

# MicroBlaze プロセッサ リファレンス ガイド

エンベデッド開発キット  
EDK 11.4

UG081 (v10.3)



Xilinx is providing this product documentation, hereinafter “Information,” to you “AS IS” with no warranty of any kind, express or implied. Xilinx makes no representation that the Information, or any particular implementation thereof, is free from any claims of infringement. You are responsible for obtaining any rights you may require for any implementation based on the Information. All specifications are subject to change without notice.

XILINX EXPRESSLY DISCLAIMS ANY WARRANTY WHATSOEVER WITH RESPECT TO THE ADEQUACY OF THE INFORMATION OR ANY IMPLEMENTATION BASED THEREON, INCLUDING BUT NOT LIMITED TO ANY WARRANTIES OR REPRESENTATIONS THAT THIS IMPLEMENTATION IS FREE FROM CLAIMS OF INFRINGEMENT AND ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

Except as stated herein, none of the Information may be copied, reproduced, distributed, republished, downloaded, displayed, posted, or transmitted in any form or by any means including, but not limited to, electronic, mechanical, photocopying, recording, or otherwise, without the prior written consent of Xilinx.

© 2009 Xilinx, Inc. XILINX, the Xilinx logo, Virtex, Spartan, ISE, and other designated brands included herein are trademarks of Xilinx in the United States and other countries. All other trademarks are the property of their respective owners.

本資料は英語版 (v.10.3) 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。

## MicroBlaze プロセッサ リファレンス ガイド UG081 (v10.3)

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
2002 年 10 月 1 日	1.0	EDK 3.1 リリース
2003 年 3 月 11 日	2.0	EDK 3.2 リリース
2003 年 9 月 24 日	3.0	EDK 6.1 リリース
2004 年 2 月 20 日	3.1	EDK 6.2 リリース
2004 年 8 月 24 日	4.0	EDK 6.3 リリース
2004 年 9 月 21 日	4.1	EDK 6.3 サービス パック 1 用に改訂
2004 年 11 月 18 日	4.2	EDK 6.3 サービス パック 2 用に改訂
2005 年 1 月 20 日	5.0	EDK 7.1i リリース
2005 年 4 月 2 日	5.1	EDK 7.1i サービス パック 1 用に改訂
2005 年 5 月 9 日	5.2	EDK 7.1i サービス パック 2 用に改訂
2005 年 10 月 5 日	5.3	EDK 8.1i 用に改訂
2006 年 2 月 21 日	5.4	EDK 8.1i サービス パック 2 用に改訂
2006 年 6 月 1 日	6.0	EDK 8.2i リリース
2006 年 7 月 24 日	6.1	EDK 8.2i サービス パック 1 用に改訂
2006 年 8 月 21 日	6.2	EDK 8.2i サービス パック 2 用に改訂

---

日付	バージョン	改訂内容
2006 年 8 月 29 日	6.3	EDK 8.2i サービスパック 2 用に改訂
2006 年 9 月 15 日	7.0	EDK 9.1i リリース
2007 年 2 月 22 日	7.1	EDK 9.1i サービスパック 1 用に改訂
2007 年 3 月 27 日	7.2	EDK 9.1i サービスパック 2 用に改訂
2007 年 6 月 25 日	8.0	EDK 9.2i リリース
2008 年 1 月 17 日	9.0	EDK 10.1 リリース
2008 年 3 月 4 日	9.1	EDK 10.1 サービス パック 1 用に改訂
2008 年 5 月 14 日	9.2	EDK 10.1 サービス パック 2 用に改訂
2008 年 7 月 14 日	9.3	EDK 10.1 サービス パック 3 用に改訂
2009 年 2 月 4 日	10.0	EDK 11.1リリース
2009 年 4 月 15 日	10.1	EDK 11.2 リリース
2009 年 5 月 28 日	10.2	EDK 11.3リリース
2009 年 10 月 26 日	10.3	EDK 11.4リリース



# 目次

---

## このマニュアルについて

マニュアルの内容 .....	7
表記規則 .....	8
書体 .....	8
オンライン マニュアル .....	9

## 第 1 章：MicroBlaze アーキテクチャ

概要 .....	12
機能 .....	12
データ型およびエンディアン .....	14
命令 .....	15
インストラクションのまとめ .....	15
セマフォの同期化 .....	25
レジスタ .....	27
汎用レジスタ .....	27
特殊用途レジスタ .....	28
パイプライン アーキテクチャ .....	52
3 段パイプライン .....	52
5 段パイプライン .....	52
分岐 .....	52
メモリ アーキテクチャ .....	54
特権命令 .....	55
仮想メモリ管理 .....	56
実モード .....	56
仮想モード .....	57
変換ルックアサイド バッファ (TLB) .....	59
アクセス保護 .....	65
UTLB 管理 .....	66
ページ アクセスおよびページ変更の記録 .....	66
リセット、割り込み、例外、ブレーク .....	67
リセット .....	68
ハードウェア例外 .....	68
ブレーク .....	71
割り込み .....	71
ユーザー ベクタ (例外) .....	72
命令キャッシュ .....	72
概要 .....	72
命令キャッシュの機能 .....	73
命令キャッシュの動作 .....	74
命令キャッシュのソフトウェア サポート .....	74
データ キャッシュ .....	75
概要 .....	75
データ キャッシュの機能 .....	75
データ キャッシュの動作 .....	76
データ キャッシュのソフトウェア サポート .....	78
浮動小数点ユニット (FPU) .....	78
概要 .....	78
フォーマット .....	79
繰り上げ/繰り下げ .....	79
操作 .....	79

例外.....	80
ソフトウェア サポート .....	80
ライブラリおよびバイナリの互換性 .....	80
オペレータ レイテンシ.....	80
C 言語のプログラミング .....	81
高速シンプレックス リンク (FSL) .....	82
ハードウェア アクセラレータへの FSL の使用.....	83
デバッグおよびトレース.....	84
デバッグの概要 .....	84
トレースの概要 .....	84

## 第 2 章：MicroBlaze の信号インターフェイス

概要 .....	85
機能.....	85
MicroBlaze I/O の概要 .....	86
プロセッサ ローカル バス (PLB) インターフェイス .....	91
オンチップ ペリフェラル バス (OPB) インターフェイス .....	91
ローカル メモリ バス (LMB) インターフェイス .....	91
LMB 信号のインターフェイス .....	91
LMB トランザクション .....	93
データの読み出しおよび書き込み操作 .....	95
高速シンプレックス リンク (FSL) インターフェイス .....	96
マスタ FSL 信号のインターフェイス .....	96
スレーブ FSL 信号のインターフェイス .....	97
FSL トランザクション .....	97
直接 FSL 接続 .....	97
ザイリンクス CacheLink (XCL) インターフェイス.....	99
CacheLink 信号のインターフェイス .....	100
CacheLink トランザクション .....	101
デバッグ インターフェイス .....	104
トレース インターフェイス .....	105
MicroBlaze コアのコンフィギュレーション .....	107

## 第 3 章：MicroBlaze アプリケーション バイナリ インターフェイス (ABI)

データ型.....	113
レジスタの使用規則 .....	114
スタックの規則 .....	115
呼び出し規則 .....	117
メモリ モデル .....	117
スモール データ領域 .....	117
データ領域 .....	117
共有の未初期化領域 .....	118
リテラルまたは定数 .....	118
割り込みおよび例外処理.....	119

## 第 4 章：MicroBlaze 命令セット アーキテクチャ

表記法 .....	121
フォーマット .....	122
命令 .....	123

# このマニュアルについて

このマニュアルでは、エンベデッド開発キットに含まれる 32 ビット ソフト プロセッサの MicroBlaze™ のハードウェアアーキテクチャについて説明します。

## マニュアルの内容

このマニュアルは、次の章から構成されています。

- 第 1 章「MicroBlaze アーキテクチャ」では、MicroBlaze の機能の概要に加え、ビッグ エンディアン形式、32 ビットの汎用レジスタ、キャッシュのソフトウェア サポートおよび高速シンプレックス リンク インターフェイスの情報を説明します。
- 第 2 章「MicroBlaze の信号インターフェイス」では、MicroBlaze への接続に使用できる信号インターフェイスのタイプを説明します。
- 第 3 章「MicroBlaze アプリケーション バイナリ インターフェイス (ABI)」では、アセンブリ言語でソフト プロセッサ用のソフトウェアを開発する際に重要なアプリケーション バイナリ インターフェイスについて説明します。
- 第 4 章「MicroBlaze 命令セット アーキテクチャ」には、MicroBlaze の命令セット アーキテクチャの表記法、フォーマット、命令が記述されています。

その他の情報については、<http://japan.xilinx.com/support/mysupport.htm> を参照してください。次の表に、リソースの一覧を示します。このリソースには、表示されている URL から直接アクセスできます。

リソース	説明/URL
チュートリアル	デザイン入力から検証やデバッグまでのザイリンクスのデザイン フローについて説明したチュートリアルです。 <a href="http://japan.xilinx.com/support/techsup/tutorials/index.htm">http://japan.xilinx.com/support/techsup/tutorials/index.htm</a>
アンサー ブラウザ	ザイリンクスのソリューション レコードのデータベースです。 <a href="http://japan.xilinx.com/xlnx/xil_ans_browser.jsp">http://japan.xilinx.com/xlnx/xil_ans_browser.jsp</a>
アプリケーション ノート	デバイス別デザイン技術およびアプローチの説明です。 <a href="http://japan.xilinx.com/xlnx/xweb/xil_publications_index.jsp?category=Application+Notes">http://japan.xilinx.com/xlnx/xweb/xil_publications_index.jsp?category=Application+Notes</a>
データ シート	リードバック、バウンダリ スキャン、コンフィギュレーション、レンジ スカウト、デバッグなど、ザイリンクス デバイスの特性に関する情報がデバイス別に記載されています。 <a href="http://japan.xilinx.com/xlnx/xweb/xil_publications_index.jsp">http://japan.xilinx.com/xlnx/xweb/xil_publications_index.jsp</a>
プロブレム ソルバー	デザインの問題をトラブルシューティングできる対話型ツールです。 <a href="http://japan.xilinx.com/support/troubleshoot/psolvers.htm">http://japan.xilinx.com/support/troubleshoot/psolvers.htm</a>

リソース	説明/URL
テクニカル ヒント	ザイリックス デザイン環境についての最新ニュース、デザインのヒント、パッチ情報です。 <a href="http://japan.xilinx.com/xlnx/xil_tt_home.jsp">http://japan.xilinx.com/xlnx/xil_tt_home.jsp</a>
GNU マニュアル	すべての GNU マニュアルを入手できます。 <a href="http://www.gnu.org/manual">http://www.gnu.org/manual</a>

## 表記規則

このマニュアルでは、次の表記規則を使用しています。各規則について、例を挙げて説明します。

### 書体

次の規則は、すべてのマニュアルで使用されています。

表記規則	使用箇所	例
Courier フォント	システムが表示するメッセージ、プロンプト、プログラム ファイルを表示します。	speed grade: - 100
<b>Courier</b> フォント (太字)	構文内で入力するコマンドを示します。	<b>ngdbuild</b> design_name
イタリック フォント	ユーザーが値を入力する必要のある構文内の変数に使用します。	<i>ngdbuild</i> design_name
二重/一重かぎカッコ『』、『』	『』はマニュアル名を、「」はセクション名を示します。	詳細は、『開発システム リファレンス ガイド』の「PAR」を参照してください。
角カッコ [ ]	オプションの入力またはパラメータを示しますが、 <b>bus[7:0]</b> のようなバス仕様では必ず使用します。また、GUI 表記にも使用します。	<b>ngdbuild</b> [option_name] design_name [File] → [Open] をクリックします。
中かっこ { }	1 つ以上の項目を選択するためのリストを示します。	<b>lowpwr</b> = {on off}
縦棒	選択するリストの項目を分離します。	<b>lowpwr</b> = {on off}
縦の省略記号 • • •	繰り返し項目が省略されていることを示します。	IOB #1: Name = QOUT' IOB #2: Name = CLKIN' • • •
横の省略記号 ...	繰り返し項目が省略されていることを示します。	<b>allow block</b> block_name loc1 loc2 ... locn;



## オンライン マニュアル

このマニュアルでは、次の規則が使用されています。

表記規則	使用箇所	例
青色の文字	マニュアル内の相互参照を示します。	詳細は、「 <a href="#">その他のリソース</a> 」を参照してください。 詳細については、第 1 章の「 <a href="#">タイトル フォーマット</a> 」を参照してください。
<a href="#">青色の下線付き文字</a>	Web サイト (URL) へのハイパーリンクです。	最新のスピード ファイルは、 <a href="http://japan.xilinx.com">http://japan.xilinx.com</a> から入手できます。



# MicroBlaze アーキテクチャ

---

この章では MicroBlaze™ の機能の概要と、ビッグ エンディアン形式、32 ビット汎用レジスタ、仮想メモリ管理、キャッシュ ソフトウェア サポート、高速シンプレックス リンク (FSL) インターフェイスを含む、MicroBlaze アーキテクチャの詳細な情報を説明します。

この章は、次のセクションから構成されています。

- 概要
- データ型およびエンディアン
- 命令
- レジスタ
- パイプライン アーキテクチャ
- メモリ アーキテクチャ
- 特権命令
- 仮想メモリ管理
- リセット、割り込み、例外、ブレーク
- 命令キャッシュ
- データ キャッシュ
- 浮動小数点ユニット (FPU)
- ソフトウェア サポート
- 高速シンプレックス リンク (FSL)
- デバッグおよびトレース

## 概要

MicroBlaze エンベデッド プロセッサ ソフト コアは、ザイリンクス FPGA でのインプリメンテーション向けに最適化された RISC (Reduced Instruction Set Computer) です。図 1-1 に、MicroBlaze コアのファンクション ブロック図を示します。

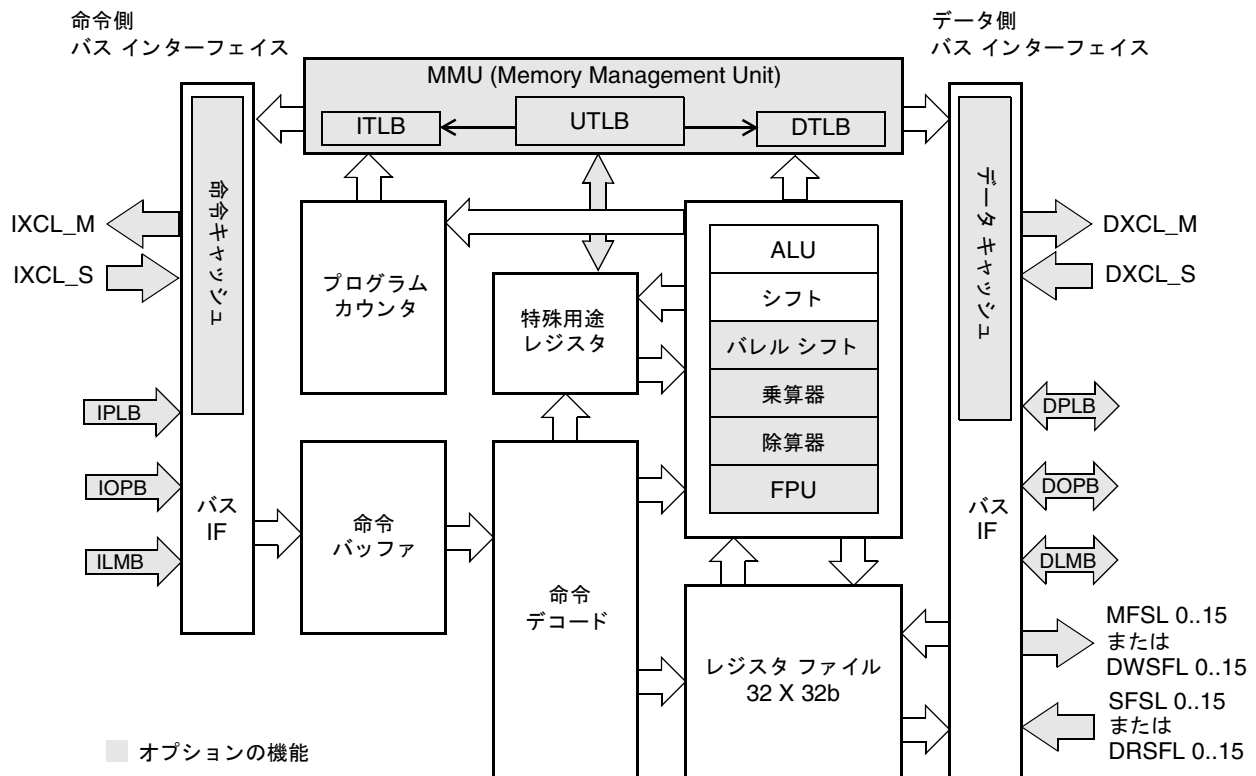


図 1-1 : MicroBlaze コアのブロック図

## 機能

MicroBlaze は、柔軟にコンフィギュレーション可能なソフト コア プロセッサで、デザインの要件に応じて機能を選択できます。

MicroBlaze プロセッサの機能は、次のとおりです。

- 32 個の 32 ビット汎用レジスタ
- 3 個のオペランドおよび 2 個のアドレス指定モードのある 32 ビット命令ワード
- 32 ビット アドレス バス
- 単一パイプライン

これらの機能に加え、選択可能なオプションの機能があります。以前のバージョンの MicroBlaze では、ここに示す機能の一部のみがサポートされます。オプションの機能がすべてサポートされているのは、最新バージョンの MicroBlaze (v7.20) のみです。

新規デザインには、最新バージョンの MicroBlaze プロセッサを使用することをお勧めします。

13 ページの表 1-1 に、Microblaze の各バージョンでコンフィギュレーション可能な機能の概要を示します。

表 1-1 : MicroBlaze のオプション機能

機能	MicroBlaze のバージョン					
	v4.00	v5.00	v6.00	v7.00	v7.10	v7.20
バージョンのステータス	廃止予定	廃止予定	廃止予定	廃止予定	推奨	推奨
プロセッサ パイプラインの段数	3	5	3/5	3/5	3/5	3/5
オンチップ ペリフェラル バス (OPB) 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
オンチップ ペリフェラル バス (OPB) 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
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 パレル シフタ	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 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
高速シンプレックス リンク (FSL) インターフェイス	0 ~ 7	0 ~ 7	0 ~ 7	0 ~ 15	0 ~ 15	0 ~ 15
マシン ステータス設定およびクリア命令	オプション	あり	オプション	オプション	オプション	オプション
IOPB インターフェイスの命令キャッシュ	オプション	なし	なし	なし	なし	なし
IOPB インターフェイスのデータ キャッシュ	オプション	なし	なし	なし	なし	なし
CacheLink (IXCL) の命令キャッシュ	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
CacheLink (DXCL) のデータ キャッシュ	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
XCL の 4 または 8 ワードのキャッシュ ライン	4	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	あり	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をディスエーブルに する <sup>1</sup>	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 デバッグで読み出し可能な ESR および EAR	あり	あり	あり	あり	あり	あり
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
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
ハードウェア乗算器の 64 ビットの結果	-	-	オプション	オプション	オプション	オプション

表 1-1 : MicroBlaze のオプション機能 (続き)

機能	MicroBlaze のバージョン					
	v4.00	v5.00	v6.00	v7.00	v7.10	v7.20
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
プロセッサ ローカル バス (PLB) 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
プロセッサ ローカル バス (PLB) 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
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
MMU (Memory Management Unit)	-	-	-	オプション	オプション	オプション
拡張高速シンプレックス リンク (FSL) 命令	-	-	-	オプション	オプション	オプション
すべての命令キャッシュ メモリ アクセス へのザイリンクス Cache Link の使用	-	-	-	-	オプション	オプション
すべてのデータ キャッシュ メモリ アクセ スへのザイリンクス Cache Link の使用	-	-	-	-	オプション	オプション
データ キャッシュのライトバック キャッ シュ ポリシーの使用	-	-	-	-	-	オプション
命令キャッシュの Cache Link (IXCL) プロトコル	-	-	-	-	-	オプション
データ キャッシュの Cache Link (DXCL) プロトコル	-	-	-	-	-	オプション

1. Virtex®-4 以降のファミリで、MUL18 および DSP48 プリミティブを節約するために使用。

## データ型およびエンディアン

MicroBlaze では、データの記述にビッグ エンディアン形式を使用します。MicroBlaze に対しハードウェアでサポートされるデータ型は、ワード、ハーフワード、およびバイトです。各データ型に対するビットおよびバイト構成を次の表に示します。

表 1-2 : ワード データ型

バイト アドレス	n	n+1	n+2	n+3
バイト表記	0	1	2	3
バイト順	MSByte			LSByte
ビット表現	0			31
ビット順	MSBit			LSBit

表 1-3: ハーフワード データ型

バイト アドレス	n	n+1
バイト表記	0	1
バイト順	MSByte	LSByte
ビット表現	0	15
ビット順	MSBit	LSBit

表 1-4: バイト データ型

バイト アドレス	n
ビット表現	0 7
ビット順	MSBit LSBit

## 命令

### インストラクションのまとめ

MicroBlaze の命令は、すべて 32 ビットで、タイプ A とタイプ B のいずれかとして定義されます。タイプ A の命令には、最大 2 個のソース レジスタ オペランドと 1 個のデスティネーション レジスタ オペランドを含めることができます。タイプ B の命令には、1 個のソース レジスタ オペランドと 16 ビットの即値オペランドが含まれます。この 16 ビットの即値オペランドは、タイプ B 命令の前に imm 命令を付けると、32 ビットまで拡張可能です。タイプ B の命令には、1 個のデスティネーション レジスタ オペランドが含まれます。命令のファンクションは、演算、論理、分岐、読み込み/格納、特殊に分類されます。表 1-6 に MicroBlaze の命令セットを示します。これらの命令の詳細は、第 4 章「MicroBlaze 命令セット アーキテクチャ」を参照してください。表 1-5 に、各命令のセマンティクスで使用される命令セットの用語を説明します。

表 1-5: 命令セットの用語

シンボル	説明
Ra	R0 ~ R31、汎用レジスタ、ソース オペランド a
Rb	R0 ~ R31、汎用レジスタ、ソース オペランド b
Rd	R0 ~ R31、汎用レジスタ、デスティネーション オペランド
SPR[x]	特殊用途レジスタ (x はレジスタ番号)
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]
PVR <sub>x</sub>	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	分岐ターゲット レジスタ = SPR[11]

表 1-5 : 命令セットの用語 (続き)

シンボル	説明
PC	実行段プログラム カウンタ = SPR[0]
$x[y]$	レジスタ $x$ のビット $y$
$x[y:z]$	レジスタ $x$ のビット $y \sim z$
$\bar{x}$	レジスタ $x$ のビット反転値
Imm	16 ビットの即値
Immx	$x$ ビットの即値
FSL $x$	4 ビットの高速シンプレックス リンク (FSL) ポート識別子 ( $x$ 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
$s(x)$	符号拡張引数 $x$ (最大 32 ビット)
*Addr	ロケーション Addr にあるメモリ内容 (データ サイズ揃え)
$:=$	代入演算子
=	等号
$\neq$	非等価
$>$	大なり
$\geq$	以上
$<$	小なり
$\leq$	以下
+	加算
*	乗算
/	除算
$\gg x$	右に $x$ ビット シフト
$\ll x$	左に $x$ ビット シフト
and	論理積 (AND)
or	論理和 (OR)
xor	排他的論理和 (XOR)
op1 if cond else op2	条件文 <i>cond</i> が真の場合は <i>op1</i> を実行、それ以外は <i>op2</i> を実行
&	連結。たとえば、「0000100 & Imm7」は固定フィールド 0000100 と 7 ビットの即値を連結したものです。
signed	符号付き整数データ型に対して実行される演算。すべての四則演算は、指定がない限り符号付きのワード オペランドで実行されます。



表 1-5 : 命令セットの用語 (続き)

シンボル	説明
unsigned	符号なし整数データ型に対して実行される演算
float	浮動小数点データ型に対して実行される演算

表 1-6 : MicroBlaze の命令セットの一覧

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	000000000000	$Rd := Rb + Ra$
RSUB Rd,Ra,Rb	000001	Rd	Ra	Rb	000000000000	$Rd := Rb + \overline{Ra} + 1$
ADDC Rd,Ra,Rb	000010	Rd	Ra	Rb	000000000000	$Rd := Rb + Ra + C$
RSUBC Rd,Ra,Rb	000011	Rd	Ra	Rb	000000000000	$Rd := Rb + \overline{Ra} + C$
ADDK Rd,Ra,Rb	000100	Rd	Ra	Rb	000000000000	$Rd := Rb + Ra$
RSUBK Rd,Ra,Rb	000101	Rd	Ra	Rb	000000000000	$Rd := Rb + \overline{Ra} + 1$
ADDKC Rd,Ra,Rb	000110	Rd	Ra	Rb	000000000000	$Rd := Rb + Ra + C$
RSUBKC Rd,Ra,Rb	000111	Rd	Ra	Rb	000000000000	$Rd := Rb + \overline{Ra} + C$
CMP Rd,Ra,Rb	000101	Rd	Ra	Rb	000000000001	$Rd := Rb + \overline{Ra} + 1$ $Rd[0] := 0$ if $(Rb \geq Ra)$ else $Rd[0] := 1$
CMPU Rd,Ra,Rb	000101	Rd	Ra	Rb	000000000011	$Rd := Rb + \overline{Ra} + 1$ (unsigned) $Rd[0] := 0$ if $(Rb \geq Ra, \text{ unsigned})$ else $Rd[0] := 1$
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	$Rd := (Ra * Rb) \gg 32$ (signed)
MULHU Rd,Ra,Rb	010000	Rd	Ra	Rb	000000000011	$Rd := (Ra * Rb) \gg 32$ (unsigned)
MULHSU Rd,Ra,Rb	010000	Rd	Ra	Rb	000000000010	$Rd := (Ra, \text{ signed} * Rb, \text{ unsigned}) \gg 32$ (signed)
BSRA Rd,Ra,Rb	010001	Rd	Ra	Rb	010000000000	$Rd := s(Ra \gg Rb)$
BSLL Rd,Ra,Rb	010001	Rd	Ra	Rb	100000000000	$Rd := (Ra \ll Rb) \& 0$
MULI Rd,Ra,Imm	011000	Rd	Ra	Imm		$Rd := Ra * s(Imm)$
BSRLI Rd,Ra,Imm	011001	Rd	Ra	000000000000 & Imm5		$Rd := 0 \& (Ra \gg Imm5)$

表 1-6 : MicroBlaze の命令セットの一覧 (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		
BSRAI Rd,Ra,Imm	011001	Rd	Ra	00000010000 & Imm5		Rd := s(Ra >> Imm5)
BSLLI Rd,Ra,Imm	011001	Rd	Ra	00000100000 & Imm5		Rd := (Ra << Imm5) & 0
IDIV Rd,Ra,Rb	010010	Rd	Ra	Rb	00000000000	Rd := Rb/Ra
IDIVU Rd,Ra,Rb	010010	Rd	Ra	Rb	00000000010	Rd := Rb/Ra, unsigned
TNEAGETD Rd,Rb	010011	Rd	00000	Rb	0N0TAE 00000	Rd := FSL Rb[28:31] (データ読み出し) 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 (データ書き込み) MSR[C] := FSL_M_Full if N = 1
TNECAGETD Rd,Rb	010011	Rd	00000	Rb	0N1TAE 00000	Rd := FSL Rb[28:31] (制御読み出し) 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 (制御書き込み) MSR[C] := FSL_M_Full if N = 1
FADD Rd,Ra,Rb	010110	Rd	Ra	Rb	00000000000	Rd := Rb+Ra, float <sup>1</sup>
FRSUB Rd,Ra,Rb	010110	Rd	Ra	Rb	00010000000	Rd := Rb-Ra, float <sup>1</sup>
FMUL Rd,Ra,Rb	010110	Rd	Ra	Rb	00100000000	Rd := Rb*Ra, float <sup>1</sup>
FDIV Rd,Ra,Rb	010110	Rd	Ra	Rb	00110000000	Rd := Rb/Ra, float <sup>1</sup>
FCMP.UN Rd,Ra,Rb	010110	Rd	Ra	Rb	01000000000	Rd := 1 if (Rb = NaN or Ra = NaN, float <sup>1</sup> ) else Rd := 0
FCMP.LT Rd,Ra,Rb	010110	Rd	Ra	Rb	01000010000	Rd := 1 if (Rb < Ra, float <sup>1</sup> ) else Rd := 0
FCMP.EQ Rd,Ra,Rb	010110	Rd	Ra	Rb	01000100000	Rd := 1 if (Rb = Ra, float <sup>1</sup> ) else Rd := 0
FCMP.LE Rd,Ra,Rb	010110	Rd	Ra	Rb	01000110000	Rd := 1 if (Rb <= Ra, float <sup>1</sup> ) else Rd := 0
FCMP.GT Rd,Ra,Rb	010110	Rd	Ra	Rb	01001000000	Rd := 1 if (Rb > Ra, float <sup>1</sup> ) else Rd := 0
FCMP.NE Rd,Ra,Rb	010110	Rd	Ra	Rb	01001010000	Rd := 1 if (Rb != Ra, float <sup>1</sup> ) else Rd := 0
FCMP.GE Rd,Ra,Rb	010110	Rd	Ra	Rb	01001100000	Rd := 1 if (Rb >= Ra, float <sup>1</sup> ) else Rd := 0
FLT Rd,Ra	010110	Rd	Ra	0	01010000000	Rd := float (Ra) <sup>1</sup>
FINT Rd,Ra	010110	Rd	Ra	0	01100000000	Rd := int (Ra) <sup>1</sup>

表 1-6 : MicroBlaze の命令セットの一覧 (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		
FSQRT Rd,Ra	010110	Rd	Ra	0	01110000000	Rd := sqrt (Ra) <sup>1</sup>
TNEAGET Rd,FSLx	011011	Rd	00000	0N07AE000000 & FSLx		Rd := FSLx (データ読み出し、N = 0 の場合 ブロッキング) MSR[FSL] := 1 if (FSLx_S_Control = 1) MSR[C] := not FSLx_S_Exists if N = 1
TNAPUT Ra,FSLx	011011	00000	Ra	1N07A0000000 & FSLx		FSLx := Ra (データ書き込み、N = 0 の場合 ブロッキング) MSR[C] := FSLx_M_Full if N = 1
TNECAGET Rd,FSLx	011011	Rd	00000	0N17AE000000 & FSLx		Rd := FSLx (制御読み出し、N = 0 の場合 ブロッキング) MSR[FSL] := 1 if (FSLx_S_Control = 0) MSR[C] := not FSLx_S_Exists if N = 1
TNCAPUT Ra,FSLx	011011	00000	Ra	1N17A0000000 & FSLx		FSLx := Ra (制御書き込み、N = 0 の場合 ブロッキング) MSR[C] := FSLx_M_Full if N = 1
OR Rd,Ra,Rb	100000	Rd	Ra	Rb	00000000000	Rd := Ra or Rb
AND Rd,Ra,Rb	100001	Rd	Ra	Rb	00000000000	Rd := Ra and Rb
XOR Rd,Ra,Rb	100010	Rd	Ra	Rb	00000000000	Rd := Ra xor Rb
ANDN Rd,Ra,Rb	100011	Rd	Ra	Rb	00000000000	Rd := Ra and $\overline{Rb}$
PCMPBF Rd,Ra,Rb	100000	Rd	Ra	Rb	10000000000	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
PCMPEQ Rd,Ra,Rb	100010	Rd	Ra	Rb	10000000000	Rd := 1 if (Rd = Ra) else Rd := 0
PCMPNE Rd,Ra,Rb	100011	Rd	Ra	Rb	10000000000	Rd := 1 if (Rd != Ra) else Rd := 0
SRA Rd,Ra	100100	Rd	Ra	00000000000000001		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])

表 1-6 : MicroBlaze の命令セットの一覧 (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	01101000	ICache_Line[Ra >> 4].Tag := 0 if (C_ICACHE_LINE_LEN = 4)  ICache_Line[Ra >> 5].Tag := 0 if (C_ICACHE_LINE_LEN = 8)
WDC Ra,Rb	100100	00000	Ra	Rb	01100100	キャッシュ ラインがクリアされ、格納デー タが廃棄されます。  DCache_Line[Ra >> 4].Tag := 0 if (C_DCACHE_LINE_LEN = 4)  DCache_Line[Ra >> 5].Tag := 0 if (C_DCACHE_LINE_LEN = 8)
WDC.FLUSH Ra,Rb	100100	00000	Ra	Rb	01100100	キャッシュ ラインがフラッシュされ、格納 データがメモリに書き込まれてからクリア されます。 D_DCACHE_USE_WRITEBACK = 1 のと きに使用。
WDC.CLEAR Ra,Rb	100100	00000	Ra	Rb	01100100	アドレスが一致するキャッシュ ラインがク リアされ、格納データが破棄されます。 D_DCACHE_USE_WRITEBACK = 1 のと きに使用。
MTS Sd,Ra	100101	00000	Ra	11 & Sd		SPR[Sd] := Ra  • SPR[0x0001] は MSR • SPR[0x0007] は FSR • SPR[0x1000] は PID • SPR[0x1001] は ZPR • SPR[0x1002] は TLBX • SPR[0x1003] は TLBLO • SPR[0x1004] は TLBHI • SPR[0x1005] は TLBSX

表 1-6 : MicroBlaze の命令セットの一覧 (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 • SPR[0x0005] は ESR • SPR[0x0007] は FSR • SPR[0x000B] は BTR • SPR[0x000D] は EDR • SPR[0x1000] は PID • SPR[0x1001] は ZPR • SPR[0x1002] は TLBX • SPR[0x1003] は TLBLO • SPR[0x1004] は TLBHI • SPR[0x2000] ~ SPR [0x200B] は PVR[0] ~ PVR [11]
MSRCLR Rd,Imm	100101	Rd	00001	00 & Imm14		Rd := MSR MSR := MSR and $\overline{\text{Imm14}}$
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
BEQ Ra,Rb	100111	00000	Ra	Rb	000000000000	PC := PC + Rb if Ra = 0
BNE Ra,Rb	100111	00001	Ra	Rb	000000000000	PC := PC + Rb if Ra != 0
BLT Ra,Rb	100111	00010	Ra	Rb	000000000000	PC := PC + Rb if Ra < 0
BLE Ra,Rb	100111	00011	Ra	Rb	000000000000	PC := PC + Rb if Ra <= 0
BGT Ra,Rb	100111	00100	Ra	Rb	000000000000	PC := PC + Rb if Ra > 0
BGE Ra,Rb	100111	00101	Ra	Rb	000000000000	PC := PC + Rb if Ra >= 0

表 1-6 : MicroBlaze の命令セットの一覧 (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		
BEQD Ra,Rb	100111	10000	Ra	Rb	000000000000	PC := PC + Rb if Ra = 0
BNED Ra,Rb	100111	10001	Ra	Rb	000000000000	PC := PC + Rb if Ra != 0
BLTD Ra,Rb	100111	10010	Ra	Rb	000000000000	PC := PC + Rb if Ra < 0
BLED Ra,Rb	100111	10011	Ra	Rb	000000000000	PC := PC + Rb if Ra <= 0
BGTD Ra,Rb	100111	10100	Ra	Rb	000000000000	PC := PC + Rb if Ra > 0
BGED Ra,Rb	100111	10101	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
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)
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)
BRAID Imm	101110	00000	11000	Imm		PC := s(Imm)
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	00000	Ra	Imm		PC := PC + s(Imm) if Ra = 0
BNEI Ra,Imm	101111	00001	Ra	Imm		PC := PC + s(Imm) if Ra != 0
BLTI Ra,Imm	101111	00010	Ra	Imm		PC := PC + s(Imm) if Ra < 0

表 1-6 : MicroBlaze の命令セットの一覧 (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		
BLEI Ra,Imm	101111	00011	Ra	Imm		PC := PC + s(Imm) if Ra <= 0
BGTI Ra,Imm	101111	00100	Ra	Imm		PC := PC + s(Imm) if Ra > 0
BGEI Ra,Imm	101111	00101	Ra	Imm		PC := PC + s(Imm) if Ra >= 0
BEQID Ra,Imm	101111	10000	Ra	Imm		PC := PC + s(Imm) if Ra = 0
BNEID Ra,Imm	101111	10001	Ra	Imm		PC := PC + s(Imm) if Ra != 0
BLTID Ra,Imm	101111	10010	Ra	Imm		PC := PC + s(Imm) if Ra < 0
BLEID Ra,Imm	101111	10011	Ra	Imm		PC := PC + s(Imm) if Ra <= 0
BGTID Ra,Imm	101111	10100	Ra	Imm		PC := PC + s(Imm) if Ra > 0
BGEID Ra,Imm	101111	10101	Ra	Imm		PC := PC + s(Imm) if Ra >= 0
LBU Rd,Ra,Rb	110000	Rd	Ra	Rb	00000000000	Addr := Ra + Rb Rd[0:23] := 0 Rd[24:31] := *Addr[0:7]
LHU Rd,Ra,Rb	110001	Rd	Ra	Rb	00000000000	Addr := Ra + Rb Rd[0:15] := 0 Rd[16:31] := *Addr[0:15]
LW Rd,Ra,Rb	110010	Rd	Ra	Rb	00000000000	Addr := Ra + Rb Rd := *Addr
LWX Rd,Ra,Rb	110010	Rd	Ra	Rb	10000000000	Addr := Ra + Rb Rd := *Addr Reservation := 1
SB Rd,Ra,Rb	110100	Rd	Ra	Rb	00000000000	Addr := Ra + Rb *Addr[0:8] := Rd[24:31]
SH Rd,Ra,Rb	110101	Rd	Ra	Rb	00000000000	Addr := Ra + Rb *Addr[0:16] := Rd[16:31]
SW Rd,Ra,Rb	110110	Rd	Ra	Rb	00000000000	Addr := Ra + Rb *Addr := Rd
SWX Rd,Ra,Rb	110110	Rd	Ra	Rb	10000000000	Addr := Ra + Rb Rd := *Rd if Reservation = 1 Reservation := 0
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]



表 1-6 : MicroBlaze の命令セットの一覧 (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		
LWI Rd,Ra,Imm	111010	Rd	Ra	Imm		Addr := Ra + s(Imm) Rd := *Addr
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

1. 浮動小数点の演算には特殊なケースが多数あるため、通常の演算のみを示します。第 4 章「MicroBlaze 命令セット アーキテクチャ」にすべての演算がリストされています。

## セマフォの同期化

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の共通セマフォ操作をインプリメントするには、**LWX** および **SWX** 命令が使用されます。また、スピンロックのインプリメントにも使用されます。

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、随時アプリケーション プログラムによって呼び出されます。一般的には、メモリからセマフォを読み込むにはプログラムにより **LWX** が使用され、予約が設定されます (まずはプロセッサが予約を保持)。セマフォの値に基づいて結果が計算され、**SWX** 命令を使用して結果値は同じメモリ ロケーションに条件付で保存されます。条件付保存は、直前の **LWX** 命令によって確立された予約を基に実行されます。保存が実行されるときに予約が存在する場合、保存が実行され **MSR [C]** が 0 にクリアされます。予約が存在しない場合、ターゲット メモリ ロケーションは変更されず、**MSR [C]** が 1 に設定されます。

保存が問題なく完了すると、セマフォ読み込みからセマフォ保存までの命令シーケンスが実行されます。ほかのデバイスは読み出しとアップデートの間のセマフォ ロケーションを変更しません。ほかのデバイスは、操作中にセマフォ ロケーションから読み出すことができます。セマフォ操作が正しく動作するには、必ず **LWX** 命令を **SWX** 命令とペアで使用し、同じアドレスを指定するようにします。**MicroBlaze** では予約設定はワード単位で行われます。両方の命令に対し、アドレスのワードが一致している必要があります。これらの命令に対してはワード不一致の例外が生成されません。

条件付保存は、保存アドレスが予約を設定する読み込みアドレスと一致していない場合でも、予約が存在する場合常に実行されます。

予約は 1 度に 1 つしか管理できません。予約に関連したアドレスは後続の **LWX** 命令を実行すると変更できます。条件付保存は先の **LWX** 命令により確立された予約に基づき実行されます。**SWX** 命令を実行すると、**LWX** で確立されたものとアドレスが一致するかどうかにかかわらず、常にプロセッサで保持されている予約がクリアになります。

リセット、割り込み、例外、およびブレイク (**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** 命令を使用するための一般的なガイドラインは次のようになっています。

- **LWX** および **SWX** 命令はペアで使用し、同じアドレスを使用します。
- ペアになっておらず、別のアドレスに指定されている **SWX** 命令は、プロセッサで保持されている予約をクリアするために使用することができます。

- **LWX** 命令で条件付シーケンスは開始します。その後、メモリ アクセスまたは読み込まれた値に基づいた計算が続きます。シーケンスは **SWX** 命令で終了します。ほとんどの場合、**SWX** 命令のエラーは **LWX** に返され、再試行されます。
- **LWX** 命令で読み込まれた値が 0 でない場合、同期化プリミティブによっては実行するときに **LWX** をペアにしなくてもかまいません。これは、テストとセットのインプリメンテーション例です。

```

loop:  lw      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:

```

- **LWX** 命令で求める値が返されない場合、**LWX** 命令のループを最小限にすることで、パフォーマンスを改善することができます。また、初期値チェックを実行するため普通の読み込み命令を使用して、パフォーマンスを改善することもできます。これは、スピン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
      lw      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** 命令ペアのループを最小限にすることで、前に進める可能性が高まります。古い値は保存実行前にテストする必要があります。順序が逆になっている場合 (読み込みの前に保存が行われる場合)、さらに多くの **SWX** 命令が実行され、**LWX** と **SWX** 間の予約が失われる可能性が高まります。

## レジスタ

MicroBlaze は直交命令セット アーキテクチャで、32 個の 32 ビット汎用レジスタと最大 18 個の 32 ビット特殊用途レジスタが含まれます。特殊用途レジスタの数は、設定するオプションによって異なります。

### 汎用レジスタ

32 個の 32 ビット汎用レジスタには、R0 ~ R31 という番号が付けられています。レジスタ ファイルは、ビットストリームのダウンロード時にリセットされます。リセット値は 0x00000000 です。図 1-2 に汎用レジスタを、表 1-7 に各レジスタの説明およびリセット値 (存在する場合) を示します。

メモ：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 1-2 : R0 ~ R31

表 1-7 : 汎用レジスタ (R0 ~ R31)

ビット	名前	説明	リセット値
0:31	R0	常に 0 です。書き込まれたデータは、すべて破棄されます。	0x00000000
0:31	R1 ~ R13	32 ビット汎用レジスタ。	-
0:31	R14	割り込みの戻りアドレスの格納に使用される 32 ビット レジスタ。	-
0:31	R15	32 ビット汎用レジスタ。ユーザー ベクタの戻りアドレス格納への使用を推奨します。	-
0:31	R16	ブレークの戻りアドレスの格納に使用される 32 ビット レジスタ。	-
0:31	R17	MicroBlaze でハードウェア例外をサポートする場合、ハードウェア例外の原因となる命令に続く命令のアドレスの読み込みに使用。ただし遅延スロットの例外では、BTR が使用されます (「分岐ターゲット レジスタ (BTR)」を参照)。それ以外の場合は汎用レジスタです。	-
0:31	R18 ~ R31	32 ビット汎用レジスタ。	-

汎用レジスタのソフトウェアでの使用規則については、表 3-2 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 (PC)

プログラム カウンタ (PC) は例外命令の 32 ビット アドレスで、MFS 命令を使用して読み出すことは可能ですが、MTS 命令を使用して書き込むことはできません。MFS 命令を使用する場合、PC レジスタは Sa = 0x0000 で指定します。図 1-3 に PC を、表 1-8 に説明およびリセット値を示します。

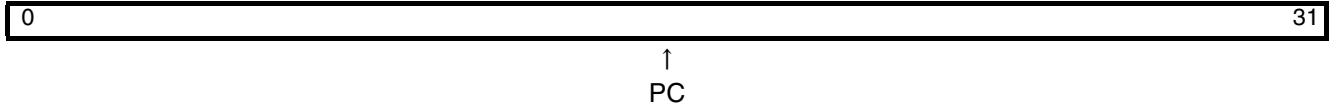


図 1-3 : PC

表 1-8 : プログラム カウンタ (PC)

ビット	名前	説明	リセット値
0:31	PC	プログラム カウンタ 実行命令のアドレス。「mfs r2 0」では、mfs 命令のアドレスが R2 に格納されます。	0x00000000

### 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、プロセッサのコントロール ビットおよびステータス ビットが含まれます。MFS 命令を使用して読み出すことができます。MSR を読み出す際、ビット 29 がキャリー コピーとしてビット 0 に複製されます。MSR は、MTS 命令または専用命令である MSRSET および MSRCLR を使用して書き込むことができます。

