スループットでエリアを最適化するインプリメンテーションを選択	C_AREA_OPTIMIZED = 1(エリア)
17、49	BS	バレルシフターを使用	C_USE_BARREL
18、50	DIV	除算器を使用	C_USE_DIV
19、51	MUL	ハードウェア乗算器を使用	C_USE_HW_MUL > 0(なし)
20、52	FPU	FPU を使用	C_USE_FPU > 0(なし)
21、53	MUL64	64ビットハードウェア乗算器を使用	C_USE_HW_MUL = 2(Mul64)
22、54	FPU2	浮動小数点変換および平方根命令を使用	C_USE_FPU = 2(拡張)
23、55	IMPEXC	LMBメモリのECCエラーに対してあいまい例外を許可	C_IMPRECISE_EXCEPTIONS
24、56	予約		0
25、57	OP0EXC	0x0の無効なオペコードに対して例外を生成	C_OPCODE_0x0_ILLEGAL
26、58	UNEXC	非境界整列データアクセスに対して例外を生成	C_UNALIGNED_EXCEPTIONS
27、59	OPEXC	任意の無効なオペコードに対して例外を生成	C_ILL_OPCODE_EXCEPTION
28、60	AXIDEXC	M_AXI_Dエラーに対して例外を生成	C_M_AXI_D_BUS_EXCEPTION
29、61	AXIIEXC	M_AXI_Iエラーに対して例外を生成	C_M_AXI_I_BUS_EXCEPTION
30、62	DIVEXC	ゼロ除算または除算オーバーフローに対して例外を生成	C_DIV_ZERO_EXCEPTION
31、63	FPUEXC	FPUからの例外を生成	C_FPU_EXCEPTION

1. ビット番号は、64ビットMicroBlaze(C_DATA_SIZE = 64)がイネーブルかどうかによります。

表 2-28: プロセッサバージョンレジスタ3(PVR3)

ビット ¹	名前	説明	値
0、32	DEBUG	デバッグロジックを使用	C_DEBUG_ENABLED > 0
1、33	EXT_DEBUG	拡張デバッグロジックを使用	C_DEBUG_ENABLED = 2(拡張)
2、34	予約		
3:6 35:38	PCBRK	PCブレークポイントの数	C_NUMBER_OF_PC_BRK
7:9 39:41	予約		
10:12 42:44	RDADDR	読み出しアドレスブレークポイントの数	C_NUMBER_OF_RD_ADDR_BRK
13:15 45:47	予約		
16:18 48:50	WRADDR	書き込みアドレスブレークポイントの数	C_NUMBER_OF_WR_ADDR_BRK
19、51	予約		0

表 2-28: プロセッサ バージョン レジスタ 3 (PVR3) (続き)

ビット ¹	名前	説明	値
20:24 52:56	FSL	AXI4-Stream リンクの数	C_FSL_LINKS
25:28 57:60	予約		
29:31 61:63	BTC_SIZE	分岐先キャッシュ サイズ	C_BRANCH_TARGET_CACHE_SIZE

1. ビット番号は、64 ビット MicroBlaze (C_DATA_SIZE = 64) がイネーブルかどうかによります。

表 2-29: プロセッサ バージョン レジスタ 4 (PVR4)

ビット ¹	名前	説明	値
0、32	ICU	命令キャッシュを使用	C_USE_ICACHE
1:5 33:37	ICTS	命令キャッシュ タグ サイズ	C_ADDR_TAG_BITS
6、38	予約		1
7、39	ICW	命令キャッシュ書き込みを許可	C_ALLOW_ICACHE_WR
8:10 40:42	ICLL	命令キャッシュ ラインの長さの 2 を底とする対数	$\log_2(C_ICACHE_LINE_LEN)$
11:15 43:47	ICBS	命令キャッシュ バイト サイズの 2 を底とする対数	$\log_2(C_CACHE_BYTE_SIZE)$
16、 48	IAU	キャッシュ可能範囲内のすべてのメモリ アクセス に対して命令キャッシュを使用	C_ICACHE_ALWAYS_USED
17:18 49:50	予約		0
19:21 51:53	ICV	命令キャッシュ ビクティム	0-3: C_ICACHE_VICTIMS = 0、2、4、8
22:23 54:55	ICS	命令キャッシュ ストリーム	C_ICACHE_STREAMS
24、 56	IFTL	命令キャッシュ タグに分散 RAM を使用	C_ICACHE_FORCE_TAG_LUTRAM
25、 57	ICDW	命令キャッシュ データ幅	C_ICACHE_DATA_WIDTH > 0
26:31 58:63	予約		0

1. ビット番号は、64 ビット MicroBlaze (C_DATA_SIZE = 64) がイネーブルかどうかによります。

表 2-30: プロセッサ バージョン レジスタ 5 (PVR5)

ビット ¹	名前	説明	値
0、32	DCU	データ キャッシュを使用	C_USE_DCACHE
1:5 33:37	DCTS	データ キャッシュ タグ サイズ	C_DCACHE_ADDR_TAG
6、38	予約		1
7、39	DCW	データ キャッシュ書き込みを許可	C_ALLOW_DCACHE_WR
8:10 40:42	DCLL	データ キャッシュ ラインの長さの 2 を底とする 対数	$\log_2(C_DCACHE_LINE_LEN)$
11:15 43:47	DCBS	データ キャッシュ バイト サイズの 2 を底とする 対数	$\log_2(C_DCACHE_BYTE_SIZE)$
16、48	DAU	キャッシュ可能範囲内のすべてのメモリ アクセスに対してデータ キャッシュを使用	C_DCACHE_ALWAYS_USED
17、49	DWB	データ キャッシュ ポリシーはライトバック	C_DCACHE_USE_WRITEBACK
18、50	予約		0
19:21 51:53	DCV	データ キャッシュ ビクトミ	0-3: C_DCACHE_VICTIMS = 0、2、4、8
22:23 54:55	予約		0
24、56	DFTL	データ キャッシュ タグに分散 RAM を使用	C_DCACHE_FORCE_TAG_LUTRAM
25、57	DCDW	データ キャッシュ データ幅	C_DCACHE_DATA_WIDTH > 0
26、58	AXI4DC	データ キャッシュ AXI インターフェイスに AXI4 プロトコルを使用 (排他的アクセスのサポートあり)	C_M_AXI_DC_EXCLUSIVE_ACCESS
27:31 59:63	予約		0

1. ビット番号は、64 ビット MicroBlaze (C_DATA_SIZE = 64) がイネーブルかどうかによります。

表 2-31: プロセッサ バージョン レジスタ 6 (PVR6)

ビット	名前	説明	値
0:C_ADDR_SIZE-1	ICBA	命令キャッシュ ベース アドレス	C_ICACHE_BASEADDR

表 2-32: プロセッサ バージョン レジスタ 7 (PVR7)

ビット	名前	説明	値
0:C_ADDR_SIZE-1	ICHA	命令キャッシュ ハイ アドレス	C_ICACHE_HIGHADDR

表 2-33: プロセッサ バージョン レジスタ 8 (PVR8)

ビット	名前	説明	値
0:C_ADDR_SIZE-1	DCBA	データ キャッシュ ベース アドレス	C_DCACHE_BASEADDR

表 2-34: プロセッサ バージョン レジスタ 9 (PVR9)

ビット	名前	説明	値
0:C_ADDR_SIZE-1	DCHA	データ キャッシュ ハイ アドレス	C_DCACHE_HIGHADDR

表 2-35: プロセッサ バージョン レジスタ 10 (PVR10)

ビット ¹	名前	説明	値
0:7 32:39	ARCH	ターゲット アーキテクチャ: 0xF = Virtex®-7、防衛グレード Virtex-7 Q 0x10 = Kintex®-7、防衛グレード Kintex-7 Q 0x11 = Artix®-7、オートモーティブ Artix-7、防衛グレード Artix-7 Q 0x12 = Zynq®-7000、オートモーティブ Zynq-7000 0x13 = 防衛グレード Zynq-7000 Q 0x14 = UltraScale™ Virtex 0x15 = UltraScale Kintex 0x16 = UltraScale+™ Zynq 0x17 = UltraScale+ Virtex 0x18 = UltraScale+ Kintex Spartan®-7	パラメーター C_FAMILY で定義
8:13 40:45	ASIZE	拡張アドレス ビットの数	C_ADDR_SIZE - 32
14:31 46:63	予約		0

1. ビット番号は、64 ビット MicroBlaze (C_DATA_SIZE = 64) がイネーブルかどうかによります。

表 2-36: プロセッサ バージョン レジスタ 11 (PVR11)

ビット ¹	名前	説明	値
0:1 32:33	MMU	MMU を使用: 0: なし 1: ユーザー モード	2: 保護 3: 仮想 C_USE_MMU
2:4 34:36	ITLB	命令シャドウ TLB サイズ	$\log_2(C_{_MMU_ITLB_SIZE})$
5:7 37:39	DTLB	データシャドウ TLB サイズ	$\log_2(C_{_MMU_DTLB_SIZE})$
8:9 40:41	TLBACC	TLB レジスタ アクセス: 0: 最小 1: 読み出し	2: 書き込み 3: フル C_MMU_TLB_ACCESS
10:14 42:46	ZONES	メモリ保護ゾーンの数	C_MMU_ZONES
15、 47	PRIVINS	特権命令: 0: フル保護 1: ストリーム命令を許可	C_MMU_PRIVILEGED_INSTR
16、 48	予約	今後の使用のために予約	0
17:31 49:63	RSTMSR	MSR のリセット値	C_RESET_MSR_IE << 2 C_RESET_MSR_BIP << 4 C_RESET_MSR_ICE << 6 C_RESET_MSR_DCE << 8 C_RESET_MSR_EE << 9 C_RESET_MSR_EIP << 10

1. ビット番号は、64 ビット MicroBlaze ($C_{_DATA_SIZE} = 64$) がイネーブルかどうかによります。

表 2-37: プロセッサ バージョン レジスタ 12 (PVR12)

ビット ¹	名前	説明	値
0:31 0:C_ADDR_SIZE-1	VECTORS	MicroBlaze ベクターの位置	C_BASE_VECTORS

1. 64 ビット MicroBlaze ($C_{_DATA_SIZE} = 64$) では $C_{_ADDR_SIZE}$ ビット、それ以外の場合は 32 ビット。

パイプラインアーキテクチャ

MicroBlaze 命令の実行はパイプライン処理されます。ほとんどの命令では、各段の完了に 1 クロック サイクルかかります。つまり、特定の命令が完了するのに必要なクロック サイクル数は、パイプラインの段数と等しくなり、データ、制御、または構造的ハザードが存在しない場合は各サイクルで命令が 1 つ完了します。

命令の結果が後続の命令で必要な場合、データ ハザードが発生します。この場合、結果を後続の命令にフォワードしないと、パイプラインがストールします。MicroBlaze GNU コンパイラでは、最適化中に命令を並べ替えることによりデータ ハザードを回避することが試みられます。

分岐が取られ、次の命令がすぐに実行できないと、制御ハザードが発生します。この結果、パイプラインがストールします。MicroBlaze には、ストール サイクル数を削減するため、遅延スロット分岐およびオプションの分岐ターゲット キャッシュが含まれています。

構造的ハザードは、実行段またはその後の段を完了するのに複数のクロック サイクルが必要になる命令で発生します。これは、パイプラインをストールさせることにより実現します。

低速のメモリにアクセスするロードおよびストア命令には、複数サイクルかかることがあります。アクセスが完了するまで、パイプラインはストールされます。MicroBlaze には、低速のメモリの平均レイテンシを向上するため、オプションのデータ キャッシュが含まれています。

低速のメモリから実行する場合、命令フェッチに複数サイクルかかることがあります。この追加レイテンシは、パイプラインの効率に直接影響します。MicroBlaze は、このような複数サイクル命令のメモリレイテンシの影響を削減する命令プリフェッチバッファーをインプリメントします。パイプラインがその他の理由でストールしている間は、プリフェッチバッファーは投機的に順次命令をロードし続けます。パイプラインが実行を再開すると、命令メモリアクセスが完了するのを待たずに、フェッチ段はプリフェッチバッファーから直接新しい命令をロードします。

実行中に命令が変更される場合(自己変更コードなど)、変更された命令が実行される前に、プリフェッチバッファーを空にして古い未変更の命令が含まれないようにする必要があります。



推奨: BRI 4 などの同期分岐命令を使用することも可能ですが、MBAR 命令を使用することを推奨します。

MicroBlaze には、低速メモリの平均命令フェッチ レイテンシを向上するため、オプションの命令キャッシュも含まれています。

すべてのハザードは独立しており、すべてが同時に発生する可能性があります。その場合、パイプラインがストールされるサイクル数は、ストール期間が 1 番長いハザードによって定義されます。

3段パイプライン

C_AREA_OPTIMIZED が 1(エリア) に設定されている場合、ハードウェアコストを最小限に抑えるため、パイプラインはフェッチ、デコード、実行の3段に分けられます。

	サイクル1	サイクル2	サイクル3	サイクル4	サイクル5	サイクル6	サイクル7
命令1	フェッチ	デコード	実行				
命令2		フェッチ	デコード	実行	実行	実行	
命令3			フェッチ	デコード	ストール	ストール	実行

3段パイプラインには、データハザードはありません。パイプラインのストールは、制御ハザード、複数サイクル命令による構造的ハザード、低速メモリを使用するメモリアクセス、低速メモリからの命令フェッチ、またはストリームアクセスにより発生します。

複数サイクル命令カテゴリは、パレルシフト、乗算、除算、浮動小数点命令です。

5段パイプライン

C_AREA_OPTIMIZED が 0(パフォーマンス) に設定されている場合、パフォーマンスを最大にするため、パイプラインはフェッチ(IF)、デコード(OF)、実行(EX)、アクセスメモリ(MEM)、およびライトバック(WB)の5段に分けられます。

	サイクル1	サイクル2	サイクル3	サイクル4	サイクル5	サイクル6	サイクル7	サイクル8	サイクル9
命令1	IF	OF	EX	MEM	WB				
命令2		IF	OF	EX	MEM	MEM	MEM	WB	
命令3			IF	OF	EX	ストール	ストール	MEM	WB

5段パイプラインには、2種類のデータハザードがあります。

- OFの命令には、ソースオペランドとして EX の命令からの結果が必要です。この場合、EX命令のカテゴリはロード、ストア、パレルシフト、乗算、除算、および浮動小数点命令です。この結果、1~2サイクルのストールとなります。
- OFの命令には、ソースオペランドとして MEM の命令からの結果が必要です。この場合、MEM命令のカテゴリはロード、乗算、および浮動小数点命令です。この結果、1サイクルのストールとなります。

パイプラインのストールは、データハザード、制御ハザード、複数サイクル命令による構造的ハザード、低速メモリを使用するメモリアクセス、低速メモリからの命令フェッチ、またはストリームアクセスにより発生します。

複数サイクル命令カテゴリは、除算および浮動小数点命令です。

8段パイプライン

C_AREA_OPTIMIZED が 2(周波数)に設定されている場合、周波数を最大にするため、パイプラインはフェッチ(IF)、デコード(OF)、実行(EX)、アクセスメモリ 0(M0)、アクセスメモリ 1(M1)、アクセスメモリ 2(M2)、アクセスメモリ 3(M3)およびライトバック(WB)の8段に分けられます。

	サイクル 1	サイクル 2	サイクル 3	サイクル 4	サイクル 5	サイクル 6	サイクル 7	サイクル 8	サイクル 9	サイクル 10	サイクル 11
命令 1	IF	OF	EX	M0	M1	M2	M3	WB			
命令 2		IF	OF	EX	M0	M0	M1	M2	M3	WB	
命令 3			IF	OF	EX	ストール	M0	M1	M2	M3	WB

8段パイプラインには、4種類のデータハザードがあります。

- OF の命令には、ソースオペランドとして EX の命令からの結果が必要です。この場合、EX 命令のカテゴリはロード、ストア、パレルシフト、乗算、除算、および浮動小数点命令です。この結果、1~5 サイクルのストールとなります。
- OF の命令には、ソースオペランドとして M0 の命令からの結果が必要です。この場合、M0 命令のカテゴリはロード、乗算、除算、および浮動小数点命令です。この結果、1~4 サイクルのストールとなります。
- OF の命令には、ソースオペランドとして M1 または M2 の命令からの結果が必要です。この場合、M1 または M2 命令のカテゴリはロード、除算、および浮動小数点命令です。この結果、それぞれ 1~3 または 1~2 サイクルのストールとなります。
- OF の命令には、ソースオペランドとして M3 の命令からの結果が必要です。この場合、M3 命令のカテゴリはロードおよび浮動小数点命令です。この結果、1 サイクルのストールとなります。

複数サイクル命令に加え、3種類の構造的ハザードがあります。

- OF の命令はストリーム命令、EX の命令は対応する例外が組み込まれたストリーム、ロード、ストア、除算、または浮動小数点命令です。この結果、1 サイクルのストールとなります。
- OF の命令はストリーム命令、M0、M1、M2、または M3 の命令は対応する例外が組み込まれたロード、ストア、除算、または浮動小数点命令です。この結果、1 サイクルのストールとなります。
- M0 の命令はロードまたはストア命令、M1、M2、または M3 の命令は対応する例外が組み込まれたロード、ストア、除算、または浮動小数点命令です。この結果、1 サイクルのストールとなります。

パイプラインのストールは、データハザード、制御ハザード、構造的ハザード、低速メモリを使用するメモリアクセス、低速メモリからの命令フェッチ、またはストリームアクセスにより発生します。

複数サイクル命令カテゴリは、除算命令および浮動小数点命令 FDIV、FINT、FSQRT、DDIV、DLONG、および DSQRT です。

分岐

通常、フェッチおよびデコード段(およびプリフェッチバッファー)の命令は、分岐を実行するときにフラッシュされます。その後、フェッチパイプライン段に算出された分岐アドレスからの新しい命令がロードされます。

MicroBlaze の分岐の実行には 3 クロックサイクルかかります。そのうち 2 サイクルは、パイプラインを再度充填するのに必要です。このレインシオーバーヘッドを削減するため、MicroBlaze では遅延スロットおよびオプションの分岐ターゲットキャッシュのある分岐がサポートされています。

遅延スロット

遅延スロットのある分岐を実行する際は、MicroBlaze のフェッチパイプライン段のみがフラッシュされ、デコード段の命令(分岐遅延スロット)が完了できるようになっています。この手法により、分岐ペナルティを 2 クロックサイクルから 1 クロックサイクルに削減できます。遅延スロットのある分岐命令には、命令ニーモニックに D が付いています。たとえば、BNE 命令は後続の命令を実行しませんが(遅延スロットなし)、BNED 命令は制御が分岐位置に転送される前に次の命令を実行します。

遅延スロットには、IMM、IMML、分岐、またはブレーク命令を含めることはできません。割り込みおよび外部ハードウェアブレークは、遅延スロット分岐が完了するまで保留されます。リカバリ可能な例外を発生させる可能性のある命令(非境界整列ワードまたはハーフワードのロードおよびストアなど)は遅延スロットで処理可能です。

ESR[DS] ビットがセットされている遅延スロットで例外が発生すると、例外ハンドラーが実行を分岐先に戻す必要があります(特殊レジスタ BTR に格納)。ESR[DS] ビットがセットされている場合、レジスタ R17 は無効です。セットされていない場合は、レジスタ R17 に例外を発生させた命令の後のアドレスが含まれます。

分岐先キャッシュ

分岐パフォーマンスを向上させるため、MicroBlaze には分岐予測機能のある分岐先キャッシュ(BTC)があります。BTC がイネーブルの場合、正しく予測された即時分岐または戻り命令にオーバーヘッドは発生しません。

BTC は、命令が初めて実行されたときに、即時分岐および戻り命令のターゲットアドレスを保存します。次回その命令が実行されると、通常分岐先キャッシュで検出されるので、分岐が取られる場合に備えて、命令フェッチプログラム カウンターが保存されているターゲットアドレスに変わります。条件なし分岐および戻り命令は常に実行されますが、条件付き分岐では、実行すべきでない分岐が実行されるのを避けるため、分岐予測が使用されます。

メモリバリア(MBAR 0)または同期分岐(BRI 4)が実行されると、BTC はクリアされます。分岐が取られても、分岐命令の直後にメモリバリアまたは同期分岐が続く場合にも BTC はクリアされます。BTC が誤ってクリアされるのを避けるため、分岐命令の直後にメモリバリアまたは同期分岐を配置しないでください。

分岐予測での予測ミスには、次の 3 つのケースがあります。

- 取るべきでない条件付き分岐が実際に取られる。
- 取るべき条件付き分岐が取られない。
- 戻り命令のターゲットアドレスが間違っている。これは、コードの別の位置から呼び出された関数から戻るときに発生する可能性があります。

これらのケースはすべて、分岐または戻り命令が実行段に達したときに検出および訂正され、実際の命令の動作を反映させるため、分岐予測ビットまたはターゲットアドレスが BTC でアップデートされます。この訂正で、5 段パイプラインの場合は 2 クロックサイクルのペナルティ、8 段パイプラインの場合は 7 ~ 9 クロックサイクルのペナルティが発生します。

BTC のサイズは、C_BRANCH_TARGET_CACHE_SIZE で指定できます。デフォルトの推奨設定では、32 ビット アドレス (C_ADDR_SIZE=32) のブロック RAM が 1 つ使用され、512 個のエントリが提供されます。64 個以下のエントリを選択した場合は、BTC のインプリメントに分散 RAM が使用されます。それ以外の場合はブロック RAM が使用されます。

BTC にブロック RAM が使用され、C_FAULT_TOLERANT が 1 に設定されている場合は、ブロック RAM はパリティで保護されます。パリティ エラーが発生した場合は、分岐は予測されません。この場合にエラーが累積するのを回避するには、同期分岐で定期的に BTC をクリアする必要があります。

分岐先キャッシュは、C_USE_BRANCH_TARGET_CACHE が 1、C_AREA_OPTIMIZED が 0 (パフォーマンス) または 2 (周波数) に設定されているときに使用できます。

パイプライン ハザード例

表 2-38 に、データ ハザードの影響を 5 段パイプラインを使用して示します。

この例は乗算命令のデータ ハザードを示しており、後続の加算命令にレジスタ r3 の結果が必要です。つまり、加算命令はサイクル 3 および 4 で乗算が完了するまでストールします。

表 2-38: 乗算のデータ ハザード例

サイクル	IF	OF	EX	MEM	WB
1	mul r3, r4, r5				
2	add r6, <u>r3</u> , r4	mul r3, r4, r5			
3		add r6, <u>r3</u> , r4	mul r3, r4, r5		
4		add r6, <u>r3</u> , r4	-	mul r3, r4, r5	
5		add r6, <u>r3</u> , r4	-	-	mul r3, r4, r5
6			add r6, <u>r3</u> , r4	-	-

データ ハザードの回避

MicroBlaze GNU コンパイラでデータ ハザードを完全に回避するようコードを最適化できない場合もありますが、通常は汎用レジスタをより良く使用するようソース コードを変更することによりこれを達成可能です。

次に、2 つの C コード例を示します。

- メモリのスタティック配列の乗算

```
static int a[4], b[4], c[4];
register int a0, a1, a2, a3, b0, b1, b2, b3, c0, c1, c2, c3;

a0 = a[0]; a1 = a[1]; a2 = a[2]; a3 = a[3];
b0 = b[0]; b1 = b[1]; b2 = b[2]; b3 = b[3];
c0 = a0 * b0; c1 = a1 * b1; c2 = a2 * b2; c3 = a3 * b3;
c[3] = c3; c2 = c[2]; c1 = c[1]; c0 = c[0];
```

このコードは、まずロード命令を実行してオペランドを個別のレジスタに読み込んでから、それらを乗算して結果を格納します。このコードは、汎用レジスタを使い切ることなく 8 つまでの乗算に拡張できます。

- AXI4-Stream インターフェイスからのデータパケットのフェッチ。

```
#include <mb_interface.h>

static int a[4];
register int a0, a1, a2, a3;

getfs1(a0, 0); getfs1(a1, 0); getfs1(a2, 0); getfs1(a3, 0);
a[3] = a3; a[1] = a1; a[2] = a2; a[0] = a0;
```

このコードは、まず異なるレジスタを使用して `get` 命令を実行してからデータを格納します。このコードは、汎用レジスタを使い切ることなく 16 個までのアクセスに拡張できます。

メモリアーキテクチャ

MicroBlaze は、命令およびデータのアクセスが個別のアドレス空間で実行されるハーバード メモリアーキテクチャでインプリメントされます。

32 ビット MicroBlaze では、命令アドレス空間は 32 ビットの仮想アドレス範囲 (4 GB までの命令を処理) で、MMU を仮想モードで使用すると 64 ビットの物理アドレス範囲まで拡張可能です。64 ビット MicroBlaze では、命令アドレス空間はデフォルトで 32 ビット範囲ですが、64 ビット範囲まで拡張可能 (4 GB ~ 16 EB の命令を処理)。

データアドレス空間はデフォルトで 32 ビット範囲ですが、64 ビット範囲まで拡張可能 (4 GB から 16 EB までのデータを処理)。命令およびデータメモリの範囲は、両方を同じ物理メモリにマップしてオーバーラップさせることができます。後者はソフトウェアデバッグに必要です。

MicroBlaze の命令およびデータインターフェイスはどちらも、デフォルトの幅は 32 ビットで、エンディアンネスの設定によってビッグエンディアン、リトルエンディアン、ビット反転フォーマットを使用します。MicroBlaze は、データメモリへのアクセスにワード、ハーフワード、バイトをサポートします。

ビッグエンディアンフォーマットは、仮想モードまたは保護モードで MMU を使用している場合 (`C_USE_MMU > 1`)、または再順序付け命令がイネーブルになっている場合 (`C_USE_REORDER_INSTR = 1`) にのみ使用可能です。

プロセッサが非境界整列例外をサポートするよう設定されていない場合は、データアクセスは境界整列(ワードアクセスの場合はワードの境界、ハーフワードの場合はハーフワードの境界で整列)されている必要があります。命令アクセスはすべてワードで界整列されている必要があります。

MicroBlaze では、パフォーマンスを向上させるため、命令プリフェッチバッファーおよび(イネーブルになっている場合は)命令キャッシュストリームを使用して命令がプリフェッチされます。物理メモリの境界を越えて命令がプリフェッチされると、命令バスエラーまたはプロセッサストールが発生する可能性があるため、それを回避するため、命令を物理メモリの境界付近に配置しないようにする必要があります。命令プリフェッチバッファーには 16 バイトのマージンが必要で、命令キャッシュストリームを使用すると、さらに 2 つのキャッシュライン (32、64、または 128 バイト) が追加されます。

MicroBlaze では、I/O へのデータアクセスとメモリへのデータアクセスは分離されません (メモリマップされた I/O を使用)。メモリアクセス用には 3 つまでのインターフェイスがあります。

- ローカルメモリバス (LMB)
- ペリフェラルアクセス用の AXI4 (Advanced eXtensible Interface)
- キャッシュアクセス用の AXI4 または AXI4 コヒーレンシ拡張 (ACE)

LMB メモリアドレス範囲は、AXI4 の範囲とオーバーラップさせることはできません。

C_ENDIANNESS パラメーターは常にリトルエンディアンに設定されています。

ローカルメモリ(LMB)へのアクセスおよびキャッシュ読み出しヒットに対し、MicroBlazeには1サイクルのレイテンシがあります。ただし、C_AREA_OPTIMIZEDが1(エリア)に設定されていてデータ側のアクセスおよびデータキャッシュ読み出しヒットに2クロックサイクル必要な場合およびC_FAULT_TOLERANTが1に設定されていてLMBへのバイト書き込みおよびハーフワード書き込みに2クロックサイクル必要な場合は例外です。

データキャッシュ書き込みレイテンシは、C_DCACHE_USE_WRITEBACKによって異なります。

C_DCACHE_USE_WRITEBACKが1に設定されている場合、書き込みレイテンシは通常1サイクルです(キャッシュがメモリにアクセスする場合はレイテンシは増加)。C_DCACHE_USE_WRITEBACKが0に設定されている場合、書き込みレイテンシは通常2サイクルです(メモリコントローラーのポステッド書き込みバッファーがフルの場合はレイテンシは増加)。

MicroBlazeの命令およびデータのキャッシュは、4、8、または16ワードのキャッシュラインを使用するよう設定できます。長いキャッシュラインを使用するとより多くのバイトがフェッチされるので、一般的には順次アクセスパターンのソフトウェアでパフォーマンスが向上します。ただし、アクセスパターンがランダムなソフトウェアでは、パフォーマンスが低下する可能性があります。これは、利用可能なキャッシュライン数が少ないためにキャッシュヒット率が低くなることが原因です。

さまざまなメモリインターフェイスの詳細は、[第3章「MicroBlaze信号インターフェイスの説明」](#)を参照してください。

特権命令

次のMicroBlaze命令は特権命令です。

- GET、GETD、PUT、PUTD(明示的に許可されている場合を除く)
- WIC、WDC
- MTS、MTSE
- MSRCLR、MSRSET(Cビットのみが影響を受ける場合を除く)
- BRK
- RTID、RTBD、RTED
- BRKI(物理アドレス C_BASE_VECTORS + 0x8 または C_BASE_VECTORS + 0x18 にジャンプする場合を除く)
- SLEEP、HIBERNATE、SUSPEND
- LBUEA、LHUEA、LWEA、SBEA、SHEA、SWEA(明示的に許可されている場合を除く)

ユーザー モードでこれらの命令を実行しようとすると、特権命令例外が発生します。パラメーター C_MMU_PRIVILEGED_INSTR を1または3に設定すると、命令 GET、GETD、PUT、およびPUTDは特権命令とはみなされず、ユーザー モードで実行できます。



注意: このようにするとアプリケーションのプロセスが互いに干渉し合うことになるので、パフォーマンス上の理由でどうしても必要な場合以外はこの設定は使用しないでください。

パラメーター C_MMU_PRIVILEGED_INSTR を 2 または 3 に設定すると、拡張アドレス命令 LBUEA、LHUEA、LWEA、SBEA、SHEA、および SWEA は特権命令とはみなされず、MMU 変換がバイパスされて、拡張アドレスが物理アドレスとして扱われます。これは、物理アドレス空間全体に直接アクセスできるようにしながらソフトウェアを仮想モードで実行する場合に便利ですが、アプリケーションプロセス間の保護が必要な場合にはお勧めしません。

ユーザー モードおよび仮想モードから抜けるには、次の 6 つの方法があります。

1. ハードウェア生成リセット (デバッグ リセットを含む)
2. ハードウェア例外
3. マスク不可能なブレークまたはハードウェアブレーク
4. 割り込み
5. ユーザー ベクター例外を実行するため 「BRALID Re,C_BASE_VECTORS + 0x8」 を実行
6. ソフトウェアブレーク命令 BRKI を実行して、物理アドレス C_BASE_VECTORS + 0x8 または C_BASE_VECTORS + 0x18 にジャンプ

ハードウェア生成リセットを除く上記すべての方法で、ユーザー モードおよび仮想モード ステータスは MSR UMS および VMS ビットに保存されます。

アプリケーション (ユーザー モード) プログラムは、BRALID または BRKI 命令を使用してシステム サービス ルーチン (特権モード プログラム) に制御を移行し、物理アドレス C_BASE_VECTORS + 0x8 にジャンプします。この命令を実行すると、システム コール例外が発生します。例外ハンドラーは、呼び出すシステム サービス ルーチンを判断し、呼び出し元のアプリケーションにそのサービスを呼び出す権限があるかどうかを確認します。権限がある場合は、アプリケーションプログラムの代わりに例外ハンドラーがシステム サービス ルーチンへの実際のプロシージャコールを実行します。

システム サービス ルーチンの実行環境を設定するには、プロローグ命令を実行する必要があります。これらの命令は通常、プロシージャ情報 (アクティベーション レコード) の保持、ポインターのアップデートおよび初期化、揮発性レジスタ (システム サービス ルーチンが使用するレジスタ) の保存を実行するストレージを作成します。プロローグ コードは、実行可能なモジュールを作成するときにリンクで挿入するか、システム コール割り込みハンドラーまたはシステム ライブラリ ルーチンにスタブ コードとして含めることができます。

システム サービス ルーチンからのリターンは、上記のプロセスの逆になります。エピローグ コードは、アクティベーション レコードをアンワインドして割り当てを解放し、ポインターを復元して、揮発性レジスタを復元します。割り込みハンドラーは、アプリケーションに戻るために例外命令 (RTED) からのリターンを実行します。

仮想メモリ管理

MicroBlaze で実行しているプログラムは、32 ビット MicroBlaze では有効アドレスを使用してフラットな 4 GB アドレス空間にアクセスし、64 ビット MicroBlaze では C_ADDR_SIZE パラメーターの設定によって 16 EB までのアドレス空間にアクセスします。

プロセッサは、変換モードによって、このアドレス空間を次のいずれかの方法で解釈します。

- リアル モードでは、有効アドレスは物理メモリに直接アクセスするために使用されます。
- 仮想モードでは、プロセッサの仮想メモリ管理ハードウェアにより、有効アドレスは物理アドレスに変換されます。

仮想モードでは、システム ソフトウェアでプログラムおよびデータを物理アドレス空間の任意の位置に移動できます。アクティブなプログラムおよびデータに空間が必要になると、システム ソフトウェアが非アクティブなプログラムおよびデータを物理メモリから取り出します。

この移動により、実際にシステムにインプリメントされているよりも多くのメモリが存在しているようにプログラムには見えます。システムにある物理メモリ量の制限内で、プログラムをうまく実行できるようになります。どの物理メモリアドレスがほかのソフトウェア プロセスやハードウェア デバイスに割り当てられているかをプログラムが知っておく必要はありません。プログラムが認識できるアドレスは、プロセッサにより適切な物理アドレスに変換されます。

仮想モードでは、メモリ保護をさらに制御できます。1 KB のメモリのブロックでも個別に不正アクセスから保護できます。メモリの保護と移動により、システム ソフトウェアでマルチタスク機能をサポートできます。この機能により、複数のプログラムは同時またはほぼ同時に近い状態で実行しているように見えます。

MicroBlaze では、`C_USE_MMU` が 3(仮想)、`C_AREA_OPTIMIZED` が 0(パフォーマンス) または 2(周波数) に設定されているときに、メモリ管理ユニット (MMU) によって仮想モードがインプリメントされます。MMU は、有効アドレスを物理アドレスにマップし、メモリ保護をサポートします。これらの機能を使用することにより、システム ソフトウェアはデマンド ページ仮想メモリおよびほかのメモリ管理機能をインプリメントできます。

MicroBlaze の MMU のインプリメントは PowerPC™ 405 プロセッサをベースにしています。

MMU の機能は次のようにまとめられます。

- 有効アドレスを物理アドレスに変換
- アドレス変換中のページ レベルのアクセスを制御
- ゾーンを使用した追加の仮想モード保護
- 命令/アドレス、データ/アドレスの変換および保護を個別に制御
- 1 kB、4 kB、64 kB、1 MB、4 MB、16 MB の 8 つのページ サイズをサポート。ページ サイズのどの組み合わせでもシステム ソフトウェアで使用できます。
- ページ置き換えストラテジをソフトウェアで制御

リアル モード

プロセッサがメモリをフェッチし、ロードまたはストア命令でデータにアクセスするとき、プロセッサはメモリを参照します。プログラムは、32 ビット MicroBlaze ではプロセッサで計算される 32 ビットの有効アドレス、64 ビット MicroBlaze では 64 ビットの有効アドレスを使用してメモリ ロケーションを参照します。

リアル モードがイネーブルの場合、物理アドレスは有効アドレスと同じになり、プロセッサはそれを使用して物理メモリにアクセスします。プロセッサをリセットすると、プロセッサはリアル モードで動作します。リアル モードは、MSR の VM ビットをクリアして、イネーブルにすることもできます。

物理メモリのデータ アクセス(ロードおよびストア)は、有効アドレスを使用してリアル モードで実行されます。リアル モードでは、システム ソフトウェアで仮想アドレス変換はできませんが、`C_USE_MMU > 1`(ユーザー モード)および`C_AREA_OPTIMIZED = 0`(パフォーマンス) または 2(周波数) のときに、フルメモリアクセス保護がインプリメントされて使用可能になります。リアル モードのメモリマネージャーのインプリメンテーションは、仮想モードよりも簡単です。リアル モードは、アクセス保護が必要で仮想アドレス変換は不要な場合に、単純なエンベデッド環境でのメモリ管理のソリューションとして適切です。

仮想モード

仮想モードでは、図 2-18 で説明されているプロセスを使用し、プロセッサが有効アドレスを物理アドレスに変換します。64 ビット MicroBlaze または物理アドレス拡張 (PAE) を使用する場合は、物理アドレスを 64 ビットまで拡張できます。仮想モードは、MSR の VM ビットをセットするとインペーブルにできます。

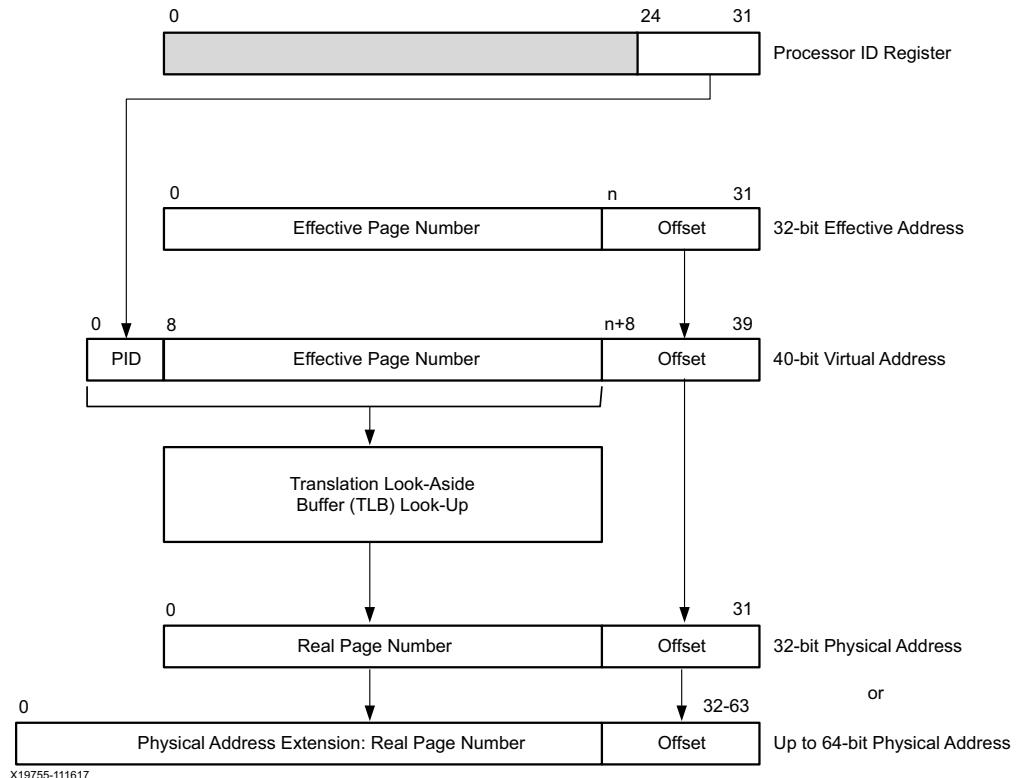


図 2-18: 仮想モードのアドレス変換

図 2-18 にある各アドレスには、ページ番号フィールドとオフセットフィールドがあります。ページ番号は、MMUによって変換されたアドレス部分です。オフセットは、ページへのバイトオフセットで、MMUで変換されていません。仮想アドレスには、プロセス ID (PID) と呼ばれる追加フィールドがあり、PID レジスタからの情報が配置されます (プロセス ID レジスタの詳細は 39 を参照)。PID と有効ページ番号 (EPN) の組み合わせは、仮想ページ番号 (VPN) と呼ばれます。値 n は、表 2-39 に示すように、ページサイズによって決まります。

各仮想ページを物理ページに変換するために使用されるエントリを含むページ変換テーブルは、システムソフトウェアが管理します。ページ変換エントリで定義されるページサイズにより、ページ番号およびオフセットフィールドのサイズが決まります。32 ビット MicroBlaze では、たとえば 4 kB のページサイズが使用される場合、ページ番号フィールドは 20 ビットで、オフセットフィールドは 12 ビットになります。この場合の VPN は 28 ビットです。

その後、最も頻繁に使用されるページ変換が、変換ルックアサイド バッファー (TLB) に格納されます。仮想アドレスを変換する際、MMU は一致する VPN (PID および EPN) をページ変換エントリから検索します。テーブルのすべてのエントリを検索するのではなく、プロセッサ TLB に含まれているエントリのみを検索します。一致する VPN のあるページ変換エントリが検出されると、そのエントリから対応する物理ページ番号が読み出され、オフセットと組み合わせて物理アドレスに変換されます。この物理アドレスは、プロセッサがメモリを参照するときに使用されます。

システム ソフトウェアは、PID を使用して、プロセッサ上で実行しているソフトウェアプロセス(タスク、サブルーン、スレッド)を特定できます。個別にコンパイルされているプロセスは、オーバーラップする有効アドレス領域で動作する可能性があります。マルチタスク機能がサポートされている場合は、このオーバーラップをシステム ソフトウェアで解決する必要があります。PID を各プロセスに割り当てると、各プロセスを仮想アドレス空間の固有の領域に移動して、オーバーラップを解決できます。仮想アドレス空間のマップにより、各プロセスを物理アドレス空間に個別に変換できます。

ページ変換テーブル

ページ変換テーブルは、ソフトウェアで定義および管理されるページ変換情報を含むデータ構造です。ソフトウェア管理のページ変換が必要であるため、エンベデッド システム アプリケーションをターゲットにしたアーキテクチャのトレードオフが発生します。エンベデッド システムには、厳密に管理された環境と、詳細に定義されたアプリケーション ソフトウェアのセットがあることが多く、そのような環境では、次の方法を使用して、各エンベデッド システム用に仮想メモリ管理を最適化できます。

- ページ変換エントリをすばやく検索できるように、ページテーブル検索のパフォーマンスを最大限にするようにページ変換テーブルを構成できます(テーブル ウォークとも呼ばれる)。最も汎用なプロセッサには、インデックス ページテーブル(検索方法が簡単でページテーブルサイズが大きい)か、ハッシュ テーブル(検索方法複雑でページテーブルサイズが小さい)のいずれかがインプリメントされています。ソフトウェア テーブル ウォークでは、特定のエンベデッド システムに合ったどんな組み合わせのテーブル構成でも使用できます。ページテーブルサイズおよびアクセス時間の両方を最適化できます。
- アプリケーション モジュール、デバイスドライバー、システム サービス ルーチン、およびデータに対して個別のページサイズを使用できます。個別のページサイズを選択することで、フラグメンテーション(未使用的メモリ)を低減することができ、システム ソフトウェアが効率よくメモリを使用できます。たとえば、大きなデータ構造を 16 MB のページに、小さな I/O デバイス ドライバーを 1 KB ページに割り当てるすることができます。
- ページ変換が見つからないケースを最小限に抑えるように、ページ置き換えを調節できます。この後のセクションで説明するように、最も頻繁に使用されるページ変換は、変換ルックアサイド バッファー(TLB)に格納されます。

どの変換を TLB に格納するか、新しい変換が必要な場合にどの変換を置き換えるかを判断するのはソフトウェアです。ページ変換エントリが頻繁に TLB に配置されたり TLB から取り出されたりするスラッシングを回避するように、置き換えストラテジを調整できます。また、クリティカルなページ変換が置き換えられるのを防ぐように調整することもできます。これはページロッキングとも呼ばれるプロセスです。

統合された 64 エントリの TLB は、ソフトウェアにより管理され、MMU でアクセスできる命令およびデータのページ変換エントリのサブセットをキャッシュします。システム メモリのページ変換テーブルからエントリを読み出し、TLB にそれを格納するのはソフトウェアです。次のセクションで、統合 TLB について詳しく説明します。内部的には、命令およびデータ用にシャドウ TLB も含まれ、それぞれ C_MMU_ITLB_SIZE および C_MMU_DTLB_SIZE でサイズを設定できます。

これらのシャドウ TLB は、完全にプロセッサで管理され(ソフトウェアには透過的)、統合 TLB とのアクセス競合を最小限に抑えるために使用されます。

変換ルックアサイド バッファー

変換ルックアサイド バッファー (TLB) は、プロセッサを仮想モードで実行しており、メモリ保護およびストレージ制御が使用されている場合に、MicroBlaze の MMU によってアドレス変換のために使用されます。TLB 内の各エントリには、仮想ページを特定し (PID および有効ページ番号)、物理ページに変換し、ページの保護特性を判断し、ページに関連付けられているストレージ属性を指定するために必要な情報が含まれています。

MicroBlaze の TLB は、物理的に次の 3 つの TLB としてインプリメントされます。

- 統合 TLB (UTLB): 64 個のエントリが含まれ、擬似連想型です。命令ページおよびデータページの変換はどの UTLB にも格納できます。UTLB はソフトウェアによって初期化および管理されます。
- 命令シャドウ TLB (ITLB): 命令ページ変換エントリが含まれ、完全連想型です。ITLB に格納されるページ変換エントリは、UTLB から最近アクセスされた命令ページ変換です。ITLB は、命令変換と UTLB アップデート操作との間の競合を最小限に抑えるために使用されます。ITLB はハードウェアによって初期化および管理され、ソフトウェアには透過的です。
- データシャドウ TLB (DTLB): データページ変換エントリが含まれ、完全連想型です。DTLB に格納されるページ変換エントリは、UTLB から最近アクセスされたデータページ変換です。DTLB は、データ変換と UTLB アップデート操作との間の競合を最小限に抑えるのに使用されます。DTLB はハードウェアによって初期化および管理され、ソフトウェアには透過的です。

次の図に、TLB の変換フローを示します。

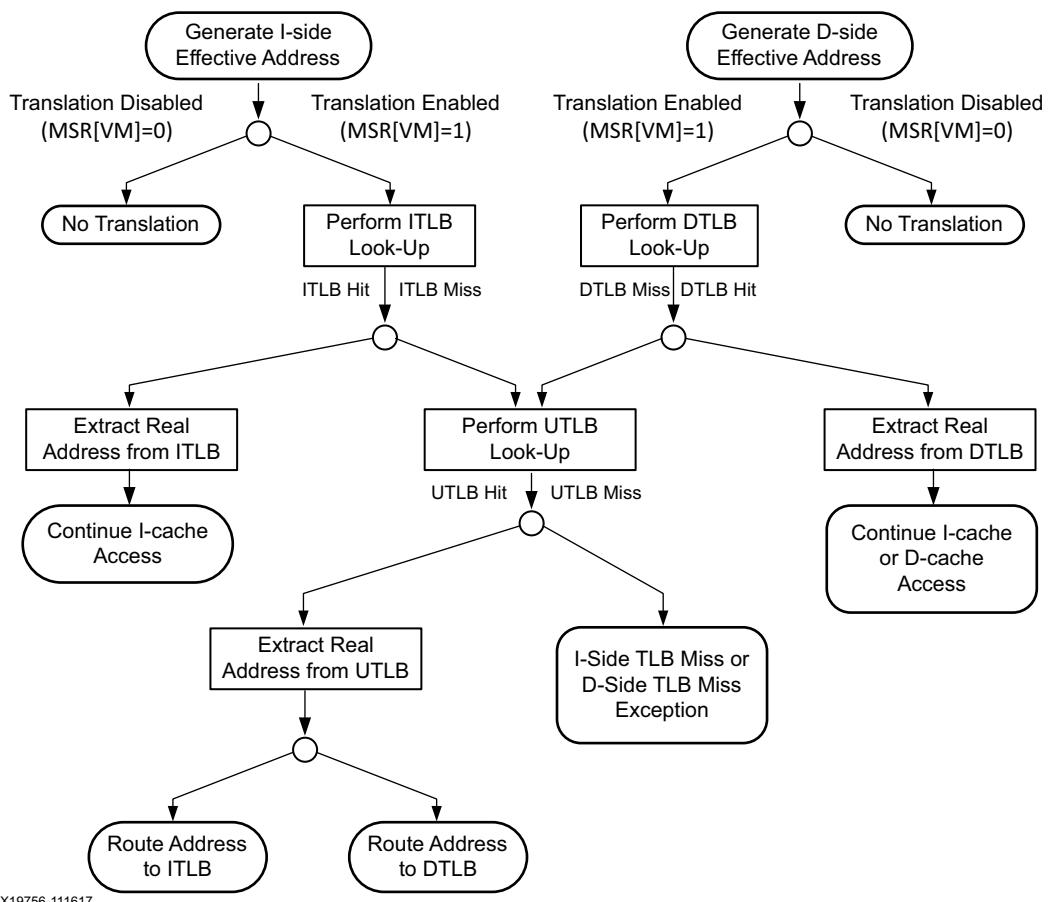
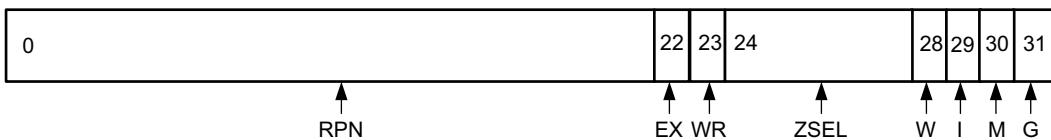


図 2-19: TLB アドレス変換フロー

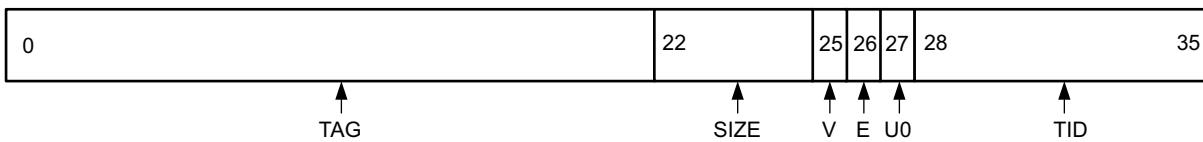
TLB エントリ フォーマット

次の図に、TLB エントリのフォーマットを示します。各 TLB エントリは 68 ~ 100 ビットの範囲で、TLBLO (データエントリとも呼ばれる) および TLBHI (タグエントリとも呼ばれる) という 2 部構成になっています。

TLBLO:



TLBHI:



X19757-111617

図 2-20: TLB エントリ フォーマット (PAE ディスエーブル)

64 ビット MicroBlaze または物理アドレス拡張 (PAE) がイネーブルの場合は、TLB エントリが TLBLO RPN フィールドで 32 ビットまで拡張され、64 ビットまでの物理アドレスがサポートされます。

[表 2-21](#) および [表 2-22](#) に、TLB エントリの内容および PAE または 64 ビット MicroBlaze をイネーブルにした場合の TLBLO フォーマットを示します。

TLB エントリ内のフィールドは次のように分類されます。

- 仮想ページの識別 (TAG、SIZE、V、TID): ページ変換エントリを特定します。変換プロセス中に仮想ページ番号と比較されます。
- 物理ページの識別 (RPN、SIZE): 物理メモリの変換されたページを特定します。
- アクセス制御 (EX、WR、ZSEL): ページで許可されるアクセスタイプを指定し、不正アクセスからページを保護するのに使用されます。
- ストレージ属性 (W、I、M、G、E、U0): データキャッシュのキャッシングポリシー (ライトバックまたはライトスルー)、ページがキャッシング可能かどうか、バイトの順序 (エンディアンネス) などのストレージ制御属性を指定します。

[表 2-39](#) に、TLB エントリの SIZE フィールドと変換されたページサイズとの関係を示します。この表には、タグ比較にどのアドレスビットが関与するかがページサイズによりどのように決定されるか、どのアドレスビットがページオフセットとして使用されるか、物理ページ番号のどのビットが物理アドレスで使用されるかも説明されています。64 ビット MicroBlaze または PAE をイネーブルにすると、物理アドレスの上位ビットは拡張された RPN フィールドから直接取り出されます。

表 2-39: ページ変換ビット範囲(ページ サイズ別)

ページ サイズ	SIZE TLBHI フィールド	タグ比較 ビット範囲 ¹	ページ オフセット	PAE ディスエーブル		PAE または 64 ビット イネーブル ²	
				物理ページ 番号	RPN ビット (0 にクリア)	物理ページ番号	RPN ビット (0 にクリア)
1 KB	000	TAG および Address[0:n-11]	Address[22:31]	RPN[0:21]	-	RPN[0:n-11]	-
4 KB	001	TAG および Address[0:n-13]	Address[20:31]	RPN[0:19]	20:21	RPN[0:n-13]	n-12:n-11
16 KB	010	TAG および Address[0:n-15]	Address[18:31]	RPN[0:17]	18:21	RPN[0:n-15]	n-14:n-11
64 KB	011	TAG および Address[0:n-17]	Address[16:31]	RPN[0:15]	16:21	RPN[0:n-17]	n-16:n-11
256 KB	100	TAG および Address[0:n-19]	Address[14:31]	RPN[0:13]	14:21	RPN[0:n-19]	n-18:n-11
1 MB	101	TAG および Address[0:n-21]	Address[12:31]	RPN[0:11]	12:21	RPN[0:n-21]	n-20:n-11
4 MB	110	TAG および Address[0:n-23]	Address[10:31]	RPN[0:9]	10:21	RPN[0:n-23]	n-22:n-11
16 MB	111	TAG および Address[0:n-25]	Address[8:31]	RPN[0:7]	8:21	RPN[0:n-25]	n-24:n-11

1. 64 ビット MicroBlaze の場合はビット インデックスは $n = \text{C_ADDR_SIZE}$ で、それ以外の場合は 32 です。

2. ビット インデックス $n = \text{C_ADDR_SIZE}$

TLB アクセス

MMU が仮想アドレス (PID と有効アドレスの組み合わせ) を物理アドレスに変換する際、まず該当するシャドウ TLB でページ変換エントリを検索します。エントリが見つかれば、それが物理メモリにアクセスするために使用されます。エントリが見つからなければ、次に UTLB を検索します。シャドウ TLB ミスのために UTLB にアクセスするたびに遅延が発生します。このミスレイテンシは 2 ~ 32 サイクルの範囲です。DTLB と ITLB の両方が UTLB に同時にアクセスする場合、DTLB のほうが ITLB よりも優先されます。

図 2-21 に、シャドウ TLB の 1 つまたは UTLB のページ変換エントリを検索する際の MMU の論理プロセスを示します。TLB のすべての有効エントリがチェックされます。

TLB エントリで次の条件がすべて満たされると、TLB ヒットになります。

- エントリが有効である
- エントリの TAG フィールドが有効アドレス EPN と一致する (エントリの SIZE フィールドの値に従う)
- エントリの TID が PID と一致する

上記の条件のいずれかが満たされていないと、TLB ミスになります。TLB ミスが発生すると、次のように例外が発生します。

TID の値が 0x00 の場合、MMU で TID と PID の比較が無視され、TAG および EA[EPN] のみが比較されます。

TID=0x00 の TLB エントリは、プロセスに依存しない変換を表します。すべてのプロセスがグローバルにアクセスするページには、TID に 0x00 を割り当てる必要があります。PID の値が 0x00 の場合、すべてのページにもアクセスできるプロセスは特定されません。PID=0x00 の場合、TID=0x00 のときのみページ変換ヒットが発生します。

EA[EPN] と PID のある組み合わせに一致するエントリが複数ある TLB をソフトウェアがロードすることは可能ですが、これはプログラミング エラーとみなされ、動作が未定義になります。

ヒットが発生すると、MMU は対応する TLB エントリから RPN フィールドを読み出します。SIZE フィールドの値により、このフィールドの一部またはすべてのビットが使用されます (表 2-39 を参照)。

たとえば、PAE がディスエーブルの 32 ビット MicroBlaze で SIZE フィールドが 256 kB のページ サイズを指定する場合、RPN[0:13] は物理ページ番号を表し、物理アドレスを形成するのに使用されます。TLB エントリを初期化するときは、RPN[14:21] は使用されず、ソフトウェアでこれらのビットを 0 にクリアする必要があります。物理アドレスの残りの部分は EA のページ オフセット部分から取られます。ページ サイズが 256 kB の場合、有効アドレスのビット 14:31 と RPN[0:13] を連結して、32 ビットの物理アドレスが形成されます。

PAE がイネーブルで、物理アドレス サイズが 40 ビット (C_ADDR_SIZE を 40 に設定) の場合、RPN[0:21] は物理ページ番号を示し、RPN[22:29] は使用されません。有効アドレスのビット 14:31 と RPN[0:21] を連結して、40 ビットの物理アドレスが形成されます。

物理メモリにアクセスする前に、MMU は TLB エントリ アクセス制御フィールドを確認します。これらのフィールドは、現在実行中のプログラムが要求されたメモリ アクセスを実行できるかどうかを示します。

アクセスが許可された場合、MMU はページにアクセスする方法を決定するため、ストレージ属性フィールドをチェックします。ストレージ属性フィールドは、メモリ アクセスのキャッシング ポリシーを指定します。

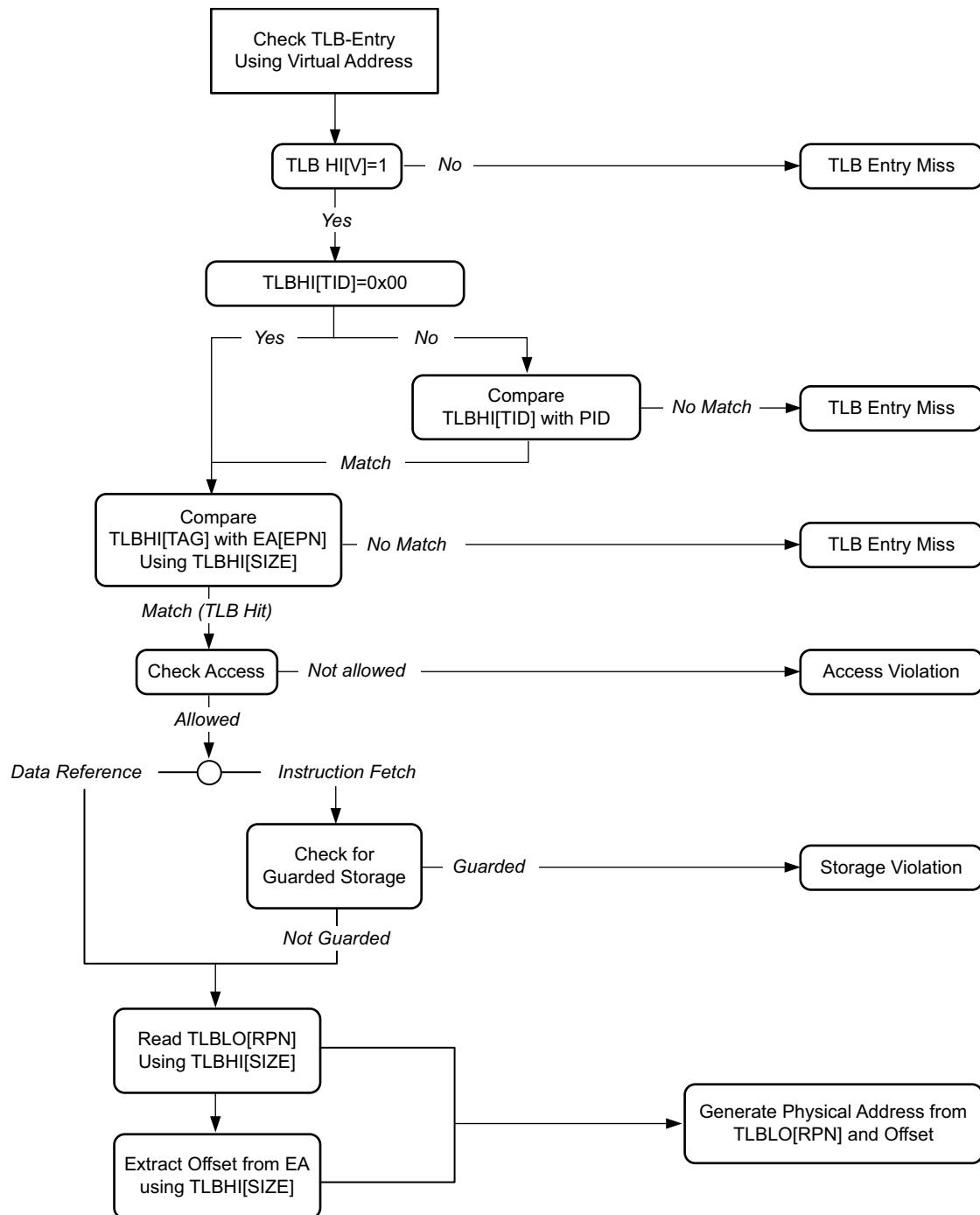
TLB アクセス エラー

TLB アクセス エラーがあると、例外が発生します。その場合、エラーを発生させた命令の実行が中断され、エラーを解決するため、制御が割り込みハンドラーに渡されます。TLB アクセス エラーは、次の 2 つの理由から発生します。

- 一致する TLB エントリが見つからず、TLB ミスになる
- 一致する TLB エントリが見つかったが、ストレージ属性またはゾーン保護によりページにアクセスできない

割り込みが発生すると、プロセッサが MSR[VM] を 0 にクリアして、リアルモードに入ります。リアルモードでは、MMU で実行されるすべてのアドレス変換およびメモリ保護チェックがディスエーブルになります。システム ソフトウェアが UTLB をページ変換エントリで初期化した後、MicroBlaze の UTLB は通常、リアルモードで実行している割り込みハンドラーを使用して管理されます。

次の図に、TLB エントリを検索する一般的なプロセスを示します。



X19758-111617

図 2-21: TLB エントリの検索プロセス

次のセクションでは、TLB アクセス エラーが原因で例外が発生する条件を説明します。

データストレージ例外

仮想モードがイネーブルの場合 (MSR[VM]=1)、次のいずれかの理由でページへのアクセスが許可されないときに、データストレージ例外が発生します。

- ユーザー モード:
 - TLB エントリでページへのアクセスを妨げるゾーン フィールドが指定されている (ZPR[Zn]=00)。これはロードおよびストア命令で発生します。
 - TLB エントリでゾーン フィールド (ZPR[Zn], 11) で上書きされない読み出し専用ページ (TLBLO[WR]=0) が指定されている。これはストア命令で発生します。
- 特権モード:
 - TLB エントリでゾーン フィールド (ZPR[Zn], 10 および ZPR[Zn], 11) で上書きされない読み出し専用ページ (TLBLO[WR]=0) が指定されている。これはストア命令で発生します。

命令ストレージ例外

仮想モードがイネーブルの場合 (MSR[VM]=1)、次のいずれかの理由でページへのアクセスが許可されないときに、命令ストレージ例外が発生します。

- ユーザー モード:
 - TLB エントリでページへのアクセスを妨げるゾーン フィールドが指定されている (ZPR[Zn]=00)。
 - TLB エントリでゾーン フィールド (ZPR[Zn], 11) で上書きされない実行不可能なページ (TLBLO[EX]=0) が指定されている。
 - TLB エントリで保護されたストレージページ (TLBLO[G]=1) が指定されている。
- 特権モード:
 - TLB エントリでゾーン フィールド (ZPR[Zn], 10 および ZPR[Zn], 11) で上書きされない実行不可能なページ (TLBLO[EX]=0) が指定されている。
 - TLB エントリで保護されたストレージページ (TLBLO[G]=1) が指定されている。

データ TLB ミス例外

仮想モードがイネーブルの場合 (MSR[VM]=1)、TLB (シャドウおよび UTLB) に有効で一致する TLB エントリが見つからないと、データ TLB ミス例外が発生します。データ TLB ミス例外は、どのロードまたはストア命令でも発生する可能性があります。

命令 TLB ミス例外

仮想モードがイネーブルの場合 (MSR[VM]=1)、TLB (シャドウおよび UTLB) に有効で一致する TLB エントリが見つからないと、命令 TLB ミス例外が発生します。命令 TLB ミス例外は、どの命令フェッチでも発生する可能性があります。

アクセス保護

システム ソフトウェアでは、不正アクセスからメモリ ロケーションを保護するためアクセス保護を使用します。システム ソフトウェアでは、ユーザー モードでも特権モードでも、メモリ アクセスを制限できます。制限は、読み出し、書き込み、および命令 フェッチに設定できます。アクセス保護は、仮想保護モードがイネーブルのときに使用可能です。

アクセス制御は、命令 フェッチ、データ ロード、およびデータ 格納に適用されます。仮想ページの TLB エントリで、ページに許可されているアクセス タイプを指定します。

また、TLB エントリで指定されるアクセス制御を上書きするのに使用されるゾーン保護レジスタにあるゾーン保護 フィールドも指定します。

TLB アクセス保護制御

各 TLB エントリは次の 3 タイプのアクセスを制御します。

- プロセス:** 各プロセスにプロセス ID (PID) を割り当てることにより、プロセスを不正アクセスから保護します。システム ソフトウェアがユーザー モード アプリケーションを開始すると、そのアプリケーションの PID が PID レジスタに読み込まれます。アプリケーションが実行されると、変換ルックアサイド バッファーハイ (TLBHI) の TID フィールドがその PID に一致する TLB エントリのみを使用して、メモリアドレスが変換されます。これにより、システム ソフトウェアで、仮想メモリの特定エリアにアプリケーションのアクセスを制限できます。TID=0x00 の TLB エントリは、プロセスに依存しない変換を表します。すべてのプロセスがグローバルにアクセスするページには、TID に 0x00 を割り当てる必要があります。
- 実行:** 実行可能に指定されている仮想ページ (TLBLO[EX]=1) から命令がフェッチされる場合のみ、プロセッサは命令を実行します。TLBLO[EX] を 0 にクリアすると、ページからフェッチされた命令は実行されなくなり、命令ストレージ割り込み (ISI) は発生しません。ISI は命令がフェッチされたときには発生しませんが、命令が実行されたときに発生します。これにより、後で実行されずに破棄される投機的にフェッチされた命令が ISI を発生させるのを回避できます。

ゾーン保護レジスタは実行保護を上書きできます。

- 読み出し/書き込み:** データは書き込み可能に指定されている仮想ページ (TLBLO[WR]=1) にのみ書き込まれます。TLBLO[WR] を 0 にクリアすると、ページは読み出し専用になります。読み出し専用ページに書き込みを実行しようとすると、データストレージ割り込み (DSI) が発生します。

ゾーン保護レジスタは書き込み保護を上書きできます。

TLB エントリを使用して、プログラムによるページの読み出しを防ぐことはできません。仮想モードでは、ゾーン保護を使用して、ページが読み出されないようにします。アクセスが許可されないゾーンを定義し (ZPR[Zn]=00)、それを使用して TLB エントリのアクセス保護を上書きします。ユーザー モードで実行しているプログラムのみがページ読み出しを実行できないようにすることができます。特権モードのプログラムには、常にページ読み出しアクセスがあります。

ゾーン保護

ゾーン保護は、TLB エントリで指定されているアクセス保護を上書きするのに使用されます。ゾーンは、任意の仮想ページに共通のアクセス保護を設定してグループ化します。ゾーンには、さまざまなページサイズを組み合わせて指定し、任意のページ数を含めることができます。ゾーンに含めるページは隣接したページである必要はありません。

ゾーン保護レジスタ (ZPR) は 32 ビットのレジスタで、16 個あるゾーンのそれぞれに適用される保護の上書きタイプを指定するのに使用されます。ゾーンの保護の上書きは、2 ビットフィールドとして ZPR にエンコードされます。

TLB エントリの 4 ビット ゾーン選択フィールド (TLBLO[ZSEL]) は、ZPR (Z0 ~ Z15) の 16 個のゾーンフィールドから 1 つを選択します。たとえば、ZSEL = 0101 はゾーン Z5 を選択します。

ZPR のゾーンフィールドを変更すると、そのゾーンのすべてのページの保護が上書きされます。ZPR を使用せずに保護を変更するには、そのゾーン内の各ページの変換エントリを個別に変更する必要があります。

インプリメントされていないゾーン (C_MMU_ZONES < 16 の場合) は、11 が含まれているものとして扱われます。

UTLB 管理

UTLB は、プロセッサの MMU とメモリ管理ソフトウェアとのインターフェイスの役割を果たします。システムソフトウェアが UTLB を管理し、仮想アドレスを物理アドレスに変換する方法を MMU に伝えます。変換がないことやアクセス違反のために問題が発生すると、MMU が例外を使用してシステムソフトウェアに問題が発生したこと伝えます。MMU がメモリ変換を進めることができるように、割り込みハンドラーを提供してこれらの問題を修正するのはシステムソフトウェアの役目です。

ソフトウェアは、MFS および MTS 命令を使用して、UTLB エントリを読み出しおよび書き込みします。PAE がイネーブルの場合は、実ページ番号の上位部分にアクセスするのに MFSE および MTSE 命令が使用されます。これらの命令は、TLBX レジスタ インデックス (0 ~ 63) を使用して UTLB の 64 個のエントリの 1 つを指定します。タグとデータは別々に読み出し/書き込みされるため、エントリに完全にアクセスするため、ソフトウェアは 2 つの MFS または MTS 命令、PAE がイネーブルの場合はさらに MFSE または MTSE 命令を実行する必要があります。

64 ビット MicroBlaze では、MFS および MTS 命令で UTLB エントリの内容全体にアクセスできます。

特定の変換を見つけ出すため、TLBSX レジスタを使用して UTLB が検索されます。TLBSX は有効アドレスを使用して変換を検索し、対応する UTLB インデックスを TLBX レジスタにロードします。

個々の UTLB エントリを無効にするには、MTS 命令を使用して TLB エントリのタグ部分の有効ビット (TLBHI[V]) をクリアします。

C_FAULT_TOLERANT が 1 に設定されている場合、UTLB のブロック RAM はパリティにより保護されます。パリティエラーが発生した場合は、TLB ミス例外が発生します。この場合にエラーが累積するのを回避するには、UTLB の各エントリを定期的に無効にする必要があります。

ページアクセスおよびページ変更の記録

仮想メモリのソフトウェア管理には、課題がいくつかあります。

- 仮想メモリ環境では、ソフトウェアおよびデータで使用されるメモリ量が、物理的に利用可能なメモリ容量を超えてしまうことがあります。そこで、ソフトウェアおよびデータのページが使用されていない場合は、その一部をハードドライブなどの外部物理メモリに格納する必要があります。理想的には、頻繁に使用されるページを物理メモリに残し、あまり使用されないページを別の場所に格納します。
- 物理メモリのページを、新しいページを読み込むために移動する場合は、移動する(古い)ページが変更されているかどうかを知ることが重要です。

変更されている場合は、新しいページを読み込む前に、古いページを保存する必要があります。古いページが変更されていない場合は、保存せずに新しいページを読み込むことができます。

- UTLBに格納されるページ変換の数は限られています。ここに格納できない変換はページ変換テーブルに格納する必要があります。ミスのために変換がUTLBにない場合、変換を読み込むことができるよう、システムソフトウェアでどのUTLBエントリを破棄するかを決定する必要があります。この場合、頻繁に使用される変換ではなく、あまり使用されない変換を移動するのが理想です。

この問題を効率よく解決するには、ページアクセスおよびページ変換を記録しておく必要があります。MicroBlazeでは、ハードウェアでページアクセスおよびページ変更は記録されません。代わりに、システムソフトウェアでTLBミス例外とデータストレージ例外を使用して、この情報を収集します。収集された情報は、ページ変換テーブルに関連付けられているデータ構造に格納できます。

ページアクセス情報は、どのページを物理メモリに置いておくか、物理メモリ空間が必要になったときにどのページを移動するかを決定するために使用されます。システムソフトウェアは、TLBエントリの有効ビット(TLBHI[V])を使用してページアクセスを監視します。このため、ページがアクセスされていないことを示すために、ページ変換を無効に初期化(TLBHI[V]=0)する必要があります。初めてページにアクセスしようとすると、UTLBエントリが無効になっているかまたはページ変換がUTLBにないため、TLBミス例外が発生します。TLBミスハンドラーは、有効な変換(TLBHI[V]=1)でUTLBをアップデートします。セットされた有効ビットは、ページおよびその変換がアクセスされたことを記録します。TLBミスハンドラーは、ページ変換エントリに関連付けられている別のデータ構造にもこの情報を記録できます。

ページ変更情報は、古いページを新しいページで上書きできるかどうか、古いページをまずハードディスクに格納する必要があるかどうかを示すために使用されます。システムソフトウェアは、TLBエントリの書き込み保護ビット(TLBLO[WR])を使用して、ページ変更を監視します。このため、ページが変更されていないことを示すために、ページ変換を読み出し専用に初期化(TLBLO[WR]=0)する必要があります。ページに既にアクセスがあり、前述のように有効になっている場合は、初めてデータをページに書き込もうとすると、データストレージ例外が発生します。ソフトウェアにページへの書き込み権限がある場合、データストレージハンドラーがそのページを書き込み可能に設定し(TLBLO[WR]=1)、戻ります。

書き込み保護ビットがセットされていることは、ページが変更されている記録となります。データストレージハンドラーは、この情報をページ変換エントリに関連付けられている別のデータ構造にも記録できます。

仮想モードに最初に入ったとき、新しいプロセスを開始したときに、ページ変更を記録しておくと有益です。

リセット、割り込み、例外、およびブレーク

MicroBlaze は、リセット、割り込み、ユーザー例外、ブレーク、およびハードウェア例外をサポートします。次のセクションでは、これらのイベントそれぞれに関連付けられている実行フローを説明します。

これらのイベントを優先順位の高いものからリストすると、次のようにになります。

1. リセット
2. ハードウェア例外
3. マスク可能でないブレーク
4. ブレーク
5. 割り込み
6. ユーザー ベクター(例外)

表 2-40 に、関連付けられているベクターのメモリアドレス ロケーションおよび戻りアドレスのハードウェアのレジスタ ファイル ロケーションを示します。各ベクターはアドレスを 2つ割り当てて、フルアドレス範囲での分岐を可能にします(IMM の後に BRAI 命令が必要)。通常、ベクターはアドレス 0 で開始しますが、パラメーター C_BASE_VECTORS を使用して、ベクターをメモリの任意の位置に配置できます。

0x28 ~ 0x4F までは、今後のソフトウェア サポートのため、ザイリンクスによって予約されています。ユーザー アプリケーションにこの範囲内のアドレスを割り当てるとき、今後の SDK ソフトウェア サポートで競合が発生する可能性が高くなります。

表 2-40: ベクターおよび戻りアドレス レジスタ ファイルのロケーション

イベント	ベクター アドレス	レジスタ ファイル 戻りアドレス
リセット	C_BASE_VECTORS + 0x0 - C_BASE_VECTORS + 0x4	-
ユーザー ベクター(例外)	C_BASE_VECTORS + 0x8 - C_BASE_VECTORS + 0xC	Rx
割り込み ¹	C_BASE_VECTORS + 0x10 - C_BASE_VECTORS + 0x14	R14
ブレーク: マスクできないハードウェア	C_BASE_VECTORS + 0x18 - C_BASE_VECTORS + 0x1C	R16
ブレーク: ハードウェア		
ブレーク: ソフトウェア		
ハードウェア例外	C_BASE_VECTORS + 0x20 - C_BASE_VECTORS + 0x24	R17 または BTR
今後のサポートのためザイリンクスにより予約	C_BASE_VECTORS + 0x28 - C_BASE_VECTORS + 0x4F	-

1. 低レインジ割り込みモードでは、ベクター アドレスは割り込みコントローラーにより供給されます。

セマフォやスピinnロックなど、相互排他機能をインプリメントするため LWX と SWX 命令と共に使用すると、これらのイベントはすべて予約ビットをクリアします。

リセット

Reset または Debug_Rst⁽¹⁾ が発生すると、MicroBlaze はパイプラインをフラッシュし、リセットベクター(アドレス C_BASE_VECTORS + 0x0)から命令をフェッチし始めます。両方の外部リセット信号はアクティブ High で、16 サイクル以上アサートする必要があります。MSR リセット値パラメーターの詳細は、第3章の「MicroBlaze コアのコンフィギュレーション」を参照してください。

等価擬似コード

```
PC ← C_BASE_VECTORS + 0x0
MSR ← C_RESET_MSR_IE << 2 | C_RESET_MSR_BIP << 4 | C_RESET_MSR_ICE << 6 |
    C_RESET_MSR_DCE << 8 | C_RESET_MSR_EE << 9 | C_RESET_MSR_EIP << 10
EAR ← 0; ESR ← 0; FSR ← 0
PID ← 0; ZPR ← 0; TLBX ← 0
Reservation ← 0
```

ハードウェア例外

無効な命令、命令およびデータバスエラー、非境界整列アクセスなどの内部エラー コンディションを捕捉するように MicroBlaze を設定できます。除算例外は、プロセッサにハードウェア除算器が設定されている (C_USE_DIV=1) 場合にのみイネーブルにできます。

ハードウェア浮動小数点ユニットが設定されている (C_USE_FPU>0) 場合は、アンダーフロー、オーバーフロー、浮動小数点のゼロ除算、無効操作、非正規化オペランドエラーなどの浮動小数点の例外も捕捉できます。

ハードウェアのメモリ管理ユニット (MMU) が設定されている場合は、無効な命令例外、データストレージ例外、命令ストレージ例外、データ TLB ミス例外、命令 TLB ミス例外などのメモリ管理例外も捕捉できます。

ハードウェア例外が発生すると、MicroBlaze はパイプラインをフラッシュし、ハードウェア例外ベクター(アドレス C_BASE_VECTORS + 0x20)に分岐します。例外サイクルの実行段の命令は実行されません。

例外は、汎用レジスタ R17 も次のようにアップデートします。

- MMU 例外の場合(データストレージ例外、命令ストレージ例外、データ TLB ミス例外、命令 TLB ミス例外)、レジスタ R17 に適切なプログラムカウンター値がロードされ、戻ったときに例外を発生させた命令を再実行します。その前に IMM 命令がある場合は、IMM 命令に戻るよう値が調節されます。例外が分岐遅延スロットの命令により発生した場合、その分岐命令(その前に IMM 命令がある場合は IMM 命令)に戻るよう値が調節されます。
- その他の例外が発生した場合は、例外が分岐遅延スロットの命令により発生している場合を除き、レジスタ R17 に後続の命令のプログラムカウンター値がロードされます。例外が分岐遅延スロットの命令により発生している場合は、ESR[DS] ビットがセットされます。この場合、例外ハンドラーは BTR に格納されている分岐先アドレスから実行を再開する必要があります。

MSR の EE および EIP ビットは、RTED 命令を実行するときに自動的に元に戻されます。

MSR の VM および UM ビットは、RTED、RTBD、および RTID 命令を実行するときに、自動的に VMS および UMS から元に戻されます。

1. リセット入力は、MDM を使用してデバッガーで制御されます。

例外の優先順位

複数の例外が同時に発生した場合は、優先順位の高いものから次の順序で処理されます。

- 命令バス例外
- 命令 TLB ミス例外
- 命令ストレージ例外
- 無効なオペコード例外
- 特権命令例外またはスタック保護違反例外
- データ TLB ミス例外
- データストレージ例外
- 非境界配列例外
- データバス例外
- 除算例外
- FPU 例外
- ストリーム例外

例外の原因

- ストリーム例外:** 制御ビットの不一致があるときに e ビットが 1 にセットされている get または getd を実行すると、AXI4-Stream 例外が発生します。
- 命令バス例外:** 命令バス例外は、メモリからデータを読み出すときのエラーにより発生します。
 - 命令ペリフェラル AXI4 インターフェイス (M_AXI_IP) 例外は、M_AXI_IP_RRESP でのエラー応答が原因で発生します。
 - 命令キャッシュペリフェラル AXI4 インターフェイス (M_AXI_IC) 例外は、M_AXI_IC_RRESP でのエラー応答が原因で発生します。この例外は、C_ICACHE_ALWAYS_USED が 1 に設定されていてキャッシュがオフのとき、または MMU 抑止キャッシングビットがアドレスに対して設定されている場合にのみ発生します。それ以外の場合、応答は無視されます。
 - 命令側ローカルメモリ (LMB) は、IUE 信号で示される LMB メモリで訂正不可能なエラーが発生したとき、または ICE 信号で示される C_ECC_USE_CE_EXCEPTION が 1 に設定されていて LMB メモリで訂正可能なエラーが発生したときにのみ、命令バス例外を発生させます。
- 無効なオペコード例外:** 無効のマジャー オペコード (命令のビット 0 ~ 5) で命令を実行すると、無効なオペコード例外が発生します。命令のビット 6 ~ 31 はチェックされません。オプションのプロセッサ命令がイネーブルになっていない場合、無効な命令として検出されます。また、オプション機能 C_OPCODE_0x0_ILLEGAL がイネーブルになっている場合、命令が 0x00000000 に等しいと、無効なオペコード例外が発生します。

- **データバス例外:** データバス例外は、メモリからデータを読み出すときまたはデータをメモリに書き込むときのエラーにより発生します。
 - データペリフェラル AXI4 インターフェイス (`M_AXI_DP`) は、`M_AXI_DP_RRESP` または `M_AXI_DP_BRESP` でのエラー応答が原因で発生します。
 - データキャッシュ AXI4 インターフェイス (`M_AXI_DC`) 例外は次が原因で発生します。
 - `M_AXI_DC_RRESP` または `M_AXI_DC_BRESP` のエラー応答。
 - `LWX` を使用した排他的アクセスの場合は、`M_AXI_DC_RRESP` の `OKAY` 応答。
- この例外は、`C_DCACHE_ALWAYS_USED` が 1 に設定されていてキャッシュがオフの場合、`LWX` または `SWX` がを使用して排他的アクセスがされているとき、または MMU 抑止キャッシングビットがアドレスに対してセットされている場合にのみ発生します。それ以外の場合、応答は無視されます。
- データ側ローカルメモリ (DLMB) は、`DUE` 信号で示される LMB メモリで訂正不可能なエラーが発生したとき、または `DCE` 信号で示される `C_ECC_USE_CE_EXCEPTION` が 1 に設定されていて LMB メモリで訂正可能なエラーが発生したときにのみ、データバス例外を発生させます。すべての読み出しアクセス、バイトおよびハーフバイトの書き込みアクセスで、エラーが発生する可能性があります。
- **非境界整列例外:** 32 ビット MicroBlaze では、非境界整列例外はデータバスへのアドレスで下位 2 ビットのいずれかがセットされているワードアクセス、または最下位ビットがセットされているハーフワードアクセスにより発生します。
- 64 ビット MicroBlaze では、非境界整列例外はデータバスへのアドレスで下位 3 ビットのいずれかがセットされている long アクセス、下位 2 ビットのいずれかがセットされているワードアクセス、または最下位ビットがセットされているハーフワードアクセスにより発生します。
- **除算例外:** 除算例外は、除数が 0 の整数除算 (`idiv` または `idivu`)、またはオーバーフローが発生する符号付き整数除算 (`idiv`) (-2147483648 / -1) により発生します。
- **FPU 例外:** FPU 例外は、アンダーフロー、オーバーフロー、ゼロ除算、無効な操作、または浮動小数点命令で発生する非正規オペランドにより発生します。
 - 結果が非正規化数の場合はアンダーフローが発生します。
 - 結果が非数 (NaN) の場合はオーバーフローが発生します。
 - `rB` が無限大でないときに、`fdiv` の `rA` オペランドがゼロの場合、ゼロ除算例外が発生します。
 - NaN オペランドを使用するか、または無効の無限大やゼロ オペランドの組み合わせによって、無効な操作が発生します。
- **特権命令例外:** ユーザー モードで特権命令を実行しようとすると、特権命令例外が発生します。
- **スタック保護違反例外:** 特殊なスタックローおよびスタックハイレジスタで定義されているスタック境界の範囲外にあるアドレスで、`rA` をスタックポインター(レジスタ R1)として使用したロードまたはストア命令を実行すると、スタック保護違反例外が発生し、スタックオーバーフローまたはスタックアンダーフローが発生します。
- **データストレージ例外:** メモリ保護違反になるメモリのデータにアクセスしようとすると、データストレージ例外が発生します。
- **命令ストレージ例外:** メモリ保護違反になるメモリの命令にアクセスしようとすると、命令ストレージ例外が発生します。
- **データ TLB ミス例外:** 有効な変換ルックアサイドバッファーエントリがなく、仮想保護モードがイネーブルになっているときに、データにアクセスしようとすると、データ TLB ミス例外が発生します。

