MicroBlaze Processor Reference Guide

Embedded Development Kit EDK 13.1

UG081 (v12.0)

XILINX®

MicroBlaze Processor Reference Guide
UG081 (v12.0)

The following table shows the revision history for this document.

Date	Version	Revision
10/01/02	1.0	Xilinx EDK 3.1 release
03/11/03	2.0	Xilinx EDK 3.2 release
09/24/03	3.0	Xilinx EDK 6.1 release
02/20/04	3.1	Xilinx EDK 6.2 release
08/24/04	4.0	Xilinx EDK 6.3 release
09/21/04	4.1	Minor corrections for EDK 6.3 SP1 release
11/18/04	4.2	Minor corrections for EDK 6.3 SP2 release
01/20/05	5.0	Xilinx EDK 7.1 release
04/02/05	5.1	Minor corrections for EDK 7.1 SP1 release
05/09/05	5.2	Minor corrections for EDK 7.1 SP2 release
10/05/05	5.3	Minor corrections for EDK 8.1 release
02/21/06	5.4	Corrections for EDK 8.1 SP2 release
06/01/06	6.0	Xilinx EDK 8.2 release
07/24/06	6.1	Minor corrections for EDK 8.2 SP1 release
08/21/06	6.2	Minor corrections for EDK 8.2 SP2 release
08/29/06	6.3	Minor corrections for EDK 8.2 SP2 release
09/15/06	7.0	Xilinx EDK 9.1 release
02/22/07	7.1	Minor corrections for EDK 9.1 SP1 release
03/27/07	7.2	Minor corrections for EDK 9.1 SP2 release

Date	Version	Revision
06/25/07	8.0	Xilinx EDK 9.2 release
10/12/07	8.1	Minor corrections for EDK 9.2 SP2 release
01/17/08	9.0	Xilinx EDK 10.1 release
03/04/08	9.1	Minor corrections for EDK 10.1 SP1 release
05/14/08	9.2	Minor corrections for EDK 10.1 SP2 release
07/14/08	9.3	Minor corrections for EDK 10.1 SP3 release
02/04/09	10.0	Xilinx EDK 11.1 release
04/15/09	10.1	Xilinx EDK 11.2 release
05/28/09	10.2	Xilinx EDK 11.3 release
10/26/09	10.3	Xilinx EDK 11.4 release
04/19/10	11.0	Xilinx EDK 12.1 release
07/23/10	11.1	Xilinx EDK 12.2 release
09/21/10	11.2	Xilinx EDK 12.3 release
11/15/10	11.3	Minor corrections for EDK 12.4 release
11/15/10	11.4	Xilinx EDK 12.4 release
03/01/11	12.0	Xilinx EDK 13.1 release

Table of Contents

Preface: About This Guide

Chapter 1: MicroBlaze Architecture

Chapter 2: MicroBlaze Signal Interface Description

Chapter 3: MicroBlaze Application Binary Interface

Chapter 4: MicroBlaze Instruction Set Architecture

About This Guide

The MicroBlaze™ Processor Reference Guide provides information about the 32-bit soft processor, MicroBlaze, which is part of the Embedded Processor Development Kit (EDK). The document is intended as a guide to the MicroBlaze hardware architecture.

Guide Contents

This guide contains the following chapters:

- Chapter 1, "MicroBlaze Architecture", contains an overview of MicroBlaze features as well as information on Big-Endian and Little-Endian bit-reversed format, 32-bit general purpose registers, cache software support, and Fast Simplex Link interfaces.
- Chapter 2, "MicroBlaze Signal Interface Description", describes the types of signal interfaces that can be used to connect MicroBlaze.
- Chapter 3, "MicroBlaze Application Binary Interface", describes the Application Binary Interface important for developing software in assembly language for the soft processor.
- Chapter 4, "MicroBlaze Instruction Set Architecture", provides notation, formats, and instructions for the Instruction Set Architecture of MicroBlaze.

For additional information, go to http://www.xilinx.com/support/mysupport.htm. The following table lists some of the resources you can access directly using the provided URLs.

Resource	Description/URL
Tutorials	Tutorials covering Xilinx® design flows, from design entry to verification and debugging. http://www.xilinx.com/support/techsup/tutorials/index.htm
Answer Browser	Database of Xilinx solution records. http://www.xilinx.com/support/answers/index.htm
Problem Solvers	Interactive tools that allow you to troubleshoot your design issues. http://www.xilinx.com/support/troubleshoot/psolvers.htm
GNU Manuals	The entire set of GNU manuals. http://www.gnu.org/manual

Conventions

This document uses the following conventions. An example illustrates each convention.

Typographical

The following typographical conventions are used in this document:

Convention	Meaning or Use	Example
`Courier font`	Messages, prompts, and program files that the system displays.	`speed grade: - 100`
`Courier bold`	Literal commands that you enter in a syntactical statement.	**`ngdbuild`** *design_name*
Helvetica bold	Commands that you select from a menu.	**File → Open**
	Keyboard shortcuts	**Ctrl+C**
Italic font	Variables in a syntax statement for which you must supply values.	**`ngdbuild`** *design_name*
	References to other manuals.	See the *Development System Reference Guide* for more information.
	Emphasis in text.	If a wire is drawn so that it overlaps the pin of a symbol, the two nets are *not* connected.
Square brackets [ ]	An optional entry or parameter. However, in bus specifications, such as **`bus[7:0]`**, they are required.	**`ngdbuild`** [*option_name*] *design_name*
Braces { }	A list of items from which you must choose one or more.	**`lowpwr ={on|off}`**
Vertical bar |	Separates items in a list of choices.	**`lowpwr ={on|off}`**

Vertical ellipsis
.
.
.
	Repetitive material that has been omitted	`IOB #1: Name = QOUT'`
		`IOB #2: Name = CLKIN'`
		.
		.
		.