MSRSET または MSRCLR を使用して MSR に書き込むと、キャリー ビットはすぐに変化し、その他のビットは 1 クロック サイクル後に変化します。MTS を使用して書き込むと、すべてのビットが 1 クロック サイクル後に有効になります。ビット 0 に書き込まれた値は、すべて破棄されます。

MSR を MTS または MFS 命令と使用する場合は、MSR を Sx = 0x0001 で指定します。図 1-4 に MSR レジスタを、表 1-9 にビットの説明およびリセット値を示します。

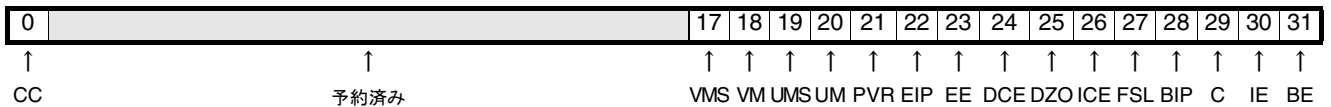


図 1-4 : MSR

表 1-9 : マシン ステータス レジスタ (MSR)

ビット	名前	説明	リセット値
0	CC	演算キャリーのコピー 演算キャリー (ビット 29) のコピーです。CC は常にビット C と同じです。	0
1:16	予約済み		
17	VMS	仮想保護モードでの保存 MMU を使用する場合のみ (C_USE_MMU > 1 かつ C_AREA_OPTIMIZED = 0) 読み出し/書き込み	0
18	VM	仮想保護モード 0 : C_USE_MMU = 3 の場合は MMU アドレス変換およびアクセス保護ディスエーブル、 C_USE_MMU = 2 の場合はアクセス保護ディスエーブル。 1 : C_USE_MMU = 3 の場合は MMU アドレス変換およびアクセス保護イネーブル、 C_USE_MMU = 2 の場合はアクセス保護イネーブル。 MMU を使用する場合のみ (C_USE_MMU > 1 かつ C_AREA_OPTIMIZED = 0) 読み出し/書き込み	0
19	UMS	ユーザー モードでの保存 MMU を使用する場合のみ (C_USE_MMU > 0 かつ C_AREA_OPTIMIZED = 0) 読み出し/書き込み	0
20	UM	ユーザー モード 0 : 特権モード。すべての命令が許可されます。 1 : ユーザー モード。一部の命令は許可されません。 MMU を使用する場合のみ (C_USE_MMU > 0 かつ C_AREA_OPTIMIZED = 0) 読み出し/書き込み	0
21	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 読み出しのみ	C_PVR のパラメータに依存

表 1-9 : マシン ステータス レジスタ (MSR) (続き)

ビット	名前	説明	リセット値
22	EIP	例外実行中 0 : 実行中のハードウェア例外なし 1 : ハードウェア例外を実行中 例外がサポートされている場合のみ (C_*_EXCEPTION または C_USE_MMU) 読み出し/書き込み	0
23	EE	例外イネーブル 0 : ハードウェア例外がディスエーブル <sup>1</sup> 1 : ハードウェア例外がイネーブル 例外がサポートされている場合のみ (C_*_EXCEPTION または C_USE_MMU) 読み出し/書き込み	0
24	DCE	データ キャッシュ イネーブル 0 : データ キャッシュがディスエーブル 1 : データ キャッシュがイネーブル データ キャッシュが使用されている場合のみ (C_USE_DCACHE = 1) 読み出し/書き込み	0
25	DZO	0 での除算または除算オーバーフロー <sup>2</sup> 0 : 0 での除算なし、または除算オーバーフローの発生 1 : 0 での除算あり、または除算オーバーフローの発生 ハードウェア除算器が使用されている場合のみ (C_USE_DIV = 1) 読み出し/書き込み	0
26	ICE	命令キャッシュ イネーブル 0 : 命令キャッシュがディスエーブル 1 : 命令キャッシュがイネーブル 命令キャッシュが使用されている場合のみ (C_USE_ICACHE = 1) 読み出し/書き込み	0

表 1-9 : マシン ステータス レジスタ (MSR) (続き)

ビット	名前	説明	リセット値
27	FSL	FSL エラー  0 : FSL の get/getd/put/putd でエラーなし 1 : FSL の get/getd/put/putd で制御タイプが不一致  FSL リンクが使用されている場合のみ (C_FSL_LINKS > 0)  読み出し/書き込み	0
28	BIP	ブレーク実行中  0 : 実行中のブレークなし 1 : ブレーク実行中  ブレークのソースは、ソフトウェアのブレーク命令または Ext_Brk または Ext_NM_Brk ピンからのハードウェア ブレークのいずれかです。  読み出し/書き込み	0
29	C	演算キャリー  0 : キャリーなし (ボロー) 1 : キャリー (ボローなし)  読み出し/書き込み	0
30	IE	割り込みイネーブル  0 : 割り込みがディスエーブル 1 : 割り込みがイネーブル  読み出し/書き込み	0
31	BE	バスロック イネーブル <sup>3</sup>  0 : データ側 OPB でのバスロック ディスエーブル 1 : データ側 OPB でのバスロック イネーブル  バスロック イネーブルは、IXCL、DXCL、ILMB、DLMB、IPLB、DPLB、または IOPB の動作に影響しません。  データ側 OPB が使用されている場合のみ  読み出し/書き込み	0

1. MMU 例外 (データ格納例外、命令格納例外、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ないため、このビットの影響を受けません。
2. このビットは、整数が 0 で除算されていること、または除算オーバーフローを示す信号でのみ使用します。FSR には、浮動小数点と等価のものがあります。DZO ビットは、プロセッサで例外処理がイネーブルかどうかにかかわらず、0 による除算または除算オーバーフローがあるかどうかを示します。
3. バス プロトコルの詳細は、IBM CoreConnect の仕様『64-Bit On-Chip Peripheral Bus, Architectural Specifications, Version 2.0』を参照してください。

## 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読み込み/格納の完全なアドレスを格納します。

- 不整列アクセスによる例外 (不整列アクセスのアドレス)
- DPLB または DOPB 例外 (実行できなかった PLB または OPB データ アクセスのアドレス)
- データ格納例外 (アクセスされた (仮想) 有効アドレス)
- 命令格納例外 (読み出された (仮想) 有効アドレス)
- データ TLB ミス 例外 (アクセスされた (仮想) 有効アドレス)
- 命令 TLB ミス例外 (読み出された (仮想) 有効アドレス)

その他の例外では、このレジスタの内容は定義されていません。EAR を MFS 命令で読み出す場合、Sa = 0x0003 で指定します。図 1-5 に EAR を、表 1-10 にビットの説明およびリセット値を示します。

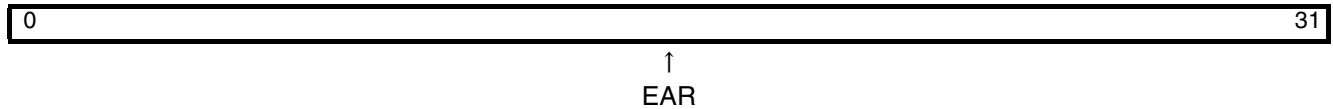


図 1-5 : EAR

表 1-10 : 例外アドレス レジスタ (EAR)

ビット	名前	説明	リセット値
0:31	EAR	例外アドレス レジスタ	0x00000000



## 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 ビットが含まれます。ESR を MFS 命令で読み出す場合、Sa = 0x0005 で指定します。図 1-6 に ESR レジスタを示します。表 1-11 にビットの説明とリセット値を、表 1-12 に例外ステータス (ESS) を示します。

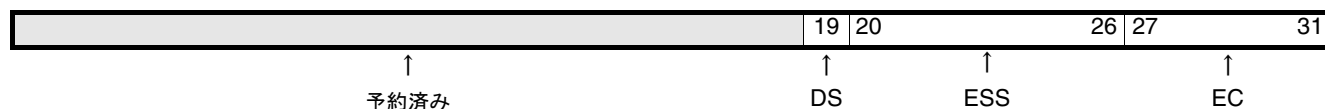


図 1-6 : ESR

表 1-11 : 例外ステータス レジスタ (ESR)

ビット	名前	説明	リセット値
0:18	予約済み		
19	DS	遅延スロットによる例外 0 : 遅延スロット命令による例外なし 1 : 遅延スロット命令による例外あり 読み出しのみ	0
20:26	ESS	例外ステータス 詳細は、表 1-12 を参照してください。 読み出しのみ	表 1-12 を参照
27:31	EC	例外の原因 00000 : 高速シンプレックス リンクの例外 00001 : 不整列データ アクセスによる例外 00010 : 不正な op コードによる例外 00011 : 命令バス エラーによる例外 00011 : データ バス エラーによる例外 00101 : 除算による例外 00110 : 浮動小数点ユニットによる例外 00111 : 特権命令による例外 10000 : データ格納による例外 10001 : 命令格納による例外 10010 : データ TLB ミスによる例外 10011 : 命令 TLB ミスによる例外 読み出しのみ	0

表 1-12 : 例外ステータス (ESS)

例外の原因	ビット	名前	説明	リセット値
不整列データ アクセス	20	W	ワード アクセスによる例外 0 : 不整列ハーフワード アクセス 1 : 不整列ワード アクセス	0
	21	S	格納アクセスによる例外 0 : 不整列読み込みアクセス 1 : 不整列格納アクセス	0
	22:26	Rx	ソース/デスティネーション レジスタ 不整列アクセスでソース (格納) また はデスティネーション (読み込み) と して使用される汎用レジスタ	0
不正な命令	20:26	予約済み		0
命令バス エラー	20:26	予約済み		0
データ バス エラー	20:26	予約済み		0
除算	20	DEC	除算 : 除算例外原因 0 : ゼロでの 除算 1 : 除算オーバーフロー	0
	21:26	予約済み		0
浮動小数点ユニット	20:26	予約済み		0
特権命令	20:26	予約済み		0
高速シンプレックス リンク	20:22	予約済み		0
	23:26	FSL	例外発生の原因となった高速シンプ レックス リンクのインデックス	0
データ格納	20	DIZ	データ格納 - ゾーン保護 0 : 実行されていない 1 : 実行された	0
	21	S	データ格納 - 格納命令 0 : 実行されていない 1 : 実行された	0
	22:26	予約済み		0
命令格納	20	DIZ	命令格納 - ゾーン保護 0 : 実行されていない 1 : 実行された	0
	21:26	予約済み		0

表 1-12 : 例外ステータス (ESS) (続き)

例外の原因	ビット	名前	説明	リセット値
データ TLB ミス	20	予約済み		0
	21	S	データ TLB ミス - 格納命令 0 : 実行されていない 1 : 実行された	0
	22:26	予約済み		0
命令 TLB ミス	20:26	予約済み		0

## 分岐ターゲット レジスタ (BTR)

分岐ターゲット レジスタは、MicroBlaze プロセッサで例外を使用するように設定した場合にのみ存在します。MSR[EIP] = 0 のとき、実行される遅延スロットの分岐命令すべてに対する分岐ターゲット アドレスがレジスタに格納されます。遅延スロットに含まれる命令により例外が発生すると (ESR[DS]=1)、例外ハンドラは R17 に格納されている通常の例外ハンドラの戻りアドレスの代わりに、BTR に格納されているアドレスに命令を戻します。BTR を MFS 命令で読み出す場合、Sa = 0x000B で指定します。図 1-7 に BTR レジスタを、表 1-13 にビットの説明およびリセット値を示します。

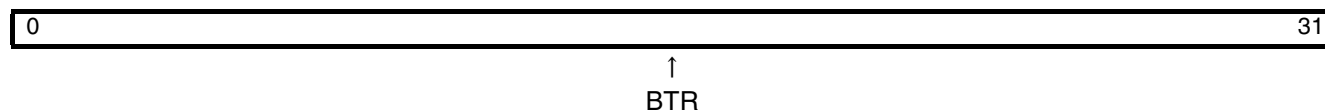


図 1-7 : BTR

表 1-13 : 分岐ターゲット レジスタ (BTR)

ビット	名前	説明	リセット値
0:31	BTR	遅延スロットに含まれる命令により発生する例外から戻るときにハンドラで使用される分岐ターゲット アドレス 読み出しのみ	0x00000000

## 浮動小数点ユニット ステータス レジスタ (FSR)

浮動小数点ユニット 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 ビットが含まれます。MFS 命令で読み出し、MTS 命令で書き込むことができます。読み出しまたは書き込みを実行する場合、Sa = 0x0007 で指定します。浮動小数点ポイント命令はレジスタのビットのみをセットし、レジスタをクリアにするには MTS 命令を使用するしかありません。図 1-8 に FSR レジスタを、表 1-14 にビットの説明およびリセット値を示します。

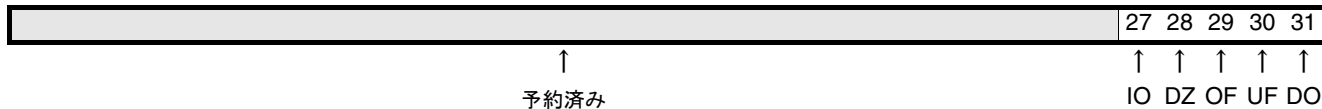


図 1-8 : FSR

表 1-14 : 浮動小数点ユニット ステータス レジスタ (FSR)

ビット	名前	説明	リセット値
0:26	予約済み		未定義
27	IO	不正な操作	0
28	DZ	0 での除算	0
29	OF	オーバーフロー	0
30	UF	アンダーフロー	0
31	DO	非正規化オペランド エラー	0

## 例外データ レジスタ (EDR)

例外データ レジスタには、FSL 例外の原因となった FSL リンクで読み出されたデータが格納されます。

その他の例外では、このレジスタの内容は定義されていません。EDR を MFS 命令で読み出す場合、Sa = 0x000D で指定します。図 1-9 に EDR レジスタを、表 1-15 にビットの説明およびリセット値を示します。

**メモ :** 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。

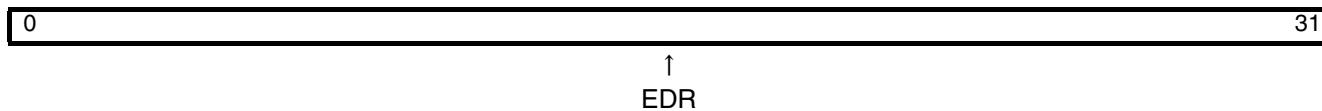


図 1-9 : EDR

表 1-15 : 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

### プロセス識別レジスタ (PID)

MMU アドレス変換の実行中に、ソフトウェア プロセスを識別するために使用します。MicroBlaze では、C\_USE\_MMU コンフィギュレーション オプションで制御します。C\_USE\_MMU が 1 より大きく、C\_AREA\_OPTIMIZED が 0 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 PID にアクセスする場合、Sa = 0x1000 で指定します。このレジスタへのアクセスは、メモリ管理用の特殊なレジスタのパラメータ、C\_MMU\_TLB\_ACCESS の設定に従います。

また、TLB エントリへのアクセスにも使用されます。

- TLBHI (Translation Look-Aside Buffer High) を書き込む際に、PID の値は TLB エントリの TID フィールドに格納されます。
- TLBHI を読み出し、また MSR[UM] が設定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。

図 1-10 に PID レジスタを、表 1-16 にビットの説明およびリセット値を示します。

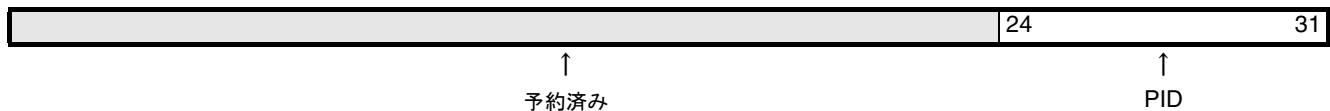


図 1-10 : PID

表 1-16 : プロセス識別レジスタ (PID)

ビット	名前	説明	リセット値
0:23	予約済み		
24:31	PID	MMU アドレス変換の実行中に、ソフトウェア プロセスを識別 読み出し/書き込み	0x00

## ゾーン保護レジスタ (ZPR)

TLB エントリ で定義された MMU メモリ 保護を無効にします。MicroBlaze では、C\_USE\_MMU コンフィギュレーション オプションで制御します。C\_USE\_MMU が 1 より大きく、C\_AREA\_OPTIMIZED が 0、および指定されたメモリ保護ゾーン数が 0 より大きい (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、指定されたメモリ保護ゾーン数 (C\_MMU\_ZONES) によって決まります。MFS および MTS 命令で ZPR にアクセスする場合、Sa = 0x1001 で指定します。このレジスタへのアクセスは、メモリ管理用の特殊なレジスタのパラメータ、C\_MMU\_TLB\_ACCESS の設定に従います。図 1-11 に ZPR レジスタを、表 1-17 にビットの説明およびリセット値を示します。

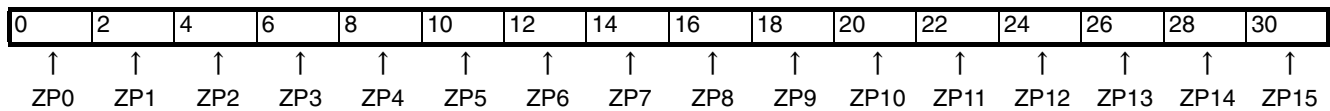


図 1-11 : ZPR

表 1-17 : ゾーン保護レジスタ (ZPR)

ビット	名前	説明	リセット値
0:1	ZP0	ゾーン保護	0x00000000
2:3	ZP1	ユーザー モード (MSR[UM] = 1)	
...	...	00 : TLB エントリの V を無効にする。ページへのアクセスは許可されません。	
30:31	ZP15	01 : 無効になるエントリなし。TLB エントリの V、WR、および EX が使用されます。 10 : 無効になるエントリなし。TLB エントリの V、WR、および EX が使用されます。 11 : TLB エントリの WR および EX を無効にする。書き込みおよび実行可能としてページにアクセスします。	
		特権モード (MSR[UM] = 0) 00 : 無効になるエントリなし。TLB エントリの V、WR、および EX が使用されます。 01 : 無効になるエントリなし。TLB エントリの V、WR、および EX が使用されます。 10 : TLB エントリの WR および EX を無効にする。書き込みおよび実行可能としてページにアクセスします。 11 : TLB エントリの WR および EX を無効にする。書き込みおよび実行可能としてページにアクセスします。	
		読み出し/書き込み	

## 変換ルックアサイド バッファ Low レジスタ (TLBLO)

MMU の UTLB (Unified Translation Look-Aside Buffer) エントリ へのアクセスに使用します。MicroBlaze では、C\_USE\_MMU コンフィギュレーション オプションで制御します。C\_USE\_MMU が 1 より大きく、C\_AREA\_OPTIMIZED が 0 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 TLBLO にアクセスする場合、Sa = 0x1003 で指定します。TLBLO の読み出しまたは書き込みでは、TLBX レジスタでインデックスを付けられた UTLB エントリがアクセスされます。このレジスタの読み出しは、メモリ管理用の特殊なレジスタのパラメータ、C\_MMU\_TLB\_ACCESS の設定に従います。

UTLB は、ビットストリームがダウンロードされるとリセットされます。リセット値はすべての TLBLO エントリで 0x00000000 です。

メモ：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。

図 1-12 に TLBLO レジスタを、表 1-18 にビットの説明およびリセット値を示します。

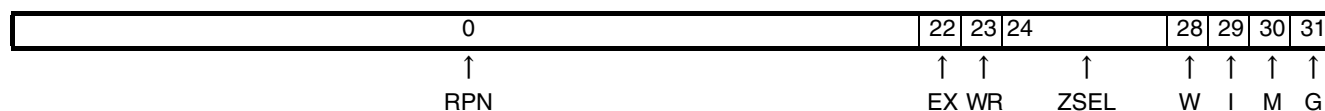


図 1-12 : TLBLO

表 1-18 : 変換ルックアサイド バッファ Low レジスタ (TLBLO)

ビット	名前	説明	リセット値
0:21	RPN	実 (物理) ページ番号  TLB ヒットが発生すると、このフィールドが TLB エントリから読み出され、物理アドレスの作成に使用されます。SIZE フィールドの値によっては RPN ビットの一部は物理アドレスには使用されないため、ソフトウェアでこのフィールドの未使用ビットを 0 にする必要があります。  C_USE_MMU=3 の場合のみ 読み出し/書き込み	0x000000
22	EX	実行ファイル  1 に設定するとページに実行コードが含まれ、命令はページよりフェッチできます。0 の場合は、命令はページから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すると、命令格納例外が発生します。 読み出し/書き込み	0

表 1-18 : 変換ルックアサイド バッファ Low レジスタ (TLBLO) (続き)

ビット	名前	説明	リセット値
23	WR	書き込み可能 1 に設定するとページは書き込み可能で、格納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、ページは読み出しのみで、書き込みできません。WR ビットが 0 のページにデータを格納すると、データ格納例外が発生します。 読み出し/書き込み	0
24:27	ZSEL	ゾーン選択 ゾーン保護レジスタ (ZPR) の 16 個のゾーン フィールド (Z0 ~ Z15) から 1 個を選択します。 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 フィールドおよび WR フィールドで指定されたアクセス保護の変更に使用されます。また、TLB エントリの V (有効) フィールドを無効にし、ページがアクセスされないようにします。 読み出し/書き込み	0x0
28	W	ライト スルー パラメータ C_DCACHE_USE_WRITEBACK が 1 に設定されている場合、このビットはキャッシュ ポリシーを制御します。ライトスルーは 1 に設定されている場合に選択され、それ以外の場合はライトバックが選択されます。 C_DCACHE_USE_WRITEBACK が 0 にクリアされない限り、このビットは 1 に固定されていて、ライトスルーが常に使用されます。 読み出し/書き込み	1
29	I	キャッシュの抑止 1 に設定されると、ページへのアクセスはキャッシュされません。 0 の場合は、ページへのアクセスはキャッシュ可能です。 読み出し/書き込み	0



表 1-18 : 変換ルックアサイド バッファ Low レジスタ (TLBLO) (続き)

ビット	名前	説明	リセット値
30	M	メモリ コヒーレント メモリのコヒーレンスは <b>MicroBlaze</b> ではインプリメントされないため、このビットは <b>0</b> に固定されています。 読み出しのみ	0
31	G	保護 ビットが <b>1</b> に設定されると、メモリは保護され、論理的なページ アクセスは許可されません。 <b>0</b> の場合は、論理的なページ アクセスが許可されます。 <b>G</b> 属性は、メモリ マップされた <b>I/O</b> デバイスが不適切な命令からアクセスされないように保護します。 読み出し/書き込み	0

## 変換ルックアサイド バッファ High レジスタ (TLBHI)

MMU の UTLB (Unified Translation Look-Aside Buffer) エントリ へのアクセスに使用します。**MicroBlaze** では、**C\_USE\_MMU** コンフィギュレーション オプションで制御します。**C\_USE\_MMU** が **1** より大きく、**C\_AREA\_OPTIMIZED** が **0** 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 **TLBHI** にアクセスする場合、**Sa = 0x1004** で指定します。**TLBHI** の読み出しまたは書き込みでは、**TLBX** レジスタでインデックスを付けられた **UTLB** エントリがアクセスされます。このレジスタの読み出しは、メモリ管理用の特殊なレジスタのパラメータ、**C\_MMU\_TLB\_ACCESS** の設定に従います。

また、**TLB** エントリへのアクセスにも使用されます。

- 書き込みでは、**PID** の値が **TLB** エントリの **TID** フィールドに格納されます。
- 読み出して、**MSR[UM]** が設定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、ビットストリームがダウンロードされるとリセットされます。リセット値はすべての **TLBHI** エントリで **0x00000000** です。

**メモ** : **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。

図 1-13 に **TLBHI** レジスタを、表 1-19 にビットの説明およびリセット値を示します。

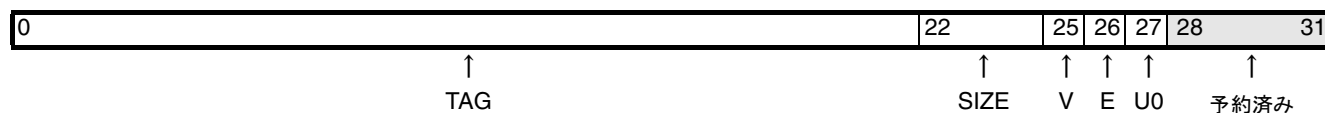


図 1-13 : TLBHI

表 1-19 : 変換ルックアサイド バッファ High レジスタ (TLBHI)

ビット	名前	説明	リセット値
0:21	TAG	TLB エントリ タグ 仮想メモリ アドレスのページ番号部分と比較されます。 比較は <b>SIZE</b> フィールドで制御されます。 読み出し/書き込み	0x000000
22:24	SIZE	サイズ ページ サイズを指定します。 <b>TAG</b> フィールドと仮想メモリアドレスのページ番号部分の比較で使用されるビットの範囲を制御します。このフィールドで定義されるページ サイズは表 1-34 を参照してください。 読み出し/書き込み	000
25	V	有効 1 に設定された場合、 <b>TLB</b> エントリは有効で、ページ変換エントリを含んでいます。 0 の場合は、 <b>TLB</b> エントリは無効です。 読み出し/書き込み	0
26	E	エンディアン <b>MicroBlaze</b> のページへのアクセスは常にビッグ エンディアン形式であるため、このビットは 0 に固定されています。 読み出しのみ	0
27	U0	ユーザー定義 <b>MicroBlaze</b> にはユーザー定義の格納属性がないため、このビットは 0 に固定されています。 読み出しのみ	0
28:31	予約済み		

## 変換ルックアサイド バッファ インデックス レジスタ (TLBX)

TLBLO および TLBHI レジスタにアクセスする場合に、UTLB (Unified Translation Look-Aside Buffer) へのインデックスとして使用します。MicroBlaze では、C\_USE\_MMU コンフィギュレーション オプションで制御します。C\_USE\_MMU が 1 より大きく、C\_AREA\_OPTIMIZED が 0 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 TLBX にアクセスする場合、Sa = 0x1002 で指定します。図 1-14 に TLBX レジスタを、表 1-20 にビットの説明およびリセット値を示します。

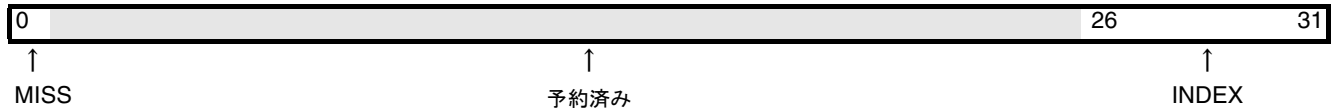


図 1-14 : TLBX

表 1-20 : 変換ルックアサイド バッファ インデックス レジスタ (TLBX)

ビット	名前	説明	リセット値
0	MISS	<p>TLB ミス</p> <p>TLBSX レジスタが、TLB エントリに含まれた仮想アドレスを使用して書き込まれると、0 になります。仮想アドレスが TLB エントリに含まれない場合は 1 になります。TLBX レジスタ自身が書き込まれた場合も 0 になります。</p> <p>読み出しのみ</p> <p>メモリ管理用の特殊レジスタのパラメータ C_MMU_TLB_ACCESS &gt; 0 の場合に読み出すことができます。</p>	0
1:25	予約済み		
26:31	INDEX	<p>TLB インデックス</p> <p>TLBLO および TLBHI レジスタがアクセスする TLB エントリのインデックスに使用されます。TLBSX レジスタが、対応する TLB エントリに含まれた仮想アドレスを使用して書き込まれると、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>メモリ管理用の特殊レジスタのパラメータが C_MMU_TLB_ACCESS &gt; 0 の場合に読み出しおよび書き込みができます。</p>	000000

変換ルックアサイド バッファ検索インデックス レジスタ (TLBSX)

UTLB (Unified Translation Look-Aside Buffer ) での仮想ページ番号の検索に使用します。MicroBlaze では、C\_USE\_MMU コンフィギュレーション オプションで制御します。C\_USE\_MMU が 1 より大きく、C\_AREA\_OPTIMIZED が 0 に設定されている場合にのみインプリメントされます。TLBSX を MTS 命令で書き込む場合、Sa = 0x1005 で指定します。図 1-15 に TLBSX レジスタを、表 1-21 にビットの説明およびリセット値を示します。

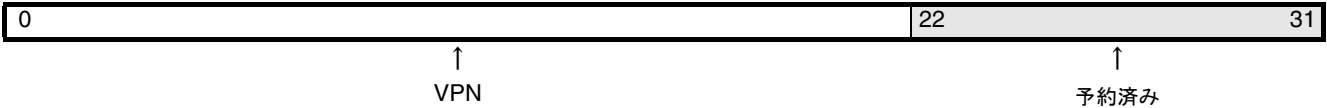


図 1-15 : TLBSX

表 1-21 : 変換ルックアサイド バッファ インデックス検索レジスタ (TLBSX)

ビット	名前	説明	リセット値
0:21	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、仮想メモリ アドレスのページ番号部分と比較されます。比較は、V ビットが 1 に設定されている各 TLB エントリの SIZE フィールドで制御されます。  仮想ページ番号が検出された場合は、TLBX レジスタは TLB エントリのインデックスを使用して書き込まれ、TLBX の MISS ビットは 0 になります。TLB エントリで仮想ページ番号が検出されない場合は、TLBX レジスタの MISS ビットが 1 に設定されます。  書き込みのみ	
22:31	予約済み		

## 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、C\_PVR コンフィギュレーション オプションにより制御されます。

- C\_PVR を 0 に設定すると PVR はインプリメントされず、MSR[PVR]=0 になります。
- 1 に設定すると最初のレジスタ PVR0 のみがインプリメントされ、2 に設定すると PVR0 ～ PVR11 の 12 個の PVR レジスタすべてがインプリメントされます。

PVR を MFS 命令で読み出す場合、Sa = 0x200x で指定します。x は 0x0 ～ 0xB のレジスタ番号で指定します。

表 1-22 ～ 表 1-33 に、ビットの説明および値を示します。

表 1-22 : プロセッサ バージョン レジスタ 0 (PVR0)

ビット	名前	説明	値
0	CFG	PVR のインプリメンテーション : 0 = 基本、1 = フル	C_PVR の値に依存
1	BS	バレル シフタの使用	C_USE_BARREL
2	DIV	除算器の使用	C_USE_DIV
3	MUL	ハードウェア乗算器の使用	C_USE_HW_MUL > 0
4	FPU	FPU の使用	C_USE_FPU > 0
5	EXC	例外の使用	C_*_EXCEPTION の値に依存 また、C_USE_MMU > 0 の場合
6	ICU	命令キャッシュの使用	C_USE_ICACHE
7	DCU	データ キャッシュの使用	C_USE_DCACHE
8	MMU	MMU の使用	C_USE_MMU > 0
9:15	予約済み		0

表 1-22 : プロセッサ バージョン レジスタ 0 (PVR0) (続き)

ビット	名前	説明	値
16:23	MBV	MicroBlaze リリース バージョン コード 0x1 = v5.00.a 0x2 = v5.00.b 0x3 = v5.00.c 0x4 = v6.00.a 0x6 = v6.00.b 0x5 = v7.00.a 0x7 = v7.00.b 0x8 = v7.10.a 0x9 = v7.10.b 0xA = v7.10.c 0xB = v7.10.d 0xC = v7.20.a 0xD = v7.20.b 0xE = v7.20.c 0xF = v7.20.d	各リリース固有
24:31	USR1	ユーザー コンフィギュレーション 値 1	C_PVR_USER1

表 1-23 : プロセッサ バージョン レジスタ 1 (PVR1)

ビット	名前	説明	値
0:31	USR2	ユーザー コンフィギュレーション 値 2	C_PVR_USER2

表 1-24 : プロセッサ バージョン レジスタ 2 (PVR2)

ビット	名前	説明	値
0	DOPB	使用中のデータ側 OPB	C_D_OPB
1	DLMB	使用中のデータ側 LMB	C_D_LMB
2	IOPB	使用中の命令側 OPB	C_I_OPB
3	ILMB	使用中の命令側 LMB	C_I_LMB
4	IRQEDGE	エッジでトリガされる割り込み	C_INTERRUPT_IS_EDGE
5	IRQPOS	立ち上がりエッジへの割り込み	C_EDGE_IS_POSITIVE
6	DPLB	使用中のデータ側 PLB	C_D_PLB

表 1-24 : プロセッサ バージョン レジスタ 2 (PVR2) (続き)

ビット	名前	説明	値
7	IPLB	使用中の命令側 PLB	C_I_PLB
8	INTERCON	PLB インターコネクトの使用	C_INTERCONNECT
9:11	予約済み		
12	FSL	拡張 FSL 命令の使用	C_USE_EXTENDED_FSL_INSTR
13	FSLEXC	FSL 制御ビット不一致による例外の生成	C_FSL_EXCEPTION
14	MSR	MSRSET 命令および MSRCLR 命令の使用	C_USE_MSR_INSTR
15	PCMP	パターン比較命令の使用	C_USE_PCMP_INSTR
16	AREA	低い命令スループットでエリアを最適化するためのインプリメンテーションの選択	C_AREA_OPTIMIZED
17	BS	バレル シフタの使用	C_USE_BARREL
18	DIV	除算器の使用	C_USE_DIV
19	MUL	ハードウェア乗算器の使用	C_USE_HW_MUL > 0
20	FPU	FPU の使用	C_USE_FPU > 0
21	MUL64	64 ビット ハードウェア乗算器の使用	C_USE_HW_MUL = 2
22	FPU2	浮動小数点変換および平方根命令の使用	C_USE_FPU = 2
23	IPLBEXC	IPLB エラーによる例外の生成	C_IPLB_BUS_EXCEPTION
24	DPLBEXC	DPLB エラーによる例外の生成	C_DPLB_BUS_EXCEPTION
25	OP0EXC	不正な op コード 0x0 による例外の生成	C_OPCODE_0x0_ILLEGAL
26	UNEXC	不整列データ アクセスによる例外の生成	C_UNALIGNED_EXCEPTIONS
27	OPEXC	不正な op コードによる例外の生成	C_ILL_OPCODE_EXCEPTION
28	IOPBEXC	IOPB エラーによる例外の生成	C_IOPB_BUS_EXCEPTION
29	DOPBEXC	DOPB エラーによる例外の生成	C_DOPB_BUS_EXCEPTION
30	DIVEXC	0 での除算または除算オーバーフローによる例外の生成	C_DIV_ZERO_EXCEPTION
31	FPUEXC	FPU による例外の生成	C_FPU_EXCEPTION

表 1-25 : プロセッサ バージョン レジスタ 3 (PVR3)

ビット	名前	説明	値
0	DEBUG	デバッグ ロジックの使用	C_DEBUG_ENABLED
1:2	予約済み		
3:6	PCBRK	PC ブレークポイントの数	C_NUMBER_OF_PC_BRK
7:9	予約済み		
10:12	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	予約済み		
16:18	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	予約済み		
20:24	FSL	FSL の数	C_FSL_LINKS
25:31	予約済み		

表 1-26 : プロセッサ バージョン レジスタ 4 (PVR4)

ビット	名前	説明	値
0	ICU	命令キャッシュの使用	C_USE_ICACHE
1:5	ICTS	命令キャッシュのタグ サイズ	C_ADDR_TAG_BITS
6	予約済み		1
7	ICW	命令キャッシュの書き込み許可	C_ALLOW_ICACHE_WR
8:10	ICLL	2 を底とする命令キャッシュ ラインの長さの対数	$\log_2(C\_ICACHE\_LINE\_LEN)$
11:15	ICBS	2 を底とする命令キャッシュのバイト サイズの対数	$\log_2(C\_CACHE\_BYTE\_SIZE)$
16	IAU	すべてのメモリ アクセスへの命令キャッシュの使用	C_ICACHE_ALWAYS_USED
17	予約済み		0
18	ICI	命令キャッシュ XCL プロトコル	C_ICACHE_INTERFACE
19:31	予約済み		0

表 1-27 : プロセッサ バージョン レジスタ 5 (PVR5)

ビット	名前	説明	値
0	DCU	データ キャッシュの使用	C_USE_DCACHE
1:5	DCTS	データ キャッシュのタグ サイズ	C_DCACHE_ADDR_TAG



表 1-27 : プロセッサ バージョン レジスタ 5 (PVR5) (続き)

ビット	名前	説明	値
6	予約済み		1
7	DCW	データ キャッシュの書き込みを許可	C_ALLOW_DCACHE_WR
8:10	DCLL	2 を底とするデータ キャッシュ ラインの長さの対数	log2(C_DCACHE_LINE_LEN)
11:15	DCBS	2 を底とするデータ キャッシュのバイト サイズの対数	log2(C_DCACHE_BYTE_SIZE)
16	DAU	すべてのメモリ アクセスへのデータ キャッシュの使用	C_DCACHE_ALWAYS_USED
17	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	DCI	データ キャッシュ XCL プロトコル	C_DCACHE_INTERFACE
19:31	予約済み		0

表 1-28 : プロセッサ バージョン レジスタ 6 (PVR6)

ビット	名前	説明	値
0:31	ICBA	命令キャッシュのベース アドレス	C_ICACHE_BASEADDR

表 1-29 : プロセッサ バージョン レジスタ 7 (PVR7)

ビット	名前	説明	値
0:31	ICHA	命令キャッシュのハイ アドレス	C_ICACHE_HIGHADDR

表 1-30 : プロセッサ バージョン レジスタ 8 (PVR8)

ビット	名前	説明	値
0:31	DCBA	データ キャッシュのベース アドレス	C_DCACHE_BASEADDR

表 1-31 : プロセッサ バージョン レジスタ 9 (PVR9)

ビット	名前	説明	値
0:31	DCHA	データ キャッシュのハイ アドレス	C_DCACHE_HIGHADDR

表 1-32 : プロセッサ バージョン レジスタ 10 (PVR10)

ビット	名前	説明	値
0:7	ARCH	ターゲット アーキテクチャ : 0x6 : Spartan®-3、XA Spartan-3 0x7 : Virtex-4、 QPro Virtex-4 Hi Rel QPro Virtex-4 Rad トラレント 0x8 : Virtex-5 0x9 : Spartan-3E、XA Spartan-3E 0xA : Spartan-3A、XA Spartan-3A 0xB : Spartan-3AN 0xC : Spartan-3A DSP XA Spartan-3A DSP 0xD : Spartan-6 Virtex-6	パラメータ C_FAMILY により定義
8:31	予約済み		0

表 1-33 : プロセッサ バージョン レジスタ 11 (PVR11)

ビット	名前	説明	値
0:1	MMU	MMU の使用 0 : なし 1 : ユーザー モード 2 : 保護 3 : 仮想	C_USE_MMU
2:4	ITLB	命令シャドウ TLB のサイズ	log2(C_MMU_ITLB_SIZE)
5:7	DTLB	データ シャドウ TLB のサイズ	log2(C_MMU_DTLB_SIZE)

表 1-33 : プロセッサ バージョン レジスタ 11 (PVR11) (続き)

ビット	名前	説明	値
8:9	TLBACC	TLB レジスタのアクセス : 0 : 最小 1 : 読み出し 2 : 書き込み 3 : フル アクセス	C_MMU_TLB_ACCESS
10:14	ZONES	メモリ保護ゾーンの数	C_MMU_ZONES
15:16	予約済み	将来の使用に予約	0
17:31	RSTMSR	MSR のリセット値	C_RESET_MSR

## パイプライン アーキテクチャ

MicroBlaze では、命令の実行にパイプライン処理を使用します。ほとんどの命令では、各段を完了するのに 1 クロック サイクルかかるため、1 つの命令を完了するのに必要なクロック サイクル数は、パイプライン段数に等しく、サイクルごとに命令が 1 つ完了します。実行段に複数のクロック サイクルを要する命令もありますが、その場合は命令が完了するまで後続命令がストールします。

処理速度が低いメモリから実行する場合、フェッチ段に複数サイクルかかることがあります。このレイテンシは、パイプラインの効率に直接影響します。MicroBlaze では命令プリフェッチ バッファがインプリメントされ、このようなレイテンシの影響が軽減されます。命令実行段に複数サイクルかかる命令でパイプラインがストールされる場合でも、プリフェッチ バッファに後続命令が読み込まれます。パイプラインがフェッチ段を実行できるようになると、プリフェッチ バッファから新しい命令を読み込むことができるので、命令メモリ アクセスの完了を待つ必要はありません。命令が自動変更コードなどで実行中に変更される場合、この変更された命令を実行する前にプリフェッチ バッファを空にして、変更前の命令が含まれていないようにする必要があります。このようにするには、BRI4 などの同期分岐命令を使用します。

### 3 段パイプライン

C\_AREA\_OPTIMIZED が 1 に設定されている場合は、ハードウェアの使用量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 段に分かれます。

	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

### 5 段パイプライン

C\_AREA\_OPTIMIZED が 0 に設定されている場合は、最大のパフォーマンスを得るため、パイプラインはフェッチ (IF)、デコード (OF)、実行 (EX)、メモリ アクセス (MEM)、ライトバック (WB) の 5 段に分かれます。

	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

### 分岐

分岐が実行されると、通常はフェッチ段およびデコード段 (プリフェッチ バッファも含む) にある命令は破棄され、実行される分岐アドレスから新しい命令がフェッチ パイプライン段に読み込まれます。MicroBlaze では分岐の実行に 3 クロック サイクルかかりますが、そのうち 2 サイクルがパイプラインのリフィルに必要です。このオーバーヘッドを少しでも削減するため、MicroBlaze では遅延スロットを使用する分岐がサポートされています。

## 遅延スロット

遅延スロットを使用する分岐が実行されると、フェッチ段階の命令のみが破棄され、デコード段 (分岐遅延スロット) は完了されます。これにより、分岐によるペナルティが 2 クロック サイクルから 1 クロック サイクルに軽減されます。遅延スロットを使用する分岐命令には、命令ニーモニックの後尾に **D** を追加します。たとえば、**BNE** 命令 (遅延スロットなし) では後続命令は実行されませんが、**BNED** 命令では分岐ロケーションに移動する前に後続命令が実行されます。

遅延スロットには、**IMM**、分岐、またはブレイク命令を含まないようにします。割り込みおよび外部ハードウェアのブレイクは、遅延スロットを使用する分岐が完了するまで保留されます。

不整列ワードまたはハーフワードの読み込みおよび格納など、回復が可能な例外が発生する可能性がある命令は、遅延スロットに含めることができます。遅延スロットで例外が発生した場合、**ESR[DS]** ビットが設定され、例外ハンドラは特殊用途レジスタの **BTR** に格納されている分岐ターゲットに命令を戻します。レジスタ **R17** は、**ESR[DS]** ビットが設定されていると無効になり、設定されていない場合は、例外が発生した命令の後続のアドレスを含みます。

## メモリ アーキテクチャ

MicroBlaze のメモリはハーバード アーキテクチャ構造であり、命令アクセスおよびデータ アクセスに別々のアドレス空間が使用されます。各アドレス空間は 32 ビットの範囲で、それぞれ 4GB までの命令メモリおよびデータ メモリを処理できます。命令メモリおよびデータ メモリの範囲は、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。これは、ソフトウェアのデバッグで有益です。

MicroBlaze の命令およびデータ インターフェイスは 32 ビット幅で、ビッグ エンディアン形式です。データ メモリへは、ワード、ハーフワード、バイトでアクセスできます。

データ アクセスは、プロセッサで不整列の例外がサポートされている場合を除き、ワード アクセスはワードの境界で、ハーフワード アクセスはハーフワードの境界で整列させる必要があります。命令アクセスは、すべてワードの境界で整列させる必要があります。

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ず、メモリ マップされた I/O が使用されます。メモリ アクセスには、次の 3 つのインターフェイスを使用できます。

- ローカル メモリ バス (LMB)
- プロセッサ ローカル バス (PLB) または オンチップ ペリフェラル バス (OPB)
- ザイリンクス CacheLink (XCL)

LMB メモリ アドレス範囲は、PLB、OPB、または XCL のアドレス範囲と重ならないようにする必要があります。

MicroBlaze では、ローカル メモリ (LMB) へのアクセスおよびキャッシュの読み出しのレイテンシは 1 クロック サイクルです。ただし、C\_AREA\_OPTIMIZED が 1 に設定されている場合のデータ側アクセスおよびデータ キャッシュの読み出しは 2 クロック サイクルです。データ キャッシュ書き込みレイテンシは C\_DCACHE\_USE\_WRITEBACK で決まります。

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必要がある場合はレイテンシはそれ以上)。

C\_DCACHE\_USE\_WRITEBACK が 0 にクリアされている場合、書き込みレイテンシは通常 2 サイクルです (ポストエド書き込みバッファがフルの場合は、レイテンシはそれ以上)。