- **命令 TLB ミス例外:** 有効な変換ロックアサイド バッファー エントリがなく、仮想保護モードがイネーブルになっているときに、命令にアクセスしようとすると、命令 TLB ミス例外が発生します。

C_FAULT_TOLERANT が 1 に設定されていて、例外が処理中 (MSR[EIP] がセットされ、MSR[EE] がクリアになる) のときに命令バス例外、無効なオペコード例外、またはデータバス例外が発生すると、パイプラインが停止し、外部信号 MB_Error がセットされます。

あいまい例外

通常、MicroBlaze の例外はすべて明確であり、パイプラインで例外を発生させた命令の後にある命令はすべて無効になり、影響しません。

C_IMPRECISE_EXCEPTIONS が 1 に設定されている場合 (ECC)、LMB メモリの ECC エラーによって発生する命令バス例外またはデータバス例外はあいまいになります。これを許容できる場合は、このパラメーターを 1 に設定することで最大周波数を向上させることができます。

等価擬似コード

```

ESR[DS] ← exception in delay slot
if ESR[DS] then
    BTR ← branch target PC
    if MMU exception then
        if branch preceded by IMM then
            r17 ← PC - 8
        else
            r17 ← PC - 4
    else
        r17 ← invalid value
else if MMU exception then
    if instruction preceded by IMM then
        r17 ← PC - 4
    else
        r17 ← PC
else
    r17 ← PC + 4
PC ← C_BASE_VECTORS + 0x20
MSR[EE] ← 0, MSR[EIP]← 1
MSR[UMS] ← MSR[UM], MSR[UM] ← 0, MSR[VMS] ← MSR[VM], MSR[VM] ← 0
ESR[EC] ← exception specific value
ESR[ESS]← exception specific value
EAR ← exception specific value
FSR ← exception specific value
Reservation ← 0

```

ブレーク

ブレークには次の 2 種類があります。

- ハードウェア (外部) ブレーク
- ソフトウェア (内部) ブレーク

ハードウェア ブレーク

ハードウェア ブレークは、外部信号(Ext_BRK および Ext_NM_BRK 入力ポート)をアサートすることにより実行されます。ブレークが実行されると、実行段の命令は完了しますが、デコード段の命令は、ブレーク ベクター(アドレス C_BASE_VECTORS + 0x18)への分岐に置き換えられます。

ブレークの戻りアドレス(ブレーク時のデコード段にある命令に関連付けられている PC)は、汎用レジスタ R16 に自動的にロードされます。MicroBlaze は、マシンステータス レジスタ (MSR) の処理中ブレーク (BIP) フラグも設定します。

標準のハードウェア ブレーク (Ext_BRK 入力ポート) は、MSR[BIP] および MSR[EIP] が 0 に設定されている(ブレークまたは処理中の例外がない)場合にのみ処理されます。処理中ブレーク フラグは割り込みをディスエーブルにします。マスク不可能なブレーク (Ext_NM_BRK 入力ポート) は常にすぐに処理されます。

MSR の BIP ビットは、RTBD 命令が実行されると自動的にクリアになります。

Ext_BRK 信号は、ブレークが発生するまでアサートし、RTBD 命令が実行される前にディアサートする必要があります。Ext_NM_BRK 信号は、1 クロック サイクルのみアサートする必要があります。

ソフトウェア ブレーク

ソフトウェア ブレークを実行するには、brk および brki 命令を使用します。ソフトウェア ブレークの詳細は、[第5章「MicroBlaze 命令セット アーキテクチャ」](#) を参照してください。

特殊なケースとして、C_DEBUG_ENABLED が 0 を超える値で brki rD,0x18 を実行する場合、C_BASE_VECTORS の値にかかわらず、ソフトウェア ブレーク ポイントがザイリンクス システム デバッガー (XSDB) などのデバッグツールに通知されます。この場合、MSR の BIP ビットは設定されません。

レイテンシ

ブレークが発生してから MicroBlaze プロセッサがブレーク サービス ルーチンに入るまでにかかる時間は、現在実行段にある命令と、ブレーク ベクターを格納するメモリまでのレイテンシによります。

等価擬似コード

```
r16 ← PC
PC ← C_BASE_VECTORS + 0x18
MSR[BIP] ← 1
MSR[UMS] ← MSR[UM], MSR[UM] ← 0, MSR[VMS] ← MSR[VM], MSR[VM] ← 0
Reservation ← 0
```

割り込み

MicroBlaze では外部割り込みソースが 1 つサポートされています(Interrupt 入力ポートに接続)。マシンステータス レジスタ (MSR) の割り込みイネーブル (IE) ビットが 1 に設定されている場合にのみ、プロセッサは割り込みに応答します。割り込みが発生すると、実行段の命令は完了しますが、デコード段の命令は割り込みベクターへの分岐に置き換えられます。このアドレスは、C_BASE_VECTORS + 0x10 であるか、低レイテンシ割り込みモードの場合は割り込みコントローラーで供給されるアドレスです。

割り込みの戻りアドレス(割り込み時のデコード段にある命令に関連付けられている PC)は、汎用レジスタ R14 に自動的にロードされます。さらに、プロセッサは、MSR の IE ビットをクリアしてその後の割り込みをディスエーブルにします。RTID 命令が実行されると、IE ビットは自動的に再びセットされます。

MSR の処理中ブレーク (BIP) または処理中例外 (EIP) が 1 にセットされていると、プロセッサは割り込みを無視します。

パラメーター C_INTERRUPT_IS_EDGE を使用すると、外部割込みをレベル センシティブまたはエッジ トリガーに設定できます。

- レベル センシティブの割り込みを使用する場合、MicroBlaze が割り込みを取って割り込みベクターにジャンプするまで、Interrupt 入力をセットしたままにする必要があります。割り込みハンドラーから戻る前に割り込みをクリアするには、ソフトウェアがソースで割り込みに対して肯定応答を送信する必要があります。そうでないと、割り込みハンドラーから戻ったときに割り込みがイネーブルなるとすぐにまた割り込みが処理されます。
- エッジ トリガーの割り込みを使用する場合、MicroBlaze は Interrupt 入力のエッジを検出してラッチするので、入力は 1 クロック サイクルだけアサートする必要があります。割り込み入力はアサートしたままにできますが、新しい割り込みを検出できるようにするには、その前に 1 クロック サイクル以上ディアサートしておく必要があります。エッジ トリガーの割り込みのラッチは、MSR の IE ビットからは独立しています。IE ビットが 0 のときに割り込みが発生した場合は、IE ビットが 1 になるとすぐに実行されます。

FIT Timer IP コアなどソフトウェアからの割り込みをクリアできないものを使用する場合は、エッジ トリガーの割り込みを使用することをお勧めします。

低レイテンシ割り込みモード

Interrupt_Address 入力ポートを使用して、割り込みコントローラーが各割り込みに対して直接割り込みベクターを供給できる低レイテンシモードを使用できます。割り込みシステムを初期化するとき、それぞれの高速割り込みハンドラーのアドレスを割り込みコントローラーに渡す必要があります。特定の割り込みが発生すると、割り込みアドレスは割り込みコントローラーによって供給され、MicroBlaze はハンドラーコードに直接ジャンプできます。

このモードでは、MicroBlaze は適切な割り込み肯定応答を割り込みコントローラーに直接送信 (Interrupt_Ack 出力ポートを使用) することができますが、ソースでレベル センシティブの割り込みに対する肯定応答を送信するのは割り込みサービス ルーチンの役目です。

この情報により、割り込みコントローラーは、レベル センシティブの割り込みとエッジ トリガーの割り込みの両方に対して、適切に割り込みを肯定応答できます。

割り込み処理イベントを割り込みコントローラーに通知するため、Interrupt_Ack が次のように設定されます。

- 01: MicroBlaze が割り込みハンドラーコードにジャンプしたとき。
- 10: RTID 命令が実行されて割り込みから戻ったとき。
- 11: MSR[IE] が 0 から 1 に変更され、割り込みが再びイネーブルになったとき。

Interrupt_Ack 出力ポートは、1 クロック サイクル間アクティブになり、その後 00 にリセットされます。

レイテンシ

割り込み発生時から MicroBlaze が割り込みサービス ルーチン (ISR) に入るまでにかかる時間は、プロセッサの設定と割り込みベクターを格納するメモリ コントローラーのレイテンシによります。MicroBlaze にハードウェア ドライバーが設定されている場合、除算命令の実行中に割り込みが発生するとレイテンシが最大になります。

低レイテンシ割り込みモードでは、各割り込みの割り込みベクターが割り込みコントローラーにより直接供給されるので、ISR に入る時間が大幅に短縮されます。高速割り込みにはコンパイラ サポートがあるので、共通 ISR は必要ありません。代わりに、各割り込みの ISR が直接読み出され、コンパイラが ISR で使用されるレジスタの保存と復元を実行します。

等価擬似コード

```
r14 ← PC
if C_USE_INTERRUPT = 2
    PC ← Interrupt_Address
    Interrupt_Ack ← 01
else
    PC ← C_BASE_VECTORS + 0x10
MSR[IE] ← 0
MSR[UMS] ← MSR[UM], MSR[UM] ← 0, MSR[VMS] ← MSR[VM], MSR[VM] ← 0
Reservation ← 0
```

ユーザー ベクター(例外)

ユーザー例外ベクターはアドレス 0x8 にあります。ユーザー例外は、ソフトウェアフローに「BRALID Rx,0x8」命令を挿入すると発生します。Rx で任意の汎用レジスタを指定できますが、ユーザー例外の戻りアドレスは R15 に格納し、ユーザー例外ハンドラーから戻るには RTSD 命令を使用することをお勧めします。

擬似コード

```
rx ← PC
PC ← C_BASE_VECTORS + 0x8
MSR[UMS] ← MSR[UM], MSR[UM] ← 0, MSR[VMS] ← MSR[VM], MSR[VM] ← 0
Reservation ← 0
```

命令キャッシュ

概要

LMB アドレス範囲外にあるコードを実行する際のパフォーマンスを改善するため、MicroBlaze にオプションの命令キャッシュを使用できます。

命令キャッシュの機能は、次のとおりです。

- 直接マップ(1 ウェイ アソシエイティブ)
- キャッシュ可能メモリ アドレス範囲を選択可能
- キャッシュおよびタグ サイズを設定可能
- AXI4 インターフェイスを介してキャッシュ(M_AXI_IC)
- 4、8、または 16 ワードのキャッシュ ラインを使用可能
- MSR のビットを使用してキャッシュのオン/オフを制御
- オプションの WIC 命令で命令キャッシュ ラインを無効化
- オプションのストリームバッファーで命令をプリフェッチしてパフォーマンスを改善
- オプションのビクティム キャッシュで追い出されたキャッシュ ラインデータを保存してパフォーマンスを改善
- オプションのparity 保護で、ブロック RAM ビット エラーが検出された場合にキャッシュ ラインを無効化
- オプションでデータ幅を 32 ビットにするか、キャッシュ ライン全体にするか、または 512 ビットにするかを選択

一般的な命令キャッシュの機能

命令キャッシュを使用すると、メモリアドレス空間はキャッシュ可能なセグメントとキャッシュ不可能なセグメントに分けられます。キャッシュ可能なセグメントは、C_ICACHE_BASEADDR および C_ICACHE_HIGHADDR という 2 つのパラメーターで決められます。この範囲内のアドレスはすべて、キャッシュ可能なアドレスセグメントに対応します。その他のアドレスはすべてキャッシュ不可能なセグメントです。

キャッシュ可能なセグメント サイズは 2^N である必要があります。N は正の整数です。C_ICACHE_BASEADDR および C_ICACHE_HIGHADDR で指定される範囲は、 2 のべき乗 ($=2^N$) で、C_ICACHE_BASEADDR の N 個の下位ビットはゼロである必要があります。

キャッシュ可能な命令アドレスは、キャッシュアドレスとタグアドレスの 2 つの部分で構成されています。MicroBlaze 命令キャッシュは、64 バイトから 64 kB までの範囲で設定できます。これは、6 から 16 ビットまでのキャッシュアドレスに対応しています。キャッシュアドレスとタグアドレスを合わせて、キャッシュ可能なメモリのフルアドレスと一致している必要があります。

2 kB 未満のキャッシュサイズを選択する場合、タグ RAM と命令 RAM をインプリメントするのに、分散 RAM が使用されます。パラメーター C_ICACHE_FORCE_TAG_LUTRAM が 1 に設定されている場合は、タグ RAM をインプリメントするのに分散 RAM が使用されます。このパラメーターは、4 ワードのキャッシュラインの場合は 8 kB 以下、8 ワードのキャッシュラインの場合は 16 kB 以下、16 ワードのキャッシュラインの場合は 32 kB 以下のキャッシュサイズでのみ使用できます。

たとえば、C_ICACHE_BASEADDR= 0x00300000、C_ICACHE_HIGHADDR=0x0030ffff、C_CACHE_BYTE_SIZE=4096、C_ICACHE_LINE_LEN=8、および C_ICACHE_FORCE_TAG_LUTRAM=0 が設定されている 32 ビット MicroBlaze では、64 kB のキャッシュ可能なメモリはバイトアドレスの 16 ビット、4 kB のキャッシュはバイトアドレスの 12 ビットのを使用するので、アドレスのタグ幅は $16-12=4$ ビットになります。このコンフィギュレーションに必要なブロック RAM プリミティブの総数は、1024 個の命令ワードを格納するのに RAMB16 が 2 つ、128 個のキャッシュラインエントリに RAMB16 が 1 つで、それぞれ、4 ビットのタグ、8 ワードの有効ビット、1 ライン有効ビットで構成されます。合計で 3 つの RAMB16 プリミティブが使用されます。

次の図に、命令キャッシュの構成を示します。

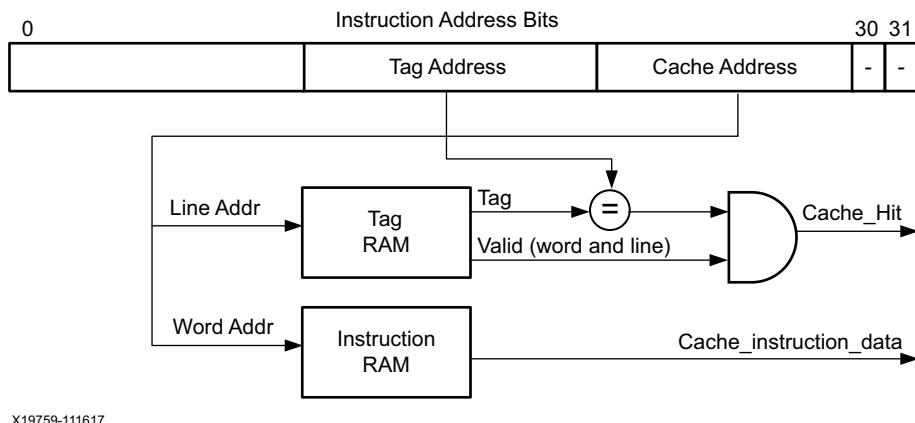


図 2-22: 命令キャッシュの構成

命令キャッシュ操作

フェッチされたすべての命令に対して、命令キャッシュで命令アドレスがキャッシュ可能なセグメントにあるかどうかが検出されます。アドレスをキャッシュできない場合、キャッシュコントローラーは命令を無視し、M_AXI_IP または LMB に要求を完了させます。アドレスをキャッシュできる場合は、要求されたアドレスが現在キャッシュされているかどうかをチェックするため、タグメモリでルックアップが実行されます。ワードおよびラインの有効ビットがセットされていて、タグアドレスと命令アドレスのタグセグメントが一致していれば、ルックアップは成功です。キャッシュミスの場合、キャッシュコントローラーは命令 AXI4 インターフェイス (M_AXI_IC) を介して新しい命令を要求し、メモリコントローラーから関連付けられているキャッシュラインが返されるまで待機します。

C_ICACHE_DATA_WIDTH はバスデータ幅を指定し、32 ビット、キャッシュライン全体 (128、256、または 512)、あるいは 512 ビットを選択します。

C_FAULT_TOLERANT が 1 に設定されている場合、タグまたは命令ブロック RAM でパリティエラーが検出されると、キャッシュミスも発生します。

32 ビットのデータ幅が選択されている場合、命令キャッシュは AXI4 インターフェイスに対してバーストアクセスを実行しますが、それ以外の場合はシングルアクセスになります。

ストリームバッファー

C_ICACHE_STREAMS を 1 に設定してストリームバッファーをイネーブルにした場合、ストリームバッファーがフルになるまで、最後に要求されたアドレスに続いて、投機的にキャッシュラインがあらかじめ順次フェッチされます。

ストリームバッファーはキャッシュラインを 2 つまで保持できます。その後、ストリームバッファーによりブリッジフェッチされたキャッシュラインからプロセッサが命令を要求すると (リニアコードで発生)、命令はすぐに使用可能になります。

ストリームバッファーをイネーブルにすると、プロセッサがメモリから命令をフェッチするのにかかる時間が短縮されるので、通常パフォーマンスが改善します。

C_ICACHE_DATA_WIDTH は、各クロックサイクルでストリームバッファーから転送されるデータ量を指定します (32 ビットまたはキャッシュライン全体のいずれかを指定)。

命令キャッシュストリームバッファーを使用できるようにするには、エリア最適化をイネーブルにしないでください。

ビクティムキャッシュ

C_ICACHE_VICTIMS を 2、4、または 8 に設定すると、ビクティムキャッシュがイネーブルになります。このパラメーターは、ビクティムキャッシュに格納できるキャッシュラインの数を定義します。キャッシュラインがキャッシュから追い出されると、ビクティムキャッシュに格納されます。最近のラインを格納しておくと、プロセッサがそれを要求した場合により高速にフェッチできるので、パフォーマンスが改善します。ビクティムキャッシュが使用されない場合、追い出されたキャッシュラインは、必要になったときにもう一度メモリから読み出す必要があります。

C_ICACHE_DATA_WIDTH は、各クロックサイクルで、ビクティムキャッシュから、またはビクティムキャッシュに転送されるデータ量を指定します (32 ビットまたはキャッシュライン全体のいずれかを指定)。

注記: ビクティムキャッシュを使用できるようにするには、エリア最適化をイネーブルにしないでください。

命令キャッシュ ソフトウェア サポート

MSR ビット

MSR の ICE ビットを使用すると、ソフトウェアでキャッシュをイネーブルおよびディスエーブルにできます。

キャッシュがディスエーブルの場合、デフォルトでキャッシュの内容は保持されます。キャッシュ ラインを無効にするには、WIC 命令を使用するか、MicroBlaze のハードウェア デバッグ ロジックを使用します。

WIC 命令

オプションの WIC 命令 (C_ALLOW_ICACHE_WR=1) は、アプリケーションから命令キャッシュのキャッシュ ラインを無効するために使用します。詳細は、[第 5 章 「MicroBlaze 命令セット アーキテクチャ」](#) を参照してください。

WIC 命令をパリティ保護と一緒に使用すると、エラーが累積するのを回避するため、定期的にキャッシュのエントリを無効にできます。

データ キャッシュ

概要

MicroBlaze プロセッサでは、パフォーマンス向上のため、オプションのデータキャッシュを使用できます。キャッシュされたメモリの範囲には、LMB アドレス範囲内のアドレスを含めないでください。データ キャッシュの機能は、次のとおりです。

- 直接マップ (1 ウェイ アソシエイティブ)
- ライトスルーまたはライトバック
- キャッシュ可能メモリ アドレス範囲を選択可能
- キャッシュ サイズおよびタグ サイズを設定可能
- AXI4 インターフェイスを介してキャッシュ (M_AXI_IC)
- 4、8、または 16 ワードのキャッシュ ラインを使用可能
- MSR のビットを使用してキャッシュのオン/オフを制御
- データ キャッシュ ラインを無効化またはフラッシュするためのオプションの WDC 命令
- オプションのビクティム キャッシュ (ライトバック) に追い出されたキャッシュ ライン データを保存してパフォーマンスを改善
- ライトスルーキャッシュのオプションのパリティ保護で、ブロック RAM ビット エラーが検出された場合にキャッシュ ラインを無効化
- オプションでデータ幅を 32 ビットにするか、キャッシュ ライン全体にするか、または 512 ビットにするかを選択

一般的なデータキャッシュの機能

データキャッシュを使用すると、メモリアドレス空間はキャッシュ可能なセグメントとキャッシュ不可能なセグメントに分けられます。キャッシュ可能なセグメントは、`C_DCACHE_BASEADDR` および `C_DCACHE_HIGHADDR` という2つのパラメーターで決められます。この範囲内のアドレスはすべて、キャッシュ可能なアドレス空間に対応します。その他のアドレスはすべてキャッシュ不可能なセグメントです。

キャッシュ可能なセグメント サイズは 2^N である必要があります。N は正の整数です。`C_DCACHE_BASEADDR` および `C_DCACHE_HIGHADDR` で指定される範囲は、2 のべき乗 ($= 2^N$) で、`C_DCACHE_BASEADDR` の N 個の下位ビットはゼロである必要があります。

次の図に、データキャッシュの構成を示します。

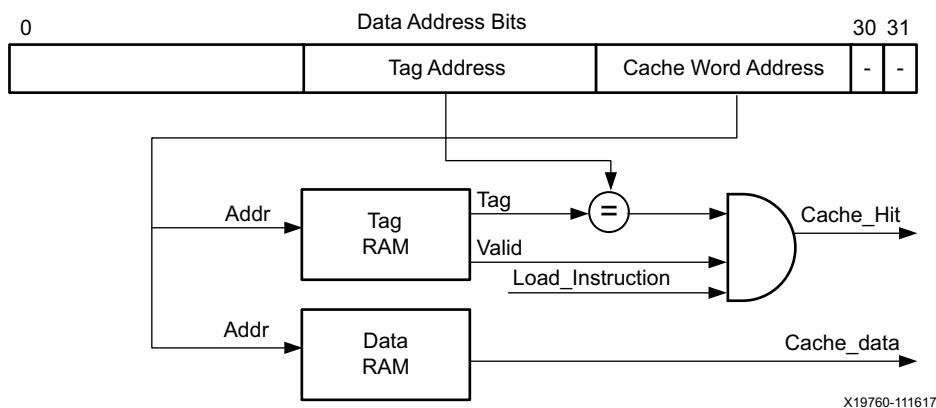


図 2-23: データキャッシュの構成

キャッシュ可能なデータアドレスは、キャッシュアドレスとタグアドレスの2つの部分で構成されています。MicroBlaze データキャッシュは、64 バイトから 64 kB までの範囲で設定できます。これは、6 から 16 ビットまでのキャッシュアドレスに対応しています。キャッシュアドレスとタグアドレスを合わせて、キャッシュ可能なメモリのフルアドレスと一致している必要があります。2 kB 未満のキャッシュサイズを選択する場合は、タグ RAM およびデータ RAM をインプリメントするのに分散 RAM が使用されますが、`C_AREA_OPTIMIZED` が 1 (エリア) に設定されていて `C_DCACHE_USE_WRITEBACK` が設定されていない場合はデータ RAM に常にブロック RAM が使用されます。パラメーター `C_DCACHE_FORCE_TAG_LUTRAM` が 1 に設定されている場合は、タグ RAM をインプリメントするのに分散 RAM が使用されます。このパラメーターは、4 ワードのキャッシュラインの場合は 8 kB 以下、8 ワードのキャッシュラインの場合は 16 kB 以下、16 ワードのキャッシュラインの場合は 32 kB 以下のキャッシュサイズでのみ使用できます。

たとえば、`C_DCACHE_BASEADDR=0x00400000`、`C_DCACHE_HIGHADDR=0x00403fff`、`C_DCACHE_BYTE_SIZE=2048`、`C_DCACHE_LINE_LEN=4`、および `C_DCACHE_FORCE_TAG_LUTRAM=0` が設定されている 32 ビット MicroBlaze では、16 kB のキャッシュ可能なメモリはバイトアドレスの 14 ビット、2 kB のキャッシュはバイトアドレスの 11 ビットを使用するので、アドレスのタグ幅は $14-11=3$ ビットになります。このコンフィギュレーションに必要なブロック RAM プリミティブの総数は、512 個のデータワードを格納するのに RAMB16 が 1 つ、128 個のキャッシュラインエントリに RAMB16 が 1 つで、それぞれ 3 ビットのタグ、4 ワードの有効ビット、1 ライン有効ビットで構成されます。合計で 2 つの RAMB16 プリミティブが使用されます。

データ キャッシュ操作

MicroBlaze のデータ キャッシュで使用されるキャッシング ポリシー(ライトバックまたはライトスルー)は、`C_DCACHE_USE_WRITEBACK` で指定されます。このパラメーターがセットされている場合はライトバック プロトコルがインプリメントされ、セットされていない場合はライトスルーがインプリメントされます。

ただし、MMU を使用する場合 (`C_USE_MMU > 1`、`C_AREA_OPTIMIZED = 0` (パフォーマンス) または 2 (周波数)、`C_DCACHE_USE_WRITEBACK = 1`) は、仮想モードのキャッシング ポリシーは TLB エントリの W ストレージ属性によって決まります。リアルモードの場合はライトバックが使用されます。

ライトバック プロトコルでは、キャッシング可能な範囲にあるアドレスへのストア命令により、キャッシングされたデータがアップデートされます。ターゲットアドレスワードがキャッシングになく(アクセスはキャッシング ミス)、キャッシングのロケーションにまだメモリに書き込まれていないデータが含まれている(キャッシング ロケーションがダーティ)場合、新しいデータでキャッシングがアップデートされる前に、古いデータがデータ AXI4 インターフェイス (`M_AXI_DC`) を介して外部メモリに書き込まれます。1ワードのみを書き込む必要がある場合は、1ワード書き込みが使用され、そ例外の場合はバースト書き込みが使用されます。バイトまたはハーフバイトのストアの場合、キャッシング ミスがあると、まずアドレスがデータ AXI4 インターフェイスを介して要求され、ワードストアでキャッシングのみがアップデートされます。

ライトスループロトコルでは、キャッシング可能な範囲にあるアドレスへのストア命令は、データ AXI4 インターフェイスを介した等価のバイト、ハーフワード、ワードの書き込みを外部メモリに対して生成します。ターゲットアドレスワードがキャッシングにある場合(書き込みがキャッシングヒット)、書き込みはキャッシングされたデータもアップデートします。書き込みキャッシング ミスの場合は、関連付けられたキャッシング ラインはキャッシングにロードされません。

キャッシングがイネーブルであれば、キャッシング可能範囲にあるアドレスからのロード命令により、要求されたデータが現在キャッシングされているかどうかを判断するためのチェックが実行されます。キャッシングされていれば(キャッシングヒット)、要求されたデータはキャッシングから読み出されます。キャッシングされていなければ(キャッシングミス)、バースト読み出しを使用してデータ AXI4 インターフェイスを介してアドレスが要求され、要求されたアドレスに関連付けられているキャッシング ラインが外部メモリ コントローラーから返されるまで、プロセッサのパイプラインはストールします。

`C_DCACHE_DATA_WIDTH` はバスデータ幅を指定し、32ビット、キャッシングライン全体(128、256、または512)、あるいは512ビットを選択します。

`C_FAULT_TOLERANT` が 1 に設定され、ライトスループロトコルが使用されている場合、タグまたはデータ ブロック RAM でパリティ エラーが検出されたときにもキャッシング ミスが発生します。

次の表に、データ キャッシュ AXI4 インターフェイスで発行されるアクセス タイプを示します。

表 2-41: データ キャッシュ インターフェイス アクセス

ポリシー	ステート	方向	アクセス タイプ
ライトスルー	キャッシュはイネーブル	読み出し	32 ビット インターフェイスの非排他的アクセスおよび ACE がイネーブルになっている排他的なアクセスの場合はバースト、それ以外の場合はシングル アクセス
		書き込み	シングル アクセス
	キャッシュはディスエーブル	読み出し	ACE がイネーブルになっている 32 ビット インターフェイスの排他的なアクセスの場合はバースト、それ以外の場合はシングル アクセス
		書き込み	シングル アクセス
ライトバック	キャッシュはイネーブル	読み出し	32 ビット インターフェイスの場合はバースト、それ以外の場合はシングル アクセス
		書き込み	複数の有効ワードがある 32 ビット インターフェイスのキャッシュ ラインの場合はバースト、それ以外はシングル アクセス
	キャッシュはディスエーブル	読み出し	32 ビット インターフェイスの非排他的アクセスの場合はバーストで目的のデータ以外はすべて破棄、それ以外の場合はシングル アクセス
		書き込み	シングル アクセス

ビクティム キャッシュ

C_DCACHE_VICTIMS を 2、4、または 8 に設定すると、ビクティム キャッシュがイネーブルになります。このパラメーターは、ビクティム キャッシュに格納できるキャッシュ ラインの数を定義します。キャッシュ ライン全体がキャッシュから追い出されると、ビクティム キャッシュに格納されます。最近のラインを格納しておくと、プロセッサがそれを要求した場合により高速にフェッチできるので、パフォーマンスが改善します。ビクティム キャッシュが使用されない場合、追い出されたキャッシュ ラインは、必要になったときにもう一度メモリから読み出す必要があります。

AXI4 インターフェイスの場合、C_DCACHE_DATA_WIDTH は、各クロック サイクルで、ビクティム キャッシュから、またはビクティム キャッシュに転送されるデータ量を指定します(32 ビットまたはキャッシュ ライン全体のいずれかを指定)。

注記: ビクティム キャッシュを使用できるようにするには、ライトバックを必ずイネーブルにし、エリア最適化をイネーブルにしないでください。

データ キャッシュ ソフトウェア サポート

MSR ビット

キャッシュをイネーブルにするかどうかは、MSR の DCE ビットで制御します。キャッシュをディスエーブルにする場合は、M_AXI_DP を介してリードバックする前に、キャッシュ可能な範囲内へのこれまでの書き込みすべてが外部メモリで完了していることをユーザーが確認する必要があります。これには、まずキャッシュをオフにする直前にセマフォに書き込みを実行し、書き込みが完了するまでループでポーリングします。キャッシュがディスエーブルの場合は、キャッシュの内容は保持されます。

WDC 命令

オプションの WDC 命令 (C_ALLOW_DCACHE_WR=1) は、アプリケーションからデータ キャッシュのキャッシュ ラインを無効化またはフラッシュするのに使用します。詳細は、[第 5 章「MicroBlaze 命令セット アーキテクチャ」](#) を参照してください。

WDC 命令をパリティ保護と一緒に使用すると、エラーが累積するのを回避するため、定期的にキャッシュのエントリを無効にできます。

ACE インターフェイスを使用して MicroBlaze に接続される System Cache などの外部 L2 キャッシュを使用する場合は、WDC を使用して外部キャッシュの無効化またはフラッシュを実行できます。System Cache の詳細は、『System Cache LogiCORE IP 製品ガイド』(PG118) [\[参照 6\]](#) を参照してください。

浮動小数点ユニット (FPU)

概要

MicroBlaze の浮動小数点ユニットは、IEEE 754-1985 規格 [参照 18] に基づいています。

- 64 ビット MicroBlaze では、無限大、非数 (NaN)、および 0 を含む IEEE 754 の单精度浮動小数点フォーマットおよび倍精度フォーマットを使用
 - 加算、減算、乗算、除算、比較、変換、平方根の命令をサポート
 - 近似値に丸めるモードをインプリメント
 - アンダーフロー、オーバーフロー、ゼロ除算、無効な操作のステイッキー ステータス ビットを生成
- パフォーマンスを向上するため、次のような規格外の簡素化が加えられています。
- 非正規化⁽¹⁾ オペランドはサポートされていません。非正規化数に対してハードウェア浮動小数点演算を実行すると、quiet NaN が返され、ステイッキー 非正規化オペランド エラー ビットが FSR にセットされます。詳細は、「浮動小数点ステータス レジスタ (FSR)」を参照してください。
 - 非正規数の結果は、符号付きの 0 として格納され、FSR のアンダーフロー ビットがセットされます。この方法は、一般的にゼロにフラッシュ (FTZ) と呼ばれます。
 - quiet NaN に対する演算は、NaN オペランドの 1 つではなく、固定の NaN (单精度では 0xFFC00000、倍精度では 0xFFFF800000000000) を返します。
 - 浮動小数点演算の結果発生するオーバーフローは、常に符号付き ∞ を返します。

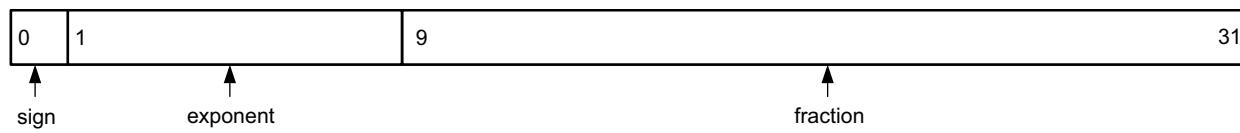
フォーマット

单精度

IEEE 754 の单精度浮動小数点数は、次の 3 つのフィールドから構成されます。

- sign* (符号): 1 ビット
- exponent* (指数): 8 ビット バイヤス
- fraction* (仮数): 23 ビット

次の図に定義されているように、これらのフィールドは 32 ビット ワードに格納されます。



X19761-111617

図 2-24: IEEE 754 单精度フォーマット

- 0 に非常に近くて全精度で表現できない $1.17549 \times 10^{-38} > n > 0$, or $(0 > n > -1.17549 \times 10^{-38}$ (单精度) および $5.562684646268 \times 10^{-309} > n > 0$ または $0 > n > -5.562684646268 \times 10^{-309}$ (倍精度) の範囲の値

MicroBlaze での浮動小数点数の値 v は次のように解釈されます。

1. $exponent = 255, fraction < 0$ の場合、 $sign$ ビットに関係なく $v = \text{NaN}$
2. $exponent = 255, fraction = 0$ の場合、 $v = (-1)^{sign} * \infty$
3. $0 < exponent < 255$ の場合、 $v = (-1)^{sign} * 2^{(exponent-127)} * (1.fraction)$
4. $exponent = 0, fraction < 0$ の場合、 $v = (-1)^{sign} * 2^{-126} * (0.fraction)$
5. $exponent = 0, fraction = 0$ の場合、 $v = (-1)^{sign} * 0$

実際には 3 と 5 のみが有用で、それ以外はエラーか、32 ビット フォーマットの全精度では表現できない数値を表します。

倍精度

IEEE 754 の倍精度浮動小数点数は、次の 3 つのフィールドから構成されます。

1. $sign$ (符号): 1 ビット
2. $exponent$ (指数): 11 ビット バイアス
3. $fraction$ (仮数): 52 ビット

次の図に定義されているように、これらのフィールドは 64 ビット long に格納されます。

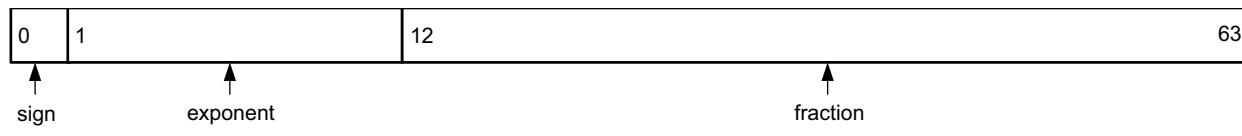


図 2-25: IEEE 754 倍精度フォーマット

MicroBlaze での浮動小数点数の値 v は次のように解釈されます。

1. $exponent = 2047, fraction < 0$ の場合、 $sign$ ビットに関係なく $v = \text{NaN}$
2. $exponent = 2047, fraction = 0$ の場合、 $v = (-1)^{sign} * \infty$
3. $0 < exponent < 2047$ の場合、 $v = (-1)^{sign} * 2^{(exponent-1023)} * (1.fraction)$
4. $exponent = 0, fraction < 0$ の場合、 $v = (-1)^{sign} * 2^{-1022} * (0.fraction)$
5. $exponent = 0, fraction = 0$ の場合、 $v = (-1)^{sign} * 0$

実際には 3 と 5 のみが有用で、それ以外はエラーか、64 ビット フォーマットの全精度では表現できない数値を表します。

丸め

MicroBlaze の FPU は、IEEE 754 で指定されているデフォルトの丸めモード「近似値へ丸め」のみをインプリメントします。定義上、任意の浮動小数点数演算の結果は、無限に正確な結果に最も近い単精度値を返すはずです。2 つの近似値が同じ近さである場合、最下位ビットが 0 の値が返されます。

演算

MicroBlaze のすべての FPU 演算では、専用の浮動小数点レジスタ ファイルではなく、プロセッサの汎用レジスタが使用されます。詳細は、「[汎用レジスタ](#)」を参照してください。

算術演算

FPU は、次の浮動小数点数演算をインプリメントします。double 演算は、64 ビット MicroBlaze で使用可能です。

- 加算、fadd および dadd
- 減算、frsub および drsub
- 乗算、fmul および dmul
- 減算、fdiv および ddiv
- 平方根、fsqrt および dsqrt ($C_USE_FPU = 2$ (拡張) の場合に使用可能)

比較

FPU は、次の浮動小数点数比較をインプリメントします。double 比較は、64 ビット MicroBlaze で使用可能です。

- 小なり比較、fcmp.lt および dcmp.lt
- 等価比較、fcmp.eq および dcmp.eq
- 以下比較、fcmp.le および dcmp.le
- 大なり比較、fcmp.gt および dcmp.gt
- 不等価比較、fcmp.ne および dcmp.ne
- 以上比較、fcmp.ge および dcmp.ge
- 順不同比較、fcmp.un および dcmp.un (NaN に対して使用)

変換

FPU は、次の変換をインプリメントします ($C_USE_FPU = 2$ (拡張) の場合に使用可能)。double 変換は、64 ビット MicroBlaze で使用可能です。

- 符号付き整数を single 浮動小数点数に変換、flt
- single 浮動小数点数を符号付き整数に変換、fint
- 符号付き long を浮動小数点数に変換、dbl
- double 浮動小数点数を符号付き long に変換、dlong

例外

浮動小数点ユニットは、MicroBlaze の一般的なハードウェア例外メカニズムを使用します。イネーブルの場合は、アンダーフロー、オーバーフロー、ゼロ除算、無効な演算などすべての IEEE 規格のコンディションと、MicroBlaze の例外である非正規化オペランド エラーに対して、例外が発生します。

浮動小数点の例外は、デスティネーション レジスタ (Rd) への書き込みを禁止します。これにより、浮動小数点例外 ハンドラーが破損していないレジスタ ファイルに対して処理を実行できます。

ソフトウェア サポート

GCC をベースにした SDK コンパイラ システムでは、MicroBlaze API に準拠した浮動小数点ユニットがサポートされています。SDK を使用する場合、システムにある FPU の種類に基づいて、GCC コマンド ラインにコンパイラ フラグが自動的に追加されます。

32 ビット MicroBlaze では、倍精度演算はすべてソフトウェアでエミュレートされます。`xil_printf()` は浮動小数点数の出力をサポートしないので注意が必要です。標準 C ライブラリの `printf()` および関連関数は、浮動小数点数の出力をサポートしていますが、プログラムのコード サイズが大きくなります。

ライブラリおよびバイナリの互換性

SDK コンパイラ システムには、ソフトウェアの浮動小数点 C ラインタイム ライブラリのみが含まれています。ハードウェアの FPU を利用するには、これらのライブラリを適切なコンパイラ オプションを使用して再コンパイルする必要があります。

異なるコンパイルが使用されている場合は、ビルド全体で FPU コンパイラ フラグが一貫していることを必ず確認してください。

演算子レイテンシ

FPU でサポートされているさまざまな演算のレイテンシは、[第 5 章の「MicroBlaze 命令セット アーキテクチャ」](#) にリストされています。FPU 命令はパイプライン化されていないため、1 回に 1 つの演算しか実行できません。

C 言語プログラミング

低水準アセンブリ言語のプログラミングを使用せずに FPU の機能を最大限に活用するには、ソース コードが C コンパイラでどのように処理されるのかを考慮することが重要です。同じアルゴリズムでも表現方法は複数あり、記述方法によって効率は異なります。

即値定数

C の浮動小数点定数はデフォルトでは倍精度です。単精度の FPU を使用する場合は、注意してコードを記述しないと、ネイティブの単精度命令ではなく、倍精度のソフトウェア エミュレーションルーチンが使用されることがあります。これを回避するには、演算式の即値定数を单精度値に明示的に指定します(キャストまたは接尾辞を使用)。

次に例を示します。

```
float x = 0.0;
...
x += (float)1.0; /* float addition */
x += 1.0F;        /* alternative to above */
x += 1.0;         /* warning - uses double addition! */
```

`-fsingle-precision-constants` というコンパイラ フラグを使用して、GNU C コンパイラですべての浮動小数点定数を单精度として処理するように指定できます(ANSI C 規格とは異なる)。

不要なキャストの回避

C_USE_FPU が 2(拡張)に設定されている場合、浮動小数点と整数の変換は FPU によりハードウェアでサポートされていますが、可能であればそのような変換は避けてください。

次の例は推奨されない例で、浮動小数点を使用して 1 ~ 10 の整数の平方根の和を計算しています。

```
float sum, t;
int i;
sum = 0.0f;
for (i = 1; i <= 10; i++) {
    t = (float)i;
    sum += t * t;
}
```

上記のコードでは、各ループ反復で、整数から浮動小数点数へのキャストが必要になります。これは次のように書き換えることができます。

```
float sum, t;
int i;
t = sum = 0.0f;
for(i = 1; i <= 10; i++) {
    t += 1.0f;
    sum += t * t;
}
```

注記: 上記 2 つのコード抜粋は、(t が非常に大きいなど) 場合によって異なる結果を生成する可能性があるので、この最適化がコンパイラで必ず実行されるとは限りません。

平方根ランタイム ライブラリ関数の使用

標準 C ランタイム数学ライブラリ関数では、倍精度演算が使用されます。単精度の FPU を使用する場合、平方根関数(sqrt())への呼び出しを使用すると、FPU 命令ではなく、非効率的なエミュレーションルーチンが使用されます。

```
#include <math.h>
...
float x=-1.0F;
...
x = sqrt(x); /* uses double precision */
```

ここでは、コンパイラから警告メッセージが表示されるのを回避するため、math.h ヘッダーが含まれています。

単精度データ型で使用すると、結果は倍精度へのキャストとなり、(FPU を使用しない) ランタイム ライブラリが呼び出され、浮動小数点への切り捨てが実行されます。

解決方法としては、単精度を使用して演算を実行し、FPU を使用して実行可能な非 ANSI 関数 sqrtf() を使用します。次に例を示します。

```
#include <math.h>
...
float x=-1.0F;
...
x = sqrtf(x); /* uses single precision */
```

注記: このコードをコンパイルする場合は、コンパイラ フラグ -fno-math-errno (-mhard-float および -mxl-float-sqrt に加えて) を必ず使用し、errno 変数をアップデートして、コンパイラでエラー コンディションを処理するために不要なコードが生成されないようにしてください。

ストリームリンクインターフェイス

MicroBlaze は、最大 16 個の AXI4-Stream インターフェイスを使用して設定でき、各インターフェイスには入力ポートと出力ポートが 1 つずつあります。チャネルは専用の単方向ポイントツー ポイントデータストリーミングインターフェイスです。

AXI4-Stream インターフェイスの詳細は、『AMBA 4 AXI4-Stream Protocol Specification, Version 1.0』(Arm IHI 0051A) [参照 14] を参照してください。

MicroBlaze のインターフェイスは 32 ビット幅です。送信/受信されたワードが制御かデータ型かは、別の 1 ビットで示されます。ポートから汎用レジスタに情報を転送するには MicroBlaze ISA の get 命令が使用され、汎用レジスタからポートにデータを転送するには put 命令が使用されます。両方の命令に、ブロッキングデータ、ノンブロッキングデータ、ブロッキング制御、ノンブロッキング制御の 4 つのタイプがあります。get および put 命令の詳細は、第 5 章「MicroBlaze 命令セットアーキテクチャ」を参照してください。

ハードウェアアクセラレーション

各リンクにより、プロセッサパイプラインへの低レイテンシ専用インターフェイスが提供されています。これらは、プロセッサ実行ユニットをカスタムハードウェアアクセラレータで拡張するのに理想的です。次の図に、単純なサンプルデザインを示します。このコードでは、RFSLx を使用して使用されるリンクを示しています。

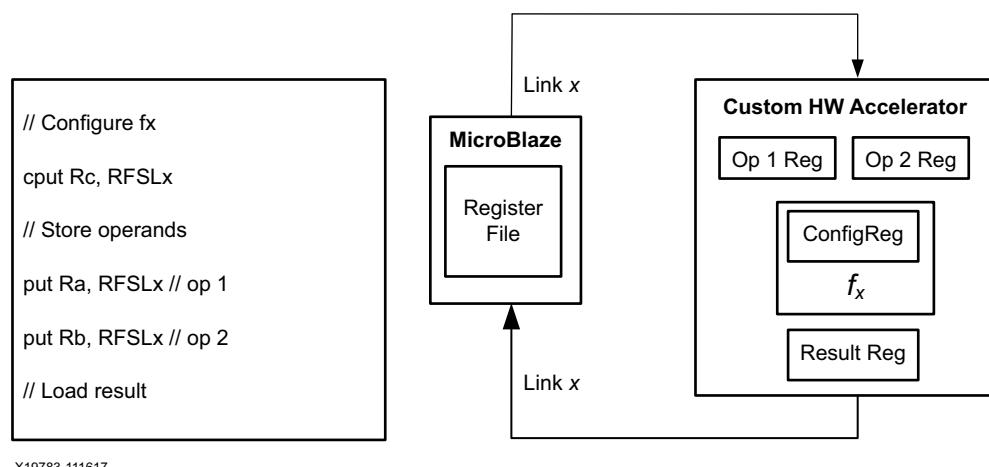


図 2-26: ストリームリンク(関数 fx をハードウェアでアクセラレーション)

この方法は、ISA をカスタム命令で拡張するのに似ていますが、プロセッサのパイプラインの全体的な速度がカスタム関数に依存しないという利点があります。また、このタイプの機能拡張には、ソフトウェアツールチェーンに対する追加要件もありません。

デバッグおよびトレース

デバッグの概要

MicroBlaze には、XSDB(ザイリンクスシステムデバッガー)などの JTAG ベースのソフトウェアデバッギングツール(一般的に BDM またはバックグラウンドデバッギングモードデバッガーと呼ばれる)をサポートするためのデバッギングインターフェイスがあります。デバッギングインターフェイスは、ザイリンクスFPGA の JTAG ポートとインターフェイスする MDM(Microprocessor Debug Module)コアに接続するように設計されています。マルチプロセッサデバッギングをイネーブルにするため、1 つの MDM に複数の MicroBlaze インスタンスを接続できます。

プログラムのダウンロード、ソフトウェアブレークポイントの設定、コードの逆アセンブルを実行可能にするには、命令およびデータのメモリ範囲がオーバーラップしていて、同じ物理メモリを使用する必要があります。

デバッギングレジスタには、デバッギングインターフェイスを使用してアクセスします。ソフトウェアでユーザーがアクセス可能なデバッギングレジスタにアクセスできるように MDM で設定されていなければ、これらのレジスタはプロセッサで実行されているソフトウェアでは認識されません。デバッギングインターフェイスには、JTAGシリアルアクセスまたは AXI4-Lite パラレルアクセスを使用でき、C_DEBUG_INTERFACE を使用して設定します。

MDM の機能については、『MicroBlaze Debug Module (MDM) 製品ガイド』(PG115) [参照 4] を参照してください。

C_DEBUG_ENABLED を 1(基本)に設定した場合にイネーブルになる基本デバッギング機能は、次のとおりです。

- ハードウェアブレークポイントおよびウォッチポイントの数、無制限のソフトウェアブレークポイントの数を設定可能
- 外部プロセッサ制御により、デバッギングツールの停止、リセット、MicroBlaze のシングルステップを設定可能
- メモリ、汎用レジスタ、特殊レジスタに対する読み出しおよび書き込み(ただし EAR、EDR、ESR、BTR、および PVR0 ~ PVR12 までは読み出し専用)
- 複数のプロセッサをサポート

C_DEBUG_ENABLED を 2(拡張)に設定した場合にイネーブルになる拡張デバッギング機能は、次のとおりです。

- パフォーマンス監視イベントおよびレイテンシカウンターの数を設定可能
- プログラムトレース
 - エンベデッドプログラムトレース(トレースバッファー サイズは設定可能)
 - MDM で接続されている複数プロセッサに対し外部プログラムトレース
- 設定可能なプロファイリングバッファー サイズで非侵入型プロファイリング サポート
- 複数のプロセッサ間のクロストリガー サポート、MDM により提供される外部クロストリガーの入力および出力

パフォーマンスの監視

MicroBlaze の拡張デバッグを使用すると、パフォーマンス監視カウンターが提供され、さまざまなイベントをカウントしてプログラム実行中のレイテンシを計測できます。イベント カウンターおよびレイテンシ カウンターの数は、それぞれ、C_DEBUG_EVENT_COUNTERS および C_DEBUG_LATENCY_COUNTERS で設定でき、カウンターフィルタ幅は C_DEBUG_COUNTER_WIDTH で 32、48、または 64 ビットに設定できます。デフォルトでは、カウンターフィルタ幅は 32 ビット、イベント カウンターは 5つ、レイテンシ カウンターは 1つに設定されます。

イベント カウンターは、単にイベントが発生した回数をカウントしますが、レイテンシ カウンターは次の情報を計測します。

- イベントが発生した回数 (N)
- イベント開始からイベント終了までのクロック サイクルをカウントして計測された各イベントのレイテンシの合計 (ΣL)。平均レイテンシを計算するのに使用されます。
- 各イベント レイテンシの平方根の合計 (ΣL^2)。レイテンシの標準偏差を計算するのに使用されます。
- すべてのイベントの最短計測レイテンシ (L_{min})
- すべてのイベントの最長計測レイテンシ (L_{max})

平均レイテンシ (μ) は次の式で計算されます。

$$\mu = \frac{\sum N}{z}$$

レイテンシの標準偏差 (σ) は、次の式で求められます。

$$\sigma = \sqrt{\frac{\sum N^2 - (\sum N)^2}{z}}$$

カウントの開始/停止は、パフォーマンス カウンター コマンド レジスタまたはクロストリガー イベントで制御します (表 2-63 を参照)。

カウンターの設定、読み出し、書き込みを実行する際、カウンターはパフォーマンス カウンターレジスタを介して順次アクセスされます。各アクセスの後、選択されたカウンター アイテムがインクリメントされます。

カウンターはパフォーマンス カウンター コマンド レジスタを使用して読み出されるので、すべて同時にサンプリングされます。これは、カウント中またはカウント停止後に実行されます。

イベント カウンターが最大値に達すると、オーバーフローステータス ビットがセットされ、外部割り込み信号 Dbg_Intr が 1 に設定されます。パフォーマンス カウンター コマンド レジスタを使用してカウンターをクリアすると、割り込み信号は 0 にリセットされます。

イベント カウンターの 1つをクロック サイクル数をカウントするために使用し、このカウンターがあらかじめ設定されているサンプリング間隔後にオーバーフローするよう初期化すると、パフォーマンス カウンターを定期的にサンプリングするために外部割り込みを使用できます。

使用可能なイベントは、表 2-42 に番号順に説明しています。

パフォーマンス監視カウンターを初期化して使用する手順は、通常次のようになります。

1. 監視するイベントを初期化します。
 - 。 パフォーマンス コマンド レジスタ ([表 2-45](#)) を使用して、リセット ビットをセットすることにより選択したカウンターを最初のカウンターにリセットします。
 - 。 パフォーマンス制御レジスタを使用して、すべてのカウンターに必要なイベント数を順に書き込みます ([表 2-44](#))。デフォルト コンフィギュレーションでは、レジスタにイベント カウンター用に5回、レイテンシ カウンター用に1回書き込みます。
2. クリア ビットおよび開始 ビットをセットし、パフォーマンス コマンド レジスタを使用してすべてのカウンターをクリアすることにより監視を開始します。
3. 監視するプログラムまたは関数を実行します。
4. カウンターをサンプリングし、パフォーマンス コマンド レジスタを使用してサンプル ビットおよび停止 ビットをセットすることにより監視を停止します。
5. すべてのカウンターから結果を読み出します。
 - 。 パフォーマンス コマンド レジスタを使用して、リセット ビットをセットし、選択したカウンターを最初のカウンターにリセットします。
 - 。 パフォーマンス カウンター ステータス レジスタを使用して、すべてのカウンターのステータスを順に読み出します ([表 2-46](#))。デフォルト コンフィギュレーションでは、レジスタをイベント カウンター用に5回、レイテンシ カウンター用に1回読み出します。オーバーフロー ビットおよびフル ビットがセットされていないことをチェックし、結果が有効であることを確認します。
 - 。 パフォーマンス コマンド レジスタを使用して、リセット ビットをセットし、選択したカウンターを最初のカウンターにリセットします。
 - 。 パフォーマンス カウンター データ読み出し レジスタを使用して、すべてのカウンターのカウンター アイテムを順に読み出します ([表 2-47](#))。表 2-48 に示すように、デフォルト コンフィギュレーションでは、レジスタをイベント カウンター用に5回、レイテンシ カウンター用に4回読み出します。
6. 計測されたイベントに応じて、最終結果を計算します。次に例を示します。
 - 。 計測されたレイテンシの平均および標準偏差を求めるには、上記の式を使用します。
 - 。 命令ごとのクロック サイクル (CPI) は、 E_{30} / E_0 で求めることができます。
 - 。 命令およびデータ キャッシュ ヒット率は、 E_{11} / E_{10} および E_{47} / E_{46} で求めることができます。
 - 。 命令キャッシュ ミス レイテンシは $(E_{60}(\Sigma L) - E_{60}(N)) / (E_{10} - E_{11})$ で求められ、データ キャッシュ読み出しおよび書き込みのミス レイテンシも同等の式を使用して求めることができます。
 - 。 プログラムの浮動小数点命令の比率は E_{29}/E_0 です。

表 2-42: MicroBlaze パフォーマンス監視イベント

イベント	説明	イベント	説明
イベント カウンターのイベント			
0	任意の有効命令実行	29	浮動小数点 (fadd、...、fsqrt)
1	ワードのロード (lw、lwi、lwx) 実行	30	クロック サイクル数
2	ハーフワードのロード (lhu、lhui) 実行	31	即値 (imm) 実行
3	バイトのロード (lbu、lbui) 実行	32	パターン比較 (pcmpbf、pcmpeq、pcmpne)
4	ワードの格納 (sw、swi、swx) 実行	33	符号拡張命令 (sext8、sext16) 実行
5	ハーフワードの格納 (sh、shi) 実行	34	命令キャッシュ無効化 (wic) 実行

表 2-42: MicroBlaze パフォーマンス監視イベント (続き)

イベント	説明	イベント	説明
6	パイプの格納 (sb、sbi) 実行	35	データキャッシュの無効化またはフラッシュ (wdc) 実行
7	無条件分岐 (br、bri、brk、brki) 実行	36	マシンステータス命令 (msrset、msrclr)
8	取られた条件付き分岐 (beq、...、bnei) 実行	37	遅延スロットのある条件なし分岐実行
9	取れていねい条件付き分岐 (beq、...、bnei) 実行	38	遅延スロットのある分岐した条件付き分岐実行
10	命令キャッシュからのデータ要求	39	遅延スロットのある分岐していない条件付き分岐実行
11	命令キャッシュでヒット	40	演算命令が実行されていない遅延スロット
12	要求されたデータをデータキャッシュから読み出し	41	ロード命令 (lbu、...、lwx) 実行
13	データキャッシュで読み出しデータヒット	42	ストア命令 (sb、...、swx) 実行
14	データキャッシュへの書き込みデータ要求	43	MMU データアクセス要求
15	データキャッシュで書き込みデータヒット	44	条件付き分岐 (beq、...、bnei) 実行
16	r1 をオペランドとして使用するロード (lbu、...、lwx) 実行	45	分岐 (br、bri、brk、brki、beq、...、bnei) 実行
17	r1 をオペランドとして使用するストア (sb、...、swx) 実行	46	データキャッシュからの読み出しありまたはデータキャッシュへの書き込みデータ要求
18	論理演算 (and、andn、or、xor) 実行	47	読み出しありまたは書き込みデータキャッシュヒット
19	算術演算 (add、idiv、mul、rsub) 実行	48	MMU 例外発生
20	乗算 (mul、mulh、mulhu、mulhsu、muli)	49	MMU 命令側例外発生
21	パレルシフター演算 (bsrl、bsra、bsll) 実行	50	MMU データ側例外発生
22	シフト演算 (sra、src、srl) 実行	51	パイプラインのストール
23	例外が取られる	52	分岐またはリターンの分岐先キャッシュヒット
24	割り込み発生	53	MMU 命令側アクセス要求
25	オペランド フェッチ段 (OF) が原因でパイプラインがストール	54	MMU 命令 TLB (ITLB) ヒット
26	実行段 (EX) が原因でパイプラインがストール	55	MMU データ TLB (DTLB) ヒット
27	メモリ段 (MEM) が原因でパイプラインがストール	56	MMU 統合 TLB (UTLB) ヒット
28	整数除算 (idiv、idivu) 実行		
レイテンシおよびイベント カウンターのイベント			
57	入力から割り込みベクターまでの割り込み レイテンシ	61	MMU アドレス ルックアップ レイテンシ
58	メモリ読み出しのデータキャッシュ レイテンシ	62	ペリフェラル AXI インターフェイス データ読み出し レイテンシ
59	メモリ書き込みのデータキャッシュ レイテンシ	63	ペリフェラル AXI インターフェイス データ書き込み レイテンシ
60	メモリ読み出しの命令キャッシュ レイテンシ		

パフォーマンス監視を設定および制御し、イベントおよびレイテンシ カウンターに読み出しおよび書き込みを実行するために使用されるデバッグ レジスタを、表 2-43 にリストします。パフォーマンス カウンター コマンド レジスタを除くこれらのレジスタはすべて、まずすべてのイベント カウンター用に、その後レイテンシ カウンター用に、情報を読み出し/書き込みするために繰り返しアクセスされます。

DBG_CTRL 値は、レジスタにアクセスするための MDM デバッグ レジスタ アクセス制御レジスタで使用される値を示し、デバッグ レジスタへの MDM ソフトウェア アクセスで使用されます。

表 2-43: MicroBlaze パフォーマンス デバッグ レジスタ

レジスタ名	サイズ(ビット)	MDM コマンド	DBG_CTRL 値	読み出し/書き込み	説明
パフォーマンス カウンター制御	8	0101 0001	4A207	書き込み	表 2-42 に示すように設定された各カウンターのイベントを選択
パフォーマンス カウンターコマンド	5	0101 0010	4A404	書き込み	カウンターのクリア、カウンターの開始または停止、カウンターのサンプリングを実行するコマンド
パフォーマンス カウンターステータス	2	0101 0011	4A601	読み出し	設定された各パフォーマンス カウンターのサンプリングされたステータスを読み出し
パフォーマンス カウンターデータ読み出し	32	0101 0110	4AC1F	読み出し	設定された各パフォーマンス カウンターのサンプリングされた値を読み出し
パフォーマンス カウンターデータ書き込み	32	0101 0111	4AE1F	書き込み	設定された各パフォーマンス カウンターの初期値を書き込み

パフォーマンス カウンター制御レジスタ

パフォーマンス カウンター制御レジスタ (PCCTRLR) は、設定されたパフォーマンス カウンターでカウントされるイベントを定義するために使用されます。設定されたカウンターすべてのイベントを定義するには、各カウンターに対してこのレジスタに繰り返し書き込む必要があります。このレジスタは書き込み専用レジスタです。読み出し要求を発行しても効果はなく、未定義データが読み出されます。

レジスタに書き込みむたびに、選択されたカウンターがインクリメントします。パフォーマンス カウンターコマンド レジスタを使用すると、選択したカウンターを最初のカウンターにリセットできます。次の図および表を参照してください。

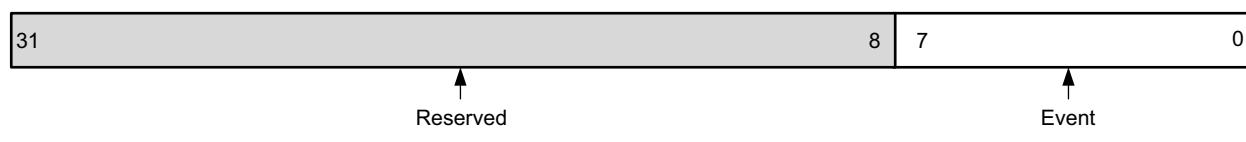


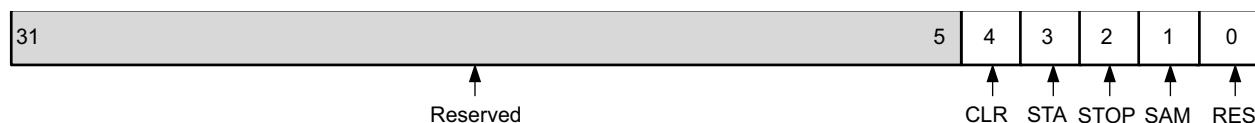
図 2-27: パフォーマンス カウンター制御レジスタ

表 2-44: パフォーマンス カウンター制御レジスタ (PCCTRLR)

ビット	名前	説明	リセット値
7:0	イベント	パフォーマンス カウンター イベント (表 2-42 を参照)。	0

パフォーマンス カウンターコマンド レジスタ

パフォーマンス カウンターコマンド レジスタ (PCCMDR) は、すべてのカウンターをクリア、開始、停止、またはサンプリングするコマンドを実行するために使用されます。このレジスタは書き込み専用レジスタです。読み出し要求を発行しても効果はなく、未定義データが読み出されます。



X19763-111617

図 2-28: パフォーマンス カウンターコマンド レジスタ

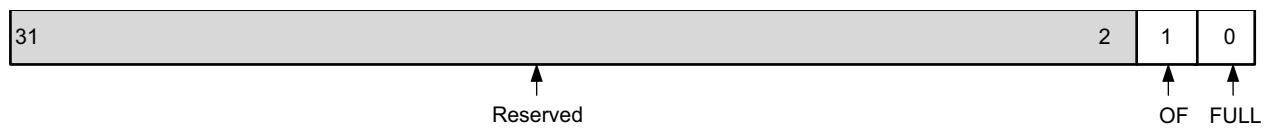
表 2-45: パフォーマンス カウンターコマンド レジスタ (PCCMDR)

ビット	名前	説明	リセット値
4	クリア	すべてのカウンターを 0 にクリア	0
3	開始	すべてのカウンターの設定されたイベントを同時にカウント開始	0
2	停止	すべてのカウンターのカウントを同時に停止	0
1	サンプリング	読み出しのためすべてのカウンターのステータスおよび値を同時にサンプリング	0
0	リセット	パフォーマンス カウンターコマンド レジスタ (PCCMDR) の値を使用して、アクセスされたカウンターを最初のイベント カウンターにリセット	0

パフォーマンス カウンターステータス レジスタ

パフォーマンス カウンターステータス レジスタ (PCSR) は、カウンターのサンプリングされたステータスを読み出します。設定されたカウンターすべてのステータスを読み出すには、各カウンターに対してこのレジスタを繰り返し読み出す必要があります。このレジスタは読み出し専用レジスタです。レジスタに書き込み要求を発行しても何も起きません。

レジスタを読み出すたびに、選択されたカウンターがインクリメントします。パフォーマンス カウンタ コマンド レジスタを使用すると、選択したカウンターを最初のカウンターにリセットできます。詳細は、[図 2-29](#) および [表 2-46](#) を参照してください。



X19764-111617

図 2-29: パフォーマンス カウンターステータス レジスタ

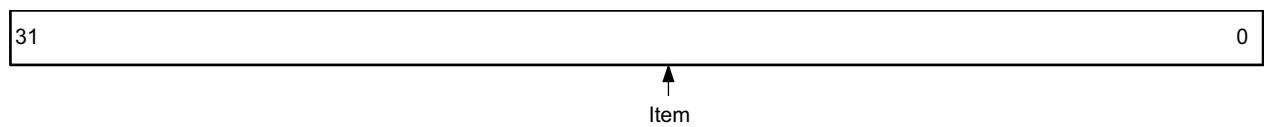
表 2-46: パフォーマンス カウンターステータス レジスタ (PCSR)

ビット	名前	説明	リセット値
1	オーバーフロー	カウンターが最大値を超えるとセットされます。	0
0	フル	前のイベントが完了する前に新しいレイテンシ カウンター イベントが開始するとセットされます。これは、計測された値の正確さが低下したことを示します。	0

パフォーマンス カウンターデータ読み出しレジスタ

パフォーマンス カウンターデータ読み出しレジスタ (PCDRR) は、カウンターのサンプリングされた値を読み出します。設定されたカウンターすべての値を読み出すには、このレジスタを繰り返し読み出す必要があります。このレジスタは読み出し専用レジスタです。レジスタに書き込み要求を発行しても何も起きません。

次の図および表を参照してください。



X19765-111617

図 2-30: パフォーマンス カウンターデータ読み出しレジスタ

表 2-47: パフォーマンス カウンターデータ読み出しレジスタ (PCDRR)

ビット	名前	説明	リセット値
31:0	項目	サンプリングされたカウンター値アイテム	0

カウンターには設定によって 33 ビット以上格納できるので、特定のカウンターの情報をすべて取得するため、このレジスタを繰り返し読み出す必要がある場合があります。詳細は、[表 2-48](#) を参照してください。

表 2-48: パフォーマンス カウンター データ アイテム

カウンター タイプ	項目	説明	
C_DEBUG_COUNTER_WIDTH = 32			
イベント カウンター	1	イベントが発生した回数	
レイテンシ カウンター	1	イベントが発生した回数	
	2	各イベント レイテンシの合計	
	3	各イベント レイテンシの平方の和	
	4	31:16 最小計測レイテンシ、16 ビット 15:0 最大計測レイテンシ、16 ビット	
C_DEBUG_COUNTER_WIDTH = 48			
イベント カウンター	1	31:16 0x0000 15:0	イベントが発生した回数、上位 16 ビット
	2	イベントが発生した回数、下位 32 ビット	
レイテンシ カウンター	1	イベントが発生した回数	
	2	31:16 0x0000 15:0	各イベント レイテンシの合計、上位 16 ビット
	3	各イベント レイテンシの合計、下位 32 ビット	
	4	31:16 0x0000 15:0	各イベント レイテンシの平方根の和、上位 16 ビット
	5	各イベント レイテンシの平方根の和、下位 32 ビット	
	6	最小計測レイテンシ、32 ビット	
	7	最大計測レイテンシ、32 ビット	
C_DEBUG_COUNTER_WIDTH = 64			
イベント カウンター	1	イベントが発生した回数、上位 32 ビット	
	2	イベントが発生した回数、下位 32 ビット	
レイテンシ カウンター	1	イベントが発生した回数、32 ビット	
	2	各イベント レイテンシの合計、上位 32 ビット	
	3	各イベント レイテンシの合計、下位 32 ビット	
	4	各イベント レイテンシの平方根の和、上位 32 ビット	
	5	各イベント レイテンシの平方根の和、下位 32 ビット	
	6	最小計測レイテンシ、32 ビット	
	7	最大計測レイテンシ、32 ビット	

パフォーマンス カウンター データ書き込みレジスタ

パフォーマンス カウンター データ書き込みレジスタ (PCDWR) は、初期値をカウンターに書き込みます。設定されたカウンターすべてに書き込むには、このレジスタに繰り返し書き込む必要があります。このレジスタは書き込み専用レジスタです。読み出し要求を発行しても効果はなく、未定義データが読み出されます。

表 2-48 に示すように、カウンターには設定によって 33 ビット以上格納できるので、特定のカウンターの情報をすべてアップデートするため、このレジスタに繰り返し書き込む必要がある場合があります。

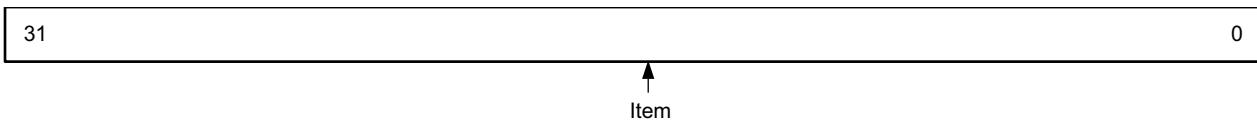


図 2-31: パフォーマンス カウンター データ書き込みレジスタ

表 2-49: パフォーマンス カウンター データ書き込みレジスタ (PCDWR)

ビット	名前	説明	リセット値
31:0	項目	カウンターに書き込むたカウンター値アイテム	0

プログラムおよびイベントトレース

MicroBlaze の拡張デバッグにはプログラムおよびイベントトレース機能が含まれ、エンベデッドトレースバッファーに情報を格納するか、MDM に情報を転送して、プログラム実行のトレースを可能にします。MDM は、C_DEBUG_EXTERNAL_TRACE が設定されているときに使用され、外部インターフェイスを介して、複数のプロセッサからのプログラムトレースの出力を可能にします。

エンベデッドトレースバッファーのサイズは、C_DEBUG_TRACE_SIZE を使用して、8 KB から 128 KB の範囲で設定できます。外部トレースを含むデフォルトのバッファー サイズは 8 KB ですが、分散 RAM を使用するために 32 KB ~ 256 KB に設定することもできます。ブロック RAM リソースが非常に少ない場合を除き、常にデフォルトの 8 KB を使用することをお勧めします。C_DEBUG_TRACE_SIZE を 0(なし) に設定すると、プログラムトレースはディスクエーブルになります。

プログラムトレースでは、トレースデータ量を削減するため圧縮が使用されますが、プログラム実行フローまたはプロセッサソフトウェアステート全体は再構築できます。圧縮には主に 3 つのレベルがあります。

- 完全トレース:** 144 ビット (エンベデッドトレースバッファー サイズの設定によって 512 ~ 8192 個のアイテム) を使用して実行された各命令のサイクルカウントなど、完全なトレース情報を格納します。完全トレースは、C_DEBUG_EXTERNAL_TRACE がセットされている場合または 64 ビット MicroBlaze (C_DATA_SIZE = 64) の場合は使用できません。
- プログラムフロー:** 取られた分岐/取られなかった分岐のシーケンス、間接的な分岐、割り込み、例外、ハードウェアブレークの新しいプログラムカウンターなど、プログラムフローの変化を格納します。

また、リターン命令がプログラムフローの再構築を単純化できるように、または取られたすべての分岐で自己変更コードを処理するようにするために、オプションでプログラムカウンターを格納することも可能です。

メモリから読み出されたデータまたは AXI4-Stream インターフェイスからフェッチされたデータは、オプションでプロセッサソフトウェアステート全体を再構築できるように格納でき、逆シングルステップ機能を可能にします。データアクセス命令がダイナミック分岐またはリターンの遅延スロットにある場合は、まずデータが格納されてから、分岐ターゲットプログラムカウンターが格納されます。スタティック分岐の遅延スロットのデータアクセス命令の場合は、まずプログラムフローの変更が保存されてから、データが保存されます。

プログラム フロー変更を明確にデコードできるように、表 2-63 で定義されているイベント (すべてのプログラム実行、割り込み、ブレーク、およびクロストリガー イベント) も格納されます。各イベントは、格納されたプログラム カウンターから始まります。

ソフトウェアは、`xori r0, rN, IMM` 命令を使用してイベントを挿入できます。これは、通常コンテキスト スイッチやシステム コールなどのオペレーティング システム イベントをトレースするために使用されますが、プログラムで重要なイベントをトレースするためにも使用できます。

- **プログラム フローおよびサイクル カウント:** 命令間のサイクル カウントと、プログラム フローのみの場合と同じ情報を格納します。プログラム実行時間の再構築も可能にします。
- **イベント トレース:** サイクル カウント イベントなどのイベント トレース情報を格納します。イベントにはすべてのプログラム実行、割り込み、ブレーク、およびクロストリガー イベントが含まれ、表 2-63 に定義されています。各イベントは、格納されたプログラム カウンターから始まります (オプション)。

プログラム カウンターは、呼び出し命令でプログラムで関数呼び出しをトレースし、リターン命令で関数呼び出しのリターンをトレースするために、オプションで格納することもできます。

ソフトウェアは、`xori r0, rA, IMM` 命令を使用してイベントを挿入できます。これは、通常コンテキスト スイッチやシステム コールなどのオペレーティング システム イベントをトレースするために使用されますが、プログラムで重要なイベントをトレースするためにも使用できます。

トレースは、トレース コマンド レジスタを使用して、プログラム トレース制御 レジスタでトレース ポイントとして設定されたブレーク ポイントまたはウォッチ ポイントに到達したときか、クロストリガー イベントによって開始できます (表 2-63 参照)。

トレースは、トレース バッファーがフルになると自動的に停止しますが、トレース コマンド レジスタまたはクロストリガー イベントにより停止させることも可能です (表 2-63 参照)。

サイクル カウントは、完全トレースを使用している場合は最大 32768 クロック サイクルまでカウントでき、プログラム フローおよびサイクル カウントを使用している場合は命令間で 8182 サイクルまでカウントできます。サイクル カウントがこの値を超えると、トレース ステータス レジスタ オーバーフロー ビットが 1 にセットされます。

トレース バッファーがフルになったとき、またはサイクル カウントがオーバーフローになったときに、プロセッサを停止するようトレースを設定することができます。そのように設定すると、トレース バッファーを読み出すのにかかる時間のためリアルタイムではありませんが、プログラム フロー全体を継続的にトレースできます。

次の表に、トレースを設定および制御し、エンベデッド トレース バッファーを読み出すのに使用されるデバッグ レジスタを示します。

`DBG_CTRL` 値は、レジスタにアクセスするための MDM デバッグ レジスタ アクセス 制御 レジスタで使用される値を示し、デバッグ レジスタへの MDM ソフトウェア アクセスで使用されます。

表 2-50: MicroBlaze プログラムトレースデバッグレジスタ

レジスタ名	サイズ(ビット)	MDMコマンド	DBG_CTRL値	読み出し/書き込み	説明
トレース制御	22	0110 0001	4C215	書き込み	トレースポイント、トレース圧縮レベルを設定、およびオプションでトレース情報を格納
トレースコマンド	4	0110 0010	4C403	書き込み	トレースバッファーのクリア、トレースの開始/停止、および現在のバッファーアイテム数をサンプリングするコマンド
トレースステータス	18	0110 0011	4C611	読み出し	サンプリングされたトレースバッファーステータスの読み出し
トレースデータ読み出し ¹	18	0110 0110	4CC11	読み出し	エンベデッドトレースバッファーから一番古いアイテムを読み出し

1. C_DEBUG_EXTERNAL_TRACE が設定されている場合、このレジスタは使用できません。

トレース制御レジスタ

トレース制御レジスタ(TCTRLR)は、トレース動作を定義するために使用されます。このレジスタは書き込み専用レジスタです。読み出し要求を発行しても効果はなく、未定義データが読み出されます。次の図および表を参照してください。

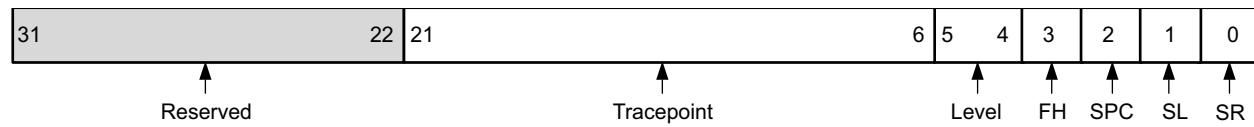


図 2-32: トレース制御レジスタ

表 2-51: トレース制御レジスタ(TCTRLR)

ビット	名前	説明	リセット値
21:6	トレースポイント	対応するブレークポイントまたはウォッチポイントをトレースポイントに変更	0
5:4	レベル	トレース圧縮レベル 00: 完全トレース(C_DEBUG_EXTERNAL_TRACEと一緒に使用することは不可) 01: プログラムフロー 10: イベント 11: プログラムフローおよびサイクルカウント	00
3	フル停止	フルトレースバッファーまたはサイクルカウントオーバーフローでデバッグ停止	0
2	プログラムカウンター(PC)保存	レベル 01 および 11: 分岐しているすべての分岐に対し新しいプログラムカウンターを保存 レベル 10: すべての関数呼び出しに対し新しいプログラムカウンターを保存	0
1	ロード保存	ロードを保存し、命令新規データ値を取得	0
0	リターン保存	戻り命令用に対し新しいプログラムカウンターを保存	0

トレース コマンド レジスタ

トレース コマンド レジスタ (TCMDR) は、トレースをクリア、開始、停止し、トレース アイテム数をサンプリングするためのコマンドを発行するのに使用されます。このレジスタは書き込み専用レジスタです。読み出し要求を発行しても効果はなく、未定義データが読み出されます。次の図および表を参照してください。

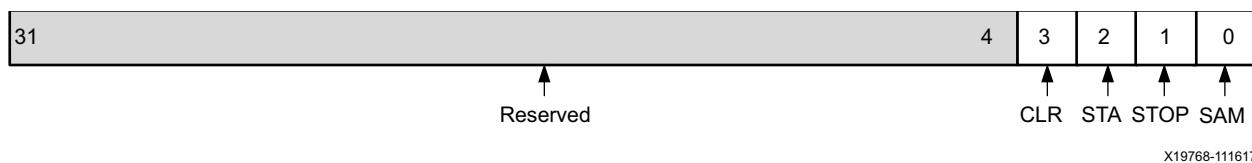


図 2-33: トレース コマンド レジスタ

表 2-52: トレース コマンド レジスタ (TCMDR)

ビット	名前	説明	リセット値
3	クリア	トレース ステータスをクリアし、トレース バッファーを空にする	0
2	開始	トレースをすぐに開始	0
1	停止	トレースをすぐに停止	0
0	サンプリング	トレース バッファーの現在のアイテム数をサンプリング	0

トレース ステータス レジスタ

トレース ステータス レジスタ (TSR) は、トレースが開始したかどうかの判断、サイクル カウント オーバーフローのチェック、エンベデッド トレース バッファーでサンプリングされたアイテム数の読み出しに使用されます。このレジスタは読み出し専用レジスタです。レジスタに書き込み要求を発行しても何も起きません。次の図および表を参照してください。

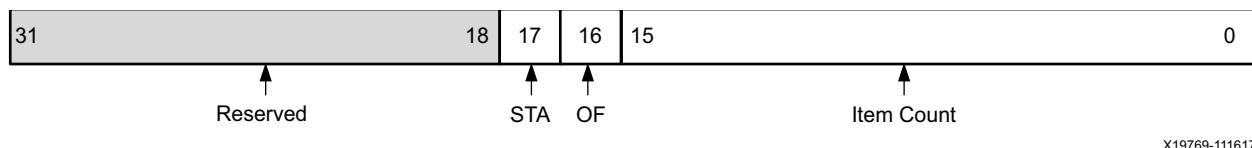


図 2-34: トレース ステータス レジスタ

表 2-53: トレース ステータス レジスタ (TSR)

ビット	名前	説明	リセット値
17	開始	トレース開始、トレースが開始したときは 1 にセット、停止したときは 0 にクリア	0
16	オーバーフロー	サイクル カウント オーバーフロー、サイクル カウント オーバーフローのときは 1 にセット、クリア コマンドにより 0 にクリア	0
15:0	アイテム カウント	トレース バッファー アイテム数をサンプリング	0x0000

トレースデータ読み出しレジスタ

トレースデータ読み出しレジスタ (TDRR) には、エンベデッド トレースバッファーから読み出された一番古いアイテムが含まれます。このレジスタが読み出されると、次のアイテムがトレースバッファーから読み出されます。トレースバッファーで使用可能なアイテムを超えて読み出すとエラーとなり、トレースステータスレジスタのアイテム数で示されます。このレジスタは読み出し専用レジスタです。レジスタに書き込み要求を発行しても何も起きません。次の図および表を参照してください。

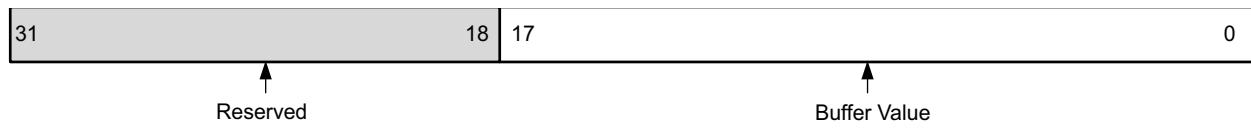


図 2-35: トレースデータ読み出しレジスタ

圧縮レベルおよび格納されているデータによってトレースデータエンティティは19ビット以上で構成されている可能性があるので、特定のデータエンティティの情報をすべて取得するため、このレジスタを繰り返し読み出す必要がある場合があります。詳細は、[表 2-55](#) を参照してください。

表 2-54: トレースデータ読み出しレジスタ (TDRR)

ビット	名前	説明	リセット値
17:0	バッファー値	エンベデッド トレースバッファー アイテム	0x00000

表 2-55: トレースカウンターデータエンティティ

エンティティ	項目	ビット	説明
完全トレース	1	17:3 2:0	実行された命令のサイクルカウント マシンステータスレジスタ [17:19]
	2	17:6 5:1 0	マシンステータスレジスタ [20:31] デスティネーションレジスタアドレス (r0 ~ r31)、書き込まれている場合は有効 1にセットされている場合はデスティネーションレジスタに書き込み
	3	17:13 12 11 10 9:6 5:0	例外の種類 、例外がある場合は有効 1にセットされている場合は例外あり 1にセットされている場合はロード命令でデータ読み出し 1にセットされている場合はストア命令でデータ書き込み バイトインペブル、ストア命令の場合有効 ストア命令の場合は書き込みデータ [0:5]、または他の命令の場合はデスティネーションレジスタデータ [0:5]
	4	17:0	書き込みデータ [6:23]、またはデスティネーションレジスタデータ [6:23]
	5	17:10 9:0	書き込みデータ [24:31]、またはデスティネーションレジスタデータ [24:31] ロードおよびストア命令の場合はデータアドレス [0:9]、または その他の命令の場合は実行された命令 [0:9]