Horizontal ellipsis **`...`**	Repetitive material that has been omitted	**`allow block`** *block_name loc1 loc2 ... locn;*

Online Document

The following conventions are used in this document:

Convention	Meaning or Use	Example
Blue text	Cross-reference link to a location in the current document	See the section "Additional Resources" for details. Refer to "Title Formats" in Chapter 1 for details.
Blue, underlined text	Hyperlink to a web-site (URL)	Go to http://www.xilinx.com for the latest speed files.

XILINX®

Chapter 1

MicroBlaze Architecture

This chapter contains an overview of MicroBlaze™ features and detailed information on MicroBlaze architecture including Big-Endian or Little-Endian bit-reversed format, 32-bit general purpose registers, virtual-memory management, cache software support, and Fast Simplex Link (FSL) or AXI4-Stream interfaces.

This chapter has the following sections:

Overview

The MicroBlaze™ embedded processor soft core is a reduced instruction set computer (RISC) optimized for implementation in Xilinx® Field Programmable Gate Arrays (FPGAs). Figure 1-1 shows a functional block diagram of the MicroBlaze core.

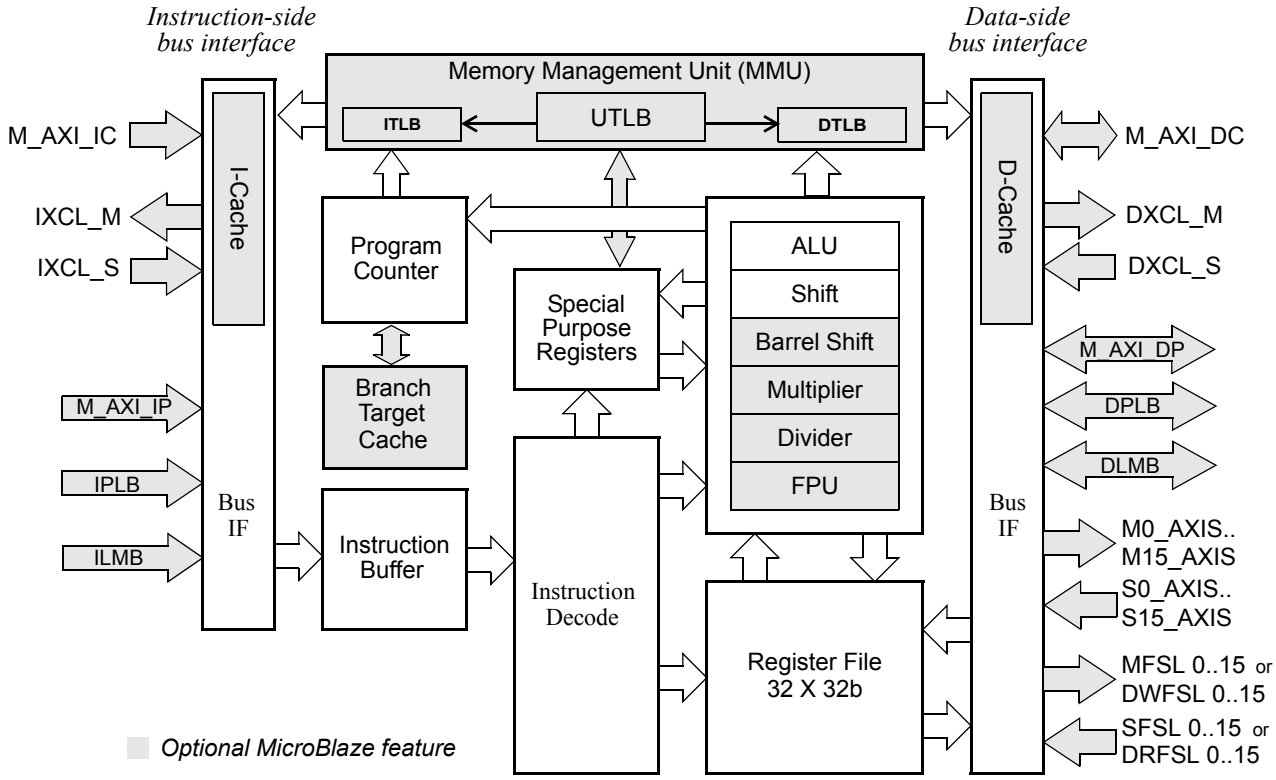


Figure 1-1: **MicroBlaze Core Block Diagram**

Features

The MicroBlaze soft core processor is highly configurable, allowing you to select a specific set of features required by your design.

The fixed feature set of the processor includes:

- Thirty-two 32-bit general purpose registers
- 32-bit instruction word with three operands and two addressing modes
- 32-bit address bus
- Single issue pipeline

In addition to these fixed features, the MicroBlaze processor is parameterized to allow selective enabling of additional functionality. Older (deprecated) versions of MicroBlaze support a subset of the optional features described in this manual. Only the latest (preferred) version of MicroBlaze (v8.00) supports all options.

Xilinx recommends that all new designs use the latest **preferred** version of the MicroBlaze processor.

Table 1-1, page 15 provides an overview of the configurable features by MicroBlaze versions.