MicroBlaze の命令およびデータ キャッシュでは、4 ワードまたは 8 ワードのキャッシュ ラインを使用できます。キャッシュ ラインを長くする場合はバイト長も増やすと、通常シーケンシャル アクセス パターンのソフトウェアのパフォーマンスが向上します。ランダム アクセス パターンのソフトウェアでは、キャッシュ サイズによってはパフォーマンスが低下することがあります。これは、使用可能なキャッシュ ラインが少なくなり、キャッシュのヒット率が下がるためです。

各メモリ インターフェイスの詳細は、第 2 章「MicroBlaze の信号インターフェイス」を参照してください。

## 特権命令

次の命令は、特権命令です。

- GET、PUT、NGET、NPUT、CGET、CPUT、NCGET、NCPUT
- WIC、WDC
- MTS
- 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 0x8 または 0x18 にジャンプする場合を除く)

ユーザー モードでこれらの命令を使用すると、特権命令例外が発生します。

次のいずれかでユーザー モードから仮想モードに切り替わります。

1. ハードウェア生成されたリセット (デバッグ リセットを含む)
2. ハードウェア例外
3. マスク不可のブレークまたはハードウェア ブレーク
4. 割り込み
5. 命令「BRALID Re, 0x8」を実行し、ユーザー ベクタ例外を発生させる
6. ソフトウェア ブレーク命令 BRKI を実行し、物理アドレス 0x8 または 0x18 にジャンプする

ハードウェア生成されたリセット以外では、ユーザー モードおよび仮想モードのステータスは、MSR の UMS ビットおよび 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) プログラムでは、BRALID 命令または BRKI 命令を使用して物理アドレス 0x8 にジャンプし、制御をシステム サービス ルーチン (特権モード プログラム) に転送します。この命令を実行すると、システム呼び出し例外が発生します。例外ハンドラで、どのシステム サービス ルーチンを呼び出すかおよび呼び出し側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が確認されます。権限がある場合は、アプリケーション プログラムに代わり、例外ハンドラでシステム サービス ルーチンへのプロシージャ呼び出しが実行されます。

システム サービス ルーチンに必要な実行環境の設定には、プロローグ命令を実行する必要があります。これらの命令を実行すると通常、プロシージャの情報 (アクティブ化の記録) を保持するストレージのブロックが作成され、ポインタがアップデートおよび初期化され、システム サービス ルーチンで使用する揮発性レジスタが保存されます。プロローグ コードは実行モジュール作成時にリンクによって挿入できます。システム呼び出し割り込みハンドラまたはシステム ライブラリ ルーチンのどちらかに、スタブ コードとして含めることもできます。

システム サービス ルーチンから戻ると、上記の手順で実行された内容が無効になります。エピローグ コードを実行すると、アクティブ化の記録が巻き戻しおよび解放され、ポインタが復元され、揮発性レジスタが復元されます。割り込みハンドラでは、例外命令 (RTED) からアプリケーションへの戻りが実行されます。

## 仮想メモリ管理

MicroBlaze で実行されるプログラムでは、有効アドレスを使用して 4GB のフラット アドレス空間にアクセスします。プロセッサでは、変換モードに応じて次の方法でこのアドレス空間に割り込みます。

- 実モード：有効なアドレスを使用して物理メモリに直接アクセス
- 仮想モード：有効なアドレスは、プロセッサの仮想メモリ管理ハードウェアにより物理アドレスに変換される

仮想メモリ 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どの位置にでも再配置できます。アクティブなプログラムおよびデータに領域が必要な場合は、システム ソフトウェアでアクティブでないプログラムおよびデータを物理メモリから移動します。

再配置すると、システムが実際にインプリメントするよりも多くのメモリをプログラムで使用できるようになり、システムの物理メモリの制限がなくなります。また、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知る必要もなくなります。プログラムで使用できるアドレスは、プロセッサで物理アドレスに変換されます。

仮想モードでは、メモリ保護を詳細に制御できます。1KB 程度の小さなメモリ ブロックでも個別に不正アクセスから保護できます。保護および再配置の機能により、システム ソフトウェアでマルチタスクがサポートされます。マルチタスクでは、複数のプログラムが同時に実行されます。

MicroBlaze では、仮想モードは MMU (Memory Management Unit) によりインプリメントされます。MMU は `C_USE_MMU` が 3 および `C_AREA_OPTIMIZED` が 0 に設定されている場合に有効です。MMU では、有効アドレスから物理アドレスへの対応付けが制御され、メモリ保護がサポートされます。これらの機能を使用すると、仮想メモリのデマンド ページングおよびその他のメモリ管理スキーマをシステム ソフトウェアでインプリメントできるようになります。

MicroBlaze の MMU インプリメンテーションは、PowerPC™ 405 に基づいています。詳細は、『PowerPC Processor Reference Guide』を参照してください。

MMU の機能の概要は次のとおりです。

- 有効アドレスを物理アドレスへ変換
- アドレス変換中の、ページ レベルでのアクセスの制御
- ゾーンを使用した、仮想モードによる保護制御
- 命令アドレスおよびデータ アドレス変換および保護の個別の制御
- 1KB、4KB、16KB、64KB、256KB、1MB、4MB、および 16MB の 8 とおりのページ サイズのサポート。システム ソフトウェアでは、どのページ サイズの組み合わせも使用可能です。
- ソフトウェアによるページ置き換えの制御

## 実モード

プロセッサは、命令をフェッチする場合および読み込みまたは格納命令でデータにアクセスする場合にメモリを参照します。プログラムでは、プロセッサより算出された 32 ビットの有効アドレスを使用してメモリ位置を参照します。実モードがイネーブルの場合は、物理アドレスは有効アドレスに等しく、プロセッサで物理メモリへのアクセスに使用されます。プロセッサは、リセットされると実モードで動作します。実モードは、MSR の VM ビットを 0 にしても、イネーブルになります。

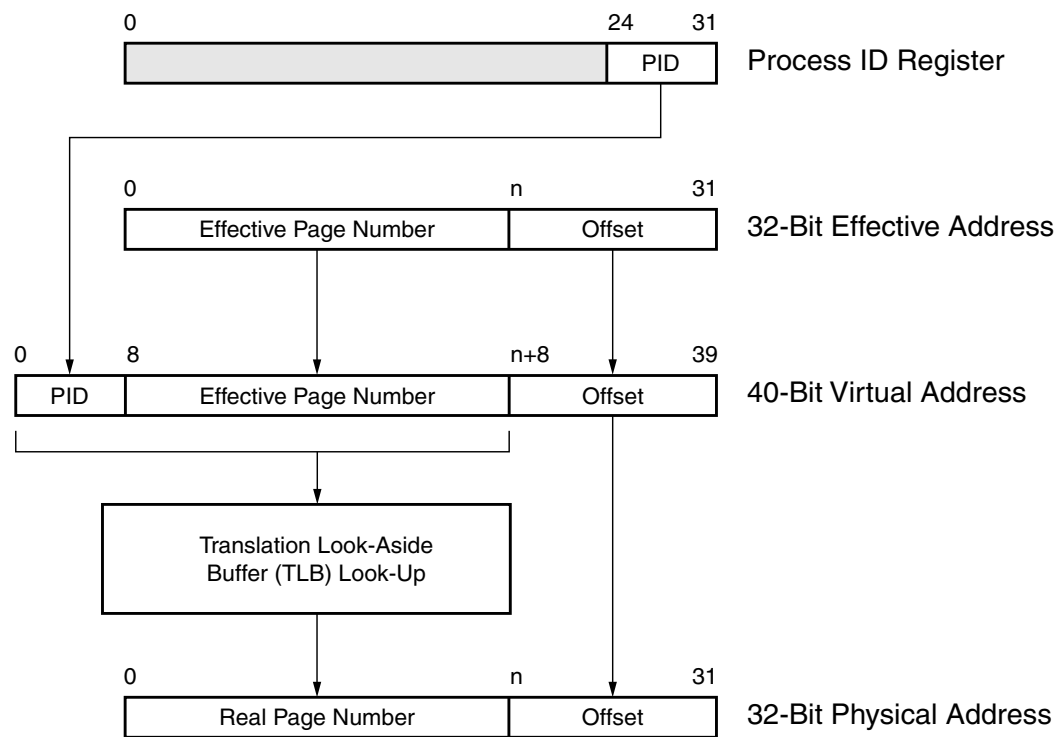
物理メモリ データ アクセス (読み込みおよび格納) は、有効アドレスを使用して実モードで実行されます。実モードでは、システム ソフトウェアで仮想アドレス変換を使用できませんが、メモリアccessの完全な保護は、`C_USE_MMU > 1` および `C_AREA_OPTIMIZED = 0` の場合にインプリメ



ントされます。実モードのメモリ管理のインプリメンテーションは、仮想モードよりも容易です。単純なエンベデッド環境で、アクセス保護が必要で仮想アドレス変換が必要でない場合は、実モードの使用が適しています。

## 仮想モード

仮想モードでは、図 1-16 に示される手順で、プロセッサで有効アドレスが物理アドレスに変換されます。仮想モードは、MSR の VM ビットを 1 にしても、イネーブルになります。



UG011\_37\_021302

図 1-16 : 仮想モードのアドレス変換

図 1-16 の各アドレスには、ページ番号フィールドおよびオフセット フィールドが含まれています。ページ番号は、MMU で変換されたアドレスの一部です。オフセットは ページへのバイト オフセットで、MMU で変換されたものではありません。仮想アドレスは、PID レジスタに含まれるプロセス ID (PID) というフィールドで構成されています (37 ページの「プロセス識別レジスタ (PID)」を参照)。PID と有効ページ番号 (EPN) を組み合わせて、仮想ページ番号 (VPN) と呼びます。この値は、表 1-34 に示すように、ページサイズで決まります。

システム ソフトウェアでは、各仮想ページを物理ページに変換する際に使用されるエントリを含む、ページ変換テーブルが維持されます。ページ変換エントリによって定義されるページ サイズにより、ページ番号フィールドおよびオフセット フィールドのサイズが決まります。ページサイズが 4KB の場合、ページ番号フィールドは 20 ビット、オフセット フィールドは 12 ビットになります。この場合、VPN は 28 ビットになります。

最も頻繁に使用されるページ変換は、TLB (Translation Look-Aside Buffer) に格納されます。仮想アドレスの変換時に MMU で、一致する VPN (PID および EPN) のページ変換エントリが確認されます。表のエントリすべてではなく、プロセッサ TLB に含まれるエントリのみが確認されます。VPN が一致するページ変換エントリが検出されると、対応する物理ページ番号がエントリから読み出さ

れ、オフセットと組み合わせて 32 ビット物理アドレスが構成されます。プロセッサではこの物理アドレスを使用して、メモリを参照します。

システム ソフトウェアでは PID を使用して、プロセッサで実行されているソフトウェア プロセス (タスク、サブルーチン、スレッド) を識別します。個別にコンパイルされた複数のプロセッサが、重なった有効アドレス領域で動作することがありますが、マルチタスクがサポートされている場合はシステム ソフトウェアで領域が重ならないようにする必要があります。各プロセスに PID を割り当てると、システム ソフトウェアで各プロセスを、仮想アドレス空間が重ならない領域に再配置できるようになります。仮想アドレス空間を対応付けると、各プロセスを個別に物理アドレス空間に変換できるようになります。

## 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たデータ構造で、ページ変換に必要な情報を含んでいます。ソフトウェアで管理されるページ変換の要件は、エンベデッド システム アプリケーションがターゲットとするアーキテクチャのトレードオフを表します。エンベデッド システムでは動作環境が厳密に制御され、アプリケーション ソフトウェアのセットが詳細に定義されています。この環境では、次の方法で仮想メモリ管理が各エンベデッド システム用に最適化されます。

- ページ変換テーブルは、ページ変換エントリの検索効率が上がるように構成することができます。ほとんどの汎用プロセッサでは、インデックス付きの表 (単純な検索方法、表サイズ大) またはハッシュ テーブル (複雑な検索方法、表サイズ小) のいずれかがインプリメントされます。ソフトウェアでの検索では、エンベデッド システムに適した構成を組み合わせる使用できます。ページ テーブルのサイズとアクセス時間はどちらも最適化可能です。
- アプリケーション モジュール、デバイス ドライバ、システム サービス ルーチン、およびデータに個別のページ サイズを使用できます。個別のページ サイズを使用するとメモリのフラグメンテーションが削減され、システム ソフトウェアでメモリが効率的に使用されます。サイズの大きなデータ構造を 16MB のページに、小型の I/O デバイス ドライバを 1KB のページに割り当てるとこのような例が考えられます。
- ページの置き換えは、ページ変換のミスを最小限に抑えるように調整できます。次に説明するように、最も頻繁に使用されるページ変換は、TLB (Translation Look-Aside Buffer) に格納されます。どの変換を TLB に格納し、新しい変換が必要になった場合にどの変換を置き換えるかは、ソフトウェアで決定されます。置き換え手法は、ページ変換エントリが常に TLB に格納されたり削除されたりするスラッシュが発生しないように調整できます。また、重要なページ変換が置き換えられないように (ページ ロック) 調整できます。

ソフトウェアで管理される統合 64 エントリ TLB は、MMU でアクセスされる命令およびデータ ページ変換エントリのサブセットをキャッシュします。ソフトウェアでシステム メモリにあるページ変換テーブルからエントリを読み出して TLB に格納します。次に、統合 TLB を詳細に説明します。内部的には、MMU には命令およびデータ (サイズはそれぞれ C\_MMU\_ITLB\_SIZE および C\_MMU\_DTLB\_SIZE で設定可) のシャドウ TLB も含まれます。

これらのシャドウ TLB はすべて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 (TLB)

TLB (Translation Look-Aside Buffer) は MicroBlaze MMU で、アドレス変換 (プロセッサが仮想モードで実行している場合)、メモリ保護、および格納制御に使用されます。TLB の各エントリには、仮想ページ (PID と 有効ページ番号) の識別、物理ページへの変換の指定、ページの保護特性の決定、およびページに関連付けられた格納属性の指定のために必要な情報が含まれています。

MicroBlaze の TLB は、次の 3 つの独立した TLB として物理的にインプリメントされます。

- **UTLB (統合 TLB)**: 64 個のエントリを含み、擬似連想です。命令ページおよびデータ ページ変換は、任意の UTLB エントリに格納できます。UTLB の初期化および管理はすべてソフトウェアで制御されます。
- **ITLB (命令シャドウ TLB)**: 命令ページ変換エントリを含み、完全に連想です。ITLB に格納されたページ変換エントリは、UTLB で最後にアクセスされた命令ページ変換です。ITLB は命令変換と UTLB のアップデート操作の競合を最小限に抑えるために使用されます。ITLB の初期化および管理はすべてハードウェアで制御され、ソフトウェアには透過的です。
- **DTLB (データ シャドウ TLB)**: データ ページ変換エントリを含み、完全に連想です。DTLB に格納されたページ変換エントリは、UTLB で最後に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 のアップデート操作の競合を最小限に抑えるために使用されます。DTLB の初期化および管理はすべてハードウェアで制御され、ソフトウェアには透過的です。

図 1-17 に TLB の変換フローを示します。

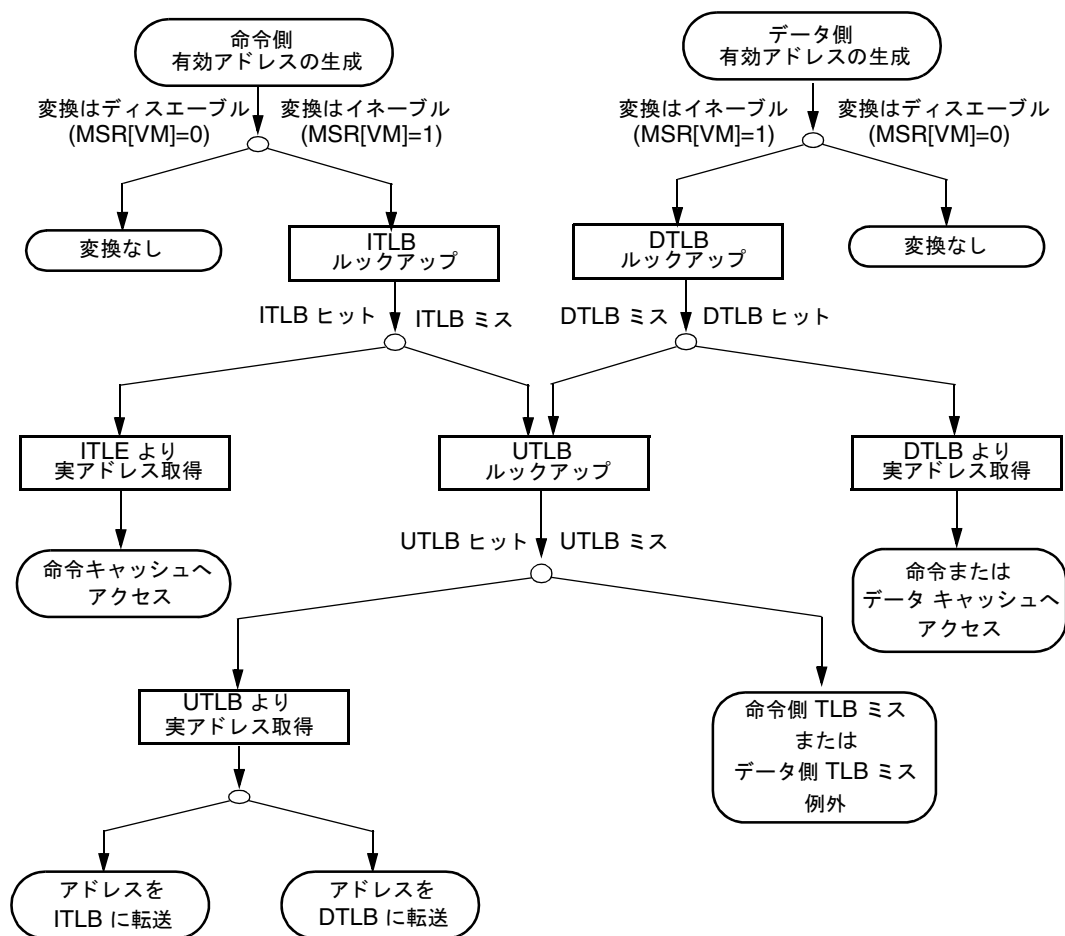
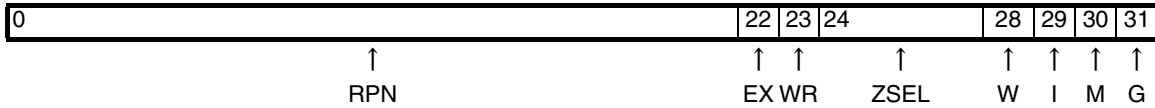


図 1-17 : TLB アドレス変換フロー

## TLB エントリのフォーマット

図 1-18 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 ビットで、TLBLO (データ エントリ) と TLBHI (タグ エントリ) の 2 つの部分で構成されます。

TLBLO :



TLBHI :

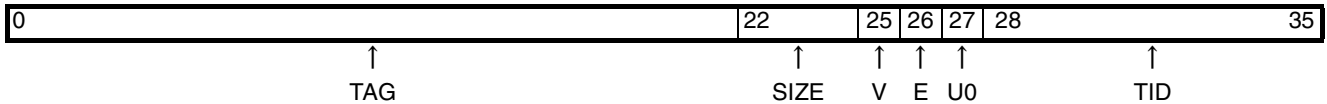


図 1-18 : TLB エントリのフォーマット

TLB エントリの内容は、39 ページの表 1-18 および 42 ページの表 1-19 に示されています。

TLB エントリ内のフィールドは次のように分類されます。

- 仮想ページ識別 (TAG、SIZE、V、TID) : ページ変換エントリを識別します。変換処理中に仮想ページ番号と比較されます。
- 物理ページ識別 (RPN、SIZE) : 物理メモリ内にある変換されたページを識別します。
- アクセス制御 (EX、WR、ZSEL) : ページで許可されているアクセスのタイプを指定し、不正アクセスからページを保護するために使用します。
- 格納属性 (W、I、M、G、E、U0) : データ キャッシュのキャッシュ ポリシー (ライトバックやライトスルー) に従っているかどうか、ページがキャッシュ可能かどうか、バイト順 (エンディアン) などの格納制御属性を指定します。

表 1-34 に、TLB エントリの SIZE フィールドと変換されたページのサイズの関係を示します。タグ比較で使用するアドレス ビットと物理アドレスに使用される物理ページ番号のビットもページサイズ別に示されています。

表 1-34 : ページ サイズによるページ変換ビット範囲

ページ サイズ	TLBHI の SIZE フィールド	TAG 比較のビット範囲	ページ オフセット	物理ページ番号	0にする RPN ビット
1KB	000	TAG[0:21]、アドレス[0:21]	アドレス[22:31]	RPN[0:21]	-
4KB	001	TAG[0:19]、アドレス[0:19]	アドレス[20:31]	RPN[0:19]	20:21
16KB	010	TAG[0:17]、アドレス[0:17]	アドレス[18:31]	RPN[0:17]	18:21
64KB	011	TAG[0:15]、アドレス[0:15]	アドレス[16:31]	RPN[0:15]	16:21
256KB	100	TAG[0:13]、アドレス[0:13]	アドレス[14:31]	RPN[0:13]	14:21
1MB	101	TAG[0:11]、アドレス[0:11]	アドレス[12:31]	RPN[0:11]	12:21
4MB	110	TAG[0:9]、アドレス[0:9]	アドレス[10:31]	RPN[0:9]	10:21
16MB	111	TAG[0:7]、アドレス[0:7]	アドレス[8:31]	RPN[0:7]	8:21

## TLB アクセス

MMU で仮想アドレス (PID と有効アドレスの組み合わせ) が物理アドレスに変換される場合、最初に、適切なページ変換エントリのシャドウ TLB があるかが確認されます。エントリが検出された場合は、物理メモリのアクセスに使用されます。検出されない場合は、UTLB にエントリがあるかが MMU で確認されます。シャドウ TLB ミスのために UTLB にアクセスすると、その都度遅延が生じます。ミスのレイテンシは、2 ~ 32 周期です。DTLB と ITLB が同時に UTLB にアクセスした場合は、DTLB が優先します。

63 ページの図 1-19 には、シャドウ TLB のいずれかまたは UTLB にページ変換エントリが存在するかを確認するために MMU で実行される論理プロセスが示されています。TLB の有効なエントリはすべてチェックされます。

TLB エントリで次の条件がすべて満たされると、TLB ヒットが発生します。

- エントリが有効である
- エントリの TAG フィールドが、エントリの SIZE フィールドで制御される有効アドレスの EPN に一致する
- エントリの TID が PID に一致する

上記のどれか 1 つでも満たされない場合は、TLB ミスが発生します。TLB ミスが発生すると、次の例外が発生します。

TID 値が 0x00 の場合、TID と PID の比較が MMU で無視され、TAG と EA[EPN] のみが比較されます。TLB エントリの TID 値が 0x00 の場合は変換がプロセスから独立しているため、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この TID 値を割り当てます。PID 値が 0x00 の場合は、任意のページにアクセスできるプロセスが識別されず、ページ変換ヒットは TID=0x00 の場合にのみ発生します。ソフトウェアでは、EA[EPN] と PID の組み合わせが一致する TLB のエントリを複数読み込むことができますが、これはプログラムのエラーと解釈され、定義されないビヘイビアを引き起こします。

ヒットが発生すると MMU で、対応する TLB エントリから RPN フィールドが読み込まれます。フィールドのビットは、SIZE フィールドの値に応じて、一部または全部が使用されます (表 1-34 を参照)。SIZE フィールドでページ サイズが 256KB に指定されている場合は、RPN[0:13] は物理ページ番号を表し、物理アドレスを構成します。RPN[14:21] は使用されず、TLB エントリの初期化の際に、ソフトウェアで 0 に設定される必要があります。物理アドレスの残りのビットには、EA のページ オフセット部を使用します。ページ サイズが 256KB の場合、RPN[0:13] を有効アドレスの 14 ~ 31 ビットと結合し、32 ビット物理アドレスを構成します。

物理メモリにアクセスする前に、MMU で TLB エントリのアクセス制御フィールドが確認されます。これらのフィールドは、要求されたメモリのアクセスが実行中のプログラムに許可されているかどうかを示します。

アクセスが許可されていれば、ページへのアクセス方法を決定するために、MMU で格納属性フィールドがチェックされます。格納属性フィールドでは、メモリ アクセスのキャッシュ方法が指定されています。

## TLB アクセス エラー

TLB アクセス エラーが発生すると、例外が発生します。この場合、エラーの原因となる命令に割り込みが発生し、エラーを解決するために制御が割り込みハンドラに渡されます。TLB アクセスのエラーの原因には次の 2 つがあります。

- 一致する TLB エントリが検出されず、TLB ミスが発生した場合
- 一致する TLB エントリは検出されたが、ページへのアクセスが格納属性かゾーン保護のいずれかで禁止されている場合

割り込みが発生すると、MSR[VM] が 0 になり、プロセッサは実モードになります。実モードでは、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。UTLB は、システム ソフトウェアによりページ変換エントリで初期化されると、通常は、実モードで実行している割り込みハンドラを使用して管理されます。

図 1-19 に、TLB エントリの確認の一般的な手順を示します。

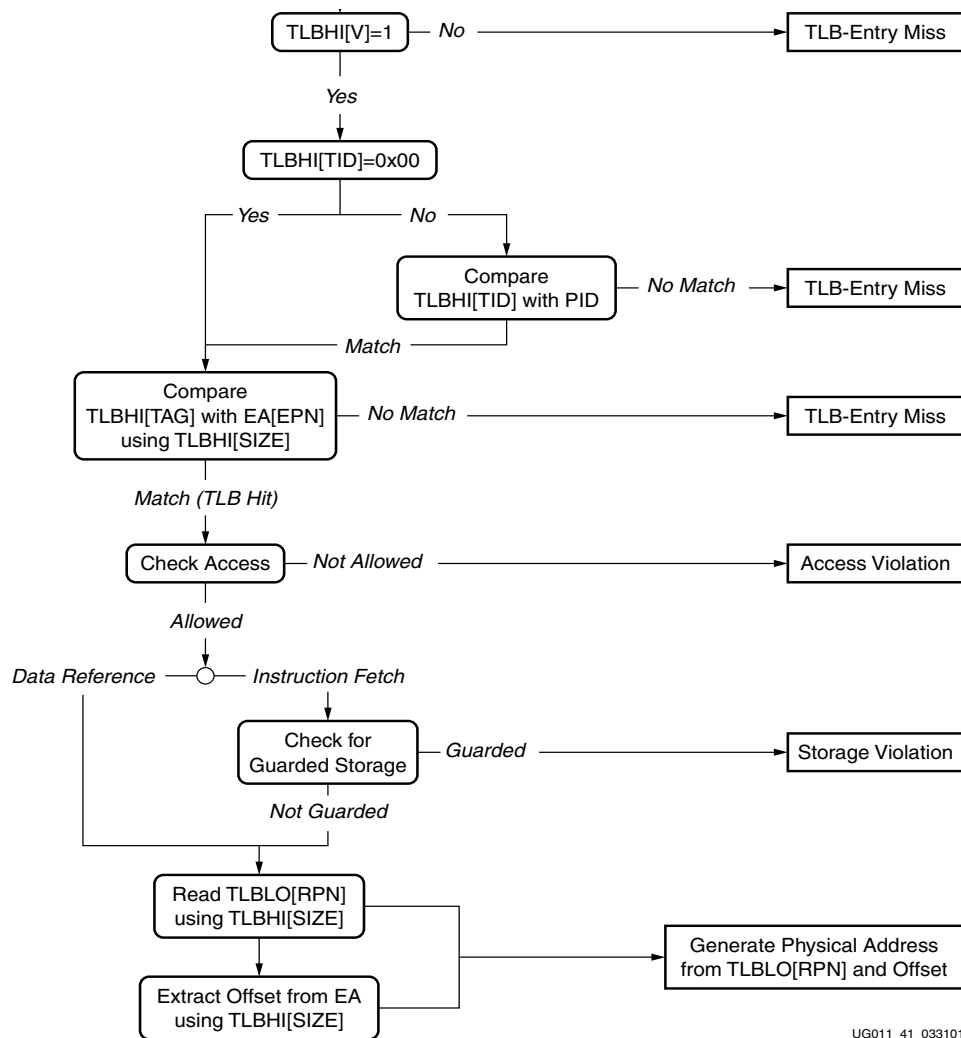


図 1-19 : TLB エントリ確認の一般的な手順

次に、TLB アクセス エラーのために例外が発生する条件を示します。

#### データ格納例外

仮想モードがイネーブル ( $\text{MSR}[\text{VM}]=1$ ) で、ページへのアクセスが次の理由のいずれかで許可されない場合、データ格納例外が発生します。

- ユーザー モードの場合
  - ◆ TLB エントリで、ページへのアクセスを禁じるゾーン フィールド ( $\text{ZPR}[\text{Zn}]=00$ ) が指定されている。これは読み込み命令および格納命令の場合です。
  - ◆ TLB エントリで、読み込み専用のページ ( $\text{TLBLO}[\text{WR}]=0$ ) が指定されている。読み込み専用でない場合は、ゾーン フィールド ( $\text{ZPR}[\text{Zn}], 11$ ) では無効にされません。これは格納命令の場合です。
- 特権モードの場合
  - ◆ TLB エントリで、読み込み専用のページ ( $\text{TLBLO}[\text{WR}]=0$ ) が指定されている。読み込み専用でない場合は、ゾーン フィールド ( $\text{ZPR}[\text{Zn}], 10$  および  $\text{ZPR}[\text{Zn}], 11$ ) では無効にされません。これは格納命令の場合です。

#### 命令格納例外

仮想モードがイネーブル ( $\text{MSR}[\text{VM}]=1$ ) で、ページへのアクセスが次の理由のいずれかで許可されない場合、命令格納例外が発生します。

- ユーザー モードの場合
  - ◆ TLB エントリで、ページへのアクセスを禁じるゾーン フィールド ( $\text{ZPR}[\text{Zn}]=00$ ) が指定されている。
  - ◆ TLB エントリで、実行不可のページ ( $\text{TLBLO}[\text{EX}]=0$ ) が指定されている。実行不可でない場合は、ゾーン フィールド ( $\text{ZPR}[\text{Zn}], 11$ ) では無効にされません。
  - ◆ TLB エントリで、保護されたページ ( $\text{TLBLO}[\text{G}]=1$ ) が指定されている。
- 特権モードの場合
  - ◆ TLB エントリで、実行不可のページ ( $\text{TLBLO}[\text{EX}]=0$ ) が指定されている。実行不可でない場合は、ゾーン フィールド ( $\text{ZPR}[\text{Zn}], 10$  および  $\text{ZPR}[\text{Zn}], 11$ ) では無効にされません。
  - ◆ TLB エントリで、保護されたページ ( $\text{TLBLO}[\text{G}]=1$ ) 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 ( $\text{MSR}[\text{VM}]=1$ ) で、一致する有効な TLB エントリが TLB (シャドウおよび UTLB) に存在しない場合、読み込み命令または格納命令でデータ TLB ミス例外が発生します。

#### 命令 TLB ミス例外

仮想モードがイネーブル ( $\text{MSR}[\text{VM}]=1$ ) で、一致する有効な TLB エントリが TLB (シャドウおよび UTLB) に存在しない場合、命令フェッチで命令 TLB ミス例外が発生します。



## アクセス保護

システム ソフトウェアではアクセス保護を使用して、不正アクセスからメモリ ロケーションを保護します。ユーザー モードと特権モードの両方で、メモリへの読み出し、書き込み、命令フェッチのアクセスが制限されます。アクセス保護は、仮想保護モードがイネーブルの場合に使用できます。

仮想ページの TLB エントリでは、ページで許可されているアクセスの種類が指定されます。また TLB エントリでは、TLB エントリで指定されたアクセス制御を無効にするゾーン保護レジスタのゾーン保護フィールドも指定されます。

### TLB によるアクセス保護の制御

各 TLB エントリでは、次の 3 つのアクセスが制御されます。

- プロセス：各プロセスには重複しないプロセス ID (PID) が割り当てられており、これを不正アクセスからの保護に使用します。システム ソフトウェアでユーザー モードのアプリケーションが起動する際に、アプリケーションの PID が PID レジスタに読み込まれます。アプリケーションの実行時に、TLBHI (Translation Look-Aside Buffer High) の TID フィールドが PID と一致する TLB エントリのみを使用してメモリ アドレスが変換されるため、仮想メモリの特定の領域へのアプリケーションのアクセスが、システム ソフトウェアにより制限されます。

TLB エントリの TID 値が 0x00 の場合は変換がプロセスから独立しているため、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この TID 値を割り当てます。

- 実行：プロセッサでは、実行可能 (TLBLO[EX]=1) である仮想ページからフェッチされた命令のみが実行されます。TLBLO[EX] = 0 の場合命令がページからフェッチされず、命令格納割り込み (ISI) が発生します。ISI は命令のフェッチ時ではなく、実行時に発生します。論理的にフェッチされた命令は実行されずに破棄されたため、ISI は発生しません。

ゾーン保護レジスタを使用して、実行保護を無効にできます。

- 読み出し/書き込み：データは、書き込み可 (TLBLO[WR]=1) の仮想ページにのみ書き込まれます。TLBLO[WR] = 0 の場合ページは読み込み専用で、書き込みを試みるとデータ格納割り込み (DSI) が発生します。

ゾーン保護レジスタを使用して、書き込み保護を無効にできます。

TLB エントリを使用してプログラムでのページの読み込みを禁止することはできません。仮想モードでは、ゾーン保護はページの読み出し保護に使用されます。アクセス不可ゾーンを ZPR[Zn] = 00 に定義すると、TLB エントリ アクセス保護が無効になります。プログラムでのページ読み出しは、ユーザー モードで実行している場合にのみ禁止できます。特権モードで実行しているプログラムには、常にページの読み出しアクセス権があります。

### ゾーン保護

ゾーン保護は、TLB エントリで指定されたアクセス保護を無効にするために使用されます。ゾーンは、アクセス保護設定が同じ仮想ページのグループです。ゾーンには、任意のページサイズの組み合わせで、任意のページ数を含めることができます。隣接したページを含める要件はありません。

ゾーン保護レジスタ (ZPR) は 32 ビットで、最大 16 個のゾーンに対して個別に、無効にする保護の種類を指定します。ZPR では、ゾーンの無効指定は 2 ビットのフィールドとしてエンコードされます。TLB エントリの 4 ビットのゾーン選択フィールド (TLBLO[ZSEL]) では、16 個の ZPR (Z0 ~ Z15) ゾーンフィールドから 1 つを指定します。ZSEL = 0101 と指定すると、ゾーン Z5 が選択されます。

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変更されます。ZPR を使用しない場合は、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。

## UTLB 管理

UTLB はプロセッサの MMU とメモリ管理ソフトウェア間のインターフェイスの役割を果たします。UTLB はシステム ソフトウェアで管理され、仮想アドレスを物理アドレスに変換する方法を MMU に指示します。変換エラーやアクセス違反が原因で問題が発生すると、例外のメカニズムを使用して問題が MMU からシステム ソフトウェアに送信されます。MMU でメモリ変換を続行できるように、システム ソフトウェアから問題を解決するための割り込みハンドラが提供されます。

ソフトウェアでは UTLB エントリを、MFS 命令を使用して読み出し、MTS 命令を使用して書き込みます。これらの命令では、UTLB の 64 個のエントリの 1 つに対応する TLBX レジスタ インデックス (0 ~ 63) が使用されます。タグ部分とデータ部分は個別に読み出しおよび書き込みされるため、エントリに完全にアクセスするには、ソフトウェアで MFS 命令または MTS 命令を 2 回実行する必要があります。UTLB での変換の検索には、TLBSX レジスタが使用されます。TLBSX は有効アドレスを使用して変換を検出し、対応する UTLB インデックスを TLBX レジスタに読み込みます。

UTLB エントリは、MTS 命令で TLB エントリのタグ部分の有効ビット (TLBHI[V]) が 0 に設定され、個別に無効にされます。

## ページ アクセスおよびページ変更の記録

ソフトウェアでの仮想メモリの管理では、次の点を考慮する必要があります。

- 仮想メモリ環境では、物理的に使用可能なメモリよりも多くのメモリがソフトウェアとデータで消費されます。ソフトウェアおよびデータ ページの一部は、使用されないときにはハード ドライブなど物理メモリ領域の外側に格納される必要があります。頻繁に使用されるページが物理メモリに格納され、あまり使用されないページはほかの場所に格納されるのが理想的です。
- 物理メモリに格納されているページを新しいページを格納するために置き換える場合、変更されているかどうかを確認する必要があります。変更されている場合は、新しいページを読み込む前に保存する必要があります。変更されていない場合は、保存する必要はありません。
- UTLB に保管できるページ変換の数には制限があります。保管されなかった変換は、ページ変換テーブルに格納します。UTLB に保管されていない変換を読み込む場合、破棄する UTLB エントリをシステム ソフトウェアで決定します。システム ソフトウェアでの使用頻度が低い変換を置き換えます。

これらを効率よく処理するには、ページ アクセスおよびページ変更の履歴を把握する必要があります。MicroBlaze では、ハードウェアのページ アクセスおよびページ変更の履歴は記録されませんが、システム ソフトウェアで TLB ミス例外およびデータ格納例外を使用して、この情報を収集できます。情報が収集されると、ページ変換テーブルに対応付けられたデータ構造に格納されます。

ページ アクセス情報は、物理メモリ領域が必要な場合に、どのページを物理メモリに保持し、どのページを置き換えるかを決定するために使用されます。システム ソフトウェアでは、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。このビットは、ページ変換がアクセスされていないことを示すために、無効 (TLBHI[V]=0) に初期化する必要があります。最初にページにアクセスしたときに、UTLB エントリが無効であるか、ページ変換が UTLB に存在しないかのいずれかの理由で TLB ミス例外が発生し、TLB ミス ハンドラで UTLB がアップデートされて変換が有効 (TLBHI[V]=1) になります。有効に設定されたビットは、ページおよびページの変換がアクセスされたという記録になります。TLB ミス ハンドラでも、ページ変換エントリに対応付けられた別のデータ構造に情報を記録できます。

ページ変更情報は、現在のページを新しいページで上書きしてもよいか、またはハード ディスクに格納する必要があるかを示すために使用されます。システム ソフトウェアでは、TLB エントリの書き込み保護ビット (TLBLO[WR]) を使用して、ページの変更を監視できます。このビットは、ページ

変換が変更されていないことを示すために、読み込み専用 (TLBLO[WR]=0) に初期化する必要があります。ページへのデータの書き込みを最初に実行したときに、上記のようにページが既にアクセスされており有効と設定されている場合は、データ格納例外が発生します。ソフトウェアにページへの書き込みが許可されている場合は、データ格納ハンドラでページが書き込み可 (TLBLO[WR]=1) に設定され、戻ります。書き込み保護設定ビットは、ページが変更されたという記録になります。データ格納ハンドラでも、ページ変換エントリに対応付けられた別のデータ構造にこの情報を記録できます。

ページ変更の追跡は、最初に仮想モードになった場合および新しいプロセスの開始時に有用です。

## リセット、割り込み、例外、ブレーク

MicroBlaze では、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がサポートされます。次のセクションでは、各イベントの実行フローを説明します。

これらのイベントの相対的な優先順位は、次のとおりです。

1. リセット
2. ハードウェア例外
3. マスク不可ブレーク
4. ブレーク
5. 割り込み
6. ユーザー ベクタ (例外)

表 1-35 に、ベクタのメモリ アドレス ロケーションとハードウェアで強制される戻りアドレスのレジスタ ファイル ロケーションを示します。各ベクタは、アドレス範囲全体の分岐を可能にするため、2 つのアドレスを割り当てます (IMM 命令の後に BRAI 命令が必要)。アドレス範囲 0x28 ~ 0x4F は、ザイリンクスが今後のソフトウェア サポート用に予約しています。ユーザー アプリケーションをこの範囲内のアドレスに割り当てると、今後リリースされる EDK のサポート ソフトウェアと競合する可能性があります。

表 1-35 : ベクタと戻りアドレスのレジスタ ファイル ロケーション

イベント	ベクタ アドレス	レジスタ ファイル 戻りアドレス
リセット	0x00000000 ~ 0x00000004	-
ユーザー ベクタ (例外)	0x00000008 ~ 0x0000000C	Rx
割り込み	0x00000010 ~ 0x00000014	R14
ブレーク : マスク不可ハードウェア	0x00000018 ~ 0x0000001C	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	0x00000020 ~ 0x00000024	R17 または BTR
将来のソフトウェア サポート用に ザイリンクスが予約	0x00000028 ~ 0x0000004F	-

LDX および STX 命令と共に使用すると、セマフォやスピンロックなどの相互排除機能をインプリメントするために、これらのイベントはすべて予約ビットをクリアにします。

## リセット

Reset または Debug\_Rst<sup>(1)</sup> がアクティブになると、パイプラインが空にされ、リセット ベクタ (アドレス 0x0) から命令がフェッチされます。これら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。

### 擬似コード

```
PC ← 0x00000000
MSR ← C_RESET_MSR (第 2 章の「MicroBlaze コアのコンフィギュレーション」を参照)
EAR ← 0; ESR ← 0; FSR ← 0
PID ← 0; ZPR ← 0; TLBX ← 0
Reservation ← 0
```

## ハードウェア例外

MicroBlaze は、不正な命令、命令およびデータ バスのエラー、不整列アクセスといった内部エラー状態を検出するようにコンフィギュレーションできます。除算の例外は、ハードウェア除算器を使用する場合 (C\_USE\_DIV=1) 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を使用する場合 (C\_USE\_FPU>0) は、浮動小数点に関する例外であるアンダーフロー、オーバーフロー、0 での浮動小数点除算、不正な処理、および非正規化オペランド エラーも検出できます。

ハードウェア MMU を使用する場合は、メモリ管理に関する例外である不正命令例外、データ格納例外、命令格納例外、データ TLB ミス例外、および命令 TLB ミス例外も検出できます。

ハードウェア例外が発生すると、パイプラインが空にされ、ハードウェア例外ベクタ (アドレス 0x20) に分岐します。実行段の命令は実行されません。

例外が発生すると、汎用レジスタ R17 が次のようにアップデートされます。

- MMU 例外 (データ格納例外、命令格納例外、データ TLB ミス例外、命令 TLB ミス例外) では、戻り時に例外が発生した命令を再実行するため、レジスタ R17 に適切なプログラムカウンタ値が読み込まれます。前に IMM 命令がある場合は、この値は IMM 命令に戻るよう調整されます。例外が分岐遅延スロットの命令で発生した場合は、この値は分岐命令に戻るよう調整されます。前に IMM 命令がある場合は、この調整も実行されます。
- その他の例外では、例外が分岐遅延スロットの命令で発生した場合を除き、後続の命令のプログラムカウンタ値がレジスタ R17 に読み込まれます。例外が分岐遅延スロットの命令で発生した場合は、ESR[DS] ビットが設定され、例外ハンドラにより BTR に格納されている分岐ターゲット アドレスから実行が再開されます。

RTED 命令を実行すると、MSR の EE ビットと EIP ビットが自動的に反転されます。

RTED 命令、RTBD 命令、および RTID 命令を実行すると、MSR の VM ビットと UM ビットが自動的に反転されます。

---

1. XMD デバッガにより MDM を介して制御されるリセット入力

## 例外の優先順位

複数の例外が同時に発生する場合、次の優先順位で処理されます。

- 命令バス例外
- 命令 TLB ミス例外
- 命令格納例外
- 不正の op コードの例外
- 特権命令による例外
- データ TLB ミス例外
- データ格納例外
- 不整列データ アクセス
- データ バス例外
- 除算例外
- FPU 例外
- 高速シンプレックス リンクの例外