表 2-55: トレース カウンターデータ エンティティ (続き)

エンティティ	項目	ビット	説明
	6	17:0	データ アドレス [10:27] または実行された命令 [10:27]
	7	17:14 13:0	データ アドレス [28:31] または実行された命令 [28:31] プログラム カウンター [0:13]
	8	17:0	プログラム カウンター [14:31]
プログラム フロー: 分岐	1	17:16 15:12 11:0	00 - アイテムにプログラム フロー分岐が含まれる アイテム (0 ~ 12) でカウントされる分岐の数 (N) 左の N ビットはプログラム フローの分岐を表す。この ビットが 1 の場合、分岐が取られており、それ以外の場合 は分岐は取られていない。 0 分岐のアイテムは無視可能。トレース パケットを完了さ せるために、外部トレースをフラッシュした場合に分岐 が発生する可能性はあり。
プログラム フロー: プログラム カウンター	1	17:16 15:0	01 - アイテムにプログラム カウンター値が含まれる プログラム カウンター [0:15]
	2	17:16 15:0	01 - アイテムにプログラム カウンター値が含まれる プログラム カウンター [16:31]
プログラム フロー: プログラム カウンター C_ADDR_SIZE = 32 ~ 48	1	17:16 15:0	01 - アイテムにプログラム カウンター値が含まれる プログラム カウンター [0:C_ADDR_SIZE-33] ゼロ拡張
	2	17:16 15:0	01 - アイテムにプログラム カウンター値が含まれる プログラム カウンター [C_ADDR_SIZE-32:C_ADDR_SIZE-17]
	3	17:16 15:0	01 - アイテムにプログラム カウンター値が含まれる プログラム カウンター [C_ADDR_SIZE-16:C_ADDR_SIZE-1]
プログラム フロー: プログラム カウンター C_ADDR_SIZE = 49 ~ 64	1	17:16 15:0	01 - アイテムにプログラム カウンター値が含まれる プログラム カウンター [0:C_ADDR_SIZE-49] ゼロ拡張
	2	17:16 15:0	01 - アイテムにプログラム カウンター値が含まれる プログラム カウンター [C_ADDR_SIZE-48:C_ADDR_SIZE-33]
	3	17:16 15:0	01 - アイテムにプログラム カウンター値が含まれる プログラム カウンター [C_ADDR_SIZE-32:C_ADDR_SIZE-17]
	4	17:16 15:0	01 - アイテムにプログラム カウンター値が含まれる プログラム カウンター [C_ADDR_SIZE-16:C_ADDR_SIZE-1]
プログラム フロー: 読み出しデータ C_DATA_SIZE = 32 または 64	1	17:16 15:0	10 - アイテムに読み出しデータが含まれる ロードおよび get 命令で読み出されたデータ [0:15]
	2	17:16 15:0	10 - アイテムに読み出しデータが含まれる ロードおよび get 命令で読み出されたデータ [15:31]

表 2-55: トレース カウンターデータ エンティティ (続き)

エンティティ	項目	ビット	説明
プログラム フロー: 読み出しデータ C_DATA_SIZE = 64	1	17:16	10 - アイテムに読み出しデータが含まれる
		15:0	long ロード命令で読み出されたデータ [0:15]
	2	17:16	10 - アイテムに読み出しデータが含まれる
		15:0	long ロード命令で読み出されたデータ [15:31]
プログラム フロー、イベント: イベント 命令イベント	3	17:16	10 - アイテムに読み出しデータが含まれる
		15:0	long ロード命令で読み出されたデータ [32:47]
	4	17:16	10 - アイテムに読み出しデータが含まれる
		15:0	long ロード命令で読み出されたデータ [48:63]
プログラム フロー、イベント: イベント 命令イベント	1	17:16	11 - アイテムにイベントが含まれる
		15:14	00 - 命令イベント
		13:0	ソフトウェアの生成するトレース イベント: xor r0, rA, IMM 命令の結果。
プログラム フロー、イベント: イベント クロストリガー イベント	1	17:16	11 - アイテムにイベントが含まれる
		15:1	10 - クロストリガー イベント
		13:8	予約
		7:0	表 2-64 で定義される「MicroBlaze クロストリガー イベント」によるイベント。各イベントはそれぞれビット フィールドの該当ビットを設定することで表されます。
プログラム フロー、イベント: イベント 例外イベント	1	17:16	11 - アイテムにイベントが含まれる
		15:14	11 - 例外イベント:
		13:5	予約
		4:0	表 2-12 で定義される「ESR 例外の原因」による例外原因: 01001 - デバッグ例外: ブレークポイント、停止 01010 - 割り込み 01011 - マスク不可能なブレーク 01100 - ブレーク
イベント: イベント タイムスタンプ	1	17:16	11 - アイテムにイベントが含まれる
		15:14	01 - タイムスタンプ
		13:0	最後のタイムスタンプからのサイクル カウント

表 2-55: トレース カウンターデータ エンティティ (続き)

エンティティ	項目	ビット	説明
プログラム フローとサイクル カウント: 分岐および短いサイクル カウント	1	17:16 15:14 13:8 7 6:1 0	00 - アイテムにプログラム フロー分岐が含まれる 01、10 - カウントされた分岐の数 (N) (1 ~ 2) 前に実行された命令のサイクル カウント 1 の場合は分岐は取られており、それ以外の場合は取られていない 前に実行された命令のサイクル カウント 1 の場合は分岐は取られており、それ以外の場合は取られていない
プログラム フローとサイクル カウント: 分岐および長いサイクル カウント	1	17:16 15:14 13:1 0	00 - アイテムにプログラム フロー分岐が含まれる 11 - アイテムに分岐および長いサイクル カウントが含まれる 前に実行された命令のサイクル カウント 1 の場合は分岐は取られており、それ以外の場合は取られていない

非侵入型のプロファイリング

拡張デバッグでは、プログラム実行の統計を格納するのにプロファイリングバッファーを使用する、非侵入型プロファイリングが提供されています。プロファイリングバッファーのサイズは、`C_DEBUG_PROFILE_SIZE`を使用して、4 KB から 128 KB の範囲で設定できます。`C_DEBUG_PROFILE_SIZE`を0(なし)に設定すると、非侵入型プロファイリングはディスエーブルになります。

プロファイリングバッファーはビンに分けられており、各ビンは特定のアドレス範囲内で実行された命令数またはクロックサイクル数をカウントします。各ビンは、命令またはサイクルを $2^{36} - 1 = 68719476735$ までカウントします。

各ビンのアドレス範囲は、バッファー サイズと、プロファイリングロードレスレジスタおよびプロファイリングハイアドレスレジスタで定義されたプロファイルされたアドレス範囲によって決まります。

プロファイリングの開始/停止は、プロファイリング制御レジスタまたはクロストリガーイベントで制御します(表 2-63 を参照)。

プロファイリングを設定および制御し、プロファイリングバッファーの読み出し/書き込みに使用されるデバッグレジスタを、表 2-56 にリストします。

`DBG_CTRL` 値は、レジスタにアクセスするための MDM デバッグレジスタアクセス制御レジスタで使用される値を示し、デバッグレジスタへの MDM ソフトウェアアクセスで使用されます。

表 2-56: MicroBlaze プロファイリング デバッグレジスタ

レジスタ名	サイズ(ビット)	MDM コマンド	DBG_CTRL 値	読み出し/書き込み	説明
プロファイリング制御	8	0111 0001	4E207	書き込み	プロファイリングをイネーブル/ディスエーブル、カウント方法およびビンの使用方法を設定
プロファイリングロードレス	<code>C_ADDR_SIZE</code> - 2	0111 0010	4E41D	書き込み	プロファイルされたアドレス範囲の下位アドレスを定義
プロファイリングハイアドレス	<code>C_ADDR_SIZE</code> - 2	0111 0011	4E61D	書き込み	プロファイルされたアドレス範囲の上位アドレスを定義
プロファイリングバッファー アドレス	9 ~ 14	0111 0100	9: 4E808 10: 4E809 ... 14: 4E80D	書き込み	読み出しほりは書き込みするプロファイリングバッファーのアドレス(ビン)を設定
プロファイリングデータ読み出し	36	0111 0110	4EC23	読み出し	プロファイリングバッファーからの読み出しデータ
プロファイリングデータ書き込み	32	0111 0111	4EE1F	書き込み	プロファイリングバッファーへの書き込みデータ

プロファイリング制御レジスタ

プロファイリング制御レジスタ (PCTRLR) は、プロファイリングのイネーブル (開始)/ディスエーブル (停止) のために使用されます。また、実行命令数または実行クロック サイクル数のどちらをカウントするかの設定、プロファイリング バッファービンの使用方法の定義にも使用されます。

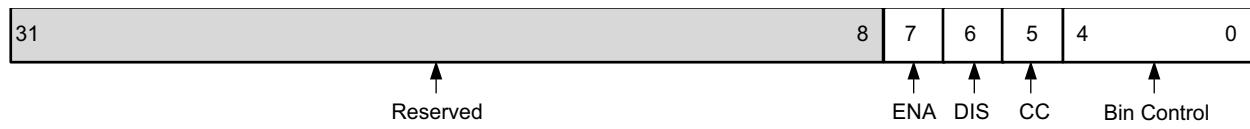
このレジスタは書き込み専用レジスタです。読み出し要求を発行しても効果はなく、未定義データが読み出されます。次の図および表を参照してください。

ビン制御値 (B) は、次の式で求めることができます。

$$B = \left\lceil \log_2 \frac{H-L+S \cdot 4}{S \cdot 4} \right\rceil$$

説明:

- L : プロファイリング ロー レジスタ
- H : プロファイリング ハイ レジスタ
- S : C_DEBUG_PROFILE_SIZE パラメーター



X19771-111617

図 2-36: プロファイリング制御レジスタ

表 2-57: プロファイリング制御レジスタ (PCTRLR)

ビット	名前	説明	リセット値
7	イネーブル	プロファイリングのイネーブル (開始)	0
6	ディスエーブル	プロファイリングのディスエーブル (停止)	0
5	サイクルカウント のイネーブル	実行された命令のクロック サイクルをカウント: 0: ディスエーブル、実行された命令のカウント数 1: イネーブル、実行された命令のクロック サイクルの数	0
4:0	ビン制御	プロファイリング バッファービンの各ビンでカウントされるアドレス数	00000

プロファイリング ロー アドレス レジスタ

プロファイリング ロー アドレス レジスタ (PLAR) は、プロファイルされたエリアの下位ワード アドレスを定義するのに使用されます。このレジスタは書き込み専用レジスタです。読み出し要求を発行しても効果はなく、未定義データが読み出されます。次の図および表 2-58 を参照してください。

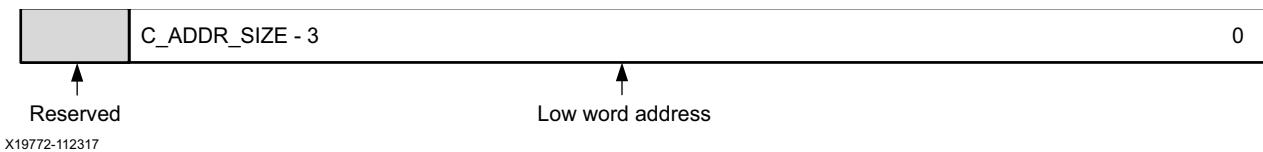


図 2-37: プロファイリング ロー アドレス レジスタ

表 2-58: プロファイリング ロー アドレス レジスタ (PLAR)

ビット	名前	説明	リセット値
C_ADDR_SIZE-3:0	下位ワード	プロファイルされたエリアの下位ワード アドレス	0

プロファイリング ハイ アドレス レジスタ

プロファイリング ハイ アドレス レジスタ (PHAR) は、プロファイルされたエリアの上位ワード アドレスを定義するのに使用されます。このレジスタは書き込み専用レジスタです。読み出し要求を発行しても効果はなく、未定義データが読み出されます。次の図および表を参照してください。

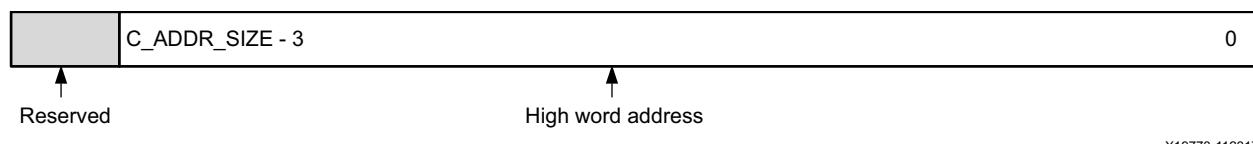


図 2-38: プロファイリング ハイ アドレス レジスタ

表 2-59: プロファイリング ハイ アドレス レジスタ (PHAR)

ビット	名前	説明	リセット値
C_ADDR_SIZE-3:0	上位ワード	プロファイルされたエリアの上位ワード アドレス	0

プロファイリング バッファー アドレス レジスタ

プロファイリング バッファー アドレス レジスタ (PBAR) は、読み出す/書き込むプロファイリング バッファーのビンを定義するのに使用されます。このレジスタのビット数は、C_DEBUG_PROFILE_SIZE で設定します。

このレジスタは書き込み専用レジスタです。読み出し要求を発行しても効果はなく、未定義データが読み出されます。次の図および表を参照してください。

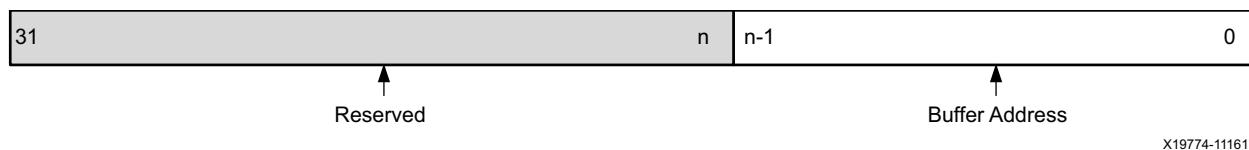


図 2-39: プロファイリング バッファー アドレス レジスタ

表 2-60: プロファイリング バッファー アドレス レジスタ (PBAR)

ビット	名前	説明	リセット値
n-1:0	バッファー アドレス	読み出しまたは書き込みをするプロファイリング バッファーのビン。ビット数 (n) は、4 KB バッファーの場合 10、8 KB バッファーの場合 11、...、128 KB バッファーの場合 15。	0

プロファイリング データ読み出しレジスタ

プロファイリング データ読み出しレジスタ (PDRR) は、プロファイリング バッファー アドレス レジスタで示されるビンの値を読み出し、プロファイリング バッファー アドレス レジスタをインクリメントします。このレジスタは読み出し専用レジスタです。レジスタに書き込み要求を発行しても何も起きません。次の図および表を参照してください。

レジスタをデバッグするため MDM ソフトウェア アクセスでこのレジスタを読み出す場合、2 回連続アクセスするとデータが読み出されます。

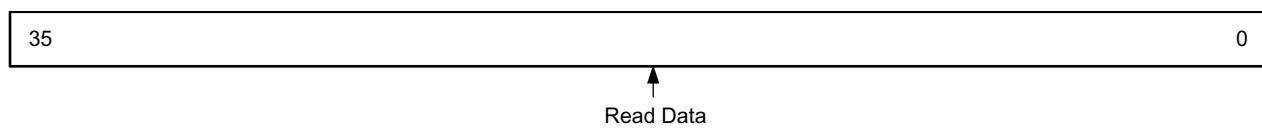


図 2-40: プロファイリング データ読み出しレジスタ

表 2-61: プロファイリング データ読み出しレジスタ (PDRR)

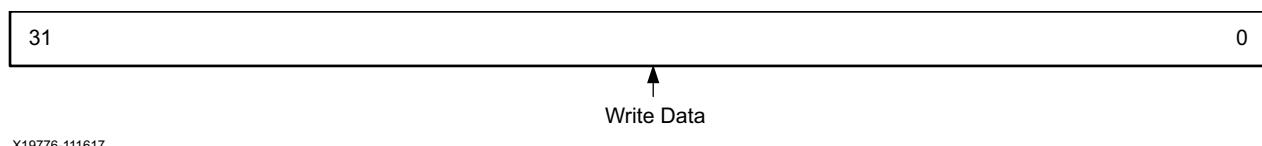
ビット	名前	説明	リセット値
35:0	読み出し データ	ビンで実行された命令数または実行されたクロック サイクル数	0

プロファイリング データ書き込みレジスタ

プロファイリング データ書き込みレジスタ (PDWR) は、プロファイリング バッファー アドレス レジスタで示されるビンに新しい値を書き込み、プロファイリング バッファー アドレス レジスタをインクリメントします。このレジスタは書き込み専用レジスタです。読み出し要求を発行しても効果はなく、未定義データが読み出されます。

このレジスタは、プロファイリングをイネーブルする前にプロファイリング バッファーをクリアするために使用できます。

プロファイリング バッファービンの最上位ビット 4 つが新しい値が書き込まれると 0 になります。次の図および表を参照してください。



X19776-111617

図 2-41: プロファイリング データ書き込みレジスタ

表 2-62: プロファイリング データ書き込みレジスタ (PDWR)

ビット	名前	説明	リセット値
31:0	書き込みデータ	ビンに書き込むデータ	0

クロストリガー サポート

クロストリガーは、DBG_STOP と MB_Halted の 2 つの信号で指定できます。

- DBG_STOP 入力を 1 になると、MicroBlaze は数命令後に停止します。XSDB は MicroBlaze の停止を検出し、停止が発生した箇所を示します。この信号は、Vivado® Integrated Logic Analyzer (ILA) がトリガーされたときなど、外部イベントで MicroBlaze プロセッサを停止するのに使用できます。
- ブレークポイントまたはウォッチポイントに到達した後、停止 XSDB コマンドの後、または DBG_STOP 入力が設定されたときなど、MicroBlaze が停止すると MB_Halted 出力信号が 1 に設定されます。XSDB コマンドで MicroBlaze の実行が再開されるとクリアされます。

MB_Halted 信号は、Vivado ILA をトリガーしたり、DBG_STOP 入力に接続してマルチプロセッサシステムでの MicroBlaze コアを停止したりするのに使用できます。

拡張デバッグでは、クロストリガーは MDM と共に使用するとサポートされます。MDM には、接続されたすべてのプロセッサ間のプログラマブルクロストリガーと、外部トリガー入力および出力が含まれます。詳細は、『MicroBlaze Debug Module (MDM) 製品ガイド』(PG115) [参照 4] を参照してください。

MicroBlaze では、8 個までのクロストリガー アクションを処理できます。クロストリガー アクションは、デバッグバスを使用して接続されている、対応する MDM のクロストリガー出力によって生成されます。表 2-63 に、各クロストリガー アクションの影響を示します。

MicroBlaze では、最大 8 個のクロストリガー イベントを生成できます。クロストリガー イベントは、デバッグバスを使用して接続されている、そのイベントに対応する MDM のクロストリガー入力に影響します。表 2-64 に、クロストリガー イベントの説明を示します。

表 2-63: MicroBlaze クロストリガー アクション

番号	操作	説明
0	デバッグ停止	プロセッサが実行中の場合は MicroBlaze を停止し、MB_Halted 出力をセット。Dbg_Stop 入力をセットした場合と同じ。
1	実行継続	プロセッサが実行中の場合は実行を継続し、MB_Halted 出力をクリア。
2	プログラムトレース停止	トレース中の場合はプログラムトレースを停止。
3	プログラムトレース開始	トレースが停止している場合はプログラムトレースを開始。
4	パフォーマンス監視停止	パフォーマンス監視中の場合は監視を停止。
5	パフォーマンス監視開始	パフォーマンス監視停止の場合は監視を開始。
6	プロファイリングのディスエーブル	プロファイリング中の場合はプロファイリングをディスエーブル。
7	プロファイリングのイネーブル	プロファイリングがディスエーブルの場合はプロファイリングをイネーブル。

表 2-64: MicroBlaze クロストリガー イベント

番号	イベント	説明
0	MicroBlaze 停止	MicroBlaze が停止したときにイベントを生成。MB_Halted 出力が設定された場合と同じイベントを送信。
1	実行再開	デバッグ停止からプロセッサの実行を再開するときにイベントを生成。MB_Halted 出力がクリアになった場合と同じイベントを送信。
2	プログラムトレース停止	プログラムトレースコマンドレジスタにコマンドを書き込むことによりプログラムトレースが停止したとき、トレースバッファーがフルになったとき、またはクロストリガーアクションによりイベントを生成。
3	プログラムトレース開始	プログラムトレースコマンドレジスタにコマンドを書き込むことによりプログラムトレースが開始したとき、トレースポイントに到達したとき、またはクロストリガーアクションによりイベントを生成。
4	パフォーマンス監視停止	パフォーマンスカウンターコマンドレジスタにコマンドを書き込むことによりパフォーマンス監視が停止したとき、またはクロストリガー アクションによりイベントを生成。
5	パフォーマンス監視開始	パフォーマンスカウンターコマンドレジスタにコマンドを書き込むことによりパフォーマンス監視が開始したとき、またはクロストリガー アクションによりイベントを生成。
6	プロファイリングのディスエーブル	プロファイリング制御レジスタにコマンドを書き込むことによりプロファイリングがイネーブルになったとき、またはクロストリガー アクションによりイベントを生成。
7	プロファイリングのイネーブル	プロファイリング制御レジスタにコマンドを書き込むことによりプロファイリングがディスエーブルになったとき、またはクロストリガー アクションによりイベントを生成。

トレース インターフェイスの概要

MicroBlaze のトレース インターフェイスは、パフォーマンスの監視および解析のため、多数の内部ステート信号をエクスポートします。



推奨: このトレース インターフェイスは、ザイリンクスが開発した解析コアを介してのみ使用することをお勧めします。

今後の MicroBlaze リリースでは、このインターフェイスに下位互換性がない可能性があります。エクスポートされる信号のリストは、[第 3 章「MicroBlaze 信号インターフェイスの説明」](#) の表 3-16 を参照してください。

フォールト トレランス

MicroBlaze に含まれるフォールト トレランス機能は、C_FAULT_TOLERANT を使用してイネーブルにすることができます、内部ブロック RAM にエラー検出機能を提供し(命令キャッシュ、データキャッシュ、分岐先キャッシュ、MMU で)、LMB ブロック RAM にエラー検出およびエラー訂正(ECC)のサポートを提供します。フォールト トレランスがイネーブルの場合、ブロック RAM のソフト エラーはすべて検出され訂正されるため、全体的なエラー発生率を大幅に下げることができます。

通常は、ブロック RAM だけではなく、FPGA のコンフィギュレーション メモリも保護する必要があります。このトピックの詳細および参考資料は、『Soft Error Mitigation Controller LogiCORE IP 製品ガイド』(PG036) [参照 2] および『UltraScale Architecture Soft Error Mitigation Controller LogiCORE IP 製品ガイド』(PG187) [参照 16] を参照してください。

フォールト トレランスをさらに増加させるには、多数決およびフォールト検出を処理するコアが追加された MicroBlaze に含まれる Triple Modular Redundancy (TMR) ソリューションを使用します。説明とインプリメンテーションの詳細は、『Triple Modular Redundancy (TMR) Subsystem 製品ガイド』(PG268) [参照 7] を参照してください。

設定

MicroBlaze コンフィギュレーションの使用

フォールト トレランスは、MicroBlaze の設定ダイアログ ボックスの [General] ページでオンにできます。

MicroBlaze でフォールト トレランスをイネーブルにすると、システムが生成されたときに接続されている LMB BRAM Interface Controller で ECC が自動的にイネーブルになります。フォールト トレランスおよび最小限の ECC サポートを有効にするのに、これ以外の設定は不要です。

接続されているすべての LMB BRAM Interface Controller の設定ダイアログ ボックスで、C_ECC をディスエーブルにして ECC サポートを手動で無効にし、LMB ブロック RAM を保護しないことも可能ですが、推奨されません。

この場合はフォールト トレランスはイネーブルなので、内部 MicroBalze ブロック RAM の保護は有効のままでです。

LMB BRAM Interface Controller の使用

先ほど説明した方法とは別 の方法で、接続されているすべての LMB BRAM Interface Controller の設定ダイアログ ボックスで ECC をイネーブルにすることも可能です。

この場合、システムが生成されたときに MicroBlaze でフォールト トレランスが自動的にイネーブルになります。ECC サポートおよび MicroBlaze フォールト トレランスを有効にするのに、これ以外の設定は不要です。

ECC はすべてのコントローラーでイネーブルまたはディスエーブルにする必要があり、これは DRC でチェックされます。

MicroBlaze の設定ダイアログ ボックスで C_FAULT_TOLERANT をディスエーブルにし、MicroBlaze でのフォールト トレランス サポートを手動で無効にすることは可能です。これは、MicroBlaze でブロック RAM が使用されていない場合、訂正不可能な ECC エラーからのバス例外を処理する必要がない場合以外はお勧めしません。

機能

MicroBlaze のフォールト トレランス機能は、次のとおりです。各機能の詳細は、次のセクションを参照してください。

- 「命令キャッシュ」
- 「データキャッシュ」
- 「UTLB 管理」
- 「分岐先キャッシュ」
- 「例外の原因」

LMB BRAM Interface Controller v4.0 以降のバージョンでは、LMB ECC インプリメンテーションが提供されています。パフォーマンス、リソース使用率などの詳細は、『LMB BRAM Interface Controller LogiCORE IP 製品ガイド』(PG112) [参照 3] を参照してください。

命令およびデータキャッシュ保護

命令およびデータキャッシュでブロック RAM を保護するには、parity が使用されます。parity エラーが検出されると、対応するキャッシュ ラインが無効になります。これにより、外部メモリから正しい値がキャッシュにリロードされます。キャッシュ ヒットが発生するたびに parity がチェックされます。

注記: これはライトスルーの場合にのみ機能するため、フォールト トレランスがイネーブルのときはライトバック データキャッシュは使用できません。これは DRC でチェックされます。

キャッシュのブロック RAM に新しい値が書き込まれると、parity も計算されて書き込まれます。タグ用に 1 parity ビット、命令キャッシュデータ用に 1 parity ビット、データキャッシュ ラインの各ワード用に 1 parity ビットが使用されます。

多くの場合、フォールト トレランスを有効にしても、スペア ビットを parity 用に使用できるので、必要なキャッシュ ブロック RAM の数は増加しません。フォールト トレランスを有効にする場合、リソース使用率の増加、特にブロック RAM の数などは、MicroBlaze の設定ダイアログ ボックスで簡単に確認できます。

メモリ管理ユニット (MMU) の保護

MMU の統合変換ルックアサイド バッファー (UTLB) でブロック RAM を保護するため、パリティが使用されます。アドレス変換中にパリティ エラーが検出されると、TLB ミス例外が発生し、ソフトウェアがエントリをリロードします。

TLBHI および TLBLO レジスタを使用して新しい TLB エントリを書き込むと、パリティが計算されます。各エントリで 1 パリティ ビットが使用されます。

TLBHI および TLBLO レジスタを使用して UTLB エントリを読み出すとき、パリティもチェックされます。この場合でパリティ エラーが検出されると、有効ビットがクリアになり、エントリは無効になります。

フォールト トレランスを有効にしても、スペア ビットをパリティに使用できるので、MMU のブロック RAM サイズは増加しません。

分岐先キャッシュ保護

分岐先キャッシュでブロック RAM を保護するため、パリティが使用されます。分岐先アドレスを検索しているときにパリティ エラーが検出されると、アドレスは無視され、標準分岐になります。

分岐先キャッシュに新しいアドレスが書き込まれると、パリティが計算されます。各アドレスに 1 パリティ ビットが使用されます。

フォールト トレランスを有効にしても、スペア ビットをパリティに使用できるので、分岐キャッシュのブロック RAM サイズは増加しません。

例外処理

フォールト トレランスが有効になっていて、LMB ブロック RAM でエラーが発生すると、LMB BRAM Interface Controller が LMB インターフェイスでエラー信号を生成します。

マシンステータスレジスタで EE ビットをセットして MicroBlaze プロセッサで例外を有効にしている場合、影響を受けるインターフェイスによって、訂正不可能なエラー信号で命令バス例外またはデータバス例外が生成されます。

例外処理中にバス例外が発生した場合、MicroBlaze は停止し、外部エラー信号 MB_Error がセットされます。これにより、訂正不可能なエラーにより破損した命令を実行することは不可能になります。

ソフトウェアサポート

スクラビング

ビット エラーがブロック RAM に累積されないようにするために、定期的にビット エラーをスクラブする必要があります。

特定のコンフィギュレーションで使用される LMB ブロック RAM 全体および MicroBlaze 内部ブロック RAM すべてをスクラブするため、スタンドアロン BSP には `microblaze_scrub()` という関数があります。この関数は、タイマー割り込みルーチンから定期的に呼び出されます。この関数が呼び出されるたびに各ブロック RAM の 1 つのロケーションスクラブされ、現在のロケーションは持続データを使用して監視されます。

次にそのコード例を示します。

```
#include "xparameters.h"
#include "xtmrctr.h"
#include "xintc.h"
#include "mb_interface.h"

#define SCRUB_PERIOD ...

XIntc InterruptController; /* The Interrupt Controller instance */
XTmrCtr TimerCounterInst; /* The Timer Counter instance */

void MicroBlazeScrubHandler(void *CallBackRef, u8 TmrCtrNumber)
{
    /* Perform other timer interrupt processing here */
    microblaze_scrub();
}

int main (void)
{
    int Status;

    /*
     * Initialize the timer counter so that it's ready to use,
     * specify the device ID that is generated in xparameters.h
     */
    Status = XTmrCtr_Initialize(&TimerCounterInst, TMRCTR_DEVICE_ID);
    if (Status != XST_SUCCESS) {
        return XST_FAILURE;
    }

    /*
     * Connect the timer counter to the interrupt subsystem such that
     * interrupts can occur.
     */
    Status = XIIntc_Initialize(&InterruptController, INTC_DEVICE_ID);
    if (Status != XST_SUCCESS) {
        return XST_FAILURE;
    }

    /*
     * Connect a device driver handler that will be called when an
     * interrupt for the device occurs, the device driver handler performs
     * the specific interrupt processing for the device
     */
}
```

```
    Status = XIIntc_Connect(&InterruptController, TMRCTR_DEVICE_ID,
                           (XIInterruptHandler)XTmrCtr_InterruptHandler,
                           (void *) &TimerCounterInst);
    if (Status != XST_SUCCESS) {
        return XST_FAILURE;
    }

    /*
     * Start the interrupt controller such that interrupts are enabled for
     * all devices that cause interrupts, specifying real mode so that the
     * timer counter can cause interrupts thru the interrupt controller.
     */
    Status = XIIntc_Start(&InterruptController, XIN_REAL_MODE);
    if (Status != XST_SUCCESS) {
        return XST_FAILURE;
    }

    /*
     * Setup the handler for the timer counter that will be called from the
     * interrupt context when the timer expires, specify a pointer to the
     * timer counter driver instance as the callback reference so the
     * handler is able to access the instance data
     */
    XTmrCtr_SetHandler(&TimerCounterInst, MicroBlazeScrubHandler,
                       &TimerCounterInst);

    /*
     * Enable the interrupt of the timer counter so interrupts will occur
     * and use auto reload mode such that the timer counter will reload
     * itself automatically and continue repeatedly, without this option
     * it would expire once only
     */
    XTmrCtr_SetOptions(&TimerCounterInst, TIMER_CNTR_0,
                      XTC_INT_MODE_OPTION | XTC_AUTO_RELOAD_OPTION);

    /*
     * Set a reset value for the timer counter such that it will expire
     * earlier than letting it roll over from 0, the reset value is loaded
     * into the timer counter when it is started
     */
    XTmrCtr_SetResetValue(TmrCtrInstancePtr, TmrCtrNumber, SCRUB_PERIOD);

    /*
     * Start the timer counter such that it's incrementing by default,
     * then wait for it to timeout a number of times
     */
    XTmrCtr_Start(&TimerCounterInst, TIMER_CNTR_0);

    ...
}
```

スクラブレートの計算方法など、スクラブのインプリメント方法の詳細は、「[スクラビング](#)」を参照してください。

ブロック RAM ドライバー

スタンダードアロンの BSP ブロック RAM ドライバーは、LMB BRAM Interface Controller の ECC レジスタにアクセスするために使用され、また包括的なセルフ テストも提供します。

SDK ザイリンクス C プロジェクトの「ペリフェラル テスト」をインプリメントすると、システムの各 LMB BRAM Interface Controller のブロック RAM セルフ テストを含むセルフ テスト例が生成されます。LMB BRAM Interface Controller で有効になっている ECC 機能によって、このコードにより ECC 機能のすべてのテストが実行されます。詳細は、SDK ヘルプ [参照 9] を参照してください。

セルフ テスト例は、スタンダードアロン BSP ブロック RAM ドライバーのソース コードにあり、通常は `microblaze_0/libsrc/bram_v3_03_a/src/xbram_selftest.c` というサブディレクトリにあります。

スクラビング

スクラビング方法

スクラビングは、ブロック RAM のタイプに特定の方法で実行されます。

- 命令およびデータ キャッシュ: キャッシュのすべてのラインは、それぞれ WIC および WDC 命令を使用して周期的に無効化されます。これにより、外部メモリからキャッシュ ラインがキャッシュに強制的にリロードされます。
- メモリ管理ユニット (MMU) 保護: UTLB のすべてのエントリは、有効ビットをクリアにして TLBHI レジスタを書き込むことにより、周期的に無効化されます。
- 分岐先キャッシュ (BTC): BTC 全体は、分岐 BRI4 を同期化することにより無効化されます。
- LMB ブロック RAM: メモリのすべてのアドレスが周期的に読み出しおよび書き込まれ、各アドレスの任意のシングル ビット エラーが訂正されます。

LMB BRAM Interface Controller からの訂正可能なエラー用に割り込みを追加して、割り込みハンドラーのこのアドレスをすぐにスクラブすることも可能ですが、ほとんどの場合は信頼性が多少向上するだけです。

エラーが発生しているアドレスは、各 LMB BRAM Interface Controller の訂正可能エラー ファースト フェイリング アドレス レジスタを読み出して、を検出できます。

割り込みを生成できるようにするには接続されている LMB BRAM Interface Controller で `C_ECC_STATUS_REGISTERS` を 1 に設定し、エラーの発生しているアドレスを読み出すには、`C_CE_FAILING_REGISTERS` を 1 に設定する必要があります。

スクラブルートの計算

スクラブルートは、エラー発生率および必要な信頼性によって決まります。

LMB メモリのスクラブルートは、次の式を使用して見積もることができます。

$$P_W \approx 760 \left(\frac{BER^2}{SR^2} \right)$$

P_W は 1 メモリ ワードでの訂正不可能なエラーの発生率、 BER は 1 メモリ ビットのソフト エラーレート、 SR はスクラブルートです。

各製品ファミリのブロック RAM に影響するソフト エラーレートについては、『デバイス信頼性レポート ユーザーガイド』(UG116) [参照 5] を参照してください。

ユース ケース

一般的なユース ケースをいくつか説明します。これらのユース ケースは、『LMB BRAM Interface Controller LogiCORE IP 製品ガイド』(PG112) [参照 3] からの抜粋です。

最小限

MicroBlaze でフォールト トレランスをイネーブルにし、ほかの設定を実行していない場合、このシステムになります。

エリア制約が厳しく、ECC 機能のテストやエラー頻度およびロケーションの解析が不要な場合に適しています。

ECC レジスタはインプリメントされません。シングルビット エラーは、MicroBlaze に渡される前に、ECC ロジックによって訂正されます。訂正不可能なエラーはエラー信号をセットし、MicroBlaze で例外が生成されます。

小型

このシステムは、エラーの頻度を監視する必要はあるが ECC 機能をテストする必要がない場合に使用してください。これは、最小限のシステムにシングルビット エラーレートを監視するための訂正可能エラーカウンターレジスタが追加されたものです。エラーレートが高い場合は、スクラブルートを増やし、シングルビット エラーが訂正不可能なダブルビット エラーになるリスクを最低限に抑えるようにしてください。パラメーターは、`C_ECC = 1` および `C_CE_COUNTER_WIDTH = 10` に設定します。

標準

このシステムは標準的なユース ケースを表しており、エラー頻度を監視する必要があり、ソフトウェアを介してシングルビット エラーをすぐに訂正するための割り込みを生成します。ECC 機能のテスト サポートはありません。

これは、小型システムに、訂正可能エラーファースト フェイリング レジスタおよびステータス レジスタが追加されたものです。シングルビット エラーが発生すると、訂正可能エラーファースト フェイリング レジスタにアクセスするためのアドレスがラッチされ、ECC ステータス レジスタの `CE_STATUS` ビットがセットされます。割り込みが生成されると、MicroBlaze がエラーの発生しているアドレスを読み出し、読み出しを実行してから書き込みを実行します。これで、ブロック RAM からシングルビット エラーが削除され、シングルビット エラーが訂正不可能なダブルビット エラーになるリスクが軽減します。パラメーター セットは、次のとおりです。

- `C_ECC = 1`
- `C_CE_COUNTER_WIDTH = 10`
- `C_ECC_STATUS_REGISTER = 1`
- `C_CE_FAILING_REGISTERS = 1`

フル

このシステムは、LMB BRAM Interface Controller で提供される機能をすべて使用し、フル エラー挿入機能だけでなく、エラー監視や割り込み生成もイネーブルにします。これは、標準システムに訂正可能なエラー ファースト フェイリング レジスタおよびフォールト挿入レジスタが追加されたものです。システム デバッグまたはフォールト トランクス要件が高いシステム用に ECC 機能を完全に制御できるよう、すべての機能がオンになっています。パラメータ セットは、次のとおりです。

- C_ECC = 1
- C_CE_COUNTER_WIDTH = 10
- C_ECC_STATUS_REGISTER = 1
- C_CE_FAILING_REGISTERS = 1
- C_UE_FAILING_REGISTERS = 1
- C_FAULT_INJECT = 1

ロックステップ操作

MicroBlaze はロックステップ コンフィギュレーションで動作可能です。このコンフィギュレーションでは、複数の同一 MicroBlaze コアが同じプログラムを実行します。コアの出力を比較して、不正アクセス、過渡エラー、恒久的なハードウェア エラーなどの問題を検出します。

システム コンフィギュレーション

マスター コア (プライマリ コア) 以外のシステムのスレーブ MicroBlaze コアでは、パラメーター C_LOCKSTEP_SLAVE は 1 に設定されます。マスター コアはすべての出力信号を駆動し、デバッグ機能を制御します。マスターのポート Lockstep_Master_Out は、デバッグを制御するため、スレーブのポート Lockstep_Slave_In に接続されています。

スレーブ コアは、出力信号を駆動せず、入力信号を受信するだけです。このため、スレーブの入力ポートには必ず信号を接続してください。バスの場合は、個々の入力ポートを明示的に接続しておく必要があります。

マスターおよびスレーブ コアのポート Lockstep_Out は、比較のための出力信号をすべて提供します。エラーが発生していないければ、各コアからの個々の信号はどのクロック サイクルでも同じです。

ロックステップ操作を正しく機能させるには、コアへのすべての入力信号が同期している必要があります。外部同期が必要な入力信号は、Interrupt、Reset、Ext_Brk、および Ext_Nm_Brk です。

ユース ケース

ここでは、一般的な 2 つのユース ケースを説明します。ロックステップ操作は、MicroBlaze コア レベルで Triple Modular Redundancy (TMR) をインプリメントするためのベースを提供します。

不正アクセス防止

このアプリケーションは、システムに不正アクセス防止機能が必要なユース ケースです。一般的な例は暗号アプリケーションです。

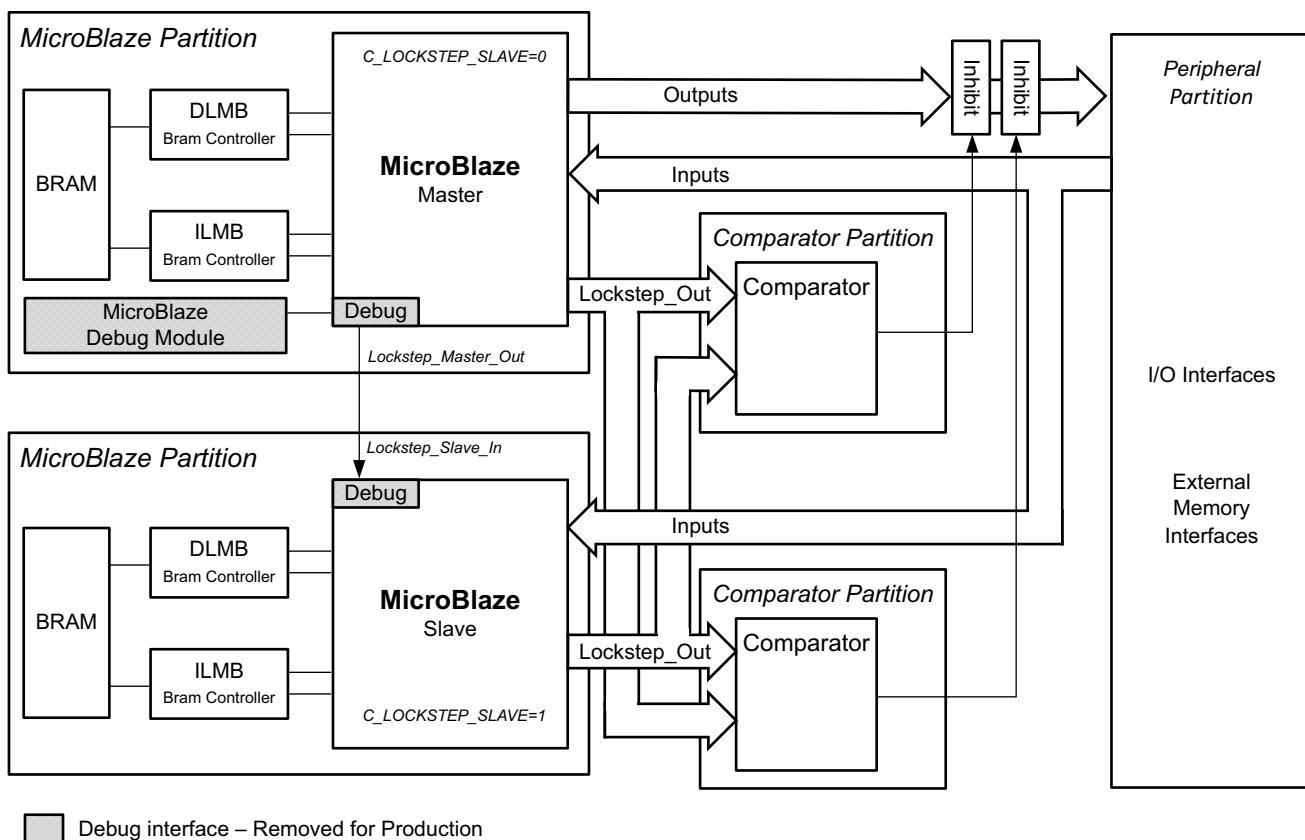
この方法では、2つの冗長 MicroBlaze プロセッサを使用し、専用ローカル メモリおよび冗長コンパレータをそれぞれ保護エリアに配置します。各プロセッサからの出力は、2つのコンパレータに入力され、それぞれのプロセッサがすべての入力信号のコピーを受信します。

冗長 MicroBlaze プロセッサは機能的に同じですが、互いに完全に独立していて、2つを接続する信号はありません。唯一の例外はデバッグ ロジックと関連信号ですが、デバッグはシステムの製品化および認証の前にディスエーブルになります。

マスターの MicroBlaze コアからの出力はシステムのペリフェラルを駆動します。保護エリアから送信されるデータは、すべてインヒビターを通してします。各インヒビターは関連付けられているコンパレータで制御されます。

デザインの各保護エリアは、階層 SCC (Single Chip Cryptography) フローを使用して、それぞれのパーティション内にインプリメントする必要があります。このフローの詳細および参考資料は、『階層デザイン設計手法ガイド』(UG748) [参照 8] を参照してください。

次に、システムのブロック図を示します。



X19777-111617

図 2-42: ロックステップ不正アクセス保護アプリケーション

エラー検出

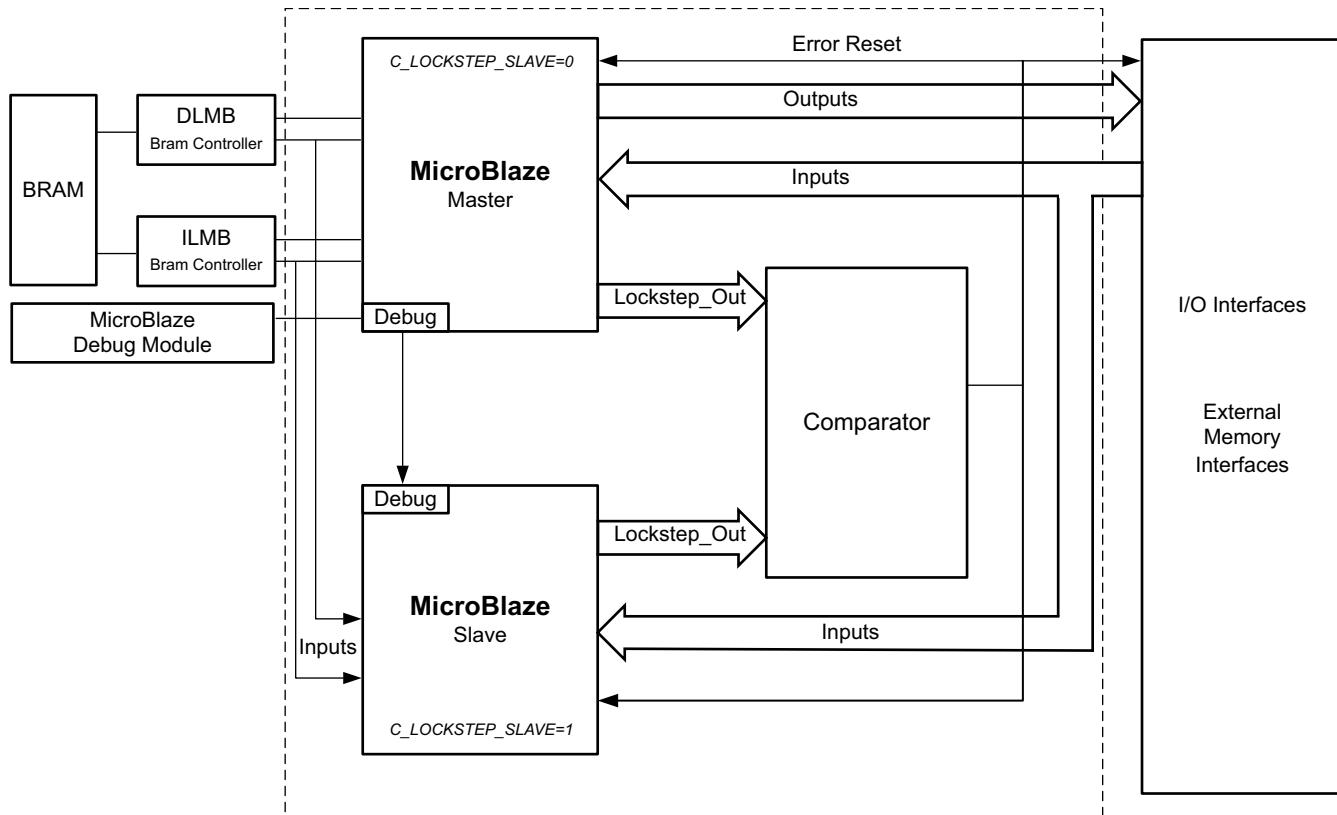
エラー検出のユース ケースでは、過渡エラーおよび恒久的エラーをすべて検出する必要があります。システムを使用可能な状態に維持するために冗長機能が利用されるフェイルセーフおよびフォールト トレンラント アプリケーションでは、これは重要です。

このシステムでは、2つの冗長 MicroBlaze プロセッサがロックステップで実行されます。2つのプロセッサの出力に不一致が検出されたときにエラーを出力するため、コンパレータが使用されます。どんなエラーが発生しても両方のプロセッサがすぐに停止し、それ以上エラーが拡散しないようにします。

デバッグ ロジックと関連信号を除き、冗長 MicroBlaze プロセッサは機能的に同じです。マスターの MicroBlaze コアの出力はシステムのペリフェラルを駆動します。スレーブの MicroBlaze コアには入力のみが接続されており、出力はすべてオープンです。

システムには、完全なフォールト トランジットのアプリケーションを設計するための基本ブロックが含まれていて、冗長性を持たせるには、1つまたは複数の追加ブロックを追加する必要があります。

次の図に、このユース ケースを示します。



X19778-111617

図 2-43: ロックステップ エラー検出アプリケーション

コヒーレンシ

MicroBlaze では、キャッシュ コヒーレンシだけでなく、『AMBA® AXI and ACE Protocol Specification』(Arm IHI 0022E) [参照 15] で定義されている AXI コヒーレンシ拡張(ACE)を使用したキャッシュおよび変換ルックアサイドバッファーの無効化もサポートされます。コヒーレンシ サポートは、C_INTERCONNECT が 3 (ACE) に設定されているときには有効になります。

ACE を使用すると、コヒーレンシ ドメインの MicroBlaze プロセッサのすべてのキャッシュ間のコヒーレンシを保つことができます。ペリフェラル ポート (AXI_IP、AXI_DP) およびローカル メモリ (ILMB、DLMB) は、コヒーレンシ ドメイン外です。

ライトバック データ キャッシュ、ワイド キャッシュ インターフェイス (32 ビットを超えるデータ)、命令 キャッシュ ストリーム、命令 キャッシュ ビクティムに対して、またはエリア最適化が有効になっていると、コヒーレンシはサポートされません。また、C_ICACHE_ALWAYS_USED および C_DCACHE_ALWAYS_USED の両方を 1 に設定する必要があります。

無効化

コヒーレンシ ハードウェアは、次の場合に無効化を処理します。

- データ キャッシュ無効化: コヒーレンシ ドメインの MicroBlaze コアが外部 キャッシュ無効化命令 (WDC_EXT_CLEAR または WDC_EXT_FLUSH) を使用してデータ キャッシュ ラインを無効にすると、コヒーレンシ ドメインのその他すべてのコアにも同じ処理が実行されることを示すハードウェア メッセージが表示されます。物理アドレスが常に使用されます。
- 命令 キャッシュ無効化: コヒーレンシ ドメインの MicroBlaze コアが命令 キャッシュ ラインを無効化にすると、コヒーレンシ ドメインのその他すべてのコアにも同じ処理が実行されることを示すハードウェア メッセージが表示されます。MMU が仮想モードの場合は、仮想アドレスが使用され、そうでない場合は物理アドレスが使用されます。
- MMU TLB 無効化: コヒーレンシ ドメインの MicroBlaze コアが UTLB のエントリを無効にすると (TLBHI に 0 の有効フラグを書き込む)、コヒーレンシ ドメインのその他すべてのコアでも、UTLB のすべてのエントリで無効化された仮想アドレスに一致するタグを持つものが無効になり、そのシャドウ TLB が空になることを示すハードウェア メッセージが表示されます。

エントリを比較するときに TID は考慮されないので、ほかのプロセスに属すエントリが無効になる可能性があります。これらのエントリが後でアクセスされると、TLB ミス例外が生成され、ソフトウェアで処理する必要があります。

MMU ページを無効にする前に、コヒーレンシ ドメイン内でハードウェア無効化が伝搬されていることを確認するため、UTLB にそのページをロードする必要があります。コヒーレンシ ドメインのほかのプロセッサの TLB にもこのエントリが格納されている可能性があるので、メモリのページを無効にするだけでは不十分です。

MicroBlaze コアが 1 つ以上のエントリを無効にしたら、ほかのすべてのプロセッサでも TLB 無効化が完了したことを確認するため、メモリ バリア命令 (MBAR) を実行する必要があります。

- 分岐先 キャッシュ無効化: コヒーレンシ ドメインの MicroBlaze コアが、メモリ バリア命令または同期分岐を使用して分岐先 キャッシュを無効にすると、コヒーレンシ ドメインのその他すべてのコアにも同じ処理が実行されることを示すハードウェア メッセージが表示されます。

これは、「[自己変更コード](#)」に示されているガイドラインに従っていれば、マルチプロセッサ システムのコヒーレンシ ドメイン内で自己変更コードを透過的に使用できることを意味します。

プロトコル準拠

MicroBlaze の命令キャッシュ インターフェイスは、ACE トランザクションの次のサブセットを発行します。

- ReadClean: キャッシュ ラインが割り当てられたときに発行。
- ReadOnce: キャッシュがオフのとき、または MMU の抑止キャッシング ビットがキャッシュ ラインに対してセットされたときに発行。

MicroBlaze のデータ キャッシュ インターフェイスは、ACE トランザクションの次のサブセットを発行します。

- ReadClean: キャッシュ ラインが割り当てられたときに発行。
- CleanUnique: 排他的アクセス シーケンスの一部として SWX 命令が実行されたときに発行。
- ReadOnce: キャッシュがオフのとき、または MMU の抑止キャッシング ビットがキャッシュ ラインに対してセットされたときに発行。
- WriteUnique: ストア命令が書き込みを実行するたびに発行。
- CleanInvalid: WDC.EXT.FLUSH 命令が実行されたときに発行。
- MakeInvalid: WDC.EXT.CLEAR 命令が実行されたときに発行。

両方のインターフェイスは、分散仮想メモリ (DVM) トランザクションの次のサブセットを発行します。

- DVM 操作
 - TLB Invalidate: VA によるハイパーバイザ TLB 無効化
 - Branch Predictor Invalidate: 分岐予測すべて無効化
 - Physical Instruction Cache Invalidate: PA による非セキュア物理命令キャッシュ無効化 (仮想インデックスなし)
 - Virtual Instruction Cache Invalidate: VA によるハイパーバイザ無効化
- DVM 同期
 - 同期化
- DVM 完了
 - 上記の DVM トランザクションに加え、インターフェイスは CleanInvalid および MakeInvalid トランザクションのみを受け付けます。これらのトランザクションは命令キャッシュでは効果ではなく、指定されたデータ キャッシュ ラインを無効化します。ほかのトランザクションが受信されると、動作は未定義になります。
 - 「キャッシュ インターフェイス」で説明されているように、インターフェイスででは AXI4 トランザクションのサブセットのみが使用されます。

データおよび命令のアドレス拡張

MicroBlaze では、C_ADDR_SIZE パラメーターを使用して 16 EBまでのデータをアドレス指定できます。また、32 ビット MicroBlaze では、C_USE_MMU = 3(仮想) を設定して MMU 物理アドレス拡張(PAE)をイネーブルにすると、16EBまでの物理命令アドレスをサポートできます。

64 ビット MicroBlaze では、仮想アドレスと物理アドレスの両方が C_ADDR_SIZE パラメーターに基づいて拡張されます。これは命令およびデータアドレス空間の両方に適用されるので、ここにリストされている 32 ビット MicroBlaze を使用する場合のすべての制限を除去できます。

C_ADDR_SIZE パラメーターに設定可能な値は次のとおりです。

◦ NONE	$4 * 1024^3$ バイト	32 ビット アドレス、拡張アドレス命令または PAE なし
◦ 64 GB	$64 * 1024^3$ バイト	36 ビット アドレス
◦ 1 TB	1024^4 バイト	40 ビット アドレス
◦ 16 TB	$16 * 1024^4$ バイト	44 ビット アドレス
◦ 256 TB	$256 * 1024^4$ バイト	48 ビット アドレス
◦ 4PB	$4 * 1024^5$ バイト	52 ビット アドレス
◦ 16EB	$16 * 1024^6$ バイト	64 ビット アドレス

32 ビット MicroBlaze でアドレス拡張を使用する場合、ソフトウェア上の制限が多数あります。

- 32 ビット MicroBlaze では、GNU ツールで 32 ビット アドレスの ELF ファイルしか生成されません。つまり、プログラム命令およびデータメモリはアドレス空間の最初の 4 GB に配置する必要があります。PAE がイネーブルになっていなければ命令アドレス空間で拡張アドレスを使用できないのもこのためです。

PAE がイネーブルの場合は、ほとんどのプログラム命令およびデータはどの物理アドレスにでも配置できますが、リアルモードで実行されるソフトウェアはすべてアドレス空間の最初の 4 GB に配置する必要があります。仮想モードをアクティブにするには、リアルモードで実行されるソフトウェアによる仮想アドレスから物理アドレスへの変換を設定するため、MMU UTLB も初期化する必要があります。

- すべてのソフトウェアドライバーで 32 ビットの符号なし整数のアドレスポインターが使用されるため、ドライバーコードを変更せずに 4 GB を超える物理拡張アドレスにアクセスすることはできません。そのため、すべての AXI ペリフェラルはアドレス空間の最初の 4 GB 内に配置する必要があります。

PAE がイネーブルの場合、仮想アドレスがアドレス空間の最初の 4 GB にあれば、AXI ペリフェラルはどの物理アドレスにでも配置できます。

- 拡張アドレスは物理アドレスとしてのみ処理され、拡張仮想アドレスを物理アドレスに変換するのに MMU を使用することはできません。

つまり、PAE サポートがない場合、Linux ではリアルモードで動作している専用ドライバーを介したデータアドレス拡張しか使用できません。

MMU がイネーブルの場合は、C_MMU_PRIVILEGED_INSTR パラメーターが適切に設定されている場合を除き、拡張アドレスのロードおよびストア命令は特権命令となります。可能であれば、命令で MMU 変換がバイパスされ、拡張アドレスが物理アドレスとして扱われます。

- GNU コンパイラでは 64 ビット アドレス ポインターは処理されません。つまり、PAE がイネーブルになっていなければ、拡張アドレスにアクセスするには、マクロとして利用可能な特定拡張アドレス指定命令を使用するが唯一の方法です。

次の C コードに、拡張アドレスを使用してデータにアクセスする方法を示します。

```
#include "xil_types.h"
#include "mb_interface.h"

int main()
{
    u64 Addr = 0x000000FF00000000LL; /* Extended address */
    u32 Word;
    u8 Byte;

    Word = lwea(Addr); /* Load word from extended address */
    swea(Addr, Word); /* Store word to extended address */
    Byte = lbuea(Addr); /* Load byte from extended address */
    sbea(Addr, Byte); /* Store byte to extended address */
}
```

MicroBlaze 信号インターフェイスの説明

はじめに

この章では、MicroBlaze™ プロセッサを接続するために使用可能な信号インターフェイスの種類を説明します。

概要

MicroBlaze コアは、データと命令のアクセスそれぞれにバス インターフェイス ユニットを持つハーバード アーキテクチャとして構成されています。サポートされているメモリインターフェイスは、ローカル メモリ バス (LMB)、AMBA® AXI4 インターフェイス (AXI4) および ACE インターフェイス (ACE) です。

LMB は、オンチップのデュアルポート ブロック RAM へのシングル サイクル アクセスを提供し、AXI4 インターフェイスは、オンチップとオフチップ両方のペリフェラルおよびメモリへの接続を提供し、ACE インターフェイスは、メモリへのキャッシュ コヒーレント接続を提供します。

また、MicroBlaze では、最大 16 個の AXI4-Stream インターフェイス ポートがサポートされ、各ポートにマスターが 1 つ、スレーブ インターフェイスが 1 つあります。

機能

MicroBlaze は、次のバス インターフェイスで設定可能です。

- ペリフェラルインターフェイス用の AMBA AXI4 インターフェイス、およびキャッシュインターフェイス用の AMBA AXI4 または AXI コヒーレンシ拡張(ACE)インターフェイス(詳細は Arm® 『AMBA® AXI and ACE Protocol Specification』(Arm IHI 0022E [参照 15] を参照))。
- 効率よくブロック RAM 転送するための簡単な同期プロトコルを提供する LMB
- 高速でノンアービトレーテッドのストリーミング通信を提供する AXI4-Stream
- MDM (Microprocessor Debug Module) コアで使用するデバッグインターフェイス
- パフォーマンス解析用のトレースインターフェイス

MicroBlaze の I/O 概要

次の図および表3-1に示すコアインターフェイスは、次のように定義されます。

- **M_AXI_DP:** ペリフェラルデータインターフェイス、AXI4-Lite または AXI4インターフェイス
- **DLMB:** データインターフェイス、ローカルメモリバス(ブロックRAMのみ)
- **M_AXI_IP:** ペリフェラル命令インターフェイス、AXI4-Liteインターフェイス
- **ILMB:** 命令インターフェイス、ローカルメモリバス(ブロックRAMのみ)
- **M0_AXIS..M15_AXIS:** AXI4-Streamインターフェイスマスター直接接続インターフェイス
- **S0_AXIS..S15_AXIS:** AXI4-Streamインターフェイススレーブ直接接続インターフェイス
- **M_AXI_DC:** データ側キャッシュAXI4インターフェイス
- **M_ACE_DC:** データ側キャッシュACIコヒーレンシ拡張(ACE)インターフェイス
- **M_AXI_IC:** 命令側キャッシュAXI4インターフェイス
- **M_ACE_IC:** 命令側キャッシュACIコヒーレンシ拡張(ACE)インターフェイス
- **コア:** クロック、リセット、割り込み、デバッグ、トレース用のさまざまな信号

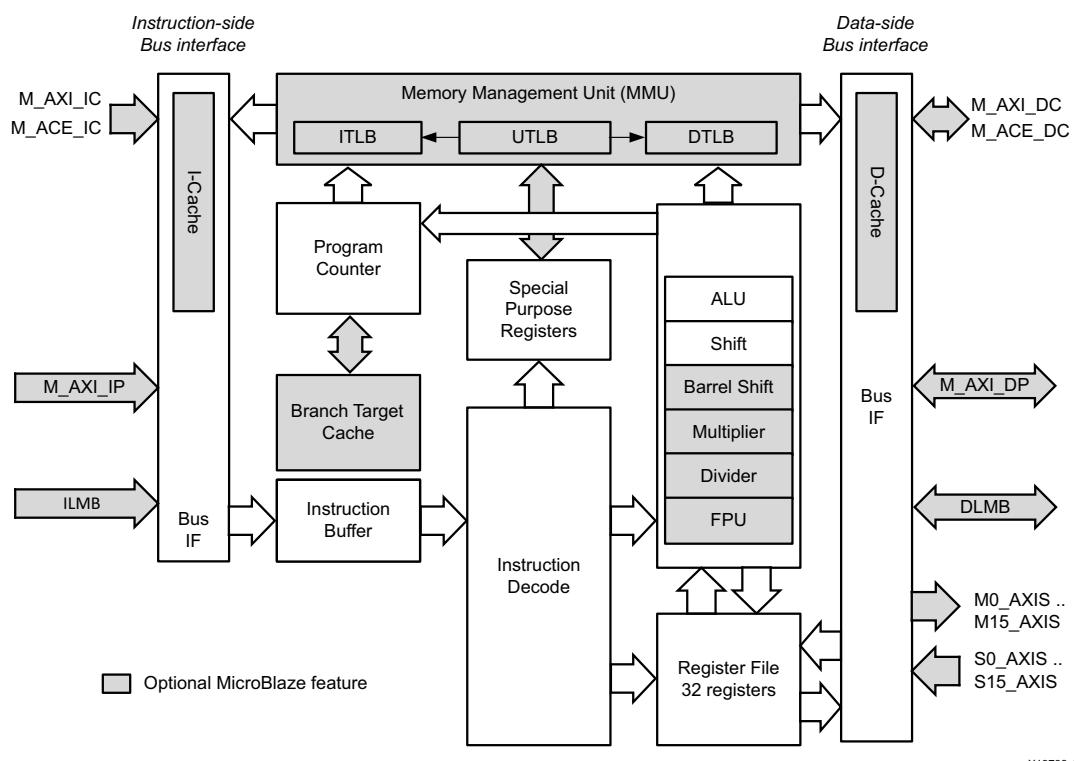


図3-1: MicroBlazeコアのブロック図

X19738-100218

表3-1: MicroBlazeコアI/Oのまとめ

信号	インターフェイス	I/O	説明
M_AXI_DP_AWID	M_AXI_DP	O	マスター書き込みアドレスID
M_AXI_DP_AWADDR	M_AXI_DP	O	マスター書き込みアドレス
M_AXI_DP_AWLEN	M_AXI_DP	O	マスターバースト長
M_AXI_DP_AWSIZE	M_AXI_DP	O	マスターバーストサイズ
M_AXI_DP_AWBURST	M_AXI_DP	O	マスター バースト タイプ
M_AXI_DP_AWLOCK	M_AXI_DP	O	マスター ロック タイプ
M_AXI_DP_AWCACHE	M_AXI_DP	O	マスター キャッシュ タイプ
M_AXI_DP_AWPROT	M_AXI_DP	O	マスター保護 タイプ
M_AXI_DP_AWQOS	M_AXI_DP	O	マスターのサービス品質 (QoS)
M_AXI_DP_AWVALID	M_AXI_DP	O	マスター書き込みアドレス有効
M_AXI_DP_AWREADY	M_AXI_DP	I	スレーブ書き込みアドレス準備完了
M_AXI_DP_WDATA	M_AXI_DP	O	マスター書き込みデータ
M_AXI_DP_WSTRB	M_AXI_DP	O	マスター書き込みストローブ
M_AXI_DP_WLAST	M_AXI_DP	O	マスター書き込み最終
M_AXI_DP_WVALID	M_AXI_DP	O	マスター書き込み有効
M_AXI_DP_WREADY	M_AXI_DP	I	スレーブ書き込み準備完了
M_AXI_DP_BID	M_AXI_DP	I	スレーブ応答ID
M_AXI_DP_BRESP	M_AXI_DP	I	スレーブ書き込み応答
M_AXI_DP_BVALID	M_AXI_DP	I	スレーブ書き込み応答有効
M_AXI_DP_BREADY	M_AXI_DP	O	マスター応答準備完了
M_AXI_DP_ARID	M_AXI_DP	O	マスター読み出しアドレスID
M_AXI_DP_ARADDR	M_AXI_DP	O	マスター読み出しアドレス
M_AXI_DP_ARLEN	M_AXI_DP	O	マスターバースト長
M_AXI_DP_ARSIZE	M_AXI_DP	O	マスターバーストサイズ
M_AXI_DP_ARBURST	M_AXI_DP	O	マスター バースト タイプ
M_AXI_DP_ARLOCK	M_AXI_DP	O	マスター ロック タイプ
M_AXI_DP_ARCACHE	M_AXI_DP	O	マスター キャッシュ タイプ
M_AXI_DP_ARPROT	M_AXI_DP	O	マスター保護 タイプ
M_AXI_DP_ARQOS	M_AXI_DP	O	マスターのサービス品質 (QoS)
M_AXI_DP_ARVALID	M_AXI_DP	O	マスター読み出しアドレス有効
M_AXI_DP_ARREADY	M_AXI_DP	I	スレーブ読み出しアドレス準備完了
M_AXI_DP_RID	M_AXI_DP	I	スレーブ読み出しIDタグ
M_AXI_DP_RDATA	M_AXI_DP	I	スレーブ読み出しだанны
M_AXI_DP_RRESP	M_AXI_DP	I	スレーブ読み出し応答
M_AXI_DP_RLAST	M_AXI_DP	I	スレーブ読み出し最終

表3-1: MicroBlazeコアI/Oのまとめ(続き)

信号	インターフェイス	I/O	説明
M_AXI_DP_RVALID	M_AXI_DP	I	スレーブ読み出し有効
M_AXI_DP_RREADY	M_AXI_DP	O	マスター読み出し準備完了
M_AXI_IP_AWID	M_AXI_IP	O	マスター書き込みアドレスID
M_AXI_IP_AWADDR	M_AXI_IP	O	マスター書き込みアドレス
M_AXI_IP_AWLEN	M_AXI_IP	O	マスターバースト長
M_AXI_IP_AWSIZE	M_AXI_IP	O	マスターバーストサイズ
M_AXI_IP_AWBURST	M_AXI_IP	O	マスターバーストタイプ
M_AXI_IP_AWLOCK	M_AXI_IP	O	マスターロックタイプ
M_AXI_IP_AWCACHE	M_AXI_IP	O	マスターキャッシュタイプ
M_AXI_IP_AWPROT	M_AXI_IP	O	マスター保護タイプ
M_AXI_IP_AWQOS	M_AXI_IP	O	マスターのサービス品質(QoS)
M_AXI_IP_AWVALID	M_AXI_IP	O	マスター書き込みアドレス有効
M_AXI_IP_AWREADY	M_AXI_IP	I	スレーブ書き込みアドレス準備完了
M_AXI_IP_WDATA	M_AXI_IP	O	マスター書き込みデータ
M_AXI_IP_WSTRB	M_AXI_IP	O	マスター書き込みストローブ
M_AXI_IP_WLAST	M_AXI_IP	O	マスター書き込み最終
M_AXI_IP_WVALID	M_AXI_IP	O	マスター書き込み有効
M_AXI_IP_WREADY	M_AXI_IP	I	スレーブ書き込み準備完了
M_AXI_IP_BID	M_AXI_IP	I	スレーブ応答ID
M_AXI_IP_BRESP	M_AXI_IP	I	スレーブ書き込み応答
M_AXI_IP_BVALID	M_AXI_IP	I	スレーブ書き込み応答有効
M_AXI_IP_BREADY	M_AXI_IP	O	マスター応答準備完了
M_AXI_IP_ARID	M_AXI_IP	O	マスター読み出しアドレスID
M_AXI_IP_ARADDR	M_AXI_IP	O	マスター読み出しアドレス
M_AXI_IP_ARLEN	M_AXI_IP	O	マスターバースト長
M_AXI_IP_ARSIZE	M_AXI_IP	O	マスターバーストサイズ
M_AXI_IP_ARBURST	M_AXI_IP	O	マスターバーストタイプ
M_AXI_IP_ARLOCK	M_AXI_IP	O	マスターロックタイプ
M_AXI_IP_ARCACHE	M_AXI_IP	O	マスターキャッシュタイプ
M_AXI_IP_ARPROT	M_AXI_IP	O	マスター保護タイプ
M_AXI_IP_ARQOS	M_AXI_IP	O	マスターのサービス品質(QoS)
M_AXI_IP_ARVALID	M_AXI_IP	O	マスター読み出しアドレス有効
M_AXI_IP_ARREADY	M_AXI_IP	I	スレーブ読み出しアドレス準備完了
M_AXI_IP_RID	M_AXI_IP	I	スレーブ読み出しIDタグ
M_AXI_IP_RDATA	M_AXI_IP	I	スレーブ読み出しだデータ

表3-1: MicroBlazeコアI/Oのまとめ(続き)

信号	インターフェイス	I/O	説明
M_AXI_IP_RRESP	M_AXI_IP	I	スレーブ読み出し応答
M_AXI_IP_RLAST	M_AXI_IP	I	スレーブ読み出し最終
M_AXI_IP_RVALID	M_AXI_IP	I	スレーブ読み出し有効
M_AXI_IP_RREADY	M_AXI_IP	O	マスター読み出し準備完了
M_AXI_DC_AWADDR	M_AXI_DC	O	マスター書き込みアドレス
M_AXI_DC_AWLEN	M_AXI_DC	O	マスターバースト長
M_AXI_DC_AWSIZE	M_AXI_DC	O	マスターバーストサイズ
M_AXI_DC_AWBURST	M_AXI_DC	O	マスターバーストタイプ
M_AXI_DC_AWLOCK	M_AXI_DC	O	マスターロックタイプ
M_AXI_DC_AWCACHE	M_AXI_DC	O	マスターキャッシュタイプ
M_AXI_DC_AWPROT	M_AXI_DC	O	マスター保護タイプ
M_AXI_DC_AWQOS	M_AXI_DC	O	マスターのサービス品質(QoS)
M_AXI_DC_AWVALID	M_AXI_DC	O	マスター書き込みアドレス有効
M_AXI_DC_AWREADY	M_AXI_DC	I	スレーブ書き込みアドレス準備完了
M_AXI_DC_AWUSER	M_AXI_DC	O	マスター書き込みアドレスユーザー信号
M_AXI_DC_AWDOMAIN	M_AXI_DC	O	マスター書き込みアドレスドメイン
M_AXI_DC_AWSNOOP	M_AXI_DC	O	マスター書き込みアドレススヌープ
M_AXI_DC_AWBAR	M_AXI_DC	O	マスター書き込みアドレスバリア
M_AXI_DC_WDATA	M_AXI_DC	O	マスター書き込みデータ
M_AXI_DC_WSTRB	M_AXI_DC	O	マスター書き込みストローブ
M_AXI_DC_WLAST	M_AXI_DC	O	マスター書き込み最終
M_AXI_DC_WVALID	M_AXI_DC	O	マスター書き込み有効
M_AXI_DC_WREADY	M_AXI_DC	I	スレーブ書き込み準備完了
M_AXI_DC_WUSER	M_AXI_DC	O	マスター書き込みユーザー信号
M_AXI_DC_BRESP	M_AXI_DC	I	スレーブ書き込み応答
M_AXI_DC_BID	M_AXI_DC	I	スレーブ応答ID
M_AXI_DC_BVALID	M_AXI_DC	I	スレーブ書き込み応答有効
M_AXI_DC_BREADY	M_AXI_DC	O	マスター応答準備完了
M_AXI_DC_BUSER	M_AXI_DC	I	スレーブ書き込み応答ユーザー信号
M_AXI_DC_WACK	M_AXI_DC	O	スレーブ書き込み肯定応答
M_AXI_DC_ARID	M_AXI_DC	O	マスター読み出しアドレスID
M_AXI_DC_ARADDR	M_AXI_DC	O	マスター読み出しアドレス
M_AXI_DC_ARLEN	M_AXI_DC	O	マスターバースト長
M_AXI_DC_ARSIZE	M_AXI_DC	O	マスターバーストサイズ
M_AXI_DC_ARBURST	M_AXI_DC	O	マスターバーストタイプ

表 3-1: MicroBlaze コア I/O のまとめ (続き)

信号	インター フェイス	I/O	説明
M_AXI_DC_ARLOCK	M_AXI_DC	O	マスター ロック タイプ
M_AXI_DC_ARCACHE	M_AXI_DC	O	マスター キャッシュ タイプ
M_AXI_DC_ARPROT	M_AXI_DC	O	マスター 保護 タイプ
M_AXI_DC_ARQOS	M_AXI_DC	O	マスターのサービス品質 (QoS)
M_AXI_DC_ARVALID	M_AXI_DC	O	マスター 読み出し アドレス 有効
M_AXI_DC_ARREADY	M_AXI_DC	I	スレーブ 読み出し アドレス 準備完了
M_AXI_DC_ARUSER	M_AXI_DC	O	マスター 読み出し アドレス ユーザー 信号
M_AXI_DC_ARDOMAIN	M_AXI_DC	O	マスター 読み出し アドレス ドメイン
M_AXI_DC_ARSNOOP	M_AXI_DC	O	マスター 読み出し アドレス スヌープ
M_AXI_DC_ARBAR	M_AXI_DC	O	マスター 読み出し アドレス バリア
M_AXI_DC RID	M_AXI_DC	I	スレーブ 読み出し ID タグ
M_AXI_DC_RDATA	M_AXI_DC	I	スレーブ 読み出し データ
M_AXI_DC_RRESP	M_AXI_DC	I	スレーブ 読み出し 応答
M_AXI_DC_RLAST	M_AXI_DC	I	スレーブ 読み出し 最終
M_AXI_DC_RVALID	M_AXI_DC	I	スレーブ 読み出し 有効
M_AXI_DC_RREADY	M_AXI_DC	O	マスター 読み出し 準備完了
M_AXI_DC_RUSER	M_AXI_DC	I	スレーブ 読み出し ユーザー 信号
M_AXI_DC_RACK	M_AXI_DC	O	マスター 読み出し 肯定応答
M_AXI_DC_ACVALID	M_AXI_DC	I	スレーブ スヌープ アドレス 有効
M_AXI_DC_ACADDR	M_AXI_DC	I	スレーブ スヌープ アドレス
M_AXI_DC_ACNSNOOP	M_AXI_DC	I	スレーブ スヌープ アドレス スヌープ
M_AXI_DC_ACPROT	M_AXI_DC	I	スレーブ スヌープ アドレス 保護 タイプ
M_AXI_DC_ACREADY	M_AXI_DC	O	マスター スヌープ 準備完了
M_AXI_DC_CRREADY	M_AXI_DC	I	スレーブ スヌープ 応答 準備完了
M_AXI_DC_CVALID	M_AXI_DC	O	マスター スヌープ 応答 有効
M_AXI_DC_CRRESP	M_AXI_DC	O	マスター スヌープ 応答
M_AXI_DC_CDVALID	M_AXI_DC	O	マスター スヌープ データ 有効
M_AXI_DC_CDREADY	M_AXI_DC	I	スレーブ スヌープ データ 準備完了
M_AXI_DC_CDDATA	M_AXI_DC	O	マスター スヌープ データ
M_AXI_DC_CDLAST	M_AXI_DC	O	マスター スヌープ データ 最終
M_AXI_IC_AWID	M_AXI_IC	O	マスター 書き込み アドレス ID
M_AXI_IC_AWADDR	M_AXI_IC	O	マスター 書き込み アドレス
M_AXI_IC_AWLEN	M_AXI_IC	O	マスター バースト 長
M_AXI_IC_AWSIZE	M_AXI_IC	O	マスター バースト サイズ
M_AXI_IC_AWBURST	M_AXI_IC	O	マスター バースト タイプ

表 3-1: MicroBlaze コア I/O のまとめ (続き)

信号	インター フェイス	I/O	説明
M_AXI_IC_AWLOCK	M_AXI_IC	O	マスター ロック タイプ
M_AXI_IC_AWCACHE	M_AXI_IC	O	マスター キャッシュ タイプ
M_AXI_IC_AWPROT	M_AXI_IC	O	マスター 保護 タイプ
M_AXI_IC_AWQOS	M_AXI_IC	O	マスターのサービス品質 (QoS)
M_AXI_IC_AWVALID	M_AXI_IC	O	マスター書き込みアドレス有効
M_AXI_IC_AWREADY	M_AXI_IC	I	スレーブ書き込みアドレス準備完了
M_AXI_IC_AWUSER	M_AXI_IC	O	マスター書き込みアドレス ユーザー信号
M_AXI_IC_AWDOMAIN	M_AXI_IC	O	マスター書き込みアドレス ドメイン
M_AXI_IC_AWSNOOP	M_AXI_IC	O	マスター書き込みアドレス スヌープ
M_AXI_IC_AWBAR	M_AXI_IC	O	マスター書き込みアドレス バリア
M_AXI_IC_WDATA	M_AXI_IC	O	マスター書き込みデータ
M_AXI_IC_WSTRB	M_AXI_IC	O	マスター書き込みストローブ
M_AXI_IC_WLAST	M_AXI_IC	O	マスター書き込み最終
M_AXI_IC_WVALID	M_AXI_IC	O	マスター書き込み有効
M_AXI_IC_WREADY	M_AXI_IC	I	スレーブ書き込み準備完了
M_AXI_IC_WUSER	M_AXI_IC	O	マスター書き込みユーザー信号
M_AXI_IC_BID	M_AXI_IC	I	スレーブ応答 ID
M_AXI_IC_BRESP	M_AXI_IC	I	スレーブ書き込み応答
M_AXI_IC_BVALID	M_AXI_IC	I	スレーブ書き込み応答有効
M_AXI_IC_BREADY	M_AXI_IC	O	マスター応答準備完了
M_AXI_IC_BUSER	M_AXI_IC	I	スレーブ書き込み応答ユーザー信号
M_AXI_IC_WACK	M_AXI_IC	O	スレーブ書き込み肯定応答
M_AXI_IC_ARID	M_AXI_IC	O	マスター読み出しアドレス ID
M_AXI_IC_ARADDR	M_AXI_IC	O	マスター読み出しアドレス
M_AXI_IC_ARLEN	M_AXI_IC	O	マスター バースト長
M_AXI_IC_ARSIZE	M_AXI_IC	O	マスター バースト サイズ
M_AXI_IC_ARBURST	M_AXI_IC	O	マスター バースト タイプ
M_AXI_IC_ARLOCK	M_AXI_IC	O	マスター ロック タイプ
M_AXI_IC_ARCACHE	M_AXI_IC	O	マスター キャッシュ タイプ
M_AXI_IC_ARPROT	M_AXI_IC	O	マスター 保護 タイプ
M_AXI_IC_ARQOS	M_AXI_IC	O	マスターのサービス品質 (QoS)
M_AXI_IC_ARVALID	M_AXI_IC	O	マスター読み出しアドレス有効
M_AXI_IC_ARREADY	M_AXI_IC	I	スレーブ読み出しアドレス準備完了
M_AXI_IC_ARUSER	M_AXI_IC	O	マスター読み出しアドレス ユーザー信号
M_AXI_IC_ARDOMAIN	M_AXI_IC	O	マスター読み出しアドレス ドメイン

表 3-1: MicroBlaze コア I/O のまとめ (続き)

信号	インター フェイス	I/O	説明
M_AXI_IC_ARSNOPP	M_AXI_IC	O	マスター読み出しアドレススヌープ
M_AXI_IC_ARBAR	M_AXI_IC	O	マスター読み出しアドレスバリア
M_AXI_IC_RID	M_AXI_IC	I	スレーブ読み出し ID タグ
M_AXI_IC_RDATA	M_AXI_IC	I	スレーブ読み出しデータ
M_AXI_IC_RRESP	M_AXI_IC	I	スレーブ読み出し応答
M_AXI_IC_RLAST	M_AXI_IC	I	スレーブ読み出し最終
M_AXI_IC_RVALID	M_AXI_IC	I	スレーブ読み出し有効
M_AXI_IC_RREADY	M_AXI_IC	O	マスター読み出し準備完了
M_AXI_IC_RUSER	M_AXI_IC	I	スレーブ読み出しユーザー信号
M_AXI_IC_RACK	M_AXI_IC	O	マスター読み出し肯定応答
M_AXI_IC_ACVALID	M_AXI_IC	I	スレーブスヌープアドレス有効
M_AXI_IC_ACADDR	M_AXI_IC	I	スレーブスヌープアドレス
M_AXI_IC_ACNSNOOP	M_AXI_IC	I	スレーブスヌープアドレススヌープ
M_AXI_IC_ACPROT	M_AXI_IC	I	スレーブスヌープアドレス保護タイプ
M_AXI_IC_ACREADY	M_AXI_IC	O	マスタースヌープ準備完了
M_AXI_IC_CRREADY	M_AXI_IC	I	スレーブスヌープ応答準備完了
M_AXI_IC_CVALID	M_AXI_IC	O	マスタースヌープ応答有効
M_AXI_IC_CRRESP	M_AXI_IC	O	マスタースヌープ応答
M_AXI_IC_CDVALID	M_AXI_IC	O	マスタースヌープデータ有効
M_AXI_IC_CDREADY	M_AXI_IC	I	スレーブスヌープデータ準備完了
M_AXI_IC_CDDATA	M_AXI_IC	O	マスタースヌープデータ
M_AXI_IC_CDLAST	M_AXI_IC	O	マスタースヌープデータ最終
Data_Addr[0:N-1]	DLMB	O	データインターフェイス LMB アドレスバス、N = 32 - 64
Byte_Enable[0:3]	DLMB	O	データインターフェイス LMB バイトイネーブル
Data_Write[0:31]	DLMB	O	データインターフェイス LMB 書き込みデータバス
D_AS	DLMB	O	データインターフェイス LMB アドレスストローブ
Read_Strobe	DLMB	O	データインターフェイス LMB 読み出ilstrope
Write_Strobe	DLMB	O	データインターフェイス LMB 書き込みストローブ
Data_Read[0:31]	DLMB	I	データインターフェイス LMB 読み出ilデータバス
DReady	DLMB	I	データインターフェイス LMB 読み出し準備完了
DWait	DLMB	I	データインターフェイス LMB データ待機
DCE	DLMB	I	データインターフェイス LMB 訂正可能エラー
DUE	DLMB	I	データインターフェイス LMB 訂正不可能エラー
Instr_Addr[0:N-1]	ILMB	O	命令インターフェイス LMB アドレスバス、N = 32 ~ 64
I_AS	ILMB	O	命令インターフェイス LMB アドレスストローブ