Table 1-1: **Configurable Feature Overview by MicroBlaze Version**

Feature	MicroBlaze Versions					
	v7.00	v7.10	v7.20	v7.30	v8.00	v8.10
Version Status	obsolete	obsolete	obsolete	obsolete	deprecated	preferred
Processor pipeline depth	3/5	3/5	3/5	3/5	3/5	3/5
On-chip Peripheral Bus (OPB) data side interface	option	option	option	No	No	No
On-chip Peripheral Bus (OPB) instruction side interface	option	option	option	No	No	No
Local Memory Bus (LMB) data side interface	option	option	option	option	option	option
Local Memory Bus (LMB) instruction side interface	option	option	option	option	option	option
Hardware barrel shifter	option	option	option	option	option	option
Hardware divider	option	option	option	option	option	option
Hardware debug logic	option	option	option	option	option	option
Stream link interfaces	0-15 FSL	0-15 FSL	0-15 FSL	0-15 FSL	0-15 FSL/AXI	0-15 FSL/AXI
Machine status set and clear instructions	option	option	option	option	option	option
Instruction cache over IOPB interface	No	No	No	No	No	No
Data cache over DOPB interface	No	No	No	No	No	No
Instruction cache over Cache Link (IXCL) interface	option	option	option	option	option	option
Data cache over Cache Link (DXCL) interface	option	option	option	option	option	option
4 or 8-word cache line	option	option	option	option	option	option
Hardware exception support	option	option	option	option	option	option
Pattern compare instructions	option	option	option	option	option	option
Floating point unit (FPU)	option	option	option	option	option	option
Disable hardware multiplier[1]	option	option	option	option	option	option
Hardware debug readable ESR and EAR	Yes	Yes	Yes	Yes	Yes	Yes
Processor Version Register (PVR)	option	option	option	option	option	option
Area or speed optimized	option	option	option	option	option	option
Hardware multiplier 64-bit result	option	option	option	option	option	option
LUT cache memory	option	option	option	option	option	option
Processor Local Bus (PLB) data side interface	option	option	option	option	option	option
Processor Local Bus (PLB) instruction side interface	option	option	option	option	option	option
Floating point conversion and square root instructions	option	option	option	option	option	option
Memory Management Unit (MMU)	option	option	option	option	option	option
Extended stream instructions	option	option	option	option	option	option

Table 1-1: **Configurable Feature Overview by MicroBlaze Version**

Feature	MicroBlaze Versions					
	v7.00	v7.10	v7.20	v7.30	v8.00	v8.10
Use Xilinx Cache Link for All I-Cache Memory Accesses	-	option	option	option	option	option
Use Xilinx Cache Link for All D-Cache Memory Accesses	-	option	option	option	option	option
Use Write-back Caching Policy for D-Cache	-	-	option	option	option	option
Cache Link (DXCL) protocol for D-Cache	-	-	option	option	option	option
Cache Link (IXCL) protocol for I-Cache	-	-	option	option	option	option
Branch Target Cache (BTC)	-	-	-	option	option	option
Streams for I-Cache				option	option	option
Victim handling for I-Cache				option	option	option
Victim handling for D-Cache				option	option	option
AXI4 (M_AXI_DP) data side interface	-	-	-	-	option	option
AXI4 (M_AXI_IP) instruction side interface	-	-	-	-	option	option
AXI4 (M_AXI_DC) protocol for D-Cache	-	-	-	-	option	option
AXI4 (M_AXI_IC) protocol for I-Cache	-	-	-	-	option	option
AXI4 protocol for stream accesses	-	-	-	-	option	option
Fault tolerant features	-	-	-	-	option	option
Tool selectable endianness	-	-	-	-	option	option
Force distributed RAM for cache tags	-	-	-	-	option	option
Configurable cache data widths	-	-	-	-	option	option
Count Leading Zeros instruction	-	-	-	-	-	option
Memory Barrier instruction	-	-	-	-	-	Yes
Stack overflow and underflow detection	-	-	-	-	-	option
Allow stream instructions in user mode	-	-	-	-	-	option

1. Used in Virtex®-4 and subsequent families, for saving MUL18 and DSP48 primitives.

Data Types and Endianness

MicroBlaze uses Big-Endian or Little-Endian format to represent data, depending on the parameter C_ENDIANNESS. The hardware supported data types for MicroBlaze are word, half word, and byte. When using the reversed load and store instructions LHUR, LWR, SHR and SWR, the bytes in the data are reversed, as indicated by the byte-reversed order.

The bit and byte organization for each type is shown in the following tables.

Table 1-2: **Word Data Type**

Big-Endian Byte Address	n	n+1	n+2	n+3
Big-Endian Byte Significance	MSByte			LSByte
Big-Endian Byte Order	n	n+1	n+2	n+3
Big-Endian Byte-Reversed Order	n+3	n+2	n+1	n
Little-Endian Byte Address	n+3	n+2	n+1	n
Little-Endian Byte Significance	MSByte			LSByte
Little-Endian Byte Order	n+3	n+2	n+1	n
Little-Endian Byte-Reversed Order	n	n+1	n+2	n+3
Bit Label	0			31
Bit Significance	MSBit			LSBit

Table 1-3: **Half Word Data Type**

Big-Endian Byte Address	n	n+1
Big-Endian Byte Significance	MSByte	LSByte
Big-Endian Byte Order	n	n+1
Big-Endian Byte-Reversed Order	n+1	n
Little-Endian Byte Address	n+1	n
Little-Endian Byte Significance	MSByte	LSByte
Little-Endian Byte Order	n+1	n
Little-Endian Byte-Reversed Order	n	n+1
Bit Label	0	15
Bit Significance	MSBit	LSBit

Table 1-4: **Byte Data Type**

Byte Address	n	
Bit Label	0	7
Bit Significance	MSBit	LSBit

Instructions

Instruction Summary

All MicroBlaze instructions are 32 bits and are defined as either Type A or Type B. Type A instructions have up to two source register operands and one destination register operand. Type B instructions have one source register and a 16-bit immediate operand (which can be extended to 32 bits by preceding the Type B instruction with an imm instruction). Type B instructions have a single destination register operand. Instructions are provided in the following functional categories: arithmetic, logical, branch, load/store, and special. Table 1-6 lists the MicroBlaze instruction set. Refer to Chapter 4, "MicroBlaze Instruction Set Architecture"for more information on these instructions. Table 1-5 describes the instruction set nomenclature used in the semantics of each instruction.