## 例外の原因

- 高速シンプレックス リンク (FSL) による例外  
制御ビットの不一致がある場合に e ビットを 1 に設定して get 命令または getd 命令を実行すると発生します。
- 命令バスによる例外  
命令側プロセッサ ローカル バス (PLB) の例外は、スレーブのアクティブ エラー信号 (IPLB\_MrdErr) またはアービタのタイムアウト信号 (IPLB\_MTimeout) により発生します。命令側オンチップ ペリフェラル バス (IOPB) の例外は、スレーブのアクティブ エラー信号 (IOPB\_errAck) またはアービタのタイムアウト信号 (IOPB\_timeout) により発生します。命令側のローカル メモリ (ILMB) および CacheLink (IXCL) インターフェイスは、命令バスの例外の原因にはなりません。
- 不正な op コードによる例外  
命令で主要 op コード (命令ビット 0 ～ 5) が無効のときに発生します。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されていない場合は、不正と検出されます。オプションの C\_OPCODE\_0x0\_ILLEGAL がイネーブルの場合は、命令が 0x00000000 に等しいと発生します。
- データ バスによる例外  
データ側プロセッサ ローカル バス (PLB) の例外は、スレーブのアクティブ エラー信号 (DPLB\_MrdErr または DPLB\_MWrErr) またはアービタのタイムアウト信号 (DPLB\_MTimeout) により発生します。データ側オンチップ ペリフェラル バス (DOPB) の例外は、スレーブのアクティブ エラー信号 (DOPB\_errAck) またはアービタのタイムアウト信号 (DOPB\_timeout) により発生します。データ側のローカル メモリ (DLMB) および CacheLink (DXCL) インターフェイスは、データ バスの例外の原因にはなりません。
- 不整列データ アクセスによる例外  
データ バスへのアドレスが 30 または 31 ビットのワード アクセスまたは 31 ビットのハーフワード アクセスにより発生します。

- 除算例外

整数を 0 で除算する (`idiv` または `idivu`) 場合に例外発生するか、または符号付き整数で除算した結果 (`idiv`) オーバーフローが発生する場合に例外が発生します。

- FPU での例外

浮動小数点命令で発生するアンダーフロー、オーバーフロー、0 での除算、不正な処理、および非正規化オペランド エラーにより発生します。

- ◆ アンダーフローは、結果が非正規化数の場合に発生します。
- ◆ オーバーフローは、結果が非数 (NaN) の場合に発生します。
- ◆ 0 での除算による FPU の例外は、`rB` が無限大ではないときに `fdiv` の `rA` オペランドが 0 であると発生します。
- ◆ 不正な処理は、NaN オペランドの送信または不正な無限大または 0 オペランドの組み合わせにより発生します。

- 特権命令による例外

特権命令をユーザー モードで実行すると発生します。

- データ格納による例外

メモリのデータにアクセスするとメモリ保護違反となる場合に発生します。

- 命令格納による例外

メモリの命令にアクセスするとメモリ保護違反となる場合に発生します。

- データ TLB ミスによる例外

有効な TLB エントリが存在せず仮想保護モードがイネーブルの場合に、メモリのデータにアクセスすると発生します。

- 命令 TLB ミスによる例外

有効な TLB エントリが存在せず仮想保護モードがイネーブルの場合に、メモリの命令にアクセスすると発生します。

## 擬似コード

```

ESR[DS] ← 遅延スロットの例外
if ESR[DS] then
    BTR ← ターゲット PC の分岐
    if MMU exception then
        if branch preceeded by IMM then
            r17 ← PC - 8
        else
            r17 ← PC - 4
    else
        r17 ← 無効な値
    else if MMU exception then
        if instruction preceeded by IMM then
            r17 ← PC - 4
        else
            r17 ← PC
    else
        r17 ← PC + 4
PC ← 0x00000020
MSR[EE] ← 0, MSR[EIP] ← 1
MSR[UMS] ← MSR[UM], MSR[UM] ← 0, MSR[VMS] ← MSR[VM], MSR[VM] ← 0

```



```

ESR[EC] ← 例外値
ESR[ESS] ← 例外値
EAR ← 例外値
FSR ← 例外値
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、外部ブレーク信号 (**Ext\_BRK** および **Ext\_NM\_BRK** 入力ポート) をアサートすると実行されます。ブレークが発生すると、実行段の命令は完了され、デコード段の命令はブレーク ベクタ (アドレス **0x18**) への分岐に置き換えられます。ブレーク戻りアドレス (ブレーク発生時にデコード段にあった命令に関連付けられている **PC**) は、汎用レジスタ **R16** に読み込まれます。また、マシン ステータス レジスタ (**MSR**) で **BIP** (ブレーク実行中) フラグが設定されます。

通常のハードウェア 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 (実行中のブレークまたは例外がない) 場合にのみ処理されます。**BIP** フラグがアクティブになると、割り込みがディセーブルになります。マスク不可ブレーク (**Ext\_NM\_BRK** 入力ポート) は即座に処理されます。

**RTBD** 命令が実行されると、**MSR** の **BIP** ビットがクリアされます。

**Ext\_BRK** 信号は、ブレークが開始するまではアサートされたままで、**RTBD** 命令が実行される前にディアサートされる必要があります。**Ext\_NM\_BRK** 信号は、1 クロック サイクルだけアサートされる必要があります。

### 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 4 章「**MicroBlaze 命令セット アーキテクチャ**」を参照してください。

### レイテンシ

ブレークが発生してからブレークのサービス ルーチンが開始されるまでにかかる時間は、その時点で実行段にある命令およびブレーク ベクタを格納しているメモリのレイテンシによって異なります。

### 擬似コード

```

r16 ← PC
PC ← 0x00000018
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 (割り込み入力ポートに接続) が 1 つサポートされています。割り込みは、マシン ステータス レジスタ (**MSR**) の割り込みイネーブルビット (**IE**) が **1** に設定されている場合にのみ処理されます。割り込みが発生すると、実行段の命令は完了され、デコード段の命令は割り込みベクタ (アドレス **0x10**) への分岐に置き換えられます。割り込み戻りアドレス (割

り込み発生時にデコード段にあった命令に関連付けられている PC) は、自動的に汎用レジスタ R14 に読み込まれます。また、MSR の IE ビットが 0 になり、これ以降の割り込みはディスエーブルになります。RTID 命令が実行されると、IE ビットが再び 1 に設定されます。

MSR の BIP (ブレイク実行中) ビットまたは EIP (例外実行中) ビットのいずれかが 1 に設定されている場合は、割り込みは無視されます。

パラメータ C\_INTERRUPT\_IS\_EDGE を使用し、外部割り込みをレベルまたはエッジで設定することができます。

- レベルでの割り込みを使用する場合、MicroBlaze 割り込みを取り込んで割り込みベクタにジャンプするまで、割り込み入力にセットされたままである必要があります。割り込みハンドラから返される前に、ソフトウェアは割り込みをクリアにする必要があります。クリアになっていなければ、割り込みハンドラから返されたに割り込みがイネーブルになり次第、割り込みは再度取り込まれます
- エッジでの割り込みを使用する場合、MicroBlaze で割り込み入力エッジが検出されラッチされます。つまり、この入力は 1 クロック サイクルでアサートされる必要があります。割り込み入力はアサートされたままの状態で構いませんが、新たな割り込みが検出される前に最低 1 クロック サイクル間はディアサートされている必要があります。

## レイテンシ

割り込みが発生してから ISR (割り込みサービス ルーチン) が開始されるまでにかかる時間は、プロセッサのコンフィギュレーションおよび割り込みベクタを格納しているメモリ コントローラのレイテンシによって異なります。MicroBlaze にハードウェア除算器が含まれる場合、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。

## 擬似コード

```
r14 ← PC
PC ← 0x00000010
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 (例外)

ユーザー例外ベクタは、アドレス 0x8 に配置されています。ソフトウェア フローに BRALID Rx, 0x8 命令を挿入すると、ユーザー例外が発生します。Rx にはどの汎用レジスタでも指定できますが、ザイリンクスでは R15 を使用してユーザー例外の戻りアドレスを格納し、RTSD 命令を使用してユーザー例外ハンドラから戻ることを推奨します。

## 擬似コード

```
rx ← PC
PC ← 0x00000008
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

MicroBlaze をオプションの命令キャッシュと共に使用して、LMB の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向上させることができます。

命令キャッシュには、次の特徴があります。



- ダイレクト マップ方式 (連想度 1)
- ユーザーがキャッシュ可能メモリのアドレス範囲を指定可能
- キャッシュ サイズおよびタグ サイズを設定可能
- CacheLink (XCL) インターフェイスを使用したキャッシュ
- 4 または 8 ワードのキャッシュ ライン
- MSR ビットを使用したキャッシュのオン/オフ制御
- 命令キャッシュ ラインを無効にする WIC 命令 (オプション)

## 命令キャッシュの機能

命令キャッシュの使用時には、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の 2 つに分割されます。キャッシュ可能なセグメントは、C\_ICACHE\_BASEADDR および C\_ICACHE\_HIGHADDR の 2 つのパラメータで定義されます。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ています。その他のアドレスは、キャッシュ不可能なアドレスです。

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)。C\_ICACHE\_BASEADDR および C\_ICACHE\_HIGHADDR で指定する範囲は完全な 2 のべき乗の範囲にし、C\_ICACHE\_BASEADDR の最下位ビットは 0 にする必要があります。

キャッシュ可能な命令アドレスは、キャッシュ アドレスおよびタグ アドレスから構成されます。MicroBlaze の命令キャッシュは 64B ~ 64KB の範囲で設定でき、これは 6 ~ 16 ビットのキャッシュ アドレスに対応しています。キャッシュ アドレスとタグ アドレスを合わせたものが、キャッシュ可能なメモリの完全アドレスに一致するように設定する必要があります。キャッシュ サイズを 2KB 未満に指定すると、タグ RAM および命令 RAM のインプリメントには分散 RAM が使用されます。

たとえば、C\_ICACHE\_BASEADDR=0x00300000、C\_ICACHE\_HIGHADDR=0x0030ffff、C\_ICACHE\_BYTE\_SIZE=4096、C\_ICACHE\_LEN=8 のように設定されている場合、キャッシュ可能な 64KB のメモリでは 16 ビットのバイト アドレスが使用され、4KB のキャッシュでは 12 ビットのバイト アドレスが使用されるため、必要なアドレス タグは  $16 - 12 = 4$  ビットです。この場合、1024 ワードの命令の格納に RAMB16 が 2 個、128 個のキャッシュ ライン (4 ビットのタグ + 8 ワード有効ビット + 1 ライン有効ビット) エントリに RAMB16 が 1 個必要となるので、合計で 3 個の RAMB16 プリミティブが必要です。

74 ページの図 1-20 に、命令キャッシュの構成を示します。

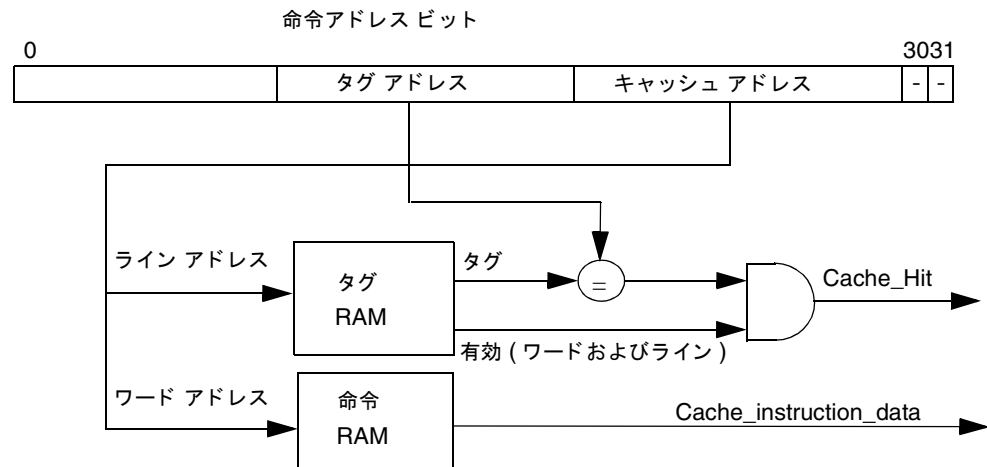


図 1-20 : 命令キャッシュの構成

## 命令キャッシュの動作

命令がフェッチされるごとに、命令アドレスがキャッシュ可能なセグメントに属するものであるかが検出されます。アドレスがキャッシュ不可能な場合は、キャッシュコントローラでは命令が無視され、PLB、OPB、または LMB で命令が実行されます。アドレスがキャッシュ可能な場合は、タグメモリでルックアップが実行され、要求されたアドレスがキャッシュに含まれるかどうかを確認されます。ワードおよびラインの有効ビットが設定されており、タグアドレスが命令アドレスのタグアドレスセグメントと一致すれば、ルックアップが成功したことになります。キャッシュミスの場合、キャッシュコントローラは命令側の CacheLink (IXCL) に新しい命令を要求し、メモリコントローラから戻される関連キャッシュラインを待ちます。

## 命令キャッシュのソフトウェア サポート

### MSR ビット

MSR の ICE ビットは、キャッシュのイネーブル、ディスエーブルを制御します。

キャッシュがディスエーブルの場合、デフォルトではキャッシュの内容が保持されます。WIC 命令または MicroBlaze のハードウェアデバッグロジックを使用すると、キャッシュラインを無効にできます。

### WIC 命令

オプションの WIC 命令 (C\_ALLOW\_ICACHE\_WR=1) を使用すると、命令キャッシュに含まれるキャッシュラインをアプリケーションから無効にできます。詳細は、第 4 章「MicroBlaze 命令セットアーキテクチャ」を参照してください。

# データ キャッシュ

## 概要

MicroBlaze は、オプションのデータ キャッシュを使用してパフォーマンスを向上させることができます。キャッシュ 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 キャッシュには、次の特徴があります。

- ダイレクト マップ方式 (連想度 1)
- ライト スルーまたはライト バック
- ユーザーがキャッシュ可能メモリのアドレス範囲を指定可能
- キャッシュ サイズおよびタグ サイズを設定可能
- CacheLink (XCL) インターフェイスを使用したキャッシュ
- 4 または 8 ワードのキャッシュ ライン
- MSR ビットを使用したキャッシュのオン/オフ制御
- データ キャッシュ ラインを無効にする、またはフラッシュする WDC 命令 (オプション)

## データ キャッシュの機能

データ キャッシュの使用時には、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の 2 つに分割されます。キャッシュ可能な領域は、C\_DCACHE\_BASEADDR および C\_DCACHE\_HIGHADDR の 2 つのパラメータで定義されます。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ています。その他のアドレスは、キャッシュ不可能なアドレスです。

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)。C\_DCACHE\_BASEADDR および C\_DCACHE\_HIGHADDR で指定する範囲は完全な 2 のべき乗の範囲にし、C\_DCACHE\_BASEADDR の最下位ビットは 0 にする必要があります。

図 1-21 に、データ キャッシュの構成を示します。

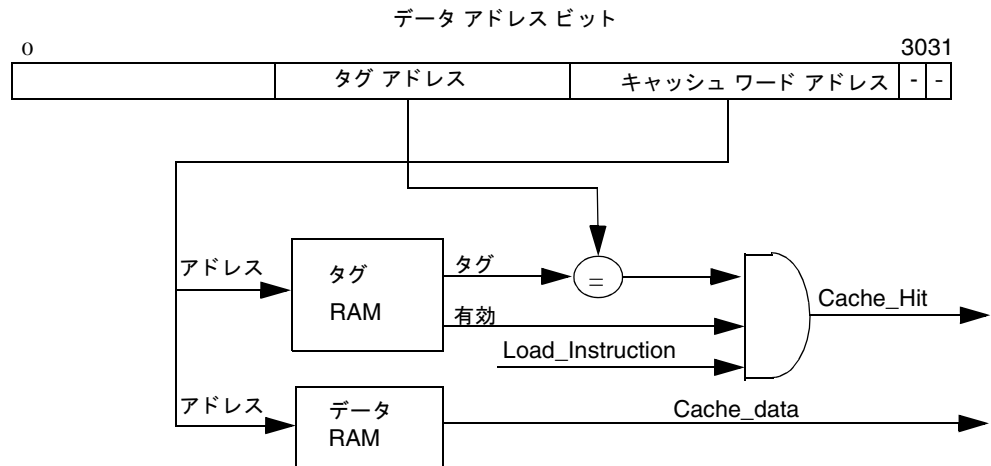


図 1-21 : データ キャッシュの構成

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 アドレスで構成されています。MicroBlaze のデータ キャッシュは 64B ~ 64KB の範囲で設定でき、これは 6 ~ 16 ビットのキャッシュ アドレスに対応しています。キャッシュ アドレスとタグ アドレスを合わせたものが、キャッシュ可能なメモリの完全アドレスに一致するように設定する必要があります。C\_AREA\_OPTIMIZED がセットされ C\_DCACHE\_USE\_WRITEBACK がセットされていない場合、ブロック RAM が常にデータ RAM のインプリメントに使用されるのを除き、2 kB 以下のキャッシュ サイズを選択する場合は、タグ RAM およびデータ RAM のインプリメントには分散 RAM が使用されます。

たとえば、C\_DCACHE\_BASEADDR=0x00400000、C\_DCACHE\_HIGHADDR=0x00403fff、C\_DCACHE\_BYTE\_SIZE=2048、および C\_DCACHE\_LEN=4 のように設定されている場合、キャッシュ可能な 16KB のメモリでは 14 ビットのバイト アドレスが使用され、2KB のキャッシュでは 11 ビットのバイト アドレスが使用されるため、必要なアドレス タグは 14 - 11 = 3 ビットです。この場合、512 ワードのデータの格納に RAMB16 が 1 個、128 個のキャッシュ ライン (3 ビットのタグ + 4 ワード有効ビット + 1 ライン有効ビット) エントリに RAMB16 が 1 個必要となるので、合計で 2 個の RAMB16 プリミティブが必要です。

## データ キャッシュの動作

MicroBlaze のデータ キャッシュ、ライトバック、またはライトスルーで使用されるキャッシュ ポリシーは、パラメータ C\_DCACHE\_USE\_WRITEBACK で決まります。このパラメータが設定されている場合ライトバック プロトコルがインプリメントされ、設定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。しかし、MMU (C\_USE\_MMU > 1、C\_AREA\_OPTIMIZED = 0、C\_DCACHE\_USE\_WRITEBACK = 1) でコンフィギュレーションされる場合、仮想モードのキャッシュ ポリシーは TLB エントリの W 保存属性で決まり、ライトバックは実モードで使用されます。

ライトバック プロトコルの場合、キャッシュ可能範囲内にあるアドレスへ保存すると常にキャッシュ データが更新されます。ターゲット アドレス ワードがキャッシュにない場合 (つまりアクセスがキャッシュ ミス) で、キャッシュのロケーションにメモリにまだ書き込まれていないデータがある場合、新しいデータでキャッシュが更新される前に、古いデータはデータ CacheLink (DXCL) を使用し外部メモリに書き込まれます。キャッシュ ライン全体を書き込む必要がある場合バースト

キャッシュ ライン書き込みが使用されますが、それ以外は、シングル ワード書き込みが使用されません。バイトまたはハーフワード保存の場合、キャッシュ ミスがあった場合は、アドレスはまず **CacheLink** を使用して要求され、ワード保存によりキャッシュが更新されます。

ライトスルー プロトコルを使用し、キャッシュ可能な範囲のアドレスに格納することにより、データ **CacheLink** を使用して同等のバイト、ハーフワード、またはワードを外部メモリに書き込むことが可能です。書き込みでは、ターゲット アドレス ワードがキャッシュ内の場合 (キャッシュ ヒットの場合) に、キャッシュ データも更新されます。キャッシュ ミスの場合は、キャッシュに関連キャッシュ ラインは読み込まれません。

キャッシュのイネーブル時に、データをキャッシュ可能な範囲のアドレスから読み出すと、要求データが現在キャッシュに含まれているかが確認されます。含まれている場合 (キャッシュ ヒットの場合) は要求データがキャッシュから取得され、含まれていない場合 (キャッシュ ミスの場合) はデータ **CacheLink** を使用してアドレスが要求され、パイプライン処理は要求アドレスに関連するキャッシュ ラインが外部メモリ コントローラから戻されるまでストールします。

## データ キャッシュのソフトウェア サポート

### MSR ビット

MSR の DCE ビットは、キャッシュのイネーブル、ディスエーブルを制御します。ディスエーブルにする場合は、PLB または OPB から読み出しを開始する前に、キャッシュ可能な範囲の書き込みがすべて外部メモリで完了していることを確認する必要があります。確認するには、キャッシュをオフにする直前にセマフォに書き込み、書き込みが完了するまでループ ポーリングを実行します。

キャッシュがディスエーブルの場合、デフォルトではキャッシュの内容が保持されます。

### WDC 命令

オプションの WDC 命令 (C\_ALLOW\_DCACHE\_WR=1) を使用すると、アプリケーションに含まれるデータ キャッシュのキャッシュ ラインを無効にするか、またはフラッシュできます。詳細は、[第 4 章「MicroBlaze 命令セット アーキテクチャ」](#)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 規格](#)に準拠しています。

- IEEE 754 の単精度浮動小数点フォーマット (無限の定義、NaN (Not A Number)、および 0 を含む) を使用
- 加算、減算、乗算、除算、比較、変換および平方根命令をサポート
- 最も近い値に繰り上げ/繰り下げるモード (Round-to-Nearest) をインプリメント
- アンダーフロー、オーバーフロー、ゼロでの除算、および無効な操作に対するステータス ビットを生成

パフォーマンスを向上させるため、次のような非標準的な簡略化も適用されています。

- 非正規化<sup>(1)</sup>オペランドはサポートされません。非正規化値に対してハードウェア浮動小数点の操作が実行されると、quiet NaN が返され、FSR の非正規化オペランド エラー ビットが 1 になります。詳細は、[36 ページの「浮動小数点ユニット ステータス レジスタ \(FSR\)」](#)を参照してください。
- 非正規化値の結果は、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 1 になります。この方法は、FTZ (Flush-to-Zero) と呼ばれます。
- quiet NaN に対する操作では、NaN オペランドの 1 つではなく固定の NaN (0xFFC00000) が返されます。
- 浮動小数点の操作によりオーバーフローが発生した場合は、常に符号付き  $\infty$  が返されます。

---

1.  $(1.17549 \times 10^{-38} > n > 0)$  または  $(0 > n > -1.17549 \times 10^{-38})$  の範囲内にある値  $n$  のように 0 に非常に近く、完全精度でも表すことができない値。

## フォーマット

IEEE 754 の単精度浮動小数点値は、次の 3 つのフィールドから構成されています。

1. 1 ビットの符号 (*sign*)
2. 8 ビットの指数部 (*exponent*) - ゲタばき表現
3. 23 ビットの仮数部 (*fraction*) - mantissa または significand

これらのフィールドは、図 1-22 に示すように 32 ビット ワードに格納されます。

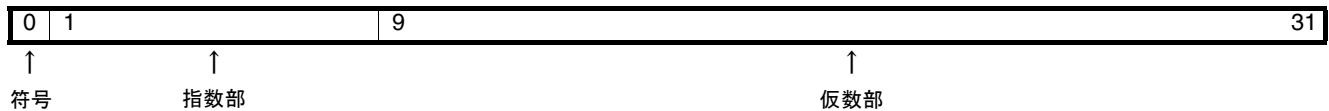


図 1-22 : IEEE 754 の単精度浮動小数点フォーマット

MicroBlaze では、浮動小数点値  $v$  は次のように解釈されます。

1.  $exponent = 255$ 、 $fraction \neq 0$  の場合、符号ビットにかかわらず  $v = NaN$  となります。
2.  $exponent = 255$ 、 $fraction = 0$  の場合、 $v = (-1)^{sign} \times \infty$  となります。
3.  $0 < exponent < 255$  の場合、 $v = (-1)^{sign} \times 2^{(exponent - 127)} \times (1.fraction)$
4.  $exponent = 0$ 、 $fraction \neq 0$  の場合、 $v = (-1)^{sign} \times 2^{-126} \times (0.fraction)$
5.  $exponent = 0$ 、 $fraction = 0$  の場合、 $v = (-1)^{sign} \times 0$  となります。

実際には 3 と 5 のみが有益で、その他の値ではエラーが発生するか、32 ビットでは完全な精度で表現できません。

## 繰り上げ/繰り下げ

MicroBlaze の FPU では、IEEE 754 で指定された、最も近い値に繰り上げ/繰り下げるデフォルトのモード (Round-to-Nearest) のみがインプリメントされます。このモードでは、浮動小数点の操作結果は、無限の精度の結果に最も近い単精度値となります。同程度に近い値が 2 つある場合は、最下位ビットが 0 のものが使用されます。

## 操作

MicroBlaze FPU 操作では、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。「汎用レジスタ」を参照してください。

### 演算

FPU では、次の浮動小数点演算がインプリメントされます。

- 加算 (fadd)
- 減算 (fsub)
- 乗算 (fmul)
- 除算 (fdiv)
- 平方根 (fsqrt、C\_USE\_FPU = 2 の場合使用可能)

### 比較

FPU では、次の浮動小数点比較がインプリメントされます。

- 小なり (fcmp.lt)
- 等価 (fcmp.eq)
- 以下 (fcmp.le)
- 大なり (fcmp.gt)
- 非等価 (fcmp.ne)
- 以上 (fcmp.ge)
- 順序付けなし (fcmp.un) - NaN で使用

## 変換

FPU では、次の変換がインプリメントされます (C\_USE\_FPU = 2 の場合)。

- 符号付き整数から浮動小数点への変換 (flt)
- 浮動小数点から符号付き整数への変換 (fint)

## 例外

MicroBlaze の浮動小数点ユニットは、通常のハードウェア例外メカニズムを使用します。例外がイネーブルの場合、IEEE 規格で指定される状態 (アンダーフロー、オーバーフロー、0 での除算、不正な操作) および MicroBlaze 特定の例外である非正規化オペランド エラーにより例外処理が実行されます。

浮動小数点の例外が発生すると、デスティネーション レジスタ (Rd) への書き込みが禁止されます。これにより、破損していないレジスタ ファイルが浮動小数点例外ハンドラにより処理されます。

## ソフトウェア サポート

GCC を基本にした EDK コンパイラ システムは、MicroBlaze API に準拠した浮動小数点ユニットをサポートしています。コンパイラのフラグは、XPS または SDK を使用する場合、システムにある FPU のタイプに基づき自動的に GCC コマンド ラインに追加されます。

倍精度の操作はすべてソフトウェアでエミュレートされます。xil\_printf() では浮動小数点の出力はサポートされていないので注意してください。標準 C ライブラリの printf() および関連関数では浮動小数点出力がサポートされていますが、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

EDK 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 FPU を利用するには、ライブラリを適切なコンパイラ オプションを設定して再コンパイルする必要があります。

別のコンパイルが使用されるケースでは、ビルド内の FPU のコンパイラ フラグが同じであることを確認することが重要です。

## オペレータ レイテンシ

FPU でサポートされているさまざまな操作のレイテンシは、第 4 章「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 度に 1 つの操作のみが行われます。



## C 言語のプログラミング

低位のアセンブリ言語プログラミングを使用せずに FPU を最大限に活用するには、C コンパイラでどのようにソースコードが解釈されるかを考慮することが重要です。同じアルゴリズムをさまざまな方法で書くことができますが、書き方によって効率の良し悪しが決まることもあります。

### 即値定数

C 言語の浮動小数点の定数はデフォルトで倍精度です。単精度の FPU を使用する場合、間違ったコード記述をしてしまうと、ソフトウェア エミュレーション ルーチンで、ネイティブの単精度命令ではなく倍精度が使用されてしまう場合があります。これを回避するには、演算式の即値定数を単精度に指定します。

例：

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

コンパイラ フラグ `-fsingle-precision-constants` を使用することで、GNU C コンパイラで浮動小数点定数が単精度として処理されるように (ANSI C 標準とは異なる) 設定することができます。

### 不要な型変換を避ける

浮動小数点と整数の変換はハードウェアでサポートされていますが、`C_USE_FPU` が 2 に設定されている場合、できる限り変換を避けるのが最善です。

次の例は「悪い例」で、浮動小数点で 1 から 10 の整数の平方和が計算されています。

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

上述のコードでは、各ループの繰り返しで整数から浮動小数点への型変換が必要です。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;
}
```

上記の 2 つのコード例で異なる結果が出力される場合があるため (t の値が非常に大きくなるなど)、通常、コンパイラではこの最適化を自由には実行できません。

### 平方根ランタイム ライブラリ関数

標準 C ランタイム数学ライブラリ関数は、倍精度の演算を行います。単精度の FPU を使用する場合、平方根関数 `sqrt()` の呼び出しで、FPU 命令ではなく非効率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 **math.h** ヘッダがコンパイラからの警告メッセージを避けるために含まれています。

単精度データ型で使用される場合、結果が倍精度に変換され、FPU を使用しないランタイム ライブラリ コールが呼び出されます。そして、浮動小数点への切捨てが実行されます。

これを解決するには、ANSI でない関数 **sqrtf()** を使用します。これは単精度を使用し、FPU を使用して実行することができます。

例 :

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

このコードをコンパイルする場合、コンパイラ フラグは **-mhard-float** および **-mxl-float-sqrt** に加え、**-fno-math-errno** を使用する必要があります。これは、**errno** 変数をアップデートすることでエラー コンディションを処理するための不必要なコードがコンパイラで生成されないようにするためです。

## 高速シンプレックス リンク (FSL)

MicroBlaze では、入力ポートと出力ポートを 1 つずつ持つ高速シンプレックス リンク (FSL) インターフェイスを 16 個まで使用できます。FSL チャネルは、ポイント ツー ポイント接続された単方向データ ストリーミング インターフェイスです。FSL インターフェイスの詳細は、ザイリンクス EDK IP の資料のデータシート (DS449) 『Fast Simplex Link (FSL) Bus』を参照してください。

MicroBlaze の FSL インターフェイスは、32 ビット幅です。別のビットで、送信/受信ワードが制御信号であるかデータであるかが示されます。FSL 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** 命令を使用します。汎用レジスタから FSL ポートに情報を転送するには、**put** 命令を使用します。どちらの命令にも、ブロッキング データ用、ノンブロッキング データ用、ブロッキング コントロール用、ノンブロッキング コントロール用の 4 種類があります。**get** 命令および **put** 命令の詳細は、第 4 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 アクセラレータへの FSL の使用

FSL は、プロセッサ パイプラインに対するレイテンシが短い専用インターフェイスとして使用できます。そのため、プロセッサ実行ユニットをカスタム ハードウェア アクセラレータを使用して拡張する際に適しています。図 1-23 に、簡単な例を示します。

### コード例