表 3-1: MicroBlaze コア I/O のまとめ (続き)

信号	インター フェイス	I/O	説明
IFetch	ILMB	O	命令インターフェイス LMB 命令フェッチ
Instr[0:31]	ILMB	I	命令インターフェイス LMB 読み出しデータバス
IReady	ILMB	I	命令インターフェイス LMB データ準備完了
IWait	ILMB	I	命令インターフェイス LMB データ待機
ICE	ILMB	I	命令インターフェイス LMB 訂正可能エラー
IUE	ILMB	I	命令インターフェイス LMB 訂正不可能エラー
Mn_AXIS_TLAST	M0_AXIS.. M15_AXIS	O	マスターインターフェイス出力 AXI4 チャネル書き込み最終
Mn_AXIS_TDATA	M0_AXIS.. M15_AXIS	O	マスターインターフェイス出力 AXI4 チャネル書き込みデータ
Mn_AXIS_TVALID	M0_AXIS.. M15_AXIS	O	マスターインターフェイス出力 AXI4 チャネル書き込み有効
Mn_AXIS_TREADY	M0_AXIS.. M15_AXIS	I	マスターインターフェイス入力 AXI4 チャネル書き込み準備完了
Sn_AXIS_TLAST	S0_AXIS.. S15_AXIS	I	スレーブインターフェイス入力 AXI4 チャネル書き込み最終
Sn_AXIS_TDATA	S0_AXIS.. S15_AXIS	I	スレーブインターフェイス入力 AXI4 チャネル書き込みデータ
Sn_AXIS_TVALID	S0_AXIS.. S15_AXIS	I	スレーブインターフェイス入力 AXI4 チャネル書き込み有効
Sn_AXIS_TREADY	S0_AXIS.. S15_AXIS	O	スレーブインターフェイス出力 AXI4 チャネル書き込み準備完了
Interrupt	コア	I	割り込み。C_ASYNC_INTERRUPT がセットされている場合は、Clk に同期します。
Interrupt_Address ¹	コア	I	割り込みベクター アドレス
Interrupt_Ack ¹	コア	O	割り込み肯定応答
Reset	コア	I	コアリセット、アクティブ High。Clk の 1 クロック サイクル以上保持する必要があります。
Reset_Mode[0:1]	コア	I	リセットモード。リセットがアクティブのときにサンプリングされます。詳細は、表 3-3 を参照。
Clk	コア	I	Clock ²
Ext_BRK	コア	I	MDM からのブレーク信号
Ext_NM_BRK	コア	I	MDM からのマスク不可能なブレーク信号
MB_Halted	コア	O	デバッグインターフェイスを使用、Dbg_Stop をセット、または Reset_Mode[0:1] を 10 に設定することにより、パイプラインが停止したことを示します。

表 3-1: MicroBlaze コア I/O のまとめ (続き)

信号	インター フェイス	I/O	説明
Dbg_Stop	コア	I	できるだけ早く無条件でパイプラインを停止します。立ち上がりエッジで検出されるパルスで、Clk の 1 クロック サイクル以上保持する必要があります。信号は C_DEBUG_ENABLED が 0 より大きい場合にのみ有効です。
Dbg_Intr	コア	O	デバッグ割り込み出力。パフォーマンス監視カウンターがオーバーフローするとセットされます。C_DEBUG_ENABLED が 2 (拡張) のときに使用可能です。
MB_Error	コア	O	C_FAULT_TOLERANT が 1 のときに、ミスした例外が原因でパイプラインが停止したことを示します。
Sleep	コア	O	SLEEP 命令の実行後または Reset_Mode[0:1] を 10 に設定したことにより MicroBlaze がスリープ モードになり、外部アクセスがすべて完了して、パイプラインが停止したことを示します。
Hibernate	コア	O	HIBERNATE 命令の実行後、MicroBlaze がスリープ モードになり、外部アクセスがすべて完了して、パイプラインが停止したことを示します。
Suspend	コア	O	SUSPEND 命令の実行後、MicroBlaze がスリープ モードになり、外部アクセスがすべて完了して、パイプラインが停止したことを示します。
Wakeup[0:1]	コア	I	いずれかのビットまたは両方のビットを 1 に設定すると、MicroBlaze がスリープ モードから復帰します。MicroBlaze がスリープ モードでないときは無視されます。C_ASYNC_WAKEUP[0:1] の設定によって、信号は個々に Clk に同期します。
Dbg_Wakeup	コア	O	デバッグ アクセスができるよう、外部ロジックで Wakeup 信号を使用して、MicroBlaze をスリープ モードから復帰させるデバッグ要求。Dbg_Update に同期します。
Pause	コア	I	この信号をセットすると、処理中のバス アクセスがすべて完了した後、MicroBlaze パイプラインが停止し、Pause_Ack 信号がセットされます。この信号をクリアすると、停止していた MicroBlaze は平常の実行を再開します。
Pause_Ack	コア	O	Pause 入力信号がセットされ、MicroBlaze が一時停止モードになったことを示します。
Dbg_Continue	コア	O	デバッグ アクセスができるように、外部ロジックで Pause 信号をクリアにするデバッグ要求。
Non_Secure[0:3]	コア	I	AXI アクセスが非セキュアかセキュアであるかを指定します。デフォルト値は 2 進数 0000 で、すべてのインターフェイスをセキュアに設定します。 ビット 0: M_AXI_DP ビット 1: M_AXI_IP ビット 2: M_AXI_DC ビット 3: M_AXI_IC
Lockstep_...	コア	IO	信頼性の高いアプリケーション用のロックステップ信号。詳細は、 表 3-13 を参照してください。
Dbg_...	コア	IO	MDM からのデバッグ信号。詳細は、 表 3-15 を参照してください。

表 3-1: MicroBlaze コア I/O のまとめ (続き)

信号	インター フェイス	I/O	説明
Trace_...	コア	O	リアルタイムハードウェア解析のトレース信号。詳細は、 表 3-16 を参照してください。

1. C_USE_INTERRUPT = 2 の場合にのみ低レイテンシ割り込みサポート用に使用されます。
2. MicroBlaze は Clk クロック信号が供給される同期デザインです (Dbg_Clk クロック信号が供給されるシリアルハードウェアデバッグロジックを除く)。シリアルハードウェアデバッグロジックが使用されていない場合は、Clk の最小周波数制限はありません。シリアルハードウェアデバッグロジックが使用されている場合は、2つのクロック領域間で転送される信号があります。この場合、Clk の周波数は Dbg_Clk のものよりも高い必要があります。
3. C_ENABLE_DISCRETE_PORTS = 1 の場合にのみ使用されます。

表 3-2: リセットモード入力の効果

Reset_Mode[0:1]	説明
00	C_BASE_VECTORS の定義に従って、MicroBlaze がリセットベクターで実行を開始します。これが標準デフォルト動作です。
01	SLEEP 命令が実行された場合と同様に、MicroBlaze がバスアクセスを実行せずにすぐにスリープモードに入ります。SLEEP 出力は 1 にセットされます。Wakeup[0:1] 信号のいずれかがセットされると、C_BASE_VECTORS の定義に従って、MicroBlaze がリセットベクターで実行を開始します。 この機能はマルチプロセッサコンフィギュレーションで有益で、セカンダリプロセッサを LMB メモリなしで設定できます。
10	C_DEBUG_ENABLED が 0 の場合、動作は Reset_Mode[0:1] = 00 のときと同じになります。C_DEBUG_ENABLED が 0 より大きい場合、MicroBlaze はバスアクセスを実行せずにすぐにデバッグを停止し、MB_Halted 出力が 1 にセットされます。デバッグインターフェイスを介して実行を継続する場合は、C_BASE_VECTORS の定義に従って、MicroBlaze がリセットベクターで実行を開始します。
11	予約

一般的に、MicroBlaze 信号は Clk 入力信号に同期します。ただし、次の表で説明されているように、パラメーターで設定されている例外がいくつかあります。

表 3-3: パラメーターで設定されている非同期信号

信号	パラメーター	デフォルト	説明
Interrupt	C_ASYNC_INTERRUPT	ツール制御	接続されている信号から設定されるパラメーター
Reset	C_NUM_SYNC_FF_CLK	2	同期リセットの場合は、パラメーターを手動で 0 に設定可能
Wakeup[0:1]	C_ASYNC_WAKEUP C_NUM_SYNC_FF_CLK	ツール制御 2	接続されている信号から設定される ツールを上書きするには、パラメーターを手動で 0 に設定可能
Dbg_Wakeup	C_DEBUG_INTERFACE	0 (シリアル)	0: Dbg_Update でクロック供給 1: DEBUG_ACLK でクロック供給、Clk に同期

スリープおよび一時停止機能

MicroBlaze 実行の停止を制御するには、次の 2 つの方法があります。

- ・ ソフトウェア制御 (MBAR 命令を実行してスリープ モードに入る)。
- ・ ハードウェア制御 (入力信号 `Pause` をセットしてパイプラインを一時停止)。

ソフトウェア制御

スリープ モードに入るため MBAR 命令を実行し、MicroBlaze がすべての外部アクセスを完了させると、パイプラインが停止し、`Sleep`、`Hibernate`、または `Suspend` のいずれかの出力信号がセットされます。

つまり、外部ハードウェアに対し、クロック停止、プロセッサや IP コアのリセットなどを実行しても安全であることを示しています。どの出力信号がセットされるかによって、実行される操作は異なります。スリープ モードになっている MicroBlaze を起動させるには、`Wakeup` 入力信号の 1 つ (または両方) を 1 に設定する必要があります。この場合、MicroBlaze は MBAR 命令の後に実行を続けます。

MicroBlaze からの `Dbg_Wakeup` 出力信号は、デバッガーによりウェークアップが要求されたことを示します。外部ハードウェアは、クロックの開始など必要なハードウェア操作をすべて実行した後に、この信号を処理し、プロセッサをウェークアップさせます。デバッグ ウェークアップが使用されている場合は、ソフトウェアはウェークアップの理由の 1 つとしてこれを認識し、ほかに何も操作が必要なればスリープ状態に戻る必要があります。

プロセッサをウェークアップさせるのに追加操作を必要としない非常に単純なケースであれば、`Wakeup` 入力の 1 つを MicroBlaze の `Interrupt` 入力と同じ信号に接続し、もう一方の `Wakeup` 入力を MicroBlaze の `Dbg_Wakeup` 出力に接続します。これで、割り込みが起きたとき、またはデバッガーから要求があったときに MicroBlaze をウェークアップさせることができます。

ソフトウェアリセット機能をインプリメントするには、たとえば、プロセッサをリセットするのか、システム全体をリセットするかによって、`Suspend` 出力信号を適切なリセット入力に接続します。

次の表は、MBAR スリープ モード命令についてまとめたものです。

表 3-4: MBAR スリープ モード命令

命令	アセンブラー疑似命令	出力信号
<code>mbar 16</code>	<code>sleep</code>	<code>Sleep</code>
<code>mbar 8</code>	<code>hibernate</code>	<code>Hibernate</code>
<code>mbar 24</code>	<code>suspend</code>	<code>Suspend</code>

図 3-2 のブロック図に、スリープ機能を使用してクロック制御をインプリメントする方法を示します。この例では、スリープが実行されるとクロックが停止し、割り込みまたはデバッグ コマンドによりクロックが有効になってプロセッサがウェークアップします。

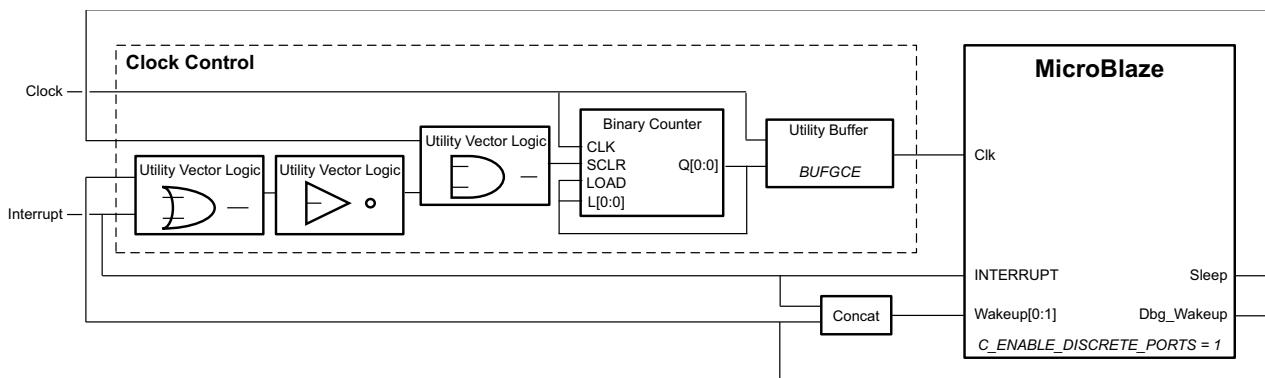


図 3-2: スリープのクロック制御のブロック図

IP コアでクロック制御をインプリメントする代わりに、RTL モジュールを使用できます。次に、図 3-2 のブロック図のクロック制御に対応する VHDL インプリメンテーションの例を示します。RTL モジュールの詳細は、『Vivado Design Suite ユーザー ガイド: IP インテグレーターを使用した IP サブシステムの設計』(UG994) [参照 12] を参照してください。

```

library IEEE;
use IEEE.STD_LOGIC_1164.all;

library UNISIM;
use UNISIM.VComponents.all;

entity clock_control is
  port (
    clkin      : in  std_logic;
    reset      : in  std_logic;
    sleep      : in  std_logic;
    interrupt  : in  std_logic;
    dbg_wakeup : in  std_logic;
    clkout     : out std_logic
  );
end clock_control;

architecture Behavioral of clock_control is
  attribute X_INTERFACE_INFO : string;
  attribute X_INTERFACE_INFO of clkin : signal is "xilinx.com:signal:clock:1.0 clk CLK";
  attribute X_INTERFACE_INFO of reset : signal is "xilinx.com:signal:reset:1.0 reset RST";
  attribute X_INTERFACE_INFO of interrupt : signal
    is "xilinx.com:signal:interrupt:1.0 interrupt INTERRUPT";
  attribute X_INTERFACE_INFO of clkout : signal is "xilinx.com:signal:clock:1.0 clk_out CLK";

  attribute X_INTERFACE_PARAMETER : string;
  attribute X_INTERFACE_PARAMETER of reset      : signal is "POLARITY ACTIVE_HIGH";
  attribute X_INTERFACE_PARAMETER of interrupt : signal is "SENSITIVITY LEVEL_HIGH";
  attribute X_INTERFACE_PARAMETER of clkout     : signal is "FREQ_HZ 100000000";

  signal clk_enable : std_logic := '1';
begin

  clock_enable_dff : process (clkin) is
  begin
    if clkin'event and clkin = '1' then
      if reset = '1' then
        clk_enable <= '1';
      elsif sleep = '1' and interrupt = '0' and dbg_wakeup = '0' then
        clk_enable <= '0';
      else
        clk_enable <= '1';
      end if;
    end if;
  end process;
end;

```

```
clk_enable <= '0';
elsif clk_enable = '0' then
    clk_enable <= '1';
end if;
end if;
end process clock_enable_dff;

clock_enable : component BUFGCE
port map (
    O  => clkout,
    CE => clk_enable,
    I   => clkin
);

end Behavioral;
```

ハードウェア制御

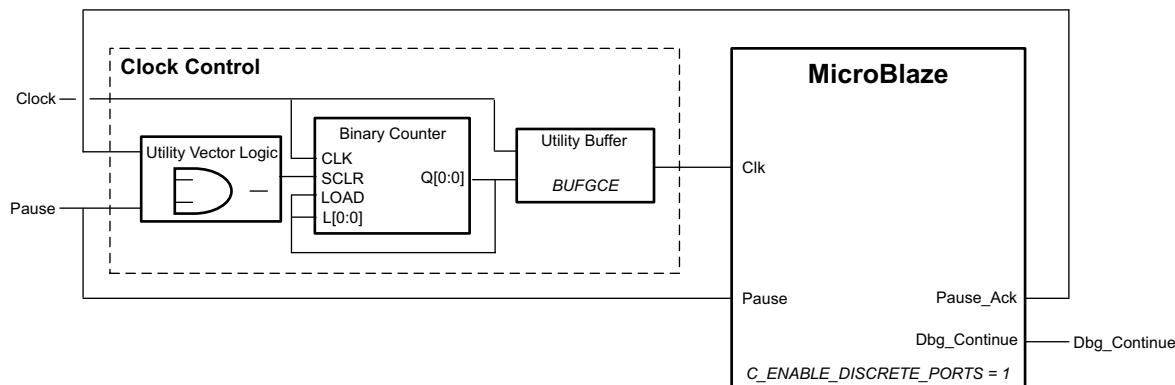
Pause 入力信号が 1 に設定されていて、MicroBlaze が外部アクセスをすべて完了すると、パイプラインが停止し、Pause_Ack 出力信号がセットされます。つまり、外部ハードウェアに対し、クロック停止、プロセッサや IP コアのリセットなどを実行しても安全であることを示しています。一時停止から復活するには、入力信号 Pause をゼロにクリアする必要があります。この場合、MicroBlaze は停止位置から命令実行を続けます。

MicroBlaze からの Dbg_Continue 出力信号は、プロセッサの実行を一時停止から再開するようデバッガーから要求があったことを示します。外部ハードウェアは、クロックの開始など必要なハードウェア操作をすべて実行した後に、この信号を処理し、Pause をクリアします。

外部ハードウェアが Pause をセットまたはクリアしたら、一時停止の肯定応答が誤って検出されることによる問題を回避するため、Pause が再び変わる前に Pause_Ack がセットまたはクリアされるのを待機することをお勧めします。

ハードウェア制御で使用される信号はすべて (Pause、Pause_Ack、および Dbg_Continue)、MicroBlaze のクロックに同期しています。

図 3-3 のブロック図に、一時停止機能を使用してプロセッサを停止し、クロック制御をインプリメントする方法を示します。この例では、Pause はプロセッサの実行を停止してクロックを停止する外部ハードウェア信号です。Pause が 0 にクリアされると、クロックが有効になり、実行が再開します。この例では、デバッグを可能にするため外部ロジックで Dbg_Continue が監視され、Pause がクリアにされると想定されます。



X20276-020818

図 3-3: 一時停止のクロック制御のブロック図

AXI4 および ACE インターフェイスについて

メモリマップドのインターフェイス

ペリフェラルインターフェイス

MicroBlaze の AXI4 メモリマップドペリフェラルインターフェイスは、32ビットのマスターとしてインプリメントされます。これらのインターフェイスには、一度に保持できるのは1つの未処理トランザクションのみで、すべてのトランザクションが順に実行されます。

- 命令ペリフェラルインターフェイス (`M_AXI_IP`) は、1ワードの読み出しアクセスのみを実行し、常に AXI4-Lite サブセットを使用するように設定されています。
- データペリフェラルインターフェイス (`M_AXI_DP`) は、1ワードアクセスを実行し、デフォルトで AXI4-Lite サブセットを使用するように設定されていますが、LWX および SWX 命令の排他的なアクセスをイネーブルにするときは AXI4 を使用するように設定されます。ハーフワードおよびバイトの書き込みは、適切なバイトストローブをセットし実行されます。

命令ペリフェラルインターフェイス (`M_AXI_IP`) のアドレス幅は、MMU 物理アドレス拡張 (PAE) がイネーブルの場合 32ビットから 64ビットまでで、`C_ADDR_SIZE` で設定される値によって決まります。

データペリフェラルインターフェイス (`M_AXI_DP`) のアドレス幅は 32ビットから 64ビットまでで、`C_ADDR_SIZE` で設定される値によって決まります。

キャッシュ インターフェイス

AXI4 メモリ マップド キャッシュ インターフェイスは、キャッシュ ラインの長さおよびデータ幅のパラメーターの設定によって、32、128、256、または 512 ビットのマスターとしてインプリメントされますが、AXI コヒーレンシ 拡張(ACE) インターフェイスは 32 ビット マスターとしてインプリメントされます。

- 32 ビット マスターの場合、命令キャッシュ インターフェイス (`M_AXI_IC` または `M_ACE_IC`) は、キャッシュ ラインの長さによって、4、8、または 16 ワードのバースト読み出しアクセスを実行します。128、256、または 512 ビット マスターの場合は、シングル読み出しアクセスのみが実行されます。

32 ビット マスターの場合、このインターフェイスで複数の未処理トランザクションを保持でき、2つまでのトランザクション、またはストリーム キャッシュがイネーブルの場合は5つまでのトランザクションを発行できます。ストリーム キャッシュは事前に2つのキャッシュ ラインを要求でき、場合によっては5つの未処理トランザクションを保持できます。この場合、未処理読み出しの数は、2のべき数にする必要があるので、8に設定されます。128、256、または 512 ビット マスターの場合は、インターフェイスに未処理トランザクションを1つしか保持できません。

アクセスするメモリ ロケーションの数は、`C_ICACHE_ALWAYS_USED` というパラメーターの設定で決まります。このパラメーターが1の場合は、キャッシュ メモリ範囲は常に、AXI4 または ACE キャッシュ インターフェイスを使用してアクセスされます。0の場合は、キャッシュがソフトウェアでディスエーブルになっていると(すなわち `MSR[ICE]=0`)、キャッシュ メモリ範囲は AXI4 ペリフェラル インターフェイスを介してアクセスされます。

- 32 ビット マスターの場合、データ キャッシュ インターフェイス (`M_AXI_DC` または `M_ACE_DC`) は、1 ワード アクセスを実行するか、またはキャッシュ ラインの長さによりますが、4、8、16 ワードのバースト読み出しアクセスを実行します。AXI4 でライトバック キャッシュが使用される場合は、バースト書き込みアクセスのみが実行されます。128、256、または 512 ビット の AXI4 マスターの場合は、シングル アクセスのみになります。

このインターフェイスは複数の未処理トランザクションを保持でき、読み出しの場合は2つまでのトランザクション、書き込みの場合は32個までのトランザクションを発行できます。プロセッサは順序付けられたメモリ モデルを保持する必要があり、AXI4 または ACE には順序付けされていない個別の読み出し/書き込みチャネルがあるため、MicroBlaze は読み出しの前にすべての未処理書き込みを完了させます。32個までの未処理書き込みトランザクションを使用すると、パイプラインをストールさせずに複数の書き込みを実行できるので、パフォーマンスが向上します。

ワード、ハーフワード、およびバイトの書き込みは、適切なバイト ストローブをセットして実行されます。

LWX および SWX 命令に対しては、排他的アクセスをイネーブルにできます。

アクセスするメモリ ロケーションの数は、`C_DCACHE_ALWAYS_USED` というパラメーターの設定で決まります。このパラメーターが1の場合は、キャッシュ メモリ範囲は常に、AXI4 または ACE キャッシュ インターフェイスを使用してアクセスされます。0の場合は、キャッシュがソフトウェアでディスエーブルになっていると(すなわち `MSR[DCE]=0`)、キャッシュ メモリ範囲は AXI4 ペリフェラル インターフェイスを介してアクセスされます。

インターフェイス パラメーターおよび信号

次の表では、パラメーターがツールで割り当てられる場合の MicroBlaze のパラメーター設定と AXI4 インターフェイスの動作の関係についてまとめられています。

表 3-5: AXI メモリ マップド インターフェイス パラメーター

インターフェイス	パラメーター	説明
M_AXI_DP	C_M_AXI_DP_PROTOCOL	AXI4-Lite: デフォルト。 AXI4: C_M_AXI_DP_EXCLUSIVE_ACCESS が 1 のときに排他的アクセスを可能にするために使用。
M_AXI_IC M_ACE_IC	C_M_AXI_IC_DATA_WIDTH	32: デフォルト。1ワード アクセス、および C_ICACHE_LINE_LEN のワード バーストでバースト アクセス (AXI4 および ACE で使用)。 128: C_ICACHE_DATA_WIDTH が 1 に、C_ICACHE_LINE_LEN が 4 に設定されているときに AXI4 で使用。シングル アクセスのみが発生。 256: C_ICACHE_DATA_WIDTH が 1 に、C_ICACHE_LINE_LEN が 8 に設定されているときに AXI4 で使用。シングル アクセスのみが発生。 512: C_ICACHE_DATA_WIDTH が 2 に設定されているとき、または、AXI4 でこれが 1 で、C_ICACHE_LINE_LEN が 16 に設定されているときに使用。シングル アクセスのみが発生。
M_AXI_DC M_ACE_DC	C_M_AXI_DC_DATA_WIDTH	32: デフォルト。1ワード アクセス、および C_DCACHE_LINE_LEN のワード バーストでバースト アクセス (AXI4 および ACE で使用)。 C_DCACHE_USE_WRITEBACK が 1 のとき AXI4 でのみ書き込みバーストを使用。 128: C_DCACHE_DATA_WIDTH が 1 に、C_DCACHE_LINE_LEN が 4 に設定されているときに AXI4 で使用。シングル アクセスのみが発生。 256: C_DCACHE_DATA_WIDTH が 1 に、C_DCACHE_LINE_LEN が 8 に設定されているときに AXI4 で使用。シングル アクセスのみが発生。 512: C_DCACHE_DATA_WIDTH が 2 に設定されているとき、または、AXI4 でこれが 1 で、C_DCACHE_LINE_LEN が 16 に設定されているときに使用。シングル アクセスのみが発生。
M_AXI_IC M_ACE_IC	NUM_READ_OUTSTANDING	1: 128、256、512 ビットのマスターのデフォルト、未処理読み出しを 1 つ保持。 2: 32 ビット マスターのデフォルト、未処理読み出しを同時に 2 つ保持。 8: C_ICACHE_STREAMS が 1 のときに 32 ビット マスターで使用、未処理読み出しを同時に 8 つ保持。 値は 1、2 または 8 に設定可能。

表 3-5: AXI メモリ マップド インターフェイス パラメーター(続き)

インターフェイス	パラメーター	説明
M_AXI_DC M_ACE_DC	NUM_READ_OUTSTANDING	1: 128、256、512 ビットのマスターのデフォルト、未処理読み出しを 1 つ保持。 2: 32 ビット マスターのデフォルト、未処理読み出しを同時に 2 つ保持。 値は 1 または 2 に設定可能。
M_AXI_DC M_ACE_DC	NUM_WRITE_OUTSTANDING	32: デフォルト、未処理読み出しを同時に 32 個保持。 値は 1、2、4、8、16、または 32 に設定可能。

次の表では、アクセス権限、メモリ タイプ、サービス品質(QoS)、共有ドメインを定義しています。

表 3-6: AXI インターフェイス信号定義

インターフェイス	信号	説明
M_AXI_IP	C_M_AXI_IP_ARPROT	アクセス権限: <ul style="list-style-type: none">入力信号が Non_Secure[1] = 0 の場合、権限なしのセキュア命令アクセス (100)入力信号が Non_Secure[1] = 1 の場合、権限なしの非セキュア命令アクセス (110)
M_AXI_DP	C_M_AXI_DP_ARCACHE	メモリ タイプ、AXI4 プロトコル: <ul style="list-style-type: none">標準ノンキヤッシャブル、バッファラブル (0011)
	C_M_AXI_DP_AWCACHE	
	C_M_AXI_DP_ARPROT	アクセス権限、AXI4 および AXI4-Lite プロトコル: <ul style="list-style-type: none">入力信号が Non_Secure[0] = 0 の場合、権限なしで、セキュアデータ アクセス (000)入力信号が Non_Secure[0] = 1 の場合、権限なしで、非セキュアデータ アクセス (010)
	C_M_AXI_DP_AWPROT	
M_AXI_IC	C_M_AXI_IC_ARCACHE	サービス品質 (QoS)、AXI4 プロトコル: <ul style="list-style-type: none">優先度 8 (1000)
	C_M_AXI_IC_AWCACHE	メモリ タイプ: <ul style="list-style-type: none">ライトバックの読み出しおよび書き込み割り当て (1111)
M_ACE_IC	C_M_AXI_IC_ARCACHE	メモリ タイプ、標準アクセス: <ul style="list-style-type: none">ライトバックの読み出しおよび書き込み割り当て (1111) メモリ タイプ、DVM アクセス: <ul style="list-style-type: none">標準ノンキヤッシャブル、ノンバッファラブル (0010)
	C_M_AXI_IC_ARDOMAIN	共有ドメイン: <ul style="list-style-type: none">内部共有可能 (01)

表 3-6: AXI インターフェイス信号定義(続き)

インターフェイス	信号	説明
M_AXI_IC M_ACE_IC	C_M_AXI_IC_ARPROT	アクセス権限: <ul style="list-style-type: none"> 入力信号が Non_Secure[3] = 0 の場合、権限なしのセキュア命令アクセス (100) 入力信号が Non_Secure[3] = 1 の場合、権限なしで、非セキュア命令アクセス (110)
	C_M_AXI_IC_ARQOS	サービス品質 (QoS): <ul style="list-style-type: none"> 優先度 7 (0111)
M_AXI_DC	C_M_AXI_DC_ARCACHE	メモリ タイプ、標準アクセス: <ul style="list-style-type: none"> ライトバックの読み出しおよび書き込み割り当て (1111) メモリ タイプ、排他的アクセス: <ul style="list-style-type: none"> 標準ノンキャッシュブル、ノンバッファラブル (0010)
M_ACE_DC	C_M_AXI_DC_ARCACHE	メモリ タイプ、標準および排他的アクセス: <ul style="list-style-type: none"> ライトバックの読み出しおよび書き込み割り当て (1111) メモリ タイプ、DVM アクセス: <ul style="list-style-type: none"> 標準ノンキャッシュブル、ノンバッファラブル (0010)
	C_M_AXI_DC_ARDOMAIN C_M_AXI_DC_AWDOMAIN	共有ドメイン: <ul style="list-style-type: none"> 内部共有可能 (01)
M_AXI_DC M_ACE_DC	C_M_AXI_DC_AWCACHE	メモリ タイプ、標準アクセス: <ul style="list-style-type: none"> ライトバックの読み出しおよび書き込み割り当て (1111) メモリ タイプ、排他的アクセス: <ul style="list-style-type: none"> 標準ノンキャッシュブル、ノンバッファラブル (0010)
	C_M_AXI_DC_ARPROT C_M_AXI_DC_AWPROT	アクセス権限: <ul style="list-style-type: none"> 入力信号が Non_Secure[2] = 0 の場合、権限なしで、セキュアデータ アクセス (000) 入力信号が Non_Secure[2] = 1 の場合、権限なしで、非セキュアデータ アクセス (010)
	C_M_AXI_DC_ARQOS	サービス品質 (QoS)、読み出しアクセス: <ul style="list-style-type: none"> 優先度 12 (1100)
	C_M_AXI_DC_AWQOS	サービス品質 (QoS)、書き込みアクセス: <ul style="list-style-type: none"> 優先度 8 (1000)

命令キャッシュインターフェイス (M_AXI_IC) のアドレス幅は、MMU 物理アドレス拡張 (PAE) がイネーブルの場合 32 ビットから 64 ビットまで、C_ADDR_SIZE で設定される値によって決まります。

データキャッシュインターフェイス (M_AXI_DC または M_ACE_DC) のアドレス幅は 32 ビットから 64 ビットまで、C_ADDR_SIZE で設定される値によって決まります。

詳細は、『AMBA AXI and ACE Protocol Specification』(Arm IHI 0022E) [参照 15] を参照してください。

ストリーム インターフェイス

MicroBlaze の AXI4-Stream インターフェイス (M0_AXIS、M15_AXIS、S0_AXIS、S15_AXIS) は 32 ビットのマスターおよびスレーブとしてインプリメントされます。詳細は、『AMBA 4 AXI4-Stream Protocol Specification, Version 1.0』(Arm IHI 0051A) [参照 14] を参照してください。

書き込み

ストリーム インターフェイスへの書き込みは、put または putd 命令の 1 つを使用して、MicroBlaze により実行されます。書き込み操作により、出力 AXI4 インターフェイスヘレジスタの内容が転送されます。ブロッキング モードの書き込み (put および cput 命令) の場合は、インターフェイスがビジーでなければ、転送は 1 クロック サイクルで完了します。インターフェイスがビジーなら、インターフェイスが利用可能になるまで、プロセッサがストールします。ノンブロッキング命令 (接頭辞 n) は、インターフェイスがビジー状態でも、常に 1 クロック サイクルで完了します。インターフェイスがビジーな場合は、書き込みは抑止され、MSR でキャリービットがセットされます。

制御命令 (接頭辞 c) は AXI4-Stream の TLAST 出力を 1 にセットします。これはパケットの境界を示すのに使用されます。

読み出し

ストリーム インターフェイスからの読み出しは、get または getd 命令の 1 つを使用して、MicroBlaze により実行されます。読み出し操作は、入力 AXI4 インターフェイスの内容を汎用レジスタに転送します。ブロッキング モードの読み出しの場合、データが使用可能であれば、転送通常 2 クロック サイクルで完了します。データが使用できない場合は、使用可能になるまで、この命令でプロセッサがストールします。ノンブロッキング モード (接頭辞 n の命令) では、データが使用可能であるかどうかにかかわらず、転送は 1 または 2 クロック サイクルで完了します。データが使用不可能な場合、データは転送されず、キャリービットが MSR でセットされます。

データの get 命令 (接頭辞 c なし) は、AXI4-Stream の TLAST 入力が 0 にクリアされるものとします。そうでない場合は、この命令は MSR[FSL] を 1 にセットします。制御の get 命令 (接頭辞 c あり) は、AXI4-Stream の TLAST 入力が 1 にセットされているものとします。そうでない場合は、この命令は MSR[FSL] を 1 にセットします。これはパケットの境界をチェックするのに使用されます。

ローカルメモリバス(LMB)インターフェイスについて

LMBは同期バスで、主にオンチップのブロックRAMにアクセスするために使用されます。このバスは、最小限の数の制御信号と単純なプロトコルを使用して、1クロックサイクルでローカルのブロックRAMにアクセスするようになります。LMB信号とその定義は次の表にリストされています。LMB信号はすべてアクティブHighです。

LMB信号インターフェイス

表3-7: LMBバス信号

信号	データインターフェイス	命令インターフェイス	タイプ	説明
Addr[0:N-1] ¹	Data_Addr[0:N-1] ¹	Instr_Addr[0:N-1] ²	O	アドレスバス
Byte_Enable[0:3]	Byte_Enable[0:3]	使用されない	O	バイトイネーブル
Data_Write[0:31]	Data_Write[0:31]	使用されない	O	書き込みデータバス
AS	D_AS	I_AS	O	アドレスストローブ
Read_Strobe	Read_Strobe	IFetch	O	読み出し処理中
Write_Strobe	Write_Strobe	使用されない	O	書き込み処理中
Data_Read[0:31]	Data_Read[0:31]	Instr[0:31]	I	読み出しデータバス
Ready	DReady	IReady	I	次の転送の準備完了
Wait ³	DWait	IWait	I	承認された転送が準備完了になるまで待機
CE ³	DCE	ICE	I	訂正可能エラー
UE ³	DUE	IUE	I	訂正不可能エラー
Clk	Clk	Clk	I	バスクロック

1. N = 32 ~ 64、C_ADDR_SIZE の設定により指定、MicroBlaze v9.6 で追加。

2. N = 32 ~ 64、PAE または 64 ビット MicroBlaze がイネーブルの場合に C_ADDR_SIZE の設定により指定、MicroBlaze v10.0 で追加。

3. MicroBlaze v8.00 で LMB に追加

Addr[0:N-1]

このアドレスバスはコアからの出力で、現在の転送でアクセスされているメモリアドレスを示します。AS が High の場合のみ有効です。マルチサイクルアクセスの場合(完了するまでに 2 クロック以上かかるもの)、Addr[0:N-1] は転送の最初のクロックサイクルでのみ有効です。

Byte_Enable[0:3]

このバイト イネーブル信号はコアからの出力で、データ バスのどのバイト レーンに有効なデータが含まれているのかを示します。Byte_Enable[0:3] は AS が High の場合のみ有効です。マルチサイクル アクセス(完了するまでに2クロック以上かかる)の場合、Byte_Enable[0:3] は転送の最初のクロック サイクルでのみ有効です。Byte_Enable[0:3] の有効値は次の表にリストされています。

表 3-8: Byte_Enable[0:3] の有効値

Byte_Enable[0:3]	使用バイト レーン			
	Data[0:7]	Data[8:15]	Data[16:23]	Data[24:31]
0001				●
0010			●	
0100		●		
1000	●			
0011			●	●
1100	●	●		
1111	●	●	●	●

Data_Write[0:31]

この書き込みデータ バスはコアからの出力で、メモリに書き込まれたデータが含まれています。AS が High の場合のみ有効です。Byte_Enable[0:3] で指定されたバイト レーンにのみ有効データが含まれています。

AS

このアドレス ストローブはコアからの出力で、転送の開始を示し、アドレス バスおよびバイト イネーブルを修飾します。これは転送の最初のクロック サイクルでのみ High で、その後は次の転送の開始まで Low のままになります。

Read_Strobe

この読み出しきストローブはコアからの出力で、読み出し転送が処理中であることを示します。この信号は、転送の最初のクロック サイクルで High になり、Ready 信号が High になったクロック サイクルが終わるまで High のままになる可能性があります。新しい読み出し転送が次のクロック サイクルですぐに開始した場合、Read_Strobe は High のままになります。

Write_Strobe

この書き込みストローブはコアからの出力で、書き込み転送が処理中であることを示します。この信号は、転送の最初のクロック サイクルで High になり、Ready 信号が High になったクロック サイクルが終わるまで High のままになる可能性があります。新しい書き込み転送が次のクロック サイクルですぐに開始した場合、Write_Strobe は High のままになります。

Data_Read[0:31]

この読み出しデータ バスはコアへの入力で、メモリから読み出されたデータを含んでいます。Data_Read は、Ready が High のときのクロックの立ち上がりエッジで有効になります。

Ready

この Ready 信号はコアへの入力で、現在の転送が完了したことと、次の転送が次のクロック サイクルで開始可能であることを示します。これはクロックの立ち上がりエッジでサンプルされます。この信号は、読み出しの場合 Data_Read[0:31] バスが有効であることを示し、書き込みの場合 Data_Write[0:31] バスがローカル メモリに書き込まれたことを示します。

Wait

Wait 信号はコアへの入力で、現在の転送は受け入れられたが、まだ完了はしていないことを示します。これはクロックの立ち上がりエッジでサンプルされます。

CE

CE 信号はコアへの入力で、現在の転送に訂正可能なエラーがあることを示します。Ready が High のときのクロックの立ち上がりエッジで有効になります。この信号は、読み出しの場合、Data_Read[0:31] バスでエラーが訂正されたことを示し、バイトおよびハーフワードの書き込みの場合は、ローカル メモリの対応データ ワードが、新しいデータを書き込む前に訂正されたことを示します。

UE

UE 信号はコアへの入力で、現在の転送に訂正不可能なエラーがあることを示します。Ready が High のときのクロックの立ち上がりエッジで有効になります。この信号は、読み出しの場合、Data_Read[0:31] バスの値が間違っていることを示し、バイトおよびハーフワードの書き込みの場合は、新しいデータを書き込む前の、ローカル メモリの対応データ ワードが間違っていることを示します。

Clk

LMB のすべての操作は MicroBlaze コア クロックに同期しています。

LMB トランザクション

次の図に、LMB バス操作の例を示します。

一般的な書き込み操作

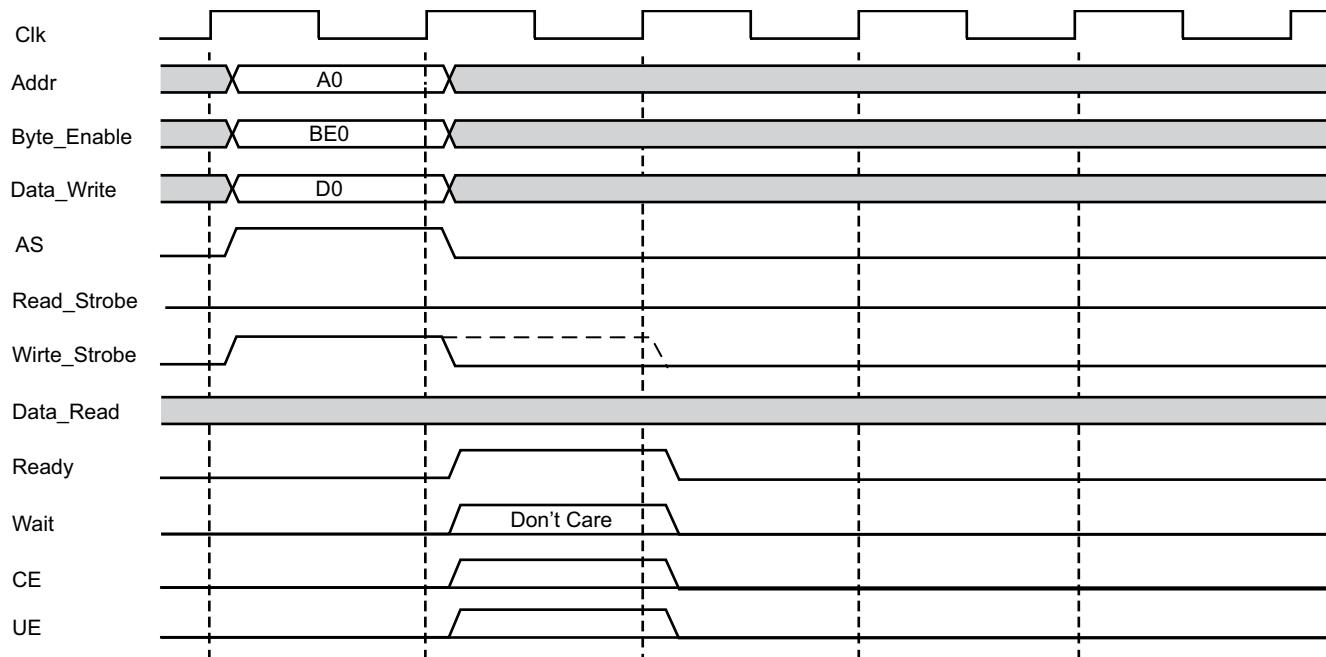


図 3-4: LMB の一般的な書き込み操作、待機ステート 0 個

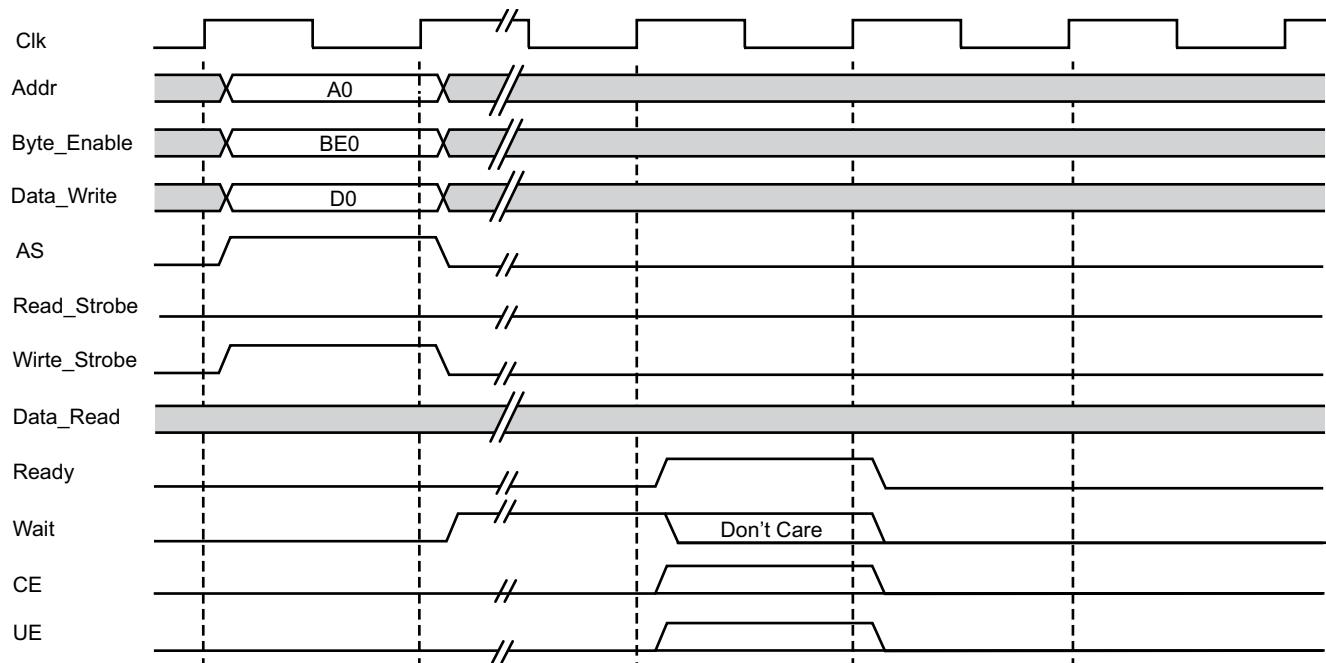


図 3-5: LMB の一般的な書き込み操作、待機ステート N 個

一般的な読み出し操作

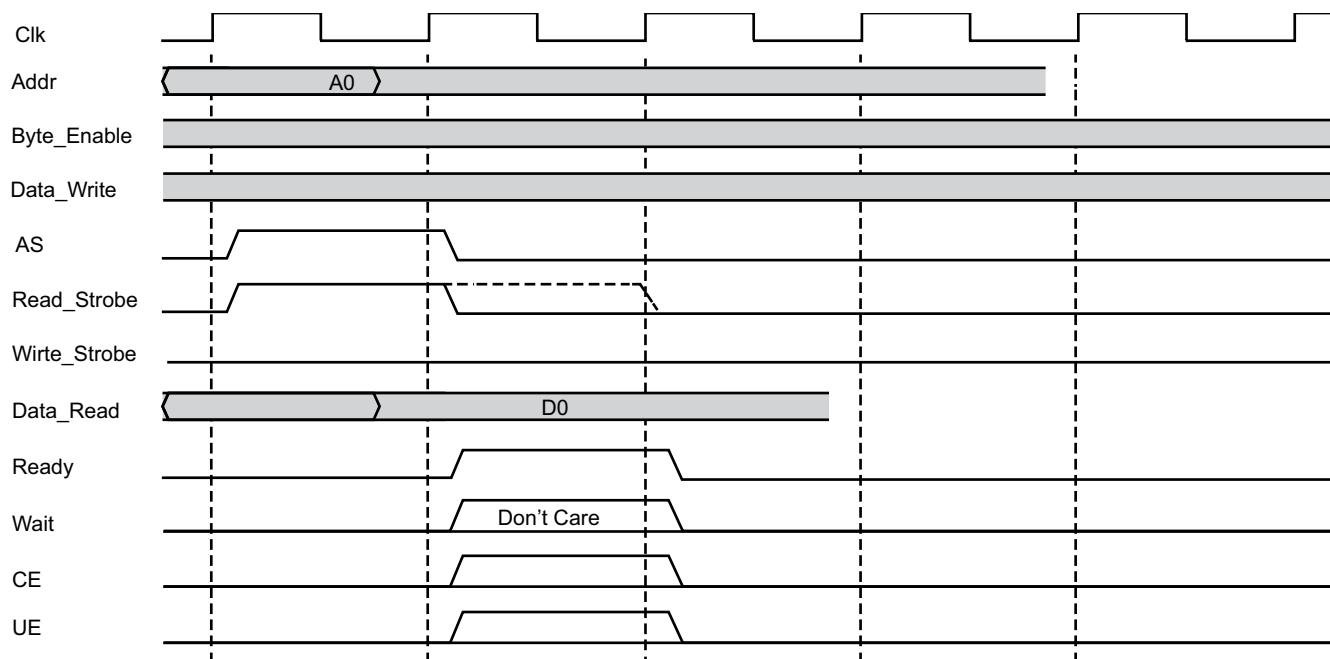


図 3-6: LMB の一般的な読み出し操作、待機ステート 0 個

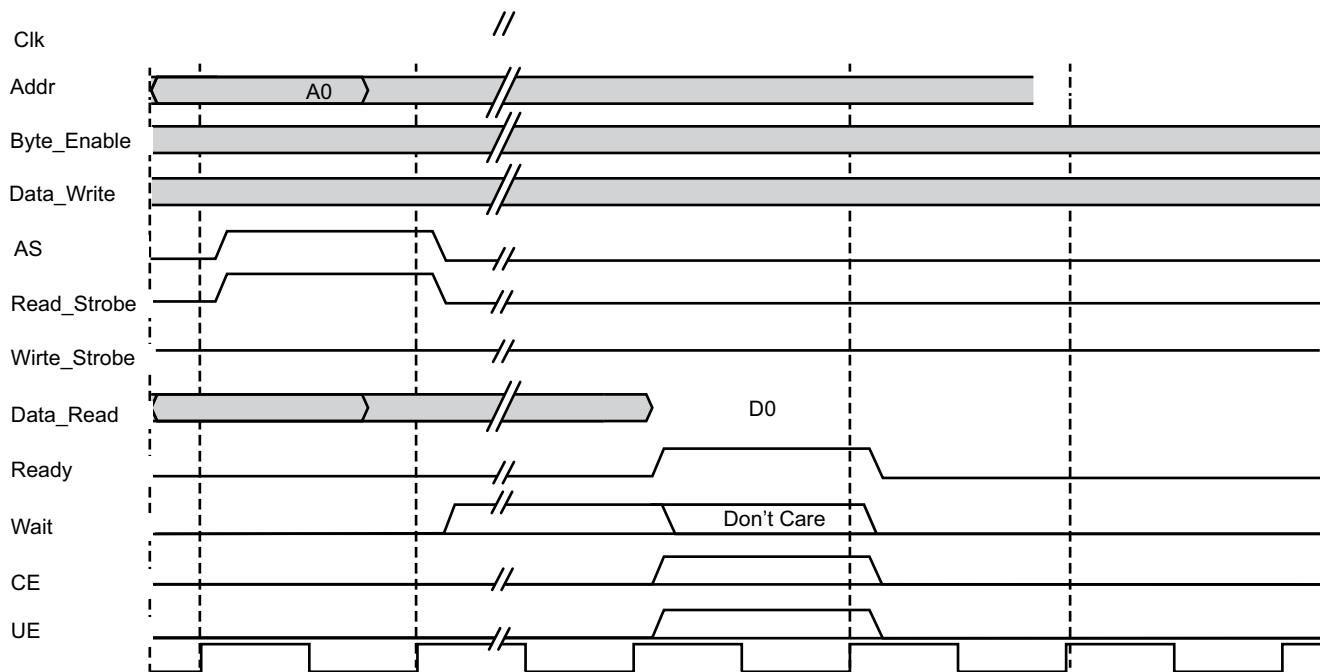


図 3-7: LMB の一般的な読み出し操作、待機ステート N 個

連続書き込み操作

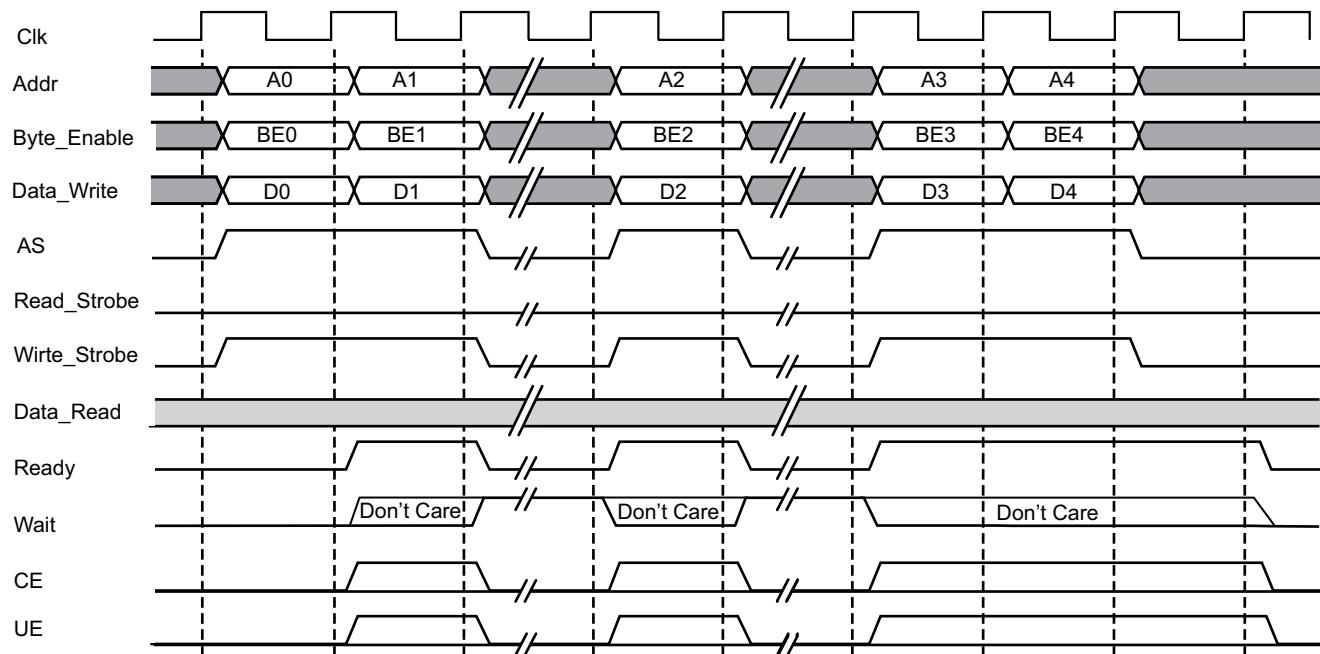


図 3-8: LMB の連続書き込み操作

連続読み出し操作

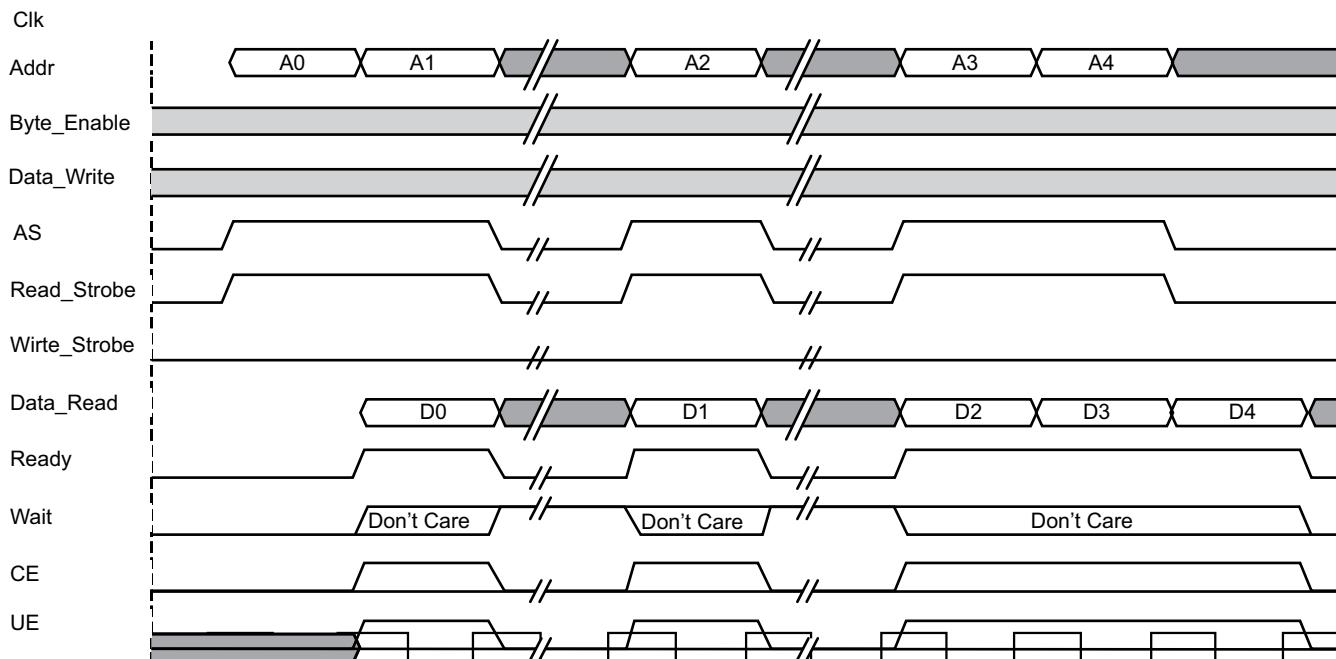


図 3-9: LMB の連続読み出し操作

連続混合書き込み/読み出し操作

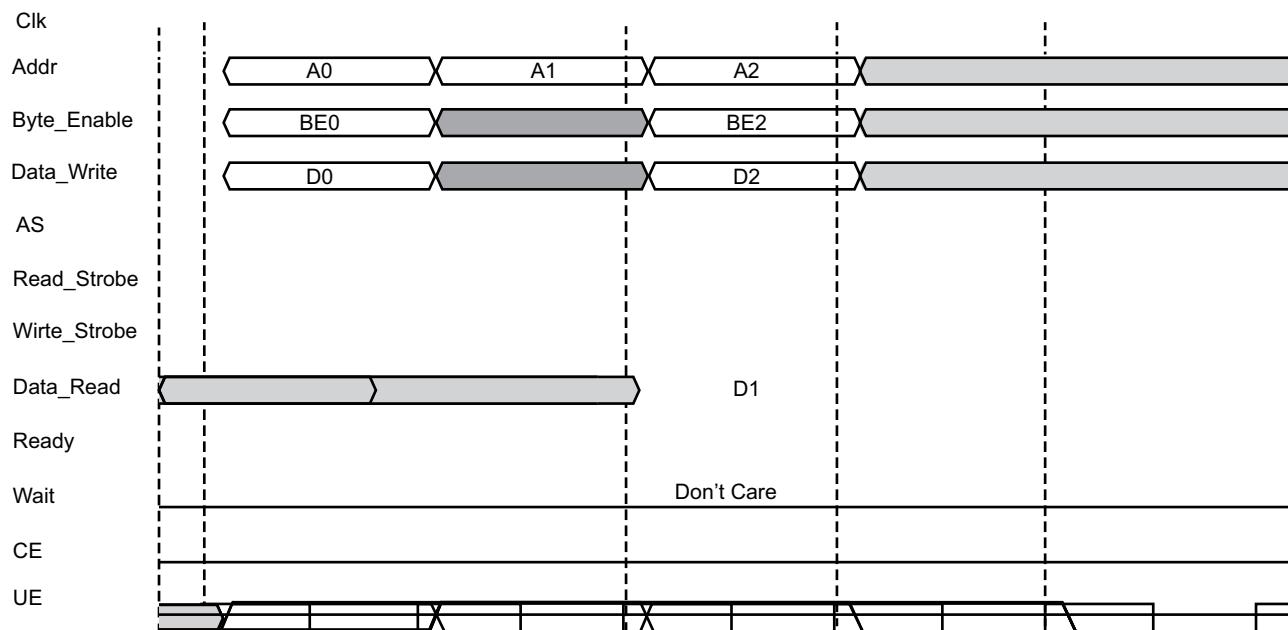


図 3-10: 連続混合書き込み/読み出し操作、待機ステート 0 個

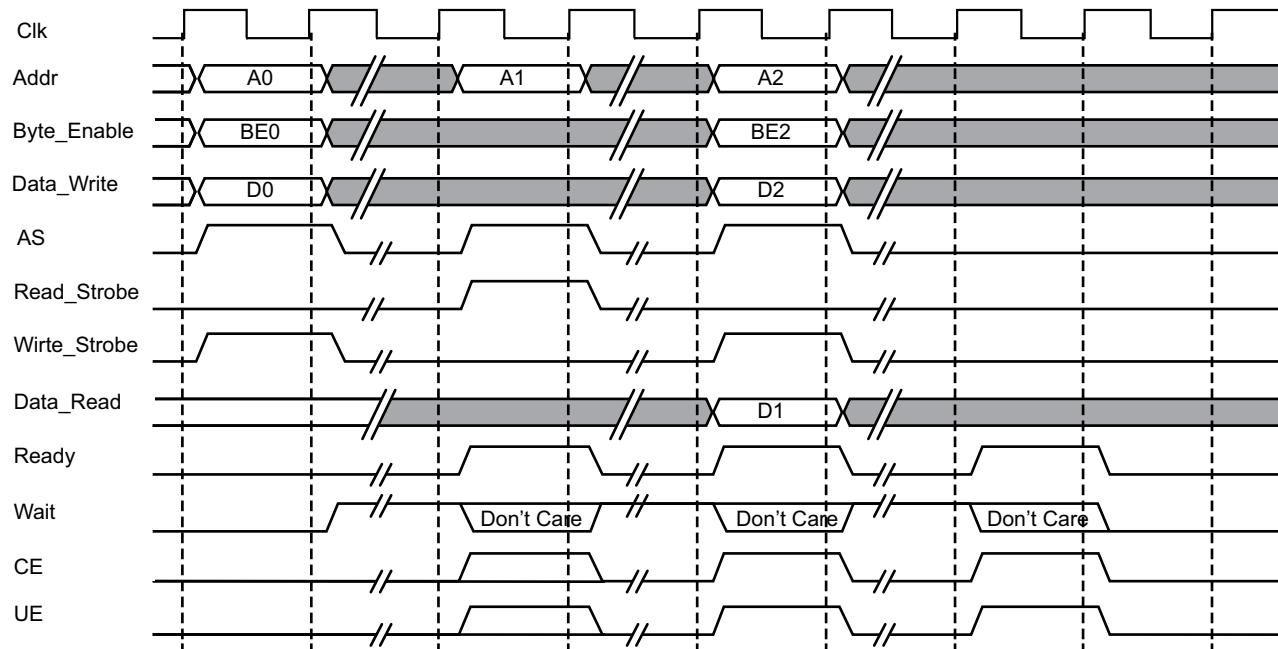


図 3-11: 連続混合書き込み/読み出し操作、待機ステート N 個

読み出しおよび書き込みデータステアリング

MicroBlaze のデータ側バス インターフェイスは、次の転送のサポートに必要な読み出しステアリングおよび書き込みステアリングを実行します。

- ワード デバイスへのバイト、ハーフワード、およびワード転送
- ハーフワード デバイスへのバイトおよびハーフワード転送
- バイト デバイスへのバイト転送

MicroBlaze は、アドレス指定されているデバイスよりもサイズが大きな転送をサポートしていません。こうしたタイプの転送には、MicroBlaze のバス インターフェイスではサポートされていないダイナミック バス サイズ調整や変換サイクルが必要になります。読み出しサイクルのデータステアリングは、[表 3-9](#) および[表 3-10](#) に、書き込みサイクルのデータステアリングは、[表 3-11](#) および[表 3-12](#) にまとめられています。

仮想モードまたは保護モードで MMU を使用している場合 (`C_USE_MMU > 1`)、または再順序付け命令がイネーブルになっている場合 (`C_USE_REORDER_INSTR = 1`) にのみ、ビッグ エンディアン フォーマットは使用可能です。

表 3-9: ビッグ エンディアン読み出しデータステアリング (レジスタ rD ヘロード)

Address [LSB-1:LSB]	Byte_Enable [0:3]	転送サイズ	レジスタ rD データ			
			rD[0:7]	rD[8:15]	rD[16:23]	rD[24:31]
11	0001	バイト				バイト 3
10	0010	バイト				バイト 2
01	0100	バイト				バイト 1
00	1000	バイト				バイト 0
10	0011	ハーフワード			バイト 2	バイト 3
00	1100	ハーフワード			バイト 0	バイト 1
00	1111	ワード	バイト 0	バイト 1	バイト 2	バイト 3

表 3-10: リトル エンディアン読み出しデータステアリング (レジスタ rD ヘロード)

Address [LSB-1:LSB]	Byte_Enable [0:3]	転送サイズ	レジスタ rD データ			
			rD[0:7]	rD[8:15]	rD[16:23]	rD[24:31]
11	1000	バイト				バイト 0
10	0100	バイト				バイト 1
01	0010	バイト				バイト 2
00	0001	バイト				バイト 3
10	1100	ハーフワード			バイト 0	バイト 1
00	0011	ハーフワード			バイト 2	バイト 3
00	1111	ワード	バイト 0	バイト 1	バイト 2	バイト 3

表 3-11: ビッグエンディアン書き込みデータステアリング(レジスタ rD からストア)

Address [LSB-1:LSB]	Byte_Enable [0:3]	転送サイズ	書き込みデータバス バイト			
			バイト 0	バイト 1	バイト 2	バイト 3
11	0001	バイト				rD[24:31]
10	0010	バイト			rD[24:31]	
01	0100	バイト		rD[24:31]		
00	1000	バイト	rD[24:31]			
10	0011	ハーフワード			rD[16:23]	rD[24:31]
00	1100	ハーフワード	rD[16:23]	rD[24:31]		
00	1111	ワード	rD[0:7]	rD[8:15]	rD[16:23]	rD[24:31]

表 3-12: リトルエンディアン書き込みデータステアリング(レジスタ rD からストア)

Address [LSB-1:LSB]	Byte_Enable [0:3]	転送サイズ	書き込みデータバス バイト			
			バイト 3	バイト 2	バイト 1	バイト 0
11	1000	バイト	rD[24:31]			
10	0100	バイト		rD[24:31]		
01	0010	バイト			rD[24:31]	
00	0001	バイト				rD[24:31]
10	1100	ハーフワード	rD[16:23]	rD[24:31]		
00	0011	ハーフワード			rD[16:23]	rD[24:31]
00	1111	ワード	rD[0:7]	rD[8:15]	rD[16:23]	rD[24:31]

注記: ほかのマスターには、MicroBlaze で認められているものよりも、バイトレーンに関して厳しい要件が課せられているものがある場合があります。スレーブデバイスは通常、バイトデバイスなら最上位バイトレーンに、ハーフワードデバイスなら最上位ハーフワードレーンに接続される「左揃え」になっています。MicroBlaze のステアリングロジックはこの接続方法を完全サポートしています。

ロックステップ インターフェイスについて

MicroBlaze のロックステップ インターフェイスは、1つのマスターおよび1つ以上のスレーブ MicroBlaze インスタンスを接続するよう設計されています。次の表は、MicroBlaze のロックステップ信号をリストしています。

表 3-13: MicroBlaze ロックステップ信号

信号名	説明	VHDL タイプ	方向
Lockstep_Master_Out	マスターからスレーブ MicroBlaze への信号で出力。スレーブでは接続されない。	std_logic	出力
Lockstep_Slave_In	マスターからスレーブ MicroBlaze への信号で入力。マスターでは接続されない。	std_logic	入力
Lockstep_Out	マスターとスレーブ両方からのすべての比較信号で出力。	std_logic	出力

次の表は、Lockstep_Out で提供される比較信号をリストしています。

表 3-14: MicroBlaze ロックステップ比較信号

信号名	バスインデックス範囲	VHDL タイプ
MB_Halted	0	std_logic
MB_Error	1	std_logic
IFetch	2	std_logic
I_AS	3	std_logic
Instr_Addr	4 ~ 67	std_logic_vector
Data_Addr	68 ~ 131	std_logic_vector
Data_Write	132 ~ 163	std_logic_vector
D_AS	196	std_logic
Read_Strobe	197	std_logic
Write_Strobe	198	std_logic
Byte_Enable	199 ~ 202	std_logic_vector
M_AXI_IP_AWID	207	std_logic
M_AXI_IP_AWADDR	208 ~ 271	std_logic_vector
M_AXI_IP_AWLEN	272 ~ 279	std_logic_vector
M_AXI_IP_AWSIZE	280 ~ 282	std_logic_vector
M_AXI_IP_AWBURST	283 ~ 284	std_logic_vector
M_AXI_IP_AWLOCK	285	std_logic
M_AXI_IP_AWCACHE	286 ~ 289	std_logic_vector
M_AXI_IP_AWPROT	290 ~ 292	std_logic_vector
M_AXI_IP_AWQOS	293 ~ 296	std_logic_vector
M_AXI_IP_AWVALID	297	std_logic
M_AXI_IP_WDATA	298 ~ 329	std_logic_vector
M_AXI_IP_WSTRB	362 ~ 365	std_logic_vector
M_AXI_IP_WLAST	370	std_logic
M_AXI_IP_WVALID	371	std_logic

表3-14: MicroBlazeロックステップ比較信号(続き)

信号名	バスインデックス範囲	VHDLタイプ
M_AXI_IP_BREADY	372	std_logic
M_AXI_IP_ARID	373	std_logic
M_AXI_IP_ARADDR	374～437	std_logic_vector
M_AXI_IP_ARLEN	438～445	std_logic_vector
M_AXI_IP_ARSIZE	446～448	std_logic_vector
M_AXI_IP_ARBURST	449～450	std_logic_vector
M_AXI_IP_ARLOCK	451	std_logic
M_AXI_IP_ARCACHE	452～455	std_logic_vector
M_AXI_IP_ARPROT	456～458	std_logic_vector
M_AXI_IP_ARQOS	459～462	std_logic_vector
M_AXI_IP_ARVALID	463	std_logic
M_AXI_IP_RREADY	464	std_logic
M_AXI_DP_AWID	465	std_logic
M_AXI_DP_AWADDR	466～529	std_logic_vector
M_AXI_DP_AWLEN	530～537	std_logic_vector
M_AXI_DP_AWSIZE	538～540	std_logic_vector
M_AXI_DP_AWBURST	541～542	std_logic_vector
M_AXI_DP_AWLOCK	543	std_logic
M_AXI_DP_AWCACHE	544～547	std_logic_vector
M_AXI_DP_AWPROT	548～550	std_logic_vector
M_AXI_DP_AWQOS	551～554	std_logic_vector
M_AXI_DP_AWVALID	555	std_logic
M_AXI_DP_WDATA	556～587	std_logic_vector
M_AXI_DP_WSTRB	620～623	std_logic_vector
M_AXI_DP_WLAST	628	std_logic
M_AXI_DP_WVALID	629	std_logic
M_AXI_DP_BREADY	630	std_logic
M_AXI_DP_ARID	631	std_logic
M_AXI_DP_ARADDR	632～695	std_logic_vector
M_AXI_DP_ARLEN	696～703	std_logic_vector
M_AXI_DP_ARSIZE	704～706	std_logic_vector
M_AXI_DP_ARBURST	707～708	std_logic_vector
M_AXI_DP_ARLOCK	709	std_logic
M_AXI_DP_ARCACHE	710～713	std_logic_vector
M_AXI_DP_ARPROT	714～716	std_logic_vector
M_AXI_DP_ARQOS	717～720	std_logic_vector
M_AXI_DP_ARVALID	721	std_logic
M_AXI_DP_RREADY	722	std_logic
Mn_AXIS_TLAST	723+n*35	std_logic
Mn_AXIS_TDATA	758+n*35～789+n*35	std_logic_vector

表3-14: MicroBlazeロックステップ比較信号(続き)

信号名	バスインデックス範囲	VHDLタイプ
Mn_AXIS_TVALID	$790 + n * 35$	std_logic
Sn_AXIS_TREADY	$791 + n * 35$	std_logic
M_AXI_IC_AWID	1283	std_logic
M_AXI_IC_AWADDR	1284 ~ 1347	std_logic_vector
M_AXI_IC_AWLEN	1348 ~ 1355	std_logic_vector
M_AXI_IC_AWSIZE	1356 ~ 1358	std_logic_vector
M_AXI_IC_AWBURST	1359 ~ 1360	std_logic_vector
M_AXI_IC_AWLOCK	1361	std_logic
M_AXI_IC_AWCACHE	1362 ~ 1365	std_logic_vector
M_AXI_IC_AWPROT	1366 ~ 1368	std_logic_vector
M_AXI_IC_AWQOS	1369 ~ 1372	std_logic_vector
M_AXI_IC_AWVALID	1373	std_logic
M_AXI_IC_AWUSER	1374 ~ 1378	std_logic_vector
M_AXI_IC_AWDOMAIN ¹	1379 ~ 1380	std_logic_vector
M_AXI_IC_AWSNOOP ¹	1381 ~ 1383	std_logic_vector
M_AXI_IC_AWBAR ¹	1384 ~ 1385	std_logic_vector
M_AXI_IC_WDATA	1386 ~ 1897	std_logic_vector
M_AXI_IC_WSTRB	1898 ~ 1961	std_logic_vector
M_AXI_IC_WLAST	1962	std_logic
M_AXI_IC_WVALID	1963	std_logic
M_AXI_IC_WUSER	1964	std_logic
M_AXI_IC_BREADY	1965	std_logic
M_AXI_IC_WACK	1966	std_logic
M_AXI_IC_ARID	1967	std_logic_vector
M_AXI_IC_ARADDR	1968 ~ 2031	std_logic_vector
M_AXI_IC_ARLEN	2032 ~ 2039	std_logic_vector
M_AXI_IC_ARSIZE	2040 ~ 2042	std_logic_vector
M_AXI_IC_ARBURST	2043 ~ 2044	std_logic_vector
M_AXI_IC_ARLOCK	2045	std_logic
M_AXI_IC_ARCACHE	2046 ~ 2049	std_logic_vector
M_AXI_IC_ARPROT	2050 ~ 2052	std_logic_vector
M_AXI_IC_ARQOS	2053 ~ 2056	std_logic_vector
M_AXI_IC_ARVALID	2057	std_logic
M_AXI_IC_ARUSER	2058 ~ 2062	std_logic_vector
M_AXI_IC_ARDOMAIN ¹	2063 ~ 2064	std_logic_vector
M_AXI_IC_ARNSNOOP ¹	2065 ~ 2068	std_logic_vector
M_AXI_IC_ARBAR ¹	2069 ~ 2070	std_logic_vector
M_AXI_IC_RREADY	2071	std_logic
M_AXI_IC_RACK ¹	2072	std_logic
M_AXI_IC_ACREADY ¹	2073	std_logic
M_AXI_IC_CRVALID ¹	2074	std_logic

表3-14: MicroBlazeロックステップ比較信号(続き)

信号名	バスインデックス範囲	VHDLタイプ
M_AXI_IC_CRRESP ¹	2075～2079	std_logic_vector
M_AXI_IC_CDVALID ¹	2080	std_logic
M_AXI_IC_CDLAST ¹	2081	std_logic
M_AXI_DC_AWID	2082	std_logic
M_AXI_DC_AWADDR	2083～2146	std_logic_vector
M_AXI_DC_AWLEN	2147～2154	std_logic_vector
M_AXI_DC_AWSIZE	2155～2157	std_logic_vector
M_AXI_DC_AWBURST	2158～2159	std_logic_vector
M_AXI_DC_AWLOCK	2160	std_logic
M_AXI_DC_AWCACHE	2161～2164	std_logic_vector
M_AXI_DC_AWPROT	2165～2167	std_logic_vector
M_AXI_DC_AWQOS	2168～2171	std_logic_vector
M_AXI_DC_AWVALID	2172	std_logic
M_AXI_DC_AWUSER	2172～2176	std_logic_vector
M_AXI_DC_AWDOMAIN ¹	2177～2178	std_logic_vector
M_AXI_DC_AWSNOOP ¹	2179～2182	std_logic_vector
M_AXI_DC_AWBAR ¹	2183～2184	std_logic_vector
M_AXI_DC_WDATA	2185～2696	std_logic_vector
M_AXI_DC_WSTRB	2697～2760	std_logic_vector
M_AXI_DC_WLAST	2761	std_logic
M_AXI_DC_WVALID	2762	std_logic
M_AXI_DC_WUSER	2863	std_logic
M_AXI_DC_BREADY	2764	std_logic
M_AXI_DC_WACK ¹	2765	std_logic
M_AXI_DC_ARID	2766	std_logic
M_AXI_DC_ARADDR	2767～2830	std_logic_vector
M_AXI_DC_ARLEN	2831～2838	std_logic_vector
M_AXI_DC_ARSIZE	2839～2841	std_logic_vector
M_AXI_DC_ARBURST	2842～2843	std_logic_vector
M_AXI_DC_ARLOCK	2844	std_logic
M_AXI_DC_ARCACHE	2845～2848	std_logic_vector
M_AXI_DC_ARPROT	2849～2851	std_logic_vector
M_AXI_DC_ARQOS	2852～2855	std_logic_vector
M_AXI_DC_ARVALID	2856	std_logic
M_AXI_DC_ARUSER	2857～2861	std_logic_vector
M_AXI_DC_ARDOMAIN ¹	2862～2863	std_logic_vector
M_AXI_DC_ARSNOOP ¹	2864～2867	std_logic_vector
M_AXI_DC_ARBAR ¹	2868～2869	std_logic_vector
M_AXI_DC_RREADY	2870	std_logic
M_AXI_DC_RACK ¹	2871	std_logic
M_AXI_DC_ACREADY ¹	2872	std_logic

表3-14: MicroBlazeロックステップ比較信号(続き)

信号名	バスインデックス範囲	VHDLタイプ
M_AXI_DC_CRVALID ¹	2873	std_logic
M_AXI_DC_CRRRESP ¹	2874～2878	std_logic_vector
M_AXI_DC_CDVALID ¹	2879	std_logic
M_AXI_DC_CDLAST ¹	2880	std_logic
Trace_Instruction	2881～2912	std_logic_vector
Trace_Valid_Instr	2913	std_logic
Trace_PC	2914～2945	std_logic_vector
Trace_Reg_Write	2978	std_logic
Trace_Reg_Addr	2979～2983	std_logic_vector
Trace_MSR_Reg	2984～2998	std_logic_vector
Trace_PID_Reg	2999～3006	std_logic_vector
Trace_New_Reg_Value	3007～3038	std_logic_vector
Trace_Exception_Taken	3071	std_logic
Trace_Exception_Kind	3072～3076	std_logic_vector
Trace_Jump_Taken	3077	std_logic
Trace_Delay_Slot	3078	std_logic
Trace_Data_Address	3079～3142	std_logic_vector
Trace_Data_Write_Value	3143～3174	std_logic_vector
Trace_Data_Byt_Enable	3207～3210	std_logic_vector
Trace_Data_Access	3215	std_logic
Trace_Data_Read	3216	std_logic
Trace_Data_Write	3217	std_logic
Trace_DCache_Req	3218	std_logic
Trace_DCache_Hit	3219	std_logic
Trace_DCache_Rdy	3220	std_logic
Trace_DCache_Read	3221	std_logic
Trace_ICache_Req	3222	std_logic
Trace_ICache_Hit	3223	std_logic
Trace_ICache_Rdy	3224	std_logic
Trace_OF_PipeRun	3225	std_logic
Trace_EX_PipeRun	3226	std_logic
Trace_MEM_PipeRun	3227	std_logic
Trace_MB_Halted	3228	std_logic
Trace_Jump_Hit	3229	std_logic
予約	3230～4095	

1. この信号は、C_INTERCONNECT = 3 (ACE) の場合のみ使用します。

デバッグインターフェイスについて

MicroBlaze のデバッグインターフェイスは、ザイリンクスの MDM (Microprocessor Debug Module) IP コアを使用して機能するように設計されています。MDM は、FPGA の JTAG ポートを介して XSDB (ザイリンクスシステムデバッガー) により制御されます。MDM は一度に複数の MicroBlaze プロセッサを制御できます。デバッグ信号は DEBUG バスにまとめられています。

デバッグインターフェイスは、JTAGシリアル信号 (`C_DEBUG_INTERFACE = 0`) または AXI4-Lite と互換性のあるパラレル信号 (`C_DEBUG_INTERFACE = 1`) を使用して、DEBUG バスにまとめることができます。MDM コンフィギュレーションもそれに従って設定できます。

MDM が使用されていない場合は、AXI4 バスにまとめられている AXI4-Lite パラレル信号 (`C_DEBUG_INTERFACE = 2`) のみを使用することも可能です。ただし、この設定はツールではサポートされていません。

表 3-15 は、MicroBlaze のデバッグ信号をリストしています。

表 3-15: MicroBlaze デバッグ信号

信号名	説明	VHDL タイプ	種類
<code>Dbg_Clk</code>	MDM からの JTAG クロック	<code>std_logic</code>	シリアル入力
<code>Dbg_TDI</code>	MDM からの JTAG TDI	<code>std_logic</code>	シリアル入力
<code>Dbg_TDO</code>	MDM への JTAG TDO	<code>std_logic</code>	シリアル出力
<code>Dbg_Reg_En</code>	MDM からのデバッグレジスタ タイネーブル	<code>std_logic_vector</code>	シリアル入力
<code>Dbg_Shift¹</code>	MDM からの JTAG BSCAN シフト信号	<code>std_logic</code>	シリアル入力
<code>Dbg_Capture</code>	MDM からの JTAG BSCAN キャプチャ信号	<code>std_logic</code>	シリアル入力
<code>Dbg_Update</code>	MDM からの JTAG BSCAN アップデート信号	<code>std_logic</code>	シリアル入力
<code>Debug_Rst¹</code>	MDM からのリセット信号、アクティブ High。 <code>C1k</code> の 1 クロック サイクル以上保持する必要があります。	<code>std_logic</code>	入力
<code>Dbg_Trig_In²</code>	MDM へのクロストリガーイベント入力	<code>std_logic_vector</code>	出力
<code>Dbg_Trig_Ack_In²</code>	MDM からのクロストリガーイベント入力肯定応答	<code>std_logic_vector</code>	入力
<code>Dbg_Trig_Out²</code>	MDM からのクロストリガーアクション出力	<code>std_logic_vector</code>	入力
<code>Dbg_Trig_Ack_Out²</code>	MDM へのクロストリガーアクション出力肯定応答	<code>std_logic_vector</code>	出力
<code>Dbg_Trace_Data³</code>	MDM への外部プログラムトレースデータ出力	<code>std_logic_vector</code>	出力
<code>Dbg_Trace_Valid³</code>	MDM への外部プログラムトレース有効	<code>std_logic</code>	出力
<code>Dbg_Trace_Ready³</code>	MDM からの外部プログラムトレース準備完了	<code>std_logic</code>	入力
<code>Dbg_Trace_Clk³</code>	MDM からの外部プログラムトレースクロック	<code>std_logic</code>	入力
<code>Dbg_ARADDR⁴</code>	MDM からの読み出しアドレス	<code>std_logic_vector</code>	パラレル入力
<code>Dbg_ARREADY⁴</code>	MDM への読み出しアドレスレディ	<code>std_logic</code>	パラレル出力
<code>Dbg_ARVALID⁴</code>	MDM からの読み出しアドレス有効	<code>std_logic</code>	パラレル入力
<code>Dbg_AWADDR⁴</code>	MDM からの書き込みアドレス	<code>std_logic_vector</code>	パラレル入力
<code>Dbg_AWREADY⁴</code>	MDM への書き込みアドレスレディ	<code>std_logic</code>	パラレル出力
<code>Dbg_AWVALID⁴</code>	MDM からの書き込みアドレス有効	<code>std_logic</code>	パラレル入力
<code>Dbg_BREADY⁴</code>	MDM への書き込み応答レディ	<code>std_logic</code>	パラレル出力