Table 1-5: **Instruction Set Nomenclature**

Symbol	Description
Ra	R0 - R31, General Purpose Register, source operand a
Rb	R0 - R31, General Purpose Register, source operand b
Rd	R0 - R31, General Purpose Register, destination operand
SPR[*x*]	Special Purpose Register number *x*
MSR	Machine Status Register = SPR[1]
ESR	Exception Status Register = SPR[5]
EAR	Exception Address Register = SPR[3]
FSR	Floating Point Unit Status Register = SPR[7]
PVR*x*	Processor Version Register, where *x* is the register number = SPR[8192 + *x*]
BTR	Branch Target Register = SPR[11]
PC	Execute stage Program Counter = SPR[0]
x[*y*]	Bit *y* of register *x*
x[*y*:*z*]	Bit range *y* to *z* of register *x*
$\overline{x}$	Bit inverted value of register *x*
Imm	16 bit immediate value
Imm*x*	*x* bit immediate value
FSL*x*	4 bit Fast Simplex Link (FSL) or AXI4-Stream port designator, where *x* is the port number
C	Carry flag, MSR[29]
Sa	Special Purpose Register, source operand
Sd	Special Purpose Register, destination operand
s(*x*)	Sign extend argument *x* to 32-bit value
*Addr	Memory contents at location Addr (data-size aligned)
:=	Assignment operator

Table 1-5: **Instruction Set Nomenclature *(Continued)***

Symbol	Description
=	Equality comparison
!=	Inequality comparison
>	Greater than comparison
>=	Greater than or equal comparison
<	Less than comparison
<=	Less than or equal comparison
+	Arithmetic add
*	Arithmetic multiply
/	Arithmetic divide
>> *x*	Bit shift right *x* bits
<< *x*	Bit shift left *x* bits
and	Logic AND
or	Logic OR
xor	Logic exclusive OR
op1 if cond else op2	Perform *op1* if condition *cond* is true, else perform *op2*
&	Concatenate. E.g. “0000100 & Imm7” is the concatenation of the fixed field “0000100” and a 7 bit immediate value.
signed	Operation performed on signed integer data type. All arithmetic operations are performed on signed word operands, unless otherwise specified
unsigned	Operation performed on unsigned integer data type
float	Operation performed on floating point data type
clz(r)	Cound leading zeros

Table 1-6: **MicroBlaze Instruction Set Summary**

Type A	0-5	6-10	11-15	16-20	21-31	Semantics
Type B	0-5	6-10	11-15	16-31		
ADD Rd,Ra,Rb	000000	Rd	Ra	Rb	00000000000	Rd := Rb + Ra
RSUB Rd,Ra,Rb	000001	Rd	Ra	Rb	00000000000	Rd := Rb + $\overline{Ra}$ + 1
ADDC Rd,Ra,Rb	000010	Rd	Ra	Rb	00000000000	Rd := Rb + Ra + C
RSUBC Rd,Ra,Rb	000011	Rd	Ra	Rb	00000000000	Rd := Rb + $\overline{Ra}$ + C
ADDK Rd,Ra,Rb	000100	Rd	Ra	Rb	00000000000	Rd := Rb + Ra
RSUBK Rd,Ra,Rb	000101	Rd	Ra	Rb	00000000000	Rd := Rb + $\overline{Ra}$ + 1
ADDKC Rd,Ra,Rb	000110	Rd	Ra	Rb	00000000000	Rd := Rb + Ra + C

Table 1-6: **MicroBlaze Instruction Set Summary** ***(Continued)***

Type A	0-5	6-10	11-15	16-20	21-31	Semantics
Type B	**0-5**	**6-10**	**11-15**	**16-31**		
RSUBKC Rd,Ra,Rb	000111	Rd	Ra	Rb	00000000000	Rd := Rb + $\overline{Ra}$ + C
CMP Rd,Ra,Rb	000101	Rd	Ra	Rb	00000000001	Rd := Rb + $\overline{Ra}$ + 1 Rd[0] := 0 if (Rb >= Ra) else Rd[0] := 1
CMPU Rd,Ra,Rb	000101	Rd	Ra	Rb	00000000011	Rd := Rb + $\overline{Ra}$ + 1 (unsigned) Rd[0] := 0 if (Rb >= Ra, 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	00000000000	Rd := Ra * Rb
MULH Rd,Ra,Rb	010000	Rd	Ra	Rb	00000000001	Rd := (Ra * Rb) >> 32 (signed)
MULHU Rd,Ra,Rb	010000	Rd	Ra	Rb	00000000011	Rd := (Ra * Rb) >> 32 (unsigned)
MULHSU Rd,Ra,Rb	010000	Rd	Ra	Rb	00000000010	Rd := (Ra, signed * Rb, unsigned) >> 32 (signed)
BSRA Rd,Ra,Rb	010001	Rd	Ra	Rb	01000000000	Rd := s(Ra >> Rb)
BSLL Rd,Ra,Rb	010001	Rd	Ra	Rb	10000000000	Rd := (Ra << Rb) & 0
MULI Rd,Ra,Imm	011000	Rd	Ra	Imm		Rd := Ra * s(Imm)
BSRLI Rd,Ra,Imm	011001	Rd	Ra	00000000000 & Imm5		Rd : = 0 & (Ra >> Imm5)
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
*TNEA*GETD Rd,Rb	010011	Rd	00000	Rb	0*N*0*TAE* 00000	Rd := FSL Rb[28:31] (data read) MSR[FSL] := 1 if (FSL_S_Control = 1) MSR[C] := not FSL_S_Exists if *N* = 1