```
// Configure  $f_x$ 
cput Rc, RFSLx

// Store operands
put Ra, RFSLx // op 1
put Rb, RFSLx // op 2

// Load result
get Rt, RFSLx
```

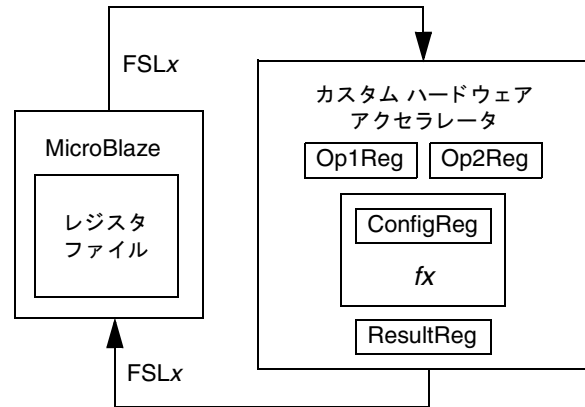


図 1-23 : ハードウェア アクセラレータ ファンクション  $f_x$  に FSL を使用した例

この方法は、ISA をカスタム命令で拡張する方法に似ていますが、プロセッサ パイプライン全体の速度がカスタム ファンクションに依存しないという利点があります。また、このように機能を拡張する場合、ソフトウェア ツール チェーンに追加の要件はありません。

## デバッグおよびトレース

### デバッグの概要

MicroBlaze には、XMD (Xilinx Microprocessor Debug) のような JTAG ベースのソフトウェア デバッグ ツール (バググラウンド デバッグ モード (BDM) デバッガと呼ばれる) をサポートするデバッグ インターフェイスが含まれています。このデバッグ インターフェイスは、ザイリンクス FPGA の JTAG ポートととのインターフェイスに使用するマイクロプロセッサ デバッグ モジュール (MDM) コアに接続されます。複数の MicroBlaze インスタンスを 1 個の MDM に接続してマルチプロセッサをデバッグできます。デバッグの機能は、次のとおりです。

- ハードウェアのブレークポイント数およびウォッチポイント数を設定可能、ソフトウェアのブレークポイント数は無制限
- 外部プロセッサ制御によりデバッグ ツールで MicroBlaze を停止、リセット、シングル ステップ可能
- メモリ、汎用レジスタ、特殊用途レジスタに対する読み出し/書き込み (EAR、EDR、ESR、BTR、および PVR0 ~ PVR11 は読み出しのみ)
- 複数のプロセッサをサポート
- 命令キャッシュおよびデータ キャッシュへの書き込み

### トレースの概要

MicroBlaze のトレース インターフェイスは、パフォーマンスをモニタおよび解析するため、多数の内部ステート信号を送信します。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でのみ使用することをお勧めします。このインターフェイスは、MicroBlaze の将来のリリースではサポートされない可能性があります。

# MicroBlaze の信号インターフェイス

---

この章では、MicroBlaze™ に接続できる信号インターフェイスのタイプについて説明します。次のセクションより構成されています。

- 概要
- MicroBlaze I/O の概要
- プロセッサ ローカル バス (PLB) インターフェイス
- オンチップ ペリフェラル バス (OPB) インターフェイス
- ローカル メモリ バス (LMB) インターフェイス
- 高速シンプレックス リンク (FSL) インターフェイス
- ザイリンクス CacheLink (XCL) インターフェイス
- デバッグ インターフェイス
- トレース インターフェイス
- MicroBlaze コアのコンフィギュレーション

## 概要

MicroBlaze コアは、データ アクセスおよび命令アクセスに対し別々のバス インターフェイス ユニットを持つハーバード アーキテクチャ構造になっています。ローカル メモリ バス (LMB)、IBM のプロセッサ ローカル バス (PLB) またはオンチップ ペリフェラル バス (OPB)、およびザイリンクス CacheLink (XCL) の 3 つのメモリ インターフェイスがサポートされています。LMB では、オンチップのデュアルポート ブロック RAM に 1 クロック サイクルでアクセスできます。PLB および OPB インターフェイスは、オンチップとオフチップの両方のペリフェラルおよびメモリに接続できます。CacheLink インターフェイスは、専用の外部メモリ コントローラ用です。また MicroBlaze では、マスタおよびスレーブ FSL インターフェイスを 1 つずつ持つ高速シンプレックス リンク (FSL) ポートを 16 個まで使用できます。

## 機能

MicroBlaze バスは、次のバス インターフェイスを使用してコンフィギュレーションできます。

- 32 ビットの PLB V4.6 インターフェイス (IBM の『128-Bit Processor Local Bus Architectural Specifications, Version 4.6』を参照)
- 32 ビットの OPB V2.0 バス インターフェイス (IBM の『64-Bit On-Chip Peripheral Bus, Architectural Specifications, Version 2.0』を参照)
- LMB ではブロック RAM を効率よく転送する単純な同期プロトコルを提供
- FSL では、高速でアービトレーションを使用しないストリーミング通信メカニズムを提供

- XCL では、キャッシュと外部メモリ コントローラの間にアービトレーションを使用した高速ストリーミング インターフェイスを提供
- マイクロプロセッサ デバッグ モジュール (MDM) コアを使用したデバッグ 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 I/O の概要

図 2-1 および表 2-1 に示すコア インターフェイスは、次のように定義されています。

DPLB : データ インターフェイス、プロセッサ ローカル バス

DOPB : データ インターフェイス、オンチップ ペリフェラル バス

DLMB : データ インターフェイス、ローカル メモリ バス (BRAM のみ)

IPLB : 命令インターフェイス、プロセッサ ローカル バス

IOPB : 命令インターフェイス、オンチップ ペリフェラル バス

ILMB : 命令インターフェイス、ローカル メモリ バス (BRAM のみ)

MFSL 0..15 : FSL マスタ インターフェイス

DWFSL 0..15 : FSL マスタ直接接続インターフェイス

SFSL 0..15 : FSL スレーブ インターフェイス

DRFSL 0..15 : FSL スレーブ直接接続インターフェイス

IXCL : 命令側ザイリンクス CacheLink インターフェイス (FSL マスタ/スレーブ ペア)

DXCL : データ側ザイリンクス CacheLink インターフェイス (FSL マスタ/スレーブ ペア)

コア : クロック、リセット、デバッグ、トレース用のその他の信号

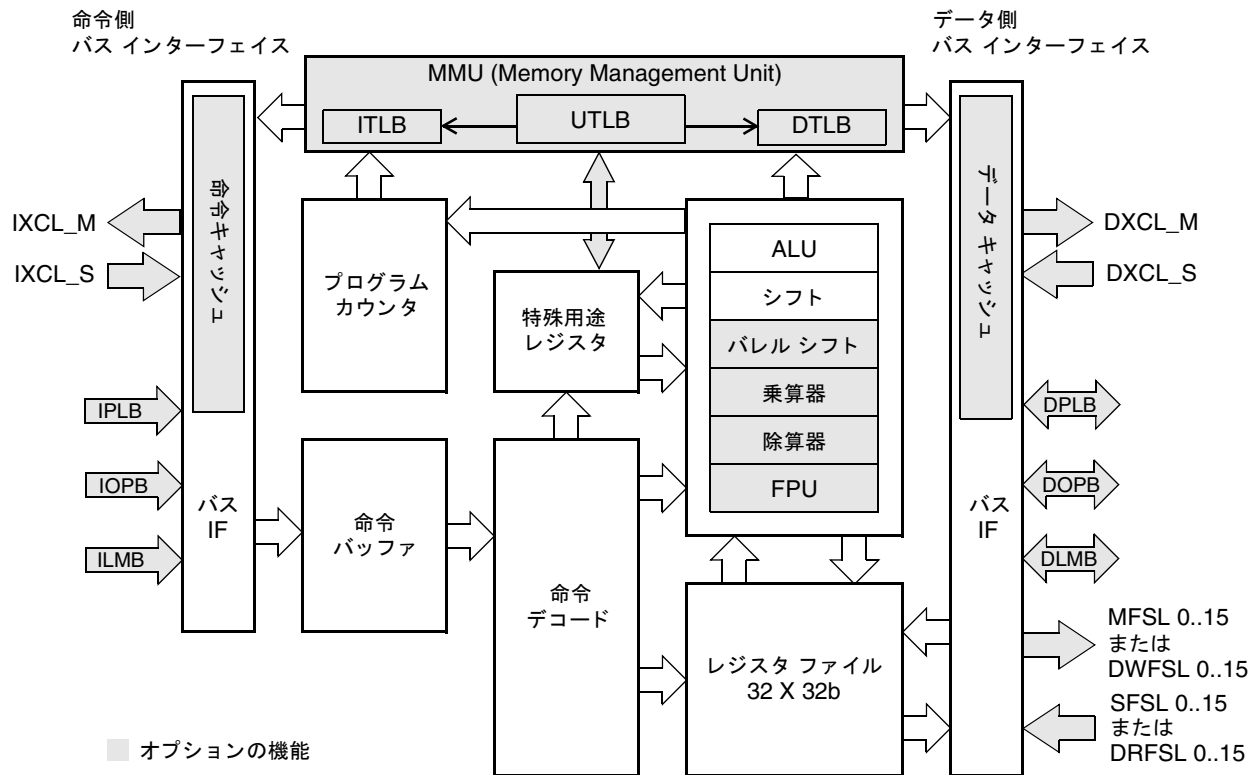


図 2-1 : MicroBlaze コアのブロック図

表 2-1 : MicroBlaze コア I/O のサマリ

信号	インターフェイス	I/O	説明
DM_ABus[0:31]	DOPB	O	データ インターフェイス OPB アドレス バス
DM_BE[0:3]	DOPB	O	データ インターフェイス OPB バイト イネーブル
DM_busLock	DOPB	O	データ インターフェイス OPB バス ロック
DM_DBus[0:31]	DOPB	O	データ インターフェイス OPB 書き込みデータ バス
DM_request	DOPB	O	データ インターフェイス OPB バス要求
DM_RNW	DOPB	O	データ インターフェイス OPB 読み出し、書き込みはなし
DM_select	DOPB	O	データ インターフェイス OPB セレクト
DM_seqAddr	DOPB	O	データ インターフェイス OPB シーケンシャル アドレス
DOPB_DBus[0:31]	DOPB	I	データ インターフェイス OPB 読み出しデータ バス
DOPB_errAck	DOPB	I	データ インターフェイス OPB エラー通知
DOPB_MGrant	DOPB	I	データ インターフェイス OPB バス許可
DOPB_retry	DOPB	I	データ インターフェイス OPB バス サイクル再試行
DOPB_timeout	DOPB	I	データ インターフェイス OPB タイムアウト エラー
DOPB_xferAck	DOPB	I	データ インターフェイス OPB 転送 ACK 信号
IM_ABus[0:31]	IOPB	O	命令インターフェイス OPB アドレス バス
IM_BE[0:3]	IOPB	O	命令インターフェイス OPB バイト イネーブル
IM_busLock	IOPB	O	命令インターフェイス OPB バス ロック
IM_DBus[0:31]	IOPB	O	命令インターフェイス OPB 書き込みデータ バス (常に 0x00000000)
IM_request	IOPB	O	命令インターフェイス OPB バス要求
IM_RNW	IOPB	O	命令インターフェイス OPB 読み出し、書き込みはなし (IM_select に接続)
IM_select	IOPB	O	命令インターフェイス OPB セレクト
IM_seqAddr	IOPB	O	命令インターフェイス OPB シーケンシャル アドレス
IOPB_DBus[0:31]	IOPB	I	命令インターフェイス OPB 読み出しデータ バス
IOPB_errAck	IOPB	I	命令インターフェイス OPB エラー通知
IOPB_MGrant	IOPB	I	命令インターフェイス OPB バス許可
IOPB_retry	IOPB	I	命令インターフェイス OPB バス サイクル再試行
IOPB_timeout	IOPB	I	命令インターフェイス OPB タイムアウト エラー
IOPB_xferAck	IOPB	I	命令インターフェイス OPB 転送 ACK 信号
DPLB_M_Abort	DPLB	O	データ インターフェイス PLB 中止バス要求インジケータ
DPLB_M_ABus	DPLB	O	データ インターフェイス PLB アドレス バス
DPLB_M_UABus	DPLB	O	データ インターフェイス PLB 上位アドレス バス

表 2-1 : MicroBlaze コア I/O のサマリ(続き)

信号	インターフェイス	I/O	説明
DPLB_M_BE	DPLB	O	データ インターフェイス PLB バイト イネーブル
DPLB_M_busLock	DPLB	O	データ インターフェイス PLB バス ロック
DPLB_M_lockErr	DPLB	O	データ インターフェイス PLB ロック エラー インジケータ
DPLB_M_MSize	DPLB	O	データ インターフェイス PLB マスタ データ バス サイズ
DPLB_M_priority	DPLB	O	データ インターフェイス PLB バス要求優先順位
DPLB_M_rdBurst	DPLB	O	データ インターフェイス PLB バースト読み出し転送インジケータ
DPLB_M_request	DPLB	O	データ インターフェイス PLB バス要求
DPLB_M_RNW	DPLB	O	データ インターフェイス PLB 読み出し、書き込みはなし
DPLB_M_size	DPLB	O	データ インターフェイス PLB 転送サイズ
DPLB_M_TAttribute	DPLB	O	データ インターフェイス PLB 転送属性バス
DPLB_M_type	DPLB	O	データ インターフェイス PLB 転送形式
DPLB_M_wrBurst	DPLB	O	データ インターフェイス PLB バースト書き込み転送インジケータ
DPLB_M_wrDBus	DPLB	O	データ インターフェイス PLB 書き込みデータ バス
DPLB_MBusy	DPLB	I	データ インターフェイス PLB スレーブ ビジー インジケータ
DPLB_MRdErr	DPLB	I	データ インターフェイス PLB スレーブ読み出しエラー インジケータ
DPLB_MWrErr	DPLB	I	データ インターフェイス PLB スレーブ書き込みエラー インジケータ
DPLB_MIRQ	DPLB	I	データ インターフェイス PLB スレーブ割り込みインジケータ
DPLB_MWrBTerm	DPLB	I	データ インターフェイス PLB 終端書き込みバーストインジケータ
DPLB_MWrDAck	DPLB	I	データ インターフェイス PLB 書き込みデータ応答
DPLB_MAddrAck	DPLB	I	データ インターフェイス PLB アドレス応答
DPLB_MRdBTerm	DPLB	I	データ インターフェイス PLB 終端読み出しバーストインジケータ
DPLB_MRdDAck	DPLB	I	データ インターフェイス PLB 読み出しデータ応答
DPLB_MRdDBus	DPLB	I	データ インターフェイス PLB 読み出しデータ バス
DPLB_MRdWdAddr	DPLB	I	データ インターフェイス PLB 読み出しワード アドレス
DPLB_MRearbitrate	DPLB	I	データ インターフェイス PLB バス再アービトレーション インジケータ
DPLB_MSSize	DPLB	I	データ インターフェイス PLB スレーブ データ バス サイズ



表 2-1 : MicroBlaze コア I/O のサマリ(続き)

信号	インターフェイス	I/O	説明
DPLB_MTimeout	DPLB	I	データ インターフェイス PLB バス タイムアウト
IPLB_M_ABoort	IPLB	O	命令インターフェイス PLB 中止バス 要求インジケータ
IPLB_M_ABus	IPLB	O	命令インターフェイス PLB アドレス バス
IPLB_M_UABus	IPLB	O	命令インターフェイス PLB 上位アドレス バス
IPLB_M_BE	IPLB	O	命令インターフェイス PLB バイト イネーブル
IPLB_M_busLock	IPLB	O	命令インターフェイス PLB バス ロック
IPLB_M_lockErr	IPLB	O	命令インターフェイス PLB ロック エラー インジケータ
IPLB_M_MSize	IPLB	O	命令インターフェイス PLB マスタ データ バス サイズ
IPLB_M_priority	IPLB	O	命令インターフェイス PLB バス要求優先順位
IPLB_M_rdBurst	IPLB	O	命令インターフェイス PLB バースト読み出し転送インジケータ
IPLB_M_request	IPLB	O	命令インターフェイス PLB バス要求
IPLB_M_RNW	IPLB	O	命令インターフェイス PLB 読み出し、書き込みはなし
IPLB_M_size	IPLB	O	命令インターフェイス PLB 転送サイズ
IPLB_M_TAttribute	IPLB	O	命令インターフェイス PLB 転送属性バス
IPLB_M_type	IPLB	O	命令インターフェイス PLB 転送形式
IPLB_M_wrBurst	IPLB	O	命令インターフェイス PLB バースト書き込み転送インジケータ
IPLB_M_wrDBus	IPLB	O	命令インターフェイス PLB 書き込みデータ バス
IPLB_MBusy	IPLB	I	命令インターフェイス PLB スレーブ ビジー インジケータ
IPLB_MRdErr	IPLB	I	命令インターフェイス PLB スレーブ読み出しエラーインジケータ
IPLB_MWrErr	IPLB	I	命令インターフェイス PLB スレーブ書き込みエラーインジケータ
IPLB_MIRQ	IPLB	I	命令インターフェイス PLB スレーブ割り込みインジケータ
IPLB_MWrBTerm	IPLB	I	命令インターフェイス PLB 終端書き込みバーストインジケータ
IPLB_MWrDAck	IPLB	I	命令インターフェイス PLB 書き込みデータ応答
IPLB_MAddrAck	IPLB	I	命令インターフェイス PLB アドレス応答
IPLB_MRdBTerm	IPLB	I	命令インターフェイス PLB 終端読み出しバーストインジケータ
IPLB_MRdDAck	IPLB	I	命令インターフェイス PLB 読み出しデータ応答
IPLB_MRdDBus	IPLB	I	命令インターフェイス PLB 読み出しデータ バス
IPLB_MRdWdAddr	IPLB	I	命令インターフェイス PLB 読み出しワード アドレス
IPLB_MRearbitrate	IPLB	I	命令インターフェイス PLB バス再アービトレーション インジケータ

表 2-1 : MicroBlaze コア I/O のサマリ(続き)

信号	インターフェイス	I/O	説明
IPLB_MSSize	IPLB	I	命令インターフェイス PLB スレーブ データ バス サイズ
IPLB_MTimeout	IPLB	I	命令インターフェイス PLB バス タイムアウト
Data_Addr[0:31]	DLMB	O	データ インターフェイス LMB アドレス バス
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 ストロープ
Read_Strobe	DLMB	O	データ インターフェイス LMB 読み出しストロープ
Write_Strobe	DLMB	O	データ インターフェイス LMB 書き込みストロープ
Data_Read[0:31]	DLMB	I	データ インターフェイス LMB 読み出しデータ バス
DReady	DLMB	I	データ インターフェイス LMB データ Ready
Instr_Addr[0:31]	ILMB	O	命令インターフェイス LMB アドレス バス
I_AS	ILMB	O	命令インターフェイス LMB アドレス ストロープ
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 Ready
FSL0_M .. FSL15_M	MFSL または DWFSL	O	出力 FSL チャンネルへのマスタ インターフェイス MFSL は FSL バス接続に、DWFSL は FSL スレーブ との直接接続に使用されます。
FSL0_S .. FSL15_S	SFSL または DRFSL	I	入力 FSL チャンネルへのスレーブ インターフェイス SFSL は FSL バス接続に、DRFSL は FSL マスタとの 直接接続に使用されます。
ICache_FSL_in...	IXCL_S	IO	命令側 CacheLink FSL スレーブ インターフェイス
ICache_FSL_out...	IXCL_M	IO	命令側 CacheLink FSL マスタ インターフェイス
DCache_FSL_in...	DXCL_S	IO	データ側 CacheLink FSL スレーブ インターフェイス
DCache_FSL_out...	DXCL_M	IO	データ側 CacheLink FSL マスタ インターフェイス
Interrupt	コア	I	割り込み
Reset <sup>1</sup>	コア	I	コアリセット、アクティブ High。1 Clk クロック サイクル以上保持する必要あり。
Mb_Reset <sup>1</sup>	コア	I	コアリセット、アクティブ High。1 Clk クロック サイクル以上保持する必要あり。
Clk	コア	I	クロック <sup>2</sup>
Ext_BRK	コア	I	MDM からの BREAK 信号
Ext_NM_BRK	コア	I	MDM からのマスク不可の BREAK 信号
MB_Halted	コア	O	デバッグ インターフェイスを介した、または Dbg_Stop の設定によるパイプライン停止

表 2-1 : MicroBlaze コア I/O のサマリ(続き)

信号	インターフェイス	I/O	説明
Dbg_Stop	コア	I	無条件でパイプラインを停止します。立ち上がりエッジの検出パルスは最低 Clk クロック サイクル間保持する必要があります。C_DEBUG_ENABLED が 1 に設定されるときのみ影響を受けます。
Dbg_...	コア	IO	MDM からのデバッグ信号 (表 2-9 参照)
Trace_...	コア	O	リアルタイム ハードウェア解析用のトレース信号 (表 2-10 参照)

1. Reset 信号および Mb\_Reset 信号の働きは同じですが、Reset は主に OPB インターフェイスで、Mb\_Reset は PLB インターフェイスで使用されます。
2. MicroBlaze is のデザインは Clk 信号に同期します。ただし、ハードウェア デバッグ ロジックは Dbg\_Clk 信号に同期します。ハードウェア デバッグ ロジックが使用されない場合は Clk 信号には最小周波数の制限はありません。使用される場合はこれらのクロック領域間で信号が送信されますが、Clk 信号の周波数を Dbg\_Clk の周波数よりも高くする必要があります。

## プロセッサ ローカル バス (PLB) インターフェイス

MicroBlaze の PLB インターフェイスは、バイト イネーブルが可能な 32 ビットのマスタとしてインプリメントされています。詳細は、『IBM 128-Bit Processor Local Bus Architectural Specification (v4.6)』を参照してください。

## オンチップ ペリフェラル バス (OPB) インターフェイス

MicroBlaze の OPB インターフェイスは、バイト イネーブルが可能なマスタとしてインプリメントされています。詳細は、『OPB Usage in Xilinx FPGA』を参照してください。

## ローカル メモリ バス (LMB) インターフェイス

LMB は、主にオンチップ ブロック RAM にアクセスするために使用する同期バスです。このバスでは、最小数の制御信号と単純なプロトコルを 1 個使用して、ローカル ブロック RAM に 1 クロック サイクルでアクセスできます。LMB の信号とその定義を次の表に示します。LMB 信号はすべてアクティブ High です。

### LMB 信号のインターフェイス

表 2-2 : LMB バスの信号

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[0:31]	Data_Addr[0:31]	Instr_Addr[0:31]	O	アドレス バス
Byte_Enable[0:3]	Byte_Enable[0:3]	未使用	O	バイト イネーブル
Data_Write[0:31]	Data_Write[0:31]	未使用	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	読み出し中
Write_Strobe	Write_Strobe	未使用	O	書き込み中

表 2-2 : LMB バスの信号

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
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
Clk	Clk	Clk	I	バス クロック

### Addr[0:31]

アドレス バスはコアの出力で、現在の転送によりアクセスされているメモリ アドレスを示します。このバスは AS が High のときのみ有効です。複数のクロック サイクルが必要となるアクセスでは、Addr[0:31] はその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]

バイト イネーブル信号はコアの出力で、有効なデータを含むデータ バスのバイト レーンを示します。Byte\_Enable[0:3] は、AS が High のときのみ有効です。複数のクロック サイクルが必要となるアクセスでは、Byte\_Enable[0:3] はその転送の最初のクロック サイクルでのみ有効です。次の表に、Byte\_Enable[0:3] の有効な値を示します。

表 2-3 : Byte\_Enable[0:3] での有効値

Byte_Enable[0:3]	使用されるバイト レーン			
	Data[0:7]	Data[8:15]	Data[16:23]	Data[24:31]
0000				
0001				x
0010			x	
0100		x		
1000	x			
0011			x	x
1100	x	x		
1111	x	x	x	x

### Data\_Write[0:31]

書き込みデータ バスはコアの出力で、メモリに書き込まれるデータが含まれています。このバスは AS が High のときのみ有効です。Byte\_Enable[0:3] で指定されるバイト レーンにのみ有効なデータが含まれています。

### AS

アドレス ストロープはコアの出力で、転送の開始を示し、アドレス バスおよびバイト イネーブルを有効にします。このストロープは、転送の最初のクロック サイクルでのみ High になり、その後次の転送の開始まで Low になります。

### Read\_Strobe

読み出しストロープはコアの出力で、読み出し転送が進行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 になった後のクロック サイクル

まで **High** のままになる場合があります。次のクロック サイクルで新たな読み出し転送が直接開始される場合は、**Read\_Strobe** は **High** のままになります。

## Write\_Strobe

書き込みストロブはコアの出力で、書き込み転送が進行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** になった後のクロック サイクルまで **High** のままになる場合があります。次のクロック サイクルで新たな書き込み転送が直接開始される場合は、**Write\_Strobe** は **High** のままになります。

## Data\_Read[0:31]

読み出し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みます。**Data\_Read[0:31]** は、**Ready** 信号が **High** のときのクロックの立ち上がりエッジで有効です。

## Ready

**Ready** 信号はコアへの入力で、現在の転送が完了し、次のクロック サイクルで次の転送が開始可能であることを示します。この信号は、クロックの立ち上がりエッジでサンプリングされます。読み出しでは、この信号は **Data\_Read[0:31]** バスが有効であることを示し、書き込みでは **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。

## Clk

LMB での動作は、すべて **MicroBlaze** コアのクロックに同期しています。

## LMB トランザクション

次に LMB バスの動作の例を示します。

### 一般的な書き込み動作

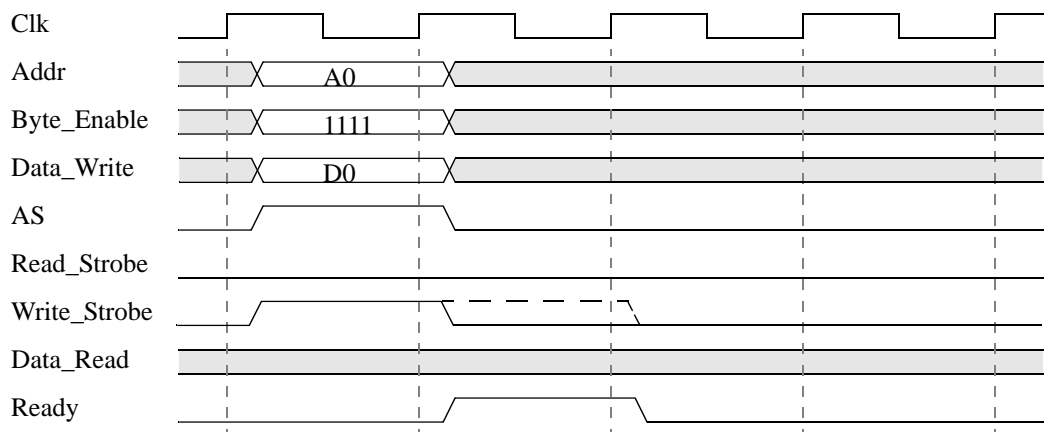


図 2-2 : LMB での一般的な書き込み動作

## 一般的な読み出し動作

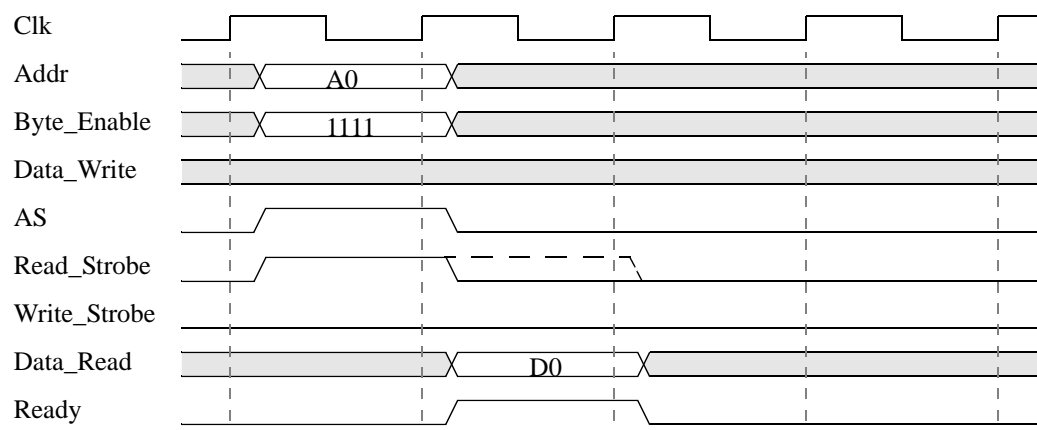


図 2-3 : LMB の一般的な読み出し動作

## 連続書き込み動作

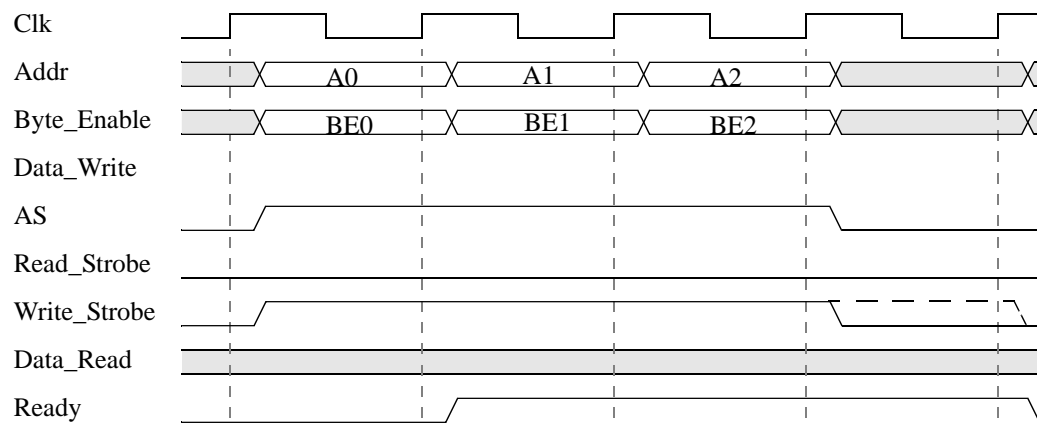


図 2-4 : LMB での連続書き込み動作

## 1 サイクルの連続読み出し動作

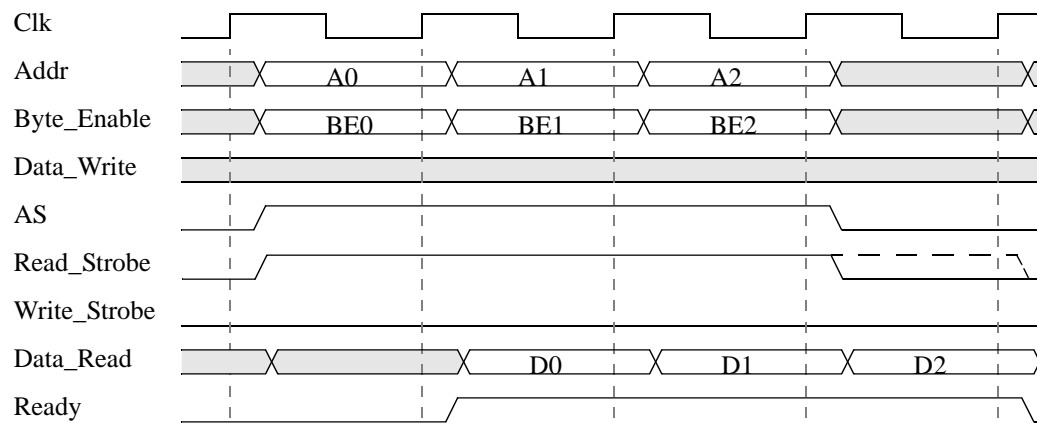


図 2-5 : LMB での 1 サイクルの連続読み出し動作

## 読み出し/書き込み混合の連続動作

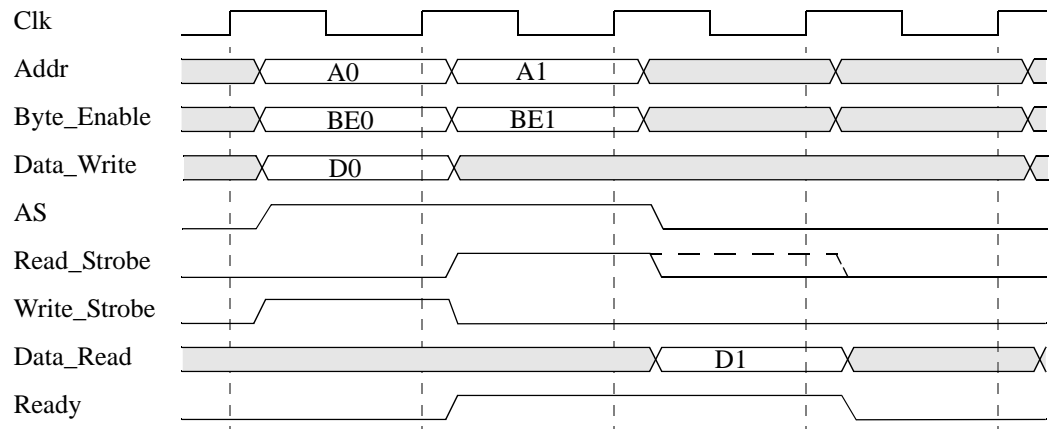


図 2-6 : 読み出し/書き込み混合の連続動作

## データの読み出しおよび書き込み操作

MicroBlaze のデータ側のバス インターフェイスでは、次の転送をサポートするのに必要となる読み出し操作および書き込み操作が実行されます。

- ワード デバイスへのバイト、ハーフワード、ワード転送
- ハーフワード デバイスへのバイト、ハーフワード転送
- バイト デバイスへのバイト転送

MicroBlaze では、指定デバイスの転送サイズを超える転送はサポートされていません。これらの転送では、MicroBlaze バス インターフェイスでサポートされていないダイナミック バス サイジングおよび変換サイクルが必要となります。読み出しサイクルに対するデータ操作を表 2-4 に、書き込みサイクルに対するデータ操作を表 2-5 に示します。

表 2-4 : 読み出しデータの操作 (レジスタ rD への読み込み)

アドレス [30:31]	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				Byte3
10	0010	バイト				Byte2
01	0100	バイト				Byte1
00	1000	バイト				Byte0
10	0011	ハーフワード			Byte2	Byte3
00	1100	ハーフワード			Byte0	Byte1
00	1111	ワード	Byte0	Byte1	Byte2	Byte3

表 2-5 : 書き込みデータの操作 (レジスタ rD からの格納)

アドレス [30:31]	Byte_Enable [0:3]	転送サイズ	書き込みデータ バスのバイト			
			Byte0	Byte1	Byte2	Byte3
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]

**メモ :** その他の OPB マスタでは、MicroBlaze で許容されるバイト レーン配置での要件よりも制限が多い可能性があります。OPB のスレーブ デバイスは、通常最上位のバイト レーンに接続されているバイト デバイスおよび最上位のハーフワード レーンに接続されているハーフワード デバイスに、左揃えで接続されています。MicroBlaze の操作ロジックでは、この接続方法が完全にサポートされています。

## 高速シンプレックス リンク (FSL) インターフェイス

FSL バスでは、出力 FIFO と入力 FIFO 間のポイント ツー ポイント通信が可能です。ジェネリック FSL プロトコルの詳細は、ザイリンクス EDK IP の資料の [データシート](#) (DS449) 『Fast Simplex Link (FSL) Bus』を参照してください。

### マスタ FSL 信号のインターフェイス

MicroBlaze では、16 個までのマスタ FSL インターフェイスを使用できます。マスタ信号を表 2-6 に示します。

表 2-6 : マスタ FSL 信号

信号名	説明	VHDL の型	方向
FSLn_M_Clk	クロック	std_logic	入力
FSLn_M_Write	データが出力 FSL に書き込まれていることを示すライト イネーブル信号	std_logic	出力
FSLn_M_Data	出力 FSL に書き込まれるデータ値	std_logic_vector	出力
FSLn_M_Control	出力 FSL に書き込まれる制御ビット値	std_logic	出力
FSLn_M_Full	出力 FIFO がフルであることを示すフルビット	std_logic	入力



## スレーブ FSL 信号のインターフェイス

MicroBlaze では、16 個までのスレーブ FSL インターフェイスを使用できます。スレーブ FSL インターフェイス信号を表 2-7 に示します。

表 2-7 : スレーブ FSL 信号

信号名	説明	VHDL の型	方向
FSLn_S_Clk	クロック	std_logic	入力
FSLn_S_Read	データが入力 FSL から読み出されていることを示す読み出し ACK 信号	std_logic	出力
FSLn_S_Data	入力 FSL の一番上で現在読み出し可能なデータ値	std_logic_vector	入力
FSLn_S_Control	入力 FSL の一番上で現在読み出し可能な制御ビット値	std_logic	入力
FSLn_S_Exists	入力 FSL にデータが存在することを示すフラグ	std_logic	入力

## FSL トランザクション

### FSL バス書き込み動作

MicroBlaze による FSL バスへの書き込みは、put 命令または putd 命令の 1 つを使用して実行されます。書き込みでは、レジスタの内容が出力 FSL に転送されます。この転送は通常、FSL FIFO がフルにならない限り、ブロッキング モードの書き込み (put および cput 命令) では 1 クロック サイクルで完了します。FSL FIFO がフルになると、FSL フル フラグが Low になるまでプロセッサがストールします。接頭辞が n のノンブロッキング命令では、FSL がフルでも、転送は 1 クロック サイクルで完了します。FSL がフルの場合、書き込みは実行されず、MSR のキャリー ビットが設定されます。

### FSL バス読み出し動作

MicroBlaze による FSL バスからの読み出しは、get 命令または getd 命令の 1 つを使用して実行されます。読み出しが実行されると、入力 FSL の内容が汎用レジスタに転送されます。この転送は通常、FSL FIFO にデータが存在すれば、ブロッキング モードの読み出しでは 2 クロック サイクルで完了します。FSL FIFO が空になると、FSL にデータが存在することを示すフラグが High になるまでプロセッサがこの命令でストールします。接頭辞が n のノンブロッキング モード命令では、FSL が空であるかどうかにかかわらず、転送は 2 クロック サイクルで完了します。FSL が空の場合、データ転送は実行されず、MSR のキャリー ビットが設定されます。

## 直接 FSL 接続

直接 FSL 接続を使用すると、FSL バスを使用する必要がなくなります。直接接続には FSL バス FIFO が含まれないため、2 つの接続された IP コア間にバッファを必要としない場合に有効です。バッファを使用しなければ、通信レイテンシおよび必要なインプリメンテーション リソースが削減されます。

各 MicroBlaze FSL インターフェイスで、直接 FSL 接続または FSL バスのいずれかを使用できます。

MicroBlaze DWFSL インターフェイスは直接 FSL 接続上のイニシエータで、DWFSL ターゲットにのみ接続できます。DWFSL イニシエータとターゲットの信号は同名で、表 2-6 に示される MFSL 信号と同一です。MicroBlaze では DWFSL インターフェイスを使用して、put または putd 命令のいずれかでデータをターゲットに書き込みます。

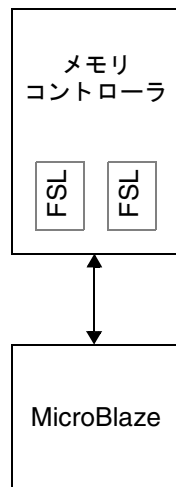
MicroBlaze DRFSL インターフェイスは直接 FSL 接続上のターゲットで、DRFSL イニシエータにのみ接続できます。DRFSL イニシエータとターゲットの信号は同名で、表 2-7 に示される SFSL 信号と同一です。MicroBlaze では DRFSL インターフェイスを使用して、get 命令または getd 命令のいずれかでデータをイニシエータから読み出します。

XCL (ザイリンクス CacheLink) インターフェイスは直接 FSL 接続を使用してインプリメントされます。

## ザイリンクス CacheLink (XCL) インターフェイス

ザイリンクス CacheLink (XCL) は、外部メモリにアクセスするためのハイ パフォーマンス ソリューションです。CacheLink インターフェイスは、MPMC などの統合 FSL バッファを使用してメモリ コントローラに直接接続するように設計されています。この方法を使用するとインスタネーション数も最少になり、レイテンシが最短になります (図 2-7 を参照)。

回路図



MHS コード例