表 3-15: MicroBlaze デバッグ信号(続き)

信号名	説明	VHDL タイプ	種類
Dbg_BRESP ⁴	MDM への書き込み応答	std_logic_vector	パラレル出力
Dbg_BVALID ⁴	MDM からの書き込み応答有効	std_logic	パラレル入力
Dbg_RDATA ⁴	MDM への読み出しデータ	std_logic_vector	パラレル出力
Dbg_RREADY ⁴	MDM への読み出しデータレディ	std_logic	パラレル出力
Dbg_RRESP ⁴	MDM への読み出しデータ応答	std_logic_vector	パラレル出力
Dbg_RVALID ⁴	MDM からの読み出しデータ有効	std_logic	パラレル入力
Dbg_WDATA ⁴	MDM からの書き込みデータ	std_logic_vector	パラレル入力
Dbg_WREADY ⁴	MDM への書き込みデータレディ	std_logic	パラレル出力
Dbg_WVALID ⁴	MDM からの書き込みデータ有効	std_logic	パラレル入力
DEBUG_ACLK ⁴	デバッグクロック、Clk と同じである必要がある。	std_logic	パラレル入力
DEBUG_ARESET ⁴	デバッグリセット、リセットと同じである必要がある。	std_logic	パラレル入力

1. MicroBlaze v7.00 でアップデート: Dbg_Shift を追加、Debug_Rst を DEBUG バスに含める
2. MicroBlaze v9.3 でアップデート: Dbg_Trig 信号を DEBUG バスに追加
3. MicroBlaze v9.4 でアップデート: 外部プログラムトレース信号を DEBUG バスに追加
4. MicroBlaze v10.0 でアップデート: パラレルデバッグ信号を DEBUG バスに追加

トレースインターフェイスについて

MicroBlazeプロセッサはトレース目的でさまざまな内部信号をエクスポートします。この信号インターフェイスは標準化されておらず、新しいリビジョンのプロセッサでは、信号選択や機能性において、下位互換性が得られない可能性があります。これらの信号のカスタムロジックを設計せず、ザイリンクス提供の解析IPを使用して信号を使用することを推奨します。トレース信号はTRACEバスでグループにまとめられます。トレース信号の現在のセットは、MicroBlaze v7.30でアップデートされたもので、表3-16にリストされています。

MSRビットのマップは表3-17に示されています。マシンステータスレジスタ(MSR)の詳細は、第2章の「特殊用途レジスタ」を参照してください。

トレース例外タイプは表3-18を参照してください。すべての未使用のトレース例外タイプは予約されています。

表3-16: MicroBlazeトレース信号

信号名	説明	VHDLタイプ	方向
Trace_Valid_Instr	トレースポートでの有効命令。	std_logic	出力
Trace_Instruction ¹	命令コード	std_logic_vector(0~31)	出力
Trace_PC ¹	プログラムカウンター、N=32~64、64ビットMicroBlazeの場合はC_ADDR_SIZEで指定、それ以外の場合は32。	std_logic_vector(0~31)	出力
Trace_Reg_Write ¹	レジスタファイルへの命令書き込み	std_logic	出力
Trace_Reg_Addr ¹	デスティネーションレジスタアドレス	std_logic_vector(0~4)	出力
Trace_MSR_Reg ¹	マシンステータスレジスタ。レジスタビットのマップは下を参照。	std_logic_vector(0~14) ¹	出力
Trace_PID_Reg ¹	プロセスIDレジスタ	std_logic_vector(0~7)	出力
Trace_New_Reg_Value ¹	デスティネーションレジスタのアップデート値、N=C_DATA_SIZE	std_logic_vector(0~N-1)	出力
Trace_Exception_Taken ^{1,2}	例外の命令値	std_logic	出力
Trace_Exception_Kind ¹	例外タイプ。例外タイプの説明は下を参照。	std_logic_vector(0~4) ²	出力
Trace_Jump_Taken ¹	True評価された分岐命令(分岐している)	std_logic	出力
Trace_Jump_Hit ^{1,3}	分岐先キャッシュヒット	std_logic	出力
Trace_Delay_Slot ¹	命令が分岐した分岐の遅延スロットにある	std_logic	出力
Trace_Data_Access ¹	有効なD側のメモリアクセス	std_logic	出力
Trace_Data_Address ¹	D側のメモリアクセスのアドレス、N=32~64、C_ADDR_SIZEで決まる。	std_logic_vector(0~N-1)	出力
Trace_Data_Write_Value ¹	D側メモリ書き込みアクセスの値、N=C_DATA_SIZE	std_logic_vector(0~N-1)	出力
Trace_Data_Byt_Enable ¹	D側メモリアクセスのバイトインデックス、N=C_DATA_SIZE/8	std_logic_vector(0~N-1)	出力
Trace_Data_Read ¹	D側のメモリアクセスが読み出し	std_logic	出力
Trace_Data_Write ¹	D側のメモリアクセスが書き込み	std_logic	出力

表 3-16: MicroBlaze トレース信号 (続き)

信号名	説明	VHDL タイプ	方向
Trace_DCache_Req	データメモリアドレスが D キャッシュ範囲内にある。メモリアクセス命令が実行される場合に設定。	std_logic	出力
Trace_DCache_Hit	データメモリアドレスが D キャッシュにある。キャッシュヒットの発生時に Trace_DCache_Req と同時に設定。	std_logic	出力
Trace_DCache_Rdy	データアドレスが D キャッシュ範囲内にあり、アクセスが完了している。 Trace_DCache_Req = 1 および Trace_DCache_Hit = 0 を使用した要求後にのみセット。	std_logic	出力
Trace_DCache_Read	D キャッシュ要求が読み出された。 Trace_DCache_Req = 1 の場合のみ有効。	std_logic	出力
Trace_ICache_Req	命令メモリアドレスが I キャッシュ範囲内にあり、そのキャッシュがマシンステータスレジスタでイネーブルになる。命令が命令ブリッヂバッファーに読み込まれる際に設定。	std_logic	出力
Trace_ICache_Hit	命令メモリアドレスが I キャッシュにある。キャッシュヒットの発生時に Trace_ICache_Req と同時に設定。	std_logic	出力
Trace_ICache_Rdy	<ul style="list-style-type: none"> 命令メモリアドレスが I キャッシュにある。キャッシュヒットの発生時に Trace_ICache_Req と同時に設定。 命令アドレスが D キャッシュ範囲内にあり、アクセスが完了している。 Trace_ICache_Req = 1 および Trace_ICache_Hit = 0 を使用した要求後にセット。 	std_logic	出力
Trace_OF_PipeRun	デコード段のパイプラインアドバンス	std_logic	出力
Trace_EX_PipeRun ³	実行段のパイプラインアドバンス	std_logic	出力
Trace_MEM_PipeRun ³	メモリ段のパイプラインアドバンス	std_logic	出力
Trace_MB_Halted	デバッグによりパイプラインが停止	std_logic	出力

1. Trace_Valid_Instr = 1 の場合にのみ有効
2. Trace_Exception_Taken = 1 の場合にのみ有効
3. エリア最適化機能と一緒にには使用できない。

表 3-17: トレース MSR のマップ

Trace_MSR_Reg	マシンステータス レジスタ		
ビット	ビット ¹	名前	説明
0	17、49	VMS	仮想保護モード保存
1	18、50	VM	仮想保護モード
2	19、51	UMS	ユーザー モード保存
3	20、52	UM	ユーザー モード
4	21、53	PVR	プロセッサ バージョン レジスタの有無
5	22、54	EIP	処理中例外
6	23、55	EE	例外イネーブル
7	24、56	DCE	データ キャッシュ イネーブル
8	25、57	DZO	ゼロ除算、または除算オーバーフロー
9	26、58	ICE	命令キャッシュ イネーブル
10	27、59	FSL	AXI4-Stream エラー
11	28、60	BIP	処理中ブレーク
12	29、61	C	演算キャリー
13	30、62	IE	割り込みイネーブル
14	31、63	予約	予約

1. ビット番号は、64ビット MicroBlaze (C_DATA_SIZE = 64) がイネーブルかどうかによります。

表 3-18: トレース例外のタイプ

Trace_Exception_Kind [0:4]	説明
00000	ストリーム例外
00001	アラインされていない例外
00010	無効なオペコード例外
00011	命令バス例外
00100	データ バス例外
00101	除算例外
00110	FPU 例外
00111	特権命令例外
01010	割り込み
01011	外部のマスク不可能なブレーク
01100	外部のマスク可能なブレーク
10000	データ ストレージ例外
10001	命令ストレージ例外
10010	データ TLB ミス例外
10011	命令 TLB ミス例外

MicroBlaze コアのコンフィギュレーション

MicroBlaze コアは、ユーザーが細かく設定できるように開発されています。つまり、具体的なコストやパフォーマンスの要件を満たすことができるように、プロセッサを設定できます。

コンフィギュレーションは、通常、機能のオン/オフやサイズを決めたり、プロセッサの機能を選択するためのパラメーターを使用して実行します。たとえば、命令キャッシュは C_USE_ICACHE パラメーターを設定するとインエーブルになります。命令キャッシュのサイズ、キャッシュ可能なメモリの範囲の設定は、C_CACHE_BYTE_SIZE、C_ICACHE_BASEADDR、C_ICACHE_HIGHADDR をそれぞれ設定できます。

最新バージョンの MicroBlaze に有効なパラメーターは表 3-19 にリストされています。この表にリストされているパラメーターがすべて古いバージョンの MicroBlaze で認識されるわけではありませんが、コンフィギュレーションは完全に下位互換性があります。

注記: グレーで色分けされている行は、パラメーターの値が固定値であって、変更できないことを示しています。

表 3-19: コンフィギュレーションパラメーター

パラメーター名	機能/説明	設定可能な値	デフォルト	ツール指定	VHDL タイプ
C_FAMILY	ターゲット ファミリ	表 3-20 にリスト	virtex7	可	文字列
C_DATA_SIZE	データ サイズ 32: 32 ビット MicroBlaze 64: 64 ビット MicroBlaze	32、64	32		整数
C_ADDR_SIZE	アドレス サイズ	32 ~ 64	32	なし	整数
C_DYNAMIC_BUS_SIZING	レガシ	1	1	なし	整数
C_SCO	ザイリンクス内部	0	0	なし	整数
C_AREA_OPTIMIZED	インプリメンテーション最適化の選択: 0: パフォーマンス 1: エリア 2: 周波数	0、1、2	0		整数
C_OPTIMIZATION	今後の使用のために予約	0	0	なし	整数
C_INTERCONNECT	インターネットを選択 2: AXI4 のみ 3: AXI4 および ACE	2、3	2		整数
C_ENDIANNESS	エンディアンを選択 1: リトル エンディアン	1	1	可	整数
C_BASE_VECTORS ¹	コンフィギュレーション可能なベース ベクター	0x0 ~ 0xFFFFFFFF FFFFFF	0x0		std_logic_vector

表 3-19: コンフィギュレーション パラメーター (続き)

パラメーター名	機能/説明	設定可能な値	デフォルト	ツール指定	VHDL タイプ
C_FAULT_TOLERANT	フォールト トレランスをインプリメント	0、1	0	可	整数
C_ECC_USE_CE_EXCEPTION	訂正可能な ECC エラーに対し例外を生成	0、1	0		整数
C_LOCKSTEP_SLAVE	ロックステップ スレーブ	0、1	0		整数
C_AVOID_PRIMITIVES	FPGA プリミティブを使用禁止 0: なし 1: SRL 2: LUTRAM 3: 両方	0、1、2、3	0		整数
C_ENABLE_DISCRETE_PORTS	ディスクリート ポートを表示	0、1	0		整数
C_PVR	プロセッサ バージョン レジスタ モードの選択 0: なし 1: 基本 2: フル	0、1、2	0		整数
C_PVR_USER1	プロセッサ バージョン レジスタ USER1 定数	0x00-0xff	0x00		std_logic_vector (0 ~ 7)
C_PVR_USER2	プロセッサ バージョン レジスタ USER2 定数	0x00000000-0xffffffff	0x00000000 0		std_logic_vector (0 ~ 31)
C_RESET_MSR_IE C_RESET_MSR_BIP C_RESET_MSR_ICE C_RESET_MSR_DCE C_RESET_MSR_EE C_RESET_MSR_EIP	MSR レジスタ ビット (IE、BIP、ICE、DCE、EE、および EIP) のリセット値。	個々のビットのどの組み合わせでも可。	0x0000		std_logic
C_INSTANCE	インスタンス名	任意のインスタンス名	microblaze	可	文字列
C_D_AXI	データ側 AXI インターフェイス	0、1	0		整数
C_D_LMB	データ側 LMB インターフェイス	0、1	1		整数
C_I_AXI	命令側 AXI インターフェイス	0、1	0		整数

表 3-19: コンフィギュレーション パラメーター (続き)

パラメーター名	機能/説明	設定可能な値	デフォルト	ツール指定	VHDL タイプ
C_I_LMB	命令側 LMB インターフェイス	0、1	1		整数
C_USE_BARREL	バレルシフターを含める	0、1	0		整数
C_USE_DIV	ハードウェア ドライバー を含める	0、1	0		整数
C_USE_HW_MUL	ハードウェア乗算器を 含める 0: なし 1: Mul32 2: Mul64	0、1、2	1		整数
C_USE_FPU	ハードウェア浮動小数点 ユニットを含める 0: なし 1: 基本 2: 拡張	0、1、2	0		整数
C_USE_MSR_INSTR	MSRSET および MSRCLR 命令を使用	0、1	1		整数
C_USE_PCMP_INSTR	CLZ、PCMPBF、 PCMPEQ、PCMPNE 命令 を使用	0、1	1		整数
C_USE_REORDER_INSTR	逆ロード、逆ストア、 スワップ命令を使用	0、1	1		整数
C_UNALIGNED_EXCEPTIONS	アラインされていない データ アクセスの例外処 理を実行	0、1	0		整数
C_ILL_OPCODE_EXCEPTION	無効なオペコードの例外 処理を実行	0、1	0		整数
C_M_AXI_I_BUS_EXCEPTION	M_AXI_I バス エラーの 例外処理を実行	0、1	0		整数
C_M_AXI_D_BUS_EXCEPTION	M_AXI_D バス エラーの 例外処理を実行	0、1	0		整数
C_DIV_ZERO_EXCEPTION	ゼロ除算または除算 オーバーフローの例外 処理を実行	0、1	0		整数
C_FPU_EXCEPTION	ハードウェア浮動小数点 ユニットの例外処理を 実行	0、1	0		整数

表 3-19: コンフィギュレーション パラメーター (続き)

パラメーター名	機能/説明	設定可能な値	デフォルト	ツール指定	VHDL タイプ
C_OPCODE_0x0_ILLEGAL	opcode 0x0 を無効な命令として検出	0、1	0		整数
C_FSL_EXCEPTION	ストリーム リンクの例外処理を実行	0、1	0		整数
C_ECC_USE_CE_EXCEPTION	訂正可能なエラーに対しバス エラー例外を生成	0、1	0		整数
C_USE_STACK_PROTECTION	スタック オーバーフローまたはスタック アンダー フローに対し例外を生成	0、1	0		整数
C_IMPRECISE_EXCEPTIONS	LMB メモリの ECC エラーに対してあいまい例外を許可	0、1	0		整数
C_DEBUG_ENABLED	MDM デバッグ インターフェイス 0: なし 1: 基本 2: 拡張	0、1、2	1		整数
C_NUMBER_OF_PC_BRK	ハードウェア ブレーク ポイントの数	0 ~ 8	1		整数
C_NUMBER_OF_RD_ADDR_BRK	読み出しアドレス ウオッチポイントの数	0 ~ 4	0		整数
C_NUMBER_OF_WR_ADDR_BRK	書き込みアドレス ウオッチポイントの数	0 ~ 4	0		整数
C_DEBUG_EVENT_COUNTERS	パフォーマンス 監視 イベント カウンターの数	0 ~ 48	5		整数
C_DEBUG_LATENCY_COUNTERS	パフォーマンス 監視 レイテンシ カウンターの数	0 ~ 7	1		整数
C_DEBUG_COUNTER_WIDTH	パフォーマンス 監視 カウンター幅	32、48、64	32		整数
C_DEBUG_TRACE_SIZE	トレース バッファー サイズ エンベデッド: 0、 \geq 8192 外部: 0、32 ~ 8192	0、32、64、 128、256、 8192、16384、 32768、 65536、 131072	8192		整数

表 3-19: コンフィギュレーション パラメーター (続き)

パラメーター名	機能/説明	設定可能な値	デフォルト	ツール指定	VHDL タイプ
C_DEBUG_PROFILE_SIZE	プロファイルバッファー サイズ	0、4096、 8192、16384、 32768、 65536、 131072	0		整数
C_DEBUG_EXTERNAL_TRACE	外部プログラムトレース	0、1	0	可	整数
C_DEBUG_INTERFACE	デバッグ インターフェイス: 0: デバッグシリアル 1: デバッグパラレル 2: AXI4-Lite	0、1、2	0		整数
C_ASYNC_INTERRUPT	非同期割り込み	0、1	0	可	整数
C_ASYNC_WAKEUP	非同期ウェークアップ	00、01、10、 11	00	可	整数
C_INTERRUPT_IS_EDGE	レベル/エッジ インクリメント	0、1	0	可	整数
C_EDGE_IS_POSITIVE	負/正のエッジ割り込み	0、1	1	可	整数
C_FSL_LINKS	AXI4-Stream インターフェイスの数	0 ~ 16	0		整数
C_USE_EXTENDED_FSL_INSTR	拡張ストリーム命令を使用	0、1	0		整数
C_ICACHE_BASEADDR	命令キャッシュベース アドレス	0x0 ~ 0xFFFFFFFF FFFFFF	0x0		std_logic _vector
C_ICACHE_HIGHADDR	命令キャッシュハイ アドレス	0x0 ~ 0xFFFFFFFF FFFFFF	0x3FFFFF FF		std_logic _vector
C_USE_ICACHE	命令キャッシュ	0、1	0		整数
C_ALLOW_ICACHE_WR	命令キャッシュライト イネーブル	0、1	1		整数
C_ICACHE_LINE_LEN	命令キャッシュラインの 長さ	4、8、16	4		整数
C_ICACHE_ALWAYS_USED	命令キャッシュインターフェイスが、キャッシュ可能範囲内のすべてのメモリアクセスに対して使用される	0、1	1		整数

表 3-19: コンフィギュレーションパラメーター(続き)

パラメーター名	機能/説明	設定可能な値	デフォルト	ツール指定	VHDL タイプ
C_ICACHE_FORCE_TAG_LUTRAM	命令キャッシュタグは分散 RAM を使用して常にインプリメント	0、1	0		整数
C_ICACHE_STREAMS	命令キャッシュストリーム	0、1	0		整数
C_ICACHE_VICTIMS	命令キャッシュビクティム	0、2、4、8	0		整数
C_ICACHE_DATA_WIDTH	命令キャッシュデータ幅 0: 32 ビット 1: フルキャッシュライン 2: 512 ビット	0、1、2	0		整数
C_ADDR_TAG_BITS	命令キャッシュアドレスタグ	0 ~ 25	17	可	整数
C_CACHE_BYTE_SIZE	命令キャッシュサイズ	64、128、 256、512、 1024、2048、 4096、8192、 16384、 32768、 65536 ¹	8192		整数
C_DCACHE_BASEADDR	データキャッシュベースアドレス	0x0 ~ 0xFFFFFFFF FFFFFF	0x0		std_logic_vector
C_DCACHE_HIGHADDR	データキャッシュハイアドレス	0x0 ~ 0xFFFFFFFF FFFFFF	0x3FFFFF FF		std_logic_vector
C_USE_DCACHE	データキャッシュ	0、1	0		整数
C_ALLOW_DCACHE_WR	データキャッシュライトイネーブル	0、1	1		整数
C_DCACHE_LINE_LEN	データキャッシュラインの長さ	4、8、16	4		整数
C_DCACHE_ALWAYS_USED	データキャッシュインターフェイスが、キャッシュ可能範囲内のすべてのアクセスに対して使用される	0、1	1		整数
C_DCACHE_FORCE_TAG_LUTRAM	データキャッシュタグは分散 RAM を使用して常にインプリメント	0、1	0		整数

表 3-19: コンフィギュレーション パラメーター (続き)

パラメーター名	機能/説明	設定可能な値	デフォルト	ツール指定	VHDL タイプ
C_DCACHE_USE_WRITEBACK	データキャッシュライトバックストレージポリシーを使用	0, 1	0		整数
C_DCACHE_VICTIMS	データキャッシュビクティム	0, 2, 4, 8	0		整数
C_DCACHE_DATA_WIDTH	データキャッシュデータ幅 0: 32 ビット 1: フルキャッシュライン 2: 512 ビット	0, 1, 2	0		整数
C_DCACHE_ADDR_TAG	データキャッシュアドレスタグ	0 ~ 25	17	可	整数
C_DCACHE_BYTE_SIZE	データキャッシュサイズ	64, 128, 256, 512, 1024, 2048, 4096, 8192, 16384, 32768, 65536 ²	8192		整数
C_USE_MMU ³	メモリ管理: 0: なし 1: ユーザー モード 2: 保護 3: 仮想	0, 1, 2, 3	0		整数
C_MMU_DTLB_SIZE ³	データシャドウ変換ルックアサイドバッファーサイズ	1, 2, 4, 8	4		整数
C_MMU_ITLB_SIZE ³	命令シャドウ変換ルックアサイドバッファーサイズ	1, 2, 4, 8	2		整数
C_MMU_TLB_ACCESS ³	メモリ管理特殊レジスタへのアクセス: 0: 最小 1: 読み出し 2: 書き込み 3: フル	0, 1, 2, 3	3		整数
C_MMU_ZONES ³	メモリ保護ゾーンの数	0 ~ 16	16		整数

表 3-19: コンフィギュレーション パラメーター (続き)

パラメーター名	機能/説明	設定可能な値	デフォルト	ツール指定	VHDL タイプ
C_MMU_PRIVILEGED_INSTR ³	特権命令 0: フル保護 1: ストリーム命令を許可 2: 拡張アドレスを許可 3: 両方を許可	0、1、2、3	0		整数
C_USE_INTERRUPT	割り込み処理を使用 0: 割り込みなし 1: 標準割り込み 2: 低レイテンシ割り込み	0、1、2	1	可	整数
C_USE_EXT_BRK	外部ブレーク処理を使用	0、1	0	可	整数
C_USE_EXT_NM_BRK	外部のマスク不可能な ブレーク処理を使用	0、1	0	可	整数
C_USE_NON_SECURE	対応する非セキュア入力 を使用	0 ~ 15	0	可	整数
C_USE_BRANCH_TARGET_CACHE ³	分岐先キャッシュを使用	0、1	0		整数
C_BRANCH_TARGET_CACHE_SIZE ³	分岐先キャッシュ サイズ: 0: デフォルト 1: 8 エントリ 2: 16 エントリ 3: 32 エントリ 4: 64 エントリ 5: 512 エントリ 6: 1024 エントリ 7: 2048 エントリ	0 ~ 7	0		整数
C_M_AXI_DP_THREAD_ID_WIDTH	データ側 AXI スレッド ID の幅	1	1		整数
C_M_AXI_DP_DATA_WIDTH	データ側 AXI データ幅	32	32		整数
C_M_AXI_DP_ADDR_WIDTH	データ側 AXI アドレス幅	32 ~ 64	32	可	整数
C_M_AXI_DP_SUPPORTS_THREADS	データ側 AXI はスレッド を使用	0	0		整数
C_M_AXI_DP_SUPPORTS_READ	読み出しアクセスのため のデータ側 AXI サポート	1	1		整数
C_M_AXI_DP_SUPPORTS_WRITE	書き込みアクセスのため のデータ側 AXI サポート	1	1		整数
C_M_AXI_DP_SUPPORTS_NARROW_BURST	データ側 AXI ナロー バースト サポート	0	0		整数

表 3-19: コンフィギュレーションパラメーター(続き)

パラメーター名	機能/説明	設定可能な値	デフォルト	ツール指定	VHDL タイプ
C_M_AXI_DP_PROTOCOL	データ側 AXI プロトコル	AXI4、 AXI4LITE	AXI4 LITE	可	文字列
C_M_AXI_DP_EXCLUSIVE_ACCESS	データ側 AXI 排他的 アクセスサポート	0、1	0		整数
C_M_AXI_IP_THREAD_ID_WIDTH	命令側 AXI スレッド ID の幅	1	1		整数
C_M_AXI_IP_DATA_WIDTH	命令側 AXI データ幅	32	32		整数
C_M_AXI_IP_ADDR_WIDTH	命令側 AXI アドレス幅	32 ~ 64	32	可	整数
C_M_AXI_IP_SUPPORTS_THREADS	命令側 AXI はスレッドを 使用	0	0		整数
C_M_AXI_IP_SUPPORTS_READ	読み出しアクセスのため の命令側 AXI サポート	1	1		整数
C_M_AXI_IP_SUPPORTS_WRITE	書き込みアクセスのため の命令側 AXI サポート	0	0		整数
C_M_AXI_IP_SUPPORTS_NARROW_BURST	命令側 AXI ナロー バーストサポート	0	0		整数
C_M_AXI_IP_PROTOCOL	命令側 AXI プロトコル	AXI4LITE	AXI4 LITE		文字列
C_M_AXI_DC_THREAD_ID_WIDTH	データキャッシュ AXI ID の幅	1	1		整数
C_M_AXI_DC_DATA_WIDTH	データキャッシュ AXI データ幅	32、64、128、 256、512	32		整数
C_M_AXI_DC_ADDR_WIDTH	データキャッシュ AXI アドレス幅	32 ~ 64	32	可	整数
C_M_AXI_DC_SUPPORTS_THREADS	データキャッシュ AXI は スレッドを使用	0	0		整数
C_M_AXI_DC_SUPPORTS_READ	読み出しアクセスのため のデータキャッシュ AXI サポート	1	1		整数
C_M_AXI_DC_SUPPORTS_WRITE	書き込みアクセスのため のデータキャッシュ AXI サポート	1	1		整数
C_M_AXI_DC_SUPPORTS_NARROW_BURST	データキャッシュ AXI ナローバーストサポート	0	0		整数
C_M_AXI_DC_SUPPORTS_USER_SIGNALS	データキャッシュ AXI ユーザー信号サポート	1	1		整数

表 3-19: コンフィギュレーション パラメーター (続き)

パラメーター名	機能/説明	設定可能な値	デフォルト	ツール指定	VHDL タイプ
C_M_AXI_DC_PROTOCOL	データ キャッシュ AXI プロトコル	AXI4	AXI4		文字列
C_M_AXI_DC_AWUSER_WIDTH	データ キャッシュ AXI ユーザー幅	5	5		整数
C_M_AXI_DC_ARUSER_WIDTH	データ キャッシュ AXI ユーザー幅	5	5		整数
C_M_AXI_DC_WUSER_WIDTH	データ キャッシュ AXI ユーザー幅	1	1		整数
C_M_AXI_DC_RUSER_WIDTH	データ キャッシュ AXI ユーザー幅	1	1		整数
C_M_AXI_DC_BUSER_WIDTH	データ キャッシュ AXI ユーザー幅	1	1		整数
C_M_AXI_DC_EXCLUSIVE_ACCESS	データ キャッシュ AXI 排他的アクセス サポート	0、1	0		整数
C_M_AXI_DC_USER_VALUE	データ キャッシュ AXI ユーザー値	0 ~ 31	31		整数
C_M_AXI_IC_THREAD_ID_WIDTH	命令キャッシュ AXI ID の幅	1	1		整数
C_M_AXI_IC_DATA_WIDTH	命令キャッシュ AXI データ幅	32、64、128、256、512	32		整数
C_M_AXI_IC_ADDR_WIDTH	命令キャッシュ AXI データ幅	32 ~ 64	32	可	整数
C_M_AXI_IC_SUPPORTS_THREADS	データ キャッシュ AXI はスレッドを使用	0	0		整数
C_M_AXI_IC_SUPPORTS_READ	読み出しアクセスのための命令キャッシュ AXI サポート	1	1		整数
C_M_AXI_IC_SUPPORTS_WRITE	書き込みアクセスのための命令キャッシュ AXI サポート	0	0		整数
C_M_AXI_IC_SUPPORTS_NARROW_BURST	命令キャッシュ AXI ナローバースト サポート	0	0		整数
C_M_AXI_IC_SUPPORTS_USER_SIGNALS	命令キャッシュ AXI ユーザー信号サポート	1	1		整数
C_M_AXI_IC_PROTOCOL	命令キャッシュ AXI プロトコル	AXI4	AXI4		文字列

表 3-19: コンフィギュレーションパラメーター(続き)

パラメーター名	機能/説明	設定可能な値	デフォルト	ツール指定	VHDL タイプ
C_M_AXI_IC_AWUSER_WIDTH	命令キャッシュ AXI ユーザー幅	5	5		整数
C_M_AXI_IC_ARUSER_WIDTH	命令キャッシュ AXI ユーザー幅	5	5		整数
C_M_AXI_IC_WUSER_WIDTH	命令キャッシュ AXI ユーザー幅	1	1		整数
C_M_AXI_IC_RUSER_WIDTH	命令キャッシュ AXI ユーザー幅	1	1		整数
C_M_AXI_IC_BUSER_WIDTH	命令キャッシュ AXI ユーザー幅	1	1		整数
C_M_AXI_IC_USER_VALUE	命令キャッシュ AXI ユーザー値	0 ~ 31	31		整数
C_STREAM_INTERCONNECT	AXI4-Stream インターコネクトを選択	0、1	0		整数
C_Mn_AXIS_PROTOCOL	AXI4-Stream プロトコル	GENERIC	GENERIC		文字列
C_Sn_AXIS_PROTOCOL	AXI4-Stream プロトコル	GENERIC	GENERIC		文字列
C_Mn_AXIS_DATA_WIDTH	AXI4-Stream マスター データ幅	32	32	なし	整数
C_Sn_AXIS_DATA_WIDTH	AXI4-Stream スレーブ データ幅	32	32	なし	整数
C_NUM_SYNC_FF_CLK	リセットおよび Wakeup[0:1] の同期段	≥ 0	2		整数
C_NUM_SYNC_FF_CLK_IRQ	割り込み入力信号の 同期段	≥ 0	1		整数
C_NUM_SYNC_FF_CLK_DEBUG	Dbg_シリアル信号の 同期段	≥ 0	2		整数
C_NUM_SYNC_FF_DBG_CLK	Dbg_Clk への内部同期段	≥ 0	1		整数
C_NUM_SYNC_FF_DBG_TRACE_CLK	Dbg_Trace_Clk への内部 同期段	≥ 0	1		整数

1. 最下位ビットから 7 つのビットはすべて 0。
2. アーキテクチャによって指定できるサイズは変わります。キャッシュは、0 から 32 までの RAMB プリミティブを使用します(キャッシュ サイズが 2048 未満の場合は 0)。
3. C_AREA_OPTIMIZED が 1(エリア)に設定されているときは使用できません。

表 3-20: パラメーター C_FAMILY で使用可能な値

	設定可能な値
Artix®	aartix7 artix7 artix7l qartix7 qartix7l
Kintex®	kintex7 kintex7l qkintex7 qkintex7l kintexu kintexuplus
Spartan®	spartan7
Virtex®	qvirtex7 virtex7 virtexu virtexuplus virtexuplusHBM
Zynq®	azynq zynq qzynq zynquplus zynquplusRFSOC

MicroBlaze アプリケーションバイナリ インターフェイス

概要

この章では、MicroBlaze™ 用にアセンブリ言語でソフトウェアを開発するのに重要なアプリケーションバイナリ インターフェイス (ABI) について説明します。MicroBlaze の GNU コンパイラは、この資料で説明されている規則に従います。アセンブリ プログラムによって記述されるコードもすべて、コンパイラで生成されたコードとの互換性を保つために、同じ規則に従います。また、割り込みおよび例外処理も手短に説明します。

データ型

次の表に、MicroBlaze のアセンブリ プログラムで使用されるデータ型を示します。通常のバイト、ハーフバイト、ワード レジスタの代わりに data8、data16、data32、data64 などのデータ型が使用されます。

表 4-1: MicroBlaze アセンブリ プログラムのデータ型

MicroBlaze データ型 (アセンブリ プログラム)	対応する ANSI C データ型 32 ビット MicroBlaze	対応する ANSI C データ型 64 ビット MicroBlaze	サイズ (バイト)
data8	char	char	1
data16	short	short	2
data32	int	int	4
	long int	-	4
	float	float	4
	enum	enum	4
data16/data32	pointer ¹	-	2/4
data64	-	long int	8
	long long int	long long int	8
	-	double	8
	-	pointer	8

1. グローバルポインターでアクセス可能なスマールデータエリアへのポインターは data16 です。

レジスタの使用規則

次の表は、MicroBlaze でのレジスタ使用規則をまとめています。

表 4-2: レジスタの使用規則

レジスタ	タイプ	SW/HW	目的
R0	専用	HW	値 0
R1	専用	SW	スタッカ ポインター
R2	専用	SW	読み出し専用のスモールデータ エリア アンカー
R3-R4	揮発性	SW	戻り値/暫定値
R5-R10	揮発性	SW	パラメーター / 暫定値
R11-R12	揮発性	SW	暫定値
R13	専用	SW	読み出し/書き込みのスモールデータ エリア アンカー
R14	専用	HW	割り込みの戻りアドレス
R15	専用	SW	サブルーチンの戻りアドレス
R16	専用	HW	トラップ(デバッガー) の戻りアドレス
R17	専用	HW/SW	例外の戻りアドレス ハードウェア例外をサポートする場合は HW、そうでなければ SW
R18	専用	SW	アセンブラー / コンパイラの暫定値のため予約
R19	不揮発性	SW	関数呼び出し全体で保存しておく必要あり。呼び出し先保存。
R20	専用 または 不揮発性	SW	位置独立コード (PIC) のグローバルオフセットテーブル (GOT) への ポインターを格納するために予約。非 PIC コードで不揮発性。関数 呼び出し全体で保存しておく必要あり。呼び出し先保存。
R21-R31	不揮発性	SW	関数呼び出し全体で保存しておく必要あり。呼び出し先保存。
RPC	特殊	HW	プログラム カウンター
RMSR	特殊	HW	マシン ステータス レジスタ
REAR	特殊	HW	例外アドレス レジスタ
RESR	特殊	HW	例外ステータス レジスタ
RFSR	特殊	HW	浮動小数点ステータス レジスタ
RBTR	特殊	HW	分岐先 レジスタ
REDR	特殊	HW	例外データ レジスタ
RPID	特殊	HW	プロセス ID レジスタ
RZPR	特殊	HW	ゾーン保護 レジスタ
RTLblo	特殊	HW	変換ルックアサイド バッファー ロウ レジスタ
RTLBHI	特殊	HW	変換ルックアサイド バッファー ハイ レジスタ
RTLBX	特殊	HW	変換ルックアサイド バッファー インデックス レジスタ
RTLBSX	特殊	HW	変換ルックアサイド バッファー 検索インデックス
RPVR0-12	特殊	HW	プロセッサ バージョン レジスタ 0 から 12 まで

MicroBlaze のアーキテクチャは 32 個の汎用レジスタ (GPR) を定義します。これらのレジスタは、揮発性、不揮発性、専用の 3 つに分類されます。

- 挥発性レジスタ (呼び出し元保存) は、一時的なものであり、関数呼び出し間で値は保持されません。レジスタ R3 ~ R12 は揮発性で、R3 および R4 は呼び出し元関数に値を返すために使用されます。レジスタ R5 ~ R10 は、サブルーチン間でパラメーターを渡すために使用されます。
- レジスタ R19 ~ R31 の内容は関数呼び出し間で保持されるので、不揮発性レジスタ (呼び出し先保存) と呼ばれます。呼び出し先関数は、使用されている不揮発性レジスタに値を保存します。これらは通常プロローグ中にスタックに保存され、エピローグ中にリロードされます。
- 一部のレジスタは専用レジスタとして使用されるので、指定された目的以外で使用することはできません。
 - レジスタ R14 ~ R17 は、それぞれ割り込み、サブルーチン、トラップ、例外からの戻りアドレスを格納するのに使用されます。サブルーチンは分岐およびリンク命令を使用して呼び出されます。この命令は現在のプログラム カウンター (PC) をレジスタ R15 に保存します。
 - スモールデータ エリア ポインターは、16 ビットの即値で、あるメモリ ロケーションにアクセスするためには使用されます。こうしたエリアについては、この資料のメモリ モデルに関するセクションで説明します。読み出し専用のスモールデータ エリア (SDA) アンカー R2 (読み出し専用) は、リテラルなどの定数にアクセスするために使用されます。もう 1 つの SDA アンカー R13 (読み出し/書き込み) は、スモールデータの読み出し/書き込みセクションの値にアクセスするのに使用されます。
 - レジスタ R1 はスタック ポインター値を格納し、関数に入るときおよびそこから出るときにアップデートされます。
 - レジスタ R18 はアセンブラー操作用の一時的なレジスタとして使用されます。
- MicroBlaze には、次のような特殊用途のレジスタが含まれます。
 - プログラム カウンター (rpc)
 - マシン ステータス レジスタ (rmsr)
 - 例外ステータス レジスタ (resr)
 - 例外アドレス レジスタ (rear)
 - 浮動小数点ステータス レジスタ (rfsr)、分岐ターゲット レジスタ (rbtr)
 - 例外データ レジスタ (redr)
 - メモリ管理 レジスタ (rpid、rzpr、rtlblo、rtlghi、rtlbx、rtlbsx)
 - プロセッサ バージョン レジスタ (0 ~ 12)

これらのレジスタは、レジスタ ファイルには直接マップされていないため、その使用方法は汎用レジスタとは異なります。特殊レジスタの値は、mts および mfs 命令をそれぞれ使用して汎用レジスタから、または汎用レジスタへと転送できます。

スタック規則

MicroBlaze により使用されるスタック規則を表4-3に示します。

表4-3の太枠で囲まれている部分は呼び出し元関数のスタックフレームを表し、それ以外は呼び出し先フレーム関数を表しています。スタックフレームのABI規則は、パラメーター渡し、不揮発性レジスタの値の保持、関数のローカル変数のスペースの割り当てに関するプロトコルを定義しています。

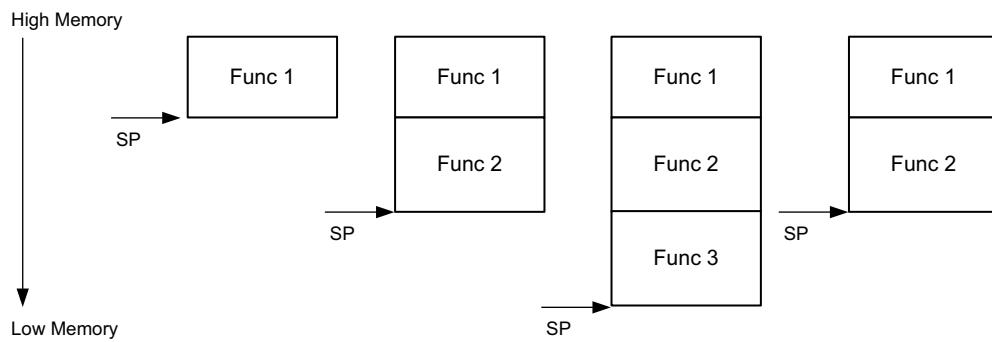
ほかのサブルーチンを呼び出すコードが含まれている関数は非リーフ関数と呼ばれます。これらの非リーフ関数は、それ用に新たなスタックフレームエリアを作成する必要があります。プログラムの実行が開始するとき、スタックポインターの値は最大値です。関数が呼び出されていくと、その値は、スタックフレームの各関数に必要なワード数分デクリメントしていきます。呼び出し元関数のスタックポインターの値は、常に呼び出し先関数よりも大きな値になります。

表4-3: スタック規則

上位アドレス	
	呼び出されたサブルーチンの関数パラメーター (Arg n .. Arg1) オプション: 現在のプロシージャから呼び出されたプロシージャに必要な引数の最大数
古いスタック ポインター	リンクレジスタ (R15)
	呼び出し先で保存されるレジスタ (R31...R19) オプション: 現在のプロシージャで使用されるレジスタのみを保存
	現在のプロシージャのローカル変数 オプション: ローカル変数がプロシージャで定義されている場合のみ存在
	ファンクションパラメーター (Arg n .. Arg 1) オプション: 現在のプロシージャから呼び出されたプロシージャに必要な引数の最大数
新しいスタック ポインター	リンクレジスタ
下位アドレス	

Func 1 が Func 2 を呼び出し、Func 2 が Func 3 を呼び出す例を考えてみます。各インスタンスのスタックは図4-1のように表されます。Func 1 が Func 2 を呼び出した後、スタックポインター (SP) の値はデクリメントします。この SP の値は、さらに Func 3 のスタックフレーム分デクリメントします。Func 3 から戻ると、SP の値は Func 2 の元の値にインクリメントします。

次の図に、スタックがどのように保持されるかを示します。



X19785-111717

図 4-1: スタック フレーム

スタックが上限を超えたとき、下限を下回らないようにするためには、スタック保護を使用できます。これには、それぞれ SHR(スタック ハイ レジスタ)および SLR(スタック ロウ レジスタ)を使用します。これらのレジスタは、crt0.o 初期化ファイルにより、リンクシンボルからのスタック制限に自動的に初期化されます。

ハードウェアでスタック保護を有効にすると、スタック サイズの問題(ほかの方法ではかなりデバッグしにくい)が原因の間違ったプログラム ビヘイビアを検出するのに便利です。

呼び出し規則

呼び出し元関数は、レジスタ (R5 ~ R10) を使用するか、それ自体のスタック フレームで、呼び出し先関数にパラメーターを渡します。呼び出し先では、渡されたパラメーターを格納するのに呼び出し元のスタック エリアを使用します。

表 4-1 を参照してください。Func 2 のパラメーターは、レジスタ (R5 ~ R10) または Func 1 に割り当てられているスタック フレームに格納されます。

Func 2 に 7つ以上の整数パラメーターがある場合、最初の 6つは R5 ~ R10 のレジスタで渡すことができますが、その後のパラメーターは Func 1 に割り当てられているスタック フレーム(オフセット SP + 28)で渡す必要があります。

メモリ モデル

MicroBlaze のメモリ モデルは、スマートデータ エリア、データ エリア、共通の未初期化エリア、リテラル/定数という 4 つの箇所にデータを分類します。

スマートデータ エリア

グローバルな初期化された変数でサイズの小さなものが、このエリアに格納されます。変数をスマートデータ エリアに格納するかどうか決め手となるサイズは、MicroBlaze の C コンパイラ (mb-gcc) で 8 バイトに設定されていますが、コンパイラでコマンド ライン オプションを指定すると変更できます。このオプションの詳細は、『エンベデッド システム ツール リファレンス マニュアル』(UG1043) [参照 13] の「GNU コンパイラ ツール」の章で説明されています。スマートデータ エリアには 64 KB のメモリが割り当てられています。このエリアには、読み出し/書き込みのスマートデータ エリア アンカー (R13) および 16 ビットのオフセットを使用してアクセスします。このエリアに小さな変数を割り当てるとき、グローバル変数にアクセスするためのコードに IMM 命令を追加する必要性を削減できます。スマートデータ エリアの変数には、絶対アドレスを使用してアクセスすることもできます。

データ エリア

データ エリアには、比較的大きな初期化済み変数が割り当てられます。データ エリアにアクセスするには、コンパイラのコマンド ライン オプションによって、読み出し/書き込みの SDA アンカー R13 または絶対アドレスを使用します。

共通の未初期化エリア

初期化されていないグローバル変数は共通エリアに割り当てられ、絶対アドレスまたは読み出し/書き込みのスマートデータ エリア アンカー (R13) を使用してアクセスできます。

リテラルまたは定数

定数は、読み出し専用のスマートデータ エリア (SDA) に格納され、読み出し専用の SDA アンカー R2 を使用してアクセスします。

コンパイラは、ベース ポインターとして動作する適切なグローバル ポインターを生成します。SDA アンカーの実際の値は、最終リンク段階でリンカーによって決定されます。メモリのさまざまなセクションの詳細は、『エンベデッド システム ツール リファレンス マニュアル』(UG1043) [参照 13] の「MicroBlaze リンカー スクリプトで割り当てられるセクション」を参照してください。

コンパイラでは、コマンド ラインのオプションに従って適切なセクションが生成されます。これらのオプションの詳細は、『エンベデッド システム ツール リファレンス マニュアル』(UG1043) [参照 13] の「GNU コンパイラ ツール」の章を参照してください。

割り込み、ブレーク、例外処理

MicroBlaze は、割り込みおよび例外を処理するため、次の表に示すアドレス ロケーションを想定します。これらのロケーションでは、コードが適切なハンドラーにジャンプするように記述されています。

表 4-4: 割り込みおよび例外処理

イベント	ハードウェアがジャンプする位置	ソフトウェア ラベル
開始/リセット	C_BASE_VECTORS + 0x0	_start
ユーザー例外	C_BASE_VECTORS + 0x8	_exception_handler
割り込み	C_BASE_VECTORS + 0x10 ¹	_interrupt_handler
ブレーク (HW/SW)	C_BASE_VECTORS + 0x18	-
ハードウェア例外	C_BASE_VECTORS + 0x20	_hw_exception_handler
ザイリンクスにより予約	C_BASE_VECTORS + 0x28 ~ C_BASE_VECTORS + 0x4F	-

1. 低レイテンシ割り込みモードでは、ベクター アドレスは割り込みコントローラーにより供給されます。

これらのロケーションに必要なコードは、次のとおりです。crt0.o 初期化ファイルは、mb-gcc コンパイラにより、リンクのため mb-ld に渡されます。このファイルは、例外ハンドラーに適切なアドレスを設定します。

次のコードは、例外、ブレーク、割り込みハンドラーに制御を渡すためのコードで、C_BASE_VECTORS はデフォルトの 0x00000000 に設定されているとします。

```

0x00: bri _start1
0x04: nop
0x08: imm high bits of address (user exception handler)
0x0c: bri _exception_handler
0x10: imm high bits of address (interrupt handler)
0x14: bri _interrupt_handler
0x18: imm high bits of address (break handler)
0x1c: bri low bits of address (break handler)
0x20: imm high bits of address (HW exception handler)
0x24: bri _hw_exception_handler

```

低レイテンシの割り込みモードの場合、このモードを使用している各割り込みに対し、直接、制御は割り込みハンドラーに渡されます。この場合、使用されているレジスタの格納および復帰は各ハンドラーで実行されます。

MicroBlaze C コンパイラ (mb-gcc) の fast_interrupt という属性を使用すると、このタスクをコンパイラで実行するように設定できます。

```
void interrupt_handler_name() __attribute__((fast_interrupt));
```

MicroBlaze では 32 ビットを使用して指定可能なアドレス ロケーションに例外および割り込みのハンドラールーチンを配置できます。

- ユーザー例外ハンドラー コードは、ラベル _exception_handler で始まります。
- ハードウェア例外ハンドラーは、ラベル _hw_exception_handler で始まります。
- 割り込みハンドラー コードは、レイテンシの短いハンドラーを使用しない割り込みの場合、ラベル _interrupt_handler で始まります。

現在の MicroBlaze システムには、割り込み、ブレーク、ユーザー例外処理に対しダミールーチンがあり、ユーザーが変更できます。これらのルーチンを上書きし、ユーザーの割り込みおよび例外ハンドラーをリンクするには、特定の属性を使用してハンドラー コードを定義する必要があります。

使用されているレジスタを格納および復帰させ、ハンドラーから戻るための `rtid` 命令を省くコードをコンパイラが生成するように、`interrupt_handler` という属性を使用して、割り込みハンドラー コードを定義する必要があります。

```
void function_name() __attribute__((interrupt_handler));
```

使用されているレジスタを格納および復帰させ、ハンドラーから戻るための `rtid` 命令を省くコードをコンパイラが生成するように、`break_handler` という属性を使用して、ブレークハンドラーコードを定義する必要があります。

```
void function_name() __attribute__((break_handler));
```

割り込みハンドラーの属性の使用方法および構文は、『エンベデッドシステムツールリファレンスマニュアル』(UG1043) [参照 13] の「GNU コンパイラツール」の章を参照してください。

ザイリンクスシステムデバッガー(XSDB)またはソフトウェア開発キット(SDK)でソフトウェアブレークポイントが使用される場合は、ブレーク(HW/SW)アドレスロケーションはソフトウェアブレークポイントを処理するため予約されます。

MicroBlaze 命令セット アーキテクチャ

概要

この章では、MicroBlaze™ プロセッサの命令セットアーキテクチャについて説明します。

記号の説明

次の表に、この章で使用される記号の定義を示します。

表 5-1: レジスタ名の記号

レジスタ名	モード	説明
rD	32 ビット	デスティネーション レジスタ r0 ~ r31、32 ビット: レジスタ全体に命令結果を割り当てる
	64 ビット	デスティネーション レジスタ r0 ~ r31、64 ビット: 下位 32 ビットに命令結果を割り当てる 上位 32 ビットを 0 にクリア
rA rB	32 ビット	ソース レジスタ r0 ~ r31、32 ビット: レジスタ全体を命令オペランドとして使用
	64 ビット	ソース レジスタ r0 ~ r31、64 ビット: 下位 32 ビットを命令オペランドとして使用 上位 32 ビットを無視
rD_L	64 ビット	デスティネーション レジスタ r0 ~ r31、64 ビット: レジスタ全体に命令結果を割り当てる
rA_L rB_L	64 ビット	ソース レジスタ r0 ~ r31、64 ビット: レジスタ全体を命令オペランドとして使用
rD_X	32 ビット 64 ビット	デスティネーション レジスタ r0 ~ r31: レジスタ全体に命令結果を割り当てる
rA_X rB_X	32 ビット	ソース レジスタ r0 ~ r31、32 ビット: レジスタ全体を命令オペランドとして使用
	64 ビット	ソース レジスタ r0 ~ r31、64 ビット: レジスタ全体を命令オペランドとして使用

表 5-2: 記号の説明

記号	説明
+	加算
-	減算
×	乗算
/	除算
∧	ビット単位の論理 AND
∨	ビット単位の論理 OR
⊕	ビット単位の論理 XOR
⊍	ビット単位の x の補数
←	代入
>>	右シフト
<<	左シフト
rx	レジスタ x
$x[i]$	レジスタ x のビット i
$x[i:j]$	レジスタ x のビット i から j まで
=	相等
≠	不一致
>	大なり
>=	それ以上
<	小なり
<=	それ以下
	信号選択
sext(x)	符号拡張 x
Mem(x)	アドレス x でのメモリ ロケーション
FSLx	AXI4-Stream インターフェイス x
LSW(x)	x の最下位ワード
isDnz(x)	浮動小数点: x が非正規化数の場合は真
isInfinite(x)	浮動小数点: x が $+\infty$ または $-\infty$ の場合は真
isPosInfinite(x)	浮動小数点: x が $+\infty$ の場合は真
isNegInfinite(x)	浮動小数点: x が $-\infty$ の場合は真
isNaN(x)	浮動小数点: x が quiet または signaling NaN の場合は真
isZero(x)	浮動小数点: x が $+0$ または -0 の場合は真
isQuietNaN(x)	浮動小数点: x が quiet NaN の場合は真
isSigNaN(x)	浮動小数点: x が signaling NaN の場合は真
signZero(x)	浮動小数点: $x > 0$ の場合は $+0$ を返し、 $x < 0$ の場合は -0 を返す
signInfinite(x)	浮動小数点: $x > 0$ の場合は $+\infty$ を返し、 $x < 0$ の場合は $-\infty$ を返す

フォーマット

MicroBlaze はタイプ A およびタイプ B という 2 つの命令フォーマットを使用します。

タイプA

タイプ A はレジスタ間の命令に使用されます。オペコード、デスティネーション レジスタを 1 つ、ソース レジスターを 2 つ含んでいます。

タイプ B

タイプ B はレジスタと即値の命令に使用されます。オペコード、デスティネーション レジスタを 1 つ、ソース レジ

オペコード	デスティネーションレジスタ	ソースレジスタA		即値
0	6	11	16	31

MicroBlaze 32 ビット命令

このセクションでは MicroBlaze の命令について説明します。命令はアルファベット順にリストされています。各命令に対し、ニーモニック、エンコーディング、説明、セマンティクスの擬似コード、変更されるレジスタのリストがザイリンクスから提供されています。

すべての命令は、32 ビット MicroBlaze の命令セットに含まれており、このセクションで定義されています。これらの命令は、64 ビット MicroBlaze の拡張命令セットの一部としても使用できます。

add 算術加算

add	rD, rA, rB	加算
addc	rD, rA, rB	キャリー付き加算
addk	rD, rA, rB	加算およびキャリー保持
addkc	rD, rA, rB	キャリー付き加算およびキャリー保持

0	0	0	K	C	0	rD	rA	rB	0	0	0	0	0	0	0	0	0	0	0
0					6		11		16		21								31

説明

レジスタ rA と rB の内容の和を、レジスタ rD に配置します。

命令のビット 3(図の K)は、ニーモニック addk に対して 1 にセットされます。命令のビット 4(図の C)は、ニーモニック addc に対して 1 にセットされます。ニーモニック addkc の場合は、両方のビットが 1 にセットされます。

add 命令でビット 3 がセットされている場合 (addk, addkc)、命令の実行結果にかかわらず、キャリー フラグは前の値を保持します。ビット 3 がクリアされている場合 (add, addc)、キャリー フラグは命令の実行結果によって変更されます。

命令のビット 4 が 1 にセットされている場合 (addc, addkc)、キャリー フラグの内容 (MSR[C]) が命令実行に影響します。ビット 4 がクリアされている場合 (add, addk)、キャリー フラグの内容は命令の実行には影響しません(標準加算の場合)。

擬似コード

```
if C = 0 then
    (rD) ← (rA) + (rB)
else
    (rD) ← (rA) + (rB) + MSR[C]
if K = 0 then
    MSR[C] ← CarryOut
```

変更されるレジスタ

- rD
- MSR[C]

レイテンシ

1 サイクル

注記

命令オペコードの C ビットは、MSB のキャリービットとは異なります。

add r0, r0, r0 (= 0x00000000) 命令は、コンパイラでは使用されず、通常はメモリが初期化されていないことを示します。無効な命令例外を使用している場合は、MicroBlaze のパラメーターを C_OPCODE_0x0_ILLEGAL=1 に設定して、これらの命令を捕捉できます。

addi 即値との算術加算

addi	rD, rA, IMM	即値との加算
addic	rD, rA, IMM	即値とのキャリー付き加算
addik	rD, rA, IMM	即値との加算およびキャリー保持
addikc	rD, rA, IMM	即値とのキャリー付き加算およびキャリー保持

0	0	1	K	C	0	rD	rA	IMM
0			6		11		16	31

説明

レジスタ rA の内容と IMM フィールドの値(32 ビットに符号拡張されたもの)の和を、レジスタ rD に配置します。命令のビット 3(図の K)は、二モニック addik に対して 1 にセットされます。命令のビット 4(図の C)は、二モニック addic に対して 1 にセットされます。二モニック addikc の場合は、両方のビットが 1 にセットされます。

addi 命令でビット 3 がセットされている場合 (addik, addikc)、命令の実行結果にかかわらず、キャリーフラグは前の値を保持します。ビット 3 がクリアされている場合 (addi, addic)、キャリーフラグは命令の実行結果によって変更されます。

命令のビット 4 が 1 にセットされている場合 (addic, addikc)、キャリーフラグの内容 (MSR[C]) が命令の実行に影響します。ビット 4 がクリアされている場合 (addi, addik)、キャリーフラグの内容は命令の実行には影響しません(標準加算の場合)。

擬似コード

```

if C = 0 then
    (rD) ← (rA) + sext(IMM)
else
    (rD) ← (rA) + sext(IMM) + MSR[C]
if K = 0 then
    MSR[C] ← CarryOut

```

変更されるレジスタ

- rD
- MSR[C]

レイテンシ

1 サイクル

注記

命令オペコードの C ビットは、MSB のキャリービットとは異なります。

デフォルトでは、タイプ B 命令は 16 ビットの IMM フィールド値を取り込み、それを 32 ビットに符号拡張して即値オペランドとして使用します。この動作は、タイプ B 命令の前に imm 命令を先行させると変更できます。32 ビットの即値の使用については、[235 ページの「imm」](#) を参照してください。

and**論理 AND**

and rD, rA, rB

1	0	0	0	0	1	rD	rA	rB	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0					6		11		16		21															31

説明

レジスタ rA の内容とレジスタ rB の内容を AND 演算し、その結果をレジスタ rD に配置します。

擬似コード
$$(rD) \leftarrow (rA) \wedge (rB)$$
変更されるレジスタ

- rD

レイテンシ

1 サイクル

andi**即値との論理 AND**

andi rD, rA, IMM

1	0	1	0	0	1	rD	rA	IMM	
0			6		11		16		31

説明

レジスタ rA の内容と IMM フィールドの値(32 ビットに符号拡張されたもの)を AND 演算し、その結果をレジスタ rD に配置します。

擬似コード

```
(rD) ← (rA) ∧ sext(IMM)
```

変更されるレジスタ

- rD

レイテンシ

1 サイクル

注記:

デフォルトでは、タイプ B 命令は 16 ビットの IMM フィールド値を取り込み、それを 32 ビットに符号拡張して即値オペランドとして使用します。この動作は、タイプ B 命令の前に imm 命令を先行させると変更できます。32 ビットの即値の使用については、「[imm](#)」を参照してください。

andn**論理 AND NOT**

andn rD, rA, rB

1	0	0	0	1	1	rD	rA	rB	0	0	0	0	0	0	0	0	0	0	0
0						6	11	16	21										31

説明

レジスタ rA の内容をレジスタ rB の内容の論理補数と AND 演算し、その結果をレジスタ rD に配置します。

擬似コード
$$(rD) \leftarrow (rA) \wedge (\overline{rB})$$
変更されるレジスタ

- rD

レイテンシ

1 サイクル

andni 即値との論理 AND NOT

andni rD, rA, IMM

1	0	1	0	1	1	rD	rA	IMM
0			6		11		16	31

説明

IMM フィールドは 32 ビットに符号拡張されています。レジスタ rA の内容と拡張された IMM フィールドの論理補数を AND 演算し、その結果をレジスタ rD に配置します。

擬似コード

$$(rD) \leftarrow (rA) \wedge \overline{(\text{sext(Imm)})}$$
変更されるレジスタ

- rD

レイテンシ

1 サイクル

注記:

デフォルトでは、タイプ B 命令は 16 ビットの IMM フィールド値を取り込み、それを 32 ビットに符号拡張して即値オペランドとして使用します。この動作は、タイプ B 命令の前に imm 命令を先行させると変更できます。32 ビットの即値の使用については、「[imm](#)」を参照してください。

beg

0 の場合は分岐

beq	rA, rB	0 の場合は分岐
beqd	rA, rB	0 の場合は遅延後に分岐

説明

`rA` が 0 の場合、`rB` のオフセット値にある命令に分岐します。分岐先は、アドレス `PC+rB` にある命令です。

二モニック **bcd** は D ビットをセットします。D ビットは、分岐遅延スロットの有無を指定します。D ビットがセットされている場合は遅延スロットがあり、分岐先の命令を実行する前に、分岐に後続する命令(分岐遅延スロットにある命令)を実行できます。D ビットがセットされていない場合は、遅延スロットではなく、分岐後に実行される命令は分岐先の命令になります。

擬似コード

```

If rA = 0 then
    PC ← PC + rB
else
    PC ← PC + 4
if D = 1 then
    allow following instruction to complete execution

```

変更されるレジスタ

- PC

レイテンシ

- 1 サイクル(分岐が取られなかった場合)
 - 2 サイクル(分岐が取られ、D ビットがセットされている場合)
 - 3 サイクル(分岐が取られ、D ビットがセットされていない場合)

注記

遅延スロットには、imm、分岐、またはブレーク命令を含めることはできません。割り込みおよび外部ハードウェアブレークは、遅延スロット分岐が完了するまで保留されます。

beqi 0 の場合は即値に分岐

beqi	rA, IMM	0 の場合は即値に分岐
beqid	rA, IMM	0 の場合は遅延後に即値に分岐

1	0	1	1	1	1	D	0	0	0	0	rA		IMM
0			6			11			16				31

説明

rA が 0 の場合、IMM のオフセット値にある命令に分岐します。分岐先は、アドレス PC + IMM にある命令です。二モニック beqid は D ビットをセットします。D ビットは、分岐遅延スロットの有無を指定します。D ビットがセットされている場合は遅延スロットがあり、分岐先の命令を実行する前に、分岐に後続する命令(分岐遅延スロットにある命令)を実行できます。D ビットがセットされていない場合は、遅延スロットではなく、分岐後に実行される命令は分岐先の命令になります。

擬似コード

```
If rA = 0 then
    PC ← PC + sext(IMM)
else
    PC ← PC + 4
if D = 1 then
    allow following instruction to complete execution
```

変更されるレジスタ

- PC

レイテンシ

- 1 サイクル(分岐が取られなかった場合、または正しく分岐予測された場合)
- 2 サイクル(分岐が取られ、D ビットがセットされている場合)
- 3 サイクル(分岐が取られ、D ビットがセットされていない場合、または C_AREA_OPTIMIZED=0 で分岐予測が間違っている場合)
- 7 ~ 9 サイクル(C_AREA_OPTIMIZED=2 で分岐予測が間違っている場合)

注記

デフォルトでは、タイプ B 命令は 16 ビットの IMM フィールド値を取り込み、それを 32 ビットに符号拡張して即値オペランドとして使用します。この動作は、タイプ B 命令の前に imm 命令を先行させると変更できます。32 ビットの即値の使用については、「[imm](#)」を参照してください。

遅延スロットには、imm、分岐、またはブレーク命令を含めることはできません。割り込みおよび外部ハードウェアブレークは、遅延スロット分岐が完了するまで保留されます。

bge 0 以上の場合は分岐

bge	rA, rB	0以上の場合は分岐
bgd	rA, rB	0以上の場合は遅延後に分岐

説明

`rA` が 0 以上の場合、`rB` のオフセット値にある命令に分岐します。分岐先は、アドレス `PC + rB` にある命令です。

二モニック **bgd** は D ビットをセットします。D ビットは、分岐遅延スロットの有無を指定します。D ビットがセットされている場合は遅延スロットがあり、分岐先の命令を実行する前に、分岐に後続する命令(分岐遅延スロットにある命令)を実行できます。D ビットがセットされていない場合は、遅延スロットではなく、分岐後に実行される命令は分岐先の命令になります。

擬似コード

```

If rA >= 0 then
  PC ← PC + rB
else
  PC ← PC + 4
if D = 1 then
  allow following instruction to complete execution

```

変更されるレジスタ

- PC

レイテンシ

- 1 サイクル(分岐が取られなかった場合)
 - 2 サイクル(分岐が取られ、D ビットがセットされている場合)
 - 3 サイクル(分岐が取られ、D ビットがセットされていない場合)

注記

遅延スロットには、imm、分岐、またはブレーク命令を含めることはできません。割り込みおよび外部ハードウェアブレークは、遅延スロット分岐が完了するまで保留されます。

bgei 0以上の場合は即値に分岐

bgei	rA, IMM	0以上の場合は即値に分岐
bgeid	rA, IMM	0以上の場合は遅延後に即値に分岐

1	0	1	1	1	1	D	0	1	0	1	rA		IMM
0			6			11			16				31

説明

rA が 0 以上の場合、IMM のオフセット値にある命令に分岐します。分岐先は、アドレス PC + IMM にある命令です。二モニック bgeid は D ビットをセットします。D ビットは、分岐遅延スロットの有無を指定します。D ビットがセットされている場合は遅延スロットがあり、分岐先の命令を実行する前に、分岐に後続する命令(分岐遅延スロットにある命令)を実行できます。D ビットがセットされていない場合は、遅延スロットではなく、分岐後に実行される命令は分岐先の命令になります。

擬似コード

```
If rA >= 0 then
    PC ← PC + sext(IMM)
else
    PC ← PC + 4
if D = 1 then
    allow following instruction to complete execution
```

変更されるレジスタ

- PC

レイテンシ

- 1 サイクル(分岐が取られなかった場合、または正しく分岐予測された場合)
- 2 サイクル(分岐が取られ、D ビットがセットされている場合)
- 3 サイクル(分岐が取られ、D ビットがセットされていない場合、または C_AREA_OPTIMIZED=0 で分岐予測が間違っている場合)
- 7 ~ 9 サイクル(C_AREA_OPTIMIZED=2 で分岐予測が間違っている場合)

注記

デフォルトでは、タイプ B 命令は 16 ビットの IMM フィールド値を取り込み、それを 32 ビットに符号拡張して即値オペランドとして使用します。この動作は、タイプ B 命令の前に imm 命令を先行させると変更できます。32 ビットの即値の使用については、「[imm](#)」を参照してください。

遅延スロットには、imm、分岐、またはブレーク命令を含めることはできません。割り込みおよび外部ハードウェアブレークは、遅延スロット分岐が完了するまで保留されます。

bgt 0より大きい場合は分岐

bgt rA, rB 0より大きい場合は分岐
bgtd rA, rB 0より大きい場合は遅延後に分岐

説明

`rA` が 0 より大きければ、`rB` のオフセット値にある命令に分岐します。分岐先は、アドレス `PC + rB` にある命令です。ニーモニック `bgtd` は D ビットをセットします。D ビットは、分岐遅延スロットの有無を指定します。D ビットがセットされている場合は遅延スロットがあり、分岐先の命令を実行する前に、分岐に後続する命令（分岐遅延スロットにある命令）を実行できます。D ビットがセットされていない場合は、遅延スロットではなく、分岐後に実行される命令は分岐先の命令になります。

擬似コード

```

If rA > 0 then
  PC ← PC + rB
else
  PC ← PC + 4
if D = 1 then
  allow following instruction to complete execution

```

変更されるレジスタ

- PC

レイテンシ

- 1 サイクル(分岐が取られなかった場合)
 - 2 サイクル(分岐が取られ、D ビットがセットされている場合)
 - 3 サイクル(分岐が取られ、D ビットがセットされていない場合)

注記

遅延スロットには、imm、分岐、またはブレーク命令を含めることはできません。割り込みおよび外部ハードウェアブレークは、遅延スロット分岐が完了するまで保留されます。

bgti 0より大きい場合は即値に分岐

bgti	rA, IMM	0より大きい場合は即値に分岐
bgtid	rA, IMM	0より大きい場合は遅延後に即値に分岐

1	0	1	1	1	1	D	0	1	0	0	rA		IMM
0			6			11			16				31

説明

rAが0より大きければ、IMMのオフセット値にある命令に分岐します。分岐先は、アドレス PC + IMM にある命令です。

二モニック bgtid は D ビットをセットします。D ビットは、分岐遅延スロットの有無を指定します。D ビットがセットされている場合は遅延スロットがあり、分岐先の命令を実行する前に、分岐に後続する命令(分岐遅延スロットにある命令)を実行できます。D ビットがセットされていない場合は、遅延スロットではなく、分岐後に実行される命令は分岐先の命令になります。

擬似コード

```
If rA > 0 then
    PC ← PC + sext(IMM)
else
    PC ← PC + 4
if D = 1 then
    allow following instruction to complete execution
```

変更されるレジスタ

- PC

レイテンシ

- 1サイクル(分岐が取られなかった場合、または正しく分岐予測された場合)
- 2サイクル(分岐が取られ、Dビットがセットされている場合)
- 3サイクル(分岐が取られ、Dビットがセットされていない場合、またはC_AREA_OPTIMIZED=0で分岐予測が間違っている場合)
- 7~9サイクル(C_AREA_OPTIMIZED=2で分岐予測が間違っている場合)

注記

デフォルトでは、タイプB命令は16ビットのIMMフィールド値を取り込み、それを32ビットに符号拡張して即値オペランドとして使用します。この動作は、タイプB命令の前にimm命令を先行させると変更できます。32ビットの即値の使用については、「[imm](#)」を参照してください。

遅延スロットには、imm、分岐、またはブレーク命令を含めることはできません。割り込みおよび外部ハードウェアブレークは、遅延スロット分岐が完了するまで保留されます。

ble**0 以下の場合は分岐**

ble	rA, rB	0 以下の場合は分岐
bled	rA, rB	0 以下の場合は遅延後に分岐

1	0	0	1	1	1	D	0	0	1	1	rA	rB	0	0	0	0	0	0	0	0	0	0	
0			6			11			16			21										31	

説明

rA が 0 以下の場合、rB のオフセット値にある命令に分岐します。分岐先は、アドレス PC + rB にある命令です。

ニーモニック bled は D ビットをセットします。D ビットは、分岐遅延スロットの有無を指定します。D ビットがセットされている場合は遅延スロットがあり、分岐先の命令を実行する前に、分岐に後続する命令(分岐遅延スロットにある命令)を実行できます。D ビットがセットされていない場合は、遅延スロットはなく、分岐後に実行される命令は分岐先の命令になります。

擬似コード

```
If rA <= 0 then
    PC ← PC + rB
else
    PC ← PC + 4
if D = 1 then
    allow following instruction to complete execution
```

変更されるレジスタ

- PC

レイテンシ

- 1 サイクル(分岐が取られなかった場合)
- 2 サイクル(分岐が取られ、D ビットがセットされている場合)
- 3 サイクル(分岐が取られ、D ビットがセットされていない場合)

注記:

遅延スロットには、imm、分岐、またはブレーク命令を含めることはできません。割り込みおよび外部ハードウェアブレークは、遅延スロット分岐が完了するまで保留されます。

blei**0 以下の場合は即値に分岐**

blei	rA, IMM	0 以下の場合は即値に分岐
bleid	rA, IMM	0 以下の場合は遅延後に即値に分岐

1	0	1	1	1	1	D	0	0	1	1	rA		IMM
0			6			11			16				31

説明

rA が 0 以下の場合、IMM のオフセット値にある命令に分岐します。分岐先は、アドレス PC + IMM にある命令です。二モニック bleid は D ビットをセットします。D ビットは、分岐遅延スロットの有無を指定します。D ビットがセットされている場合は遅延スロットがあり、分岐先の命令を実行する前に、分岐に後続する命令(分岐遅延スロットにある命令)を実行できます。D ビットがセットされていない場合は、遅延スロットではなく、分岐後に実行される命令は分岐先の命令になります。

擬似コード

```
If rA <= 0 then
    PC ← PC + sext(IMM)
else
    PC ← PC + 4
if D = 1 then
    allow following instruction to complete execution
```

変更されるレジスタ

- PC

レイテンシ

- 1 サイクル(分岐が取られなかった場合、または正しく分岐予測された場合)
- 2 サイクル(分岐が取られ、D ビットがセットされている場合)
- 3 サイクル(分岐が取られ、D ビットがセットされていない場合、または C_AREA_OPTIMIZED=0 で分岐予測が間違っている場合)
- 7 ~ 9 サイクル(C_AREA_OPTIMIZED=2 で分岐予測が間違っている場合)

注記

デフォルトでは、タイプ B 命令は 16 ビットの IMM フィールド値を取り込み、それを 32 ビットに符号拡張して即値オペランドとして使用します。この動作は、タイプ B 命令の前に imm 命令を先行させると変更できます。32 ビットの即値の使用については、「[imm](#)」を参照してください。

遅延スロットには、imm、分岐、またはブレーク命令を含めることはできません。割り込みおよび外部ハードウェアブレークは、遅延スロット分岐が完了するまで保留されます。

bit 0より小さい場合は分岐

blt rA, rB 0より小さい場合は分岐
bltd rA, rB 0より小さい場合は遅延後に分岐

$$\begin{array}{cccccc|cccc|c|c|c|c} 1 & 0 & 0 & 1 & 1 & 1 & D & 0 & 0 & 1 & 0 & \text{rA} & & \text{rB} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & & & & & & 6 & & & & & 11 & & 16 & & 21 & & & & & & & 31 \end{array}$$

説明

rA が 0 より小さければ、rB のオフセット値にある命令に分岐します。分岐先は、アドレス PC+rB にある命令です。ニーモニック bltd は D ビットをセットします。D ビットは、分岐遅延スロットの有無を指定します。D ビットがセットされている場合は遅延スロットがあり、分岐先の命令を実行する前に、分岐に後続する命令(分岐遅延スロットにある命令)を実行できます。D ビットがセットされていない場合は、遅延スロットはなく、分岐後に実行される命令は分岐先の命令になります。

擬似コード

```

If rA < 0 then
  PC ← PC + rB
else
  PC ← PC + 4
if D = 1 then
  allow following instruction to complete execution

```

変更されるレジスタ

- PC

レイテンシ

- 1 サイクル(分岐が取られなかった場合)
 - 2 サイクル(分岐が取られ、D ビットがセットされている場合)
 - 3 サイクル(分岐が取られ、D ビットがセットされていない場合)

注記

遅延スロットには、imm、分岐、またはブレーク命令を含めることはできません。割り込みおよび外部ハードウェアブレークは、遅延スロット分岐が完了するまで保留されます。

blti 0より小さい場合は即値に分岐

blti	rA, IMM	0より小さい場合は即値に分岐
bltid	rA, IMM	0より小さい場合は遅延後に即値に分岐

1	0	1	1	1	1	D	0	0	1	0	rA		IMM
0			6			11			16				31

説明

rAが0より小さい場合は、IMMのオフセット値にある命令に分岐します。分岐先は、アドレス PC + IMM にある命令です。

二モニック bltid は D ビットをセットします。D ビットは、分岐遅延スロットの有無を指定します。D ビットがセットされている場合は遅延スロットがあり、分岐先の命令を実行する前に、分岐に後続する命令(分岐遅延スロットにある命令)を実行できます。D ビットがセットされていない場合は、遅延スロットはなく、分岐後に実行される命令は分岐先の命令になります。

擬似コード

```
If rA < 0 then
    PC ← PC + sext(IMM)
else
    PC ← PC + 4
if D = 1 then
    allow following instruction to complete execution
```

変更されるレジスタ

- PC

レイテンシ

- 1サイクル(分岐が取られなかった場合、または正しく分岐予測された場合)
- 2サイクル(分岐が取られ、Dビットがセットされている場合)
- 3サイクル(分岐が取られ、Dビットがセットされていない場合、またはC_AREA_OPTIMIZED=0で分岐予測が間違っている場合)
- 7~9サイクル(C_AREA_OPTIMIZED=2で分岐予測が間違っている場合)

注記

デフォルトでは、タイプB命令は16ビットのIMMフィールド値を取り込み、それを32ビットに符号拡張して即値オペランドとして使用します。この動作は、タイプB命令の前にimm命令を先行させると変更できます。32ビットの即値の使用については、「[imm](#)」を参照してください。

遅延スロットには、imm、分岐、またはブレーク命令を含めることはできません。割り込みおよび外部ハードウェアブレークは、遅延スロット分岐が完了するまで保留されます。

bne 0 でない場合は分岐

bne	rA, rB	0 でない場合は分岐
bned	rA, rB	0 でない場合は遅延後に分岐

説明

rA が 0 でない場合、 rB のオフセット値にある命令に分岐します。分岐先は、アドレス $PC + rB$ にある命令です。

二モニック **bned** は D ビットをセットします。D ビットは、分岐遅延スロットの有無を指定します。D ビットがセットされている場合は遅延スロットがあり、分岐先の命令を実行する前に、分岐に後続する命令(分岐遅延スロットにある命令)を実行できます。D ビットがセットされていない場合は、遅延スロットではなく、分岐後に実行される命令は分岐先の命令になります。

擬似コード

```

If rA ≠ 0 then
    PC ← PC + rB
else
    PC ← PC + 4
if D = 1 then
    allow following instruction to complete execution

```

変更されるレジスタ

- PC

レイテンシ

- 1 サイクル(分岐が取られなかった場合)
 - 2 サイクル(分岐が取られ、D ビットがセットされている場合)
 - 3 サイクル(分岐が取られ、D ビットがセットされていない場合)

注記

遅延スロットには、imm、分岐、またはブレーク命令を含めることはできません。割り込みおよび外部ハードウェアブレークは、遅延スロット分岐が完了するまで保留されます。

bnei 0でない場合は即値に分岐

bnei	rA, IMM	0でない場合は即値に分岐
bneid	rA, IMM	0でない場合は遅延後に即値に分岐

1	0	1	1	1	D	0	0	0	1	rA		IMM
0			6			11			16			31

説明

rA が 0 でない場合、IMM のオフセット値にある命令に分岐します。分岐先は、アドレス PC + IMM にある命令です。二モニック bneid は D ビットをセットします。D ビットは、分岐遅延スロットの有無を指定します。D ビットがセットされている場合は遅延スロットがあり、分岐先の命令を実行する前に、分岐に後続する命令(分岐遅延スロットにある命令)を実行できます。D ビットがセットされていない場合は、遅延スロットではなく、分岐後に実行される命令は分岐先の命令になります。

擬似コード

```
If rA ≠ 0 then
    PC ← PC + sext(IMM)
else
    PC ← PC + 4
if D = 1 then
    allow following instruction to complete execution
```

変更されるレジスタ

- PC

レイテンシ

- 1 サイクル(分岐が取られなかった場合、または正しく分岐予測された場合)
- 2 サイクル(分岐が取られ、D ビットがセットされている場合)
- 3 サイクル(分岐が取られ、D ビットがセットされていない場合、または C_AREA_OPTIMIZED=0 で分岐予測が間違っている場合)
- 7 ~ 9 サイクル(C_AREA_OPTIMIZED=2 で分岐予測が間違っている場合)

注記

デフォルトでは、タイプ B 命令は 16 ビットの IMM フィールド値を取り込み、それを 32 ビットに符号拡張して即値オペランドとして使用します。この動作は、タイプ B 命令の前に imm 命令を先行させると変更できます。32 ビットの即値の使用については、「[imm](#)」を参照してください。

遅延スロットには、imm、分岐、またはブレーク命令を含めることはできません。割り込みおよび外部ハードウェアブレークは、遅延スロット分岐が完了するまで保留されます。

br**無条件分岐**

br	rB	分岐
bra	rB	絶対分岐
brd	rB	遅延後に分岐
brad	rB	遅延後に絶対分岐
brld	rD, rB	遅延後に分岐およびリンク
brald	rD, rB	遅延後に絶対分岐およびリンク

1	0	0	1	1	0	rD	D	A	L	0	0	rB	0	0	0	0	0	0	0	0	0	0	
0						6				11			16				21						31

説明

rB で指定されたアドレスにある命令に分岐します。

ニーモニック brld および brald は L ビットをセットします。L ビットがセットされると、リンクが実行されます。PC の現在の値は rD に格納されます。

ニーモニック bra、brad、および brald は A ビットをセットします。A ビットがセットされると絶対値に分岐し、ターゲットは rB の値になります。それ以外の場合は相対分岐となり、ターゲットは PC + rB になります。

ニーモニック brd、brad、brld、および brald は D ビットをセットします。D ビットは、分岐遅延スロットの有無を指定します。D ビットがセットされている場合は遅延スロットがあり、分岐先の命令を実行する前に、分岐に後続する命令(分岐遅延スロットにある命令)を実行できます。

D ビットがセットされていない場合は、遅延スロットはなく、分岐後に実行される命令は分岐先の命令になります。

擬似コード

```

if L = 1 then
    (rD) ← PC
if A = 1 then
    PC ← (rB)
else
    PC ← PC + (rB)
if D = 1 then
    allow following instruction to complete execution

```

変更されるレジスタ

- rD
- PC

レイテンシ

- 2 サイクル(D ビットがセットされている場合)
- 3 サイクル(D ビットがセットされていない場合)

注記:

命令 brl および bral は使用できません。遅延スロットには、imm、分岐、またはブレーク命令を含めることはできません。割り込みおよび外部ハードウェアブレークは、遅延スロット分岐が完了するまで保留されます。

64 ビット モードでは、絶対分岐命令 bra、brad、および brald は 64 ビット レジスタ rB_L 全体、brald は 64 ビット レジスタ rD_L 全体を使用し、これらの命令は拡張アドレス分岐に使用できます。

bri**即値に無条件分岐**

bri	IMM	即値に分岐
brai	IMM	絶対即値に分岐
brid	IMM	遅延後に即値に分岐
braid	IMM	遅延後に絶対即値に分岐
brlid	rD, IMM	遅延後に即値に分岐およびリンク
bralid	rD, IMM	遅延後に絶対即値に分岐およびリンク

1	0	1	1	1	0	rD	D	A	L	0	0	IMM
0			6			11			16			31

説明

IMM(32ビットに符号拡張)で指定されたアドレスにある命令に分岐します。

ニーモニック brlid および bralid は、L ビットをセットします。L ビットがセットされると、リンクが実行されます。PC の現在の値は rD に格納されます。

ニーモニック brai、braid、および bralid は、A ビットをセットします。A ビットがセットされると絶対値に分岐し、ターゲットは IMM の値になります。それ以外の場合は相対分岐となり、ターゲットは PC + IMM になります。

ニーモニック brid、braid、brlid、および bralid は、D ビットをセットします。D ビットは、分岐遅延スロットの有無を指定します。D ビットがセットされている場合は遅延スロットがあり、分岐先の命令を実行する前に、分岐に後続する命令(分岐遅延スロットにある命令)を実行できます。D ビットがセットされていない場合は、遅延スロットはなく、分岐後に実行される命令は分岐先の命令になります。

特殊なケースとして、MicroBlaze が MMU を使用するように設定されていて(C_USE_MMU >= 1)、ユーザー ベクター例外を実行するために「bralid rD, C_BASE_VECTORS+0x8」を使用している場合、マシン ステータス レジスタ ビットのユーザー モードと仮想モードがクリアされます。

擬似コード

```

if L = 1 then
    (rD) ← PC
if A = 1 then
    PC ← sext(IMM)
else
    PC ← PC + sext(IMM)
if D = 1 then
    allow following instruction to complete execution
if D = 1 and A = 1 and L = 1 and IMM = C_BASE_VECTORS+0x8 then
    MSR[UMS] ← MSR[UM]
    MSR[VMS] ← MSR[VM]
    MSR[UM] ← 0
    MSR[VM] ← 0

```

変更されるレジスタ

- rD
- PC
- MSR[UM]、MSR[VM]

レイテンシ

- 1 サイクル(正しく分岐予測がされる場合)
- 2 サイクル(D ビットがセットされている場合)
- 3 サイクル(D ビットがセットされていない場合、または C_AREA_OPTIMIZED=0 で分岐予測が間違っている場合)
- 7 ~ 9 サイクル(C_AREA_OPTIMIZED=2 で分岐予測が間違っている場合)