Table 1-6: **MicroBlaze Instruction Set Summary *(Continued)***

Type A	0-5	6-10	11-15	16-20	21-31	Semantics
Type B	0-5	6-10	11-15	16-31		
*TNA*PUTD Ra,Rb	010011	00000	Ra	Rb	0*N*0*TA*0 00000	FSL Rb[28:31] := Ra (data write) MSR[C] := FSL_M_Full if *N* = 1
*TNECA*GETD Rd,Rb	010011	Rd	00000	Rb	0*N*1*TAE* 00000	Rd := FSL Rb[28:31] (control read) MSR[FSL] := 1 if (FSL_S_Control = 0) MSR[C] := not FSL_S_Exists if *N* = 1
*TNCA*PUTD Ra,Rb	010011	00000	Ra	Rb	0*N*1*TA*0 00000	FSL Rb[28:31] := Ra (control write) MSR[C] := FSL_M_Full if *N* = 1
FADD Rd,Ra,Rb	010110	Rd	Ra	Rb	00000000000	Rd := Rb+Ra, float[1]
FRSUB Rd,Ra,Rb	010110	Rd	Ra	Rb	00010000000	Rd := Rb-Ra, float[1]
FMUL Rd,Ra,Rb	010110	Rd	Ra	Rb	00100000000	Rd := Rb*Ra, float[1]
FDIV Rd,Ra,Rb	010110	Rd	Ra	Rb	00110000000	Rd := Rb/Ra, float[1]
FCMP.UN Rd,Ra,Rb	010110	Rd	Ra	Rb	01000000000	Rd := 1 if (Rb = NaN or Ra = NaN, float[1]) else Rd := 0
FCMP.LT Rd,Ra,Rb	010110	Rd	Ra	Rb	01000010000	Rd := 1 if (Rb < Ra, float[1]) else Rd := 0
FCMP.EQ Rd,Ra,Rb	010110	Rd	Ra	Rb	01000100000	Rd := 1 if (Rb = Ra, float[1]) else Rd := 0
FCMP.LE Rd,Ra,Rb	010110	Rd	Ra	Rb	01000110000	Rd := 1 if (Rb <= Ra, float[1]) else Rd := 0
FCMP.GT Rd,Ra,Rb	010110	Rd	Ra	Rb	01001000000	Rd := 1 if (Rb > Ra, float[1]) else Rd := 0
FCMP.NE Rd,Ra,Rb	010110	Rd	Ra	Rb	01001010000	Rd := 1 if (Rb != Ra, float[1]) else Rd := 0
FCMP.GE Rd,Ra,Rb	010110	Rd	Ra	Rb	01001100000	Rd := 1 if (Rb >= Ra, float[1]) else Rd := 0
FLT Rd,Ra	010110	Rd	Ra	0	01010000000	Rd := float (Ra)[1]
FINT Rd,Ra	010110	Rd	Ra	0	01100000000	Rd := int (Ra)[1]
FSQRT Rd,Ra	010110	Rd	Ra	0	01110000000	Rd := sqrt (Ra)[1]
*TNEA*GET Rd,FSLx	011011	Rd	00000	0*N*0*TAE*000000 & FSLx		Rd := FSLx (data read, blocking if *N* = 0) MSR[FSL] := 1 if (FSL*x*_S_Control = 1) MSR[C] := not FSL*x*_S_Exists if *N* = 1
*TNA*PUT Ra,FSLx	011011	00000	Ra	1*N*0*TA*0000000 & FSLx		FSLx := Ra (data write, blocking if *N* = 0) MSR[C] := FSL*x*_M_Full if *N* = 1
*TNECA*GET Rd,FSLx	011011	Rd	00000	0*N*1*TAE*000000 & FSLx		Rd := FSLx (control read, blocking if *N* = 0) MSR[FSL] := 1 if (FSL*x*_S_Control = 0) MSR[C] := not FSL*x*_S_Exists if *N* = 1

Table 1-6: **MicroBlaze Instruction Set Summary *(Continued)***

Type A	0-5	6-10	11-15	16-20	21-31	Semantics
Type B	0-5	6-10	11-15	16-31		
*TNCA*PUT Ra,FSLx	011011	00000	Ra	1*N*1*TA*0000000 & FSLx		FSLx := Ra (control write, blocking if *N* = 0) MSR[C] := FSL*x*_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	0000000000000001		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])
CLZ Rd, Ra	100100	Rd	Ra	0000000011100000		Rd = clz(Ra)
WIC Ra,Rb	100100	00000	Ra	Rb	00001101000	ICache_Line[Ra >> 4].Tag := 0 if (C_ICACHE_LINE_LEN = 4) ICache_Line[Ra >> 5].Tag := 0 if (C_ICACHE_LINE_LEN = 8)
WDC Ra,Rb	100100	00000	Ra	Rb	00001100100	Cache line is cleared, discarding stored data. 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	00001110100	Cache line is flushed, writing stored data to memory, and then cleared. Used when C_DCACHE_USE_WRITEBACK = 1.

Table 1-6: **MicroBlaze Instruction Set Summary *(Continued)***

Type A	0-5	6-10	11-15	16-20	21-31	Semantics
Type B	**0-5**	**6-10**	**11-15**	**16-31**		
WDC.CLEAR Ra,Rb	100100	00000	Ra	Rb	00001110110	Cache line with matching address is cleared, discarding stored data. Used when C_DCACHE_USE_WRITEBACK = 1.
MBAR Imm	101110	Imm	00000	0000000000000100		PC := PC + 4; Wait for memory accesses.
MTS Sd,Ra	100101	00000	Ra	11 & Sd		SPR[Sd] := Ra, where: • SPR[0x0001] is MSR • SPR[0x0007] is FSR • SPR[0x0800] is SLR • SPR[0x0802] is SHR • SPR[0x1000] is PID • SPR[0x1001] is ZPR • SPR[0x1002] is TLBX • SPR[0x1003] is TLBLO • SPR[0x1004] is TLBHI • SPR[0x1005] is TLBSX
MFS Rd,Sa	100101	Rd	00000	10 & Sa		Rd := SPR[Sa], where: • SPR[0x0000] is PC • SPR[0x0001] is MSR • SPR[0x0003] is EAR • SPR[0x0005] is ESR • SPR[0x0007] is FSR • SPR[0x000B] is BTR • SPR[0x000D] is EDR • SPR[0x0800] is SLR • SPR[0x0802] is SHR • SPR[0x1000] is PID • SPR[0x1001] is ZPR • SPR[0x1002] is TLBX • SPR[0x1003] is TLBLO • SPR[0x1004] is TLBHI • SPR[0x2000 to 0x200B] is PVR[0 to 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	00000000000	PC := PC + Rb
BRD Rb	100110	00000	10000	Rb	00000000000	PC := PC + Rb
BRLD Rd,Rb	100110	Rd	10100	Rb	00000000000	PC := PC + Rb Rd := PC