```

BEGIN microblaze
  ...
  BUS_INTERFACE IXCL = myIXCL
  ...
END

BEGIN mch_opb_sdr
  ...
  BUS_INTERFACE MCH0 = myIXCL
  ...
END
  
```

図 2-7 : 統合 FSL バッファを使用した CacheLink の接続 (命令キャッシュのみを使用)

このインターフェイスは、キャッシュをイネーブルにしたときのみ使用可能です。命令側またはデータ側のいずれか一方のみに CacheLink のキャッシュを使用することもできます。

メモリ ロケーションのアクセス方法は、命令キャッシュではパラメータ `C_ICACHE_ALWAYS_USED`、データ キャッシュではパラメータ `C_DCACHE_ALWAYS_USED` によって決まります。パラメータが 1 に設定されている場合は、キャッシュ可能な範囲内のメモリ は常に CacheLink を介してアクセスされ、0 に設定されている場合は、ソフトウェアでオフに設定されている場合 (`MSR[DCE]=0` または `MSR[ICE]=0`) に、PLB または OPB を使用してアクセスされます。

キャッシュ可能な範囲外のメモリ ロケーションは、PLB、OPB または LMB を使用してアクセスされます。

CacheLink キャッシュ コントローラでは、選択されているプロトコルにより、クリティカルワードファーストまたはリニア フェッチを使用して、4 または 8 キャッシュ ラインが処理されます。また、キャッシュへのアクセスに PLB または OPB バスを使用しないため、キャッシュ以外のメモリアクセスの競合が減少します。

## CacheLink 信号のインターフェイス

MicroBlaze の CacheLink 信号を表 2-8 に示します。

表 2-8 : MicroBlaze の CacheLink 信号

信号名	説明	VHDL の型	方向
ICACHE_FSL_IN_Clk	命令側戻り読み出しデータ FSL へのクロック出力	std_logic	出力
ICACHE_FSL_IN_Read	命令側戻り読み出しデータ FSL への読み出し信号	std_logic	出力
ICACHE_FSL_IN_Data	命令側戻り読み出しデータ FSL からの読み出しデータ	std_logic_vector (0 ~ 31)	入力
ICACHE_FSL_IN_Control	命令側戻り読み出しデータ FSL からの FSL 制御ビット (将来の使用に予約)	std_logic	入力
ICACHE_FSL_IN_Exists	命令側戻り FSL に読み出しデータが存在	std_logic	入力
ICACHE_FSL_OUT_Clk	命令側読み出しアクセス FSL へのクロック出力	std_logic	出力
ICACHE_FSL_OUT_Write	命令側読み出しアクセス FSL に新規キャッシュミスアクセス要求を書き込み	std_logic	出力
ICACHE_FSL_OUT_Data	命令側読み出しアクセス FSL の新規キャッシュミスアクセス (アドレス)	std_logic_vector (0 ~ 31)	出力
ICACHE_FSL_OUT_Control	命令側読み出しアクセス FSL への FSL 制御ビット (将来の使用に予約)	std_logic	出力
ICACHE_FSL_OUT_Full	命令側読み出しアクセス用の FSL アクセスバッファはフル	std_logic	入力
DCACHE_FSL_IN_Clk	データ側戻り読み出しデータ FSL へのクロック出力	std_logic	出力
DCACHE_FSL_IN_Read	データ側戻り読み出しデータ FSL への読み出し信号	std_logic	出力
DCACHE_FSL_IN_Data	データ側戻り読み出しデータ FSL からの読み出しデータ	std_logic_vector (0 ~ 31)	入力
DCACHE_FSL_IN_Control	データ側戻り読み出しデータ FSL からの FSL 制御ビット	std_logic	入力
DCACHE_FSL_IN_Exists	データ側戻り FSL に読み出しデータが存在	std_logic	入力
DCACHE_FSL_OUT_Clk	データ側読み出しアクセス FSL へのクロック出力	std_logic	出力

表 2-8 : MicroBlaze の CacheLink 信号 (続き)

信号名	説明	VHDL の型	方向
DCACHE_FSL_OUT_Write	データ側読み出しアクセス FSL に新規キャッシュ ミス アクセス要求を書き込み	std_logic	出力
DCACHE_FSL_OUT_Data	データ側読み出しアクセス FSL へのキャッシュ ミス アクセス (読み出しアドレスまたは書き込みアドレス + 書き込みデータ + バイト ライト イネーブル + バースト書き込みエンコード)	std_logic_vector (0 ~ 31)	出力
DCACHE_FSL_OUT_Control	データ側読み出しアクセス FSL への FSL 制御ビット。読み出し/書き込み、バイト イネーブル、およびバースト書き込みエンコード用のアドレス ビット [30 ~ 31]	std_logic	出力
DCACHE_FSL_OUT_Full	データ側読み出しアクセス用の FSL アクセス バッファはフル	std_logic	入力

## CacheLink トランザクション

すべての CacheLink アクセスは、FSL FIFO ベースのトランザクション プロトコルに従います。

- アクセス情報は、FSL データおよび制御信号 (DCACHE\_FSL\_OUT\_Data、DCACHE\_FSL\_OUT\_Control、ICACHE\_FSL\_IN\_Data、および ICACHE\_FSL\_IN\_Control など) を介してエンコードされます。
- 情報は、ライト イネーブル信号 (DCACHE\_FSL\_OUT\_Write など) が High になると送信または保存されます。
- 書き込みは、フル信号が非アクティブ (DCACHE\_FSL\_OUT\_Full = 0 など) の場合にのみ可能です。このフル信号は、命令キャッシュ コントローラでは使用されません。
- ICACHE\_FSL\_IN\_Read および DCACHE\_FSL\_IN\_Read の使用はどのインターフェイス プロトコルが選択されているかによって決まります。
  - ◆ IXCL および DXCL プロトコルが選択されている場合、読み出し信号を High にすると情報が受信され (読み込まれ) ます。送信側が新しいデータがあることを通知するとき以外、信号は Low です。
  - ◆ IXCL2 および DXCL2 プロトコルが選択されている場合、読み出し信号が Low になると、受信側が新しいデータを受信できないことを示します。読み出し信号が High の場合にのみ、新しいデータは読み込まれ、送信側はデータが存在することを通知します。
- 読み出しは、新しいデータが存在する (ICACHE\_FSL\_IN\_Exists = 1 など) 場合にのみ可能です。

ジェネリック FSL プロトコルの詳細は、ザイリンクス EDK IP の資料の [データシート](#) (DS449) 『Fast Simplex Link (FSL) Bus』を参照してください。

CacheLink では、キャッシュ コントロール 1 つに対して受信 (スレーブ) FSL と送信 (マスタ) FSL が 1 つずつ使用されます。送信 FSL は アクセス要求を送信するのに使用され、受信 FSL は要求されたキャッシュ ラインを受信するのに使用されます。また、CacheLink では、トランザクション情報に対して、FSL データおよび制御信号を介した特定のエンコード手法が使用されます。

CacheLink プロトコルでの読み出しに使用されるキャッシュ ラインの長さは 4 または 8 ワードです。各キャッシュ ラインはクリティカル ワード ファーストでフェッチされるか、またはリニアにフェッチされますが、これは、どのインターフェイス プロトコルが選択されているかによります。

- クリティカル ワード ファーストは、IXCL および DXCL プロトコルで使用され、**C\_ICACHE\_INTERFACE = 0** および **C\_DCACHE\_INTERFACE = 0** とそれぞれ指定されている場合に選択されます。各キャッシュ ラインは、クリティカル ワード から始まります。つまり、アドレス 0x348 へのアクセスが 4 ワード キャッシュ ラインのミスであった場合、戻りキャッシュ ラインのアドレス シーケンスは、0x348、0x34c、0x340、0x344 となります。キャッシュ コントローラにより、最初のワードが実行ユニットに送信され、キャッシュ メモリにも保存されます。これにより、最初のワードが戻ると同時に実行が再開します。その後、キャッシュ ラインに残りの 3 つまたは 7 つのワードが保存されます。
- リニア フェッチは、IXCL2 および DXCL2 プロトコルで使用され、**C\_ICACHE\_INTERFACE = 1** および **C\_DCACHE\_INTERFACE = 1** とそれぞれ指定されている場合に選択されます。CacheLink のアドレス出力は、キャッシュ ライン サイズに調整されます。つまり、アドレス 0x348 へのアクセスは、4 ワードのキャッシュ ラインでミスとなり、CacheLink のアドレス出力が 0x340 となります。キャッシュ コントローラは、キャッシュ メモリにデータを保存し、要求されたワードを実行ユニットに随時転送します。

パラメータ **C\_DCACHE\_USE\_WRITEBACK** が 1 に設定されている場合、書き込み操作は、バースト書き込みおよびシングルス ワードを使用して、キャッシュ ライン全体を保存することができます。各キャッシュ ラインは常にリニアに保存され、CacheLink のアドレス出力はキャッシュ ライン サイズに調整されます。パラメータ **C\_DCACHE\_USE\_WRITEBACK** が 0 に設定されている場合、CacheLink のすべての書き込み操作はシングルス ワードによるものです。ライトバックが使用されている場合、バースト書き込みは DXCL2 プロトコルでのみ使用可能なので、**C\_DCACHE\_INTERFACE** は 1 に設定する必要があります。

## 命令キャッシュの読み出しミス

読み出しミスが発生すると、キャッシュ コントローラは次のシーケンスを実行します。

1. 制御ビットを **Low** (**ICACHE\_FSL\_OUT\_Control = 0**) に設定して読み出しアクセスを示し、ワードを揃えた<sup>(1)</sup> またはキャッシュ ラインでそろえたミス アドレスを **ICACHE\_FSL\_OUT\_Data** に書き込みます。
2. データが存在することを示す **ICACHE\_FSL\_IN\_Exists** が **High** になるまで待機します。

**メモ :** **ICACHE\_FSL\_IN\_Exists** が **High** になる前に、少なくとも 1 クロック サイクル間 **wait** 状態である必要があります。

IXCL プロトコル (クリティカル ワード ファースト)

3. **ICACHE\_FSL\_IN\_Data** からのワードをキャッシュに保存します。
4. クリティカル ワードを実行ユニットに送信し、実行を再開します。
5. キャッシュ ラインの残りの 3 ワードに対し、手順 3 ~ 4 を繰り返します。

IXCL2 プロトコル (リニア フェッチ)

3. **ICACHE\_FSL\_IN\_Data** からのワードをキャッシュに保存します。
4. 関連ワードを実行ユニットに送信し、実行を再開します。
5. **ICACHE\_FSL\_IN\_Data** からの残りのワードをキャッシュに保存します。

---

1. バイトおよびハーフワードの読み出しミスでは、完全なワードが返され、キャッシュ コントローラにより実行ユニットに正しいバイトが送信されます。

## データ キャッシュの読み出しミス

1. DCACHE\_FSL\_OUT\_Full = 1 の場合、この信号が Low になるまで動作をストールします。
2. 制御ビットを Low (DCACHE\_FSL\_OUT\_Control = 0) に設定して読み出しアクセスを示し、ワードに揃えた<sup>(1)</sup>またはキャッシュ ラインで揃えたミス アドレスを DCACHE\_FSL\_OUT\_Data に書き込みます。
3. データが存在することを示す DCACHE\_FSL\_IN\_Exists が High になるまで待機します。

メモ : DCACHE\_FSL\_IN\_Exists が High になる前に、少なくとも 1 クロック サイクル間 wait 状態である必要があります。

### DXCL プロトコル (クリティカル ワード ファースト)

4. DCACHE\_FSL\_IN\_Data からのワードをキャッシュに保存します。
5. クリティカル ワードを実行ユニットに送信し、実行を再開します。
6. キャッシュ ラインの残りの 3 ワードまたは 7 ワードに対し、手順 4 ~ 5 を繰り返します。

### DXCL2 プロトコル (リニア フェッチ)

3. DCACHE\_FSL\_IN\_Data からのワードをキャッシュに保存します。
4. 要求されたワードを実行ユニットに送信し、実行を再開します。
5. DCACHE\_FSL\_IN\_Data からの残りのワードをキャッシュに保存します。

## データ キャッシュ書き込み

C\_DCACHE\_INTERFACE が 1 に設定されている場合、CacheLink はバースト書き込みまたはシングル ワード書き込みを実行することができます。C\_DCACHE\_USE\_WRITEBACK が 1 に設定されている場合、キャッシュ ライン全体が有効です。

C\_DCACHE\_USE\_WRITEBACK がクリアされて 0 になる場合、データ キャッシュへの書き込みは常にライト スルーなので、キャッシュのデータ有無にかかわらず、CacheLink を介した書き込みが行われます。

DXCL2 プロトコルが選択されている場合、バースト キャッシュ ライン書き込みで、キャッシュ コントローラは次のシーケンスを実行します。

1. DCACHE\_FSL\_OUT\_Full = 1 の場合、この信号が Low になるまで動作を停止します。
2. DCACHE\_FSL\_OUT\_Data へのアドレスにキャッシュを揃えた場合、制御ビットを High (DCACHE\_FSL\_OUT\_Control = 1) に設定して書き込みアクセスを示します。このアドレスの最下位の 2 ビット (30:31) は、0b10=burst のバースト アクセスをエンコードするために使用されます。シングル バイト書き込みからのバースト アクセスを隔離するには、ステップ 4 の最初のデータ ワードの制御ビットは、バースト アクセスに対し、Low (DCACHE\_FSL\_OUT\_Control = 0) になります。
3. DCACHE\_FSL\_OUT\_Full = 1 の場合、この信号が Low になるまで動作を停止します。
4. 保存するデータを DCACHE\_FSL\_OUT\_Data に書き込みます。制御ビットはバーストアクセスに対し、Low (DCACHE\_FSL\_OUT\_Control = 0) です。
5. キャッシュラインの後続ワードに対して手順 3 および 4 を繰り返します。

DXCL または DXCL2 プロトコルでは、シングル ワード書き込みで、キャッシュ コントローラは次のシーケンスを実行します。

1. DCACHE\_FSL\_OUT\_Full = 1 の場合、この信号が Low になるまで動作を停止します。



2. CACHE\_FSL\_OUT\_Data にミスしたアドレスを書き込み、制御ビットを High (DCACHE\_FSL\_OUT\_Control = 1) に設定して書き込みアクセスを示します。0b00=byte0、0b01=byte1、halfword0、0x10=byte2、0x11=byte3、halfword1 などのバイト イネーブルおよびハーフ ワード イネーブルのエンコードには、アドレスの最下位ビット (30:31) の 2 ビットが使用されます。手順 4 のデータの制御ビットによってハーフワードまたはバイト アクセスのいずれかが選択されます。
3. DCACHE\_FSL\_OUT\_Full = 1 の場合、この信号が Low になるまで動作を停止します。
4. 保存するデータを DCACHE\_FSL\_OUT\_Data に書き込みます。バイトおよびハーフワード アクセスでは、データはバイト レーンに書き込まれます。バイトは 4 つのバイト レーンすべてに、ハーフワードは両方のハーフワード レーンに書き込まれます。バースト アクセスから隔離するには、制御ビットは、ワードおよびハーフワード アクセスでは Low (DCACHE\_FSL\_OUT\_Control = 0)、バイト アクセスでは High である必要があります。ワード アクセスとハーフワード アクセスは、アドレスの最下位ビットで区別できます。0 の場合はワードで、1 の場合はハーフワードです。

## デバッグ インターフェイス

MicroBlaze のデバッグ インターフェイスは、ザイリンクスのマイクロプロセッサ デバッグ モジュール (MDM) IP コアと共に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 Xilinx Microprocessor Debugger (XMD) により制御します。MDM では、同時に複数の MicroBlaze プロセッサを制御できます。MicroBlaze のデバッグ信号は、DEBUG バスにグループ化されています。この信号を表 2-9 に示します。

表 2-9 : MicroBlaze デバッグ信号

信号名	説明	VHDL の型	方向
Dbg_Clk	MDM からの JTAG クロック	std_logic	入力
Dbg_TDI	MDM からの JTAG TDI	std_logic	入力
Dbg_TDO	MDM への JTAG TDO	std_logic	出力
Dbg_Reg_En	MDM からのデバッグレジスタイネーブル	std_logic	入力
Dbg_Shift <sup>1</sup>	MDM からの JTAG BSCAN シフト信号	std_logic	入力
Dbg_Capture	MDM からの JTAG BSCAN キャプチャ信号	std_logic	入力
Dbg_Update	MDM からの JTAG BSCAN アップデート信号	std_logic	入力
Debug_Rst <sup>1</sup>	MDM からのリセット信号 (アクティブ High)。1 Clk クロック サイクル以上保持する必要があります。	std_logic	入力

1. MicroBlaze v7.00 では Dbg\_Shift が追加され、Debug\_Rst は DEBUG バスに含まれるようにアップデートされています。



## トレース インターフェイス

MicroBlaze コアは、トレースの目的で多数の内部信号を送信します。この信号のインターフェイスは標準化されておらず、プロセッサの新しいリビジョンでは、同じ信号の選択または機能がサポートされない可能性があります。これらの信号にはカスタム ロジックを設計するのではなく、ザイリックスの解析用 IP を使用することをお勧めします。トレース信号は、TRACE バスにグループ化されています。MicroBlaze v7.00 向けに更新されているトレース信号を表 2-10 に、トレースの例外のタイプを表 2-11 に示します。未使用の例外はすべて予約済みです。

表 2-10 : MicroBlaze トレース信号

信号名	説明	VHDL の型	方向
Trace_Valid_Instr	プロセッサ実行段の有効な命令	std_logic	出力
Trace_Instruction <sup>1</sup>	命令コード	std_logic_vector (0 ~ 31)	出力
Trace_PC <sup>1</sup>	プログラム カウンタ	std_logic_vector (0 ~ 31)	出力
Trace_Reg_Write <sup>1</sup>	レジスタ ファイルに書き込む命令	std_logic	出力
Trace_Reg_Addr <sup>1</sup>	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 <sup>1</sup>	マシン ステータス レジスタ (MSR)	std_logic_vector (0 ~ 14) <sup>2</sup>	出力
Trace_PID_Reg <sup>1,2</sup>	プロセス識別レジスタ	std_logic_vector (0 ~ 7)	出力
Trace_New_Reg_Value <sup>1</sup>	デスティネーションレジスタのアップデート値	std_logic_vector (0 ~ 31)	出力
Trace_Exception_Taken <sup>1</sup>	例外の命令結果	std_logic	出力
Trace_Exception_Kind <sup>1,3</sup>	例外のタイプ。次の表を参照。	std_logic_vector (0 ~ 4) <sup>2</sup>	出力
Trace_Jump_Taken <sup>1</sup>	真であると評価された分岐命令	std_logic	出力
Trace_Delay_Slot <sup>1</sup>	分岐の遅延スロットに含まれる命令	std_logic	出力
Trace_Data_Access <sup>1</sup>	有効なデータ側アクセス	std_logic	出力
Trace_Data_Address <sup>1</sup>	データ側メモリ アクセスのアドレス	std_logic_vector (0 ~ 31)	出力
Trace_Data_Write_Value <sup>1</sup>	有効なデータ側メモリ 書き込みアクセス	std_logic_vector (0 ~ 31)	出力
Trace_Data_Byte_Enable <sup>1</sup>	有効なデータ側メモリ アクセスのバイト イネーブル	std_logic_vector (0 ~ 3)	出力
Trace_Data_Read <sup>1</sup>	データ側メモリ アクセスは読み出し	std_logic	出力
Trace_Data_Write <sup>1</sup>	データ側メモリ アクセスは書き込み	std_logic	出力

表 2-10 : MicroBlaze トレース信号 (続き)

信号名	説明	VHDL の型	方向
Trace_DCache_Req	データ メモリ アドレスはデータ キャッシュ範囲内	std_logic	出力
Trace_DCache_Hit	データ メモリ アドレスはデータ キャッシュに存在	std_logic	出力
Trace_ICache_Req	命令メモリ アドレスは命令キャッシュ範囲内	std_logic	出力
Trace_ICache_Hit	命令メモリ アドレスはメモリ キャッシュに存在	std_logic	出力
Trace_OF_PipeRun	デコード段のパイプライン予約	std_logic	出力
Trace_EX_PipeRun <sup>4</sup>	実行段のパイプライン予約	std_logic	出力
Trace_MEM_PipeRun <sup>4</sup>	メモリ段のパイプライン予約	std_logic	出力
Trace_MB_Halted <sup>2</sup>	パイプラインのデバッグによる停止	std_logic	出力

1. Trace\_Valid\_Instr = 1 のときのみ有効
2. MicroBlaze v7.00 での変更 : Trace\_MSR\_Reg に 4 ビット追加、Trace\_PID\_Reg の追加、Trace\_MB\_Halted の追加、Trace\_Exception Kind に 1 ビット追加
3. Trace\_Exception\_Taken = 1 のときのみ有効
4. エリア最適化との同時使用不可

表 2-11 : トレースの例外のタイプ

Trace_Exception_Kind [0:4]	説明
00000	高速シンプレックス リンクによる例外 <sup>1</sup>
00001	不整列データ アクセスによる例外
00010	不正な op コード による例外
00011	命令バス による例外
00100	データ バスによる例外
00101	0 での除算 による例外
00110	FPU での例外
00111	特権命令による例外 <sup>1</sup>
01010	割り込み
01011	マスク不可の外部ブレーク
01100	マスク可の外部ブレーク
10000	データ格納による例外 <sup>1</sup>
10001	命令格納による例外 <sup>1</sup>
10010	データ TLB ミスによる例外 <sup>1</sup>
10011	命令 TLB ミスによる例外 <sup>1</sup>

1. MicroBlaze v7.00 で追加

## MicroBlaze コアのコンフィギュレーション

MicroBlaze コアは、ユーザーが柔軟にコンフィギュレーションできるよう開発されているため、コストおよびパフォーマンスの要件を満たすよう設定できます。

コンフィギュレーションには、通常プロセッサの特定の機能をイネーブルにし、サイズを設定するパラメータを使用します。たとえば、命令キャッシュは `C_USE_ICACHE` パラメータを設定することによりイネーブルにし、命令キャッシュのサイズは `C_DCACHE_BYTE_SIZE`、キャッシュ可能なメモリ範囲は `C_ICACHE_BASEADDR` および `C_ICACHE_HIGHADDR` で指定します。

MicroBlaze v7.00 で使用可能なパラメータを表 2-12 に示します。これらのパラメータには以前のバージョンの MicroBlaze では認識されないものもありますが、コンフィギュレーションには後方互換性があります。

メモ：影付きの行は、パラメータが固定で変更できないことを示します。

表 2-12：MPD パラメータ

パラメータ名	機能/説明	許容値	デフォルト値	ツールによる自動割り当て	VHDL の型
C_FAMILY	ターゲット ファミリ	aspartan3 aspartan3a aspartan3adsp aspartan3e spartan3 spartan3a spartan3adsp spartan3an spartan3e spartan6 qrvirtex4 qvirtex4 virtex4 virtex5 virtex6	virtex5	あり	文字列
C_DATA_SIZE	データ サイズ	32	32	なし	整数
C_DINAMIC_BUS_SIZING		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	低い命令スループットでエリアを最適化するためのインプリメンテーションの選択	0、1	0		整数
C_INTERCONNECT	PLB インターコネクトの選択	0、1	1		整数
C_PVR	プロセッサ バージョン レジスタのモード選択	0、1、2	0		整数
C_PVR_USER1	プロセッサ バージョン レジスタ USER1 の定数	0x00 ~ 0xff	0x00		std_logic_vector (0 ~ 7)

表 2-12 : MPD パラメータ (続き)

パラメータ名	機能/説明	許容値	デフォルト値	ツールによる自動割り当て	VHDL の型
C_PVR_USER2	プロセッサ バージョン レジスタ USER2 の定数	0x00000000 ~ 0xffffffff	0x00000000		std_logic_vector (0 ~ 31)
C_RESET_MSR	MSR のリセット値	0x00、0x20、0x80、0xa0	0x00		std_logic_vector
C_INSTANCE	インスタンス名	任意のインスタンス名	microblaze	あり	文字列
C_D_PLB	データ側 PLB インターフェイス	0、1	0	あり	整数
C_D_OPB	データ側 OPB インターフェイス	0、1	1	あり	整数
C_D_LMB	データ側 LMB インターフェイス	0、1	1	あり	整数
C_I_PLB	命令側 PLB インターフェイス	0、1	0	あり	整数
C_I_OPB	命令側 OPB インターフェイス	0、1	1	あり	整数
C_I_LMB	命令側 LMB インターフェイス	0、1	1	あり	整数
C_USE_BARREL	バレル シフタを使用	0、1	0		整数
C_USE_DIV	ハードウェア除算器を使用	0、1	0		整数
C_USE_HW_MUL	ハードウェア乗算器を使用	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使用	0、1、2	0		整数
C_USE_MSR_INSTR	MSRSET および MSRCLR 命令をイネーブルにする	0、1	1		整数
C_USE_PCMP_INSTR	PCMPBF、PCMPEQ、PCMPNE 命令をイネーブルにする	0、1	1		整数
C_UNALIGNED_EXCEPTIONS	不整列データ アクセスに対する例外処理をイネーブルにする	0、1	0		整数
C_ILL_OPCODE_EXCEPTION	不正な op コードに対する例外処理をイネーブルにする	0、1	0		整数
C_IPLB_BUS_EXCEPTION	IPLB バス エラーに対する例外処理をイネーブルにする	0、1	0		整数
C_DPLB_BUS_EXCEPTION	DPLB バス エラーに対する例外処理をイネーブルにする	0、1	0		整数
C_IOPB_BUS_EXCEPTION	IOPB バス エラーに対する例外処理をイネーブルにする	0、1	0		整数
C_DOPB_BUS_EXCEPTION	DOPB バス エラーに対する例外処理をイネーブルにする	0、1	0		整数

表 2-12 : MPD パラメータ (続き)

パラメータ名	機能/説明	許容値	デフォルト値	ツールによる自動割り当て	VHDL の型
C_DIV_ZERO_EXCEPTION	0 での除算および除算オーバーフローに対する例外処理をイネーブルにする	0、1	0		整数
C_FPU_EXCEPTION	ハードウェア浮動小数点ユニットに関する例外処理をイネーブルにする	0、1	0		整数
C_OPCODE_0x0_ILLEGAL	op コード 0x0 を不正な命令として検出	0、1	0		整数
C_FSL_EXCEPTION	高速シンプレックス リンクに対する例外処理をイネーブルにする	0、1	0		整数
C_DEBUG_ENABLED	MDM デバッグ インターフェイス	0、1	0		整数
C_NUMBER_OF_PC_BRK	ハードウェアブレークポイント数	0 ~ 8	1		整数
C_NUMBER_OF_RD_ADDR_BRK	読み出しアドレスウォッチポイント数	0 ~ 4	0		整数
C_NUMBER_OF_WR_ADDR_BRK	書き込みアドレスウォッチポイント数	0 ~ 4	0		整数
C_INTERRUPT_IS_EDGE	レベル/エッジ割り込み	0、1	0	あり	整数
C_EDGE_IS_POSITIVE	立ち上がり / 立ち下がりエッジ割り込み	0、1	1	あり	整数
C_FSL_LINKS <sup>1</sup>	FSL インターフェイスの数	0 ~ 16	0	あり	整数
C_FSL_DATA_SIZE	FSL のデータ バス サイズ	32	32	なし	整数
C_USE_EXTENDED_FSL_INSTR	拡張 FSL 命令の使用をイネーブルにする	0、1	0		整数
C_ICACHE_BASEADDR	命令キャッシュのベース アドレス	0x00000000 ~ 0xFFFFFFFF	0x00000000		std_logic_vector
C_ICACHE_HIGHADDR	命令キャッシュのハイ アドレス	0x00000000 ~ 0xFFFFFFFF	0x3FFFFFFF		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_LEN	命令キャッシュ ラインの長さ	4、8	4		整数
C_ICACHE_ALWAYS_USED	命令キャッシュの CacheLink をすべてのメモリ アクセスに使用	0、1	1		整数

表 2-12 : MPD パラメータ (続き)

パラメータ名	機能/説明	許容値	デフォルト値	ツールによる自動割り当て	VHDL の型
C_ICACHE_INTERFACE	命令キャッシュ CacheLink インターフェイス プロトコル 0 = IXCL 1 = IXCL2	0、1	0	あり <sup>2</sup>	整数
C_ADDR_TAG_BITS	命令キャッシュのアドレス タグ	0 ~ 25	17	あり	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 <sup>3</sup>	8192		整数
C_ICACHE_USE_FSL	OPB の代わりに CacheLink を命令キャッシュに使用	1	1		整数
C_DCACHE_BASEADDR	データ キャッシュのベース アドレス	0x00000000 ~ 0xFFFFFFFF	0x00000000		std_logic_vector
C_DCACHE_HIGHADDR	データ キャッシュのハイ アドレス	0x00000000 ~ 0xFFFFFFFF	0x3FFFFFFF		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_LEN	データ キャッシュ ラインの長さ	4、8	4		整数
C_DCACHE_ALWAYS_USED	データ キャッシュの CacheLink をすべてのアクセスに使用	0、1	1		整数
C_DCACHE_INTERFACE	データ キャッシュ CacheLink インターフェイス プロトコル 0 = DXCL 1 = DXCL2	0、1	0	あり <sup>2</sup>	整数
C_DCACHE_USE_WRITEBACK	データ キャッシュ ライトバック 格納ポリシーを使用	0、1	0		整数
C_DCACHE_ADDR_TAG	データ キャッシュのアドレス タグ	0 ~ 25	17	あり	整数

表 2-12: MPD パラメータ (続き)

パラメータ名	機能/説明	許容値	デフォルト値	ツールによる自動割り当て	VHDL の型
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 <sup>3</sup>	8192		整数
C_DCACHE_USE_FSL	OPB の代わりに CacheLink をデータ キャッシュに使用	1	1		整数
C_DPLB_DWIDTH	データ側 PLB のデータ幅	32	32		整数
C_DPLB_NATIVE_DWIDTH	データ側 PLB 固有のデータ幅	32	32		整数
C_DPLB_BURST_EN	データ側 PLB バースト イネーブル	0	0		整数
C_DPLB_P2P	データ側 PLB Point-to-point	0、1	0		整数
C_IPLB_DWIDTH	命令側 PLB のデータ幅	32	32		整数
C_IPLB_NATIVE_DWIDTH	命令側 PLB 固有のデータ幅	32	32		整数
C_IPLB_BURST_EN	命令側 PLB バースト イネーブル	0	0		整数
C_IPLB_P2P	命令側 PLB Point-to-point	0、1	0		整数
C_USE_MMU <sup>4</sup>	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 <sup>4</sup>	データ シャドウ TLB のサイズ	1、2、4、8	4		整数
C_MMU_ITLB_SIZE <sup>4</sup>	命令シャドウ TLB のサイズ	1、2、4、8			整数
C_MMU_TLB_ACCESS <sup>4</sup>	メモリ管理用の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 アクセス	0、1、2、3	3		整数
C_MMU_ZONES <sup>4</sup>	メモリ保護ゾーンの数	0 ~ 16	16		整数
C_USE_INTERRUPT	割り込み処理を有効	0、1	0	あり	整数
C_USE_EXT_BRK	外部ブレーク処理を有効	0、1	0	あり	整数
C_USE_EXT_NM_BRK	外部マスク不可のブレークを有効	0、1	0	あり	整数

1. コプロセッサ ウィザードを使用している場合は、FSL リンクの数ツールにより割り当てられます。手動で IP を追加する場合は、パラメータも手動でアップデートしてください。

2. EDK ツールで割り当てられた値は手動で割り当てられた値に上書きされます。
3. すべてのアーキテクチャですべてのサイズが設定できるわけではありません。キャッシュでは、0 ～ 32 個の RAMB プリミティブが使用されます。キャッシュサイズが 2048 未満の場合は 0 です。
4. C\_AREA\_OPTIMIZED が 1 に設定されている場合は使用できません。



# MicroBlaze アプリケーション バイナリ インターフェイス (ABI)

この章では、ソフト プロセッサに対してアセンブリ言語でソフトウェアを開発する上で重要な MicroBlaze™ のアプリケーション 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 コンパイラはこの章に記述されている規則に従うので、アセンブリ プログラムで記述されるコードも、コンパイラで生成されるコードと互換性を持たせるため、同じ規則に従う必要があります。割り込みおよび例外処理についても簡単に説明します。

この章は次のセクションより構成されています。

- データ型
- レジスタの使用規則
- スタックの規則
- メモリ モデル
- 割り込みおよび例外処理

## データ型

MicroBlaze アセンブリ プログラムで使用されるデータ型を、表 3-1 に示します。data8、data16、および data32 のようなデータ型は、通常のバイト、ハーフバイト、およびワードで使用されます。

表 3-1 : MicroBlaze アセンブリ プログラムでのデータ型

MicroBlaze のデータ型 (アセンブリ プログラム用)	対応する ANSI C データ型	サイズ (バイト)
data8	char	1
data16	short	2
data32	int	4
data32	long int	4
data32	float	4
data32	enum	4
data16/data32	pointer <sup>a</sup>	2/4

a. グローバル ポインタでアクセス可能なスモール データ領域へのポインタは、data16 です。

## レジスタの使用規則

MicroBlaze に対するレジスタの使用規則を表 3-2 に示します。

表 3-2 : レジスタの使用規則

レジスタ	タイプ	対象	用途
R0	専用	ハードウェア	値 0
R1	専用	ソフトウェア	スタック ポインタ
R2	専用	ソフトウェア	読み出し専用のスモール データ領域アンカー
R3 ~ R4	揮発性	ソフトウェア	戻り値/一時保存
R5 ~ R10	揮発性	ソフトウェア	パラメータの引渡し/一時保存
R11 ~ R12	揮発性	ソフトウェア	一時保存
R13	専用	ソフトウェア	読み出し/書き込みのスモール データ領域アンカー
R14	専用	ハードウェア	割り込みの戻りアドレス
R15	専用	ソフトウェア	サブルーチンの戻りアドレス
R16	専用	ハードウェア	トラップ用の戻りアドレス (デバッグ)
R17	専用	ハードウェア ( サポートしている場合)、それ 以外の場合は ソフトウェア	例外用の戻りアドレス
R18	専用	ソフトウェア	アセンブラ用に予約済み/コンパイラ一時保存
R19	不揮発性	ソフトウェア	関数呼び出しを越えて保存する必要あり、 callee-save
R20	専用 または 不揮発性	ソフトウェア	PIC (Position Independent Code) では GOT (Global Offset Table) へのポインタの格納用に予 約済み。非 PIC コードでは不揮発性。関数呼び出し を越えて保存する必要あり、callee-save
R21 ~ R31	不揮発性	ソフトウェア	関数呼び出しを越えて保存する必要あり、 callee-save
RPC	特殊	ハードウェア	プログラム カウンタ
RMSR	特殊	ハードウェア	マシン ステータス レジスタ
REAR	特殊	ハードウェア	例外アドレス レジスタ
RESR	特殊	ハードウェア	例外ステータス レジスタ
RFSR	特殊	ハードウェア	浮動小数点ステータス レジスタ
RBTR	特殊	ハードウェア	分岐ターゲット レジスタ
REDR	特殊	ハードウェア	例外データ レジスタ
RPID	特殊	ハードウェア	プロセス識別レジスタ
RZPR	特殊	ハードウェア	ゾーン保護レジスタ
RTLBLO	特殊	ハードウェア	変換ルックアサイド バッファ Low レジスタ
RTLBHI	特殊	ハードウェア	変換ルックアサイド バッファ High レジスタ

表 3-2 : レジスタの使用規則

レジスタ	タイプ	対象	用途
RTL BX	特殊	ハードウェア	変換ルックアサイド バッファ インデックス レジスタ
RTL BSX	特殊	ハードウェア	変換ルックアサイド バッファ検索インデックス
RPVR0 ~ RPVR11	特殊	ハードウェア	プロセッサ バージョン レジスタ 0 ~ 11

MicroBlaze アーキテクチャでは、32 個の汎用レジスタ (GPR) が定義されます。これらのレジスタは、揮発性、不揮発性、および専用に分類されます。

- 揮発性レジスタ (caller-save) は、一時保存として使用され、関数呼び出しを越えて値を保持することはありません。R3 ~ R12 のレジスタは揮発性で、R3 および R4 は呼び出し関数の戻り値に使用されます。R5 ~ R10 のレジスタは、サブルーチン間のパラメータの引き渡しに使用されます。
- R19 ~ R31 のレジスタでは、関数呼び出しを越えて値が保持されるので、不揮発性のレジスタ (callee-save) と定義されています。呼び出される側の関数で、これらの不揮発性のレジスタが保存され、使用されます。これらは、通常プロローグ中にスタックに保存され、エピローグ中に再び読み込まれます。
- レジスタには、専用レジスタとして使用されるものもあり、プログラマではこれらのレジスタがほかの用途に使用されることはありません。
  - R14 ~ R17 のレジスタは、割り込み、サブルーチン、トラップ、例外という順序で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、これらの命令では、現在のプログラム カウンタ (PC) がレジスタ R15 に保存されます。
  - スモール データ領域ポインタは、16 ビットの即値があるメモリ ロケーションにアクセスするのに使用されます。これらの領域については、「メモリ モデル」で説明します。読み出し専用のスモール データ領域 (SDA) アンカー R2 (読み出し専用) は、リテラルのような定数にアクセスするのに使用されます。また SDA アンカー R13 (読み出し/書き込み) は、スモール データ読み出し/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 には、スタック ポインタの値が格納され、関数の入力と終了で更新されます。
  - レジスタ R18 は、アセンブラ操作の一時レジスタとして使用されます。
- MicroBlaze には、プログラム カウンタ (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)、浮動小数点ユニット 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、rtlbhi、rtlbx、rtlbsx)、プロセッサ バージョン レジスタ (rpvr0 ~ rpvr11) などの特殊用途レジスタがあります。これらのレジスタは直接レジスタ ファイルにマップされていないため、使用法は汎用レジスタとは異なります。特殊用途レジスタの値は、mts 命令を使用して汎用レジスタに、mfs 命令を使用して汎用レジスタから転送できます。

## スタックの規則

MicroBlaze で使用するスタックの規則を表 3-3 に示します。

表 3-3 の影付きの部分は、呼び出し側の関数のスタック フレームの部分を示し、影なしの部分は、呼び出される側のフレーム関数を示します。スタック フレームの ABI 規則では、パラメータの引渡

し、不揮発性レジスタの値の保持、および関数内のローカル変数に対するスペースの割り当てに関するプロトコルが定義されています。

ほかのサブルーチンの呼び出しを含む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では、使用するスタック フレーム領域を新規に作成する必要があります。スタック ポインタの値はプログラムの開始時に最大で、関数が呼び出されるたびに、各関数のスタック フレームで必要となるワード数分だけ減少します。呼び出し側の関数のスタック ポインタの値は、呼び出される側の関数のスタック ポインタの値に比べて大きいのが通常です。

表 3-3 : スタックの規則

ハイ アドレス	
	呼び出されたサブルーチンに対する関数パラメータ (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で必要となる引数の最大数)
古いスタック ポインタ	リンク レジスタ (R15)
	呼び出される側で保存されたレジスタ (R31.....R19) (オプション : 現在のプロシージャで使用するレジスタのみが保存される)
	現在のプロシージャに対するローカル変数 (オプション : プロシージャでローカルが定義されているときのみ表示)
	ファンクション パラメータ (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で必要となる引数の最大数)
新しいスタック ポインタ	リンク レジスタ
下位アドレス	

たとえば、Func1 が Func2 を呼び出し、Func2 が Func3 を呼び出すとします。それぞれのインスタンスでのスタック表現を図 3-1 に示します。Func1 からの Func2 の呼び出し後に、スタック ポインタ (SP) の値は減少します。この SP 値は、Func3 のスタック フレームに対応してさらに減少します。Func3 から戻ると、SP 値は Func 2 の元の値まで増加します。

図 3-1 を見ると、どのようにスタックが保持されるかがわかります。

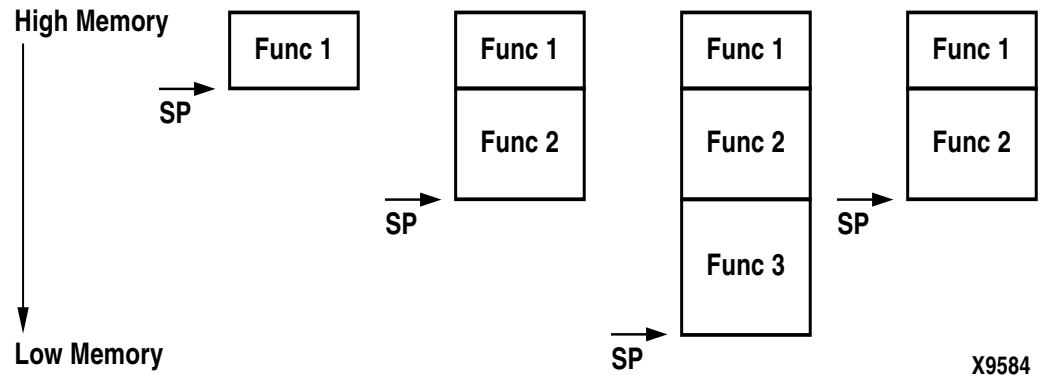


図 3-1 : スタック フレーム

## 呼び出し規則

呼び出し側の関数は、レジスタ (R5 ~ R10) または固有のスタック フレームを使用してパラメータを呼び出される側の関数に渡します。読み出される側では、呼び出し側のスタック領域を使用して渡されたパラメータを格納します。

図 3-1 を参照してください。Func2 のパラメータは、レジスタ R5 ~ R10 または Func1 に割り当てられたスタック フレームに格納されます。

## メモリ モデル

MicroBlaze のメモリ モデルは、データを、スモール データ領域、データ領域、共有の未初期化領域、リテラルまたは定数の 4 つの領域に分類します。

### スモール データ領域

サイズが小さく、グローバルに初期化された変数がこの領域に格納されます。このスモール データ領域に格納される変数のサイズを決定するしきい値は、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 ラインのオプションを使用して変更可能です。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』の「GNU コンパイラ ツール」の章を参照してください。スモール データ領域には、64KB のメモリが割り当てられます。このスモール データ領域には、読み出し/書き込み可能なスモール データ領域アンカー (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。この領域にサイズの小さい変数を割り当てると、グローバル変数にアクセスするコードに IMM 命令を追加する必要がなくなります。このエリアに含まれる変数は、絶対アドレスを使用してもアクセスできます。

### データ領域

値が比較的大きく初期化された変数は、データ領域に割り当てられます。この領域には、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/書き込み可能な SDA アンカー (R13) または絶対アドレスを使用してアクセスできます。

## 共有の未初期化領域

初期化されていないグローバル変数は、共有の領域に割り当てられます。この領域には、絶対アドレスまたは読み出し/書き込み可能なスモール データ領域アンカー (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、読み出し専用のスモール データ領域に配置され、読み出し専用のスモール データ領域アンカー (R2) を使用してアクセスされます。

コンパイラで、ベース ポインタとして動作するグローバル ポインタが生成されます。SDA アンカーの値は、最終のリンク段階でリンカにより決定されます。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』の「**MicroBlaze** リンカ 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、使用されるコマンド ライン オプションに応じ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については、『エンベデッド システム ツール リファレンス マニュアル』の「**GNU** コンパイラ ツール」の章を参照してください。

## 割り込みおよび例外処理

MicroBlaze では、表 3-4 に示すように、割り込みおよび例外処理に対して、特定のアドレス ロケーションが使用されます。これらのロケーションのコードは、適切なハンドラにジャンプするよう記述されています。

表 3-4 : 割り込みおよび例外処理

イベント	ハードウェアのジャンプ先	ソフトウェア ラベル
開始/リセット	0x0	_start
ユーザー例外	0x8	_exception_handler
割り込み	0x10	_interrupt_handler
ブレイク (ハードウェア/ ソフトウェア)	0x18	-
ハードウェア例外	0x20	_hw_exception_handler
将来の使用に予約	0x28 ~ 0x4F	-

これらのロケーションに記述されているコードを次に示します。-xl-mode-xmdstub コンパイラ オプションを使用せずにコンパイルされたプログラムの場合、mb-ld リンカで crt0.o 初期化ファイルが使用され、例外ハンドラでの適切なアドレスが設定されます。

-xl-mode-xmdstub コンパイラ オプションを使用してコンパイルされたプログラムの場合、crt1.o 初期化ファイルが出力プログラムにリンクされます。このプログラムは、アドレス ロケーション 0x0 に XMDStub が含まれている場合のみ実行できます。ランタイムには、例外および割り込みハンドラのアドレスに応じて、crt1.o の初期化コードにより適切な命令がロケーション 0x8 ~ 0x14 に書き込まれます。

例外および割り込みハンドラに制御を渡すコードを次に示します。

```

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
0x20:  imm      high bits of address (HW exception handler)
0x24:  bri      _hw_exception_handler

```

MicroBlaze では、32 ビットで指定可能なアドレス ロケーションであれば、どこにでも例外および割り込みハンドラ ルーチンを配置できます。ユーザー例外ハンドラのコードはラベル \_\_handler で、ハードウェア例外ハンドラのコードはラベル \_hw\_\_handler で、割り込みハンドラのコードはラベル \_interrupt\_handler で始まります。

現段階の MicroBlaze システムでは、割り込みおよび例外処理に対してユーザーが変更可能なダミー ルーチンがあります。これらのダミー ルーチンの代わりに、独自の割り込みハンドラおよび例外ハンドラをリンクするには、割り込みハンドラのコードを属性 interrupt\_handler で定義する必要があります。割り込みハンドラ属性の使用法および構文の詳細は、『エンベデッド システム ツール リファレンス マニュアル』の「GNU コンパイラ ツール」の章を参照してください。

XMD (Xilinx Microprocessor Debug) ツールでソフトウェアのブレイクポイントが使用される場合は、ハードウェアおよびソフトウェアのブレイク アドレス ロケーションは、ソフトウェアのブレイクポイントの処理用に予約されています。





## MicroBlaze 命令セット アーキテクチャ

この章では、MicroBlaze™ の命令セット アーキテクチャについて詳しく説明します。次のセクションより構成されています。

- [表記法](#)
- [フォーマット](#)
- [命令](#)

### 表記法

この章で使用するシンボルの定義を [表 4-1](#) に示します。

表 4-1：シンボル表記

シンボル	意味
+	加算
-	減算
×	乗算
^	ビット単位の論理積 (AND)
∨	ビット単位の論理和 (OR)
⊕	ビット単位の排他論理和 (XOR)
x	$x$ のビット単位の論理補数
←	代入
>>	右方向へシフト
<<	左方向へシフト
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 $i \sim j$
=	等価
≠	非等価
>	大なり
>=	以上
<	小なり
<=	以下

表 4-1 : シンボル表記(続き)

シンボル	意味
$\text{sext}(x)$	符号拡張 $x$
$\text{Mem}(x)$	アドレス $x$ のメモリ ロケーション
$\text{FSL}x$	FSL インターフェイス $x$
$\text{LSW}(x)$	$x$ の最下位ワード
$\text{isDnz}(x)$	浮動小数点: $x$ が非正規化されている場合に真
$\text{isInfinite}(x)$	浮動小数点: $x$ が $+\infty$ または $-\infty$ の場合に真
$\text{isPosInfinite}(x)$	浮動小数点: $x$ が $+\infty$ の場合に真
$\text{isNegInfinite}(x)$	浮動小数点: $x$ が $-\infty$ の場合に真
$\text{isNaN}(x)$	浮動小数点: $x$ が quiet または signalling NaN の場合に真
$\text{isZero}(x)$	浮動小数点: $x$ が $+0$ または $-0$ の場合に真
$\text{isQuietNaN}(x)$	浮動小数点: $x$ が quiet NaN の場合に真
$\text{isSigNaN}(x)$	浮動小数点: $x$ が signaling NaN の場合に真
$\text{signZero}(x)$	浮動小数点: $x > 0$ の場合は $+0$ , $x < 0$ の場合は $-0$ を返す
$\text{signInfinite}(x)$	浮動小数点: $x > 0$ の場合は $+\infty$ , $x < 0$ の場合は $-\infty$ を返す

## フォーマット

MicroBlaze では、タイプ A とタイプ B という 2 つの命令フォーマットが使用されます。

### タイプ A

タイプ A は、レジスタ間の命令に使用されます。このタイプには、opcode、デスティネーションレジスタ 1 個、ソース レジスタ 2 個が含まれます。

opcode	デスティネーション レジスタ	ソース レジスタ A	ソース レジスタ B	0	0	0	0	0	0	0	0	0	0	0	0
0	6	11	16	21											31

### タイプ B

タイプ B は、レジスタと即値間の命令に使用されます。このタイプには、opcode、デスティネーションレジスタ 1 個、ソース レジスタ 1 個、および 16 ビットの即値が含まれます。

opcode	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

## 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掲載されており、各命令に対して簡略コード、エンコード、説明、命令セマンティックスの擬似コード、およびその命令で変更されるレジスタの一覧が示されます。

add

## Arithmetic Add

<b>add</b>	rD, rA, rB	加算
<b>addc</b>	rD, rA, rB	キャリイ付き加算
<b>addk</b>	rD, rA, rB	加算 (キャリイを保持)
<b>addkc</b>	rD, rA, rB	キャリイ付き加算 (キャリイを保持)

0	0	0	K	C	0	rD	rA	rB	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 の内容の合計をレジスタ rD に格納します。

**addk** に対しては、命令のビット 3 (図で **K** と表記) が 1 に設定されます。**addc** に対しては、命令のビット 4 (図で **C** と表記) が 1 に設定されます。**addkc** に対しては、両方のビットが 1 に設定されます。

ビット 3 が 1 の場合 (addk、addkc)、命令の実行結果にかかわらず、キャリー フラグに既存の値が保持されます。ビット 3 が 0 の場合 (add、addc)、キャリー フラグが命令の実行結果に応じて変更されます。

ビット 4 が 1 の場合 (addc、addkc)、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 0 の場合 (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 (通常の加算を実行)。

擬似コード

```

if C = 0 then
    (rD)  $\leftarrow$  (rA) + (rB)
else
    (rD)  $\leftarrow$  (rA) + (rB) + MSR[C]
if K = 0 then
    MSR[C]  $\leftarrow$  CarryOut

```

## 変更されるレジスタ

- rD
- MSR[C]

## レイテンシ

1 クロック サイクル

## メモ

命令 opcode 内の C ビットは、MSR のキャリービットとは異なります。

「add r0, r0, r0」 (= 0x00000000) 命令は、コンパイラで使用されることはなく、通常は初期化されていないメモリを示します。不正な命令の例外を使用している場合、MicroBlaze のパラメータを C\_OPCODE 0x0\_ILLEGAL=1 に設定すると、これらの命令をトラップできます。

## addi

## Arithmetic Add Immediate

<b>addi</b>	rD, rA, IMM	即値加算
<b>addic</b>	rD, rA, IMM	キャリー付きの即値加算
<b>addik</b>	rD, rA, IMM	即値加算 (キャリーを保持)
<b>addikc</b>	rD, rA, IMM	キャリー付きの即値加算 (キャリーを保持)

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 32 ビットに符号拡張された **IMM** フィールドの値の合計を、レジスタ **rD** に格納します。**addik** に対しては、命令のビット 3 (図で **K** と表記) が 1 に設定されます。**addc** に対しては、命令のビット 4 (図で **C** と表記) が 1 に設定されます。**addikc** に対しては、両方のビットが 1 に設定されます。

ビット 3 が 1 の場合 (**addik**、**addikc**)、命令の実行結果にかかわらず、キャリー フラグに既存の値が保持されます。ビット 3 が 0 の場合 (**addi**、**addic**)、キャリー フラグが命令の実行結果に応じて変更されます。

ビット 4 が 1 の場合 (**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 0 の場合 (**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 (通常の加算を実行)。

## 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 クロック サイクル

## メモ

命令 **opcode** 内の **C** ビットは、**MSR** 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** フィールドの値を取り込み、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。タイプ **B** 命令の前に **imm** 命令を使用すると、32 ビットの即値を使用できます。32 ビットの即値の使用に関する詳細は、166 ページの「**imm**」を参照してください。

**and**

Logical 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						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 に格納します。

**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 クロック サイクル

## andi

Logical AND with Immediate

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 32 ビットに符号拡張された **IMM** フィールドの値を AND 演算し、その結果をレジスタ **rD** に格納します。

## 擬似コード

$$(rD) \leftarrow (rA) \wedge \text{sext}(IMM)$$

## 変更されるレジスタ

- rD

## レイテンシ

1 クロック サイクル

## メモ

デフォルトでは、タイプ **B** 命令は 16 ビットの **IMM** フィールドの値を取り込み、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。タイプ **B** 命令の前に **imm** 命令を使用すると、32 ビットの即値を使用できます。32 ビットの即値の使用に関する詳細は、[166 ページの「imm」](#)を参照してください。

## andn

Logical 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	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 の内容の論理補数を AND 演算し、その結果をレジスタ rD に格納します。

### 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 クロック サイクル



## andni Logical AND NOT with Immediate

**andni**      rD, rA, IMM

1	0	1	0	1	1	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** フィールドの値の論理補数を AND 演算し、その結果をレジスタ **rD** に格納します。

### 擬似コード

$$(rD) \leftarrow (rA) \wedge (\text{sext}(IMM))$$

### 変更されるレジスタ

- rD

### レイテンシ

1 クロック サイクル

### メモ

デフォルトでは、タイプ **B** 命令は 16 ビットの **IMM** フィールドの値を取り込み、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。タイプ **B** 命令の前に **imm** 命令を使用すると、32 ビットの即値を使用できます。32 ビットの即値の使用に関する詳細は、[166 ページの「imm」](#)を参照してください。

## beq

### Branch if Equal

<b>beq</b>	rA, rB	0 の場合に分岐
<b>beqd</b>	rA, rB	0 の場合に分岐 (遅延スロットを使用)

1	0	0	1	1	1	D	0	0	0	0	rA	rB	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0						6					11			16					21								31

### 説明

rA が 0 の場合、rB のオフセット値に含められている命令に分岐します。分岐先は、アドレス PC + rB の命令です。

beqd に対しては、D ビットが 1 に設定されます。この D ビットは、分岐遅延スロットを使用するかどうかを指定します。D ビットが 1 の場合、遅延スロットが使用され、分岐後の命令 (分岐遅延スロットにある命令) を分岐先の命令を実行する前に完了させることができます。D ビットが 0 の場合、遅延スロットは使用されず、分岐後に分岐先の命令が実行されます。

### 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 クロック サイクル (分岐が不成立の場合)
- 2 クロック サイクル (分岐が成立し、D ビットが 1 の場合)
- 3 クロック サイクル (分岐が成立し、D ビットが 0 の場合)

### メモ

遅延スロットは、imm、分岐、ブレイク命令で使用できません。割り込みおよび外部ハードウェアブレイクは、遅延スロットの分岐が完了するまで延期されます。

## beqi

## Branch Immediate if Equal

<b>beqi</b>	rA, IMM	0 の場合に即値に分岐
<b>beqid</b>	rA, IMM	0 の場合に即値に分岐 (遅延スロットを使用)

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 のオフセット値に含められている命令に分岐します。分岐先は、アドレス PC + IMM の命令です。

beqid に対しては、D ビットが 1 に設定されます。この D ビットは、分岐遅延スロットを使用するかどうかを指定します。D ビットが 1 の場合、遅延スロットが使用され、分岐後の命令 (分岐遅延スロットにある命令) を分岐先の命令を実行する前に完了させることができます。D ビットが 0 の場合、遅延スロットは使用されず、分岐後に分岐先の命令が実行されます。

## 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 クロック サイクル (分岐が不成立の場合)
- 2 クロック サイクル (分岐が成立し、D ビットが 1 の場合)
- 3 クロック サイクル (分岐が成立し、D ビットが 0 の場合)

## メモ

デフォルトでは、タイプ B 命令は 16 ビットの IMM フィールドの値を取り込み、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。タイプ B 命令の前に imm 命令を使用すると、32 ビットの即値を使用できます。32 ビットの即値の使用に関する詳細は、[166 ページの「imm」](#)を参照してください。

遅延スロットは、imm、分岐、ブレイク命令で使用できません。割り込みおよび外部ハードウェアブレイクは、遅延スロットの分岐が完了するまで延期されます。

## bge

### Branch if Greater or Equal

<b>bge</b>	rA, rB	0 以上の場合に分岐
<b>bged</b>	rA, rB	0 以上の場合に分岐 (分岐遅延スロットを使用)

1	0	0	1	1	1	D	0	1	0	1	rA	rB	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
0						6					11		16									21							31

### 説明

rA が 0 以上の場合、rB のオフセット値に含められている命令に分岐します。分岐先は、アドレス PC + rB の命令です。

bged に対しては、D ビットが 1 に設定されます。この D ビットは、分岐遅延スロットを使用するかどうかを指定します。D ビットが 1 の場合、遅延スロットが使用され、分岐後の命令 (分岐遅延スロットにある命令) を分岐先の命令を実行する前に完了させることができます。D ビットが 0 の場合、遅延スロットは使用されず、分岐後に分岐先の命令が実行されます。

### 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 クロック サイクル (分岐が不成立の場合)
- 2 クロック サイクル (分岐が成立し、D ビットが 1 の場合)
- 3 クロック サイクル (分岐が成立し、D ビットが 0 の場合)

### メモ

遅延スロットは、imm、分岐、ブレーク命令で使用できません。割り込みおよび外部ハードウェアブレークは、遅延スロットの分岐が完了するまで延期されます。

## bgei

## Branch Immediate if Greater or Equal

<b>bgei</b>	rA, IMM	0 以上の場合に即値に分岐
<b>bgeid</b>	rA, IMM	0 以上の場合に即値に分岐 (分岐遅延スロットを使用)

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 のオフセット値に含められている命令に分岐します。分岐先は、アドレス  $PC + IMM$  の命令です。

bgeid に対しては、D ビットが 1 に設定されます。この D ビットは、分岐遅延スロットを使用するかどうかを指定します。D ビットが 1 の場合、遅延スロットが使用され、分岐後の命令 (分岐遅延スロットにある命令) を分岐先の命令を実行する前に完了させることができます。D ビットが 0 の場合、遅延スロットは使用されず、分岐後に分岐先の命令が実行されます。

## 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 クロック サイクル (分岐不成立の場合)
- 2 クロック サイクル (分岐が成立し、D ビットが 1 の場合)
- 3 クロック サイクル (分岐が成立し、D ビットが 0 の場合)

## メモ

デフォルトでは、タイプ B 命令は 16 ビットの IMM フィールドの値を取り込み、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。タイプ B 命令の前に imm 命令を使用すると、32 ビットの即値を使用できます。32 ビットの即値の使用に関する詳細は、[166 ページの「imm」](#)を参照してください。

遅延スロットは、imm、分岐、ブレイク命令で使用できません。割り込みおよび外部ハードウェアブレイクは、遅延スロットの分岐が完了するまで延期されます。

## bgt

Branch if Greater Than

<b>bgt</b>	rA, rB	0 より大きい場合に分岐
<b>bgtD</b>	rA, rB	0 より大きい場合に分岐 (分岐遅延スロットを使用)

1	0	0	1	1	1	D	0	1	0	0	rA	rB	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
0						6					11		16									21							31

## 説明

rA が 0 より大きい場合、rB のオフセット値に含められている命令に分岐します。分岐先は、アドレス PC + rB の命令です。

bgtD に対しては、D ビットが 1 に設定されます。この D ビットは、分岐遅延スロットを使用するかどうかを指定します。D ビットが 1 の場合、遅延スロットが使用され、分岐後の命令 (分岐遅延スロットにある命令) を分岐先の命令を実行する前に完了させることができます。D ビットが 0 の場合、遅延スロットは使用されず、分岐後に分岐先の命令が実行されます。

## 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 クロック サイクル (分岐不成立の場合)
- 2 クロック サイクル (分岐が成立し、D ビットが設定されている場合)
- 3 クロック サイクル (分岐が成立し、D ビットが設定されていない場合)

## メモ

遅延スロットは、imm、分岐、ブレーク命令で使用できません。割り込みおよび外部ハードウェアブレークは、遅延スロットの分岐が完了するまで延期されます。

## bgti

## Branch Immediate if Greater Than

<b>bgti</b>	rA, IMM	0 より大きい場合に即値に分岐
<b>bgtid</b>	rA, IMM	0 より大きい場合に分岐 (分岐遅延スロットを使用)

1	0	1	1	1	1	D	0	1	0	0	rA	IMM	
0						6					11	16	31

## 説明

rA が 0 より大きい場合、IMM のオフセット値に含まれている命令に分岐します。分岐先は、アドレス  $PC + IMM$  の命令です。

bgtid に対しては、D ビットが 1 に設定されます。この D ビットは、分岐遅延スロットを使用するかどうかを指定します。D ビットが 1 の場合、遅延スロットが使用され、分岐後の命令 (分岐遅延スロットにある命令) を分岐先の命令を実行する前に完了させることができます。D ビットが 0 の場合、遅延スロットは使用されず、分岐後に分岐先の命令が実行されます。

## 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 クロック サイクル (分岐不成立の場合)
- 2 クロック サイクル (分岐が成立し、D ビットが 1 の場合)
- 3 クロック サイクル (分岐が成立し、D ビットが 0 の場合)

## メモ

デフォルトでは、タイプ B 命令は 16 ビットの IMM フィールドの値を取り込み、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。タイプ B 命令の前に imm 命令を使用すると、32 ビットの即値を使用できます。32 ビットの即値の使用に関する詳細は、[166 ページの「imm」](#)を参照してください。

遅延スロットは、imm、分岐、ブレイク命令で使用できません。割り込みおよび外部ハードウェアブレイクは、遅延スロットの分岐が完了するまで延期されます。

## ble

Branch if Less or Equal

**ble**                    rA, rB                    0 以下の場合に分岐  
**bled**                   rA, rB                    0 以下の場合に分岐 (分岐遅延スロットを使用)

1	0	0	1	1	1	D	0	0	1	1	rA				rB				0	0	0	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 のオフセット値に含まれている命令に分岐します。分岐先は、アドレス PC + rB の命令です。

bled に対しては、D ビットが 1 に設定されます。この D ビットは、分岐遅延スロットを使用するかどうかを指定します。D ビットが 1 の場合、遅延スロットが使用され、分岐後の命令 (分岐遅延スロットにある命令) を分岐先の命令を実行する前に完了させることができます。D ビットが 0 の場合、遅延スロットは使用されず、分岐後に分岐先の命令が実行されます。

### 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 クロック サイクル (分岐不成立の場合)
- 2 クロック サイクル (分岐が成立し、D ビットが 1 の場合)
- 3 クロック サイクル (分岐が成立し、D ビットが 0 の場合)

### メモ

遅延スロットは、imm、分岐、ブレーク命令で使用できません。割り込みおよび外部ハードウェアブレークは、遅延スロットの分岐が完了するまで延期されます。



## blei

## Branch Immediate if Less or Equal

<b>blei</b>	rA, IMM	0 以下の場合に即値に分岐
<b>bleid</b>	rA, IMM	0 以下の場合に即値に分岐 (分岐遅延スロットを使用)

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 のオフセット値に含められている命令に分岐します。分岐先は、アドレス  $PC + IMM$  の命令です。

bleid に対しては、D ビットが 1 に設定されます。この D ビットは、分岐遅延スロットを使用するかどうかを指定します。D ビットが 1 の場合、遅延スロットが使用され、分岐後の命令 (分岐遅延スロットにある命令) を分岐先の命令を実行する前に完了させることができます。D ビットが 0 の場合、遅延スロットは使用されず、分岐後に分岐先の命令が実行されます。

## 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 クロック サイクル (分岐不成立の場合)
- 2 クロック サイクル (分岐が成立し、D ビットが 1 の場合)
- 3 クロック サイクル (分岐が成立し、D ビットが 0 の場合)

## メモ

デフォルトでは、タイプ B 命令は 16 ビットの IMM フィールドの値を取り込み、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。タイプ B 命令の前に imm 命令を使用すると、32 ビットの即値を使用できます。32 ビットの即値の使用に関する詳細は、[166 ページの「imm」](#)を参照してください。

遅延スロットは、imm、分岐、ブレイク命令で使用できません。割り込みおよび外部ハードウェアブレイクは、遅延スロットの分岐が完了するまで延期されます。

## blt

Branch if Less Than

<b>blt</b>	rA, rB	0 より小さい場合に分岐
<b>bltd</b>	rA, rB	0 より小さい場合に分岐 (分岐遅延スロットを使用)

1	0	0	1	1	1	D	0	0	1	0	rA				rB				0	0	0	0	0	0	0	0	0	0		
0						6					11					16					21					31				

### 説明

rA が 0 より小さい場合、rB のオフセット値に含められている命令に分岐します。分岐先は、アドレス  $PC + rB$  の命令です。

bltd に対しては、D ビットが 1 に設定されます。この D ビットは、分岐遅延スロットを使用するかどうかを指定します。D ビットが 1 の場合、遅延スロットが使用され、分岐後の命令 (分岐遅延スロットにある命令) を分岐先の命令を実行する前に完了させることができます。D ビットが 0 の場合、遅延スロットは使用されず、分岐後に分岐先の命令が実行されます。

### 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 クロック サイクル (分岐不成立の場合)
- 2 クロック サイクル (分岐が成立し、D ビットが 1 の場合)
- 3 クロック サイクル (分岐が成立し、D ビットが 0 の場合)

### メモ

遅延スロットは、imm、分岐、ブレーク命令で使用できません。割り込みおよび外部ハードウェアブレークは、遅延スロットの分岐が完了するまで延期されます。

**bti****Branch Immediate if Less Than**

<b>bti</b>	<b>rA, IMM</b>	0 より小さい場合に即値に分岐
<b>btiid</b>	<b>rA, IMM</b>	0 より小さい場合に即値に分岐 (分岐遅延スロットを使用)

1	0	1	1	1	1	D	0	0	1	0	rA	IMM	
0						6					11	16	31

**説明**

**rA** が 0 未満の場合、**IMM** のオフセット値に含められている命令に分岐します。分岐先は、アドレス  $PC + IMM$  の命令です。

**btiid** に対しては、**D** ビットが 1 に設定されます。この **D** ビットは、分岐遅延スロットを使用するかどうかを指定します。**D** ビットが 1 の場合、遅延スロットが使用され、分岐後の命令 (分岐遅延スロットにある命令) を分岐先の命令を実行する前に完了させることができます。**D** ビットが 0 の場合、遅延スロットは使用されず、分岐後に分岐先の命令が実行されます。

**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 クロック サイクル (分岐不成立の場合)
- 2 クロック サイクル (分岐が成立し、**D** ビットが 1 の場合)
- 3 クロック サイクル (分岐が成立し、**D** ビットが 0 の場合)

**メモ**

デフォルトでは、タイプ B 命令は 16 ビットの **IMM** フィールドの値を取り込み、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。タイプ B 命令の前に **imm** 命令を使用すると、32 ビットの即値を使用できます。32 ビットの即値の使用に関する詳細は、166 ページの「**imm**」を参照してください。

遅延スロットは、**imm**、分岐、ブレイク命令で使用できません。割り込みおよび外部ハードウェアブレイクは、遅延スロットの分岐が完了するまで延期されます。

## bne

### Branch if Not Equal

<b>bne</b>	rA, rB	0 でない場合に分岐
<b>bned</b>	rA, rB	0 でない場合に分岐 (分岐遅延スロットを使用)

1	0	0	1	1	1	D	0	0	0	1	rA	rB	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
0						6					11		16								21								31

### 説明

rA が 0 ではない場合、rB のオフセット値に含められている命令に分岐します。分岐先は、アドレス  $PC + rB$  の命令です。

bned に対しては、D ビットが 1 に設定されます。この D ビットは、分岐遅延スロットを使用するかどうかを指定します。D ビットが 1 の場合、遅延スロットが使用され、分岐後の命令 (分岐遅延スロットにある命令) を分岐先の命令を実行する前に完了させることができます。D ビットが 0 の場合、遅延スロットは使用されず、分岐後に分岐先の命令が実行されます。

### 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 クロック サイクル (分岐不成立の場合)
- 2 クロック サイクル (分岐が成立し、D ビットが 1 の場合)
- 3 クロック サイクル (分岐が成立し、D ビットが 0 の場合)

### メモ

遅延スロットは、imm、分岐、ブレイク命令で使用できません。割り込みおよび外部ハードウェアブレイクは、遅延スロットの分岐が完了するまで延期されます。

## bnei

## Branch Immediate if Not Equal

<b>bnei</b>	rA, IMM	0 でない場合に即値に分岐
<b>bneid</b>	rA, IMM	0 でない場合に即値に分岐 (分岐遅延スロットを使用)

1	0	1	1	1	1	D	0	0	0	1	rA	IMM	
0						6					11	16	31

## 説明

rA が 0 ではない場合、IMM のオフセット値に含められている命令に分岐します。分岐先は、アドレス  $PC + IMM$  の命令です。

bneid に対しては、D ビットが 1 に設定されます。この D ビットは、分岐遅延スロットを使用するかどうかを指定します。D ビットが 1 の場合、遅延スロットが使用され、分岐後の命令 (分岐遅延スロットにある命令) を分岐先の命令を実行する前に完了させることができます。D ビットが 0 の場合、遅延スロットは使用されず、分岐後に分岐先の命令が実行されます。

## 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 クロック サイクル (分岐不成立の場合)
- 2 クロック サイクル (分岐が成立し、D ビットが 1 の場合)
- 3 クロック サイクル (分岐が成立し、D ビットが 0 の場合)

## メモ

デフォルトでは、タイプ B 命令は 16 ビットの IMM フィールドの値を取り込み、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。タイプ B 命令の前に imm 命令を使用すると、32 ビットの即値を使用できます。32 ビットの即値の使用に関する詳細は、166 ページの「imm」を参照してください。

遅延スロットは、imm、分岐、ブレイク命令で使用できません。割り込みおよび外部ハードウェアブレイクは、遅延スロットの分岐が完了するまで延期されます。

## br

## Unconditional Branch

<b>br</b>	rB	分岐
<b>bra</b>	rB	絶対値に分岐
<b>brd</b>	rB	分岐 (分岐遅延スロットを使用)
<b>brad</b>	rB	絶対値に分岐 (分岐遅延スロットを使用)
<b>brld</b>	rD, rB	分岐およびリンク (分岐遅延スロットを使用)
<b>brald</b>	rD, rB	絶対値に分岐およびリンク (分岐遅延スロットを使用)

1	0	0	1	1	0		rD		D	A	L	0	0		rB		0	0	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含まれる命令に分岐します。

brld および brald に対しては、L ビットが 1 に設定されます。L ビットが 1 の場合、リンクが実行され、PC の現在の値が rD に格納されます。

bra、brad、および brald に対しては、A ビットが 1 に設定されます。A ビットが 1 の場合は、rB の値を絶対値とみなし、その値に分岐します。A ビットが 0 の場合は、相対分岐となり、分岐先は PC + rB となります。

bra、brad、brld、および brald に対しては、D ビットが 1 に設定されます。この D ビットは、分岐遅延スロットを使用するかどうかを指定します。D ビットが 1 の場合、遅延スロットが使用され、分岐後の命令 (分岐遅延スロットにある命令) を分岐先の命令を実行する前に完了させることができます。

D ビットが 0 の場合、遅延スロットは使用されず、分岐後に分岐先の命令が実行されます。

## 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 クロック サイクル (D ビットが 1 の場合)

3 クロック サイクル (D ビットが 0 の場合)

## メモ

命令 **brl** および **bral** は、使用できません。遅延スロットは、**imm**、分岐、ブレーク命令で使用できません。割り込みおよび外部ハードウェアブレークは、遅延スロットの分岐が完了するまで延期されます。

## bri

## Unconditional Branch Immediate

<b>bri</b>	IMM	即値に分岐
<b>brai</b>	IMM	即値の絶対値に分岐
<b>brid</b>	IMM	即値に分岐 (分岐遅延スロットを使用)
<b>braid</b>	IMM	即値の絶対値に分岐 (分岐遅延スロットを使用)
<b>brlid</b>	rD, IMM	即値に分岐およびリンク (分岐遅延スロットを使用)
<b>bralid</b>	rD, IMM	即値の絶対値に分岐およびリンク (分岐遅延スロットを使用)

1	0	1	1	1	0	rD	D	A	L	0	0	IMM					
0						6						11					31

## 説明

32 ビットに符号拡張された **IMM** で指定されるアドレスに含まれている命令に分岐します。

**brlid** および **bralid** に対しては、**L** ビットが 1 に設定されます。**L** ビットが 1 の場合、リンクが実行され、PC の現在の値が **rD** に格納されます。

**brai**、**braid**、および **bralid** に対しては、**A** ビットが 1 に設定されます。**A** ビットが 1 の場合は、**IMM** の値を絶対値とみなし、その値に分岐します。**A** ビットが 0 の場合は、相対分岐となり、分岐先は **PC + IMM** となります。

**brid**、**braid**、**brlid**、および **bralid** に対しては、**D** ビットが 1 に設定されます。この **D** ビットは、分岐遅延スロットを使用するかどうかを指定します。**D** ビットが 1 の場合、遅延スロットが使用され、分岐後の命令 (分岐遅延スロットにある命令) を分岐先の命令を実行する前に完了させることができます。**D** ビットが 0 の場合、遅延スロットは使用されず、分岐後に分岐先の命令が実行されます。

MicroBlaze で MMU が使用され (**C\_USE\_MMU** >= 1)、**brki rD, 0x18** を使用してユーザー ベクタ例外が処理される場合は、**MSR** のユーザー モード ビットおよび仮想モード ビットが 0 になります。

## 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 = 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

2 クロック サイクル (D ビットが 1 の場合)

3 クロック サイクル (D ビットが 0 の場合)

## メモ

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** フィールドの値を取り込み、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。タイプ **B** 命令の前に **imm** 命令を使用すると、32 ビットの即値を使用できます。32 ビットの即値の使用に関する詳細は、[166 ページの「imm」](#)を参照してください。

遅延スロットは、**imm**、分岐、ブレーク命令で使用できません。割り込みおよび外部ハードウェアブレークは、遅延スロットの分岐が完了するまで延期されます。



## brki

Break Immediate

brki                    rD, IMM

1	0	1	1	1	0		rD		0	1	1	0	0		IMM	
0						6			11					16		31

## 説明

32 ビットに符号拡張された IMM のアドレス値に格納されている命令に分岐、リンクします。PC の現在の値が rD に格納されます。MSR の BIP フラグが 1 に設定され、予約ビットがクリアになります。

MicroBlaze で MMU が使用される場合 (C\_USE\_MMU >= 1)、brki rD, 0x8 または brki rD, 0x18 を使用してソフトウェア ブレークが実行される場合を除いてこの命令は特権命令になり、ユーザー モード (MSR[UM] = 1) で使用すると特権命令例外が発生します。

MicroBlaze で MMU が使用され (C\_USE\_MMU >= 1)、brki rD, 0x8 を使用してユーザー ベクタ例外が処理される場合は、MSR のユーザー モード ビットおよび仮想モード ビットが 0 になります。

## 擬似コード