注記

命令 brli および brali は使用できません。

デフォルトでは、タイプ B 命令は 16 ビットの IMM フィールド値を取り込み、それを符号拡張して即値オペランドとして使用します。この動作は、タイプ B 命令の前に imm 命令を先行させると変更できます。即値の使用については、「imm」を参照してください。

遅延スロットには、imm、分岐、またはブレーク命令を含めることはできません。割り込みおよび外部ハードウェアブレークは、遅延スロット分岐が完了するまで保留されます。

64 ビット モードでは、絶対分岐命令 brai、braid、bralid の前に imml 命令を先行させることもでき、bralid は 64 ビットレジスタ rD_L 全体を使用し、これらの命令は拡張アドレス分岐に使用できます。

brk ブレーク

brk rD, rB

$\begin{array}{cccccc} 1 & 0 & 0 & 1 & 1 & 0 \end{array}$	rD	$\begin{array}{cccccc} 0 & 1 & 1 & 0 & 0 \end{array}$	rB	$\begin{array}{cccccc} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{array}$
0	6	11	16	21

說明

rB のアドレス値にある命令に分岐およびリンクします。PC の現在の値は **rD** に格納されます。MSR の BIP フラグがセットされ、予約ビットがクリアされます。

MicroBlaze が MMU を使用するように設定されている場合 ($C_USE_MMU >= 1$)、この命令は特権命令となるので、命令がユーザー モード ($MSR[UM] = 1$) で実行されると特権命令例外が発生します。

擬似コード

```

if MSR[UM] = 1 then
  ESR[EC] ← 00111
else
  (rD) ← PC
  PC ← (rB)
  MSR[BIP] ← 1
  Reservation ← 0

```

変更されるレジスタ

- rD
 - PC
 - MSR[BIP]
 - ESR[EC] (特権命令例外が生成される場合)

レイテンシ

- ### • 3サイクル

注記

64 ビット モードでは、この命令は 64 ビット レジスタ rB_L および rD_L 全体を使用し、拡張アドレス分岐に使用できます。

brki**即値ブレーク**

brki rD, IMM

1	0	1	1	1	0	rD	0	1	1	0	0	IMM
0			6			11		16				31

説明

IMM(32ビットに符号拡張)のアドレス値にある命令に分岐およびリンクします。PCの現在の値はrDに格納されます。MSRのBIPフラグがセットされ(ソフトウェアブレークの場合を除く)、予約ビットがクリアされます。

MicroBlazeがMMUを使用するように設定されている場合($C_USE_MMU \geq 1$)、ソフトウェアブレークを実行するために「brki rD, C_BASE_VECTORS+0x8」または「brki rD, C_BASE_VECTORS+0x18」が使用されているとき以外はこの命令は特権命令となり、命令がユーザー モード($(MSR[UM] = 1)$ で実行されると特権命令例外が発生します。

特殊なケースとして、MicroBlazeがMMUを使用するように設定されていて($C_USE_MMU \geq 1$)、ソフトウェアブレークを実行するために「brki rD, C_BASE_VECTORS+0x8」または「brki rD, C_BASE_VECTORS+0x18」を使用している場合、マシンステータスレジスタビットのユーザー モードと仮想モードがクリアされます。

擬似コード

```
if MSR[UM] and IMM ≠ C_BASE_VECTORS+0x8 and IMM ≠ C_BASE_VECTORS+0x18 then
    ESR[EC] ← 00111
else
    (rD) ← PC
    PC ← sext(IMM)
    if IMM ≠ 0x18 then
        MSR[BIP] ← 1
    Reservation ← 0
    if IMM = C_BASE_VECTORS+0x8 or IMM = C_BASE_VECTORS+0x18 then
        MSR[UFS] ← MSR[UM]
        MSR[UM] ← 0
        MSR[VFS] ← MSR[VM]
        MSR[VM] ← 0
```

変更されるレジスタ

- rD(例外が生成されない場合。例外が生成されるとこのレジスタは変更されません)
- PC
- MSR[BIP]、MSR[UM]、MSR[VM]
- ESR[EC](特権命令例外が生成される場合)

レイテンシ

- 3サイクル

注記

デフォルトでは、タイプB命令は16ビットのIMMフィールド値を取り込み、それを符号拡張して即値オペランドとして使用します。この動作は、タイプB命令の前にimm命令を先行させると変更できます。即値の使用については、「imm」を参照してください。

特殊なケースとして、C_DEBUG_ENABLEDが0より大きい場合は、C_BASE_VECTORSの値に関係なく、imm命令の後にソフトウェアブレークを許可するためimm命令はソフトウェアブレーク「brki rD, 0x18」を無効にしません。

64ビットモードでは、この命令の前にimml命令を先行させることもでき、64ビットレジスタrDL全体を使用し、拡張アドレス分岐に使用できます。

bs バレルシフト

bsrl	rD, rA, rB	右論理バレルシフト
bsra	rD, rA, rB	右算術バレルシフト
bsll	rD, rA, rB	左論理バレルシフト

0	1	0	0	0	1	rD	rA	rB	S	T	0	0	0	0	0	0	0	0	0	0
0						6	11	16			21									31

説明

レジスタ rA の内容をレジスタ rB で指定した値分シフトし、その結果をレジスタ rD に出力します。

ニーモニック bsll は、S ビット (サイドビット) をセットします。S ビットがセットされている場合、左バレルシフトが実行されます。ニーモニック bsrl および bsra は、S ビットをクリアし、右シフトを実行します。

ニーモニック bsra は、T ビット (タイプビット) をセットします。T ビットがセットされている場合、算術バレルシフトが実行されます。ニーモニック bsrl および bsll は、T ビットをクリアし、論理シフトを実行します。

擬似コード

```

if S = 1 then
    (rD) ← (rA) << (rB) [27:31]
else
    if T = 1 then
        if ((rB) [27:31]) ≠ 0 then
            (rD) [0:(rB) [27:31]-1] ← (rA) [0]
            (rD) [(rB) [27:31]:31] ← (rA) >> (rB) [27:31]
        else
            (rD) ← (rA)
    else
        (rD) ← (rA) >> (rB) [27:31]

```

変更されるレジスタ

- rD

レイテンシ

- 1 サイクル (C_AREA_OPTIMIZED=0 または 2)
- 2 サイクル (C_AREA_OPTIMIZED=1)

注記:

これらの命令はオプションです。これらの命令を使用するには、MicroBlaze をバレルシフト命令が使用されるように設定する必要があります (C_USE_BARREL=1)。

bsi

即値バレルシフト

bsrli	rD, rA, IMM	右論理即値バレルシフト
bsrai	rD, rA, IMM	右算術即値バレルシフト
bslli	rD, rA, IMM	左論理即値バレルシフト
bsefi	rD, rA, IMM _W , IMM _S	即値バレルシフト抽出フィールド
bsifi	rD, rA, Width ¹ , IMM _S	即値バレルシフト挿入フィールド

1. Width = IMM_W - IMM_S + 1

0	1	1	0	0	1	rD	rA	0	0	0	0	0	S	T	0	0	0	0	IMM
0						6		11					16		21			27	31

0	1	1	0	0	1	rD	rA	I	E	0	0	0	IMM _W	0	IMM _S				
0						6		11					16		21		25	27	31

説明

最初の3つの命令は、レジスタ rA の内容を IMM で指定した値分シフトし、その結果をレジスタ rD に配置します。バレルシフト抽出フィールドは、レジスタ rA からビットフィールドを抽出し、その結果をレジスタ rD に配置します。ビットフィールドの幅は IMM_W で指定し、シフト量は IMM_S で指定します。ビットフィールドの幅は 1 ~ 31 の範囲で、 $IMM_W + IMM_S \leq 32$ という条件を満たす必要があります。

バレルシフト挿入フィールドは、レジスタ rA からのビットフィールドをレジスタ rD に挿入し、レジスタ rD の既存の値を変更します。ビットフィールドの幅は IMM_W - IMM_S + 1 で指定し、シフト量は IMM_S で指定します。 $IMM_W \geq IMM_S$ という条件を満たす必要があります。

ニーモニック bsrli は、S ビット(サイドビット)をセットします。S ビットがセットされている場合、左バレルシフトが実行されます。ニーモニック bsrai および bsrai は、S ビットをクリアし、右シフトを実行します。

ニーモニック bsrai は、T ビット(タイプビット)をセットします。T ビットがセットされている場合、算術バレルシフトが実行されます。ニーモニックの bsrli および bslli は T ビットをクリアし、論理シフトを実行します。

ニーモニック bsefi は、E ビット(抽出ビット)をセットします。この場合、S および T ビットは使用されません。

ニーモニック bsifi は、I ビット(挿入ビット)をセットします。この場合、S および T ビットは使用されません。

擬似コード

```

if E = 1 then
    (rD)[0:31-IMMW] ← 0
    (rD)[32-IMMW:31] ← (rA) >> IMMS
else if I = 1 then
    mask ← (0xffffffff << (IMMW + 1)) ⊕ (0xffffffff << IMMS)
    (rD) ← ((rA) << IMMS) ∧ mask) ∨ ((rD) ∧ mask)
else if S = 1 then
    (rD) ← (rA) << IMM
else if T = 1 then
    if IMM ≠ 0 then
        (rD)[0:IMM-1] ← (rA)[0]
        (rD)[IMM:31] ← (rA) >> IMM
    else
        (rD) ← (rA)
else
    (rD) ← (rA) >> IMM

```

変更されるレジスタ

- rD

レイテンシ

- 1 サイクル (C_AREA_OPTIMIZED=0 または 2)
- 2 サイクル (C_AREA_OPTIMIZED=1)

注記

これらはタイプ B 命令ではありません。先行する imm 命令からの影響はありません。

これらの命令はオプションです。これらの命令を使用するには、MicroBlaze をバレルシフト命令が使用されるように設定する必要があります (C_USE_BARREL=1)。

「bsifi rD, rA, width, shift」というアセンブラー コードは、IMM_W フィールドではなく、実際のビットフィールドの幅を表し、IMM_W=シフト+幅-1 で計算されます。

clz**先行ゼロをカウント**

clz rD, rA rA の先行ゼロをカウント

1	0	0	1	0	0	rD	rA	0	0	0	0	0	0	0	0	0	0	1	1	1	0	0	0	0	0
0			6			11		16									21								31

説明

レジスタ rA の先行ゼロの数を最上位ビットからカウントします。結果は 0 ~ 32 の値で、レジスタ rD に格納されます。rA が 0 の場合は rD の値は 32、rA が 0xFFFFFFFF の場合は 0 になります。

擬似コード

```
n ← 0
while (rA) [n] = 0
    n ← n + 1
(rD) ← n
```

変更されるレジスタ

- rD

レイテンシ

- 1 サイクル

注記:

この命令は C_USE_PCMP_INSTR を 1 に設定している場合のみ使用可能です。

cmp**整数比較**

cmp	rD, rA, rB	rB を rA と比較(符号付き)
cmpl	rD, rA, rB	rB を rA と比較(符号なし)

0	0	0	1	0	1	rD	rA	rB	0	0	0	0	0	0	0	0	U	1
0				6		11		16	21								31	

説明

レジスタ rA の内容をレジスタ rB の内容から減算し、その結果をレジスタ rD に配置します。

rD の MSB ビットに rA と rB の真の関係が示されます。U ビットがセットされている場合、rA および rB は符号なしの値とみなされます。U ビットがクリアされている場合、rA および rB は符号付きの値とみなされます。

擬似コード

```
(rD) ← (rB) + (rA) + 1
(rD)(MSB) ← (rA) > (rB)
```

変更されるレジスタ

- rD

レイテンシ

- 1 サイクル

fadd 浮動小数点算術加算

fadd rD, rA, rB 加算

0	1	0	1	1	0	rD	rA	rB	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0					6		11		16		21														31

説明

レジスタ rA と rB の浮動小数点和を算出し、レジスタ rD に配置します。

擬似コード

```

if isDnz(rA) or isDnz(rB) then
    (rD) ← 0xFFC00000
    FSR[DO] ← 1
    ESR[EC] ← 00110
else if isSigNaN(rA) or isSigNaN(rB) or
    (isPosInfinite(rA) and isNegInfinite(rB)) or
    (isNegInfinite(rA) and isPosInfinite(rB))) then
    (rD) ← 0xFFC00000
    FSR[IO] ← 1
    ESR[EC] ← 00110
else if isQuietNaN(rA) or isQuietNaN(rB) then
    (rD) ← 0xFFC00000
else if isDnz((rA)+(rB)) then
    (rD) ← signZero((rA)+(rB))
    FSR[UF] ← 1
    ESR[EC] ← 00110
else if isNaN((rA)+(rB)) then
    (rD) ← signInfinite((rA)+(rB))
    FSR[OF] ← 1
    ESR[EC] ← 00110
else
    (rD) ← (rA) + (rB)

```

変更されるレジスタ

- rD (FP例外が生成されない場合。例外が生成された場合は変更なし)
- ESR[EC] (FP例外が生成される場合)
- FSR[IO,UF,OF,DO]

レイテンシ

- 4 サイクル (C_AREA_OPTIMIZED=0)
- 6 サイクル (C_AREA_OPTIMIZED=1)
- 1 サイクル (C_AREA_OPTIMIZED=2)

注記:

この命令は、MicroBlaze のパラメーター C_USE_FPU が 0 より大きい場合のみ使用可能です。

frsub 浮動小数点算術逆減算

frsub rD, rA, rB 逆減算

0	1	0	1	1	0	rD	rA	rB	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
0					6		11		16			21													31

説明

レジスタ rA の浮動小数点値をレジスタ rB の浮動小数点値から減算し、その結果をレジスタ rD に配置します。

擬似コード

```
if isDnz(rA) or isDnz(rB) then
    (rD) ← 0xFFC00000
    FSR[DO] ← 1
    ESR[EC] ← 00110
else if (isSigNaN(rA) or isSigNaN(rB) or
        (isPosInfinite(rA) and isPosInfinite(rB)) or
        (isNegInfinite(rA) and isNegInfinite(rB))) then
    (rD) ← 0xFFC00000
    FSR[IO] ← 1
    ESR[EC] ← 00110
else if isQuietNaN(rA) or isQuietNaN(rB) then
    (rD) ← 0xFFC00000
else if isDnz((rB)-(rA)) then
    (rD) ← signZero((rB)-(rA))
    FSR[UF] ← 1
    ESR[EC] ← 00110
else if isNaN((rB)-(rA)) then
    (rD) ← signInfinite((rB)-(rA))
    FSR[OF] ← 1
    ESR[EC] ← 00110
else
    (rD) ← (rB) - (rA)
```

変更されるレジスタ

- rD (FP例外が生成されない場合。例外が生成された場合は変更なし)
- ESR[EC] (FP例外が生成される場合)
- FSR[IO,UF,OF,DO]

レイテンシ

- 4 サイクル (C_AREA_OPTIMIZED=0)
- 6 サイクル (C_AREA_OPTIMIZED=1)
- 1 サイクル (C_AREA_OPTIMIZED=2)

注記:

この命令は、MicroBlaze のパラメーター C_USE_FPU が 0 より大きい場合のみ使用可能です。

fmul 浮動小数点算術乗算

fmul rD, rA, rB 乗算

0	1	0	1	1	0	rD	rA	rB	0	0	1	0	0	0	0	0	0	0	0
0						6	11	16	21										31

説明

レジスタ rA の浮動小数点値とレジスタ rB の浮動小数点値を乗算し、その結果をレジスタ rD に配置します。

擬似コード

```

if isDnz(rA) or isDnz(rB) then
    (rD) ← 0xFFC00000
    FSR[DO] ← 1
    ESR[EC] ← 00110
else
    if isNaN(rA) or isNaN(rB) or (isZero(rA) and isInfinite(rB)) or
        (isZero(rB) and isInfinite(rA)) then
        (rD) ← 0xFFC00000
        FSR[IO] ← 1
        ESR[EC] ← 00110
    else if isQuietNaN(rA) or isQuietNaN(rB) then
        (rD) ← 0xFFC00000
    else if isDnz((rB)*(rA)) then
        (rD) ← signZero((rA)*(rB))
        FSR[UF] ← 1
        ESR[EC] ← 00110
    else if isNaN((rB)*(rA)) then
        (rD) ← signInfinite((rB)*(rA))
        FSR[OF] ← 1
        ESR[EC] ← 00110
    else
        (rD) ← (rB) * (rA)

```

変更されるレジスタ

- rD (FP例外が生成されない場合。例外が生成された場合は変更なし)
- ESR[EC] (FP例外が生成される場合)
- FSR[IO,UF,OF,DO]

レイテンシ

- 4 サイクル (C_AREA_OPTIMIZED=0)
- 6 サイクル (C_AREA_OPTIMIZED=1)
- 1 サイクル (C_AREA_OPTIMIZED=2)

注記:

この命令は、MicroBlaze のパラメーター C_USE_FPU が 0 より大きい場合のみ使用可能です。

fdiv

浮動小数点算術除算

fdiv rD, rA, rB 除算

0	1	0	1	1	0	rD	rA	rB	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0
0			6			11		16			21												31

説明

レジスタ rB の浮動小数点値をレジスタ rA の浮動小数点値で割り、その結果をレジスタ rD に配置します。

擬似コード

```
if isDnz(rA) or isDnz(rB) then
    (rD) ← 0xFFC00000
    FSR[DO] ← 1
    ESR[EC] ← 00110
else
    if isSigNaN(rA) or isSigNaN(rB) or (isZero(rA) and isZero(rB)) or
        (isInfinite(rA) and isInfinite(rB)) then
        (rD) ← 0xFFC00000
        FSR[IO] ← 1
        ESR[EC] ← 00110
    else if isQuietNaN(rA) or isQuietNaN(rB) then
        (rD) ← 0xFFC00000
    else if isZero(rA) and not isInfinite(rB) then
        (rD) ← signInfinite((rB) / (rA))
        FSR[DZ] ← 1
        ESR[EC] ← 00110
    else if isDnz((rB) / (rA)) then
        (rD) ← signZero((rB) / (rA))
        FSR[UF] ← 1
        ESR[EC] ← 00110
    else if isNaN((rB) / (rA)) then
        (rD) ← signInfinite((rB) / (rA))
        FSR[OF] ← 1
        ESR[EC] ← 00110
    else
        (rD) ← (rB) / (rA)
```

変更されるレジスタ

- rD (FP例外が生成されない場合。例外が生成された場合は変更なし)
- ESR[EC] (FP例外が生成される場合)
- FSR[IO,UF,OF,DO,DZ]

レイテンシ

- 28 サイクル (C_AREA_OPTIMIZED=0)
- 30 サイクル (C_AREA_OPTIMIZED=1)
- 24 サイクル (C_AREA_OPTIMIZED=2)

注記:

この命令は、MicroBlaze のパラメーター C_USE_FPU が 0 より大きい場合のみ使用可能です。

fcmp 浮動小数点値比較

fcmp.un	rD, rA, rB	浮動小数点値の比較(順不同)
fcmp.lt	rD, rA, rB	浮動小数点値の比較(小なり)
fcmp.eq	rD, rA, rB	浮動小数点値の比較(等価)
fcmp.le	rD, rA, rB	浮動小数点値の比較(以下)
fcmp.gt	rD, rA, rB	浮動小数点値の比較(大なり)
fcmp.ne	rD, rA, rB	浮動小数点値の比較(不等価)
fcmp.ge	rD, rA, rB	浮動小数点値の比較(以上)

0 1 0 1 1 0	rD	rA	rB	0 1 0 0	OpSel	0 0 0 0
0	6	11	16	21	25	28 31

説明

レジスタ rB の浮動小数点値をレジスタ rA の浮動小数点値と比較し、その結果をレジスタ rD に配置します。命令コードの OpSel フィールドは、実行される比較のタイプを指定します。

擬似コード

```
if isDnz(rA) or isDnz(rB) then
    (rD) ← 0
    FSR[DO] ← 1
    ESR[EC] ← 00110
else
    {read out behavior from 表 5-3}
```

変更されるレジスタ

- rD (FP例外が生成されない場合。例外が生成された場合は変更なし)
- ESR[EC] (FP例外が生成される場合)
- FSR[IO,DO]

レイテンシ

- 1 サイクル (C_AREA_OPTIMIZED=0 または 2)
- 3 サイクル (C_AREA_OPTIMIZED=1)

注記:

これらの命令は、MicroBlaze のパラメーター C_USE_FPU が 0 より大きい場合のみ使用可能です。

表 5-3 に、浮動小数点値の比較演算をリストします。

表 5-3: 浮動小数点比較演算

比較タイプ		オペランド関係				
説明	OpSel	(rB) > (rA)	(rB) < (rA)	(rB) = (rA)	isSigNaN(rA) または isSigNaN(rB)	isQuietNaN(rA) または isQuietNaN(rB)
順不同	000	(rD) ← 0	(rD) ← 0	(rD) ← 0	(rD) ← 1 FSR[IO] ← 1 ESR[EC] ← 00110	(rD) ← 1
小なり	001	(rD) ← 0	(rD) ← 1	(rD) ← 0	(rD) ← 0 FSR[IO] ← 1 ESR[EC] ← 00110	(rD) ← 0 FSR[IO] ← 1 ESR[EC] ← 00110
等価	010	(rD) ← 0	(rD) ← 0	(rD) ← 1	(rD) ← 0 FSR[IO] ← 1 ESR[EC] ← 00110	(rD) ← 0
以下	011	(rD) ← 0	(rD) ← 1	(rD) ← 1	(rD) ← 0 FSR[IO] ← 1 ESR[EC] ← 00110	(rD) ← 0 FSR[IO] ← 1 ESR[EC] ← 00110
大なり	100	(rD) ← 1	(rD) ← 0	(rD) ← 0	(rD) ← 0 FSR[IO] ← 1 ESR[EC] ← 00110	(rD) ← 0 FSR[IO] ← 1 ESR[EC] ← 00110
不等価	101	(rD) ← 1	(rD) ← 1	(rD) ← 0	(rD) ← 1 FSR[IO] ← 1 ESR[EC] ← 00110	(rD) ← 1
以上	110	(rD) ← 1	(rD) ← 0	(rD) ← 1	(rD) ← 0 FSR[IO] ← 1 ESR[EC] ← 00110	(rD) ← 0 FSR[IO] ← 1 ESR[EC] ← 00110

flt 整数値を浮動小数点値に変換

flt rD, rA

0	1	0	1	1	0	rD	rA	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0
0						6		11						16		21												31

説明

レジスタ rA の符号付き整数を浮動小数点に変換し、その結果をレジスタ rD に配置します。これは 32 ビットの丸め符号付き変換で、32 ビットの浮動小数点値を生成します。

擬似コード

```
(rD) ← float ((rA))
```

変更されるレジスタ

- rD

レイテンシ

- 4 サイクル (C_AREA_OPTIMIZED=0)
- 6 サイクル (C_AREA_OPTIMIZED=1)
- 1 サイクル (C_AREA_OPTIMIZED=2)

注記:

この命令は、MicroBlaze のパラメーター C_USE_FPU が 2 (拡張) の場合のみ使用可能です。

fint 浮動小数点値を整数値に変換

fint rD, rA

0	1	0	1	1	0	rD	rA	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	
0						6		11						16			21							31

説明

レジスタ rA の浮動小数点値を符号付き整数に変換し、その結果をレジスタ rD に配置します。これは 32 ビットの符号付き変換で、32 ビットの整数値を出力します。

擬似コード

```

if isDnz(rA) then
    (rD) ← 0xFFC00000
    FSR[DO] ← 1
    ESR[EC] ← 00110
else if isNaN(rA) then
    (rD) ← 0xFFC00000
    FSR[IO] ← 1
    ESR[EC] ← 00110
else if isInf(rA) or (rA) < -231 or (rA) > 231 - 1 then
    (rD) ← 0xFFC00000
    FSR[IO] ← 1
    ESR[EC] ← 00110
else
    (rD) ← int ((rA))

```

変更されるレジスタ

- rD (FP 例外が生成されない場合。例外が生成された場合は変更なし)
- ESR[EC] (FP 例外が生成される場合)
- FSR[IO,DO]

レイテンシ

- 5 サイクル (C_AREA_OPTIMIZED=0)
- 7 サイクル (C_AREA_OPTIMIZED=1)
- 2 サイクル (C_AREA_OPTIMIZED=2)

注記:

この命令は、MicroBlaze のパラメーター C_USE_FPU が 2 (拡張) の場合のみ使用可能です。

fsqrt 浮動小数点値の算術平方根

fsqrt rD, rA 平方根 (Sqrt)

0	1	0	1	1	0	rD	rA	0	0	0	0	0	0	0	1	1	1	0	0	0	0	0	0	0
0			6			11		16		21														31

説明

rA の値に対して浮動小数点の平方根を計算し、その結果を rD に配置します。

擬似コード

```

if isDnz(rA) then
    (rD) ← 0xFFC00000
    FSR[DO] ← 1
    ESR[EC] ← 00110
else if isSigNaN(rA) then
    (rD) ← 0xFFC00000
    FSR[IO] ← 1
    ESR[EC] ← 00110
else if isQuietNaN(rA) then
    (rD) ← 0xFFC00000
else if (rA) < 0 then
    (rD) ← 0xFFC00000
    FSR[IO] ← 1
    ESR[EC] ← 00110
else if (rA) = -0 then
    (rD) ← -0
else
    (rD) ← sqrt ((rA))

```

変更されるレジスタ

- rD (FP 例外が生成されない場合。例外が生成された場合は変更なし)
- ESR[EC] (FP 例外が生成される場合)
- FSR[IO,DO]

レイテンシ

- 27 サイクル (C_AREA_OPTIMIZED=0)
- 29 サイクル (C_AREA_OPTIMIZED=1)
- 23 サイクル (C_AREA_OPTIMIZED=2)

注記:

この命令は、MicroBlaze のパラメーター C_USE_FPU が 2 (拡張) の場合のみ使用可能です。

get ストリーム インターフェイスから取得

<i>tneaget</i>	rD、FSLx	リンク x からデータを取得 t: テストのみ n: ノンブロッキング e: 制御ビットがセットされている場合は例外 a: アトミック
<i>tneaget</i>	rD、FSLx	リンク x から制御を取得 t: テストのみ n: ノンブロッキング e: 制御ビットがセットされていない場合は例外 a: アトミック

0	1	1	0	1	1	rD	0	0	0	0	0	0	0	n	c	t	a	e	0	0	0	0	0	0	FSLx
---	---	---	---	---	---	----	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	------

0

6

11

16

28

31

説明

リンク x インターフェイスから読み出しを実行し、その結果をレジスタ rD に配置します。C_FSL_LINKS により設定されているリンク数が FSLx 以下の場合、リンク 0 が使用されます。

get 命令には 32 個のバリエーションがあります。

ブロッキング バージョン (n ビットが 0 のとき) は、インターフェイスからのデータが有効になるまで MicroBlaze をストールします。ノンブロッキング バージョンは、MicroBlaze をストールし、データが有効であればキャリーを 0 に、データが無効であればキャリーを 1 にセットします。アクセスが無効な場合は、デスティネーション レジスタの内容は未定義になります。

すべてのデータ get 命令 (c ビットが 0 のとき) は、インターフェイスからの制御ビットが 0 になるものとします。そうでない場合は、命令は MSR[FSL] を 1 にセットします。すべての制御 get 命令 (c ビットが 1 のとき) は、インターフェイスからの制御ビットが 1 になるものとします。そうでない場合は、命令は MSR[FSL] を 1 にセットします。

例外バージョン (e ビットが 1 のとき) は、制御ビットが一致しない場合に例外を生成します。この場合、ESR が、例外の原因に設定されている EC と、リンク インデックスに設定されている ESS でアップデートされます。ターゲット レジスタ rD は、例外が生成されるとアップデートされず、代わりにデータが EDR に格納されます。

テスト バージョン (t ビットが 1 のとき) は、リンクへの読み出し信号がアサートされていない場合を除き、標準ケースとして処理されます。

アトミック バージョン (a ビットが 1 のとき) は、割り込みできません。つまり、アトミック命令のシーケンスは、プログラム フローを割り込むことなく、グループにまとめることができます。ただし、それでも例外は発生する可能性があります。

MicroBlaze が MMU を使用するように設定されていて (C_USE_MMU >= 1)、明示的に C_MMU_PRIVILEGED_INSTR が 1 に設定されていない場合は、これらの命令は特権になります。つまり、命令がユーザー モード (MSR[UM]=1) で実行されると、特権命令例外が発生します。

擬似コード

```

if MSR[UM] = 1 then
    ESR[EC] ← 00111
else
    x ← FSLx
    if x >= C_FSL_LINKS then
        x ← 0
    (rD) ← Sx_AXIS_TDATA
    if (n = 1) then
        MSR[Carry] ← Sx_AXIS_TVALID
    if Sx_AXIS_TLAST ≠ c and Sx_AXIS_TVALID then
        MSR[FSL] ← 1
        if (e = 1) then
            ESR[EC] ← 00000
            ESR[ESS] ← instruction bits [28:31]
            EDR ← Sx_AXIS_TDATA

```

変更されるレジスタ

- rD (例外が生成されない場合。例外が生成されるとこのレジスタは変更されません)
- MSR[FSL]
- MSR[Carry]
- ESR[EC] (ストリーム例外または特権命令例外が生成される場合)
- ESR[ESS] (ストリーム例外が生成される場合)
- EDR (ストリーム例外が生成される場合)

レイテンシ

- 1 サイクル (C_AREA_OPTIMIZED=0 または 2)
- 2 サイクル (C_AREA_OPTIMIZED=1)

この命令のブロッキング バージョンは、命令が完了できるまで MicroBlaze のパイプラインをストールさせます。C_USE_EXTENDED_FSL_INSTR が 1 に設定されていて、命令がアトミックでない場合に、割り込みが実行されます。

注記

アセンブリ言語で FSLx インターフェイスを参照するには、rfsl0、rfsl1、... rfsl15 を使用してください。

この命令のブロッキング バージョンは、割り込みが実行されなくなるので、C_USE_EXTENDED_FSL_INSTR が 1 に設定されているときに遅延スロットに配置しないでください。

ノンブロッキング バージョンの場合は、インデックス変数をデクリメントするのに rsubc 命令を使用できます。

C_FSL_EXCEPTION が 1 でない限り、e ビットに効力はありません。

これらの命令は、MicroBlaze のパラメーター C_FSL_LINKS が 0 より大きい場合のみ使用可能です。

拡張命令 (例外、テスト、アトミック バージョン) は、MicroBlaze のパラメーター C_USE_EXTENDED_FSL_INSTR が 1 に設定されている場合にのみ使用可能です。

パフォーマンス上の理由からどうしてもこの命令をユーザー モードで実行しなければならない場合を除き、この命令をユーザー モードで実行しないようにしてください。リンクの誤用を防ぐためのハードウェア保護がすべて削除されてしまいます。

getd ストリーム インターフェイスから動的取得

<i>tneagetd</i>	rD, rB	リンク rB[28:31] からデータを取得 t: テストのみ n: ノンブロッキング e: 制御ビットがセットされている場合は例外 a: アトミック
<i>tneagetd</i>	rD, rB	リンク rB[28:31] から制御を取得 t: テストのみ n: ノンブロッキング e: 制御ビットがセットされていない場合は例外 a: アトミック

0	1	0	0	1	1	rD	0	0	0	0	0	rB	0	n	c	t	a	e	0	0	0	0	0	0
---	---	---	---	---	---	----	---	---	---	---	---	----	---	---	---	---	---	---	---	---	---	---	---	---

説明

MicroBlaze は、rB の最下位 4 ビットで定義されたインターフェイスから読み出しを実行し、レジスタ rD を結果に配置します。C_FSL_LINKS で設定されているリンク数が rB の最下位 4 ビット以下の場合、リンク 0 が使用されます。getd 命令には 32 個のバリエーションがあります。

ブロッキング バージョン (n ビットが 0 のとき) は、インターフェイスからのデータが有効になるまで MicroBlaze をストールします。ノンブロッキング バージョンは、MicroBlaze をストールし、データが有効であればキャリーを 0 に、データが無効であればキャリーを 1 にセットします。アクセスが無効な場合は、デスティネーション レジスタの内容は未定義になります。

すべてのデータ get 命令 (c ビットが 0 のとき) は、インターフェイスからの制御ビットが 0 になるものとします。そうでない場合は、命令は MSR[FSL] を 1 にセットします。すべての制御 get 命令 (c ビットが 1 のとき) は、インターフェイスからの制御ビットが 1 になるものとします。そうでない場合は、命令は MSR[FSL] を 1 にセットします。

例外バージョン (e ビットが 1 のとき) は、制御ビットが一致しない場合に例外を生成します。この場合、ESR が、例外の原因に設定されている EC と、リンク インデックスに設定されている ESS でアップデートされます。ターゲット レジスタ rD は、例外が生成されるとアップデートされず、代わりにデータが EDR に格納されます。

テスト バージョン (t ビットが 1 のとき) は、リンクへの読み出し信号がアサートされていない場合を除き、標準ケースとして処理されます。

アトミック バージョン (a ビットが 1 のとき) は、割り込みできません。つまり、アトミック命令のシーケンスは、プログラム フローを割り込むことなく、グループにまとめることができます。ただし、それでも例外は発生する可能性があります。

MicroBlaze が MMU を使用するように設定されていて (C_USE_MMU >= 1)、明示的に C_MMU_PRIVILEGED_INSTR が 1 に設定されていない場合は、これらの命令は特権になります。つまり、命令がユーザー モード (MSR[UM]=1) で実行されると、特権命令例外が発生します。

擬似コード

```

if MSR[UM] = 1 then
    ESR[EC] ← 00111
else
    x ← rB[28:31]
    if x >= C_FSL_LINKS then
        x ← 0
    (rD) ← Sx_AXIS_TDATA
    if (n = 1) then
        MSR[Carry] ← Sx_AXIS_TVALID
    if Sx_AXIS_TLAST ≠ c and Sx_AXIS_TVALID then
        MSR[FSL] ← 1
        if (e = 1) then
            ESR[EC] ← 00000
            ESR[ESS] ← rB[28:31]
            EDR ← Sx_AXIS_TDATA

```

変更されるレジスタ

- rD (例外が生成されない場合。例外が生成されるとこのレジスタは変更されません)
- MSR[FSL]
- MSR[Carry]
- ESR[EC] (ストリーム例外または特権命令例外が生成される場合)
- ESR[ESS] (ストリーム例外が生成される場合)
- EDR (ストリーム例外が生成される場合)

レイテンシ

- 1 サイクル (C_AREA_OPTIMIZED=0 または 2)
- 2 サイクル (C_AREA_OPTIMIZED=1)

この命令のブロッキング バージョンは、命令が完了できるまで MicroBlaze のパイプラインをストールさせます。命令がアトミックでなければ割り込みは実行されます。アトミックだと割り込みは実行されません。

注記

割り込みが実行されなくなってしまうので、この命令のブロッキング バージョンは、遅延スロットに配置しないでください。

ノンブロッキング バージョンの場合は、インデックス変数をデクリメントするのに rsubc 命令を使用できます。

C_FSL_EXCEPTION が 1 でない限り、e ビットに効力はありません。

MicroBlaze パラメーター C_FSL_LINKS が 0 より大きく、パラメーター C_USE_EXTENDED_FSL_INSTR が 1 に設定されている場合にのみ、これらの命令は使用可能です。

パフォーマンス上の理由からどうしてもこの命令をユーザー モードで実行しなければならない場合を除き、この命令をユーザー モードで実行しないようにしてください。リンクの誤用を防ぐためのハードウェア保護がすべて削除されてしまいます。

idiv**整数除算**

idiv	rD, rA, rB	rB を rA で割る (符号付き)
idivu	rD, rA, rB	rB を rA で割る (符号なし)

0	1	0	0	1	0	rD	rA	rB	0	0	0	0	0	0	0	U	0
0				6		11		16	21							31	

説明

レジスタ rB の内容をレジスタ rA の内容で割り、その結果がレジスタ rD に配置されます。

U ビットがセットされている場合、rA および rB は符号なしの値とみなされます。U ビットがクリアされている場合、rA および rB は符号付きの値とみなされます。

rA の値が 0 (ゼロによる除算) の場合、例外が生成されなければ、MSR の DZO ビットがセットされ、rD の値は 0 になります。

U ビットがクリアで、rA の値が -1 で、rB の値が -2147483648 (除算オーバーフロー) の場合、例外が生成されなければ、MSR の DZO ビットがセットされ、rD の値は -2147483648 になります。

擬似コード

```

if (rA) = 0 then
    (rD) <- 0
    MSR[DZO] <- 1
    ESR[EC] <- 00101
    ESR[DEC] <- 0
else if U = 0 and (rA) = -1 and (rB) = -2147483648 then
    (rD) <- -2147483648
    MSR[DZO] <- 1
    ESR[EC] <- 00101
    ESR[DEC] <- 1
else
    (rD) ← (rB) / (rA)

```

変更されるレジスタ

- rD (除算例外が生成されない場合。例外が生成されるとこのレジスタは変更されません)
- MSR[DZO] (ゼロによる除算または除算オーバーフローが発生した場合)
- ESR[EC] (ゼロによる除算または除算オーバーフローが発生した場合)

レイテンシ

- (rA)=0 の場合は 1 サイクル、それ以外の場合は 34 サイクル (C_AREA_OPTIMIZED=0)
- (rA)=0 の場合は 1 サイクル、それ以外の場合は 35 サイクル (C_AREA_OPTIMIZED=1)
- (rA)=0 の場合は 1 サイクル、それ以外の場合は 30 サイクル (C_AREA_OPTIMIZED=2)

注記:

MicroBlaze がハードウェア除算を使用するように設定されている場合にのみ (C_USE_DIV = 1)、この命令は有効です。

imm 即値

imm IMM

1 0 1 1 0 0	0 0 0 0 0	0 0 0 0 0	IMM
0	6	11	16

説明

imm 命令は、IMM 値を一時レジスタにロードします。また、その値を後続の命令で使用できるようにロックし、32 ビットの即値を形成します。

imm 命令は、タイプ B 命令と共に使用されます。タイプ B 命令には 16 ビットの即値フィールドしかないので、32 ビットの即値は直接使用できませんが、MicroBlaze では 32 ビットの即値を使用できます。デフォルトでは、タイプ B 命令は 16 ビットの IMM フィールド値を取り込み、それを 32 ビットに符号拡張して即値オペランドとして使用します。この動作は、タイプ B 命令の前に imm 命令を先行させると変更できます。imm 命令は、16 ビットの IMM 値を次の命令用に一時的にロックします。imm 命令の直後のタイプ B 命令は、imm 命令の 16 ビットの IMM 値(上位 16 ビット)とその命令自体の 16 ビットの即値フィールド(下位 16 ビット)から 32 ビットの即値を形成します。imm 命令の後にタイプ B 命令が続かない場合は、ロックされた値はロック解除され、無用になります。

レイテンシ

- 1 サイクル

注記

imm 命令とそれに続くタイプ B 命令はアトミックなので、この 2 つの間に割り込みは使用できません。

ザイリンクが提供するアセンブラーでは、imm 命令が必要かどうかが自動的に検出されます。32 ビットの IMM 値がタイプ B 命令で指定されると、命令をアセンブルするためアセンブラーにより IMM 値が 16 ビット値に変換され、実行ファイルのその値の前に imm 命令が挿入されます。

Ibu バイトをロード (符号なし)

lbu	rD _x , rA _x , rB _x
lbur	rD _x , rA _x , rB _x
lbuea	rD, rA, rB

1	1	0	0	0	0	rD _x	rA _x	rB _x	0	R	0	EA	0	0	0	0	0	0	0	
0						6	11	16	21											31

説明

レジスタ rA_X および rB_X の内容を加算した結果のメモリ ロケーションから 1 バイト (8 ビット) をロードします。データはレジスタ rD_X の最下位バイトに配置され、rD_X の残りのバイトはクリアされます。

R ビットがセットされている場合、バイト反転メモリ ロケーションを使用し、E ビットで定義されているエンディアンネスとは逆のエンディアンネスでデータをロードします (仮想保護モードがイネーブルの場合)。

EA ビットがセットされている場合、拡張アドレスが使用され、rA と rB を足すのではなく連結させて形成されます。

仮想保護モードがイネーブルの場合、データ TLB ミス例外が発生し、アドレスに対応する有効な変換エントリは TLB では検出されません。

ノーアクセス ゾーン保護によりアクセスが禁止されている場合は、データストレージ例外が発生します。これはユーザー モードおよび仮想保護モードがイネーブルになっているアクセスのみに適用されます。

EA ビットがセットされていて、物理アドレス拡張 (PAE) がイネーブルで、命令が明示的に許可されていない場合は、特権命令エラーが発生します。

擬似コード

```

if EA = 1 then
    Addr ← (rA) & (rB)
else
    Addr ← (rAx) + (rBx)
if TLB_Miss(Addr) and MSR[VM] = 1 then
    ESR[EC] ← 10010; ESR[S] ← 0
    MSR[UMS] ← MSR[UM]; MSR[VMS] ← MSR[VM]; MSR[UM] ← 0; MSR[VM] ← 0
else if Access_Protected(Addr) and MSR[UM] = 1 and MSR[VM] = 1 then
    ESR[EC] ← 10000; ESR[S] ← 0; ESR[DIZ] ← 1
    MSR[UMS] ← MSR[UM]; MSR[VMS] ← MSR[VM]; MSR[UM] ← 0; MSR[VM] ← 0
else
    (rDx) [C_DATA_SIZE-8:C_DATA_SIZE-1] ← Mem(Addr)
    (rDx) [0:C_DATA_SIZE-9] ← 0

```

変更されるレジスタ

- rD_X (例外が生成されない場合。例外が生成された場合は変更なし)
- MSR[UM]、MSR[VM]、MSR[Ums]、MSR[VMS] (例外が生成される場合)
- ESR[EC]、ESR[S] (例外が生成される場合)
- ESR[DIZ] (データストレージ例外が生成される場合)

レイテンシ

- 1 サイクル (C_AREA_OPTIMIZED=0 または 2)
- 2 サイクル (C_AREA_OPTIMIZED=1)

注記

MicroBlaze が再順序付け命令を使用するように設定されている場合にのみ (C_USE_REORDER_INSTR = 1)、このバイト反転命令は有効です。

拡張アドレス命令は、MicroBlaze が拡張アドレス命令を使用するよう設定されており (C_ADDR_SIZE > 32)、32 ビットモードを使用している場合 (C_DATA_SIZE = 32) にのみ有効です。

Ibui バイトをロード (符号なし即値)

Ibui rD_x, rA_x, IMM

1	1	1	0	0	0	rD _x	rA _x	IMM
0			6		11		16	31

説明

レジスタ rA_x の内容に IMM の符号拡張した値を加算した結果のメモリ ロケーションから、1 バイト (8 ビット) をロードします。データはレジスタ rD_x の最下位バイトに配置され、rD_x の残りのバイトはクリアされます。

仮想保護モードがイネーブルの場合、データ TLB ミス例外が発生し、アドレスに対応する有効な変換エンtriesは TLB では検出されません。

ノーアクセス ゾーン保護によりアクセスが禁止されている場合は、データストレージ例外が発生します。これはユーザー モードおよび仮想保護モードがイネーブルになっているアクセスのみに適用されます。

擬似コード

```

Addr ← (rAx) + sext(IMM)
if TLB_Miss(Addr) and MSR[VM] = 1 then
    ESR[EC] ← 10010; ESR[S] ← 0
    MSR[UMS] ← MSR[UM]; MSR[VMS] ← MSR[VM]; MSR[UM] ← 0; MSR[VM] ← 0
else if Access_Protected(Addr) and MSR[UM] = 1 and MSR[VM] = 1 then
    ESR[EC] ← 10000; ESR[S] ← 0; ESR[DIZ] ← 1
    MSR[UMS] ← MSR[UM]; MSR[VMS] ← MSR[VM]; MSR[UM] ← 0; MSR[VM] ← 0
else
    (rDx) [C_DATA_SIZE-8:C_DATA_SIZE-1] ← Mem(Addr)
    (rDx) [0:C_DATA_SIZE-9] ← 0

```

変更されるレジスタ

- rD_x (例外が生成されない場合。例外が生成された場合は変更なし)
- MSR[UM]、MSR[VM]、MSR[Ums]、MSR[VMS] (例外が生成される場合)
- ESR[EC]、ESR[S] (例外が生成される場合)
- ESR[DIZ] (データストレージ例外が生成される場合)

レイテンシ

- 1 サイクル (C_AREA_OPTIMIZED=0 または 2)
- 2 サイクル (C_AREA_OPTIMIZED=1)

注記:

デフォルトでは、タイプ B ロード命令は 16 ビットの IMM フィールド値を取り込み、それを符号拡張して即値オペランドとして使用します。この動作は、imm または imm1 命令を先行させると変更できます。即値の使用については、「imm」および「imm1」を参照してください。

Ihu ハーフワードをロード (符号なし)

lhu rD_X, rA_X, rB_X

lhur rD_X, rA_X, rB_X

lhuea rD, rA, rB

1	1	0	0	0	1	rD _X	rA _X	rB _X	0	R	0	EA	0	0	0	0	0	0	0	0	0	0	
0						6			11				16				21						31

説明

レジスタ rA_X および rB_X の内容を加算した、ハーフワードで境界整列された結果のメモリ ロケーションから 1 ハーフワード (16 ビット) をロードします。データはレジスタ rD_X の最下位ハーフワードに配置され、rD_X の残りのハーフワードはクリアされます。

R ビットがセットされている場合、ハーフワード反転メモリ ロケーションを使用し、ハーフワードの 2 バイトを反転して、E ビットで定義されているエンディアンネスとは逆のエンディアンネスでデータをロードします (仮想保護モードがイネーブルの場合)。

EA ビットがセットされている場合、拡張アドレスが使用され、rA と rB を足すのではなく連結させて形成されます。

仮想保護モードがイネーブルの場合、データ TLB ミス例外が発生し、アドレスに対応する有効な変換エントリは TLB では検出されません。

ノーアクセス ゾーン保護によりアクセスが禁止されている場合は、データストレージ例外が発生します。これはユーザー モードおよび仮想保護モードがイネーブルになっているアクセスのみに適用されます。

アドレスの最下位ビットがゼロでない場合は、非境界整列データアクセス例外が発生します。

EA ビットがセットされていて、物理アドレス拡張 (PAE) がイネーブルで、命令が明示的に許可されていない場合は、特権命令エラーが発生します。

擬似コード

```

if EA = 1 then
    Addr ← (rA) & (rB)
else
    Addr ← (rAx) + (rBx)
if TLB_Miss(Addr) and MSR[VM] = 1 then
    ESR[EC] ← 10010; ESR[S] ← 0
    MSR[UMS] ← MSR[UM]; MSR[VMS] ← MSR[VM]; MSR[UM] ← 0; MSR[VM] ← 0
else if Access_Protected(Addr) and MSR[UM] = 1 and MSR[VM] = 1 then
    ESR[EC] ← 10000; ESR[S] ← 0; ESR[DIZ] ← 1
    MSR[UMS] ← MSR[UM]; MSR[VMS] ← MSR[VM]; MSR[UM] ← 0; MSR[VM] ← 0
else if Addr[31] ≠ 0 then
    ESR[EC] ← 00001; ESR[W] ← 0; ESR[S] ← 0; ESR[Rx] ← rD
else if (VM = 0 and R = 1) or
    (VM = 1 and R = 1 and E = 1) or
    (VM = 1 and R = 0 and E = 0) then
    (rDx) [C_DATA_SIZE-16:C_DATA_SIZE-9] ← Mem(Addr);
    (rDx) [C_DATA_SIZE-8:C_DATA_SIZE-1] ← Mem(Addr+1);
    (rDx) [0:C_DATA_SIZE-17] ← 0
else
    (rDx) [C_DATA_SIZE-16:C_DATA_SIZE-9] ← Mem(Addr+1);
    (rDx) [C_DATA_SIZE-8:C_DATA_SIZE-1] ← Mem(Addr);
    (rDx) [0:C_DATA_SIZE-17] ← 0

```

変更されるレジスタ

- rD_X (例外が生成されない場合。例外が生成された場合は変更なし)
- MSR[UM]、MSR[VM]、MSR[UMS]、MSR[VMS] (例外が生成される場合)
- ESR[EC]、ESR[S] (例外が生成される場合)
- ESR[DIZ] (データストレージ例外が生成される場合)
- ESR[W]、ESR[Rx] (非境界整列データアクセス例外が生成される場合)

レイテンシ

- 1 サイクル (C_AREA_OPTIMIZED=0 または 2)
- 2 サイクル (C_AREA_OPTIMIZED=1)

注記

MicroBlaze が再順序付け命令を使用するように設定されている場合にのみ (C_USE_REORDER_INSTR = 1)、このハーフワード反転命令は有効です。

拡張アドレス命令は、MicroBlaze が拡張アドレス命令を使用するよう設定されており (C_ADDR_SIZE > 32)、32 ビットモードを使用している場合 (C_DATA_SIZE = 32) にのみ有効です。

Ihui ハーフワードをロード(符号なし即値)

Ihui rD_X, rA_X, IMM

1	1	1	0	0	1	rD _X	rA _X	IMM
0			6		11		16	31

説明

レジスタ rA_X の内容に IMM の符号拡張した値を加算した結果の、ハーフワードで境界整列されたメモリ ロケーションから、1 ハーフワード (16 ビット) をロードします。データはレジスタ rD_X の最下位ハーフワードに配置され、rD_X の残りのハーフワードはクリアされます。

仮想保護モードがイネーブルの場合、データ TLB ミス例外が発生し、アドレスに対応する有効な変換エントリは TLB では検出されません。ノーアクセスゾーン保護によりアクセスが禁止されている場合は、データストレージ例外が発生します。これはユーザー モードおよび仮想保護モードがイネーブルになっているアクセスのみに適用されます。アドレスの最下位ビットがゼロでない場合は、非境界整列データ アクセス例外が発生します。

擬似コード

```

Addr ← (rAX) + sext(IMM)
if TLB_Miss(Addr) and MSR[VM] = 1 then
    ESR[EC] ← 10010; ESR[S] ← 0
    MSR[UMS] ← MSR[UM]; MSR[VMS] ← MSR[VM]; MSR[UM] ← 0; MSR[VM] ← 0
else if Access_Protected(Addr) and MSR[UM] = 1 and MSR[VM] = 1 then
    ESR[EC] ← 10000; ESR[S] ← 0; ESR[DIZ] ← 1
    MSR[UMS] ← MSR[UM]; MSR[VMS] ← MSR[VM]; MSR[UM] ← 0; MSR[VM] ← 0
else if Addr[31] ≠ 0 then
    ESR[EC] ← 00001; ESR[W] ← 0; ESR[S] ← 0; ESR[Rx] ← rD
else
    (rDX) [C_DATA_SIZE-16:C_DATA_SIZE-1] ← Mem(Addr)
    (rDX) [0:C_DATA_SIZE-17] ← 0

```

変更されるレジスタ

- rD_X (例外が生成されない場合。例外が生成された場合は変更なし)
- MSR[UM]、MSR[VM]、MSR[Ums]、MSR[VMS] (TLB ミス例外またはデータストレージ例外が生成される場合)
- ESR[EC]、ESR[S] (例外が生成される場合)
- ESR[DIZ] (データストレージ例外が生成される場合)
- ESR[W]、ESR[Rx] (非境界整列データ アクセス例外が生成される場合)

レイテンシ

- 1 サイクル (C_AREA_OPTIMIZED=0 または 2)
- 2 サイクル (C_AREA_OPTIMIZED=1)

注記:

デフォルトでは、タイプ B ロード命令は 16 ビットの IMM フィールド値を取り込み、それを符号拡張して即値オペランドとして使用します。この動作は、imm または imm1 命令を先行させると変更できます。即値の使用については、「imm」および「imm1」を参照してください。

lw**ワードをロード**

lw	rD _X , rA _X , rB _X
lwr	rD _X , rA _X , rB _X
lwea	rD, rA, rB

1 1 0 0 1 0	rD _X	rA _X	rB _X	0 R 0 EA 0 0 0 0 0 0 0 0	
0	6	11	16	21	31

説明

レジスタ rA_X および rB_X の内容を加算した結果の、ワードで境界整列されたメモリ ロケーションから、1 ワード (32 ビット) をロードします。データはレジスタ rD_X の最下位ワードに配置され、最上位ワードがある場合はクリアされます。

R ビットがセットされている場合、ロードしたワードのバイトを反転し、E ビットで定義されているエンディアンネスとは逆のエンディアンネスでデータをロードします (仮想保護モードがイネーブルの場合)。

EA ビットがセットされている場合、拡張アドレスが使用され、rA と rB を足すのではなく連結させて形成されます。

仮想保護モードがイネーブルの場合、データ TLB ミス例外が発生し、アドレスに対応する有効な変換エントリは TLB では検出されません。

ノーアクセス ゾーン保護によりアクセスが禁止されている場合は、データストレージ例外が発生します。これはユーザー モードおよび仮想保護モードがイネーブルになっているアクセスのみに適用されます。

アドレスの最下位 2 ビットがゼロでない場合は、非境界整列データ アクセス例外が発生します。

EA ビットがセットされていて、物理アドレス拡張 (PAE) がイネーブルで、命令が明示的に許可されていない場合は、特権命令エラーが発生します。

擬似コード

```

if EA = 1 then
    Addr ← (rA) & (rB)
else
    Addr ← (rAX) + (rBX)
if TLB_Miss(Addr) and MSR[VM] = 1 then
    ESR[EC] ← 10010; ESR[S] ← 0
    MSR[UFS] ← MSR[UM]; MSR[VMS] ← MSR[VM]; MSR[UM] ← 0; MSR[VM] ← 0
else if Access_Protected(Addr) and MSR[UM] = 1 and MSR[VM] = 1 then
    ESR[EC] ← 10000; ESR[S] ← 0; ESR[DIZ] ← 1
    MSR[UFS] ← MSR[UM]; MSR[VMS] ← MSR[VM]; MSR[UM] ← 0; MSR[VM] ← 0
else if Addr[30:31] ≠ 0 then
    ESR[EC] ← 00001; ESR[W] ← 1; ESR[S] ← 0; ESR[Rx] ← rD
else
    (rDX[C_DATA_SIZE-32:C_DATA_SIZE-1]) ← Mem(Addr)
    (rDX[0:C_DATA_SIZE-33]) ← 0

```

変更されるレジスタ

- rD_X (例外が生成されない場合。例外が生成された場合は変更なし)
- MSR[UM]、MSR[VM]、MSR[UMS]、MSR[VMS] (TLB ミス例外またはデータストレージ例外が生成される場合)
- ESR[EC]、ESR[S] (例外が生成される場合)
- ESR[DIZ] (データストレージ例外が生成される場合)
- ESR[W]、ESR[Rx] (非境界整列データアクセス例外が生成される場合)

レイテンシ

- 1 サイクル (C_AREA_OPTIMIZED=0 または 2)
- 2 サイクル (C_AREA_OPTIMIZED=1)

注記

MicroBlaze が再順序付け命令を使用するように設定されている場合にのみ (C_USE_REORDER_INSTR = 1)、このワード反転命令は有効です。

拡張アドレス命令は、MicroBlaze が拡張アドレス命令を使用するよう設定されており (C_ADDR_SIZE > 32)、32 ビットモードを使用している場合 (C_DATA_SIZE = 32) にのみ有効です。

lwi**ワードをロード (即値)**lwi rD_X, rA_X, IMM

1	1	1	0	1	0	rD _X	rA _X	IMM
0			6		11		16	31

説明

レジスタ rA_X の内容に IMM の符号拡張した値を加算した結果の、ワードで境界整列されたメモリロケーションから、1 ワード (32 ビット) をロードします。データはレジスタ rD_X の最下位ワードに配置され、最上位ワードがある場合はクリアされます。

仮想保護モードがイネーブルの場合、データ TLB ミス例外が発生し、アドレスに対応する有効な変換エントリは TLB では検出されません。

ノーアクセスゾーン保護によりアクセスが禁止されている場合は、データストレージ例外が発生します。これはユーザー モードおよび仮想保護モードがイネーブルになっているアクセスのみに適用されます。

アドレスの最下位 2 ビットがゼロでない場合は、非境界整列データアクセス例外が発生します。

擬似コード

```

Addr ← (rAX) + sext(IMM)
if TLB_Miss(Addr) and MSR[VM] = 1 then
    ESR[EC] ← 10010; ESR[S] ← 0
    MSR[UMS] ← MSR[UM]; MSR[VMS] ← MSR[VM]; MSR[UM] ← 0; MSR[VM] ← 0
else if Access_Protected(Addr) and MSR[UM] = 1 and MSR[VM] = 1 then
    ESR[EC] ← 10000; ESR[S] ← 0; ESR[DIZ] ← 1
    MSR[UMS] ← MSR[UM]; MSR[VMS] ← MSR[VM]; MSR[UM] ← 0; MSR[VM] ← 0
else if Addr[30:31] ≠ 0 then
    ESR[EC] ← 00001; ESR[W] ← 1; ESR[S] ← 0; ESR[Rx] ← rD
else
    (rDX[C_DATA_SIZE-32:C_DATA_SIZE-1]) ← Mem(Addr); (rDX[0:C_DATA_SIZE-33]) ← 0

```

変更されるレジスタ

- rD_X (例外が生成されない場合。例外が生成された場合は変更なし)
- MSR[UM]、MSR[VM]、MSR[Ums]、MSR[VMS] (TLB ミス例外またはデータストレージ例外が生成される場合)
- ESR[EC]、ESR[S] (例外が生成される場合)
- ESR[DIZ] (データストレージ例外が生成される場合)
- ESR[W]、ESR[Rx] (非境界整列データアクセス例外が生成される場合)

レイテンシ

- 1 サイクル (C_AREA_OPTIMIZED=0 または 2)
- 2 サイクル (C_AREA_OPTIMIZED=1)

注記:

デフォルトでは、タイプ B ロード命令は 16 ビットの IMM フィールド値を取り込み、それを符号拡張して即値オペランドとして使用します。この動作は、imm または imml 命令を先行させると変更できます。即値の使用については、「imm」および「imml」を参照してください。

lwx ワードをロード(排他的)

lwx rD, rA, rB

1	1	0	0	1	0	rD	rA	rB	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0						6	11	16	21															31

説明

レジスタ rA および rB の内容を加算した結果の、ワードで境界整列されたメモリ ロケーションから 1 ワード (32 ビット) をロードします。データはレジスタ rD に配置され、予約ビットがセットされます。排他的アクセスがイネーブルになっている AXI4 インターコネクトが使用され、インターフェース応答が EXOKAY でない場合、キャリー フラグ (MSR[C]) がセットされます。そうでない場合はキャリー フラグはクリアになります。

仮想保護モードがイネーブルの場合、データ TLB ミス例外が発生し、アドレスに対応する有効な変換エントリは TLB では検出されません。

ノーアクセス ゾーン保護によりアクセスが禁止されている場合は、データストレージ例外が発生します。これはユーザー モードおよび仮想保護モードがイネーブルになっているアクセスのみに適用されます。

アドレスの最下位 2 ビットがゼロでない場合は、非境界整列データ アクセス例外は発生しません。

排他的アクセスがイネーブルになっている AXI4 インターコネクトが使用され、インターフェース応答が EXOKAY でないと(つまり排他的アクセスが処理できない)、データバス例外が発生する可能性があります。

AXI の排他的アクセスをイネーブルにすると、ほかのバスマスターから操作が保護されますが、アドレス指定したスレーブで排他的アクセスをサポートするようにしておく必要があります。排他的アクセスがイネーブルになつていないと、内部予約ビットのみが使用されます。排他的アクセスは、ペリフェラルおよびキャッシュインターフェースにそれぞれ、C_M_AXI_DP_EXCLUSIVE_ACCESS および C_M_AXI_DC_EXCLUSIVE_ACCESS という 2 つのパラメーターを設定するとイネーブルになります。

擬似コード

```

Addr ← (rA) + (rB)
if TLB_Miss(Addr) and MSR[VM] = 1 then
    ESR[EC] ← 10010; ESR[S] ← 0
    MSR[UFS] ← MSR[UM]; MSR[VFS] ← MSR[VM]; MSR[UM] ← 0; MSR[VM] ← 0
else if Access_Protected(Addr) and MSR[UM] = 1 and MSR[VM] = 1 then
    ESR[EC] ← 10000; ESR[S] ← 0; ESR[DIZ] ← 1
    MSR[UFS] ← MSR[UM]; MSR[VFS] ← MSR[VM]; MSR[UM] ← 0; MSR[VM] ← 0
else if AXI_Exclusive(Addr) and AXI_Response ≠ EXOKAY and MSR[EE] then
    ESR[EC] ← 00100; ESR[ECC] ← 0;
    MSR[UFS] ← MSR[UM]; MSR[VFS] ← MSR[VM]; MSR[UM] ← 0; MSR[VM] ← 0
else
    (rD) ← Mem(Addr); Reservation ← 1;
    if AXI_Exclusive(Addr) and AXI_Response ≠ EXOKAY then
        MSR[C] ← 1
    else
        MSR[C] ← 0

```

変更されるレジスタ

- rD および MSR[C] (例外が生成されない場合。生成されるとこの 2 つは変更されない)
- MSR[UM]、MSR[VM]、MSR[UMS]、MSR[VMS] (TLB ミス例外またはデータストレージ例外が生成される場合)
- ESR[EC]、ESR[S] (例外が生成される場合)
- ESR[DIZ] (データストレージ例外が生成される場合)

レイテンシ

- 1 サイクル (C_AREA_OPTIMIZED=0 または 2)
- 2 サイクル (C_AREA_OPTIMIZED=1)

注記

この命令は、セマフォやスピントロックなど、排他的アクセスをインプリメントするために、SWXと一緒に使用されます。

キャリー フラグ (MSR[C]) はすぐにはセットされない可能性があります (パイプラインストールの動作による)。キャリー フラグの正しい値を得るために、LWX 命令は、MSRCLR、MSRSET、MTS、または SRC 命令のすぐ後に続けることはできません。

mbar メモリバリア

mbar IMM メモリバリア

1 0 1 1 1 0	IMM	0 0 0 1 0	0 1 0 0
0	6	11	16

説明

この命令は、メモリインターフェイスの未処理のメモリアクセスが、後に続く命令が実行される前に完了するようになります。これは、自己変更コードが正しく処理されるようにするために、そしてDMA転送が問題なく開始できるようにするために必要です。

自己変更コードの場合、まず、データアクセスがあるまで待機するため、IMMを1に設定し、MBAR命令を使用する必要があります。次に、分岐先キャッシュをクリアにし、命令プリフェッчバッファーを空にするために、IMMを2に設定し、別のMBAR命令を使用する必要があります。

DMAユニットによって読み出されるデータがメモリに書き込まれたことを確認するには、IMMを1に設定し、データアクセスがあるまで待機するだけです。

MicroBlazeをMMUを使用するように設定すると($C_USE_MMU \geq 1$)、IMMの最上位ビットが1にセットされている場合、この命令は特権命令になります。つまり、命令がユーザー モード($MSR[UM] = 1$)で実行されると、特権命令例外が発生します。

IMMの最上位2ビットが10(Sleep)、01(Hibernate)、または11(Suspend)に設定されており、例外が発生しない場合、未処理のアクセスがすべて完了した後にMicroBlazeはスリープモードに入り、Sleep、Hibernate、またはSuspend出力信号がそれぞれセットされます。パイプラインは停止し、Wakeup入力信号の1ビットがアサートされるまで、MicroBlazeは実行を再開しません。

擬似コード

```

if (IMM & 1) = 0 then
    wait for instruction side memory accesses
if (IMM & 2) = 0 then
    wait for data side memory accesses
PC ← PC + 4
if (IMM & 24) != 0 then
    enter sleep mode

```

変更されるレジスタ

- PC
- ESR[EC] (特権命令例外が生成される場合)

レイテンシ

- $C_INTERCONNECT = 2$ (AXI) の場合 $2 + N$ サイクル
- $C_INTERCONNECT = 3$ (ACE) の場合 $8 + N$ サイクル
(Nはメモリアクセスが完了するのを待機するサイクル数)

注記

この命令の前にimm命令を先行させることはできません。また遅延スロットに配置することもできません。

スリープモードに入るには、mbar 16、mbar 8、mbar 24の代わりに、アセンブラー疑似命令のsleep、hibernate、suspendを使用できます。

mfs 特殊レジスタから移動

mfs rD、rS
mfse rD、rS

1	0	0	1	0	1	rD	0	E	0	0	0	1	0	rS
0			6			11			16		18			31

説明

特殊レジスタ rS の内容をレジスタ rD にコピーします。特殊レジスタ TLBLO および TLBHI は、TLBX によりインデックス化された統合 TLB エントリの内容をコピーするのに使用されます。

E ビットがセットされている場合、特殊レジスタの拡張部分は移動されます。拡張アドレス指定がイネーブルになっていると、EAR、PVR[8] および PVR[9] レジスタには拡張部分が含まれます ($C_ADDR_SIZE > 32$)。物理アドレス拡張 (PAE) がイネーブルの場合、TLBLO、PVR[6] および PVR[7] レジスタに拡張部分が含まれます。

擬似コード

```
if E = 1 then
    switch (rS):
        case 0x0003 : (rD) ← EAR[0:C_ADDR_SIZE-32-1]
        case 0x1003 : (rD) ← TLBLO[0:C_ADDR_SIZE-32-1]
        case 0x2006 : (rD) ← PVR6[0:C_ADDR_SIZE-32-1]
        case 0x2007 : (rD) ← PVR7[0:C_ADDR_SIZE-32-1]
        case 0x2008 : (rD) ← PVR8[0:C_ADDR_SIZE-32-1]
        case 0x2009 : (rD) ← PVR9[0:C_ADDR_SIZE-32-1]
        default : (rD) ← Undefined
else
    switch (rS):
        case 0x0000 : (rD) ← PC
        case 0x0001 : (rD) ← MSR
        case 0x0003 : (rD) ← EAR[C_ADDR_SIZE-32:C_ADDR_SIZE-1]
        case 0x0005 : (rD) ← ESR
        case 0x0007 : (rD) ← FSR
        case 0x000B : (rD) ← BTR
        case 0x000D : (rD) ← EDR
        case 0x0800 : (rD) ← SLR
        case 0x0802 : (rD) ← SHR
        case 0x1000 : (rD) ← PID
        case 0x1001 : (rD) ← ZPR
        case 0x1002 : (rD) ← TLBX
        case 0x1003 : (rD) ← TLBLO[C_ADDR_SIZE-32:C_ADDR_SIZE-1]
        case 0x1004 : (rD) ← TLBHI
        case 0x200x : (rD) ← PVRx[C_ADDR_SIZE-32:C_ADDR_SIZE-1] (where x = 0 to 12)
        default : (rD) ← Undefined
```

変更されるレジスタ

- rD

レイテンシ

- 1 サイクル

注記

アセンブリ言語で特殊レジスタを参照するには、PC には rpc、MSR には rmsr、EAR には rear、ESR には resr、FSR には rfsr、BTR には rbtr、EDR には redr、SLR には rslr、SHR には rshr、PID には rpid、ZPR には rzpr、TLBLO には rtlblo、TLBHI には rtlbhi、TLBX には rtlbx、PVR0 から PVR12 には rpvr0 から rpvr12 を使用します。

MSR から読み出された値には、そのすぐ前に先行する命令の効果を含めることができない場合があります(パイプラインストールの動作による)。MSR に影響しない命令は、正しい MSR 値を確約するため、MFS 命令より先行させる必要があります。

FSR から読み出された値には、そのすぐ前に先行する命令の効果を含めることができない場合があります(パイプラインストールの動作による)。FSR に影響しない命令は、正しい FSR 値を確約するため、MFS 命令より先行させる必要があります。

MicroBlaze のパラメーター C_*_EXCEPTION が少なくとも 1 つ 1 に設定されていると、EAR、ESR、および BTR はオペランドとしてのみ有効です。

C_FSL_EXCEPTION が 1 に設定されていて、C_FSL_LINKS が 0 よりも大きな値に設定されていると、EDR はオペランドとしてのみ有効です。

C_USE_FPU が 0 よりも大きな値に設定されていると、FSR はオペランドとしてのみ有効です。

C_USE_STACK_PROTECTION が 1 に設定されていると、SLR および SHR はオペランドとしてのみ有効です。

PID、ZPR、TLBLO、および TLBHI は、C_USE_MMU > 1 (ユーザー モード) および C_MMU_TLB_ACCESS = 1 (読み出し) または 3 (フル) の場合にのみオペランドとして有効です。

C_USE_MMU > 1 (ユーザー モード) および C_MMU_TLB_ACCESS > 0 (最小) に設定されていると、TLBX はオペランドとしてのみ有効です。

C_PVR が 1 (基本) または 2 (フル) に設定されていると、PVR0 はオペランドとしてのみ有効です。また、C_PVR が 2 (フル) に設定されていると、PVR1 から PVR12 はオペランドとしてのみ有効です。

MicroBlaze が拡張命令を使用するように設定されている場合にのみ (C_ADDR_SIZE > 32)、拡張アドレス命令は有効です。

msrclr MSR を読み出し MSR のビットをクリア

msrclr rD、Imm

1 0 0 1 0 1	rD	1 0 0 0 1 0	Imm15
0	6	11	17

説明

特殊レジスタ MSR の内容をレジスタ rD にコピーします。IMM 値でビット位置が 1 のものは MSR でクリアになります。IMM 値でビット位置が 0 のものはそのままになります。

MicroBlaze が MMU を使用するように設定されている場合 (C_USE_MMU >= 1)、この命令は、C のみに影響を与えるものを除き、すべての IMM 値に対して特権命令になります。命令がユーザー モード (MSR[UM] = 1) で実行されると特権命令例外が発生します。

擬似コード

```
if MSR[UM] = 1 and IMM ≠ 0x4 then
    ESR[EC] ← 00111
else
    (rD) ← (MSR)
    (MSR) ← (MSR) ∧ (IMM)
```

変更されるレジスタ

- rD
- MSR
- ESR[EC] (特権命令例外が生成される場合)

レイテンシ

- 1 サイクル

注記

MSRCLR はすぐにキャリービットに影響しますが、残りのビットは、命令実行後 1 クロックして効力を持ちます。IE ビットをクリアにすると、後続の命令に対する割り込みにはプロセッサは反応しなくなります。

MSR から読み出された値には、そのすぐ前に先行する命令の効果を含めることができない場合があります(パイプラインストールの動作による)。MSR に影響しない命令は、正しい MSR 値を確約するため、MSRCLR 命令より先行させる必要があります。これは、レジスタ rD にコピーされた値および変更された MSR 値自体の両方に適用されます。

即値は、C_USE_MMU >= 1 (ユーザー モード) のとき 215 未満で、それ以外のときは 214 未満になる必要があります。C_USE_MMU >= 1 (ユーザー モード) のとき、MSR のビット 17 から 31 までののみがクリアになり、それ以外のときはビット 18 から 31 までがクリアになります。

この命令は C_USE_MSR_INSTR を 1 に設定している場合のみ使用可能です。

MSR[VM] をクリアするとき、この命令の後には、常に、BRI 4 などの同期分岐命令が続く必要があります。

msrset MSR を読み出し MSR のビットをセット

msrset rD、Imm

1 0 0 1 0 1	rD	1 0 0 0 0 0	Imm15
0	6	11	17

説明

特殊レジスタ MSR の内容をレジスタ rD にコピーします。IMM 値でビット位置が 1 のものは MSR でセットされます。IMM 値でビット位置が 0 のものはそのままになります。

MicroBlaze が MMU を使用するように設定されている場合 ($C_USE_MMU >= 1$)、この命令は、C のみに影響を与えるものを除き、すべての IMM 値に対して特権命令になります。すなわち、この場合に命令がユーザー モード (MSR[UM] = 1) で実行されると、特権命令例外が発生します。

低レイテンシ割り込みモードでは ($C_USE_INTERRUPT = 2$)、この命令を実行して MSR{IE} ビットがセットされる場合に、Interrupt_Ack 出力ポートが 1 に設定されます。

擬似コード

```

if MSR[UM] = 1 and IMM ≠ 0x4 then
    ESR[EC] ← 00111
else
    (rD) ← (MSR)
    (MSR) ← (MSR) ∨ (IMM)
    if (IMM) & 2
        Interrupt_Ack ← 11

```

変更されるレジスタ

- rD
- MSR
- ESR[EC] (特権命令例外が生成される場合)

レイテンシ

- 1 サイクル

注記

MSRSET はすぐにキャリービットに影響しますが、残りのビットは、命令実行後 1 クロックして効力を持ります。EIP または BIP ビットをセットすると、後続の命令に対する割り込みまたは標準ハードウェアブレークにはプロセッサは反応しなくなります。

MSR から読み出された値には、そのすぐ前に先行する命令の効果を含めることができない場合があります(パイプラインストールの動作による)。MSR に影響しない命令は、正しい MSR 値を確約するため、MSRSET 命令より先行させる必要があります。これは、レジスタ rD にコピーされた値および変更された MSR 値自体の両方に適用されます。

即値は、 $C_USE_MMU >= 1$ (ユーザー モード)のとき 215 未満で、それ以外のときは 214 未満になる必要があります。 $C_USE_MMU >= 1$ (ユーザー モード)のとき、MSR のビット 17 から 31 までのみがクリアになり、それ以外のときはビット 18 から 31 までがセットされます。

この命令は $C_USE_MSR_INSTR$ を 1 に設定している場合のみ使用可能です。

MSR[VM] をセットすると、この命令の後には、常に、BRI 4 などの同期分岐命令が続く必要があります。

mts
特殊レジスタに移動

mts rS、rA
 mtse rS、rA

1	0	0	1	0	1	0	E	0	0	0		rA	1	1			rS	
0				6				11				16	18				31	

説明

レジスタ rD の内容を特殊レジスタ rS にコピーします。特殊レジスタ TLBLO および TLBHI は、TLBX によりインデックス化された統合 TLB エントリにコピーするのに使用されます。

E ビットがセットされている場合、特殊レジスタの拡張部分は移動されます。物理アドレス拡張 (PAE) がイネーブルの場合、TLBLO レジスタに拡張部分が含まれます。

MicroBlaze が MMU を使用するように (*c_USE_MMU* >= 1) 設定されている場合、この命令は特権命令です。つまり、命令がユーザー モード (MSR[UM] = 1) で実行されると、特権命令例外が発生します。

低レイテンシ割り込みモードでは (*c_USE_INTERRUPT* = 2)、この命令を実行して MSR[IE] ビットがセットされる場合に、Interrupt_Ack 出力ポートが 1 に設定されます。

擬似コード

```

if MSR[UM] = 1 then
    ESR[EC] ← 00111
else
    if E = 1 then
        if (rS) = 0x1003 then
            TLBLO[0:C_ADDR_SIZE-32-1] ← (rA)
        else
            switch (rS)
                case 0x0001 : MSR ← (rA)
                case 0x0007 : FSR ← (rA)
                case 0x0800 : SLR ← (rA)
                case 0x0802 : SHR ← (rA)
                case 0x1000 : PID ← (rA)
                case 0x1001 : ZPR ← (rA)
                case 0x1002 : TLBX ← (rA)
                case 0x1003 : TLBLO[C_ADDR_SIZE-32:C_ADDR_SIZE-1] ← (rA)
                case 0x1004 : TLBHI ← (rA)
                case 0x1005 : TLBSX ← (rA)
            if (rS) = 0x0001 and (rA) & 2
                Interrupt_Ack ← 11

```

変更されるレジスタ

- rS
- ESR[EC] (特権命令例外が生成される場合)

レイテンシ

- 1 サイクル

注記

MTS を使用して MSR を書き込むとき、この命令の実行後 1 サイクルで、すべてのビットが効力を持ちます。MSR を書き込む MTS 命令の後に、MSR の内容を使用する命令が連續して続くことはできません。IE ビットをクリアになると、後続の命令に対する割り込みにはプロセッサは反応しなくなります。EIP または BIP ビットをセットすると、後続の命令に対する割り込みまたは標準ハードウェアブレークにはプロセッサは反応しなくなります。

アセンブリ言語で特殊レジスタを参照するには、MSR には rmsr、FSR には rfsr、SLR には rslr、SHR には rshr、PID には rpid、ZPR には rzpr、TLBLO には rtlblo、TLBHI には rtlbhi、TLBX には rtlbx、TLBSX には rtlbsx を使用します。

PC、ESR、EAR、BTR、EDR、および PVR0 から PVR12 は、MTS 命令で書き込むことはできません。

MicroBlaze のパラメーター C_USE_FPU が 0 よりも大きな値のとき、FSR はデスティネーションとしてのみ有効です。

MicroBlaze のパラメーター C_USE_STACK_PROTECTION が 1 に設定されていると、SLR および SHR はデスティネーションとしてのみ有効です。

C_USE_MMU > 1 (ユーザー モード) および C_MMU_TLB_ACCESS > 1 (読み出し) に設定されていると、PID、ZPR、および TLBSX はデスティネーションとしてのみ有効です。C_USE_MMU > 1 (ユーザー モード) のとき、TLBLO、TLBHI、および TLBX はデスティネーションとしてのみ有効です。

MSR[VM] または PID を変更するとき、この命令の後には、常に、BRI 4 などの同期分岐命令が続く必要があります。

UTLB エントリを 1 つ以上無効化するため TLBHI に書き込んだ後は、コヒーレントのマルチプロセッサシステムでコヒーレンシが保持されるようにするために、MBAR 1 命令を発行する必要があります。

PAE がイネーブルの場合、まず拡張命令を使用して最上位ビットを書き込んだ直後に最下位ビットを書き込むことで、TLBLO レジスタ全体を書き込む必要があります。

拡張命令は、MicroBlaze が仮想モード (C_USE_MMU = 3) および拡張アドレス (C_ADDR_SIZE > 32) で MMU を使用するように設定されている場合にのみ有効です。

mul 乗算

mul rD, rA, rB

0	1	0	0	0	0	rD	rA	rB	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0						6	11	16	21																	31	

説明

レジスタ rA および rB の内容を乗算し、レジスタ rD に結果を出力します。これは 32 ビット値を 32 ビットの値で乗算したもので、64 ビットを出力します。この値の最下位ワードが rD に配置されます。最上位ワードは破棄されます。

擬似コード

```
(rD) ← LSW( (rA) × (rB) )
```

変更されるレジスタ

- rD

レイテンシ

- 1 サイクル (C_AREA_OPTIMIZED=0 または 2)
- 3 サイクル (C_AREA_OPTIMIZED=1)

注記:

この命令はターゲット アーキテクチャに乗算器プリミティブがある場合にのみ有効で、乗算器プリミティブがある場合は、MicroBlaze のパラメーター C_USE_HW_MUL は 0 よりも大きい値になります。

mulh**乗算上位**

mulh rD, rA, rB

0	1	0	0	0	0	rD	rA	rB	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
0						6	11	16	21															31

説明

レジスタ rA および rB の内容を乗算し、レジスタ rD に結果を出力します。これは 32 ビット値を 32 ビットの符号付きの値で乗算したもので、64 ビットを出力します。この値の最上位ワードが rD に配置されます。最下位ワードは破棄されます。

擬似コード

```
(rD) ← MSW( (rA) × (rB) ), signed
```

変更されるレジスタ

- rD

レイテンシ

- 1 サイクル (C_AREA_OPTIMIZED=0 または 2)
- 3 サイクル (C_AREA_OPTIMIZED=1)

注記

この命令は、ターゲットアーキテクチャに乗算器プリミティブがある場合にのみ有効で、もしそのプリミティブがある場合は、MicroBlaze のパラメーター C_USE_HW_MUL は 2 (Mul64) に設定されます。

MULH が使用されている場合は、MUL 命令のビット 30 および 31 は、この 2 つの命令を区別するため、0 にする必要があります。古いバージョンの MicroBlaze では、これらのビットは 0 に定義されていましたが、実際の値は関係ありませんでした。

mulhu 乗算上位(符号なし)

mulhu rD, rA, rB

0	1	0	0	0	0	rD	rA	rB	0	0	0	0	0	0	0	0	0	1	1
0						6	11	16	21									31	

説明

レジスタ rA および rB の内容を乗算し、レジスタ rD に結果を出力します。これは 32 ビットの値を 32 ビットの符号なしの値で乗算したもので、符号なし 64 ビットを出力します。この値の最上位ワードが rD に配置されます。最下位ワードは破棄されます。

擬似コード

```
(rD) ← MSW( (rA) × (rB) ), unsigned
```

変更されるレジスタ

- rD

レイテンシ

- 1 サイクル (C_AREA_OPTIMIZED=0 または 2)
- 3 サイクル (C_AREA_OPTIMIZED=1)

注記

この命令は、ターゲットアーキテクチャに乗算器プリミティブがある場合にのみ有効で、もしそのプリミティブがある場合は、MicroBlaze のパラメーター C_USE_HW_MUL は 2 (Mul64) に設定されます。

MULHU が使用されている場合は、MUL 命令のビット 30 および 31 は、この 2 つの命令を区別するため、0 にする必要があります。古いバージョンの MicroBlaze では、これらのビットは 0 に定義されていましたが、実際の値は関係がありませんでした。

mulhsu 乗算上位(符号付き/符号なし)

mulhsu rD, rA, rB

0	1	0	0	0	0	rD	rA	rB	0	0	0	0	0	0	0	0	0	1	0
0						6	11	16	21									31	

説明

レジスタ rA および rB の内容を乗算し、レジスタ rD に結果を出力します。これは 32 ビットの符号付きの値を 32 ビットの符号なしの値で乗算したもので、符号なし 64 ビットを出力します。この値の最上位ワードが rD に配置されます。最下位ワードは破棄されます。

擬似コード

```
(rD) ← MSW( (rA), signed × (rB), unsigned ), signed
```

変更されるレジスタ

- rD

レイテンシ

- 1 サイクル (C_AREA_OPTIMIZED=0 または 2)
- 3 サイクル (C_AREA_OPTIMIZED=1)

注記

この命令は、ターゲットアーキテクチャに乗算器プリミティブがある場合にのみ有効で、もしそのプリミティブがある場合は、MicroBlaze のパラメーター C_USE_HW_MUL は 2 (Mul64) に設定されます。

MULHSU が使用されている場合は、MUL 命令のビット 30 および 31 は、この 2 つの命令を区別するため、0 にする必要があります。古いバージョンの MicroBlaze では、これらのビットは 0 に定義されていましたが、実際の値は関係がありませんでした。

muli**即値との乗算**

muli rD, rA, IMM

0	1	1	0	0	0	rD	rA	IMM
0			6		11		16	31

説明

レジスタ rA の内容と 32 ビットに符号拡張された IMM 値を乗算し、その結果をレジスタ rD に出力します。これは 32 ビット値を 32 ビットの値で乗算したもので、64 ビットを出力します。この値の最下位ワードが rD に配置されます。最上位ワードは破棄されます。

擬似コード

```
(rD) ← LSW( (rA) × sext(IMM) )
```

変更されるレジスタ

- rD

レイテンシ

- 1 サイクル (C_AREA_OPTIMIZED=0 または 2)
- 3 サイクル (C_AREA_OPTIMIZED=1)