Table 1-6: **MicroBlaze Instruction Set Summary** ***(Continued)***

Type A	0-5	6-10	11-15	16-20	21-31	Semantics
Type B	0-5	6-10	11-15	16-31		
BRA Rb	100110	00000	01000	Rb	00000000000	PC := Rb
BRAD Rb	100110	00000	11000	Rb	00000000000	PC := Rb
BRALD Rd,Rb	100110	Rd	11100	Rb	00000000000	PC := Rb Rd := PC
BRK Rd,Rb	100110	Rd	01100	Rb	00000000000	PC := Rb Rd := PC MSR[BIP] := 1
BEQ Ra,Rb	100111	00000	Ra	Rb	00000000000	PC := PC + Rb if Ra = 0
BNE Ra,Rb	100111	00001	Ra	Rb	00000000000	PC := PC + Rb if Ra != 0
BLT Ra,Rb	100111	00010	Ra	Rb	00000000000	PC := PC + Rb if Ra < 0
BLE Ra,Rb	100111	00011	Ra	Rb	00000000000	PC := PC + Rb if Ra <= 0
BGT Ra,Rb	100111	00100	Ra	Rb	00000000000	PC := PC + Rb if Ra > 0
BGE Ra,Rb	100111	00101	Ra	Rb	00000000000	PC := PC + Rb if Ra >= 0
BEQD Ra,Rb	100111	10000	Ra	Rb	00000000000	PC := PC + Rb if Ra = 0
BNED Ra,Rb	100111	10001	Ra	Rb	00000000000	PC := PC + Rb if Ra != 0
BLTD Ra,Rb	100111	10010	Ra	Rb	00000000000	PC := PC + Rb if Ra < 0
BLED Ra,Rb	100111	10011	Ra	Rb	00000000000	PC := PC + Rb if Ra <= 0
BGTD Ra,Rb	100111	10100	Ra	Rb	00000000000	PC := PC + Rb if Ra > 0
BGED Ra,Rb	100111	10101	Ra	Rb	00000000000	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 $\overline{\text{s(Imm)}}$
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)

Table 1-6: **MicroBlaze Instruction Set Summary *(Continued)***

Type A	0-5	6-10	11-15	16-20	21-31	Semantics
Type B	0-5	6-10	11-15	16-31		
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
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 LBUR Rd,Ra,Rb	110000	Rd	Ra	Rb	00000000000 01000000000	Addr := Ra + Rb Rd[0:23] := 0 Rd[24:31] := *Addr[0:7]
LHU Rd,Ra,Rb LHUR Rd,Ra,Rb	110001	Rd	Ra	Rb	00000000000 01000000000	Addr := Ra + Rb Rd[0:15] := 0 Rd[16:31] := *Addr[0:15]
LW Rd,Ra,Rb LWR Rd,Ra,Rb	110010	Rd	Ra	Rb	00000000000 01000000000	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 SBR Rd,Ra,Rb	110100	Rd	Ra	Rb	00000000000 01000000000	Addr := Ra + Rb *Addr[0:8] := Rd[24:31]

Table 1-6: **MicroBlaze Instruction Set Summary *(Continued)***

Type A	0-5	6-10	11-15	16-20	21-31	Semantics
Type B	0-5	6-10	11-15	16-31		
SH Rd,Ra,Rb SHR Rd,Ra,Rb	110101	Rd	Ra	Rb	00000000000 01000000000	Addr := Ra + Rb *Addr[0:16] := Rd[16:31]
SW Rd,Ra,Rb SWR Rd,Ra,Rb	110110	Rd	Ra	Rb	00000000000 01000000000	Addr := Ra + Rb *Addr := Rd
SWX Rd,Ra,Rb	110110	Rd	Ra	Rb	10000000000	Addr := Ra + Rb *Addr := Rd if Reservation = 1 Reservation := 0
LBUI Rd,Ra,Imm	111000	Rd	Ra	Imm		Addr := Ra + s(Imm) Rd[0:23] := 0 Rd[24:31] := *Addr[0:7]
LHUI Rd,Ra,Imm	111001	Rd	Ra	Imm		Addr := Ra + s(Imm) Rd[0:15] := 0 Rd[16:31] := *Addr[0:15]
LWI Rd,Ra,Imm	111010	Rd	Ra	Imm		Addr := Ra + s(Imm) Rd := *Addr
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. Due to the many different corner cases involved in floating point arithmetic, only the normal behavior is described. A full description of the behavior can be found in Chapter 4, “MicroBlaze Instruction Set Architecture.”

Semaphore Synchronization

The LWX and SWX. instructions are used to implement common semaphore operations, including test and set, compare and swap, exchange memory, and fetch and add. They are also used to implement spinlocks.

These instructions are typically used by system programs and are called by application programs as needed. Generally, a program uses LWX to load a semaphore from memory, causing the reservation to be set (the processor maintains the reservation internally). The program can compute a result based on the semaphore value and conditionally store the result back to the same memory location using the SWX instruction. The conditional store is performed based on the existence of the reservation established by the preceding LWX instruction. If the reservation exists when the store is executed, the store is performed and MSR[C] is cleared to 0. If the reservation does not exist when the store is executed, the target memory location is not modified and MSR[C] is set to 1.

If the store is successful, the sequence of instructions from the semaphore load to the semaphore store appear to be executed atomically—no other device modified the semaphore location between the read and the update. Other devices can read from the semaphore location during the operation. For a semaphore operation to work properly, the LWX instruction must be paired with an SWX instruction, and both must specify identical addresses. The reservation granularity in MicroBlaze is

a word. For both instructions, the address must be word aligned. No unaligned exceptions are generated for these instructions.

The conditional store is always performed when a reservation exists, even if the store address does not match the load address that set the reservation.

Only one reservation can be maintained at a time. The address associated with the reservation can be changed by executing a subsequent LWX instruction. The conditional store is performed based upon the reservation established by the last LWX instruction executed. Executing an SWX instruction always clears a reservation held by the processor, whether the address matches that established by the LWX or not.