```

if MSR[UM] = 1 and IMM ≠ 0x8 and IMM ≠ 0x18 then
    ESR[EC] ← 00111
else
    (rD) ← PC
    PC ← sext(IMM)
    MSR[BIP] ← 1
    Reservation ← 0
    if IMM = 0x8 or IMM = 0x18 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 (例外が発生した場合は変更なし)
- PC
- MSR[BIP]、MSR[UM]、MSR[VM]
- ESR[EC] (特権命令例外が発生した場合)

## レイテンシ

3 クロック サイクル

## メモ

デフォルトでは、タイプ B 命令は 16 ビットの IMM フィールドの値を取り込み、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。タイプ B 命令の前に imm 命令を使用すると、32 ビットの即値を使用できます。32 ビットの即値の使用に関する詳細は、166 ページの「imm」を参照してください。

## bs

## Barrel Shift

<b>bsrl</b>	rD, rA, rB	論理バレルシフトライト
<b>bsra</b>	rD, rA, rB	算術バレルシフトライト
<b>bsll</b>	rD, rA, rB	論理バレルシフトレフト

[illegible]

## 説明

レジスタ **rB** で指定された分だけレジスタ **rA** の内容をシフトし、その結果をレジスタ **rD** に格納します。

**bsll** に対しては、**S** ビット (サイド ビット) が 1 に設定されます。**S** ビットが 1 の場合は、左側へのシフトが実行されます。**bsrl** および **bsra** に対しては **S** ビットは 0 に設定され、右側へのシフトが実行されます。

bsra に対しては、T ビット (タイプ ビット) が 1 に設定されます。T ビット が 1 の場合、算術シフトが実行されます。bsrl および bsll に対しては T ビット は 0 に設定され、論理シフトが実行されます。

擬似コード

```

if S = 1 then
    (rD)  $\leftarrow$  (rA)  $\ll$  (rB)[27:31]
else
    if T = 1 then
        if ((rB)[27:31])  $\neq$  0 then
            (rD)[0:(rB)[27:31]-1]  $\leftarrow$  (rA)[0]
            (rD)[(rB)[27:31]:31]  $\leftarrow$  (rA)  $\gg$  (rB)[27:31]
        else
            (rD)  $\leftarrow$  (rA)
    else
        (rD)  $\leftarrow$  (rA)  $\gg$  (rB)[27:31]

```

## 変更されるレジスタ

- rD

## レイテンシ

C\_AREA\_OPTIMIZED=0 の場合 1 クロック サイクル

C AREA OPTIMIZED=1 の場合 2 クロック サイクル

## メモ

これらの命令は、オプションです。MicroBlaze でパラメータ C\_USE\_BARREL が 1 に設定されている場合にのみ使用可能です。

## bsi

## Barrel Shift Immediate

<b>bsrli</b>	rD, rA, IMM	論理バレル シフト ライト (即値を使用)
<b>bsrai</b>	rD, rA, IMM	算術バレル シフト ライト (即値を使用)
<b>bslli</b>	rD, rA, IMM	論理バレル シフト レフト (即値を使用)

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

## 説明

IMM で指定された分だけレジスタ **rA** の内容をシフトし、その結果をレジスタ **rD** に格納します。

**bslli** に対しては、**S** ビット (サイド ビット) が 1 に設定されます。**S** ビットが 1 の場合は、左側へのシフトが実行されます。**bsrli** および **bsrai** に対しては **S** ビットは 0 に設定され、右側へのシフトが実行されます。

**bsrai** に対しては、**T** ビット (タイプ ビット) が 1 に設定されます。**T** ビットが 1 の場合、算術シフトが実行されます。**bsrli** および **bslli** に対しては **T** ビットは 0 に設定され、論理シフトが実行されます。

## 擬似コード

```

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

C\_AREA\_OPTIMIZED=0 の場合 1 クロック サイクル

C\_AREA\_OPTIMIZED=1 の場合 2 クロック サイクル

## メモ

これらは、タイプ B 命令ではありません。これらの命令の前に **imm** 命令を実行しても、影響はありません。

これらの命令は、オプションです。MicroBlaze でパラメータ **C\_USE\_BARREL** が 1 に設定されている場合にのみ使用可能です。

## cmp

### Integer Compare

<b>cmp</b>	rD, rA, rB	rB を rA と比較 (符号付き)
<b>cmpu</b>	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 **rB** の内容からレジスタ **rA** の内容を減算し、その結果をレジスタ **rD** に格納します。

**rD** の MSB ビットは、**rA** と **rB** の関係が真であるかどうかを示すよう調整されます。**U** ビットが **1** の場合、**rA** と **rB** は符号なしの値とみなされます。**U** ビットが **0** の場合、**rA** と **rB** は符号付きの値とみなされます。

### 擬似コード

$$(rD) \leftarrow (rB) + \overline{(rA)} + 1$$

$$(rD) (MSB) \leftarrow (rA) > (rB)$$

### 変更されるレジスタ

- **rD**

### レイテンシ

1 クロック サイクル

## fadd

## Floating Point Arithmetic Add

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	0
0						6	11	16	21																		31

## 説明

レジスタ **rA** と **rB** の浮動小数点値の合計をレジスタ **rD** に格納します。

## 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)) and 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 例外が発生した場合は変更なし)
- ESR[EC] (FP 例外が発生した場合)
- FSR[IO,UF,OF,DO]

## レイテンシ

C\_AREA\_OPTIMIZED=0 の場合 4 クロック サイクル

C\_AREA\_OPTIMIZED=1 の場合 6 クロック サイクル

## メモ

この命令は、MicroBlaze でパラメータ C\_USE\_FPU が 0 より大きい値に設定されている場合にのみ使用可能です。

## frsub

## Reverse Floating Point Arithmetic Subtraction

<b>frsub</b>	rD, rA, rB	逆減算
--------------	------------	-----

[illegible]

## 説明

レジスタ **rB** の浮動小数点値からレジスタ **rA** の浮動小数点値を減算し、その結果をレジスタ **rD** に格納します。

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 例外が発生した場合は変更なし)
- ESR[EC] (FP 例外が発生した場合)
- FSR[IO,UF,OF,DO]

## レイテンシ

C\_AREA\_OPTIMIZED=0 の場合 4 クロック サイクル

C AREA OPTIMIZED=1 の場合 6 クロック サイクル

## メモ

この命令は、MicroBlaze でパラメータ C\_USE\_FPU が 0 より大きい値に設定されている場合のみ使用可能です。



## fmul

## Floating Point Arithmetic Multiplication

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	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** に格納します。

## 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 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 例外が発生した場合は変更なし)
- ESR[EC] (FP 例外が発生した場合)
- FSR[IO,UF,OF,DO]

## レイテンシ

C\_AREA\_OPTIMIZED=0 の場合 4 クロック サイクル

C\_AREA\_OPTIMIZED=1 の場合 6 クロック サイクル

## メモ

この命令は、MicroBlaze でパラメータ **C\_USE\_FPU** が 0 より大きい値に設定されている場合にのみ使用可能です。

fdiv

## Floating Point Arithmetic Division

**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						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** の浮動小数点値で除算し、その結果をレジスタ **rD** に格納します。

擬似コード

```

if isDnz(rA) or isDnz(rB) then
    (rD)  $\leftarrow$  0xFFC00000
    FSR[DO]  $\leftarrow$  1
    ESR[EC]  $\leftarrow$  00110
else
    if isSigNaN(rA) or isSigNaN(rB) or (isZero(rA) and isZero(rB)) or
        (isInfinite(rA) and isInfinite(rB)) then
        (rD)  $\leftarrow$  0xFFC00000
        FSR[IO]  $\leftarrow$  1
        ESR[EC]  $\leftarrow$  00110
    else if isQuietNaN(rA) or isQuietNaN(rB) then
        (rD)  $\leftarrow$  0xFFC00000
    else if isZero(rA) and not isInfinite(rB) then
        (rD)  $\leftarrow$  signInfinite((rB)/(rA))
        FSR[DZ]  $\leftarrow$  1
        ESR[EC]  $\leftarrow$  00110
    else if isDnz((rB)/(rA)) then
        (rD)  $\leftarrow$  signZero((rB)/(rA))
        FSR[UF]  $\leftarrow$  1
        ESR[EC]  $\leftarrow$  00110
    else if isNaN((rB)/(rA)) then
        (rD)  $\leftarrow$  signInfinite((rB)/(rA))
        FSR[OF]  $\leftarrow$  1
        ESR[EC]  $\leftarrow$  00110
    else
        (rD)  $\leftarrow$  (rB) / (rA)

```

## 変更されるレジスタ

- rD (FP 例外が発生した場合は変更なし)
- ESR[EC] (FP 例外が発生した場合)
- FSR[IO,UF,OF,DO,DZ]

## レイテンシ

C\_AREA\_OPTIMIZED=0 の場合 28 クロック サイクル

C\_AREA\_OPTIMIZED=1 の場合 30 クロック サイクル

## メモ

この命令は、MicroBlaze でパラメータ `C_USE_FPU` が 0 より大きい値に設定されている場合にのみ使用可能です。

# fcmp

## Floating Point Number Comparison

<b>fcmp.un</b>	rD, rA, rB	順序付けなし浮動小数点比較
<b>fcmp.lt</b>	rD, rA, rB	浮動小数点の小なり比較
<b>fcmp.eq</b>	rD, rA, rB	浮動小数点の等価比較
<b>fcmp.le</b>	rD, rA, rB	浮動小数点の以下比較
<b>fcmp.gt</b>	rD, rA, rB	浮動小数点の大なり比較
<b>fcmp.ne</b>	rD, rA, rB	浮動小数点の非等価比較
<b>fcmp.ge</b>	rD, rA, rB	浮動小数点の以上比較

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** に格納します。命令コードの **OpSel** フィールドで、実行する比較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
  { 表 4-2 の演算を参照 }
```

### 変更されるレジスタ

- rD (FP 例外が発生した場合は変更なし)
- ESR[EC] (FP 例外が発生した場合)
- FSR[IO,DO]

### レイテンシ

C\_AREA\_OPTIMIZED=0 の場合 1 クロック サイクル

C\_AREA\_OPTIMIZED=1 の場合 3 クロック サイクル

### メモ

この命令は、MicroBlaze でパラメータ **C\_USE\_FPU** が 0 より大きい値に設定されている場合にのみ使用可能です。

157 ページの表 4-2 に、浮動小数点比較演算を示します。

表 4-2：浮動小数点の比較演算

比較タイプ		オペランドの関係				
説明	OpSel	(rB) > (rA)	(rB) < (rA)	(rB) = (rA)	isSigNaN(rA) または isSigNaN(rB)	isQuietNaN(rA) または isQuietNaN(rB)
順序付けなし	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] ← 0011	(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
非等価	101	(rD) ← 1	(rD) ← 1	(rD) ← 0	(rD) ← 1 FSR[IO] ← 1 ESR[EC] ← 0011	(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

## fit

## Floating Point Convert Integer to Float

**flt** rD, rA

0	1	0	1	1	0	rD	rA	0	0	1	0	1	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値に変換し、その結果をレジスタ **rD** に格納します。これは 32 ビットの符号付き繰り上げ/繰り下げ変換で、結果は 32 ビットの浮動小数点値になります。

擬似コード

```
(rD) ← float ((rA))
```

## 変更されるレジスタ

- rD

## レイテンシ

C\_AREA\_OPTIMIZED=0 の場合 4 クロック サイクル

C\_AREA\_OPTIMIZED=1 の場合 6 クロック サイクル

## メモ

この命令は、MicroBlaze でパラメータ C\_USE\_FPU が 2 に設定されている場合にのみ使用可能です。



## fsqrt

## Floating Point Arithmetic Square Root

**fsqrt** $r_D, r_A$ 

平方根

0	1	0	1	1	0	rD	rA	0	0	1	1	1	0	0	0	0	0	0
0						6	11	16	21									31

## 説明

**rA** の値の浮動小数点平方根演算を実行し、その結果をレジスタ **rD** に格納します。

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 例外が発生した場合は変更なし)
- ESR[EC] (FP が発生した場合)
- FSR[IO,UF,OF,DO]

## レイテンシ

C\_AREA\_OPTIMIZED=0 の場合 27 クロック サイクル

C\_AREA\_OPTIMIZED=1 の場合 29 クロック サイクル

## メモ

この命令は、MicroBlaze でパラメータ C\_USE\_FPU が 2 に設定されている場合にのみ使用可能です。



## get

get from fsl interface

<b>tneaget</b>	rD, FSLx	FSL x からデータを読み出し t: テストのみ n: ノンブロッキング e: 制御ビットが 1 の場合例外 a: 不可分
<b>tnecaget</b>	rD, FSLx	FSL x から制御を読み出し t: テストのみ n: ノンブロッキング e: 制御ビットが 0 の場合例外 a: 不可分

0	1	1	0	1	1	rD	0	0	0	0	0	0	n	c	t	a	e	0	0	0	0	0	0	0	FSLx	
0						6							11												28	31

## 説明

FSLx インターフェイスから読み出した値をレジスタ rD に格納します。

get 命令には、32 種類あります。

ブロッキングの get 命令 (n ビットが 0) では、FSL インターフェイスから有効なデータが取得されるまでの間 MicroBlaze をストールします。ノンブロッキングの get 命令では MicroBlaze はストールされず、データが有効な場合はキャリーが 0 に、無効の場合は 1 になります。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定義されません。

すべてのデータ get 命令 (c ビットが 0) では、FSL インターフェイスの制御ビットが 0 であることが想定されます。0 ではない場合は、MSR[FSL\_Error] が 1 に設定されます。すべての制御 get 命令 (c ビットが 1) では、FSL インターフェイスの制御ビットが 1 であることが想定されます。1 でない場合は、MSR[FSL\_Error] が 1 に設定されます。

例外の get 命令 (e ビットが 1) では、制御ビットの不一致がある場合に例外が生成されます。この場合、ESR で EC に例外の原因が、ESS に FSL インデックスが設定されます。例外が発生した場合、ターゲット レジスタ rD はアップデートされず、FSL データが EDR に格納されます。

テストの get 命令 (t ビットが 1) は、FSL リンクへの読み出し信号がアサートされない点を除き、通常どおり処理されます。

不可分の get 命令 (a ビットが 1) は割り込み不可です。不可分 FSL 命令のシーケンスはグループ化され、プログラムのフローへは割り込みできません。ただし、例外は発生します。

MicroBlaze で MMU が使用される場合 (C\_USE\_MMU >= 1) これらの命令は特権命令になり、ユーザー モード (MSR[UM] = 1) で使用す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) ← FSLx
    if (n = 1) then
        MSR[Carry] ← not (FSLx Exists bit)
    if (FSLx Control bit ≠ c) and (FSLx Exists bit) then
        MSR[FSL_Error] ← 1
    if (e = 1) then
        ESR[EC] ← 00000
        ESR[ESS] ← instruction bits [28:31]
        EDR ← FSLx

```

## 変更されるレジスタ

- rD (例外が発生した場合は変更なし)
- MSR[FSL\_Error]
- MSR[Carry]
- ESR[EC] (FSL 例外または特権命令例外が発生した場合)
- ESR[ESS] (FSL 例外が発生した場合)
- EDR (FSL 例外が発生した場合)

## レイテンシ

C\_AREA\_OPTIMIZED=0 の場合 1 クロック サイクル

C\_AREA\_OPTIMIZED=1 の場合 2 クロック サイクル

ブロッキングの `get` 命令では、命令が完了可能になるまで MicroBlaze のパイプラインがストールします。割り込みは、パラメータ `C_USE_EXTENDED_FSL_INSTR` が 1 に設定され、命令が不可分でない場合に実行されます。

## メモ

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` を使用します。

ブロッキングの `get` 命令を遅延スロットに含めると割り込みが発生しなくなるため、パラメータ `C_USE_EXTENDED_FSL_INSTR` が 1 に設定されている場合は、この命令を遅延スロットに含めることができません。

ノンブロッキングの `get` 命令では、`rsubc` 命令を使用して、インデックス変数をデクリメントできます。

`C_FSL_EXCEPTION` が 1 に設定されている場合を除き、`e` ビットの設定は影響しません。

これらの命令は、MicroBlaze でパラメータ `C_FSL_LINKS` が 0 より大きい値に設定されている場合にのみ使用可能です。

拡張命令 (例外、テスト、および不可分の `get` 命令) は、パラメータ `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。

## getd

get from fsl interface dynamic

<b>tneagetd</b>	rD, rB	FSL rB[28:31] からデータを読み出し t: テストのみ n: ノンブロッキング e: 制御ビットが 1 の場合例外 a: 不可分
<b>tnecagetd</b>	rD, rB	FSL rB[28:31] から制御を読み出し t: テストのみ n: ノンブロッキング e: 制御ビットが 0 の場合例外 a: 不可分

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						6					11												31

## 説明

rB の下位 4 ビットで定義された FSL インターフェイスから読み出した結果が、レジスタ rD に格納されます。

getd 命令には、32 種類あります。

ブロッキング データの getd 命令 (n ビットが 0) では、FSL インターフェイスから有効なデータが取得されるまでの間 MicroBlaze をストールします。ノンブロッキング データの getd 命令では MicroBlaze はストールされず、データが有効な場合はキャリーが 0 に、無効の場合は 1 になります。アクセスが無効な場合は、デスティネーション レジスタの内容は定義されません。

すべてのデータ getd 命令 (c ビットが 0) では、FSL インターフェイスの制御ビットが 0 であることが想定されます。0 ではない場合は、MSR[FSL\_Error] が 1 に設定されます。すべての制御 getd 命令 (c ビットが 1) では、FSL インターフェイスの制御ビットが 1 であることが想定されます。1 ではない場合は、MSR[FSL\_Error] が 1 に設定されます。

例外の getd 命令 (e ビットが 1) では、制御ビットの不一致がある場合に例外が生成されます。この場合、ESR で EC に例外の原因が、ESS に FSL インデックスが設定されます。例外が発生した場合、ターゲット レジスタ rD はアップデートされず、FSL データが EDR に格納されます。

テストの getd 命令 (t ビットが 1) は、FSL リンクへの読み出し信号がアサートされない点を除き、通常どおり処理されます。

不可分の getd 命令 (a ビットが 1) は割り込み不可です。不可分 FSL 命令のシーケンスはグループ化され、プログラムのフローへは割り込みできません。ただし、例外は発生します。

MicroBlaze で MMU が使用される場合 (C\_USE\_MMU >= 1) これらの命令は特権命令になり、ユーザー モード (MSR[UM] = 1) で使用す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) ← (FSL rB[28:31])
    if (n = 1) then
        MSR[Carry] ← not (FSL rB[28:31] Exists bit)
    if (FSL rB[28:31] Control bit ≠ c) and (FSL rB[28:31] Exists bit) then
        MSR[FSL_Error] ← 1
    if (e = 1) then
        ESR[EC] ← 00000
        ESR[ESS] ← rB[28:31]
        EDR ← (FSL rB[28:31])

```

### 変更されるレジスタ

- rD (例外が発生した場合は変更なし)
- MSR[FSL\_Error]
- MSR[Carry]
- ESR[EC] (FSL 例外または特権命令例外が発生した場合)
- ESR[ESS] (FSL 例外が発生した場合)
- EDR (FSL 例外が発生した場合)

### レイテンシ

C\_AREA\_OPTIMIZED=0 の場合 1 クロック サイクル

C\_AREA\_OPTIMIZED=1 の場合 2 クロック サイクル

ブロッキングの `getd` 命令では、命令が完了可能になるまで **MicroBlaze** のパイプラインがストールします。割り込みは、命令が不可分 (割り込み不可) ではない場合に実行されます。

### メモ

ブロッキングの `getd` 命令を遅延スロットに含めると割り込みが発生しなくなるため、この命令を遅延スロットに含めることができません。

ノンブロッキングの `getd` 命令では、`rsubc` 命令を使用して、インデックス変数をデクリメントできます。

C\_FSL\_EXCEPTION が 1 に設定されている場合を除き、e ビットの設定は影響しません。

これらの命令は、パラメータ C\_FSL\_LINKS が 0 より大きい値に設定され、パラメータ 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。

## idiv

## Integer Divide

<b>idiv</b>	<b>rD, rA, rB</b>	rB を rA で除算(符号付き)
<b>idivu</b>	<b>rD, rA, rB</b>	rB を rA で除算 (符号なし)

0	1	0	0	1	0	rD	rA	rB	0	0	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** の内容で除算し、その結果をレジスタ **rD** に格納します。

U ビットが 1 の場合、**rA** と **rB** は符号なしの値とみなされます。U ビットが 0 の場合、**rA** と **rB** は符号付きの値とみなされます。

**rA** の値が 0 の場合、例外が発生しない限り、MSR の DZO ビットが 1 になり、**rD** の値は 0 になります。

U ビットがクリアの場合、**rA** の値は -1、**rB** の値は -2147483648 の場合、MSR の DZO ビットはセットされ **rD** の値は、例外が生成されない限り、-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** (除算例外が発生した場合は変更なし)
- MSR[DZO] (**rA** の値が 0 の場合)
- ESR[EC] (**rA** の値が 0 の場合)

## レイテンシ

(**rA**) = 0 の場合 1 クロック サイクル、それ以外は C\_AREA\_OPTIMIZED=0 の場合 32 クロック サイクル

(**rA**) = 0 の場合 1 クロック サイクル、それ以外は C\_AREA\_OPTIMIZED=1 の場合 34 クロック サイクル

## メモ

この命令は、MicroBlaze でパラメータ C\_USE\_DIV が 1 に設定されている場合にのみ使用可能です。

## imm

Immediate

**imm**

IMM

1	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	IMM														
0						6					11					16					31									

## 説明

**imm** 命令は、**IMM** の値を一時レジスタに読み込み、次の命令で 32 ビットの即値を形成できるように、この値をロックします。

**imm** 命令は、**タイプ B** 命令と共に使用します。**タイプ B** 命令では 16 ビットの即値フィールドしかないため、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** フィールドの値を取り込み、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。**タイプ B** 命令の前に **imm** 命令を使用すると、32 ビットの即値を使用できます。**imm** 命令は、16 ビットの **IMM** 値を次の命令で使用できるよう一時的にロックします。この **imm** 命令の直後に **タイプ B** 命令を実行すると、この **imm** 命令の 16 ビットの **IMM** 値 (上位 16 ビット) と **タイプ B** 命令の 16 ビットの **IMM** 値 (下位 16 ビット) から 32 ビットの即値が形成されます。**imm** 命令の後に **タイプ B** 命令が実行されない場合は、ロックされた値は解除されます。

## レイテンシ

1クロック サイクル

## メモ

**imm** 命令および後続のタイプ **B** 命令は不可分のため、この 2 つの命令の間にほかの命令を割り込ませることはできません。

ザイリンクスで提供するアセンブラでは、**imm** 命令が必要であることが自動的に検出されます。32 ビットの **IMM** 値がタイプ B 命令で指定されている場合、アセンブラで **IMM** 値が 16 ビットの値に変換され、実行ファイルでその命令の前に **imm** 命令が挿入されます。

## lbu

Load Byte Unsigned

lbu                      rD, rA, rB

1	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						6		11		16		21																	31

## 説明

レジスタ rA と rB 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) 読み込みます。データはレジスタ rD の最下位バイトに格納され、その他の 3 バイトはクリアされます。

データ TLB ミス例外は、仮想保護モードがイネーブルで、アドレスに対応する有効な変換エントリが TLB で検出されない場合に発生します。

アクセス不可ゾーン保護でアクセスが禁止されている場合は、データ格納例外が発生します。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の場合のアクセスにのみ適用されます。

## 擬似コード

```

Addr ← (rA) + (rB)
if TLB_Miss(Addr) and MSR[VM] = 1 then
    ESR[EC] ← 10010; ESR[S] ← 0
    MSR[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
    (rD)[24:31] ← Mem(Addr)
    (rD)[0:23] ← 0

```

## 変更されるレジスタ

- rD (例外が発生した場合は変更なし)
- MSR[UM]、MSR[VM]、MSR[UMS]、MSR[VMS] (例外が発生した場合)
- ESR[EC]、ESR[S] (例外が発生した場合)
- ESR[DIZ] (データ格納例外が発生した場合)

## レイテンシ

C\_AREA\_OPTIMIZED=0 の場合 1 クロック サイクル

C\_AREA\_OPTIMIZED=1 の場合 2 クロック サイクル

## lbui

Load Byte Unsigned Immediate

lbui                      rD, rA, IMM

1	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) 読み込みます。データはレジスタ **rD** の最下位バイトに格納され、その他の 3 バイトはクリアされます。

データ TLB ミス例外は、仮想保護モードがイネーブルで、アドレスに対応する有効な変換エントリが TLB で検出されない場合に発生します。

アクセス不可ゾーン保護でアクセスが禁止されている場合は、データ格納例外が発生します。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の場合のアクセスにのみ適用されます。

## 擬似コード

```

Addr ← (rA) + 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
    (rD)[24:31] ← Mem(Addr)
    (rD)[0:23] ← 0

```

## 変更されるレジスタ

- **rD** (例外が発生した場合は変更なし)
- **MSR[UM]**、**MSR[VM]**、**MSR[UMS]**、**MSR[VMS]** (例外が発生した場合)
- **ESR[EC]**、**ESR[S]** (例外が発生した場合)
- **ESR[DIZ]** (データ格納例外が発生した場合)

## レイテンシ

**C\_AREA\_OPTIMIZED=0** の場合 1 クロック サイクル

**C\_AREA\_OPTIMIZED=1** の場合 2 クロック サイクル

## メモ

デフォルトでは、タイプ **B** 命令は 16 ビットの **IMM** フィールドの値を取り込み、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。タイプ **B** 命令の前に **imm** 命令を使用すると、32 ビットの即値を使用できます。32 ビットの即値の使用に関する詳細は、166 ページの「**imm**」を参照してください。



## lhu

Load Halfword Unsigned

lhu                      rD, rA, rB

1	1	0	0	0	1		rD		rA		rB		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 を加算した結果を、ハーフワードで揃えたメモリ ロケーションから 1 ハーフワード (16 ビット) 読み込みます。データはレジスタ rD の下位ハーフワードに格納され、上位ハーフワードはクリアされます。

データ TLB ミス例外は、仮想保護モードがイネーブルで、アドレスに対応する有効な変換エントリが TLB で検出されない場合に発生します。

アクセス不可ゾーン保護でアクセスが禁止されている場合は、データ格納例外が発生します。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の場合のアクセスにのみ適用されます。

不整列データ アクセスによる例外は、アドレスの最下位ビットが 0 でない場合に発生します。

## 擬似コード

```

Addr ← (rA) + (rB)
if TLB_Miss(Addr) and MSR[VM] = 1 then
    ESR[EC] ← 10010; ESR[S] ← 0
    MSR[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
    (rD)[16:31] ← Mem(Addr)
    (rD)[0:15] ← 0

```

## 変更されるレジスタ

- rD (例外が発生した場合は変更なし)
- MSR[UM]、MSR[VM]、MSR[UMS]、MSR[VMS] (TLB ミス例外またはデータ格納例外が発生した場合)
- ESR[EC]、ESR[S] (例外が発生した場合)
- ESR[DIZ] (データ格納例外が発生した場合)
- ESR[W]、ESR[Rx] (不整列データ アクセスによる例外が発生した場合)

## レイテンシ

C\_AREA\_OPTIMIZED=0 の場合 1 クロック サイクル

C\_AREA\_OPTIMIZED=1 の場合 2 クロック サイクル

# lhui

## Load Halfword Unsigned Immediate

**lhui**                      rD, rA, IMM

1	1	1	0	0	1		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 値を加算した結果をハーフワードで揃えたメモリ ロケーションから 1 ハーフワード (16 ビット) 読み込みます。データはレジスタ rD の下位ハーフワードに格納され、上位ハーフワードはクリアされます。

データ TLB ミス例外は、仮想保護モードがイネーブルで、アドレスに対応する有効な変換エントリが TLB で検出されない場合に発生します。アクセス不可ゾーン保護でアクセスが禁止されている場合は、データ格納例外が発生します。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の場合のアクセスにのみ適用されます。不整列データ アクセスによる例外は、アドレスの最下位ビットが 0 でない場合に発生します。

### 擬似コード

```

Addr ← (rA) + 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
    (rD)[16:31] ← Mem(Addr)
    (rD)[0:15] ← 0

```

### 変更されるレジスタ

- rD (例外が発生した場合は変更なし)
- MSR[UM]、MSR[VM]、MSR[UMS]、MSR[VMS] (TLB ミス例外またはデータ格納例外が発生した場合)
- ESR[EC]、ESR[S] (例外が発生した場合)
- ESR[DIZ] (データ格納例外が発生した場合)
- ESR[W]、ESR[Rx] (不整列データ アクセスによる例外が発生した場合)

### レイテンシ

C\_AREA\_OPTIMIZED=0 の場合 1 クロック サイクル

C\_AREA\_OPTIMIZED=1 の場合 2 クロック サイクル

### メモ

デフォルトでは、タイプ B 命令は 16 ビットの IMM フィールドの値を取り込み、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。タイプ B 命令の前に imm 命令を使用すると、

32 ビットの即値を使用できます。32 ビットの即値の使用に関する詳細は、[166 ページの「imm」](#)を参照してください。

## lw

Load Word

lw                    rD, rA, rB

1	1	0	0	1	0		rD		rA		rB		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** を加算した結果をワードで揃えたメモリ ロケーションから1 ワード (32 ビット) 読み込みます。データは、レジスタ **rD** に格納されます。

データ TLB ミス例外は、仮想保護モードがイネーブルで、アドレスに対応する有効な変換エントリが TLB で検出されない場合に発生します。

アクセス不可ゾーン保護でアクセスが禁止されている場合は、データ格納例外が発生します。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の場合のアクセスにのみ適用されます。

不整列データ アクセスによる例外は、アドレスの最下位ビットが 0 でない場合に発生します。

## 擬似コード

```

Addr ← (rA) + (rB)
if TLB_Miss(Addr) and MSR[VM] = 1 then
    ESR[EC] ← 10010; ESR[S] ← 0
    MSR[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
    (rD) ← Mem(Addr)

```

## 変更されるレジスタ

- **rD** (例外が発生した場合は変更なし)
- **MSR[UM]**、**MSR[VM]**、**MSR[UMS]**、**MSR[VMS]** (TLB ミス例外またはデータ格納例外が発生した場合)
- **ESR[EC]**、**ESR[S]** (例外が発生した場合)
- **ESR[DIZ]** (データ格納例外が発生した場合)
- **ESR[W]**、**ESR[Rx]** (不整列データ アクセスによる例外が発生した場合)

## レイテンシ

C\_AREA\_OPTIMIZED=0 の場合 1 クロック サイクル

C\_AREA\_OPTIMIZED=1 の場合 2 クロック サイクル

## lwi

## Load Word Immediate

lwi                    rD, rA, IMM

1	1	1	0	1	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 値を加算した結果をワードで揃えたメモリロケーションから 1 ワード (16 ビット) 読み込みます。データは、レジスタ rD に格納されます。データ TLB ミス例外は、仮想保護モードがイネーブルで、アドレスに対応する有効な変換エントリが TLB で検出されない場合に発生します。アクセス不可ゾーン保護でアクセスが禁止されている場合は、データ格納例外が発生します。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の場合のアクセスにのみ適用されます。不整列データ アクセスによる例外は、アドレスの最下位ビットが 0 でない場合に発生します。

## 擬似コード

```

Addr ← (rA) + 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
    (rD) ← Mem(Addr)

```

## 変更されるレジスタ

- rD (例外が発生した場合は変更なし)
- MSR[UM]、MSR[VM]、MSR[UMS]、MSR[VMS] (TLB ミス例外またはデータ格納例外が発生した場合)
- ESR[EC]、ESR[S] (例外が発生した場合)
- ESR[DIZ] (データ格納例外が発生した場合)
- ESR[W]、ESR[Rx] (不整列データ アクセスによる例外が発生した場合)

## レイテンシ

C\_AREA\_OPTIMIZED=0 の場合 1 クロック サイクル

C\_AREA\_OPTIMIZED=1 の場合 2 クロック サイクル

## メモ

デフォルトでは、タイプ B 命令は 16 ビットの IMM フィールドの値を取り込み、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。タイプ B 命令の前に imm 命令を使用すると、32 ビットの即値を使用できます。32 ビットの即値の使用に関する詳細は、166 ページの「imm」を参照してください。

## lwx

## Load Word Exclusive

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	6	11	16	21
				31

## 説明

レジスタ **rA** と **rB** を加算した結果をワードで揃えたメモリ ロケーションから1 ワード (32 ビット) 読み込みます。データはレジスタ **rD** に配置され、予約ビットはセットされ、キャリーフラグ (**MSR[C]**) はクリアになります。

データ TLB ミス例外は、仮想保護モードがイネーブルで、アドレスに対応する有効な変換エントリが TLB で検出されない場合に発生します。

アクセス不可ゾーン保護でアクセスが禁止されている場合は、データ格納例外が発生します。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の場合のアクセスにのみ適用されます。

不整列データ アクセスによる例外は、アドレスの最下位ビットが 0 であっても発生しません。

## 擬似コード

```

Addr  $\leftarrow$  (rA) + (rB)
if TLB_Miss(Addr) and MSR[VM] = 1 then
    ESR[EC]  $\leftarrow$  10010; ESR[S]  $\leftarrow$  0
    MSR[UMS]  $\leftarrow$  MSR[UM]; MSR[VMS]  $\leftarrow$  MSR[VM]; MSR[UM]  $\leftarrow$  0; MSR[VM]  $\leftarrow$  0
else if Access_Protected(Addr) and MSR[UM] = 1 and MSR[VM] = 1 then
    ESR[EC]  $\leftarrow$  10000; ESR[S]  $\leftarrow$  0; ESR[DIZ]  $\leftarrow$  1
    MSR[UMS]  $\leftarrow$  MSR[UM]; MSR[VMS]  $\leftarrow$  MSR[VM]; MSR[UM]  $\leftarrow$  0; MSR[VM]  $\leftarrow$  0
else
    (rD)  $\leftarrow$  Mem(Addr); Reservation  $\leftarrow$  1; MSR[C]  $\leftarrow$  0