注記

デフォルトでは、タイプ B 命令は 16 ビットの IMM フィールド値を取り込み、それを 32 ビットに符号拡張して即値オペランドとして使用します。この動作は、タイプ B 命令の前に imm 命令を先行させると変更できます。32 ビットの即値の使用については、「imm」を参照してください。

この命令はターゲット アーキテクチャに乗算器プリミティブがある場合にのみ有効で、乗算器プリミティブがある場合は、MicroBlaze のパラメータ C_USE_HW_MUL は 0 よりも大きい値になります。

or**論理 OR**

or rD, rA, rB

1	0	0	0	0	0	rD	rA	rB	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0						6	11	16	21																	31

説明

レジスタ rA の内容をレジスタ rB の内容と OR 演算し、その結果をレジスタ rD に配置します。

擬似コード
$$(rD) \leftarrow (rA) \vee (rB)$$
変更されるレジスタ

- rD

レイテンシ

- 1 サイクル

注記:

アセンブラーの擬似命令 nop は「or r0, r0, r0」としてインプリメントされます。

ori**即値との論理 OR**

ori rD, rA, IMM

1	0	1	0	0	0	rD	rA	IMM
0			6		11		16	31

説明

レジスタ rA の内容を 32 ビットに符号拡張された IMM フィールドの値と OR 演算し、その結果をレジスタ rD に配置します。

擬似コード

$$(rD) \leftarrow (rA) \vee \text{sext(imm)}$$
変更されるレジスタ

- rD

レイテンシ

- 1 サイクル

注記:

デフォルトでは、タイプ B 命令は 16 ビットの IMM フィールド値を取り込み、それを 32 ビットに符号拡張して即値オペランドとして使用します。この動作は、タイプ B 命令の前に imm 命令を先行させると変更できます。32 ビットの即値の使用については、「[imm](#)」を参照してください。

pcmpbf バイト単位のパターン比較

pcmpbf rD, rA, rB バイト単位の比較を実行し、最初に一致した位置を返す

1 0 0 0 0 0	rD	rA	rB	1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	
0	6	11	16	21	31

説明

レジスタ rA の内容をレジスタ rB の内容とバイト単位で比較します。

- MSB (位置 1) から LSB (位置 4) までを比較し、最初に一致したバイトペアの位置を rD にロードします。
- 一致するバイトペアがない場合は、rD を 0 にセットします。

擬似コード

```

if rB[0:7] = rA[0:7] then
    (rD) ← 1
else
    if rB[8:15] = rA[8:15] then
        (rD) ← 2
    else
        if rB[16:23] = rA[16:23] then
            (rD) ← 3
        else
            if rB[24:31] = rA[24:31] then
                (rD) ← 4
            else
                (rD) ← 0

```

変更されるレジスタ

- rD

レイテンシ

- 1 サイクル

注記:

この命令は C_USE_PCMP_INSTR を 1 に設定している場合のみ使用可能です。

pccmpeq パターン比較等価

pccmpeq rD, rA, rB 等しいかどうかを比較して正のブール値を返す

1 0 0 0 1 0	rD	rA	rB	1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	
0	6	11	16	21	31

説明

レジスタ rA の内容をレジスタ rB の内容と比較します。

- 一致した場合は rD に 1 がロードされ、一致しない場合は 0 がロードされます。

擬似コード

```
if (rB) = (rA) then
    (rD) ← 1
else
    (rD) ← 0
```

変更されるレジスタ

- rD

レイテンシ

- 1 サイクル

注記:

この命令は C_USE_PCMP_INSTR を 1 に設定している場合のみ使用可能です。

pcmpne パターン不等価比較

pcmpne rD, rA, rB 等しいかどうかを比較して負のブール値を返す

1 0 0 0 1 1	rD	rA	rB	1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	
0	6	11	16	21	31

説明

レジスタ rA の内容をレジスタ rB の内容と比較します。

- 一致した場合は rD に 0 がロードされ、一致しない場合は 1 がロードされます。

擬似コード

```
if (rB) = (rA) then
    (rD) ← 0
else
    (rD) ← 1
```

変更されるレジスタ

- rD

レイテンシ

- 1 サイクル

注記:

この命令は C_USE_PCMP_INSTR を 1 に設定している場合のみ使用可能です。

put ストリーム インターフェイスに配置

<i>nput</i>	rA、FSLx	リンク x にデータを配置 n: ノンブロッキング a: アトミック
<i>tput</i>	FSLx	リンク x にデータを配置 (テストのみ) n: ノンブロッキング a: アトミック
<i>ncput</i>	rA、FSLx	リンク x に制御を配置 n: ノンブロッキング a: アトミック
<i>tncput</i>	FSLx	リンク x に制御を配置 (テストのみ) n: ノンブロッキング a: アトミック

0	1	1	0	1	1	0	0	0	0	0	rA	1	n	c	t	a	0	0	0	0	0	0	0	FSLx
0						6					11					16							28	31

説明

レジスタ rA からの値をリンク x インターフェイスに書き込みます。C_FSL_LINKS により設定されているリンク数が FSLx 以下の場合、リンク 0 が使用されます。

put 命令には 16 個のバリエーションがあります。

ブロッキング バージョン (n が 0 のとき) は、インターフェイスに空きができるまで MicroBlaze をストールします。ノンブロッキング バージョンは、MicroBlaze をストールし、空きがあればキャリーを 0 に、空きがなければキャリーを 1 にセットします。

すべてのデータ put 命令 (c が 0 のとき) はインターフェイスへの制御ビットを 0 にセットし、すべての制御 put 命令 (c が 1 のとき) は制御ビットを 1 にセットします。

テスト バージョン (t ビットが 1 のとき) は、リンクへの書き込み信号がアサートされていない場合を除き、標準ケースとして処理されます(つまりソース レジスタは不要)。

アトミック バージョン (a ビットが 1 のとき) は、割り込みできません。つまり、アトミック命令のシーケンスは、プログラム フローを割り込むことなく、グループにまとめることができます。ただし、それでも例外は発生する可能性があります。

MicroBlaze が MMU を使用するように設定されていて (C_USE_MMU >= 1)、明示的に C_MMU_PRIVILEGED_INSTR が 1 に設定されていない場合は、これらの命令は特権になります。つまり、命令がユーザー モード (MSR[UM]=1) で実行されると、特権命令例外が発生します。

擬似コード

```

if MSR[UM] = 1 then
    ESR[EC] ← 00111
else
    x ← FSLx
    if x >= C_FSL_LINKS then
        x ← 0
    Mx_AXIS_TDATA ← (rA)
    if (n = 1) then
        MSR[Carry] ← Mx_AXIS_TVALID ∧ Mx_AXIS_TREADY
    Mx_AXIS_TLAST ← C

```

変更されるレジスタ

- MSR[Carry]
- ESR[EC] (特権命令例外が生成される場合)

レイテンシ

- 1 サイクル (C_AREA_OPTIMIZED=0 または 2)
- 2 サイクル (C_AREA_OPTIMIZED=1)

この命令のブロッキング バージョンは、命令が完了できるまで MicroBlaze のパイプラインをストールさせます。C_USE_EXTENDED_FSL_INSTR が 1 に設定されていて、命令がアトミックでない場合に、割り込みが実行されます。

注記

アセンブリ言語で FSLx インターフェイスを参照するには、rfsl0、rfsl1、... rfsl15 を使用してください。

この命令のブロッキング バージョンは、割り込みが実行されなくなるので、C_USE_EXTENDED_FSL_INSTR が 1 に設定されているときに遅延スロットに配置しないでください。

これたの命令は、MicroBlaze のパラメーター C_FSL_LINKS が 0 より大きい場合のみ使用可能です。

拡張命令 (テストおよびアトミック バージョン) は、MicroBlaze のパラメーター C_USE_EXTENDED_FSL_INSTR が 1 に設定されている場合にのみ使用可能です。

パフォーマンス上の理由からどうしてもこの命令をユーザー モードで実行しなければならない場合を除き、この命令をユーザー モードで実行しないようにしてください。リンクの誤用を防ぐためのハードウェア保護がすべて削除されてしまいます。

putd ストリーム インターフェイスに配置 (動的)

<i>n</i> aputd	rA, rB	データをリンク rB[28:31] に配置 n: ノンブロッキング a: アトミック
<i>t</i> nputd	rB	データをリンク rB[28:31] に配置 (テストのみ) n: ノンブロッキング a: アトミック
<i>n</i> caputd	rA, rB	制御をリンク rB[28:31] に配置 n: ノンブロッキング a: アトミック
<i>t</i> ncaputd	rB	制御をリンク rB[28:31] テストのみに配置 n: ノンブロッキング a: アトミック

0	1	0	0	1	1	0	0	0	0	0	rA	rB	1	n	c	t	a	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
0						6					11		16					21														31			

説明

レジスタ rA からの値を、rB の最下位 4 ビットで定義されたリンク インターフェイスに書き込みます。
C_FSL_LINKS で設定されているリンク数が rB の最下位 4 ビット以下の場合、リンク 0 が使用されます。

putd 命令には 16 個のバリエーションがあります。

プロッキング バージョン (n が 0 のとき) は、インターフェイスに空きができるまで MicroBlaze をストールします。ノンプロッキング バージョンは、MicroBlaze をストールし、空きがあればキャリーを 0 に、空きがなければキャリーを 1 にセットします。

すべてのデータ putd 命令 (c が 0 のとき) はインターフェイスへの制御ビットを 0 にセットし、すべての制御 putd 命令 (c が 1 のとき) は制御ビットを 1 にセットします。

テスト バージョン (t ビットが 1 のとき) は、リンクへの書き込み信号がアサートされていない場合を除き、標準ケースとして処理されます(つまりソース レジスタは不要)。

アトミック バージョン (a ビットが 1 のとき) は、割り込みできません。つまり、アトミック命令のシケンスは、プログラム フローを割り込むことなく、グループにまとめることができます。ただし、それでも例外は発生する可能性があります。

MicroBlaze が MMU を使用するように設定されていて (C_USE_MMU >= 1)、明示的に C_MMU_PRIVILEGED_INSTR が 1 に設定されていない場合は、これらの命令は特権になります。つまり、命令がユーザー モード (MSR[UM]=1) で実行されると、特権命令例外が発生します。

擬似コード

```
if MSR[UM] = 1 then
    ESR[EC] ← 00111
else
    x ← rB[28:31]
    if x >= C_FSL_LINKS then
        x ← 0
    Mx_AXIS_TDATA ← (rA)
    if (n = 1) then
        MSR[Carry] ← Mx_AXIS_TVALID ∧ Mx_AXIS_TREADY
    Mx_AXIS_TLAST ← C
```

変更されるレジスタ

- MSR[Carry]
- ESR[EC] (特権命令例外が生成される場合)

レイテンシ

- 1 サイクル (C_AREA_OPTIMIZED=0 または 2)
- 2 サイクル (C_AREA_OPTIMIZED=1)

この命令のブロッキング バージョンは、命令が完了できるまで MicroBlaze のパイプラインをストールさせます。命令がアトミックでなければ割り込みは実行されます。アトミックだと割り込みは実行されません。

注記

割り込みが実行されなくなってしまうので、この命令のブロッキング バージョンは、遅延スロットに配置しないでください。

MicroBlaze パラメーター C_FSL_LINKS が 0 より大きく、パラメーター C_USE_EXTENDED_FSL_INSTR が 1 に設定されている場合にのみ、これらの命令は使用可能です。

パフォーマンス上の理由からどうしてもこの命令をユーザー モードで実行しなければならない場合を除き、この命令をユーザー モードで実行しないようにしてください。リンクの誤用を防ぐためのハードウェア保護がすべて削除されてしまいます。

rsub**算術逆減算**

rsub	rD, rA, rB	減算
rsubc	rD, rA, rB	キャリー付き減算
rsubk	rD, rA, rB	減算およびキャリー保持
rsubkc	rD, rA, rB	キャリー付き減算およびキャリー保持

0	0	0	K	C	1	rD	rA	rB	0	0	0	0	0	0	0	0	0	0
---	---	---	---	---	---	----	----	----	---	---	---	---	---	---	---	---	---	---

説明

レジスタ rA の内容をレジスタ rB の内容から減算し、その結果をレジスタ rD に配置します。命令のビット 3(図の K)は、ニーモニック rsubk に対して 1 にセットされます。命令のビット 4(図の C)は、ニーモニック rsubc に対して 1 にセットされます。ニーモニック rsubkc の場合は、両方のビットが 1 にセットされます。

rsub 命令でビット 3 がセットされている場合(rsubk, rsubkc)、命令の実行結果にかかわらず、キャリーフラグは前の値を保持します。ビット 3 がクリアされている場合(rsub, rsubc)、キャリーフラグは命令の実行結果によって変更されます。

命令のビット 4 が 1 にセットされている場合(rsubc, rsubkc)、キャリーフラグの内容(MSR[C])が命令実行に影響します。ビット 4 がクリアされている場合(rsub, rsubk)、キャリーフラグの内容は命令実行には影響しません(標準減算の場合)。

擬似コード

```

if C = 0 then
    (rD) ← (rB) + (rA) + 1
else
    (rD) ← (rB) + (rA) + MSR[C]
if K = 0 then
    MSR[C] ← CarryOut

```

変更されるレジスタ

- rD
- MSR[C]

レイテンシ

- 1 サイクル

注記:

減算では、キャリー=(ボロー)です。減算でキャリー(繰り上がり)が設定される場合は、ボロー(繰り下がり)がないということで、キャリーがクリアになると、ボローがあるということになります。

rsubi 算術即値逆減算

rsubi	rD, rA, IMM	即値減算
rsubic	rD, rA, IMM	キャリー付き即値減算
rsubik	rD, rA, IMM	即値減算およびキャリー保持
rsubikc	rD, rA, IMM	キャリー付き即値減算およびキャリー保持

0	0	1	K	C	1	rD	rA	IMM
0			6		11		16	31

説明

レジスタ rA の内容を IMM の 32 ビットに符号拡張された値から引き、その結果をレジスタ rD に配置します。命令のビット 3(図の K)は、二モニック rsubik に対して 1 にセットされます。命令のビット 4(図の C)は、二モニック rsubic に対して 1 にセットされます。二モニック rsubikc の場合は、両方のビットが 1 にセットされます。

rsubi 命令でビット 3 がセットされている場合(rsubik, rsubikc)、命令の実行結果にかかわらず、キャリー フラグは前の値を保持します。ビット 3 がクリアされている場合(rsubi, rsubic)、キャリー フラグは命令の実行結果によって変更されます。命令のビット 4 が 1 にセットされている場合(rsubic, rsubikc)、キャリー フラグの内容(MSR[C])が命令実行に影響します。ビット 4 がクリアされている場合(rsubi, rsubik)、キャリー フラグの内容は命令実行には影響しません(標準減算の場合)。

擬似コード

```

if C = 0 then
    (rD) ← sext(IMM) + (rA) + 1
else
    (rD) ← sext(IMM) + (rA) + MSR[C]
if K = 0 then
    MSR[C] ← CarryOut

```

変更されるレジスタ

- rD
- MSR[C]

レイテンシ

- 1 サイクル

注記:

減算では、キャリー=(ボロー)です。減算でキャリー(繰り上がり)が設定される場合は、ボロー(繰り下がり)がないということで、キャリーがクリアになると、ボローがあるということになります。デフォルトでは、タイプ B 命令は 16 ビットの IMM フィールド値を取り込み、それを 32 ビットに符号拡張して即値オペランドとして使用します。この動作は、タイプ B 命令の前に imm 命令を先行させると変更できます。32 ビットの即値の使用については、「imm」を参照してください。

rtbd ブレークから戻る

rtbd rA_X, IMM

1	0	1	1	0	1	1	0	0	1	0	rA _X	IMM
0			6			11			16			31

説明

ブレークから戻ったときに、rA_X の内容と符号拡張された IMM フィールドの内容を加算して指定されたロケーションに分岐します。また、MSR の BIP フラグをクリアすることにより実行後ブレークをイネーブルにします。

この命令には常に遅延スロットがあります。RTBD の後に続く命令は、常に、分岐先の前に実行されます。その遅延スロット命令ではブレークはディスエーブルになっています。

MicroBlaze が MMU を使用するように (C_USE_MMU >= 1) 設定されている場合、この命令は特権命令です。つまり、命令がユーザー モード (MSR[UM] = 1) で実行されると、特権命令例外が発生します。

擬似コード

```
if MSR[UM] = 1 then
    ESR[EC] ← 00111
else
    PC ← (rAX) + sext(IMM)
    allow following instruction to complete execution
    MSR[BIP] ← 0
    MSR[UM] ← MSR[UMS]
    MSR[VM] ← MSR[VMS]
```

変更されるレジスタ

- PC
- MSR[BIP]、MSR[UM]、MSR[VM]
- ESR[EC] (特権命令例外が生成される場合)

レイテンシ

- 2 サイクル

注記

変換は汎用レジスタ r16 を rA_X として使用するためのものです。

遅延スロットには、imm、分岐、またはブレーク命令を含めることはできません。割り込みおよび外部ハードウェアブレークは、遅延スロット分岐が完了するまで保留されます。

rtid 割り込みから戻る

rtid rA_X, IMM

1	0	1	1	0	1	1	0	0	0	1	rA _X	IMM
0			6			11			16			31

説明

割り込みから戻ったときに、rA_X の内容と符号拡張された IMM フィールドの内容を加算して指定されたロケーションに分岐します。また、実行後に割り込みがイネーブルになります。

この命令には常に遅延スロットがあります。RTID の後に続く命令は、常に、分岐先の前に実行されます。その遅延スロット命令では割り込みはディスエーブルになっています。

MicroBlaze が MMU を使用するように (C_USE_MMU >= 1) 設定されている場合、この命令は特権命令です。つまり、命令がユーザー モード (MSR[UM] = 1) で実行されると、特権命令例外が発生します。

低レイテンシ割り込みモードだと C_USE_INTERRUPT = 2)、この命令が実行されると Interrupt_Ack 出力ポートが 10 にセットされ、その後で MSR[IE] ビットがセットされると 11 になります。

擬似コード

```
if MSR[UM] = 1 then
    ESR[EC] ← 00111
else
    PC ← (rAX) + sext(IMM)
    Interrupt_Ack ← 10
    allow following instruction to complete execution
    MSR[IE] ← 1
    MSR[UM] ← MSR[UMS]
    MSR[VM] ← MSR[VMS]
    Interrupt_Ack ← 11
```

変更されるレジスタ

- PC
- MSR[IE]、MSR[UM]、MSR[VM]
- ESR[EC] (特権命令例外が生成される場合)

レイテンシ

- 2 サイクル

注記

変換は汎用レジスタ r14 を rA_X として使用するためのものです。

遅延スロットには、imm、分岐、またはブレーク命令を含めることはできません。割り込みおよび外部ハードウェアブレークは、遅延スロット分岐が完了するまで保留されます。

rted 例外から戻る

rted rAx, IMM

1	0	1	1	0	1	1	0	1	0	0		rAx		IMM	
0					6			11			16				31

説明

例外から戻ったときに、rAx の内容と符号拡張された IMM フィールドの内容を加算して指定されたロケーションに分岐します。また、実行後に例外がイネーブルになります。

この命令には常に遅延スロットがあります。RTED の後に続く命令は、常に、分岐先の前に実行されます。

MicroBlaze が MMU を使用するように (C_USE_MMU >= 1) 設定されている場合、この命令は特権命令です。つまり、命令がユーザー モード (MSR[UM] = 1) で実行されると、特権命令例外が発生します。

擬似コード

```
if MSR[UM] = 1 then
    ESR[EC] ← 00111
else
    PC ← (rAx) + sext(IMM)
    allow following instruction to complete execution
    MSR[EE] ← 1
    MSR[EIP] ← 0
    MSR[UM] ← MSR[UMS]
    MSR[VM] ← MSR[VMS]
    ESR ← 0
```

変更されるレジスタ

- PC
- MSR[EE]、MSR[EIP]、MSR[UM]、MSR[VM]
- ESR

レイテンシ

- 2 サイクル

注記

変換は汎用レジスタ r17 を rAx として使用するためのものです。この命令の場合、MicroBlaze のパラメーター C_*_EXCEPTION が 1 つ以上 1 に設定されているか、C_USE_MMU > 0 である必要があります。

遅延スロットには、imm、分岐、またはブレーク命令を含めることはできません。割り込みおよび外部ハードウェア ブレークは、遅延スロット分岐が完了するまで保留されます。

遅延スロットの命令が例外を引き起こす場合、例外ハンドラーには、例外がイネーブルになっていないと入れないため、MSR[EE] がセットされているときは、この命令を通常は使用しないでください。

MSR[DS] がセットされている場合は、まず例外から戻るコードをチェックする必要があり、その場合は BTR のアドレスに戻ります。

rtsd サブルーチンから戻る

rtsd rA_X, IMM

1	0	1	1	0	1	1	0	0	0	0	rA _X	IMM
0			6			11		16				31

説明

サブルーチンから戻ったときに、rA_X の内容と符号拡張された IMM フィールドの内容を加算して指定されたロケーションに分岐します。

この命令には常に遅延スロットがあります。RTSD の後に続く命令は、常に、分岐先の前に実行されます。

擬似コード

```
PC ← (rAX) + sext(IMM)
allow following instruction to complete execution
```

変更されるレジスタ

- PC

レイテンシ

- 1 サイクル(正しく分岐予測がされる場合)
- 2 サイクル(分岐先キャッシュがディスエーブルの場合)
- 3 サイクル(C_AREA_OPTIMIZED=0 で分岐予測が間違っている場合)
- 7 ~ 9 サイクル(C_AREA_OPTIMIZED=2 で分岐予測が間違っている場合)

注記

変換は汎用レジスタ r15 を rA_X として使用するためのものです。

遅延スロットには、imm、分岐、またはブレーク命令を含めることはできません。割り込みおよび外部ハードウェアブレークは、遅延スロット分岐が完了するまで保留されます。

sb

バイトを格納

sb	rD, rA _X , rB _X
sbr	rD, rA _X , rB _X
sbea	rD, rA, rB

1	1	0	1	0	0	rD	rA _X	rB _X	0	R	0	EA	0	0	0	0	0	0	0	
0			6			11		16		21										31

説明

レジスタ rD の最下位バイトの内容を、レジスタ rA_X および rB_X の内容を加算した結果のメモリ ロケーションに格納します。

R ビットがセットされている場合、バイト反転メモリ ロケーションを使用し、E ビットで定義されているエンディアンネスとは逆のエンディアンネスでデータを格納します(仮想保護モードがイネーブルの場合)。

EA ビットがセットされている場合、拡張アドレスが使用され、rA と rB を足すのではなく連結させて形成されます。

仮想保護モードがイネーブルの場合、データ TLB ミス例外が発生し、アドレスに対応する有効な変換エントリは TLB では検出されません。

仮想保護モードがイネーブルになっているとデータストレージ例外が発生し、ノーアクセスまたは読み出し専用ゾーン保護によってアクセスが禁止されます。ノーアクセス許可はユーザー モードの場合にのみ発生します。

EA ビットがセットされていて、物理アドレス拡張(PAE) がイネーブルで、命令が明示的に許可されていない場合は、特権命令エラーが発生します。

擬似コード

```

if EA = 1 then
    Addr ← (rA) & (rB)
else
    Addr ← (rAX) + (rBX)
if TLB_Miss(Addr) and MSR[VM] = 1 then
    ESR[EC] ← 10010; ESR[S] ← 1
    MSR[UMS] ← MSR[UM]; MSR[VMS] ← MSR[VM]; MSR[UM] ← 0; MSR[VM] ← 0
else if Access_Protected(Addr) and MSR[VM] = 1 then
    ESR[EC] ← 10000; ESR[S] ← 1; ESR[DIZ] ← No-access-allowed
    MSR[UMS] ← MSR[UM]; MSR[VMS] ← MSR[VM]; MSR[UM] ← 0; MSR[VM] ← 0
else
    Mem(Addr) ← (rD) [C_DATA_SIZE-8:C_DATA_SIZE-1]

```

変更されるレジスター

- MSR[UM]、MSR[VM]、MSR[Ums]、MSR[Vms] (例外が生成される場合)
- ESR[EC]、ESR[S] (例外が生成される場合)
- ESR[DIZ] (データストレージ例外が生成される場合)

レイテンシ

- 1 サイクル (C_AREA_OPTIMIZED=0 または 2)
- 2 サイクル (C_AREA_OPTIMIZED=1)

注記

MicroBlaze が再順序付け命令を使用するように設定されている場合にのみ (C_USE_REORDER_INSTR = 1)、このバイト反転命令は有効です。

拡張アドレス命令は、MicroBlaze が拡張アドレス命令を使用するよう設定されており (C_ADDR_SIZE > 32)、32 ビット モードを使用している場合 (C_DATA_SIZE = 32) にのみ有効です。

sbi**バイトを格納(即値)**sbi rD, rA_X, IMM

1	1	1	1	0	0	rD	rA _X	IMM
0			6		11		16	31

説明

レジスタ rD の最下位バイトの内容を、レジスタ rA_X の内容と符号拡張された IMM 値を加算した結果のメモリ ロケーションに格納します。

仮想保護モードがイネーブルの場合、データ TLB ミス例外が発生し、アドレスに対応する有効な変換エントリは TLB では検出されません。

仮想保護モードがイネーブルになっているとデータストレージ例外が発生し、ノーアクセスまたは読み出し専用ゾーン保護によってアクセスが禁止されます。ノーアクセス許可はユーザー モードの場合にのみ発生します。

擬似コード

```

Addr ← (rAX) + sext(IMM)
if TLB_Miss(Addr) and MSR[VM] = 1 then
    ESR[EC] ← 10010; ESR[S] ← 1
    MSR[UMS] ← MSR[UM]; MSR[VMS] ← MSR[VM]; MSR[UM] ← 0; MSR[VM] ← 0
else if Access_Protected(Addr) and MSR[VM] = 1 then
    ESR[EC] ← 10000; ESR[S] ← 1; ESR[DIZ] ← No-access-allowed
    MSR[UMS] ← MSR[UM]; MSR[VMS] ← MSR[VM]; MSR[UM] ← 0; MSR[VM] ← 0
else
    Mem(Addr) ← (rD)[C_DATA_SIZE-8:C_DATA_SIZE-1]

```

変更されるレジスタ

- MSR[UM]、MSR[VM]、MSR[Ums]、MSR[VMS] (例外が生成される場合)
- ESR[EC]、ESR[S] (例外が生成される場合)
- ESR[DIZ] (データストレージ例外が生成される場合)

レイテンシ

- 1 サイクル (C_AREA_OPTIMIZED=0 または 2)
- 2 サイクル (C_AREA_OPTIMIZED=1)

注記:

デフォルトでは、タイプ B ストア命令は 16 ビットの IMM フィールド値を取り込み、それを符号拡張して即値オペランドとして使用します。この動作は、imm または imml 命令を先行させると変更できます。即値の使用については、「imm」および「imml」を参照してください。

sext16 ハーフワードを符号拡張

sext16 rD, rA

1	0	0	1	0	0	rD	rA	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	1
0			6			11		16									31						

説明

この命令は、ハーフワード(16ビット)をワード(32ビット)に符号拡張します。rAのビット16が、rDのビット0～15にコピーされます。rAのビット16～31は、rDのビット16～31にコピーされます。

擬似コード

```
(rD)[0:15] ← (rA)[16]  
(rD)[16:31] ← (rA)[16:31]
```

変更されるレジスタ

- rD

レイテンシ

- 1サイクル

sext8 バイトを符号拡張

sext8 rD, rA

1	0	0	1	0	0	rD	rA	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0
0			6			11		16														31

説明

この命令は、バイト(8ビット)をワード(32ビット)に符号拡張します。rAのビット24が、rDのビット0～23にコピーされます。rAのビット24～31は、rDのビット24～31にコピーされます。

擬似コード

```
(rD)[0:23] ← (rA)[24]
(rD)[24:31] ← (rA)[24:31]
```

変更されるレジスタ

- rD

レイテンシ

- 1サイクル

sh**ハーフワードを格納**

sh	rD, rA _X , rB _X
shr	rD, rA _X , rB _X
shea	rD, rA, rB

1	1	0	1	0	1	rD	rA _X	rB _X	0	R	0	EA	0	0	0	0	0	0	0	
0						6	11	16				21								31

説明

レジスタ rD の最下位ハーフワードの内容を、レジスタ rA_X および rB_X の内容を加算した結果の、ハーフワードに境界整列されたメモリロケーションに格納します。

R ビットがセットされている場合、ハーフワード反転メモリロケーションを使用し、ハーフワードの 2 バイトを反転して、E ビットで定義されているエンディアンネスとは逆のエンディアンネスでデータを格納します(仮想保護モードがイネーブルの場合)。

EA ビットがセットされている場合、拡張アドレスが使用され、rA と rB を足すのではなく連結させて形成されます。

仮想保護モードがイネーブルの場合、データ TLB ミス例外が発生し、アドレスに対応する有効な変換エントリは TLB では検出されません。

仮想保護モードがイネーブルになっているとデータストレージ例外が発生し、ノーアクセスまたは読み出し専用ゾーン保護によってアクセスが禁止されます。ノーアクセス許可はユーザー モードの場合にのみ発生します。

アドレスの最下位ビットがゼロでない場合は、非境界整列データ アクセス例外が発生します。

EA ビットがセットされていて、物理アドレス拡張(PAE) がイネーブルで、命令が明示的に許可されていない場合は、特権命令エラーが発生します。

擬似コード

```

if EA = 1 then
    Addr ← (rA) & (rB)
else
    Addr ← (rAX) + (rBX)
if TLB_Miss(Addr) and MSR[VM] = 1 then
    ESR[EC] ← 10010; ESR[S] ← 1
    MSR[UMS] ← MSR[UM]; MSR[VMS] ← MSR[VM]; MSR[UM] ← 0; MSR[VM] ← 0
else if Access_Protected(Addr) and MSR[VM] = 1 then
    ESR[EC] ← 10000; ESR[S] ← 1; ESR[DIZ] ← No-access-allowed
    MSR[UMS] ← MSR[UM]; MSR[VMS] ← MSR[VM]; MSR[UM] ← 0; MSR[VM] ← 0
else if Addr[31] ≠ 0 then
    ESR[EC] ← 00001; ESR[W] ← 0; ESR[S] ← 1; ESR[Rx] ← rD
else
    Mem(Addr) ← (rD)[C_DATA_SIZE-16:C_DATA_SIZE-1]

```

変更されるレジスタ

- MSR[UM]、MSR[VM]、MSR[UMS]、MSR[VMS] (TLB ミス例外またはデータストレージ例外が生成される場合)
- ESR[EC]、ESR[S] (例外が生成される場合)
- ESR[DIZ] (データストレージ例外が生成される場合)
- ESR[W]、ESR[Rx] (非境界整列データアクセス例外が生成される場合)

レイテンシ

- 1 サイクル (C_AREA_OPTIMIZED=0 または 2)
- 2 サイクル (C_AREA_OPTIMIZED=1)

注記

MicroBlaze が再順序付け命令を使用するように設定されている場合にのみ (C_USE_REORDER_INSTR = 1)、このハーフワード反転命令は有効です。

拡張アドレス命令は、MicroBlaze が拡張アドレス命令を使用するよう設定されており (C_ADDR_SIZE > 32)、32 ビット モードを使用している場合 (C_DATA_SIZE = 32) にのみ有効です。

shi ハーフワードを格納(即値)

shi rD, rA_X, IMM

1	1	1	1	0	1	rD	rA _X	IMM
0			6		11		16	31

説明

レジスタ rD の最下位ハーフワードの内容を、レジスタ rA_X の内容と符号拡張された IMM 値を加算した結果の、ハーフワードで境界整列されたメモリロケーションに格納します。

仮想保護モードがイネーブルの場合、データ TLB ミス例外が発生し、アドレスに対応する有効な変換エントリは TLB では検出されません。仮想保護モードがイネーブルになっているとデータストレージ例外が発生し、ノーアクセスまたは読み出し専用ゾーン保護によってアクセスが禁止されます。ノーアクセス許可はユーザー モードの場合にのみ発生します。アドレスの最下位ビットがゼロでない場合は、非境界整列データアクセス例外が発生します。

擬似コード

```

Addr ← (rAX) + sext(IMM)
if TLB_Miss(Addr) and MSR[VM] = 1 then
    ESR[EC] ← 10010; ESR[S] ← 1
    MSR[UMS] ← MSR[UM]; MSR[VMS] ← MSR[VM]; MSR[UM] ← 0; MSR[VM] ← 0
else if Access_Protected(Addr) and MSR[VM] = 1 then
    ESR[EC] ← 10000; ESR[S] ← 1; ESR[DIZ] ← No-access-allowed
    MSR[UMS] ← MSR[UM]; MSR[VMS] ← MSR[VM]; MSR[UM] ← 0; MSR[VM] ← 0
else if Addr[31] ≠ 0 then
    ESR[EC] ← 00001; ESR[W] ← 0; ESR[S] ← 1; ESR[Rx] ← rD
else
    Mem(Addr) ← (rD)[C_DATA_SIZE-16:C_DATA_SIZE-1]

```

変更されるレジスタ

- MSR[UM]、MSR[VM]、MSR[Ums]、MSR[VMS] (TLB ミス例外またはデータストレージ例外が生成される場合)
- ESR[EC]、ESR[S] (例外が生成される場合)
- ESR[DIZ] (データストレージ例外が生成される場合)
- ESR[W]、ESR[Rx] (非境界整列データアクセス例外が生成される場合)

レイテンシ

- 1 サイクル (C_AREA_OPTIMIZED=0 または 2)
- 2 サイクル (C_AREA_OPTIMIZED=1)

注記:

デフォルトでは、タイプ B ストア命令は 16 ビットの IMM フィールド値を取り込み、それを符号拡張して即値オペランドとして使用します。この動作は、imm または imm1 命令を先行させると変更できます。即値の使用については、「imm」および「imm1」を参照してください。

sra**算術右シフト**

sra rD, rA

1	0	0	1	0	0	rD	rA	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
0			6			11		16																	31	

説明

rA の内容を 1 ビット右に算術シフトし、その結果を rD に配置します。rA の最上位ビット(符号ビット)は、rD の最上位ビットに配置されます。シフト チェーンから出力される最下位ビットは、キャリー フラグに配置されます。

擬似コード

```
(rD)[0] ← (rA)[0]
(rD)[1:31] ← (rA)[0:30]
MSR[C] ← (rA)[31]
```

変更されるレジスタ

- rD
- MSR[C]

レイテンシ

- 1 サイクル

src**キャリー付き右シフト**

src rD, rA

1	0	0	1	0	0	rD	rA	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
0						6		11																			31

説明

rA の内容を 1 ビット右にシフトし、その結果を rD に配置します。シフト チェーンにキャリーフラグをシフト インし、rD の最上位ビットに配置します。シフト チェーンから出力される最下位ビットは、キャリーフラグに配置されます。

擬似コード

```
(rD)[0] ← MSR[C]
(rD)[1:31] ← (rA)[0:30]
MSR[C] ← (rA)[31]
```

変更されるレジスタ

- rD
- MSR[C]

レイテンシ

- 1 サイクル

srl**論理右シフト**

srl rD, rA

1	0	0	1	0	0	rD	rA	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	1
0			6			11		16										31						

説明

rA の内容を 1 ビット右に論理シフトし、その結果を rD に配置します。シフト チェーンに 0 をシフト インし、rD の最上位ビットに配置します。シフト チェーンから出力される最下位ビットは、キャリー フラグに配置されます。

擬似コード

```
(rD)[0] ← 0
(rD)[1:31] ← (rA)[0:30]
MSR[C] ← (rA)[31]
```

変更されるレジスタ

- rD
- MSR[C]

レイテンシ

- 1 サイクル

SW**ワードを格納**

sw	rD, rA _X , rB _X
swr	rD, rA _X , rB _X
swea	rD, rA, rB

1	1	0	1	1	0	rD	rA _X	rB _X	0	R	0	EA	0	0	0	0	0	0	0
0						6		11				16			21				31

説明

レジスタ rD の内容を、レジスタ rA_X および rB_X の内容を加算した結果を、ワードで境界整列されたメモリ ロケーションに格納します。

R ビットがセットされている場合、格納するワードのバイトを反転し、E ビットで定義されているエンディアンネスとは逆のエンディアンネスでデータを格納します(仮想保護モードがイネーブルの場合)。

EA ビットがセットされている場合、拡張アドレスが使用され、rA と rB を足すのではなく連結させて形成されます。

仮想保護モードがイネーブルの場合、データ TLB ミス例外が発生し、アドレスに対応する有効な変換エントリは TLB では検出されません。

仮想保護モードがイネーブルになっているとデータストレージ例外が発生し、ノーアクセスまたは読み出し専用ゾーン保護によってアクセスが禁止されます。ノーアクセス許可はユーザー モードの場合にのみ発生します。

アドレスの最下位 2 ビットがゼロでない場合は、非境界整列データアクセス例外が発生します。

EA ビットがセットされていて、物理アドレス拡張(PAE) がイネーブルで、命令が明示的に許可されていない場合は、特権命令エラーが発生します。

擬似コード

```

if EA = 1 then
    Addr ← (rA) & (rB)
else
    Addr ← (rAX) + (rBX)
if TLB_Miss(Addr) and MSR[VM] = 1 then
    ESR[EC] ← 10010; ESR[S] ← 1
    MSR[UMS] ← MSR[UM]; MSR[VMS] ← MSR[VM]; MSR[UM] ← 0; MSR[VM] ← 0
else if Access_Protected(Addr) and MSR[VM] = 1 then
    ESR[EC] ← 10000; ESR[S] ← 1; ESR[DIZ] ← No-access-allowed
    MSR[UMS] ← MSR[UM]; MSR[VMS] ← MSR[VM]; MSR[UM] ← 0; MSR[VM] ← 0
else if Addr[30:31] ≠ 0 then
    ESR[EC] ← 00001; ESR[W] ← 1; ESR[S] ← 1; ESR[Rx] ← rD
else
    Mem(Addr) ← (rD)

```

変更されるレジスタ

- MSR[UM]、MSR[VM]、MSR[UMS]、MSR[VMS] (TLB ミス例外またはデータストレージ例外が生成される場合)
- ESR[EC]、ESR[S] (例外が生成される場合)
- ESR[DIZ] (データストレージ例外が生成される場合)
- ESR[W]、ESR[Rx] (非境界整列データアクセス例外が生成される場合)

レイテンシ

- 1 サイクル (C_AREA_OPTIMIZED=0 または 2)
- 2 サイクル (C_AREA_OPTIMIZED=1)

注記

MicroBlaze が再順序付け命令を使用するように設定されている場合にのみ (C_USE_REORDER_INSTR = 1)、このワード反転命令は有効です。

拡張アドレス命令は、MicroBlaze が拡張アドレス命令を使用するよう設定されており (C_ADDR_SIZE > 32)、32 ビットモードを使用している場合 (C_DATA_SIZE = 32) にのみ有効です。

swapb バイトのスワップ

swapb rD, rA

1	0	0	1	0	0	rD	rA	0	0	0	0	0	0	0	1	1	1	1	0	0	0	0	0	0	
0			6			11		16																31	

説明

4 バイトとして処理されたレジスタ rA の内容をスワップし、その結果を rD に配置します。リトルエンディアンからビッグエンディアン、またはその逆に、レジスタのバイトシーケンスのエンディアンネスフォーマットを変換します。

擬似コード

```
(rD)[24:31] ← (rA)[0:7]
(rD)[16:23] ← (rA)[8:15]
(rD)[8:15] ← (rA)[16:23]
(rD)[0:7] ← (rA)[24:31]
```

変更されるレジスタ

- rD

レイテンシ

- 1 サイクル

注記:

MicroBlaze が再順序付け命令を使用するように設定されている場合にのみ (C_USE_REORDER_INSTR = 1)、この命令は有効です。

swaph ハーフワードのスワップ

swaph rD, rA

1	0	0	1	0	0	rD	rA	0	0	0	0	0	0	0	1	1	1	1	0	0	0	1	0
0			6			11		16															31

説明

2 ハーフワードとして処理されたレジスタ rA の内容をスワップし、その結果を rD に配置します。リトルエンディアンからビッグエンディアン、またはその逆に、レジスタの 2 ハーフワード シーケンスのエンディアンネス フォーマットを変換します。

擬似コード

```
(rD)[0:15] ← (rA)[16:31]
(rD)[16:31] ← (rA)[0:15]
```

変更されるレジスタ

- rD

レイテンシ

- 1 サイクル

注記:

MicroBlaze が再順序付け命令を使用するように設定されている場合にのみ (C_USE_REORDER_INSTR = 1)、この命令は有効です。

swi**ワードを格納(即値)**swi rD, rA_X, IMM

1	1	1	1	1	0	rD	rA _X	IMM
0			6		11		16	31

説明

レジスタ rD の内容を、レジスタ rA_X の内容と符号拡張された IMM 値を加算した結果の、ワードで境界整列されたメモリロケーションに格納します。

仮想保護モードがイネーブルの場合、データ TLB ミス例外が発生し、アドレスに対応する有効な変換エントリは TLB では検出されません。

仮想保護モードがイネーブルになっているとデータストレージ例外が発生し、ノーアクセスまたは読み出し専用ゾーン保護によってアクセスが禁止されます。ノーアクセス許可はユーザー モードの場合にのみ発生します。

アドレスの最下位 2 ビットがゼロでない場合は、非境界整列データアクセス例外が発生します。

擬似コード

```

Addr ← (rAX) + sext(IMM)
if TLB_Miss(Addr) and MSR[VM] = 1 then
    ESR[EC] ← 10010; ESR[S] ← 1
    MSR[UMS] ← MSR[UM]; MSR[VMS] ← MSR[VM]; MSR[UM] ← 0; MSR[VM] ← 0
else if Access_Protected(Addr) and MSR[VM] = 1 then
    ESR[EC] ← 10000; ESR[S] ← 1; ESR[DIZ] ← No-access-allowed
    MSR[UMS] ← MSR[UM]; MSR[VMS] ← MSR[VM]; MSR[UM] ← 0; MSR[VM] ← 0
else if Addr[30:31] ≠ 0 then
    ESR[EC] ← 00001; ESR[W] ← 1; ESR[S] ← 1; ESR[Rx] ← rD
else
    Mem(Addr) ← (rD)

```

変更されるレジスタ

- MSR[UM]、MSR[VM]、MSR[UMS]、MSR[VMS] (TLB ミス例外またはデータストレージ例外が生成される場合)
- ESR[EC]、ESR[S] (例外が生成される場合)
- ESR[DIZ] (データストレージ例外が生成される場合)
- ESR[W]、ESR[Rx] (非境界整列データアクセス例外が生成される場合)

レイテンシ

- 1 サイクル (C_AREA_OPTIMIZED=0 または 2)
- 2 サイクル (C_AREA_OPTIMIZED=1)

注記:

デフォルトでは、タイプ B ストア命令は 16 ビットの IMM フィールド値を取り込み、それを符号拡張して即値オペランドとして使用します。この動作は、imm または imm1 命令を先行させると変更できます。即値の使用については、「[imm](#)」および「[imm1](#)」を参照してください。

SWX

ワードを格納(排他的)

SWX rD, rA, rB

1	1	0	1	1	0	rD	rA	rB	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0					6		11		16		21														31

説明

レジスタ rD の内容を、レジスタ rA および rB の内容を加算した結果の、ワードに境界整列されたメモリ ロケーションに条件付きで格納します。排他的アクセスがイネーブルになっている AXI4 インターコネクトを使用している場合、インターフェクト応答が EXOKAY で、予約ビットがセットされると、格納されます。それ以外の場合は、予約ビットがセットされると格納されます。格納されない場合は、キャリーフラグ (MSR[C]) がセットされます。それ以外の場合はキャリーフラグはクリアになります。予約ビットはクリアになります。

仮想保護モードがイネーブルの場合、データ TLB ミス例外が発生し、アドレスに対応する有効な変換エントリは TLB では検出されません。

仮想保護モードがイネーブルになっているとデータストレージ例外が発生し、ノーアクセスまたは読み出し専用ゾーン保護によってアクセスが禁止されます。ノーアクセス許可はユーザー モードの場合にのみ発生します。

アドレスの最下位 2 ビットがゼロでない場合は、非境界整列データアクセス例外は発生しません。

AXI の排他的アクセスをイネーブルにすると、ほかのバス マスターから操作が保護されますが、アドレス指定したスレーブで排他的アクセスをサポートするようにしておく必要があります。排他的アクセスがイネーブルになっていないと、内部予約ビットのみが使用されます。排他的アクセスは、ペリフェラルおよびキャッシュ インターコネクトにそれぞれ、C_M_AXI_DP_EXCLUSIVE_ACCESS および C_M_AXI_DC_EXCLUSIVE_ACCESS という 2 つのパラメーターを設定するとイネーブルになります。

擬似コード

```

Addr ← (rA) + (rB)
if Reservation = 0 then
    MSR[C] ← 1
else
    if TLB_Miss(Addr) and MSR[VM] = 1 then
        ESR[EC]← 10010; ESR[S]← 1
        MSR[UUM] ← MSR[UM]; MSR[VMS] ← MSR[VM]; MSR[UM] ← 0; MSR[VM] ← 0
    else if Access_Protected(Addr) and MSR[VM] = 1 then
        ESR[EC] ← 10000; ESR[S]← 1; ESR[DIZ] ← No-access-allowed
        MSR[UUM] ← MSR[UM]; MSR[VMS] ← MSR[VM]; MSR[UM] ← 0; MSR[VM] ← 0
    else
        Reservation ← 0
        if AXI_Exclusive(Addr) and AXI_Response ≠ EXOKAY then
            MSR[C] ← 1
        else
            Mem(Addr) ← (rD)[0:31]
            MSR[C] ← 0

```

変更されるレジスタ

- MSR[C] (予約が生成されない限り)
- MSR[UM]、MSR[VM]、MSR[UUM]、MSR[VMS] (TLB ミス例外またはデータストレージ例外が生成される場合)
- ESR[EC]、ESR[S] (例外が生成される場合)
- ESR[DIZ] (データストレージ例外が生成される場合)

レイテンシ

- 1 サイクル (C_AREA_OPTIMIZED=0 または 2)
- 2 サイクル (C_AREA_OPTIMIZED=1)

注記

この命令は、セマフォやスピノロックなど、排他的アクセスをインプリメントするために、LWX と一緒に使用されます。

キャリー フラグ (MSR[C]) はすぐにはセットされない可能性があります (パイプラインストールの動作による)。キャリー フラグの正しい値を得るため、SWX 命令は、MSRCLR、MSRSET、MTS、または SRC 命令のすぐ後に続けることはできません。

wdc データ キャッシュに書き込み

wdc	rA, rB
wdc.flush	rA, rB
wdc.clear	rA, rB
wdc.clear.ea	rA, rB
wdc.ext.flush	rA, rB
wdc.ext.clear	rA, rB

1	0	0	1	0	0	0	0	0	rA	rB	E	0	0	EA	1	1	F	0	1	T	0
0			6		11		16			21											31

説明

キャッシュ ラインを無効化またはフラッシュするため、データ キャッシュ タグに書き込みます。F ビットをセットするにはニーモニック wdc.flush を、T ビットをセットするには wdc.clear を、T ビットおよび EA ビットをセットするのに wdc.clear.ea を、E、F、T ビットをセットするには wdc.ext.flush を、E および T ビットをセットするには wdc.ext.clear を使用します。

C_DCACHE_USE_WRITEBACK が 1 に設定された場合:

- F ビットがセットされる場合は、命令がキャッシュ ラインをフラッシュし無効にします。
- それ以外の場合は、命令はキャッシュ ラインを無効にし、メモリに書き込まれていないデータを破棄するだけです。
- T ビットがセットされている場合は、一致しているアドレスのキャッシュ ラインのみが無効になります。
 - EA ビットがセットされている場合は、rB と連結しているレジスタ rA が、対象のキャッシュ ライン拡張アドレスになります。
 - それ以外の場合は、rB を加えたレジスタ rA は、対象のキャッシュ ラインのアドレスです。
 - C_ADDR_SIZE > 32 に設定されているときにのみ、EA ビットは考慮されます。
- E ビットは考慮されません。
- F ビットと T ビットを同時に使用することはできません。

C_DCACHE_USE_WRITEBACK が 0 にクリアになった場合:

- E ビットがセットされる場合は、命令がキャッシュ ラインを無効にします。レジスタ rA には関係するキャッシュ ラインのアドレスが含まれ、レジスタ rB の値は使用されません。
- それ以外の場合は、MicroBlaze から外部キャッシュの一貫しているアドレスを無効またはフラッシュ (F ビットの値による) するよう要求され、影響を受ける内部キャッシュ ラインが無効になります。レジスタ rB を加えたレジスタ rA が外部キャッシュのアドレスで、影響のあるキャッシュ ラインのものです。
- C_INTERCONNECT が 3 (ACE) に設定されているときにのみ、E ビットは考慮されます。

MicroBlaze が MMU を使用するように (C_USE_MMU >= 1) 設定されている場合、この命令は特権命令です。つまり、命令がユーザー モード (MSR[UM] = 1) で実行されると、特権命令例外が発生します。

擬似コード

```

if MSR[UM] = 1 then
    ESR[EC] ← 00111
else
    if C_DCACHE_USE_WRITEBACK = 1 then
        if T = 1 and EA = 1 then
            address ← (rA) & (rB)
        else
            address ← (rA) + (rB)
    else if E = 0 then
        address ← (rA)
    else
        address ← (rA) + (rB)
    if C_DCACHE_LINE_LEN = 4 then
        cacheline_mask ← (1 << log2(C_DCACHE_BYTE_SIZE) - 4) - 1
        cacheline ← (DCache Line)[(address >> 4) ^ cacheline_mask]
        cacheline_addr ← address & 0xffffffff0
    if C_DCACHE_LINE_LEN = 8 then
        cacheline_mask ← (1 << log2(C_DCACHE_BYTE_SIZE) - 5) - 1
        cacheline ← (DCache Line)[(address >> 5) ^ cacheline_mask]
        cacheline_addr ← address & 0xfffffff0
    if C_DCACHE_LINE_LEN = 16 then
        cacheline_mask ← (1 << log2(C_DCACHE_BYTE_SIZE) - 6) - 1
        cacheline ← (DCache Line)[(address >> 6) ^ cacheline_mask]
        cacheline_addr ← address & 0xfffffc0
    if E = 0 and F = 1 and cacheline.Dirty then
        for i = 0 .. C_DCACHE_LINE_LEN - 1 loop
            if cacheline.Valid[i] then
                Mem(cacheline_addr + i * 4) ← cacheline.Data[i]
    if T = 0 then
        cacheline.Tag ← 0
    else if cacheline.Address = cacheline_addr then
        cacheline.Tag ← 0
    if E = 1 then
        if F = 1 then
            request external cache flush with address
        else
            request external cache invalidate with address

```

変更されるレジスタ

- ESR[EC] (特権命令例外が生成される場合)

レイテンシ

- 2 サイクル (wdc.clear)
- 2 サイクル (wdc、C_AREA_OPTIMIZED=0 または 2)
- 3 サイクル (wdc、C_AREA_OPTIMIZED=0)
- 2+N サイクル (wdc.flush。N は随時メモリにキャッシュ ラインをフラッシュするのに必要なクロック サイクル数)

注記

wdc、wdc.flush、wdc.clear、および wdc.clear.ea 命令は、データ キャッシュ イネーブル (MSR[DCE]) からは独立していて、データ キャッシュがイネーブルになっていてもディスエーブルになっていても使用できます。

wdc.clear および wdc.clear.ea 命令は、ダイレクト メモリ アクセス デバイスによって書き込まれるバッファーなど、メモリの特定エリアを無効化するためのものです。

この命令を使用することにより、ほかのキャッシュ ラインが誤って無効化されたり、メモリにまだ書き込まれていないデータを破棄してしまわないようにできます。

パラメーター C_USE_MMU の設定にかかわらず、また MMU が仮想モードであってもリアルモードであっても、影響するキャッシュ ラインのアドレスは常に物理アドレスです。

キャッシュ全体をフラッシュするのにループで wdc.flush を使用している場合、キャッシュベースアドレスに rA、ループ カウンターに rB を使用することにより、ループを最適化できます。

```
addik      r5,r0,C_DCACHE_BASEADDR
addik      r6,r0,C_DCACHE_BYTE_SIZE-C_DCACHE_LINE_LEN*4
loop:   wdc.flush r5,r6
        bgtid    r6,loop
        addik    r6,r6,-C_DCACHE_LINE_LEN*4
```

キャッシュのあるメモリ エリアをするのにループで wdc.clear を使用している場合、メモリ エリアベースアドレスに rA、ループ カウンターに rB を使用することにより、ループを最適化できます。

```
addik      r5,r0,memory_area_base_address
addik      r6,r0,memory_area_byte_size-C_DCACHE_LINE_LEN*4
loop:   wdc.clear r5,r6
        bgtid    r6,loop
        addik    r6,r6,-C_DCACHE_LINE_LEN*4
```

wic**命令キヤッショに書き込み**

wic rA、rB

1	0	0	1	0	0	0	0	0	0	0	0	rA	rB	0	0	0	0	1	1	0	1	0	0	0
0			6			11			16			21									31			

説明

キヤッショ ラインを無効化するため、命令キヤッショ タグに書き込みます。レジスタ rB の値は使用されません。レジスタ rA には関係するキヤッショ ラインのアドレスが含まれます。

MicroBlaze が MMU を使用するように (C_USE_MMU >= 1) 設定されている場合、この命令は特権命令です。つまり、命令がユーザー モード (MSR[UM] = 1) で実行されると、特権命令例外が発生します。

擬似コード

```

if MSR[UM] = 1 then
    ESR[EC] ← 00111
else
    if C_ICACHE_LINE_LEN = 4 then
        cacheline_mask ← (1 << log2(C_CACHE_BYTE_SIZE) - 4) - 1
        (ICache Line)[((Ra) >> 4) ^ cacheline_mask].Tag ← 0
    if C_ICACHE_LINE_LEN = 8 then
        cacheline_mask ← (1 << log2(C_CACHE_BYTE_SIZE) - 5) - 1
        (ICache Line)[((Ra) >> 5) ^ cacheline_mask].Tag ← 0
    if C_ICACHE_LINE_LEN = 16 then
        cacheline_mask ← (1 << log2(C_CACHE_BYTE_SIZE) - 6) - 1
        (ICache Line)[((Ra) >> 6) ^ cacheline_mask].Tag ← 0

```

変更されるレジスタ

- ESR[EC] (特権命令例外が生成される場合)

レイテンシ

- 2 サイクル

注記

WIC 命令は、命令キヤッショ イネーブル (MSR[ICE]) からは独立していて、命令キヤッショ がイネーブルになっていてもディスエーブルになっていても使用できます。

パラメーターが C_USE_MMU = 3 で MMU が仮想モードのとき、影響を受けるキヤッショ ラインのアドレスは仮想アドレスになります。それ以外のときは物理アドレスです。

xor**排他的論理 OR**

xor rD, rA, rB

1	0	0	0	1	0	rD	rA	rB	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0					6		11		16		21															31

説明

レジスタ rA の内容とレジスタ rB の内容を XOR 演算し、その結果をレジスタ rD に配置します。

擬似コード
$$(rD) \leftarrow (rA) \oplus (rB)$$
変更されるレジスタ

- rD

レイテンシ

- 1 サイクル

xori**即値との排他的論理 OR**

xori rD, rA, IMM

1	0	1	0	1	0	rD	rA	IMM
0			6		11		16	31

説明

IMM フィールドの左側に 16 個の 0 ビットを連結して 32 ビットに拡張します。レジスタ rA の内容を拡張された IMM フィールドと XOR 演算し、その結果をレジスタ rD に配置します。

擬似コード

$$(rD) \leftarrow (rA) \oplus \text{sext(imm)}$$
変更されるレジスタ

- rD

レイテンシ

- 1 サイクル

注記

デフォルトでは、タイプ B 命令は 16 ビットの IMM フィールド値を取り込み、それを 32 ビットに符号拡張して即値オペランドとして使用します。この動作は、タイプ B 命令の前に imm 命令を先行させると変更できます。32 ビットの即値の使用については、「[imm](#)」を参照してください。

この命令を rD が r0 に設定された状態で使用すると、プログラムトレースイベントが結果の最下位 14 ビットを使用して発生します。これは、通常コンテキストスイッチやシステムコールなどのオペレーティングシステムイベントをトレースするために使用されますが、プログラムで重要なイベントをトレースするためにも使用できます。この機能は、C_DEBUG_ENABLED = 2 (拡張) および C_DEBUG_TRACE_SIZE > 0 に設定すると有効になります。詳細は、[第2章の「プログラムおよびイベントトレース」](#) を参照してください。

MicroBlaze 64 ビット命令

このセクションでは、64 ビット MicroBlaze の命令セットに含まれる追加の命令の定義を示します。

これらの命令は 64 ビット レジスタ全体を使用し、long 算術および論理演算を実行します。

すべてのタイプ B 64 ビット算術および論理命令には、64 ビット命令であることを示すため、IMML 命令を先行させる必要があります。64 ビットの即値の使用については、「[imml](#)」を参照してください。

拡張命令セットでは、倍精度浮動小数点命令も定義されています。

addl long の算術加算

addl	rD _L , rA _L , rB _L	long の加算
addlc	rD _L , rA _L , rB _L	long のキャリー付き加算
addlk	rD _L , rA _L , rB _L	long の加算およびキャリー保持
addlkc	rD _L , rA _L , rB _L	long のキャリー付き加算およびキャリー保持

0 0 0 K C 0	rD _L	rA _L	rB _L	0 0 1 0 0 0 0 0 0 0 0					
0	6	11	16	21	31				

説明

レジスタ rA_L と rB_L の内容の和を、レジスタ rD_L に配置します。

命令のビット 3(図の K)は、ニーモニック addlk に対して 1 にセットされます。命令のビット 4(図の C)は、ニーモニック addlc に対して 1 にセットされます。ニーモニック addlkc の場合は、両方のビットが 1 にセットされます。

addl 命令のビット 3 がセットされている場合 (addlk, addlkc)、命令の実行結果にかかわらず、キャリー フラグは前の値を保持します。ビット 3 がクリアされている場合 (addl, addlc)、キャリー フラグは命令の実行結果によって変更されます。

命令のビット 4 が 1 にセットされている場合 (addlc, addlkc)、キャリー フラグの内容 (MSR[C]) が命令の実行に影響します。ビット 4 がクリアされている場合 (addl, addlk)、キャリー フラグの内容は命令の実行には影響しません(標準加算の場合)。

擬似コード

```
if C = 0 then
    (rDL) ← (rAL) + (rBL)
else
    (rDL) ← (rAL) + (rBL) + MSR[C]
if K = 0 then
    MSR[C] ← CarryOut64
```

変更されるレジスタ

- rD_L
- MSR[C]

レイテンシ

1 サイクル

注記

命令オペコードの C ビットは、MSB のキャリー ビットとは異なります。

addli long と即値の算術加算

addli	rD _L , rA _L , IMM	long と即値の加算
addlic	rD _L , rA _L , IMM	long と即値のキャリー付き加算
addlik	rD _L , rA _L , IMM	long と即値の加算およびキャリー保持
addlike	rD _L , rA _L , IMM	long と即値のキャリー付き加算およびキャリー保持

0	0	1	K	C	0	rD _L	rA _L	IMM
0			6		11		16	31

説明

レジスタ rA_L の内容と、IMM フィールドの値を先行する imml 命令からの即値で拡張した値の和を、レジスタ rD_L に配置します。命令のビット 3(図の K) は、二一モニック addik に対して 1 にセットされます。命令のビット 4(図の C) は、二一モニック addlic に対して 1 にセットされます。二一モニック addlike の場合は、両方のビットが 1 にセットされます。

addli 命令でビット 3 がセットされている場合 (addlik, addlike)、命令の実行結果にかかわらず、キャリーフラグは前の値を保持します。ビット 3 がクリアされている場合 (addli, addlic)、キャリーフラグは命令の実行結果によって変更されます。

命令のビット 4 が 1 にセットされている場合 (addlic, addlike)、キャリーフラグの内容 (MSR[C]) が命令の実行に影響します。ビット 4 がクリアされている場合 (addli, addlik)、キャリーフラグの内容は命令の実行には影響しません (標準加算の場合)。

擬似コード

```

if C = 0 then
    (rDL) ← (rAL) + sext(IMM)
else
    (rDL) ← (rAL) + sext(IMM) + MSR[C]
if K = 0 then
    MSR[C] ← CarryOut64

```

変更されるレジスタ

- rD_L
- MSR[C]

レイテンシ

1 サイクル

注記

命令オペコードの C ビットは、MSB のキャリービットとは異なります。

タイプ B の算術 long 命令には、imml 命令を先行させる必要があります。long 即値の使用については、「[imml](#)」を参照してください。

andl**long の論理 AND**andl rD_L, rA_L, rB_L

1	0	0	0	0	1	rD _L	rA _L	rB _L	0	0	1	0	0	0	0	0	0	0	0	0	0	
0					6		11		16		21										31	

説明

レジスタ rA_L の内容とレジスタ rB_L の内容を AND 演算し、その結果をレジスタ rD_L に配置します。

擬似コード
$$(rD_L) \leftarrow (rA_L) \wedge (rB_L)$$
変更されるレジスタ

- rD_L

レイテンシ

1 サイクル

andli long と即値の論理 AND

andli rD_L, rA_L, IMM

1	0	1	0	0	1	rD _L	rA _L	IMM
0			6		11		16	31

説明

レジスタ rA_L の内容と、IMM フィールドの値を先行する imml 命令からの即値で拡張した値を AND 演算し、その結果をレジスタ rD_L に配置します。

擬似コード

```
(rDL) ← (rAL) ∧ sext(IMM)
```

変更されるレジスタ

- rD_L

レイテンシ

1 サイクル

注記:

タイプ B の論理 long 命令には、imml 命令を先行させる必要があります。long 即値の使用については、「[imml](#)」を参照してください。

andnl long の論理 AND NOT

 andnl rD_L, rA_L, rB_L

1	0	0	0	1	1	rD _L	rA _L	rB _L	0	0	1	0	0	0	0	0	0	0	0	0
0				6		11		16		21										31

説明

レジスタ rA_L の内容をレジスタ rB_L の内容の論理補数と AND 演算し、その結果をレジスタ rD_L に配置します。

擬似コード

$$(rD_L) \leftarrow (rA_L) \wedge (\overline{rB_L})$$
変更されるレジスタ

- rD_L

レイテンシ

1 サイクル

andnli long と即値の論理 AND NOTandnli rD_L, rA_L, IMM

1	0	1	0	1	1	rD _L	rA _L	IMM
0			6		11		16	31

説明

IMM フィールドは、先行する imml 命令からの即値で符号拡張されます。レジスタ rA_L の内容と拡張された IMM フィールドの論理補数を AND 演算し、その結果をレジスタ rD_L に配置します。

擬似コード

$$(rD_L) \leftarrow (rA_L) \wedge \overline{(\text{sext(IMM)})}$$
変更されるレジスタ

- rD_L

レイテンシ

1 サイクル

注記:

タイプ B の論理 long 命令には、imml 命令を先行させる必要があります。long 即値の使用については、「[imml](#)」を参照してください。

beaeq 0 の場合は拡張アドレスに分岐

beaeq	rA, rB _L	0 の場合は拡張アドレスに分岐
bealeq	rA _L , rB _L	long が 0 の場合は拡張アドレスに分岐
beaeqd	rA, rB _L	0 の場合は遅延後に拡張アドレスに分岐
bealeqd	rA _L , rB _L	long が 0 の場合は遅延後に拡張アドレスに分岐

1 0 0 1 1 1	D 1 0 0 0	rA _L	rB _L	0 0 L 0 0 0 0 0 0 0 0 0
0	6	11	16	21 31

説明

rA または rA_L が 0 の場合、rB_L のオフセット値にある命令に分岐します。分岐先は、アドレス PC + rB_L にある命令です。

二モニック bealeq および bealeqd は、L ビットをセットします。L ビットがセットされている場合は rA_L を使用した long 比較が実行され、それ以外の場合は rA を使用した 32 ビット比較が実行されます。

二モニック beaeqd および bealeqd は、D ビットをセットします。D ビットは、分岐遅延スロットの有無を指定します。D ビットがセットされている場合は遅延スロットがあり、分岐先の命令を実行する前に、分岐に後続する命令(分岐遅延スロットにある命令)を実行できます。D ビットがセットされていない場合は、遅延スロットはなく、分岐後に実行される命令は分岐先の命令になります。

擬似コード

```

if L = 1 and rAL = 0 then
    PC ← PC + rBL
else if rA = 0 then
    PC ← PC + rBL
else
    PC ← PC + 4
if D = 1 then
    allow following instruction to complete execution

```

変更されるレジスタ

- PC

レイテンシ

- 1 サイクル(分岐が取られなかった場合)
- 2 サイクル(分岐が取られ、D ビットがセットされている場合)
- 3 サイクル(分岐が取られ、D ビットがセットされていない場合)

注記:

遅延スロットには、imm、imml、分岐、またはブレーク命令を含めることはできません。割り込みおよび外部ハードウェアブレークは、遅延スロット分岐が完了するまで保留されます。

beaeqi 0 の場合は拡張アドレス即値に分岐

beaeqi	rA, IMM	0 の場合は拡張アドレス即値に分岐
beaeqid	rA, IMM	0 の場合は遅延後に拡張アドレス即値に分岐

1	0	1	1	1	1	D	1	0	0	0	rA _L		IMM		
0						6					11		16		31

説明

rA または rA_L が 0 の場合、IMM の値を先行する imm または imml 命令からの即値で拡張したオフセット値にある命令に分岐します。分岐先は、アドレス PC + IMM にある命令です。

先行する命令が imml 命令の場合は rA_L を使用した long 比較が実行され、それ以外の場合は rA を使用した 32 ビット比較が実行されます。

ニーモニック beaeqid は D ビットをセットします。D ビットは、分岐遅延スロットの有無を指定します。D ビットがセットされている場合は遅延スロットがあり、分岐先の命令を実行する前に、分岐に後続する命令(分岐遅延スロットにある命令)を実行できます。D ビットがセットされていない場合は、遅延スロットではなく、分岐後に実行される命令は分岐先の命令になります。

擬似コード

```
If (preceded by imml) and rAL = 0 then
    PC ← PC + sext(IMM)
else if rA = 0 then
    PC ← PC + sext(IMM)
else
    PC ← PC + 4
if D = 1 then
    allow following instruction to complete execution
```

変更されるレジスタ

- PC

レイテンシ

- 1 サイクル(分岐が取られなかった場合、または正しく分岐予測された場合)
- 2 サイクル(分岐が取られ、D ビットがセットされている場合)
- 3 サイクル(分岐が取られ、D ビットがセットされていない場合、または C_AREA_OPTIMIZED=0 で分岐予測が間違っている場合)
- 7 ~ 9 サイクル(C_AREA_OPTIMIZED=2 で分岐予測が間違っている場合)

注記

デフォルトでは、タイプ B 分岐 long 命令は 16 ビットの IMM フィールド値を取り込み、それを 64 ビットに符号拡張して、即値オペランドとして使用します。この動作は、タイプ B 命令に imm または imml 命令を先行させると変更できます。64 ビットの即値の使用については、「[imm](#)」および「[imml](#)」を参照してください。

アセンブラー疑似命令 beaeqi および bealeqid は、long 比較を示すために使用されます。

遅延スロットには、imm、imml、分岐、またはブレーク命令を含めることはできません。割り込みおよび外部ハードウェアブレークは、遅延スロット分岐が完了するまで保留されます。

beage 0以上の場合は拡張アドレスに分岐

beage	rA, rB _L	0以上の場合は拡張アドレスに分岐
bealge	rA _L , rB _L	longが0以上の場合は拡張アドレスに分岐
beaged	rA, rB _L	0以上の場合は遅延後に拡張アドレスに分岐
bealged	rA _L , rB _L	longが0以上の場合は遅延後に拡張アドレスに分岐

1 0 0 1 1 1	D 1 1 0 1	rA _L	rB _L	0 0 L 0 0 0 0 0 0 0 0 0	
0	6	11	16	21	31

説明

rA または rA_L が 0 以上の場合、rB_L のオフセット値にある命令に分岐します。分岐先は、アドレス PC + rB_L にある命令です。

二モニック bealge および bealged は、L ビットをセットします。L ビットがセットされている場合は rA_L を使用した long 比較が実行され、それ以外の場合は rA を使用した 32 ビット比較が実行されます。

二モニック beaged および bealged は、D ビットをセットします。D ビットは、分岐遅延スロットの有無を指定します。D ビットがセットされている場合は遅延スロットがあり、分岐先の命令を実行する前に、分岐に後続する命令(分岐遅延スロットにある命令)を実行できます。D ビットがセットされていない場合は、遅延スロットはなく、分岐後に実行される命令は分岐先の命令になります。

擬似コード

```
if L = 1 and rAL >= 0 then
    PC ← PC + rBL
else if rA >= 0 then
    PC ← PC + rBL
else
    PC ← PC + 4
if D = 1 then
    allow following instruction to complete execution
```

変更されるレジスタ

- PC

レイテンシ

- 1 サイクル(分岐が取られなかった場合)
- 2 サイクル(分岐が取られ、D ビットがセットされている場合)
- 3 サイクル(分岐が取られ、D ビットがセットされていない場合)

注記:

遅延スロットには、imm、imml、分岐、またはブレーク命令を含めることはできません。割り込みおよび外部ハードウェアブレークは、遅延スロット分岐が完了するまで保留されます。

beagei 0以上の場合は拡張アドレス即値に分岐

beagei	rA, IMM	0以上の場合は拡張アドレス即値に分岐
beageid	rA, IMM	0以上の場合は遅延後に拡張アドレス即値に分岐

1	0	1	1	1	1	D	1	1	0	1	rA _L		IMM
0						6					11	16	31

説明

rA または rA_L が 0 以上の場合、IMM の値を先行する imm または imml 命令からの即値で拡張したオフセット値にある命令に分岐します。分岐先は、アドレス PC + IMM にある命令です。

先行する命令が imml 命令の場合は rA_L を使用した long 比較が実行され、それ以外の場合は rA を使用した 32 ビット比較が実行されます。

ニーモニック beaqid は D ビットをセットします。D ビットは、分岐遅延スロットの有無を指定します。D ビットがセットされている場合は遅延スロットがあり、分岐先の命令を実行する前に、分岐に後続する命令(分岐遅延スロットにある命令)を実行できます。D ビットがセットされていない場合は、遅延スロットではなく、分岐後に実行される命令は分岐先の命令になります。

擬似コード

```
If (preceded by imml) and rAL >= 0 then
    PC ← PC + sext(IMM)
else if rA >= 0 then
    PC ← PC + sext(IMM)
else
    PC ← PC + 4
if D = 1 then
    allow following instruction to complete execution
```

変更されるレジスタ

- PC

レイテンシ

- 1 サイクル(分岐が取られなかった場合、または正しく分岐予測された場合)
- 2 サイクル(分岐が取られ、D ビットがセットされている場合)
- 3 サイクル(分岐が取られ、D ビットがセットされていない場合、または C_AREA_OPTIMIZED=0 で分岐予測が間違っている場合)
- 7 ~ 9 サイクル(C_AREA_OPTIMIZED=2 で分岐予測が間違っている場合)

注記

デフォルトでは、タイプ B 分岐 long 命令は 16 ビットの IMM フィールド値を取り込み、それを 64 ビットに符号拡張して、即値オペランドとして使用します。この動作は、タイプ B 命令に imm または imml 命令を先行させると変更できます。64 ビットの即値の使用については、「imm」および「imml」を参照してください。

アセンブラー疑似命令 bealgei および bealgeid は、long 比較を示すために使用されます。

遅延スロットには、imm、imml、分岐、またはブレーク命令を含めることはできません。割り込みおよび外部ハードウェアブレークは、遅延スロット分岐が完了するまで保留されます。

beagt 0より大きい場合は拡張アドレスに分岐

beagt	rA, rB _L	0より大きい場合は拡張アドレスに分岐
bealgt	rA _L , rB _L	longが0より大きい場合は拡張アドレスに分岐
beagtd	rA, rB _L	0より大きい場合は遅延後に拡張アドレスに分岐
bealgtd	rA _L , rB _L	longが0より大きい場合は遅延後に拡張アドレスに分岐

1 0 0 1 1 1	D 1 1 0 0	rA _L	rB _L	0 0 L 0 0 0 0 0 0 0 0 0
0	6	11	16	21 31

説明

rA または rA_L が 0 より大きい場合、rB_L のオフセット値にある命令に分岐します。分岐先は、アドレス PC + rB_L にある命令です。

二モニック bealgt および bealgtd は、L ビットをセットします。L ビットがセットされている場合は rA_L を使用した long 比較が実行され、それ以外の場合は rA を使用した 32 ビット比較が実行されます。

二モニック beagtd および bealgtd は、D ビットをセットします。D ビットは、分岐遅延スロットの有無を指定します。D ビットがセットされている場合は遅延スロットがあり、分岐先の命令を実行する前に、分岐に後続する命令(分岐遅延スロットにある命令)を実行できます。D ビットがセットされていない場合は、遅延スロットはなく、分岐後に実行される命令は分岐先の命令になります。

擬似コード

```

if L = 1 and rAL > 0 then
    PC ← PC + rBL
else if rA > 0 then
    PC ← PC + rBL
else
    PC ← PC + 4
if D = 1 then
    allow following instruction to complete execution

```

変更されるレジスタ

- PC

レイテンシ

- 1サイクル(分岐が取られなかった場合)
- 2サイクル(分岐が取られ、D ビットがセットされている場合)
- 3サイクル(分岐が取られ、D ビットがセットされていない場合)

注記:

遅延スロットには、imm、imml、分岐、またはブレーク命令を含めることはできません。割り込みおよび外部ハードウェアブレークは、遅延スロット分岐が完了するまで保留されます。

beagti 0より大きい場合は拡張アドレス即値に分岐

beagti	rA, IMM	0より大きい場合は拡張アドレス即値に分岐
beagtid	rA, IMM	0より大きい場合は遅延後に拡張アドレス即値に分岐

1 0 1 1 1	D 1 1 0 0	rA _L	IMM
0	6	11	16

説明

rA または rA_L が 0 より大きい場合、IMM の値を先行する imm または imml 命令からの即値で拡張したオフセット値にある命令に分岐します。分岐先は、アドレス PC + IMM にある命令です。

先行する命令が imml 命令の場合は rA_L を使用した long 比較が実行され、それ以外の場合は rA を使用した 32 ビット比較が実行されます。

ニーモニック beagtid は D ビットをセットします。D ビットは、分岐遅延スロットの有無を指定します。D ビットがセットされている場合は遅延スロットがあり、分岐先の命令を実行する前に、分岐に後続する命令(分岐遅延スロットにある命令)を実行できます。D ビットがセットされていない場合は、遅延スロットではなく、分岐後に実行される命令は分岐先の命令になります。

擬似コード

```
If (preceded by imml) and rAL > 0 then
    PC ← PC + sext(IMM)
else if rA > 0 then
    PC ← PC + sext(IMM)
else
    PC ← PC + 4
if D = 1 then
    allow following instruction to complete execution
```

変更されるレジスタ

- PC

レイテンシ

- 1 サイクル(分岐が取られなかった場合、または正しく分岐予測された場合)
- 2 サイクル(分岐が取られ、D ビットがセットされている場合)
- 3 サイクル(分岐が取られ、D ビットがセットされていない場合、または C_AREA_OPTIMIZED=0 で分岐予測が間違っている場合)
- 7 ~ 9 サイクル(C_AREA_OPTIMIZED=2 で分岐予測が間違っている場合)

注記

デフォルトでは、タイプ B 分岐 long 命令は 16 ビットの IMM フィールド値を取り込み、それを 64 ビットに符号拡張して、即値オペランドとして使用します。この動作は、タイプ B 命令に imm または imml 命令を先行させると変更できます。64 ビットの即値の使用については、「[imm](#)」および「[imml](#)」を参照してください。

アセンブラー疑似命令 bealgti および beagtid は、long 比較を示すために使用されます。

遅延スロットには、imm、imml、分岐、またはブレーク命令を含めることはできません。割り込みおよび外部ハードウェアブレークは、遅延スロット分岐が完了するまで保留されます。

beale 0以下の場合は拡張アドレスに分岐

beale	rA, rB _L	0以下の場合は拡張アドレスに分岐
bealle	rA _L , rB _L	longが0以下の場合は拡張アドレスに分岐
bealed	rA, rB _L	0以下の場合は遅延後に拡張アドレスに分岐
bealled	rA _L , rB _L	longが0以下の場合は遅延後に拡張アドレスに分岐

1 0 0 1 1 1	D 1 0 1 1	rA _L	rB _L	0 0 L 0 0 0 0 0 0 0 0
0	6	11	16	21 31

説明

rA または rA_L が 0以下の場合、rB_L のオフセット値にある命令に分岐します。分岐先は、アドレス PC + rB_L にある命令です。

二モニック bealle および bealled は、L ビットをセットします。L ビットがセットされている場合は rA_L を使用した long 比較が実行され、それ以外の場合は rA を使用した 32 ビット比較が実行されます。

二モニック bealed および bealled は、D ビットをセットします。D ビットは、分岐遅延スロットの有無を指定します。D ビットがセットされている場合は遅延スロットがあり、分岐先の命令を実行する前に、分岐に後続する命令(分岐遅延スロットにある命令)を実行できます。D ビットがセットされていない場合は、遅延スロットはなく、分岐後に実行される命令は分岐先の命令になります。

擬似コード

```
if L = 1 and rAL <= 0 then
    PC ← PC + rBL
else if rA <= 0 then
    PC ← PC + rBL
else
    PC ← PC + 4
if D = 1 then
    allow following instruction to complete execution
```

変更されるレジスタ

- PC

レイテンシ

- 1サイクル(分岐が取られなかった場合)
- 2サイクル(分岐が取られ、D ビットがセットされている場合)
- 3サイクル(分岐が取られ、D ビットがセットされていない場合)

注記:

遅延スロットには、imm、imml、分岐、またはブレーク命令を含めることはできません。割り込みおよび外部ハードウェアブレークは、遅延スロット分岐が完了するまで保留されます。

bealei 0以下の場合は拡張アドレス即値に分岐

bealei	rA, IMM	0以下の場合は拡張アドレス即値に分岐
bealleid	rA, IMM	0以下の場合は遅延後に拡張アドレス即値に分岐

1	0	1	1	1	1	D	1	0	1	1	rA _L		IMM
0						6					11	16	31

説明

rA または rA_L が 0以下の場合、IMM の値を先行する imm または imml 命令からの即値で拡張したオフセット値にある命令に分岐します。分岐先は、アドレス PC + IMM にある命令です。

先行する命令が imml 命令の場合は rA_L を使用した long 比較が実行され、それ以外の場合は rA を使用した 32 ビット比較が実行されます。

ニーモニック bealleid は D ビットをセットします。D ビットは、分岐遅延スロットの有無を指定します。D ビットがセットされている場合は遅延スロットがあり、分岐先の命令を実行する前に、分岐に後続する命令(分岐遅延スロットにある命令)を実行できます。D ビットがセットされていない場合は、遅延スロットではなく、分岐後に実行される命令は分岐先の命令になります。

擬似コード

```
If (preceded by imml) and rAL <= 0 then
    PC ← PC + sext(IMM)
else if rA <= 0 then
    PC ← PC + sext(IMM)
else
    PC ← PC + 4
if D = 1 then
    allow following instruction to complete execution
```

変更されるレジスタ

- PC

レイテンシ

- 1 サイクル(分岐が取られなかった場合、または正しく分岐予測された場合)
- 2 サイクル(分岐が取られ、D ビットがセットされている場合)
- 3 サイクル(分岐が取られ、D ビットがセットされていない場合、または C_AREA_OPTIMIZED=0 で分岐予測が間違っている場合)
- 7 ~ 9 サイクル(C_AREA_OPTIMIZED=2 で分岐予測が間違っている場合)

注記

デフォルトでは、タイプ B 分岐 long 命令は 16 ビットの IMM フィールド値を取り込み、それを 64 ビットに符号拡張して、即値オペランドとして使用します。この動作は、タイプ B 命令に imm または imml 命令を先行させると変更できます。64 ビットの即値の使用については、「imm」および「imml」を参照してください。

アセンブラー疑似命令 bealei および bealleid は、long 比較を示すために使用されます。

遅延スロットには、imm、imml、分岐、またはブレーク命令を含めることはできません。割り込みおよび外部ハードウェアブレークは、遅延スロット分岐が完了するまで保留されます。

bealt 0より小さい場合は拡張アドレスに分岐

bealt	rA, rB _L	0より小さい場合は拡張アドレスに分岐
beallt	rA _L , rB _L	longが0より小さい場合は拡張アドレスに分岐
bealtd	rA, rB _L	0より小さい場合は遅延後に拡張アドレスに分岐
bealltd	rA _L , rB _L	longが0より小さい場合は遅延後に拡張アドレスに分岐

1 0 0 1 1 1	D 1 0 1 0	rA _L	rB _L	0 0 L 0 0 0 0 0 0 0 0 0	
0	6	11	16	21	31

説明

rA または rA_L が 0 より小さい場合、rB_L のオフセット値にある命令に分岐します。分岐先は、アドレス PC + rB_L にある命令です。

二モニック beallt および bealltd は、L ビットをセットします。L ビットがセットされている場合は rA_L を使用した long 比較が実行され、それ以外の場合は rA を使用した 32 ビット比較が実行されます。

二モニック bealtd および bealltd は、D ビットをセットします。D ビットは、分岐遅延スロットの有無を指定します。D ビットがセットされている場合は遅延スロットがあり、分岐先の命令を実行する前に、分岐に後続する命令(分岐遅延スロットにある命令)を実行できます。D ビットがセットされていない場合は、遅延スロットはなく、分岐後に実行される命令は分岐先の命令になります。

擬似コード

```
if L = 1 and rAL < 0 then
    PC ← PC + rBL
else if rA < 0 then
    PC ← PC + rBL
else
    PC ← PC + 4
if D = 1 then
    allow following instruction to complete execution
```

変更されるレジスタ

- PC

レイテンシ

- 1サイクル(分岐が取られなかった場合)
- 2サイクル(分岐が取られ、D ビットがセットされている場合)
- 3サイクル(分岐が取られ、D ビットがセットされていない場合)

注記:

遅延スロットには、imm、imml、分岐、またはブレーク命令を含めることはできません。割り込みおよび外部ハードウェアブレークは、遅延スロット分岐が完了するまで保留されます。

bealti 0より小さい場合は拡張アドレス即値に分岐

bealti	rA, IMM	0より小さい場合は拡張アドレス即値に分岐
bealtid	rA, IMM	0より小さい場合は遅延後に拡張アドレス即値に分岐

1	0	1	1	1	1	D	1	0	1	0	rA _L		IMM
0						6					11	16	31

説明

rA または rA_L が 0 より小さい場合、IMM の値を先行する imm または imml 命令からの即値で拡張したオフセット値にある命令に分岐します。分岐先は、アドレス PC + IMM にある命令です。