Reset, interrupts, exceptions, and breaks (including the BRK and BRKI instructions) all clear the reservation.

The following provides general guidelines for using the LWX and SWX instructions:

- The LWX and SWX instructions should be paired and use the same address.
- An unpaired SWX instruction to an arbitrary address can be used to clear any reservation held by the processor.
- A conditional sequence begins with an LWX instruction. It can be followed by memory accesses and/or computations on the loaded value. The sequence ends with an SWX instruction. In most cases, failure of the SWX instruction should cause a branch back to the LWX for a repeated attempt.
- An LWX instruction can be left unpaired when executing certain synchronization primitives if the value loaded by the LWX is not zero. An implementation of Test and Set exemplifies this:

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

- Performance can be improved by minimizing looping on an LWX instruction that fails to return a desired value. Performance can also be improved by using an ordinary load instruction to do the initial value check. An implementation of a spinlock exemplifies this:

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

- Minimizing the looping on an LWX/SWX instruction pair increases the likelihood that forward progress is made. The old value should be tested before attempting the store. If the order is reversed (store before load), more SWX instructions are executed and reservations are more likely to be lost between the LWX and SWX instructions.

Registers

MicroBlaze has an orthogonal instruction set architecture. It has thirty-two 32-bit general purpose registers and up to eighteen 32-bit special purpose registers, depending on configured options.

General Purpose Registers

The thirty-two 32-bit General Purpose Registers are numbered R0 through R31. The register file is reset on bit stream download (reset value is 0x00000000). Figure 1-2 is a representation of a General Purpose Register and Table 1-7 provides a description of each register and the register reset value (if existing).

Note: The register file is **not** reset by the external reset inputs: `Reset` and `Debug_Rst`.

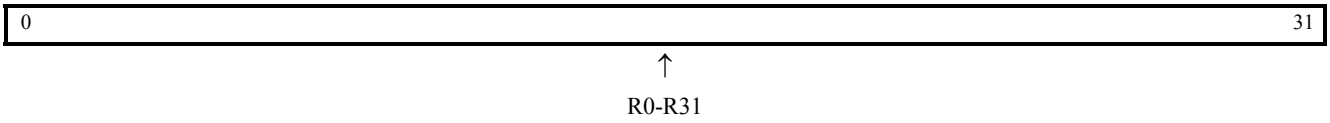


Figure 1-2: **R0-R31**

Table 1-7: **General Purpose Registers (R0-R31)**

Bits	Name	Description	Reset Value
0:31	R0	Always has a value of zero. Anything written to R0 is discarded	0x00000000
0:31	R1 through R13	32-bit general purpose registers	-
0:31	R14	32-bit register used to store return addresses for interrupts.	-
0:31	R15	32-bit general purpose register. Recommended for storing return addresses for user vectors.	-
0:31	R16	32-bit register used to store return addresses for breaks.	-
0:31	R17	If MicroBlaze is configured to support hardware exceptions, this register is loaded with the address of the instruction following the instruction causing the HW exception, except for exceptions in delay slots that use BTR instead (see "Branch Target Register (BTR)"); if not, it is a general purpose register.	-
0:31	R18 through R31	R18 through R31 are 32-bit general purpose registers.	-

Refer to Table 3-2 for software conventions on general purpose register usage.

Special Purpose Registers

Program Counter (PC)

The Program Counter (PC) is the 32-bit address of the execution instruction. It can be read with an MFS instruction, but it cannot be written with an MTS instruction. When used with the MFS instruction the PC register is specified by setting Sa = 0x0000. Figure 1-3 illustrates the PC and Table 1-8 provides a description and reset value.

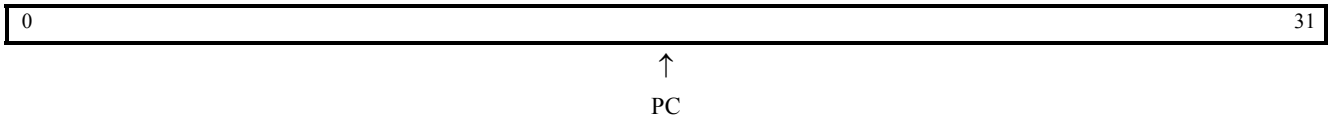


Figure 1-3: **PC**

Table 1-8: **Program Counter (PC)**

Bits	Name	Description	Reset Value
0:31	PC	Program Counter Address of executing instruction, that is, “mfs r2 0” stores the address of the mfs instruction itself in R2.	0x00000000

Machine Status Register (MSR)

The Machine Status Register contains control and status bits for the processor. It can be read with an MFS instruction. When reading the MSR, bit 29 is replicated in bit 0 as the carry copy. MSR can be written using either an MTS instruction or the dedicated MSRSET and MSRCLR instructions.

When writing to the MSR using MSRSET or MSRCLR, the Carry bit takes effect immediately and the remaining bits take effect one clock cycle later. When writing using MTS, all bits take effect one clock cycle later. Any value written to bit 0 is discarded.

When used with an MTS or MFS instruction, the MSR is specified by setting S*x* = 0x0001. Figure 1-4 illustrates the MSR register and Table 1-9 provides the bit description and reset values.