```

## 変更されるレジスタ

- **rD** および **MSR[C]** (例外が発生しない場合)、例外が発生した場合は変更なし
- **MSR[UM]**、**MSR[VM]**、**MSR[UMS]**、**MSR[VMS]** (TLB ミス例外またはデータ格納例外が発生した場合)
- **ESR[EC]**、**ESR[S]** (例外が発生した場合)
- **ESR[DIZ]** (データ格納例外が発生した場合)

## レイテンシ

C\_AREA\_OPTIMIZED=0 の場合 1 クロック サイクル

C\_AREA\_OPTIMIZED=1の場合 2 クロック サイクル

## メモ

この命令は **STX** と共に使用して、セマフォやスピン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使用します。



EDR は、C\_FSL\_EXCEPTION パラメータが 1 に、C\_FSL\_LINKS パラメータが 0 より大きい値に設定されている場合にのみオペランドとして使用可能です。

FSR は、C\_USE\_FPU パラメータが 0 より大きい値に設定されている場合にのみオペランドとして使用可能です。

PID、ZPR、TLBLO および TLBHI は、C\_USE\_MMU > 1 および C\_MMU\_TLB\_ACCESS = 1 または 3 の場合にのみオペランドとして使用可能です。

TLBX は C\_USE\_MMU > 1 および C\_MMU\_TLB\_ACCESS > 0 の場合にのみオペランドとして使用可能です。

PVR0 は C\_PVR が 1 または 2 に、PVR1 ~ PVR11 は C\_PVR が 2 に設定されている場合にのみオペランドとして使用可能です。





## msrset

## Read MSR and set bits in MSR

msrset rD, Imm

1 0 0 1 0 1	rD	1 0 0 0 0 0	Imm15
0 6	11	16 17	31

## 説明

特殊用途のレジスタ **MSR** の内容をレジスタ **rD** にコピーします。**IMM** 値が 1 であるビット位置は、**MSR** でセットされます。**IMM** 値が 0 であるビット位置は、そのままになります。

MicroBlaze で MMU が使用される場合 (C\_USE\_MMU >= 1)、IMM 値が C のみに影響する場合を除いてこの命令は特権命令になり、ユーザー モード (MSR[UM] = 1) で使用す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  $\neq$  0x4 then
    ESR[EC]  $\leftarrow$  00111
else
    (rD)  $\leftarrow$  (MSR)
    (MSR)  $\leftarrow$  (MSR)  $\vee$  (IMM)

```

## 変更されるレジスタ

- rD
- MSR
- ESR[EC] (特権命令例外が発生した場合)

## レイテンシ

### 1 クロック サイクル

## メモ

**MSRSET** を実行すると、キャリー ビット はすぐに変化しますが、その他のビット は命令が実行されてから 1 クロック サイクル後に変化します。**EIP** ビットまたは **BIP** ビット がオンに設定された場合、プロセッサはその後の命令の割り込みおよび通常のハードウェア ブレークには反応しません。

**MSR** の値には、パイプラインのストール ビヘイビアによっては直前の命令の結果が含まれていない場合もあります。**MSR** に影響しない命令は、**MSRSET** 命令の前に実行し、正しい **MSR** 値が得られるようにする必要があります。

C\_USE\_MMU >= 1 の場合即値は  $2^{15}$  以下、それ以外の場合は  $2^{14}$  である必要があります。  
C\_USE\_MMU >= 1 の場合 MSR のビット 17 ~ 31 のみがセットになり、それ以外の場合はビット 18 ~ 31 がセットになります。

この命令は、パラメータ C USE MSR INSTR が 1 に設定されている場合にのみ使用可能です。

**MSR[VM]** を設定する場合、命令の後に **BRI 4** のような同期分岐命令が必要です。

## mts

## Move To Special Purpose Register

mts                  rS, rA

1	0	0	1	0	1	0	0	0	0	0	rA	1	1	rS											
0						6					11			16	18										31

## 説明

レジスタ **rD** の内容を特殊用途レジスタ **rS** にコピーします。特殊用途レジスタ **TLBLO** および **TLBHI** は、インデックスが **TLBX** である 統合 **TLB** のエントリへのコピーに使用されます。

**MicroBlaze** で **MMU** が使用される場合 (**C\_USE\_MMU** >= 1) この命令は特権命令になり、ユーザー モード (**MSR[UM]** = 1) で使用す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
    switch (rS)
    case 0x0001 : MSR   ← (rA)
    case 0x0007 : FSR   ← (rA)
    case 0x1000 : PID    ← (rA)
    case 0x1001 : ZPR    ← (rA)
    case 0x1002 : TLBX   ← (rA)
    case 0x1003 : TLBLO ← (rA)
    case 0x1004 : TLBHI ← (rA)
    case 0x1005 : TLBSX ← (rA)

```

## 変更されるレジスタ

- **rS**
- **ESR[EC]** (特権命令例外が発生した場合)

## レイテンシ

1 クロック サイクル

## メモ

**MTS** を使用して **MSR** に書き込むと、すべてのビットは命令が実行されてから 1 クロック サイクル後に変化します。**MSR** を書き込む **MTS** 命令は **MSR** のデータを使用するほかの命令と連続して実行しないでください。**IE** ビットがクリアされた場合、プロセッサはその後の命令の割り込みには反応しません。**EIP** ビットまたは **BIP** ビットがオンに設定された場合、プロセッサはその後の命令の割り込みおよび通常のハードウェア ブレークには反応しません。

アセンブリ言語で特殊用途レジスタを参照するには、**MSR** では **rmsr**、**FSR** では **rfsr**、**PID** では **rpil**、**ZPR** では **rzpr**、**TLBLO** では **rtlblo**、**TLBHI** では **rtlbhi**、**TLBX** では **rtlbx**、**TLBSX** では **rtlsx** を使用してください。

**PC**、**ESR**、**EAR**、**BTR**、**EDR**、**PVR0** ~ **PVR11**は、**MTS** 命令を使用して書き込むことはできません。

**FSR** は、**MicroBlaze** でパラメータ **C\_USE\_FPU** が 0 より大きい値に設定されている場合にのみデスティネーションとして使用可能です。

PID、ZPR、および TLBSX は、 $C\_USE\_MMU > 1$  および  $C\_MMU\_TLB\_ACCESS > 1$  の場合にのみデスティネーションとして使用可能です。TLBLO、TLBHI、および TLBX は、 $C\_USE\_MMU > 1$  の場合にのみデスティネーションとして使用可能です。

MSR[VM] または PID を変更する場合、命令の後に BRI 4 のような同期分岐命令が必要です。



# mulh

Multiply High

mulh            rD, rA, rB

0	1	0	0	0	0		rD		rA		rB		0	0	0	0	0	0	0	0	0	0	1
0						6			11		16		21										31

## 説明

レジスタ rA と rB の内容を乗算し、その結果をレジスタ rD に格納します。この 32 X 32 の符号付き乗算では、64 ビットの結果が出力されます。この値の上位ワードは rD に格納され、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

C\_AREA\_OPTIMIZED=0 の場合 1 クロック サイクル

C\_AREA\_OPTIMIZED=1 の場合 3 クロック サイクル

## メモ

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含まれており、MicroBlaze でパラメータ C\_USE\_HW\_MUL が 2 に設定されている場合にのみ使用可能です。

MULH を使用する場合は MUL 命令の 30 ビット目と 31 ビット目を 0 にし、2 つの命令が区別できるようにする必要があります。MicroBlaze の以前のバージョンでは、これらのビットは 0 に定義されていましたが、実際の値は定義どおりではありませんでした。

## mulhu

Multiply High Unsigned

mulhu rD, rA, rB

0	1	0	0	0	0		rD		rA		rB		0	0	0	0	0	0	0	0	1	1
0						6			11		16		21									31

## 説明

レジスタ **rA** と **rB** の内容を乗算し、その結果をレジスタ **rD** に格納します。この 32 X 32 の符号なし乗算では、符号なし 64 ビットの結果が出力されます。この値の上位ワードは **rD** に格納され、下位ワードは破棄されます。

## 擬似コード

$$(rD) \leftarrow \text{MSW}((rA) \times (rB)), \text{ unsigned}$$

## 変更されるレジスタ

- rD

## レイテンシ

C\_AREA\_OPTIMIZED=0 の場合 1 クロック サイクル

C\_AREA\_OPTIMIZED=1 の場合 3 クロック サイクル

## メモ

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含まれており、MicroBlaze でパラメータ **C\_USE\_HW\_MUL** が 2 に設定されている場合にのみ使用可能です。

**MULHU** を使用する場合は **MUL** 命令の 30 ビット目と 31 ビット目を 0 にし、2 つの命令が区別できるようにする必要があります。MicroBlaze の以前のバージョンでは、これらのビットは 0 に定義されていましたが、実際の値は定義どおりではありませんでした。

## mulhsu Multiply High Signed Unsigned

**mulhsu**            rD, rA, rB

0	1	0	0	0	0		rD		rA		rB		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、その結果をレジスタ **rD** に格納します。この 符号付き 32 ビット **X** 符号なし 32 ビットの乗算では、符号付き 64 ビットの結果が出されます。この値の最上位ワードが **rD** に格納されます。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

**C\_AREA\_OPTIMIZED=0** の場合 1 クロック サイクル

**C\_AREA\_OPTIMIZED=1** の場合 3 クロック サイクル

### メモ

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り、**MicroBlaze** のパラメータ **C\_USE\_HW\_MUL** が 2 に設定されている場合にのみ有効です。

**MULHSU** を使用する場合は **MUL** 命令の 30 ビット目と 31 ビット目を 0 にし、2 つの命令が区別できるようにする必要があります。**MicroBlaze** の以前のバージョンでは、これらのビットは 0 に定義されていましたが、実際の値は定義どおりではありませんでした。



## mul

## Multiply Immediate

mul 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** の値を乗算し、その結果を **rD** に格納します。この 32 X 32 乗算器では、64 ビットの結果が出力されます。この値の下位ワードが **rD** に格納されます。上位ワードは破棄されます。

## 擬似コード

$$(rD) \leftarrow \text{LSW} ( (rA) \times \text{sext}(IMM) )$$

## 変更されるレジスタ

- **rD**

## レイテンシ

**C\_AREA\_OPTIMIZED=0** の場合 1 クロック サイクル

**C\_AREA\_OPTIMIZED=1** の場合 3 クロック サイクル

## メモ

デフォルトでは、タイプ **B** 命令は 16 ビットの **IMM** フィールドの値を取り込み、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。タイプ **B** 命令の前に **imm** 命令を使用すると、32 ビットの即値を使用できます。32 ビットの即値の使用に関する詳細は、[166 ページの「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含まれており、**MicroBlaze** でパラメータ **C\_USE\_HW\_MUL** が 0 より大きい値に設定されている場合にのみ使用可能です。

# Or

## Logical 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						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 OR 演算し、その結果をレジスタ rD に格納します。

### 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 クロック サイクル

# ori Logical OR with Immediate

**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 の内容と 32 ビットに符号拡張された IMM フィールドの値を OR 演算し、その結果をレジスタ rD に格納します。

## 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 クロック サイクル

## メモ

デフォルトでは、タイプ B 命令は 16 ビットの IMM フィールドの値を取り込み、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。タイプ B 命令の前に imm 命令を使用すると、32 ビットの即値を使用できます。32 ビットの即値の使用に関する詳細は、[166 ページの「imm」](#)を参照してください。

## pcmpbf

### Pattern Compare Byte Find

**pcmpbf**      **rD, rA, rB**      バイト単位で比較し、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						6			11			16		21													31

#### 説明

レジスタ **rA** の内容とレジスタ **rB** の内容をバイト単位で比較します。

- **rD** には、MSB を位置 1、LSB を位置 4 として、最初に一致したバイトの位置が格納されます。
- 一致するバイト ペアがない場合は、**rD** は 0 に設定されます。

#### 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 クロック サイクル

#### メモ

この命令は、パラメータ **C\_USE\_PCMP\_INSTR** が 1 に設定されている場合にのみ使用可能です。

**pcmpeq**

Pattern Compare Equal

**pcmpeq**      rD, rA, rB      一致した場合に 1 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** の内容とレジスタ **rB** の内容を比較します。

- 2 つのレジスタの内容が一致した場合は **rD** に 1 が格納され、一致しない場合は 0 が格納されます。

**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 クロック サイクル

**メモ**この命令は、パラメータ **C\_USE\_PCOMP\_INSTR** が 1 に設定されている場合にのみ使用可能です。

## pcmpne

Pattern Compare Not Equal

**pcmpne**      rD, rA, rB      一致しない場合に 1 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** の内容とレジスタ **rB** の内容を比較します。

- 2 つのレジスタの内容が一致した場合は **rD** に 0 が格納され、一致しない場合は 1 が格納されます。

### 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 クロック サイクル

### メモ

この命令は、パラメータ **C\_USE\_PCMP\_INSTR** が 1 に設定されている場合にのみ使用可能です。

## put

Put to fsl interface

<b>naput</b>	rA, FSLx	FSL x にデータを書き込み n: ノンブロッキング a: 不可分
<b>tnaput</b>	FSLx	FSL x にデータを書き込み (テストのみ) n: ノンブロッキング a: 不可分
<b>ncaput</b>	rA, FSLx	FSL x に制御を書き込み n: ノンブロッキング a: 不可分
<b>tncaput</b>	FSLx	FSL x に制御を書き込み (テストのみ) n: ノンブロッキング a: 不可分

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 の値を FSLx インターフェイスに書き込みます。

put 命令には、16 種類あります。

ブロッキングの put 命令 (n ビットが 0) では、FSL インターフェイスに書き込むスペースができるまでの間 MicroBlaze をストールします。ノンブロッキングの put 命令では MicroBlaze はストールされず、スペースがある場合はキャリーが 0 に、スペースがない場合は 1 に設定されます。

すべてのデータ put 命令 (c ビットが 0) では、FSL インターフェイスへの制御ビットが 0 に、すべての制御 put 命令 (c ビットが 1) では 1 に設定されます。

テストの put 命令 (t ビットが 1) は、FSL リンクへの書き込み信号がアサートされず、ソースレジスタが必要でない点を除き、通常どおり処理されます。

不可分の put 命令 (a ビットが 1) は割り込み不可です。不可分 FSL 命令のシーケンスはグループ化され、プログラムのフローへは割り込みできません。ただし、例外は発生します。

MicroBlaze で MMU が使用される場合 (C\_USE\_MMU >= 1) これらの命令は特権命令になり、ユーザー モード (MSR[UM] = 1) で使用す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
    (FSLx) ← (rA)
    if (n = 1) then
        MSR[Carry] ← (FSLx Full bit)
    (FSLx Control bit) ← C

```

## 変更されるレジスタ

- MSR[Carry]
- ESR[EC] (特権命令例外が発生した場合)

## レイテンシ

C\_AREA\_OPTIMIZED=0 の場合 1 クロック サイクル

C\_AREA\_OPTIMIZED=1 の場合 2 クロック サイクル

ブロッキングの put 命令では、命令が完了可能になるまで MicroBlaze のパイプラインがストールします。割り込みは、パラメータ C\_USE\_EXTENDED\_FSL\_INSTR が 1 に設定され、命令が不可分でない場合に実行されます。

## メモ

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 を使用します。

ブロッキングの put 命令を遅延スロットに含めると割り込みが発生しなくなるため、パラメータ C\_USE\_EXTENDED\_FSL\_INSTR が 1 に設定されている場合は、この命令を遅延スロットに含めることができません。

これらの命令は、MicroBlaze でパラメータ C\_FSL\_LINKS が 0 より大きい値に設定されている場合にのみ使用可能です。

拡張命令 (不可分の put 命令) は、MicroBlaze でパラメータ 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。



## putd

Put to fsl interface dynamic

<b>naputd</b>	rA, rB	FSL rB[28:31] にデータを書き込み n: ノンブロッキング a: 不可分
<b>tnaputd</b>	rB	FSL rB[28:31] にデータを書き込み (テストのみ) n: ノンブロッキング a: 不可分
<b>ncaputd</b>	rA, rB	FSL rB[28:31] に制御を書き込み n: ノンブロッキング a: 不可分
<b>tncaputd</b>	rB	FSL rB[28:31] に制御を書き込み (テストのみ) n: ノンブロッキング a: 不可分

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						6					11		16					21						31

## 説明

レジスタ rA から読み出した値を、rB の下位 4 ビットで定義された FSL インターフェイスに書き込みます。

putd 命令には、16 種類あります。

ブロッキングの putd 命令 (n ビットが 0) では、FSL インターフェイスに書き込むスペースができるまでの間 MicroBlaze をストールします。ノンブロッキングの putd 命令では MicroBlaze はストールされず、スペースがある場合はキャリーが 0 に、スペースがない場合は 1 に設定されます。

すべてのデータ putd 命令 (c ビットが 0) では、FSL インターフェイスへの制御ビットが 0 に、すべての制御 putd 命令 (c ビットが 1) では 1 に設定されます。

テストの putd 命令 (t ビットが 1) では、FSL リンクへの書き込み信号がアサートされず、ソース レジスタが必要でない点を除き、通常どおり処理されます。

不可分の putd 命令 (a ビットが 1) は割り込み不可です。不可分 FSL 命令のシーケンスはグループ化され、プログラムのフローへは割り込みできません。ただし、例外は発生します。

MicroBlaze で MMU が使用される場合 (C\_USE\_MMU >= 1) これらの命令は特権命令になり、ユーザー モード (MSR[UM] = 1) で使用す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
    (FSL rB[28:31]) ← (rA)
    if (n = 1) then
        MSR[Carry] ← (FSL rB[28:31] Full bit)
        (FSL rB[28:31] Control bit) ← C

```

### 変更されるレジスタ

- MSR[Carry]
- ESR[EC] (特権命令例外が発生した場合)

### レイテンシ

C\_AREA\_OPTIMIZED=0 の場合 1 クロック サイクル

C\_AREA\_OPTIMIZED=1 の場合 2 クロック サイクル

ブロッキングの **putd** 命令では、命令が完了可能になるまで **MicroBlaze** のパイプラインがストールします。割り込みは、命令が不可分 (割り込み不可) ではない場合に実行されます。

### メモ

ブロッキングの **putd** 命令を遅延スロットに含めると割り込みが発生しなくなるため、この命令を遅延スロットに含めることができません。

これらの命令は、パラメータ **C\_FSL\_LINKS** が 0 より大きい値に設定され、パラメータ **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。

## rsub

## Arithmetic Reverse Subtract

<b>rsub</b>	rD, rA, rB	減算
<b>rsubc</b>	rD, rA, rB	キャリー付き減算
<b>rsubk</b>	rD, rA, rB	減算 (キャリーを保持)
<b>rsubkc</b>	rD, rA, rB	キャリー付き減算 (キャリーを保持)

0	0	0	K	C	1	rD					rA					rB					0	0	0	0	0	0	0	0	0	0	
0						6					11					16					21					31					

## 說明

レジスタ **rB** の内容からレジスタ **rA** の内容を減算し、その結果をレジスタ **rD** に格納します。**rsubk** に対しては、命令のビット 3 (図で **K** と表記) が 1 に設定されています。**rsubc** に対しては、命令のビット 4 (図で **C** と表記) が 1 に設定されています。**rsubkc** に対しては、両方のビットが 1 に設定されています。

ビット 3 が 1 の場合 (**rsubk**、**rsubkc**)、命令の実行結果にかかわらず、キャリー フラグに既存の値が保持されます。ビット 3 が 0 の場合 (**rsub**、**rsubc**)、キャリー フラグが命令の実行結果に応じて変更されます。

ビット 4 が 1 の場合 (rsbuc、rsbucb)、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 0 の場合 (rsbu、rsbub)、キャリー フラグの内容は命令の実行には影響しません (通常の加算を実行)。

## 擬似コード

```

if C = 0 then                
  (rD) ← (rB) + (rA) + 1
else
  (rD) ← (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 クロック サイクル

## メモ

減算では、 $\text{キャリー} = (\overline{\text{ボロー}})$  です。つまり、減算でキャリーが設定されている場合はボローはなく、キャリーが設定されていない場合は、ボローがあります。

## rsubi

### Arithmetic Reverse Subtract Immediate

<b>rsubi</b>	rD, rA, IMM	即値減算
<b>rsubic</b>	rD, rA, IMM	キャリー付き即値減算
<b>rsubik</b>	rD, rA, IMM	即値減算 (キャリーを保持)
<b>rsubikc</b>	rD, rA, IMM	キャリー付き即値減算 (キャリーを保持)

0	0	1	K	C	1	rD	rA	IMM	
0						6	11	16	31

#### 説明

32 ビットに符号拡張された **IMM** フィールドの値からレジスタ **rA** の内容を減算し、その結果をレジスタ **rD** に格納します。**rsubik** に対しては、命令のビット 3 (図で **K** と表記) が 1 に設定されています。**rsubic** に対しては、命令のビット 4 (図で **C** と表記) が 1 に設定されています。**rsubikc** に対しては、両方のビットが 1 に設定されています。

ビット 3 が 1 の場合 (**rsubik**、**rsubikc**)、命令の実行結果にかかわらず、キャリー フラグに既存の値が保持されます。ビット 3 が 0 の場合 (**rsubi**、**rsubic**)、キャリー フラグが命令の実行結果に応じて変更されます。ビット 4 が 1 の場合 (**rsubic**、**rsubikc**)、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 0 の場合 (**rsubi**、**rsubik**)、キャリー フラグの内容は命令の実行には影響しません (通常の加算を実行)。

#### 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 クロック サイクル

#### メモ

減算では、キャリー = (ボロー) です。つまり、減算でキャリーが設定されている場合はボローはなく、キャリーが設定されていない場合は、ボローがあります。デフォルトでは、タイプ **B** 命令は 16 ビットの **IMM** フィールドの値を取り込み、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。タイプ **B** 命令の前に **imm** 命令を使用すると、32 ビットの即値を使用できます。32 ビットの即値の使用に関する詳細は、166 ページの「**imm**」を参照してください。

## rtbd

Return from Break

rtbd

rA, IMM

1	0	1	1	0	1	1	0	0	1	0	rA	IMM		
0						6					11	16		31

## 説明

ブレークから戻った際に、**rA** の内容と 32 ビットに符号拡張された **IMM** フィールドの値を加算して指定されるロケーションに分岐します。この命令の後、**MSR** の **BIP** がクリアされ、ブレークがイネーブルになります。

この命令では、常に遅延スロットが使用されます。**RTBD** に続く命令は、常に分岐ターゲットの前に実行されます。この遅延スロット命令では、ブレークがディスエーブルにされます。

**MicroBlaze** で **MMU** が使用される場合 (**C\_USE\_MMU** >= 1) この命令は特権命令になり、ユーザー モード (**MSR[UM]** = 1) で使用す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 ← (rA) + 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] (特権命令例外が発生した場合)

## レイテンシ

2 クロック サイクル

## メモ

通常は、汎用レジスタ **r16** を **rA** として使用します。

遅延スロットは、**imm**、分岐、ブレーク命令で使用できません。割り込みおよび外部ハードウェアブレークは、遅延スロットの分岐が完了するまで延期されます。

## rtid

## Return from Interrupt

rtid	rA, IMM
------	---------

1	0	1	1	0	1	1	0	0	0	1	rA	IMM									
0						6					11	16				31					

## 説明

割り込みから戻った際に、**rA** の内容と 32 ビットに符号拡張された **IMM** フィールドの値を加算して指定されるロケーションに分岐します。この命令の後、割り込みがイネーブルになります。

この命令では、常に遅延スロットが使用されます。**RTID** に続く命令は、常に分岐ターゲットの前に実行されます。この遅延スロット命令では、割り込みがディスエーブルにされます。

MicroBlaze で MMU が使用される場合 (C\_USE\_MMU >= 1) この命令は特権命令になり、ユーザー モード (MSR[UM] = 1) で使用すると特権命令例外が発生します。

## 疑似コード

```

if MSR[UM] = 1 then
    ESR[EC] ← 00111
else
    PC ← (rA) + sext(IMM)
    allow following instruction to complete execution
    MSR[IE] ← 1
    MSR[UM] ← MSR[UMS]
    MSR[VM] ← MSR[VMS]

```

## 変更されるレジスタ

- PC
- MSR[IE]、MSR[UM]、MSR[VM]
- ESR[EC] (特権命令例外が発生した場合)

## レイテンシ

## 2クロック サイクル

## メモ

通常は、汎用レジスタ **r14** を **rA** として使用します。

遅延スロットは、imm、分岐、ブレーク命令で使用できません。割り込みおよび外部ハードウェアブレークは、遅延スロットの分岐が完了するまで延期されます。

## rtd

## Return from Exception

rtd                  rA, IMM

1	0	1	1	0	1	1	0	1	0	0	rA	IMM	
0						6					11	16	31

## 説明

例外から戻った際に、**rA** の内容と 32 ビットに符号拡張された **IMM** フィールドの値を加算して指定されるロケーションに分岐します。この命令の後、例外がイネーブルになります。

この命令では、常に遅延スロットが使用されます。**RTED** に続く命令は、常に分岐ターゲットの前に実行されます。

MicroBlaze で MMU が使用される場合 (**C\_USE\_MMU** >= 1) この命令は特権命令になり、ユーザー モード (**MSR[UM]** = 1) で使用す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 ← (rA) + 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 クロック サイクル

## メモ

通常は、汎用レジスタ **r17** を **rA** として使用します。この命令は、MicroBlaze **C\*\_EXCEPTION** パラメータの少なくとも 1 つが 1 に設定されている場合または **C\_USE\_MMU** > 0 の場合にのみ使用できます。

遅延スロットは、**imm**、分岐、ブレーク命令で使用できません。割り込みおよび外部ハードウェアブレークは、遅延スロットの分岐が完了するまで延期されます。

**MSR[EE]** が設定されている場合は、遅延スロットにある命令で例外が発生した場合に例外がイネーブルの状態では例外ハンドラの処理を開始するため、通常この命令は使用できません。

メモ：例外から戻ったら、MSR[DS] が設定されているかをチェックする必要があります。設定されていれば、BTR に格納されているアドレスに戻ります。



## rtsd

Return from Subroutine

**rtsd**      rA, IMM

1	0	1	1	0	1	1	0	0	0	0	rA	IMM			
0						6				11		16		31	

## 説明

サブルーチンから戻った際に、**rA** の内容と 32 ビットに符号拡張された **IMM** フィールドの値を加算して指定されるロケーションに分岐します。

この命令では、常に遅延スロットが使用されます。**RTSD** に続く命令は、常に分岐ターゲットの前に実行されます。

## 擬似コード

```
PC ← (rA) + sext (IMM)
allow following instruction to complete execution
```

## 変更されるレジスタ

- PC

## レイテンシ

2 クロック サイクル

## メモ

通常は、汎用レジスタ **r15** を **rA** として使用します。

遅延スロットは、**imm**、分岐、ブレーク命令で使用できません。割り込みおよび外部ハードウェアブレークは、遅延スロットの分岐が完了するまで延期されます。

## sb

Store Byte

**sb**                      rD, rA, rB

1	1	0	1	0	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 **rD** の最下位バイトの内容を、レジスタ **rA** と **rB** 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。

データ TLB ミス例外は、仮想保護モードがイネーブルで、アドレスに対応する有効な変換エントリが TLB で検出されない場合に発生します。

仮想保護モードがイネーブルで、アクセス不可または読み出しのみのゾーン保護でアクセスが禁止されている場合は、データ格納例外が発生します。アクセス不可は、ユーザー モードでのみ使用できます。

### 擬似コード

```

Addr ← (rA) + (rB)
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) [24:31]

```

### 変更されるレジスタ

- MSR[UM]、MSR[VM]、MSR[UMS]、MSR[VMS] (例外が発生した場合)
- ESR[EC]、ESR[S] (例外が発生した場合)
- ESR[DIZ] (データ格納例外が発生した場合)

### レイテンシ

C\_AREA\_OPTIMIZED=0 の場合 1 クロック サイクル

C\_AREA\_OPTIMIZED=1 の場合 2 クロック サイクル

## sbi

Store Byte Immediate

**sbi**            rD, rA, IMM

1	1	1	1	0	0		rD		rA		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** と 32 ビットに符号拡張された **IMM** の値を加算した結果のメモリ ロケーションに格納します。

データ TLB ミス例外は、仮想保護モードがイネーブルで、アドレスに対応する有効な変換エントリが TLB で検出されない場合に発生します。

仮想保護モードがイネーブルで、アクセス不可または読み出しのみのゾーン保護でアクセスが禁止されている場合は、データ格納例外が発生します。アクセス不可は、ユーザー モードでのみ使用できます。

## 擬似コード

```

Addr ← (rA) + 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)[24:31]

```

## 変更されるレジスタ

- MSR[UM]、MSR[VM]、MSR[UMS]、MSR[VMS] (例外が発生した場合)
- ESR[EC]、ESR[S] (例外が発生した場合)
- ESR[DIZ] (データ格納例外が発生した場合)

## レイテンシ

C\_AREA\_OPTIMIZED=0 の場合 1 クロック サイクル

C\_AREA\_OPTIMIZED=1 の場合 2 クロック サイクル

## メモ

デフォルトでは、タイプ B 命令は 16 ビットの **IMM** フィールドの値を取り込み、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。タイプ B 命令の前に **imm** 命令を使用すると、32 ビットの即値を使用できます。32 ビットの即値の使用に関する詳細は、166 ページの「**imm**」を参照してください。

## sext16

Sign Extend Halfword

**sext16**      rD, rA

1	0	0	1	0	0		rD		rA		0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	1
0						6			11		16																31

### 説明

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、rA のビット 16 ～ 31 が 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 クロック サイクル

## sext8

Sign Extend Byte

**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		
0						6			11		16																31	

### 説明

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、rA ビット 24 ～ 31 が 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 クロック サイクル

## sh

## Store Halfword

sh                    rD, rA, rB

1	1	0	1	0	1		rD		rA		rB		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 の下位ハーフ ワードの内容を、レジスタ rA と rB の内容を加算した結果をハーフワードで揃えたメモリ ロケーションに格納します。

データ TLB ミス例外は、仮想保護モードがイネーブルで、アドレスに対応する有効な変換エントリが TLB で検出されない場合に発生します。

仮想保護モードがイネーブルで、アクセス不可または読み出しのみのゾーン保護でアクセスが禁止されている場合は、データ格納例外が発生します。アクセス不可は、ユーザー モードでのみ使用できます。

不整列データ アクセスによる例外は、アドレスの最下位ビットが 0 でない場合に発生します。

## 擬似コード

```

Addr ← (rA) + (rB)
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)[16:31]

```

## 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格納例外が発生した場合)
- ESR[EC]、ESR[S] (例外が発生した場合)
- ESR[DIZ] (データ格納例外が発生した場合)
- ESR[W]、ESR[Rx] (不整列データ アクセスによる例外が発生した場合)

## レイテンシ

C\_AREA\_OPTIMIZED=0 の場合 1 クロック サイクル

C\_AREA\_OPTIMIZED=1 の場合 2 クロック サイクル

## shi

Store Halfword Immediate

shi rD, rA, IMM

1	1	1	1	0	1	rD	rA	IMM
0					6	11	16	31

## 説明

レジスタ **rD** の下位ハーフワードの内容を、レジスタ **rA** と 32 ビットに符号拡張された **IMM** の値を加算した結果をハーフワードで揃えたメモリ ロケーションに格納します。

データ TLB ミス例外は、仮想保護モードがイネーブルで、アドレスに対応する有効な変換エントリが TLB で検出されない場合に発生します。仮想保護モードがイネーブルで、アクセス不可または読み出しのみのゾーン保護でアクセスが禁止されている場合は、データ格納例外が発生します。アクセス不可は、ユーザー モードでのみ使用できます。不整列データ アクセスによる例外は、アドレスの最下位ビットが 0 でない場合に発生します。

## 擬似コード

```

Addr ← (rA) + 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)[16:31]

```

## 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格納例外が発生した場合)
- ESR[EC]、ESR[S] (例外が発生した場合)
- ESR[DIZ] (データ格納例外が発生した場合)
- ESR[W]、ESR[Rx] (不整列データ アクセスによる例外が発生した場合)

## レイテンシ

C\_AREA\_OPTIMIZED=0 の場合 1 クロック サイクル

C\_AREA\_OPTIMIZED=1 の場合 2 クロック サイクル

## メモ

デフォルトでは、タイプ **B** 命令は 16 ビットの **IMM** フィールドの値を取り込み、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。タイプ **B** 命令の前に **imm** 命令を使用すると、32 ビットの即値を使用できます。32 ビットの即値の使用に関する詳細は、[166 ページの「imm」](#)を参照してください。

## sra Shift Right Arithmetic

**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																				31

### 説明

レジスタ **rA** の内容を 1 ビット右に算術シフトし、その結果を **rD** に格納します。**rA** の **MSB** (符号ビット) は、**rD** の **MSB** に配置されます。シフト チェーンから外れた最下位ビットは、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 クロック サイクル



## src Shift Right with Carry

**src** rD, rA

1	0	0	1	0	0		rD		rA		0	0	0	0	0	0	0	0	0	1	0	0	0	0	1
0						6		11		16															31

### 説明

レジスタ **rA** の内容を 1 ビット右にシフトし、その結果を **rD** に格納します。キャリー フラグもシフト チェーンにシフトされ、**rD** の **MSB** に配置されます。シフト チェーンから外れた最下位ビットは、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 クロック サイクル

# srl

Shift Right Logical

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

レジスタ **rA** の内容を 1 ビット右に論理シフトし、その結果を **rD** に格納します。0 がシフト チェーンにシフトされ、**rD** の **MSB** に配置されます。シフト チェーンから外れた最下位ビットは、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 クロック サイクル

## SW

## Store Word

**sw**                    rD, rA, rB

1	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						6		11		16		21																31

## 説明

レジスタ rD の内容を、レジスタ rA と rB の内容を加算した結果をワードで揃えたメモリ ロケーションに格納します。

データ TLB ミス例外は、仮想保護モードがイネーブルで、アドレスに対応する有効な変換エントリが TLB で検出されない場合に発生します。

仮想保護モードがイネーブルで、アクセス不可または読み出しのみのゾーン保護でアクセスが禁止されている場合は、データ格納例外が発生します。アクセス不可は、ユーザー モードでのみ使用できます。

不整列データ アクセスによる例外は、アドレスの最下位ビットが 0 でない場合に発生します。

## 擬似コード

```

Addr ← (rA) + (rB)
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)[0:31]

```

## 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格納例外が発生した場合)
- ESR[EC]、ESR[S] (例外が発生した場合)
- ESR[DIZ] (データ格納例外が発生した場合)
- ESR[W]、ESR[Rx] (不整列データ アクセスによる例外が発生した場合)

## レイテンシ

C\_AREA\_OPTIMIZED=0 の場合 1 クロック サイクル

C\_AREA\_OPTIMIZED=1 の場合 2 クロック サイクル

## SWI

### Store Word Immediate

**swi**                    rD, rA, IMM

1	1	1	1	1	0	rD	rA	IMM	
0						6	11	16	31

### 説明

レジスタ rD の内容を、レジスタ rA と 32 ビットに符号拡張された IMM の値を加算した結果をワードで揃えたメモリ ロケーションに格納します。

データ TLB ミス例外は、仮想保護モードがイネーブルで、アドレスに対応する有効な変換エントリが TLB で検出されない場合に発生します。

仮想保護モードがイネーブルで、アクセス不可または読み出しのみのゾーン保護でアクセスが禁止されている場合は、データ格納例外が発生します。アクセス不可は、ユーザー モードでのみ使用できます。

不整列データ アクセスによる例外は、アドレスの最下位ビットが 0 でない場合に発生します。

### 擬似コード

```

Addr ← (rA) + 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)[0:31]

```

### 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格納例外が発生した場合)
- ESR[EC]、ESR[S] (例外が発生した場合)
- ESR[DIZ] (データ格納例外が発生した場合)
- ESR[W]、ESR[Rx] (不整列データ アクセスによる例外が発生した場合)

### レイテンシ

C\_AREA\_OPTIMIZED=0 の場合 1 クロック サイクル

C\_AREA\_OPTIMIZED=1 の場合 2 クロック サイクル

### メモ

デフォルトでは、タイプ B 命令は 16 ビットの IMM フィールドの値を取り込み、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。タイプ B 命令の前に imm 命令を使用すると、

32 ビットの即値を使用できます。32 ビットの即値の使用に関する詳細は、[166 ページの「imm」](#)を参照してください。

## SWX

## Store Word Exclusive

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	6	11	16	21
				31

## 説明

予約ビットがセットされている場合、レジスタ **rD** の内容をレジスタ **rA** と **rB** の内容を加算した結果をワードで揃えたメモリ ロケーションに条件付で格納します。格納される場合はキャリービット (**MSR[C]**) がセットされ、格納されない場合はクリアになります。予約ビットはクリアになります。

データ TLB ミス例外は、仮想保護モードがイネーブルで、アドレスに対応する有効な変換エントリが TLB で検出されない場合に発生します。

仮想保護モードがイネーブルで、アクセス不可または読み出しのみのゾーン保護でアクセスが禁止されている場合は、データ格納例外が発生します。アクセス不可は、ユーザー モードでのみ使用できます。

不整列データ アクセスによる例外は、アドレスの最下位ビットが 0 であっても発生しません。

## 擬似コード

```

Addr  $\leftarrow$  (rA) + (rB)
if Reservation = 0 then
    MSR[C]  $\leftarrow$  1
else
    if TLB_Miss(Addr) and MSR[VM] = 1 then
        ESR[EC]  $\leftarrow$  10010; ESR[S]  $\leftarrow$  1
        MSR[UMS]  $\leftarrow$  MSR[UM]; MSR[VMS]  $\leftarrow$  MSR[VM]; MSR[UM]  $\leftarrow$  0; MSR[VM]  $\leftarrow$  0
    else if Access_Protected(Addr) and MSR[VM] = 1 then
        ESR[EC]  $\leftarrow$  10000; ESR[S]  $\leftarrow$  1; ESR[DI]  $\leftarrow$  No-access-allowed
        MSR[UMS]  $\leftarrow$  MSR[UM]; MSR[VMS]  $\leftarrow$  MSR[VM]; MSR[UM]  $\leftarrow$  0; MSR[VM]  $\leftarrow$  0
    else
        Mem(Addr)  $\leftarrow$  (rD)[0:31]; Reservation  $\leftarrow$  0; MSR[C]  $\leftarrow$  0

```

## 変更されるレジスタ

- MSR[C] (例外が発生しない場合)
- MSR[UM]、MSR[VM]、MSR[UMS]、MSR[VMS] (TLB ミス例外またはデータ格納例外が発生した場合)
- ESR[EC]、ESR[S] (例外が発生した場合)
- ESR[DIZ] (データ格納例外が発生した場合)

## レイテンシ

C\_AREA\_OPTIMIZED=0 の場合 1 クロック サイクル

C AREA OPTIMIZED=1 の場合 2 クロック サイクル

## メモ

この命令は **LDX** と共に使用して、セマフォやスピン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使用します。

## wdc

### Write to Data Cache

**wdc**                    rA,rB  
**wdc.flush**            rA,rB  
**wdc.clear**            rA,rB

1 0 0 1 0 0	0 0 0 0 0	rA	rB	0 0 0 0 1 1 F 0 1 T 0
0	6	1 1	1 6	2 7      3 1

### 説明

データ キャッシュ タグに書き込み、キャッシュ ラインを無効に、またはフラッシュします。**wdc.flush** は F ビットのセット、**wdc.clear** は T ビットのセットに使用します。

**C\_DCACHE\_USE\_WRITEBACK** が 1 に設定されている場合、F ビットがセットされていれば、キャッシュ ラインがフラッシュされ無効になります。それ以外の場合は、キャッシュ ラインが無効になり、メモリに書き込まれていないデータが破棄されます。T ビットがセットされていれば、アドレスが一致するキャッシュ ラインのみが無効になります。rB が追加されたレジスタ rA は、影響を受けたキャッシュ ラインのアドレスです。

**C\_DCACHE\_USE\_WRITEBACK** が 0 にクリアされている場合、キャッシュ ラインは常に無効になります。レジスタ rA には影響を受けたキャッシュ ラインのアドレスが含まれ、rB の値は使用されません。

MicroBlaze で MMU が使用される場合 (**C\_USE\_MMU** >= 1) この命令は特権命令になり、ユーザー モード (**MSR[UM]** = 1) で使用す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
        address ← (Ra) + (Rb)
    else
        address ← (Ra)
    if C_DCACHE_LINE_LEN = 4 then
        cacheline ← (DCache Line)[(Ra) >> 4]
        address ← address & 0xffffffff0
    if C_DCACHE_LINE_LEN = 8 then
        cacheline ← (DCache Line)[(Ra) >> 5]
        address ← address & 0xffffffe0
    if F = 1 and cacheline.Dirty then
        for i = 0 .. C_DCACHE_LINE_LEN - 1 loop
            if cacheline.Valid[i] then
                Mem(address + i * 4) ← cacheline.Data[i]
    if T = 0 then
        cacheline.Tag ← 0
    else if cacheline.Address = address then
        cacheline.Tag ← 0

```



## 変更されるレジスタ

- ESR[EC] (特権命令例外が発生した場合)

## レイテンシ

**wdc** および **wdc.clear** の場合 2 クロック サイクル

**wdc.flush** の場合  $2 + N$  クロック サイクル ( $N$  はキャッシュ ラインをメモリにフラッシュするのに必要なクロック サイクル数)

## メモ

**wdc**、**wdc.flush** および **wdc.clear** 命令は、データ キャッシュ イネーブル (MSR[DCE]) から独立して、データ キャッシュがイネーブルまたはディスエーブルされていても一緒に使用できます。

**wdc.clear** 命令は、ディレクト 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など、メモリの特定のエリアを無効にするためのものです。この命令を使用し、ほかのキャッシュ ラインが誤って無効にならないよう、まだメモリに書き込まれていないデータが誤って破棄されないようにします。

影響を受けるキャッシュ ラインのアドレスは、パラメータ **C\_USE\_MMU** や **MMU** が仮想モードまたは実モードであるかに関係しておらず、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ために **wdc.flush** をループで使用すると、ループはキャッシュ ベース アドレスとして **Ra** を、ループ カウンタとして **Rb** を使用して最適化することができます。

```

        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

```

キャッシュのメモリ エリアを無効にするために **wdc.clear** をループで使用すると、ループはメモリ エリア ベース アドレスとして **Ra** を、ループ カウンタとして **Rb** を使用して最適化することができます。

```

        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

Write to Instruction Cache

**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												11	16											31

### 説明

命令キャッシュ タグに書き込み、キャッシュ ラインを無効にします。レジスタ **rB** は使用されず、レジスタ **rA** には無効になったキャッシュ ラインのアドレスが含まれます。

MicroBlaze で MMU が使用される場合 (**C\_USE\_MMU** >= 1) この命令は特権命令になり、ユーザー モード (**MSR[UM]** = 1) で使用す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
        (ICache Line)[(Ra) >> 4].Tag ← 0
    if C_ICACHE_LINE_LEN = 8 then
        (ICache Line)[(Ra) >> 5].Tag ← 0

```

### 変更されるレジスタ

- **ESR[EC]** (特権命令例外が発生した場合)

### レイテンシ

2 クロック サイクル

### メモ

IC 命令は、命令キャッシュ イネーブル (**MSR[ICE]**) からは独立していて、命令キャッシュがイネーブルであっても、ディスエーブルであっても使用することができます。

影響を受けるキャッシュ ラインのアドレスは、パラメータ **C\_USE\_MMU** や MMU が仮想モードまたは実モードであるかに関係しておらず、常に物理アドレス (**MSR[ICE]**) です。

## XOR

Logical Exclusive 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						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 に格納します。

## 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 クロック サイクル

## xori

Logical Exclusive OR with Immediate

**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 0 を 16 ビット分追加して 32 ビットに拡張し、拡張された IMM の内容とレジスタ rA の内容を XOR 演算して、その結果をレジスタ rD に格納します。

### 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 クロック サイクル

### メモ

デフォルトでは、タイプ B 命令は 16 ビットの IMM フィールドの値を取り込み、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。タイプ B 命令の前に imm 命令を使用すると、32 ビットの即値を使用できます。32 ビットの即値の使用に関する詳細は、[166 ページの「imm」](#)を参照してください。