先行する命令が imml 命令の場合は rA_L を使用した long 比較が実行され、それ以外の場合は rA を使用した 32 ビット比較が実行されます。

ニーモニック bealtd は D ビットをセットします。D ビットは、分岐遅延スロットの有無を指定します。D ビットがセットされている場合は遅延スロットがあり、分岐先の命令を実行する前に、分岐に後続する命令(分岐遅延スロットにある命令)を実行できます。D ビットがセットされていない場合は、遅延スロットではなく、分岐後に実行される命令は分岐先の命令になります。

擬似コード

```
If (preceded by imml) and rAL < 0 then
    PC ← PC + sext(IMM)
else if rA < 0 then
    PC ← PC + sext(IMM)
else
    PC ← PC + 4
if D = 1 then
    allow following instruction to complete execution
```

変更されるレジスタ

- PC

レイテンシ

- 1 サイクル(分岐が取られなかった場合、または正しく分岐予測された場合)
- 2 サイクル(分岐が取られ、D ビットがセットされている場合)
- 3 サイクル(分岐が取られ、D ビットがセットされていない場合、または C_AREA_OPTIMIZED=0 で分岐予測が間違っている場合)
- 7 ~ 9 サイクル(C_AREA_OPTIMIZED=2 で分岐予測が間違っている場合)

注記

デフォルトでは、タイプ B 分岐 long 命令は 16 ビットの IMM フィールド値を取り込み、それを 64 ビットに符号拡張して、即値オペランドとして使用します。この動作は、タイプ B 命令に imm または imml 命令を先行させると変更できます。64 ビットの即値の使用については、「[imm](#)」および「[imml](#)」を参照してください。

アセンブラー疑似命令 bealti および bealtd は、long 比較を示すために使用されます。

遅延スロットには、imm、imml、分岐、またはブレーク命令を含めることはできません。割り込みおよび外部ハードウェアブレークは、遅延スロット分岐が完了するまで保留されます。

beane 0でない場合は拡張アドレスに分岐

beane	rA, rB _L	0でない場合は拡張アドレスに分岐
bealne	rA _L , rB _L	longが0でない場合は拡張アドレスに分岐
beaned	rA, rB _L	0でない場合は遅延後に拡張アドレスに分岐
bealned	rA _L , rB _L	longが0でない場合は遅延後に拡張アドレスに分岐

1 0 0 1 1 1	D 1 0 0 1	rA _L	rB _L	0 0 L 0 0 0 0 0 0 0 0 0
0	6	11	16	21 31

説明

rA または rA_L が 0 でない場合、rB_L のオフセット値にある命令に分岐します。分岐先は、アドレス PC + rB_L にある命令です。

二モニック bealne および bealned は、L ビットをセットします。L ビットがセットされている場合は rA_L を使用した long 比較が実行され、それ以外の場合は rA を使用した 32 ビット比較が実行されます。

二モニック beaned および bealned は、D ビットをセットします。D ビットは、分岐遅延スロットの有無を指定します。D ビットがセットされている場合は遅延スロットがあり、分岐先の命令を実行する前に、分岐に後続する命令(分岐遅延スロットにある命令)を実行できます。D ビットがセットされていない場合は、遅延スロットはなく、分岐後に実行される命令は分岐先の命令になります。

擬似コード

```

if L = 1 and rAL ≠ 0 then
    PC ← PC + rBL
else if rA ≠ 0 then
    PC ← PC + rBL
else
    PC ← PC + 4
if D = 1 then
    allow following instruction to complete execution

```

変更されるレジスタ

- PC

レイテンシ

- 1 サイクル(分岐が取られなかった場合)
- 2 サイクル(分岐が取られ、D ビットがセットされている場合)
- 3 サイクル(分岐が取られ、D ビットがセットされていない場合)

注記:

遅延スロットには、imm、imml、分岐、またはブレーク命令を含めることはできません。割り込みおよび外部ハードウェアブレークは、遅延スロット分岐が完了するまで保留されます。

beanei 0でない場合は拡張アドレス即値に分岐

beanei	rA, IMM	0でない場合は拡張アドレス即値に分岐
beaneid	rA, IMM	0でない場合は遅延後に拡張アドレス即値に分岐

1	0	1	1	1	1	D	1	0	0	1		rA _L		IMM
0						6					11		16	

説明

rA または rA_L が 0 でない場合、IMM の値を先行する imm または imml 命令からの即値で拡張したオフセット値にある命令に分岐します。分岐先は、アドレス PC + IMM にある命令です。

先行する命令が imml 命令の場合は rA_L を使用した long 比較が実行され、それ以外の場合は rA を使用した 32 ビット比較が実行されます。

ニーモニック beaneid は D ビットをセットします。D ビットは、分岐遅延スロットの有無を指定します。D ビットがセットされている場合は遅延スロットがあり、分岐先の命令を実行する前に、分岐に後続する命令(分岐遅延スロットにある命令)を実行できます。D ビットがセットされていない場合は、遅延スロットではなく、分岐後に実行される命令は分岐先の命令になります。

擬似コード

```
If (preceded by imml) and rAL ≠ 0 then
    PC ← PC + sext(IMM)
else if rA ≠ 0 then
    PC ← PC + sext(IMM)
else
    PC ← PC + 4
if D = 1 then
    allow following instruction to complete execution
```

変更されるレジスタ

- PC

レイテンシ

- 1 サイクル(分岐が取られなかった場合、または正しく分岐予測された場合)
- 2 サイクル(分岐が取られ、D ビットがセットされている場合)
- 3 サイクル(分岐が取られ、D ビットがセットされていない場合、または C_AREA_OPTIMIZED=0 で分岐予測が間違っている場合)
- 7 ~ 9 サイクル(C_AREA_OPTIMIZED=2 で分岐予測が間違っている場合)

注記

デフォルトでは、タイプ B 分岐 long 命令は 16 ビットの IMM フィールド値を取り込み、それを 64 ビットに符号拡張して、即値オペランドとして使用します。この動作は、タイプ B 命令に imm または imml 命令を先行させると変更できます。64 ビットの即値の使用については、「imm」および「imml」を参照してください。

アセンブラー疑似命令 bealnei および beaneid は、long 比較を示すために使用されます。

遅延スロットには、imm、imml、分岐、またはブレーク命令を含めることはできません。割り込みおよび外部ハードウェアブレークは、遅延スロット分岐が完了するまで保留されます。

brea 拡張アドレスに無条件分岐

brea	rB _L	拡張アドレスに分岐
bread	rB _L	遅延後に拡張アドレスに分岐
breald	rD _L , rB _L	遅延後に拡張アドレスに分岐およびリンク

1 0 0 1 1 0	rD _L	D 0 L 0 1	rB _L	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0	6	11	16	21 31

説明

PC + rB_L で指定されたアドレスにある命令に分岐します。

ニーモニック breald は L ビットをセットします。L ビットがセットされると、リンクが実行されます。PC の現在の値は rD_L に格納されます。

ニーモニック bread および breald は、D ビットをセットします。D ビットは、分岐遅延スロットの有無を指定します。D ビットがセットされている場合は遅延スロットがあり、分岐先の命令を実行する前に、分岐に後続する命令(分岐遅延スロットにある命令)を実行できます。

D ビットがセットされていない場合は、遅延スロットではなく、分岐後に実行される命令は分岐先の命令になります。

擬似コード

```
if L = 1 then
    (rDL) ← PC
    PC ← PC + (rBL)
if D = 1 then
    allow following instruction to complete execution
```

変更されるレジスタ

- rD_L
- PC

レイテンシ

- 2 サイクル(D ビットがセットされている場合)
- 3 サイクル(D ビットがセットされていない場合)

注記:

命令 breal は使用できません。

絶対拡張アドレス分岐は、命令 bra、brad、および breald で実行できます。

遅延スロットには、imm、imml、分岐、またはブレーク命令を含めることはできません。割り込みおよび外部ハードウェアブレークは、遅延スロット分岐が完了するまで保留されます。

breati 拡張アドレス即値に無条件分岐

breai	IMM	拡張アドレス即値に分岐
breadi	IMM	遅延後に拡張アドレス即値に分岐
brealid	rD _L , IMM	遅延後に拡張アドレス即値に分岐およびリンク

1 0 1 1 1 0	rD _L	D 0 L 0 1	IMM
0	6	11	16

31

説明

先行する imm または imml 命令からの即値で拡張した PC + IMM で指定されたアドレスにある命令に分岐します。ニーモニック brealid は L ビットをセットします。L ビットがセットされると、リンクが実行されます。PC の現在の値は rD_L に格納されます。

ニーモニック breadi および brealid は、D ビットをセットします。D ビットは、分岐遅延スロットの有無を指定します。D ビットがセットされている場合は遅延スロットがあり、分岐先の命令を実行する前に、分岐に後続する命令(分岐遅延スロットにある命令)を実行できます。D ビットがセットされていない場合は、遅延スロットはなく、分岐後に実行される命令は分岐先の命令になります。

擬似コード

```
if L = 1 then
    (rDL) ← PC
    PC ← PC + sext(IMM)
if D = 1 then
    allow following instruction to complete execution
```

変更されるレジスタ

- rD_L
- PC

レイテンシ

- 1 サイクル(正しく分岐予測がされる場合)
- 2 サイクル(D ビットがセットされている場合)
- 3 サイクル(D ビットがセットされていない場合、または C_AREA_OPTIMIZED=0 で分岐予測が間違っている場合)
- 7 ~ 9 サイクル(C_AREA_OPTIMIZED=2 で分岐予測が間違っている場合)

注記

命令 breali は使用できません。

絶対拡張アドレス分岐は、命令 brai、braid、および bralid で実行できます。

デフォルトでは、タイプ B 分岐 long 命令は 16 ビットの IMM フィールド値を取り込み、それを 64 ビットに符号拡張して、即値オペランドとして使用します。この動作は、タイプ B 命令に imm または imml 命令を先行させると変更できます。64 ビットの即値の使用については、「[imm](#)」および「[imml](#)」を参照してください。

遅延スロットには、imm、imml、分岐、またはブレーク命令を含めることはできません。割り込みおよび外部ハードウェアブレークは、遅延スロット分岐が完了するまで保留されます。

bsl long のバレルシフト

bslrl	rD _L , rA _L , rB	long の右論理バレルシフト
bslra	rD _L , rA _L , rB	long の右算術バレルシフト
bslll	rD _L , rA _L , rB	long の左論理バレルシフト

0	1	0	0	0	1	rD _L	rA _L	rB	S	T	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
						0			6			11			16			21											31		

説明

レジスタ rA_L の内容をレジスタ rB で指定した値分シフトし、その結果をレジスタ rD_L に出力します。

ニーモニック bsll は、S ビット(サイドビット)をセットします。S ビットがセットされている場合、左バレルシフトが実行されます。ニーモニック bslrl および bslra は、S ビットをクリアし、右シフトを実行します。

ニーモニック bslra は、T ビット(タイプビット)をセットします。T ビットがセットされている場合、算術バレルシフトが実行されます。ニーモニック bslrl および bslll は、T ビットをクリアし、論理シフトを実行します。

擬似コード

```

if S = 1 then
    (rDL) ← (rAL) << (rB) [26:31]
else
    if T = 1 then
        if ((rB) [26:31]) ≠ 0 then
            (rDL) [0:(rB) [26:31]-1] ← (rAL) [0]
            (rDL) [(rB) [26:31]:31] ← (rAL) >> (rB) [26:31]
        else
            (rDL) ← (rAL)
    else
        (rDL) ← (rAL) >> (rB) [26:31]

```

変更されるレジスタ

- rD_L

レイテンシ

- 1 サイクル(C_AREA_OPTIMIZED=0 または 2)
- 2 サイクル(C_AREA_OPTIMIZED=1)

注記:

これらの命令はオプションです。これらの命令を使用するには、MicroBlaze をバレルシフト命令が使用されるように設定する必要があります(C_USE_BARREL=1)。

bsli long の即値バレルシフト

bslrl	rD _L , rA _L , IMM	long の右論理即値バレルシフト
bslrai	rD _L , rA _L , IMM	long の右算術即値バレルシフト
bslli	rD _L , rA _L , IMM	long の左論理即値バレルシフト
bslef	rD _L , rA _L , IMM _W , IMM _S	long の即値バレルシフト抽出フィールド
bslifi	rD _L , rA _L , Width ¹ , IMM _S	long の即値バレルシフト挿入フィールド

1. Width = IMM_W - IMM_S + 1

0	1	1	0	0	1	rD _L	rA _L	0	0	1	0	0	S	T	0	0	0	IMM
0						6		11		16			21		26		31	

0	1	1	0	0	1	rD _L	rA _L	I	E	1	0	IMM _W	IMM _S				
0						6		11		16		20	25	26		31	

説明

最初の3つの命令は、レジスタ rA_L の内容を IMM で指定した値分シフトし、その結果をレジスタ rD_L に配置します。バレルシフト抽出フィールドは、レジスタ rA_L からビットフィールドを抽出し、その結果をレジスタ rD_L に配置します。ビットフィールドの幅は IMM_W で指定し、シフト量は IMM_S で指定します。ビットフィールドの幅は 1 ~ 63 の範囲で、IMM_W + IMM_S ≤ 64 という条件を満たす必要があります。

バレルシフト挿入フィールドは、レジスタ rA_L からのビットフィールドをレジスタ rD_L に挿入し、レジスタ rD_L の既存の値を変更します。ビットフィールドの幅は IMM_W - IMM_S + 1 で指定し、シフト量は IMM_S で指定します。IMM_W ≥ IMM_S という条件を満たす必要があります。

ニーモニック bslli は、S ビット(サイドビット)をセットします。S ビットがセットされている場合、左バレルシフトが実行されます。ニーモニック bslrl および bslrai は、S ビットをクリアし、右シフトを実行します。

ニーモニック bslrai は、T ビット(タイプビット)をセットします。T ビットがセットされている場合、算術バレルシフトが実行されます。ニーモニック bslrl および bslli は、T ビットをクリアし、論理シフトを実行します。

ニーモニック bslef は、E ビット(抽出ビット)をセットします。この場合、S および T ビットは使用されません。

ニーモニック bslifi は、I ビット(挿入ビット)をセットします。この場合、S および T ビットは使用されません。

擬似コード

```

if E = 1 then
    ( $rD_L$ ) [0:63- $IMM_W$ ] ← 0
    ( $rD_L$ ) [64- $IMM_W$ :63] ← ( $rA_L$ ) >>  $IMM_S$ 
else if I = 1 then
    mask ← (0xffffffffffffffff << ( $IMM_W$  + 1)) ⊕ (0xffffffffffffffff <<  $IMM_S$ )
    ( $rD_L$ ) ← (( $rA_L$ ) <<  $IMM_S$ ) ∧ mask) ∨ (( $rD_L$ ) ∧ mask)
else if S = 1 then
    ( $rD_L$ ) ← ( $rA_L$ ) << IMM
else if T = 1 then
    if IMM ≠ 0 then
        ( $rD_L$ ) [0: $IMM$ -1] ← ( $rA_L$ ) [0]
        ( $rD_L$ ) [ $IMM$ :31] ← ( $rA_L$ ) >> IMM
    else
        ( $rD_L$ ) ← ( $rA_L$ )
else
    ( $rD_L$ ) ← ( $rA_L$ ) >> IMM

```

変更されるレジスタ

- rD_L

レイテンシ

- 1 サイクル ($C_AREA_OPTIMIZED=0$ または 2)
- 2 サイクル ($C_AREA_OPTIMIZED=1$)

注記

これらはタイプ B 命令ではありません。先行する imm または imml 命令からの影響はありません。

これらの命令はオプションです。これらの命令を使用するには、MicroBlaze をバレルシフト命令が使用されるように設定する必要があります ($C_USE_BARREL=1$)。

「bslifi rD, rA, width, shift」というアセンブラー コードは、 IMM_W フィールドではなく、実際のビットフィールドの幅を表し、 $IMM_W = \text{シフト} + \text{幅} - 1$ で計算されます。

cmpl**整数比較 (long)**

cmpl	rD_L, rA_L, rB_L	compare rB_L with rA_L (signed)
cmplu	rD_L, rA_L, rB_L	compare rB_L with rA_L (unsigned)

0	0	0	1	0	1	rD_L	rA_L	rB_L	0	0	1	0	0	0	0	0	U	1
0			6			11		16			21							31

説明

レジスタ rA_L の内容をレジスタ rB_L の内容から減算し、その結果をレジスタ rD_L に配置します。

rD_L の MSB ビットに rA_L と rB_L の真の関係が示されます。U ビットがセットされている場合、 rA_L および rB_L は符号なしの値とみなされます。U ビットがクリアされている場合、 rA_L および rB_L は符号付きの値とみなされます。

擬似コード

```
(rD_L) ← (rB_L) + (rA_L) + 1
(rD_L) (MSB) ← (rA_L) > (rB_L)
```

変更されるレジスタ

- rD_L

レイテンシ

- 1 サイクル

dadd 倍精度浮動小数点算術加算

dadd rD_L, rA_L, rB_L 加算

0	1	0	1	1	0	rD _L	rA _L	rB _L	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0					6		11		16		21													31

説明

レジスタ rA_L と rB_L の倍精度浮動小数点和を算出し、レジスタ rD_L に配置します。

擬似コード

```

if isDnz(rAL) or isDnz(rBL) then
    (rDL) ← 0xFFFF8000000000000000
    FSR[DO] ← 1
    ESR[EC] ← 00110
else if isSigNaN(rAL) or isSigNaN(rBL) or
    (isPosInfinite(rAL) and isNegInfinite(rBL)) or
    (isNegInfinite(rAL) and isPosInfinite(rBL)) then
    (rDL) ← 0xFFFF8000000000000000
    FSR[IO] ← 1
    ESR[EC] ← 00110
else if isQuietNaN(rAL) or isQuietNaN(rBL) then
    (rDL) ← 0xFFFF8000000000000000
else if isDnz((rAL)+(rBL)) then
    (rDL) ← signZero((rAL)+(rBL))
    FSR[UF] ← 1
    ESR[EC] ← 00110
else if isNaN((rAL)+(rBL)) then
    (rDL) ← signInfinite((rAL)+(rBL))
    FSR[OF] ← 1
    ESR[EC] ← 00110
else
    (rDL) ← (rAL) + (rBL)

```

変更されるレジスタ

- rD_L (FP例外が生成されない場合。例外が生成された場合は変更なし)
- ESR[EC] (FP例外が生成される場合)
- FSR[IO,UF,OF,DO]

レイテンシ

- 4 サイクル (C_AREA_OPTIMIZED=0)
- 6 サイクル (C_AREA_OPTIMIZED=1)
- 1 サイクル (C_AREA_OPTIMIZED=2)

注記:

この命令は、MicroBlaze のパラメーター C_USE_FPU が 0 より大きい場合のみ使用可能です。

drsub 倍精度浮動小数点算術逆減算

drsub rD_L, rA_L, rB_L 逆減算

0	1	0	1	1	0	rD _L	rA _L	rB _L	1	0	0	1	0	0	0	0	0	0	0	0
0			6			11		16		21										31

説明

レジスタ rA_L の倍精度浮動小数点値をレジスタ rB_L の浮動小数点値から減算し、その結果をレジスタ rD_L に配置します。

擬似コード

```

if isDnz(rAL) or isDnz(rBL) then
    (rDL) ← 0xFFFF8000000000000000
    FSR[DO] ← 1
    ESR[EC] ← 00110
else if (isSigNaN(rAL) or isSigNaN(rBL) or
        (isPosInfinite(rAL) and isPosInfinite(rBL)) or
        (isNegInfinite(rAL) and isNegInfinite(rBL))) then
    (rDL) ← 0xFFFF8000000000000000
    FSR[IO] ← 1
    ESR[EC] ← 00110
else if isQuietNaN(rAL) or isQuietNaN(rBL) then
    (rDL) ← 0xFFF8000000000000
else if isDnz((rBL) - (rAL)) then
    (rDL) ← signZero((rBL) - (rAL))
    FSR[UF] ← 1
    ESR[EC] ← 00110
else if isNaN((rBL) - (rAL)) then
    (rDL) ← signInfinite((rBL) - (rAL))
    FSR[OF] ← 1
    ESR[EC] ← 00110
else
    (rDL) ← (rBL) - (rAL)

```

変更されるレジスタ

- rD_L (FP例外が生成されない場合。例外が生成された場合は変更なし)
- ESR[EC] (FP例外が生成される場合)
- FSR[IO,UF,OF,DO]

レイテンシ

- 4 サイクル (C_AREA_OPTIMIZED=0)
- 6 サイクル (C_AREA_OPTIMIZED=1)
- 1 サイクル (C_AREA_OPTIMIZED=2)

注記:

この命令は、MicroBlazeのパラメーター C_USE_FPU が 0 より大きい場合のみ使用可能です。

dmul 倍精度浮動小数点算術乗算

dmul rD_L, rA_L, rB_L 乗算

0	1	0	1	1	0	rD _L	rA _L	rB _L	1	0	1	0	0	0	0	0	0	0	0
0			6			11		16		21									31

説明

レジスタ rA_L の倍精度浮動小数点値とレジスタ rB_L の浮動小数点値を乗算し、その結果をレジスタ rD_L に配置します。

擬似コード

```

if isDnz(rAL) or isDnz(rBL) then
    (rDL) ← 0xFFFF8000000000000000
    FSR[DO] ← 1
    ESR[EC] ← 00110
else
    if isNaN(rAL) or isNaN(rBL) or (isZero(rAL) and isInfinite(rBL)) or
        (isZero(rBL) and isInfinite(rAL)) then
        (rDL) ← 0xFFFF8000000000000000
        FSR[IO] ← 1
        ESR[EC] ← 00110
    else if isQuietNaN(rAL) or isQuietNaN(rBL) then
        (rDL) ← 0xFFFF8000000000000000
    else if isDnz((rBL) * (rAL)) then
        (rDL) ← signZero((rAL) * (rBL))
        FSR[UF] ← 1
        ESR[EC] ← 00110
    else if isNaN((rBL) * (rAL)) then
        (rDL) ← signInfinite((rBL) * (rAL))
        FSR[OF] ← 1
        ESR[EC] ← 00110
    else
        (rDL) ← (rBL) * (rAL)

```

変更されるレジスタ

- rD_L (FP例外が生成されない場合。例外が生成された場合は変更なし)
- ESR[EC] (FP例外が生成される場合)
- FSR[IO,UF,OF,DO]

レイテンシ

- 4 サイクル (C_AREA_OPTIMIZED=0)
- 6 サイクル (C_AREA_OPTIMIZED=1)
- 1 サイクル (C_AREA_OPTIMIZED=2)

注記:

この命令は、MicroBlaze のパラメーター C_USE_FPU が 0 より大きい場合のみ使用可能です。

ddiv 倍精度浮動小数点算術除算

ddiv rD_L, rA_L, rB_L 除算

0	1	0	1	1	0	rD _L	rA _L	rB _L	1	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0					6		11		16				21																	31

説明

レジスタ rA_L の倍精度浮動小数点値をレジスタ rB_L の浮動小数点値で割り、その結果をレジスタ rD_L に配置します。

擬似コード

```
if isDnz(rAL) or isDnz(rBL) then
    (rDL) ← 0xFFFF8000000000000000
    FSR[DO] ← 1
    ESR[EC] ← 00110
else
    if isSignNaN(rAL) or isSignNaN(rBL) or (isZero(rAL) and isZero(rBL)) or
        (isInfinite(rAL) and isInfinite(rBL)) then
        (rDL) ← 0xFFFF8000000000000000
        FSR[IO] ← 1
        ESR[EC] ← 00110
    else if isQuietNaN(rAL) or isQuietNaN(rBL) then
        (rDL) ← 0xFFFF8000000000000000
    else if isZero(rAL) and not isInfinite(rBL) then
        (rDL) ← signInfinite((rBL) / (rAL))
        FSR[DZ] ← 1
        ESR[EC] ← 00110
    else if isDnz((rBL) / (rAL)) then
        (rDL) ← signZero((rBL) / (rAL))
        FSR[UF] ← 1
        ESR[EC] ← 00110
    else if isNaN((rBL) / (rAL)) then
        (rDL) ← signInfinite((rBL) / (rAL))
        FSR[OF] ← 1
        ESR[EC] ← 00110
    else
        (rDL) ← (rBL) / (rAL)
```

変更されるレジスタ

- rD_L (FP例外が生成されない場合。例外が生成された場合は変更なし)
- ESR[EC] (FP例外が生成される場合)
- FSR[IO,UF,OF,DO,DZ]

レイテンシ

- 28サイクル (C_AREA_OPTIMIZED=0)
- 30サイクル (C_AREA_OPTIMIZED=1)
- 24サイクル (C_AREA_OPTIMIZED=2)

注記:

この命令は、MicroBlaze のパラメーター C_USE_FPU が 0 より大きい場合のみ使用可能です。

dcmp 倍精度浮動小数点値比較

dcmp.un	rD, rA _L , rB _L	倍精度浮動小数点値の比較(順不同)
dcmp.lt	rD, rA _L , rB _L	倍精度浮動小数点値の比較(小なり)
dcmp.eq	rD, rA _L , rB _L	倍精度浮動小数点値の比較(等価)
dcmp.le	rD, rA _L , rB _L	倍精度浮動小数点値の比較(以下)
dcmp.gt	rD, rA _L , rB _L	倍精度浮動小数点値の比較(大なり)
dcmp.ne	rD, rA _L , rB _L	倍精度浮動小数点値の比較(不等価)
dcmp.ge	rD, rA _L , rB _L	倍精度浮動小数点値の比較(以上)

0	1	0	1	1	0	rD	rA _L	rB _L	1	1	0	0	OpSel	0	0	0	0
0			6			11		16		21		25		28		31	

説明

レジスタ rA_L の倍精度浮動小数点値とレジスタ rB_L の浮動小数点値を比較し、その比較結果をレジスタ rD に配置します。命令コードの OpSel フィールドは、実行される比較のタイプを指定します。

擬似コード

```
if isDnz(rAL) or isDnz(rBL) then
    (rD) ← 0
    FSR[DO] ← 1
    ESR[EC] ← 00110
else
    {read out behavior from 表 5-3}
```

変更されるレジスタ

- rD_L (FP 例外が生成されない場合。例外が生成された場合は変更なし)
- ESR[EC] (FP 例外が生成される場合)
- FSR[IO,DO]

レイテンシ

- 1 サイクル (C_AREA_OPTIMIZED=0 または 2)
- 3 サイクル (C_AREA_OPTIMIZED=1)

注記:

これらの命令は、MicroBlaze のパラメーター C_USE_FPU が 0 より大きい場合のみ使用可能です。

表 5-3 に、浮動小数点値の比較演算をリストします。

表 5-4: 倍精度浮動小数点比較演算

比較タイプ		オペランド関係				
説明	OpSel	(rB _L) > (rA _L)	(rB _L) < (rA _L)	(rB _L) = (rA _L)	isSigNaN(rA _L) または isSigNaN(rB _L)	isQuietNaN(rA _L) または isQuietNaN(rB _L)
順不同	000	(rD) ← 0	(rD) ← 0	(rD) ← 0	(rD) ← 1 FSR[IO] ← 1 ESR[EC] ← 00110	(rD) ← 1
小なり	001	(rD) ← 0	(rD) ← 1	(rD) ← 0	(rD) ← 0 FSR[IO] ← 1 ESR[EC] ← 00110	(rD) ← 0 FSR[IO] ← 1 ESR[EC] ← 00110
等価	010	(rD) ← 0	(rD) ← 0	(rD) ← 1	(rD) ← 0 FSR[IO] ← 1 ESR[EC] ← 00110	(rD) ← 0
以下	011	(rD) ← 0	(rD) ← 1	(rD) ← 1	(rD) ← 0 FSR[IO] ← 1 ESR[EC] ← 00110	(rD) ← 0 FSR[IO] ← 1 ESR[EC] ← 00110
大なり	100	(rD) ← 1	(rD) ← 0	(rD) ← 0	(rD) ← 0 FSR[IO] ← 1 ESR[EC] ← 00110	(rD) ← 0 FSR[IO] ← 1 ESR[EC] ← 00110
不等価	101	(rD) ← 1	(rD) ← 1	(rD) ← 0	(rD) ← 1 FSR[IO] ← 1 ESR[EC] ← 00110	(rD) ← 1
以上	110	(rD) ← 1	(rD) ← 0	(rD) ← 1	(rD) ← 0 FSR[IO] ← 1 ESR[EC] ← 00110	(rD) ← 0 FSR[IO] ← 1 ESR[EC] ← 00110

dbl long 値を倍精度浮動小数点値に変換

dbl rD_L, rA_L

0	1	0	1	1	0	rD _L	rA _L	0	0	0	0	0	1	1	0	1	0	0	0	0	0	0	0
0			6			11		16					21										31

説明

レジスタ rA_L の符号付き long 値を、倍精度浮動小数点値に変換し、その結果をレジスタ rD_L に配置します。これは 64 ビットの丸め符号付き変換で、64 ビットの浮動小数点値を生成します。

擬似コード

```
(rDL) ← double ((rAL))
```

変更されるレジスタ

- rD_L

レイテンシ

- 4 サイクル (C_AREA_OPTIMIZED=0)
- 6 サイクル (C_AREA_OPTIMIZED=1)
- 1 サイクル (C_AREA_OPTIMIZED=2)

注記:

この命令は、MicroBlaze のパラメーター C_USE_FPU が 2 (拡張) の場合のみ使用可能です。

dlong 倍精度浮動小数点値を long 値に変換

dlong rD_L, rA_L

0	1	0	1	1	0	rD _L	rA _L	0	0	0	0	0	1	1	1	0	0	0	0	0	0	0	0
0			6			11		16		21													31

説明

レジスタ rA_L の倍精度浮動小数点値を符号付き long 値に変換し、その結果をレジスタ rD_L に配置します。これは 64 ビットの符号付き変換で、64 ビットの long 値を生成します。

擬似コード

```

if isDnz(rAL) then
    (rDL) ← 0xFFFF8000000000000000
    FSR[DO] ← 1
    ESR[EC] ← 00110
else if isNaN(rAL) then
    (rDL) ← 0xFFFF8000000000000000
    FSR[IO] ← 1
    ESR[EC] ← 00110
else if isInf(rAL) or (rAL) < -263 or (rAL) > 263 - 1 then
    (rDL) ← 0xFFFF8000000000000000
    FSR[IO] ← 1
    ESR[EC] ← 00110
else
    (rDL) ← long ((rAL))

```

変更されるレジスタ

- rD_L (FP 例外が生成されない場合。例外が生成された場合は変更なし)
- ESR[EC] (FP 例外が生成される場合)
- FSR[IO,DO]

レイテンシ

- 5 サイクル (C_AREA_OPTIMIZED=0)
- 7 サイクル (C_AREA_OPTIMIZED=1)
- 2 サイクル (C_AREA_OPTIMIZED=2)

注記:

この命令は、MicroBlaze のパラメーター C_USE_FPU が 2 (拡張) の場合のみ使用可能です。

dsqrt 倍精度浮動小数点値の算術平方根

dsqrt rD_L, rA_L 平方根 (Sqrt)

0	1	0	1	1	0	rD _L	rA _L	0	0	0	0	0	1	1	1	1	0	0	0	0	0	0	0
0			6			11		16		21													31

説明

rA_L の値に対して倍精度浮動小数点平方根を計算し、その結果を rD_L に配置します。

擬似コード

```

if isDnz(rAL) then
    (rDL) ← 0xFFFF8000000000000000
    FSR[DO] ← 1
    ESR[EC] ← 00110
else if isSigNaN(rAL) then
    (rDL) ← 0xFFFF8000000000000000
    FSR[IO] ← 1
    ESR[EC] ← 00110
else if isQuietNaN(rAL) then
    (rDL) ← 0xFFFF8000000000000000
else if (rAL) < 0 then
    (rDL) ← 0xFFFF8000000000000000
    FSR[IO] ← 1
    ESR[EC] ← 00110
else if (rAL) = -0 then
    (rDL) ← -0
else
    (rDL) ← sqrt ((rAL))

```

変更されるレジスタ

- rD_L (FP例外が生成されない場合。例外が生成された場合は変更なし)
- ESR[EC] (FP例外が生成される場合)
- FSR[IO,DO]

レイテンシ

- 27サイクル (C_AREA_OPTIMIZED=0)
- 29サイクル (C_AREA_OPTIMIZED=1)
- 23サイクル (C_AREA_OPTIMIZED=2)

注記:

この命令は、MicroBlaze のパラメーター C_USE_FPU が 2 (拡張) の場合のみ使用可能です。

imml 即値 (long)

imml IMM24

1	0	1	1	0	0	1	0	IMM24
0			6		8			31

説明

imml 命令は、IMM24 値を一時レジスタにロードします。また、その値を後続の命令で使用できるようにロックして 40 ビットの即値を形成し、後続の命令が 64 ビットのタイプ B 命令として扱われるようになります。

imml 命令は、タイプ B 64 ビット命令と共に使用されます。MicroBlaze では、64 ビットの即値 long 命令に 40 ビットまでの即値を使用できます。imml 命令は、24 ビットの IMM24 値を次の命令用に一時的にロックします。imml 命令の直後のタイプ B 命令は、imml 命令の 24 ビットの IMM24 値(上位 24 ビット)とその命令自体の 16 ビットの即値フィールド(下位 16 ビット)から 40 ビットの即値を形成します。imml 命令の後にタイプ B 命令が続かない場合は、ロックされた値はロック解除され、無用になります。

レイテンシ

- 1 サイクル

注記

imml 命令とそれに続くタイプ B 命令はアトミックなので、この 2 つの間に割り込みは使用できません。

ザイリンクが提供するアセンブラーでは、imml 命令が必要かどうかが自動的に検出されます。40 ビットの IMM 値がタイプ B 命令で指定されると、命令をアセンブルするためアセンブラーにより IMM 値が 16 ビット値に変換され、実行ファイルのその値の前に imml 命令が挿入されます。即値が 40 ビットを超える場合は、エラーが生成されます。

II long をロード

ll rD_L, rA_L, rB_L
 llr rD_L, rA_L, rB_L

1	1	0	0	1	0	rD _L	rA _L	rB _L	0	R	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
0						6		11			16					21									31		

説明

レジスタ rA_L と rB_L の内容を加算した、long で境界整列された結果のメモリ ロケーションから long (64 ビット) をロードします。データはレジスタ rD_L に配置されます。

R ビットがセットされている場合、ロードしたワードのバイトを反転し、E ビットで定義されているエンディアンネスとは逆のエンディアンネスでデータをロードします(仮想保護モードがイネーブルの場合)。

仮想保護モードがイネーブルの場合、データ TLB ミス例外が発生し、アドレスに対応する有効な変換エントリは TLB では検出されません。

ノーアクセスゾーン保護によりアクセスが禁止されている場合は、データストレージ例外が発生します。これはユーザー モードおよび仮想保護モードがイネーブルになっているアクセスのみに適用されます。

アドレスの最下位 3 ビットがゼロでない場合は、非境界整列データ アクセス例外が発生します。

擬似コード

```

Addr ← (rAL) + (rBL)
if TLB_Miss(Addr) and MSR[VM] = 1 then
  ESR[EC] ← 10010; ESR[S] ← 0
  MSR[UMS] ← MSR[UM]; MSR[VMS] ← MSR[VM]; MSR[UM] ← 0; MSR[VM] ← 0
else if Access_Protected(Addr) and MSR[UM] = 1 and MSR[VM] = 1 then
  ESR[EC] ← 10000; ESR[S] ← 0; ESR[DIZ] ← 1
  MSR[UMS] ← MSR[UM]; MSR[VMS] ← MSR[VM]; MSR[UM] ← 0; MSR[VM] ← 0
else if Addr[C_ADDR_SIZE-3:C_ADDR_SIZE-1] ≠ 0 then
  ESR[EC] ← 00001; ESR[W] ← 1; ESR[S] ← 0; ESR[Rx] ← rD
else
  (rDL) ← Mem(Addr)

```

変更されるレジスタ

- rD_L (例外が生成されない場合。例外が生成された場合は変更なし)
- MSR[UM]、MSR[VM]、MSR[UMS]、MSR[VMS] (TLB ミス例外またはデータストレージ例外が生成される場合)
- ESR[EC]、ESR[S] (例外が生成される場合)
- ESR[DIZ] (データストレージ例外が生成される場合)
- ESR[W]、ESR[Rx] (非境界整列データ アクセス例外が生成される場合)

レイテンシ

- 2 サイクル (C_AREA_OPTIMIZED=0 または 2)
- 3 サイクル (C_AREA_OPTIMIZED=1)

注記

この long 反転命令は、MicroBlaze が再順序付け命令を使用するように設定されている場合 (C_USE_REORDER_INSTR = 1) にのみ有効です。

lli long 即値をロード

lli rD_L, rA_L, IMM

1	1	1	0	1	1	rD _L	rA _L	IMM
0			6		11		16	31

説明

レジスタ rA_L の内容に IMM の符号拡張した値を加算した結果の、long で境界整列されたメモリ ロケーションから、long (64 ビット) をロードします。データはレジスタ rD_L に配置されます。

仮想保護モードがイネーブルの場合、データ TLB ミス例外が発生し、アドレスに対応する有効な変換エントリは TLB では検出されません。

ノーアクセスゾーン保護によりアクセスが禁止されている場合は、データストレージ例外が発生します。これはユーザー モードおよび仮想保護モードがイネーブルになっているアクセスのみに適用されます。

アドレスの最下位 3 ビットがゼロでない場合は、非境界整列データアクセス例外が発生します。

擬似コード

```

Addr ← (rAL) + sext(IMM)
if TLB_Miss(Addr) and MSR[VM] = 1 then
    ESR[EC] ← 10010; ESR[S] ← 0
    MSR[UMS] ← MSR[UM]; MSR[VMS] ← MSR[VM]; MSR[UM] ← 0; MSR[VM] ← 0
else if Access_Protected(Addr) and MSR[UM] = 1 and MSR[VM] = 1 then
    ESR[EC] ← 10000; ESR[S] ← 0; ESR[DIZ] ← 1
    MSR[UMS] ← MSR[UM]; MSR[VMS] ← MSR[VM]; MSR[UM] ← 0; MSR[VM] ← 0
else if Addr[C_ADDR_SIZE-3:C_ADDR_SIZE-1] ≠ 0 then
    ESR[EC] ← 00001; ESR[W] ← 1; ESR[S] ← 0; ESR[Rx] ← rD
else
    (rDL) ← Mem(Addr)

```

変更されるレジスタ

- rD_L (例外が生成されない場合。例外が生成された場合は変更なし)
- MSR[UM]、MSR[VM]、MSR[Ums]、MSR[VMS] (TLB ミス例外またはデータストレージ例外が生成される場合)
- ESR[EC]、ESR[S] (例外が生成される場合)
- ESR[DIZ] (データストレージ例外が生成される場合)
- ESR[W]、ESR[Rx] (非境界整列データアクセス例外が生成される場合)

レイテンシ

- 2 サイクル (C_AREA_OPTIMIZED=0 または 2)
- 3 サイクル (C_AREA_OPTIMIZED=1)

注記:

デフォルトでは、タイプ B ロード命令は 16 ビットの IMM フィールド値を取り込み、それを 64 ビットに符号拡張して即値オペランドとして使用します。この動作は、タイプ B 命令に imm または imml 命令を先行させると変更できます。64 ビットの即値の使用については、「[imm](#)」および「[imml](#)」を参照してください。

orl**long の論理 OR**orl rD_L, rA_L, rB_L

1	0	0	0	0	0	rD _L	rA _L	rB _L	0	0	1	0	0	0	0	0	0	0	0	0	0
0						6	11	16	21												31

説明

レジスタ rA_L の内容とレジスタ rB_L の内容を OR 演算し、その結果をレジスタ rD_L に配置します。

擬似コード

$$(rD_L) \leftarrow (rA_L) \vee (rB_L)$$

変更されるレジスタ

- rD_L

レイテンシ

- 1 サイクル

orli**long と即値の論理 OR**

orli rD_L, rA_L, IMM

1	0	1	0	0	0	rD _L	rA _L	IMM
0			6		11		16	31

説明

レジスタ rA_L の内容と、IMM フィールドの値を先行する imml 命令からの 24 ビット即値で符号拡張した値を OR 演算し、その結果をレジスタ rD_L に配置します。

擬似コード

$(rD_L) \leftarrow (rA_L) \vee \text{sext(IMM)}$

変更されるレジスタ

- rD_L

レイテンシ

- 1 サイクル

注記:

タイプ B の論理 long 命令には、imml 命令を先行させる必要があります。long 即値の使用については、「[imml](#)」を参照してください。

pcmplbf long のバイトバイト単位のパターン比較

pcmplbf rD, rA_L, rB_L バイト単位の比較を実行し、最初に一致した位置を返す

1 0 0 0 0 0	rD	rA _L	rB _L	1 0 1 0
0	6	11	16	21

説明

レジスタ rA_L の内容をレジスタ rB_L の内容とバイト単位で比較します。

- MSB (位置 1) から LSB (位置 8) までを比較し、最初に一致したバイトペアの位置を rD にロードします。
- 一致するバイトペアがない場合は、rD を 0 にセットします。

擬似コード

```

if rBL[0:7] = rAL[0:7] then
    (rD) ← 1
else if rBL[8:15] = rAL[8:15] then
    (rD) ← 2
else if rBL[16:23] = rAL[16:23] then
    (rD) ← 3
else if rBL[24:31] = rAL[24:31] then
    (rD) ← 4
else if rBL[32:39] = rAL[32:39] then
    (rD) ← 5
else if rBL[40:47] = rAL[40:47] then
    (rD) ← 6
else if rBL[48:55] = rAL[48:55] then
    (rD) ← 7
else if rBL[56:63] = rAL[56:63] then
    (rD) ← 8
else
    (rD) ← 0

```

変更されるレジスタ

- rD

レイテンシ

- 1 サイクル

注記:

この命令は C_USE_PCMP_INSTR を 1 に設定している場合のみ使用可能です。

pcmp eq long のパターン等価比較

pcmp eq rD, rA_L, rB_L 等しいかどうかを比較して正のブール値を返す

1	0	0	0	1	0	rD	rA _L	rB _L	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	
0						6			11			16			21									31

説明

レジスタ rA_L の内容をレジスタ rB_L の内容と比較します。

- 一致した場合は rD に 1 がロードされ、一致しない場合は 0 がロードされます。

擬似コード

```
if (rBL) = (rAL) then
    (rD) ← 1
else
    (rD) ← 0
```

変更されるレジスタ

- rD

レイテンシ

- 1 サイクル

注記:

この命令は C_USE_PCMP_INSTR を 1 に設定している場合のみ使用可能です。

pcmplne long のパターン不等価比較

`pcmplne rD, rAL, rBL` 等しいかどうかを比較して負のブール値を返す

1 0 0 0 1 1	rD	rA _L	rB _L	1 0 1 0 0 0 0 0 0 0 0 0 0 0	
0	6	11	16	21	31

説明

レジスタ rA_L の内容をレジスタ rB_L の内容と比較します。

- 一致した場合は rD に 0 がロードされ、一致しない場合は 1 がロードされます。

擬似コード

```
if ( $rB_L$ ) = ( $rA_L$ ) then
    ( $rD$ )  $\leftarrow$  0
else
    ( $rD$ )  $\leftarrow$  1
```

変更されるレジスタ

- rD

レイテンシ

- 1 サイクル

注記:

この命令は `C_USE_PCMP_INSTR` を 1 に設定している場合のみ使用可能です。

rsUBL long の算術逆減算

rsUBL	rD _L , rA _L , rB _L	long の減算
rsUBLc	rD _L , rA _L , rB _L	long のキャリー付き減算
rsUBLk	rD _L , rA _L , rB _L	long の減算およびキャリー保持
rsUBLkc	rD _L , rA _L , rB _L	long のキャリー付き減算およびキャリー保持

0 0 0 K C 1	rD _L	rA _L	rB _L	0 0 1 0 0 0 0 0 0 0 0
0	6	11	16	21

説明

レジスタ rA_L の内容をレジスタ rB_L の内容から減算し、その結果をレジスタ rD_L に配置します。命令のビット 3 (図の K) は、ニーモニック rsUBLk に対して 1 にセットされます。命令のビット 4 (図の C) は、ニーモニック rsUBLc に対して 1 にセットされます。ニーモニック rsUBLkc の場合は、両方のビットが 1 にセットされます。

rsUBL 命令でビット 3 がセットされている場合 (rsUBLk, rsUBLkc)、命令の実行結果にかかわらず、キャリーフラグは前の値を保持します。ビット 3 がクリアされている場合 (rsUBL, rsUBLc)、キャリーフラグは命令の実行結果によって変更されます。

命令のビット 4 が 1 にセットされている場合 (rsUBLc, rsUBLkc)、キャリーフラグの内容 (MSR[C]) が命令実行に影響します。ビット 4 がクリアされている場合 (rsUBL, rsUBLk)、キャリーフラグの内容は命令実行には影響しません (標準減算の場合)。

擬似コード

```

if C = 0 then
    (rDL) ← (rBL) + (rAL) + 1
else
    (rDL) ← (rBL) + (rAL) + MSR[C]
if K = 0 then
    MSR[C] ← CarryOut64

```

変更されるレジスタ

- rD_L
- MSR[C]

レイテンシ

- 1 サイクル

注記:

減算では、キャリー=(ボロー)です。減算でキャリー(繰り上がり)が設定される場合は、ボロー(繰り下がり)がないということで、キャリーがクリアになると、ボローがあるということになります。

rsubli long と即値の算術逆減算

rsubli	rD _L , rA _L , IMM	long と即値の減算
rsublic	rD _L , rA _L , IMM	long と即値のキャリー付き減算
rsublik	rD _L , rA _L , IMM	long と即値の減算およびキャリー保持
rsublikc	rD _L , rA _L , IMM	long と即値のキャリー付き減算およびキャリー保持

0	0	1	K	C	1	rD _L	rA _L	IMM
0			6		11		16	31

説明

レジスタ rA_L の内容を、IMM の値を先行する imml 命令からの 24 ビット即値で符号拡張した値から減算し、その結果をレジスタ rD_L に配置します。命令のビット 3(図の K)は、ニーモニック rsublik に対して 1 にセットされます。命令のビット 4(図の C)は、ニーモニック rsublic に対して 1 にセットされます。ニーモニック rsublikc の場合は、両方のビットが 1 にセットされます。

rsubli 命令でビット 3 がセットされている場合(rsublik、rsublikc)、命令の実行結果にかかわらず、キャリー フラグは前の値を保持します。ビット 3 がクリアされている場合(rsubli、rsublic)、キャリー フラグは命令の実行結果によって変更されます。

命令のビット 4 が 1 にセットされている場合(rsublic、rsublikc)、キャリー フラグの内容(MSR[C])が命令実行に影響します。ビット 4 がクリアされている場合(rsubli、rsublik)、キャリー フラグの内容は命令実行には影響しません(標準減算の場合)。

擬似コード

```

if C = 0 then
    (rDL) ← sext(IMM24 & IMM) + (rAL) + 1
else
    (rDL) ← sext(IMM24 & IMM) + (rAL) + MSR[C]
if K = 0 then
    MSR[C] ← CarryOut64

```

変更されるレジスタ

- rD_L
- MSR[C]

レイテンシ

- 1 サイクル

注記:

減算では、キャリー=(ボロー)です。減算でキャリー(繰り上がり)が設定される場合は、ボロー(繰り下がり)がないということで、キャリーがクリアになると、ボローがあるということになります。

タイプ B の算術 long 命令には、imml 命令を先行させる必要があります。long 即値の使用については、「imml」を参照してください。

sextl16 ハーフワードを long に符号拡張sextl16 rD_L, rA_L

1	0	0	1	0	0	rD _L	rA _L	0	0	0	0	0	0	0	1	0	1	1	0	0	0	0	1
0			6			11		16															31

説明

この命令は、ハーフワード(16ビット)をlong(64ビット)に符号拡張します。rA_Lのビット48が、rD_Lのビット0～47にコピーされます。rA_Lのビット48～63は、rD_Lのビット48～63にコピーされます。

擬似コード

```
(rDL) [0:47] ← (rAL) [48]
(rDL) [48:63] ← (rAL) [48:63]
```

変更されるレジスタ

- rD_L

レイテンシ

- 1サイクル

sextl32 ワードを long に符号拡張

sextl32 rD_L, rA_L

1	0	0	1	0	0	rD _L	rA _L	0	0	0	0	0	0	0	1	0	1	1	0	0	0	1	0
0			6			11		16															31

説明

この命令は、ワード(32ビット)をlong(64ビット)に符号拡張します。rA_Lのビット32が、rD_Lのビット0～31にコピーされます。rA_Lのビット32～63は、rD_Lのビット32～63にコピーされます。

擬似コード

```
(rDL) [0:31] ← (rAL) [32]
(rDL) [32:63] ← (rAL) [32:63]
```

変更されるレジスタ

- rD_L

レイテンシ

- 1サイクル

sextl8 バイトを long に符号拡張

sextl8 rD_L, rA_L

1	0	0	1	0	0	rD _L	rA _L	0	0	0	0	0	0	0	1	0	1	1	0	0	0	0	0
0			6			11		16															31

説明

この命令は、バイト(8ビット)をlong(64ビット)に符号拡張します。rA_Lのビット56が、rD_Lのビット0～55にコピーされます。rA_Lのビット56～63は、rD_Lのビット56～63にコピーされます。

擬似コード

```
(rDL) [0:55] ← (rAL) [56]
(rDL) [56:63] ← (rAL) [56:63]
```

変更されるレジスタ

- rD_L

レイテンシ

- 1サイクル

srla long の算術右シフトsrla rD_L, rA_L

1	0	0	1	0	0	rD _L	rA _L	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1
0			6			11		16															31

説明

rA_L の内容を 1 ビット右に算術シフトし、その結果を rD_L に配置します。rA_L の最上位ビット(符号ビット)は、rD_L の最上位ビットに配置されます。シフト チェーンから出力される最下位ビットは、キャリー フラグに配置されます。

擬似コード

```
(rDL) [0] ← (rAL) [0]
(rDL) [1:63] ← (rAL) [0:62]
MSR[C] ← (rAL) [63]
```

変更されるレジスタ

- rD_L
- MSR[C]

レイテンシ

- 1 サイクル

srlc long のキャリー付き右シフトsrlc rD_L, rA_L

1	0	0	1	0	0	rD _L	rA _L	0	0	0	0	0	0	0	1	0	0	1	0	0	0	0	0	1
0			6			11		16																31

説明

rA_L の内容を 1 ビット右にシフトし、その結果を rD_L に配置します。シフト チェーンにキャリー フラグをシフト インし、rD_L の最上位ビットに配置します。シフト チェーンから出力される最下位ビットは、キャリー フラグに配置されます。

擬似コード

```
(rDL) [0] ← MSR[C]
(rDL) [1:63] ← (rAL) [0:62]
MSR[C] ← (rAL) [63]
```

変更されるレジスタ

- rD_L
- MSR[C]

レイテンシ

- 1 サイクル

srl1**long の論理右シフト**srl1 rD_L, rA_L

1	0	0	1	0	0	rD _L	rA _L	0	0	0	0	0	0	0	1	0	1	0	0	0	0	0	1
0			6			11		16															31

説明

rA_L の内容を 1 ビット右に論理シフトし、その結果を rD_L に配置します。シフト チェーンに 0 をシフト インし、rD_L の最上位ビットに配置します。シフト チェーンから出力される最下位ビットは、キャリー フラグに配置されます。

擬似コード

```
(rDL) [0] ← 0
(rDL) [1:63] ← (rAL) [0:62]
MSR[C] ← (rAL) [63]
```

変更されるレジスタ

- rD_L
- MSR[C]

レイテンシ

- 1 サイクル

s|**long を格納**

sl rD_L, rA_L, rB_L
 slr rD_L, rA_L, rB_L

1	1	0	1	1	0	rD _L	rA _L	rB _L	0	R	1	0	0	0	0	0	0	0	0
0			6			11		16			21								31

説明

レジスタ rD_L の内容を、レジスタ rA_L と rB_L の内容を加算した結果の、long に境界整列されたメモリ ロケーションに格納します。

R ビットがセットされている場合、格納する long のバイトを反転し、E ビットで定義されているエンディアンネスとは逆のエンディアンネスでデータを格納します(仮想保護モードがイネーブルの場合)。

仮想保護モードがイネーブルの場合、データ TLB ミス例外が発生し、アドレスに対応する有効な変換エントリは TLB では検出されません。

仮想保護モードがイネーブルになっているとデータストレージ例外が発生し、ノーアクセスまたは読み出し専用ゾーン保護によってアクセスが禁止されます。ノーアクセス許可はユーザー モードの場合のみ発生します。

アドレスの最下位 3 ビットがゼロでない場合は、非境界整列データ アクセス例外が発生します。

擬似コード

```

Addr ← (rAL) + (rBL)
if TLB_Miss(Addr) and MSR[VM] = 1 then
  ESR[EC] ← 10010; ESR[S] ← 1
  MSR[UMS] ← MSR[UM]; MSR[VMS] ← MSR[VM]; MSR[UM] ← 0; MSR[VM] ← 0
else if Access_Protected(Addr) and MSR[VM] = 1 then
  ESR[EC] ← 10000; ESR[S] ← 1; ESR[DIZ] ← No-access-allowed
  MSR[UMS] ← MSR[UM]; MSR[VMS] ← MSR[VM]; MSR[UM] ← 0; MSR[VM] ← 0
else if Addr[C_ADDR_SIZE-3:C_ADDR_SIZE-1] ≠ 0 then
  ESR[EC] ← 00001; ESR[W] ← 1; ESR[S] ← 1; ESR[Rx] ← rD
else
  Mem(Addr) ← (rDL)

```

変更されるレジスタ

- MSR[UM]、MSR[VM]、MSR[UMS]、MSR[VMS] (TLB ミス例外またはデータストレージ例外が生成される場合)
- ESR[EC]、ESR[S] (例外が生成される場合)
- ESR[DIZ] (データストレージ例外が生成される場合)
- ESR[W]、ESR[Rx] (非境界整列データアクセス例外が生成される場合)

レイテンシ

- 2 サイクル (C_AREA_OPTIMIZED=0 または 2)
- 3 サイクル (C_AREA_OPTIMIZED=1)

注記

この long 反転命令は、MicroBlaze が再順序付け命令を使用するように設定されている場合 (C_USE_REORDER_INSTR = 1) にのみ有効です。

sli**long を格納 (即値)**sli rD_L, rA_L, IMM

1	1	1	1	1	1	rD _L	rA _L	IMM
0			6		11		16	31

説明

レジスタ rD_L の内容を、レジスタ rA_L の内容と符号拡張された IMM 値を加算した結果の、long で境界整列されたメモリロケーションに格納します。

仮想保護モードがイネーブルの場合、データ TLB ミス例外が発生し、アドレスに対応する有効な変換エントリは TLB では検出されません。

仮想保護モードがイネーブルになっているとデータストレージ例外が発生し、ノーアクセスまたは読み出し専用ゾーン保護によってアクセスが禁止されます。ノーアクセス許可はユーザー モードの場合にのみ発生します。

アドレスの最下位 3 ビットがゼロでない場合は、非境界整列データアクセス例外が発生します。

擬似コード

```

Addr ← (rAL) + sext(IMM)
if TLB_Miss(Addr) and MSR[VM] = 1 then
    ESR[EC] ← 10010; ESR[S] ← 1
    MSR[UMS] ← MSR[UM]; MSR[VMS] ← MSR[VM]; MSR[UM] ← 0; MSR[VM] ← 0
else if Access_Protected(Addr) and MSR[VM] = 1 then
    ESR[EC] ← 10000; ESR[S] ← 1; ESR[DIZ] ← No-access-allowed
    MSR[UMS] ← MSR[UM]; MSR[VMS] ← MSR[VM]; MSR[UM] ← 0; MSR[VM] ← 0
else if Addr[C_ADDR_SIZE-3:C_ADDR_SIZE-1] ≠ 0 then
    ESR[EC] ← 00001; ESR[W] ← 1; ESR[S] ← 1; ESR[Rx] ← rD
else
    Mem(Addr) ← (rDL)

```

変更されるレジスタ

- MSR[UM]、MSR[VM]、MSR[UMS]、MSR[VMS] (TLB ミス例外またはデータストレージ例外が生成される場合)
- ESR[EC]、ESR[S] (例外が生成される場合)
- ESR[DIZ] (データストレージ例外が生成される場合)
- ESR[W]、ESR[Rx] (非境界整列データアクセス例外が生成される場合)

レイテンシ

- 2 サイクル (C_AREA_OPTIMIZED=0 または 2)
- 3 サイクル (C_AREA_OPTIMIZED=1)

注記:

デフォルトでは、タイプ B ストア命令は 16 ビットの IMM フィールド値を取り込み、それを 64 ビットに符号拡張して即値オペランドとして使用します。この動作は、タイプ B 命令に imm または imml 命令を先行させると変更できます。

xorl long の排他的論理 ORxorl rD_L, rA_L, rB_L

1	0	0	0	1	0	rD _L	rA _L	rB _L	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0					6		11		16		21														31

説明

レジスタ rA_L の内容とレジスタ rB_L の内容を XOR 演算し、その結果をレジスタ rD_L に配置します。

擬似コード
$$(rD_L) \leftarrow (rA_L) \oplus (rB_L)$$
変更されるレジスタ

- rD_L

レイテンシ

- 1 サイクル

xorli long の即値との排他的論理 OR

xorli rD_L, rA_L, IMM

1	0	1	0	1	0	rD _L	rA _L	IMM
0			6		11		16	31

説明

レジスタ rA_L の内容と、IMM フィールドの値を先行する imm1 命令からの 24 ビット即値で符号拡張した値を XOR 演算し、その結果をレジスタ rD_L に配置します。

擬似コード

```
(rDL) ← (rAL) ⊕ sext(IMM24 & IMM)
```

変更されるレジスタ

- rD_L

レイテンシ

- 1 サイクル

注記

タイプ B の論理 long 命令には、imm1 命令を先行させる必要があります。long 即値の使用については、「[imm1](#)」を参照してください。

パフォーマンスおよびリソース使用率

パフォーマンス

このコアのパフォーマンス特性化には、マージンシステム手法が使用されています。マージンシステム手法についての詳細は、「[IP 特性化および \$f_{MAX}\$ マージンシステム手法](#)」を参照してください。

パフォーマンスおよびリソース使用率に関する追加の詳細は、「[パフォーマンスおよびリソース使用率](#)」を参照してください。

最大周波数

表 A-1 に、MicroBlaze™ コアの最大周波数を示します。この表は、各ファミリの最高速グレードを使用した結果を示しています。

表 A-1: 最大周波数

ファミリ	F_{max} (MHz)
Virtex®-7	382
Kintex®-7	398
Artix-7	267
Zynq®-7000	265
Spartan®-7	234
Virtex UltraScale™	460
Kintex UltraScale	463
Virtex UltraScale+™	682
Kintex UltraScale+	650
Zynq UltraScale+	661

リソース使用量

さまざまなパラメーター設定での MicroBlaze コアリソース使用量が次のデバイスで計測されています。

- Virtex-7 ([表 A-2](#))
- Kintex-7 ([表 A-3](#))
- Artix-7 ([表 A-4](#))
- Zynq-7000 ([表 A-5](#))
- Spartan-7 ([表 A-6](#))
- Virtex UltraScale ([表 A-7](#))
- Kintex UltraScale ([表 A-8](#))
- Virtex UltraScale+ ([表 A-9](#))
- Kintex UltraScale+ ([表 A-10](#))
- Zynq UltraScale+ ([表 A-11](#))

[表 A-12](#) に、計測された設定のパラメーター値をリストします。これらの設定は、MicroBlaze Configuration ウィザードに含まれているプリセットおよびテンプレートに対応しています。

32 ビットプロセッサのインプリメンテーションデータではパラメーター C_DATA_SIZE = 32 および C_ADDR_SIZE = 32 を使用し、64 ビットプロセッサのインプリメンテーションデータではパラメーター C_DATA_SIZE = 64 および C_ADDR_SIZE = 48 を使用しています。

表 A-2: デバイス使用率 - Virtex-7 FPGA (XC7VX485T ffg1761-3)

設定	デバイス リソース					
	32-bit			64-bit		
	LUT	FF	F _{max} (MHz)	LUT	FF	F _{max} (MHz)
マイクロコントローラー プリセット	1173	811	308	2139	1262	286
リアルタイム プリセット	2484	2121	245	3793	3150	241
アプリケーション プリセット	4340	3807	212	6492	4919	165
最小エリア	629	230	382	1133	400	329
最大パフォーマンス	4096	3210	218	6813	4778	172
最大周波数	915	553	382	1815	858	329
MMU を使用した Linux	3512	3126	213	5084	4496	198
MMU を使用した 低性能 Linux	2986	2511	233	4519	3726	207
標準	2007	1680	253	3389	2498	251
周波数最適化	6011	5791	252	9398	8735	168

表 A-3: デバイス使用率 - Kintex-7 FPGA (XC7K325T ffg900-3)

設定	デバイス リソース					
	32-bit			64-bit		
	LUT	FF	F_{max} (MHz)	LUT	FF	F_{max} (MHz)
マイクロコントローラー プリセット	1176	811	318	2129	1226	287
リアルタイム プリセット	2477	2121	246	3792	3151	220
アプリケーション プリセット	4368	3779	214	6479	4899	174
最小エリア	637	234	398	1146	398	330
最大パフォーマンス	4129	3207	222	6816	4778	171
最大周波数	908	553	398	1817	862	330
MMU を使用した Linux	3507	3149	206	5088	4493	205
MMU を使用した 低性能 Linux	2986	2537	213	4521	3708	202
標準	2017	1679	257	3404	2496	252
周波数最適化	6004	5874	263	9425	8765	172

表 A-4: デバイス使用率 - Artix-7 FPGA (XC7A200T fbg676-3)

設定	デバイス リソース					
	32-bit			64-bit		
	LUT	FF	F_{max} (MHz)	LUT	FF	F_{max} (MHz)
マイクロコントローラー プリセット	1174	811	218	2145	1226	187
リアルタイム プリセット	2467	2121	177	3797	3153	178
アプリケーション プリセット	4326	3747	149	6461	4891	136
最小エリア	625	227	267	1141	397	221
最大パフォーマンス	4106	3208	153	6802	4799	142
最大周波数	911	553	267	1815	858	221
MMU を使用した Linux	3515	3122	150	5081	4492	139
MMU を使用した 低性能 Linux	2987	2506	151	4490	3711	136
標準	2014	1682	187	3398	2500	190
周波数最適化	5956	5787	166	9366	8725	137

表 A-5: デバイス使用率 - Zynq-7000 FPGA (XC7Z020 clg484-3)

設定	デバイス リソース					
	32-bit			64-bit		
	LUT	FF	F_{max} (MHz)	LUT	FF	F_{max} (MHz)
マイクロコントローラー プリセット	1174	811	221	2148	1226	191
リアルタイム プリセット	2465	2120	176	3785	3156	178
アプリケーション プリセット	4345	3744	148	6496	4979	141
最小エリア	626	226	265	1138	400	222
最大パフォーマンス	4105	3197	152	6791	4760	138
最大周波数	908	553	265	1813	858	222
MMU を使用した Linux	3507	3125	147	5086	4489	135
MMU を使用した 低性能 Linux	2988	2506	159	4489	3711	138
標準	2021	1680	191	3416	2501	192
周波数最適化	5953	5785	176	9381	8724	134

表 A-6: デバイス使用率 - Spartan-7 FPGA (XC7S25 csga225-2)

設定	デバイス リソース					
	32-bit			64-bit		
	LUT	FF	F_{max} (MHz)	LUT	FF	F_{max} (MHz)
マイクロコントローラー プリセット	1175	811	188	2134	1226	161
リアルタイム プリセット	2461	2125	161	3810	3151	148
アプリケーション プリセット	4342	3812	130	6465	4872	120
最小エリア	625	225	234	1145	406	199
最大パフォーマンス	4085	3197	134	6779	4757	118
最大周波数	909	553	234	1816	858	199
MMU を使用した Linux	3505	3128	133	5077	4490	118
MMU を使用した 低性能 Linux	2980	2509	144	4466	3709	122
標準	2020	1680	168	3406	2492	160
周波数最適化	5955	5783	154	9363	8724	119

表 A-7: デバイス使用率 - Virtex UltraScale FPGA (XCVU095 ffvd1924-3)

設定	デバイス リソース					
	32-bit			64-bit		
	LUT	FF	F_{max} (MHz)	LUT	FF	F_{max} (MHz)
マイクロコントローラー プリセット	1105	821	413	2090	1226	345
リアルタイム プリセット	2520	2121	295	3822	3158	293
アプリケーション プリセット	4355	3801	262	6617	4826	238
最小エリア	567	231	460	991	415	374
最大パフォーマンス	4102	3208	286	6936	4776	244
最大周波数	913	553	460	1817	860	374
MMU を使用した Linux	3523	3221	258	5149	4511	239
MMU を使用した 低性能 Linux	3002	2518	271	4559	3728	234
標準	2035	1680	316	3482	2497	307
周波数最適化	6150	5806	301	9579	8814	240

表 A-8: デバイス使用率 - Kintex UltraScale FPGA (XCKU040 ffva1156-3)

設定	デバイス リソース					
	32-bit			64-bit		
	LUT	FF	F_{max} (MHz)	LUT	FF	F_{max} (MHz)
マイクロコントローラー プリセット	1106	811	417	2046	1261	352
リアルタイム プリセット	2507	2119	300	3820	3155	302
アプリケーション プリセット	4336	3760	255	6621	4961	233
最小エリア	578	240	463	988	401	391
最大パフォーマンス	4117	3209	285	6943	4784	249
最大周波数	913	556	463	1832	869	391
MMU を使用した Linux	3502	3129	247	5142	4492	239
MMU を使用した 低性能 Linux	2997	2507	267	4560	3745	233
標準	2033	1683	319	3471	2505	313
周波数最適化	6172	5837	307	9574	8777	243

表 A-9: デバイス使用率 - Virtex UltraScale FPGA (XCVU3P ffvc1517-3)

設定	デバイス リソース					
	32-bit			64-bit		
	LUT	FF	F_{max} (MHz)	LUT	FF	F_{max} (MHz)
マイクロコントローラー プリセット	1107	823	573	2063	1226	462
リアルタイム プリセット	2543	2122	399	3911	3156	389
アプリケーション プリセット	4403	3745	360	6679	4872	333
最小エリア	563	225	682	991	397	602
最大パフォーマンス	4207	3208	371	7044	4772	330
最大周波数	910	553	682	1816	858	602
MMU を使用した Linux	3553	3129	350	5213	4486	333
MMU を使用した 低性能 Linux	3020	2508	374	4595	3705	333
標準	2064	1679	433	3492	2496	427
周波数最適化	6227	5789	416	9649	8773	344

表 A-10: デバイス使用率 - Kintex UltraScale FPGA (XCKU15P ffva1156-3)

設定	デバイス リソース					
	32-bit			64-bit		
	LUT	FF	F_{max} (MHz)	LUT	FF	F_{max} (MHz)
マイクロコントローラー プリセット	1105	811	573	2049	1226	522
リアルタイム プリセット	2449	2122	416	3914	3151	407
アプリケーション プリセット	4404	3744	349	6693	4873	341
最小エリア	566	225	650	996	397	602
最大パフォーマンス	4209	3237	371	7045	4778	340
最大周波数	915	553	650	1814	858	602
MMU を使用した Linux	3554	3190	351	5216	4491	323
MMU を使用した 低性能 Linux	3023	2507	365	4598	3712	344
標準	2067	1681	441	3489	2493	421
周波数最適化	6223	5787	433	9652	8766	340

表 A-11: デバイス使用率 - Zynq UltraScale+ FPGA (XCZU9EG ffvb1156-3)

設定	デバイス リソース					
	32-bit			64-bit		
	LUT	FF	F_{max} (MHz)	LUT	FF	F_{max} (MHz)
マイクロコントローラー プリセット	1112	822	561	2046	1226	476
リアルタイム プリセット	2540	2120	409	3912	3150	388
アプリケーション プリセット	4415	3743	346	6684	4888	336
最小エリア	566	229	661	995	407	573
最大パフォーマンス	4212	3207	372	7027	4779	336
最大周波数	908	553	661	1818	858	573
MMU を使用した Linux	3552	3121	338	5227	4553	335
MMU を使用した 低性能 Linux	3018	2501	369	4597	3703	319
標準	2068	1681	430	3490	2493	428
周波数最適化	6248	5819	413	9647	8781	348

表 A-12: パラメーターコンフィギュレーション

パラメーター	コンフィギュレーション パラメーター値									
	マイクロコントローラー ^{ブリセット}	リアルタイム ^{ブリセット}	アプリケーション ^{ブリセット}	最小エリア	最大パフォーマンス	最大周波数	Linux (MMU を使用)	低性能 Linux (MMU を使用)	標準	周波数 (最適化済み)
C_ALLOW_DCACHE_WR	1	1	1	1	1	1	1	1	1	1
C_ALLOW_ICACHE_WR	1	1	1	1	1	1	1	1	1	1
C_AREA_OPTIMIZED	1	0	0	1	0	0	0	0	0	2
C_CACHE_BYTE_SIZE	4096	8192	32768	4096	32768	4096	16384	8192	8192	16384
C_DCACHE_BYTE_SIZE	4096	8192	32768	4096	32768	4096	16384	8192	8192	16384
C_DCACHE_LINE_LEN	4	4	4	4	8	4	4	4	4	4
C_DCACHE_USE_WRITEBACK	0	1	1	0	1	0	0	0	0	1
C_DEBUG_ENABLED	1	1	1	0	1	0	1	1	1	1
C_DIV_ZERO_EXCEPTION	0	1	1	0	0	0	1	0	0	1
C_M_AXI_D_BUS_EXCEPTION	0	1	1	0	0	0	1	1	1	1
C_FPU_EXCEPTION	0	0	1	0	0	0	0	0	0	1
C_FSL_EXCEPTION	0	0	0	0	0	0	0	0	0	0
C_FSL_LINKS	0	0	0	0	0	1	0	0	0	0
C_ICACHE_LINE_LEN	4	4	8	4	8	4	8	4	8	8
C_ILL_OPCODE_EXCEPTION	0	1	1	0	0	0	1	1	0	1
C_M_AXI_I_BUS_EXCEPTION	0	1	1	0	0	0	1	1	0	1
C_MMU_DTLB_SIZE	2	2	4	2	4	2	4	4	4	4
C_MMU_ITLB_SIZE	1	1	2	1	2	1	2	2	2	2
C_MMU_TLB_ACCESS	3	3	3	3	3	3	3	3	3	3
C_MMU_ZONES	2	2	2	2	2	2	2	2	2	2
C_NUMBER_OF_PC_BRK	1	2	2	0	1	1	1	1	2	1
C_NUMBER_OF_RD_ADDR_BRK	0	0	1	0	0	0	0	0	0	0
C_NUMBER_OF_WR_ADDR_BRK	0	0	1	0	0	0	0	0	0	0
C_OPCODE_0x0_ILLEGAL	0	1	1	0	0	0	1	1	0	1
C_PVR	0	0	2	0	0	0	2	0	0	2
C_UNALIGNED_EXCEPTIONS	0	1	1	0	0	0	1	1	0	1
C_USE_BARREL	1	1	1	0	1	0	1	1	1	1
C_USE_DCACHE	0	1	1	0	1	0	1	1	1	1
C_USE_DIV	0	1	1	0	1	0	1	0	0	1
C_USE_EXTENDED_FSL_INSTR	0	0	0	0	0	0	0	0	0	0

表 A-12: パラメーターコンフィギュレーション (続き)

パラメーター	コンフィギュレーション パラメーター値								標準	周波数 (最適化済み)
	マイクロコントローラー プリセット	リアルタイム プリセット	アプリケーション プリセット	最小 エリア	最大 パフォーマンス	最大 周波数	Linux (MMUを使用)	低性能 Linux (MMUを使用)		
C_USE_FPU	0	0	1	0	2	0	0	0	0	2
C_USE_HW_MUL	1	1	2	0	2	0	2	1	1	2
C_USE_ICACHE	0	1	1	0	1	0	1	1	1	1
C_USE_MMU	0	0	3	0	0	0	3	3	0	3
C_USE_MSR_INSTR	1	1	1	0	1	0	1	1	1	1
C_USE_PCMP_INSTR	1	1	1	0	1	0	1	1	1	1
C_USE_REORDER_INSTR	0	1	1	0	1	1	1	1	1	1
C_USE_BRANCH_TARGET_CACHE	0	0	0	0	1	0	0	0	0	1
C_BRANCH_TARGET_CACHE_SIZE	0	0	0	0	0	0	0	0	0	0
C_ICACHE_STREAMS	0	0	1	0	1	0	1	0	0	0
C_ICACHE_VICTIMS	0	0	8	0	8	0	8	0	0	0
C_DCACHE_VICTIMS	0	0	0	0	8	0	8	0	0	0
C_ICACHE_FORCE_TAG_LUTRAM	0	0	0	0	0	0	0	0	0	0
C_DCACHE_FORCE_TAG_LUTRAM	0	0	0	0	0	0	0	0	0	0
C_ICACHE_ALWAYS_USED	0	1	1	0	1	0	1	1	0	1
C_DCACHE_ALWAYS_USED	0	1	1	0	1	0	1	1	0	1
C_D_AXI	1	1	1	0	1	0	1	1	0	1
C_USE_INTERRUPT	1	1	1	0	0	0	1	1	0	1
C_USE_STACK_PROTECTION	0	1	0	0	0	0	0	0	0	0

IP 特性化および f_{MAX} マージン システム手法

概要

このセクションでは、システムデザイン内の IP 操作の最大周波数 (F_{MAX}) を決める方法について説明します。この方法を利用すると、どのザイリンクス FPGA アーキテクチャに対しても現実的なパフォーマンスをレポートすることが可能になります。デザインの最大周波数とは、タイミングに関する問題がない状態でシステム全体をインプリメントできる最大周波数を指します。

F_{MAX} マージン システム手法

IP パフォーマンスはユーザー システムという文脈の中で決定するのが重要です。MicroBlaze 特性化の場合、システムには次のアイテムが含まれます。

- 被試験 IP (MicroBlaze プロセッサ)
- ローカル メモリ (LMB)
- 1 レベルのインターフェクト (AXI4、AXI4-Lite、AXI4-Stream)
- メモリ コントローラー (EMC)
- オンチップ BRAM コントローラー
- ペリフェラル (UART、タイマー、割り込み、割り込みコントローラー、MDM)

これらのコンポーネントをもったエンベデッド IP の F_{MAX} を決定しておくと、さらに現実的なパフォーマンスをターゲットにできます。

上記のシステムには、3 種類の AXI インターフェクトがあります。ペリフェラルのコマンドおよび制御に使用される AXI4-Lite、メモリ アクセスに使用される AXI4、MicroBlaze ストリームに使用される AXI4-Stream の 3 つです。

F_{MAX} マージン システム解析の場合、システムのクロック周波数は、タイミング違反が発生してシステムに問題が起きてしまう最大周波数までインクリメントされます(ワースト ケースの負のスラック)。レポートされる周波数は、このワースト ケースの負のスラックをエラーが起きている周波数から差し引いたものです。

ツール オプションおよびその他の要因

ザイリンクス ツールには、デザイン パフォーマンス、リソース 使用率、インプリメンテーション run タイム、メモリ フットプリントをトレードオフするためのオプションや設定が多く用意されています。あるデザインで最善の結果が得られる設定が、別のデザインでうまくいくとは限りません。

F_{MAX} マージン システム解析のため、IP デザインは特定の制約なしに(クロッキング制約を除く)デフォルト 設定で特性化されています。この解析は、すべての FPGA アーキテクチャとその最大スピード グレードで実行されます。

その他のリソースおよび法的通知

ザイリンクス リソース

アンサー、資料、ダウンロード、フォーラムなどのサポート リソースは、[ザイリンクス サポート](#) サイトを参照してください。

ソリューションセンター

デバイス、ツール、IP のサポートについては、[ザイリンクス ソリューション センター](#)を参照してください。デザインアシスタント、デザインアドバイザリ、トラブルシューティングのヒントなどが含まれます。

Documentation Navigator およびデザイン ハブ

ザイリンクス Documentation Navigator (DocNav) では、ザイリンクスの資料、ビデオ、サポート リソースにアクセスでき、特定の情報を取得するためにフィルター機能や検索機能を利用できます。DocNav を開くには、次のいずれかを実行します。

- Vivado® IDE で [Help] → [Documentation and Tutorials] をクリックします。
- Windows で [スタート] → [すべてのプログラム] → [Xilinx Design Tools] → [DocNav] をクリックします。
- Linux コマンド プロンプトに「docnav」と入力します。

ザイリンクス デザイン ハブには、資料やビデオへのリンクがデザイン タスクおよびトピックごとにまとめられており、これらを参照することでキー コンセプトを学び、よくある質問 (FAQ) を参考に問題を解決できます。デザイン ハブにアクセスするには、次のいずれかを実行します。

- DocNav で [Design Hubs View] タブをクリックします。
- ザイリンクス ウェブサイトで[デザイン ハブ](#) ページを参照します。

注記: DocNav の詳細は、ザイリンクス ウェブサイトの [Documentation Navigator](#) ページを参照してください。

注意: DocNav からは、日本語版は参照できません。ウェブサイトのデザイン ハブ ページをご利用ください。



参考資料

次の資料は、Vivado® のインストールディレクトリに含まれます。

参考資料には次のものがあります。

1. 『MicroBlaze プロセッサ リファレンス ガイド』(UG011)
2. 『Soft Error Mitigation Controller LogiCORE IP 製品ガイド』(PG036: 英語版、日本語版)
3. 『LMB BRAM Interface Controller LogiCORE IP 製品ガイド』(PG112)
4. 『MicroBlaze Debug Module (MDM) 製品ガイド』(PG115)
5. 『デバイス信頼性レポート ユーザー ガイド』(PG116: 英語版、日本語版)
6. 『System Cache LogiCORE IP 製品ガイド』(PG118)
7. 『Triple Modular Redundancy (TMR) Subsystem 製品ガイド』(PG268)
8. 『階層デザイン設計手法ガイド』(UG748)
9. ザイリンクス ソフトウェア開発キット (SDK) ヘルプ (UG782)
10. 『Vivado Design Suite ユーザー ガイド: IP を使用した設計』(UG896)
11. 『Vivado Design Suite ユーザー ガイド: エンベデッド プロセッサ ハードウェア デザイン』(UG898)
12. 『Vivado Design Suite ユーザー ガイド: IP インテグレーターを使用した IP サブシステムの設計』(UG994)
13. 『エンベデッド システム ツール リファレンスマニュアル』(UG1043)
14. 『AMBA 4 AXI4-Stream Protocol Specification, Version 1.0』(Arm IHI 0051A)
15. 『AMBA AXI and ACE Protocol Specification』(Arm IHI 0022E)
16. 『UltraScale Architecture Soft Error Mitigation Controller LogiCORE IP 製品ガイド』(PG187: 英語版、日本語版)

次の追加資料には、URL を直接クリックするとアクセスできます。

17. GNU マニュアル セット: <https://www.gnu.org/manual>
18. IEEE 754-1985 規格 (https://en.wikipedia.org/wiki/IEEE_754-1985)

トレーニング リソース

ザイリンクスでは、この資料に含まれるコンセプトを説明するさまざまなトレーニング コースおよび QuickTake ビデオを提供しています。次のリンクから関連するトレーニング リソースを参照してください。

19. トレーニング クラス: Vivado Design Suite を使用した FPGA の設計 1
20. トレーニング コース: エンベデッド システム デザイン
21. トレーニング コース: エンベデッド システム ソフトウェア デザインのアドバンス機能とテクニック
22. トレーニング コース: エンベデッド システム ソフトウェア デザイン
23. Vivado Design Suite QuickTake ビデオ チュートリアル

お読みください: 重要な法的通知

本通知に基づいて貴殿または貴社(本通知の被通知者が個人の場合には「貴殿」、法人その他の団体の場合には「貴社」。以下同じ)に開示される情報(以下「本情報」といいます)は、ザイリンクスの製品を選択および使用することのためにのみ提供されます。適用される法律が許容する最大限の範囲で、(1) 本情報は「現状有姿」、およびすべて受領者の責任で(with all faults)という状態で提供され、ザイリンクスは、本通知をもって、明示、黙示、法定を問わず(商品性、非侵害、特定目的適合性の保証を含みますがこれらに限られません)、すべての保証および条件を負わない(否認する)ものとします。また、(2) ザイリンクスは、本情報(貴殿または貴社による本情報の使用を含む)に関係し、起因し、関連する、いかなる種類・性質の損失または損害についても、責任を負わない(契約上、不法行為上(過失の場合を含む)、その他のいかなる責任の法理によるかを問わない)ものとし、当該損失または損害には、直接、間接、特別、付随的、結果的な損失または損害(第三者が起こした行為の結果被った、データ、利益、業務上の信用の損失、その他あらゆる種類の損失や損害を含みます)が含まれるものとし、それは、たとえ当該損害や損失が合理的に予見可能であったり、ザイリンクスがそれらの可能性について助言を受けていた場合であつたとしても同様です。ザイリンクスは、本情報に含まれるいかなる誤りも訂正する義務を負わず、本情報または製品仕様のアップデートを貴殿または貴社に知らせる義務も負いません。事前の書面による同意のない限り、貴殿または貴社は本情報を再生産、変更、頒布、または公に展示してはなりません。一定の製品は、ザイリンクスの限定的保証の諸条件に従うこととなるので、<https://japan.xilinx.com/legal.htm#tos>で見られるザイリンクスの販売条件を参照してください。IPコアは、ザイリンクスが貴殿または貴社に付与したライセンスに含まれる保証と補助的条件に従うことになります。ザイリンクスの製品は、フェイルセーフとして、または、フェイルセーフの動作を要求するアプリケーションに使用するために、設計されたり意図されたりしていません。そのような重大なアプリケーションにザイリンクスの製品を使用する場合のリスクと責任は、貴殿または貴社が単独で負うものです。<https://japan.xilinx.com/legal.htm#tos>で見られるザイリンクスの販売条件を参照してください。

© Copyright 2013-2018 Xilinx, Inc. Xilinx、Xilinx のロゴ、Artix、ISE、Kintex、Spartan、Virtex、Vivado、Zynq、およびこの文書に含まれるその他の指定されたブランドは、米国およびその他各国のザイリンクス社の商標です。すべての他の商標は、それぞれの保有者に帰属します。

この資料に関するフィードバックおよびリンクなどの問題につきましては、jpn_trans_feedback@xilinx.comまで、または各ページの右下にある [フィードバック送信] ボタンをクリックすると表示されるフォームからお知らせください。フィードバックは日本語で入力可能です。いただきましたご意見を参考に早急に対応させていただきます。なお、このメールアドレスへのお問い合わせは受け付けておりません。あらかじめご了承ください。