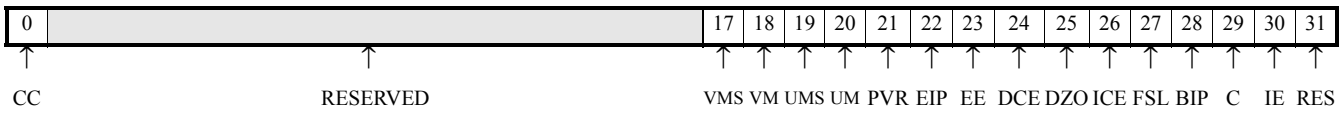


Figure 1-4: **MSR**

Table 1-9: **Machine Status Register (MSR)**

Bits	Name	Description	Reset Value
0	CC	Arithmetic Carry Copy Copy of the Arithmetic Carry (bit 29). CC is always the same as bit C.	0
1:16	Reserved		
17	VMS	Virtual Protected Mode Save Only available when configured with an MMU (if C_USE_MMU > 1 and C_AREA_OPTIMIZED = 0) Read/Write	0
18	VM	Virtual Protected Mode 0 = MMU address translation and access protection disabled, with C_USE_MMU = 3. Access protection disabled, with C_USE_MMU = 2. 1 = MMU address translation and access protection enabled, with C_USE_MMU = 3. Access protection enabled, with C_USE_MMU = 2. Only available when configured with an MMU (if C_USE_MMU > 1 and C_AREA_OPTIMIZED = 0) Read/Write	0
19	UMS	User Mode Save Only available when configured with an MMU (if C_USE_MMU > 0 and C_AREA_OPTIMIZED = 0) Read/Write	0
20	UM	User Mode 0 = Privileged Mode, all instructions are allowed 1 = User Mode, certain instructions are not allowed Only available when configured with an MMU (if C_USE_MMU > 0 and C_AREA_OPTIMIZED = 0) Read/Write	0
21	PVR	Processor Version Register exists 0 = No Processor Version Register 1 = Processor Version Register exists Read only	Based on parameter C_PVR
22	EIP	Exception In Progress 0 = No hardware exception in progress 1 = Hardware exception in progress Only available if configured with exception support (C_*_EXCEPTION or C_USE_MMU) Read/Write	0

Table 1-9: **Machine Status Register (MSR) *(Continued)***

Bits	Name	Description	Reset Value
23	EE	Exception Enable 0 = Hardware exceptions disabled[1] 1 = Hardware exceptions enabled Only available if configured with exception support (C_*_EXCEPTION or C_USE_MMU) Read/Write	0
24	DCE	Data Cache Enable 0 = Data Cache disabled 1 = Data Cache enabled Only available if configured to use data cache (C_USE_DCACHE = 1) Read/Write	0
25	DZO	Division by Zero or Division Overflow[2] 0 = No division by zero or division overflow has occurred 1 = Division by zero or division overflow has occurred Only available if configured to use hardware divider (C_USE_DIV = 1) Read/Write	0
26	ICE	Instruction Cache Enable 0 = Instruction Cache disabled 1 = Instruction Cache enabled Only available if configured to use instruction cache (C_USE_ICACHE = 1) Read/Write	0
27	FSL	Stream (FSL or AXI) Error 0 = get/getd/put/putd had no error 1 = get/getd/put/putd control type mismatch Only available if configured to use stream links (C_FSL_LINKS > 0) Read/Write	0
28	BIP	Break in Progress 0 = No Break in Progress 1 = Break in Progress Break Sources can be software break instruction or hardware break from Ext_Brk or Ext_NM_Brk pin. Read/Write	0

Table 1-9: **Machine Status Register (MSR) *(Continued)***

Bits	Name	Description	Reset Value
29	C	Arithmetic Carry 0 = No Carry (Borrow) 1 = Carry (No Borrow) Read/Write	0
30	IE	Interrupt Enable 0 = Interrupts disabled 1 = Interrupts enabled Read/Write	0
31	-	Reserved	0

1. The MMU exceptions (Data Storage Exception, Instruction Storage Exception, Data TLB Miss Exception, Instruction TLB Miss Exception) cannot be disabled, and are not affected by this bit.
2. This bit is only used for integer divide-by-zero or divide overflow signaling. There is a floating point equivalent in the FSR. The DZO-bit flags divide by zero or divide overflow conditions regardless if the processor is configured with exception handling or not.

Exception Address Register (EAR)

The Exception Address Register stores the full load/store address that caused the exception for the following:

- An unaligned access exception that means the unaligned access address
- A DPLB or M_AXI_DP exception that specifies the failing PLB or AXI4 data access address
- A data storage exception that specifies the (virtual) effective address accessed
- An instruction storage exception that specifies the (virtual) effective address read
- A data TLB miss exception that specifies the (virtual) effective address accessed
- An instruction TLB miss exception that specifies the (virtual) effective address read

The contents of this register is undefined for all other exceptions. When read with the MFS instruction, the EAR is specified by setting Sa = 0x0003. The EAR register is illustrated in Figure 1-5 and Table 1-10 provides bit descriptions and reset values.

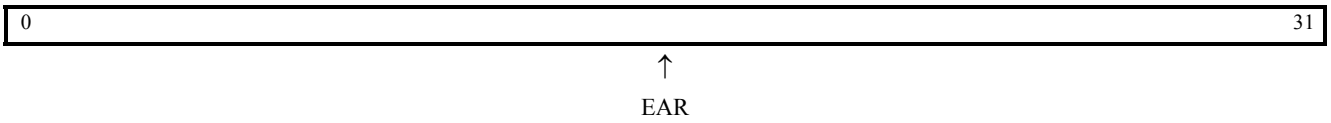


Figure 1-5: **EAR**

Table 1-10: **Exception Address Register (EAR)**

Bits	Name	Description	Reset Value
0:31	EAR	Exception Address Register	0x00000000

Exception Status Register (ESR)

The Exception Status Register contains status bits for the processor. When read with the MFS instruction, the ESR is specified by setting Sa = 0x0005. The ESR register is illustrated in Figure 1-6, Table 1-11 provides bit descriptions and reset values, and Table 1-12 provides the Exception Specific Status (ESS).

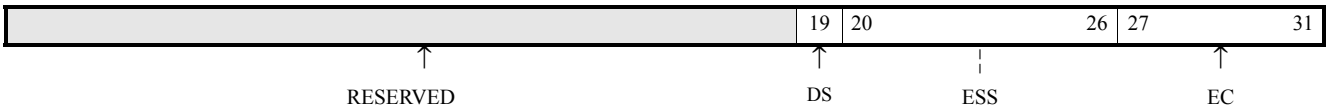


Figure 1-6: **ESR**

Table 1-11: **Exception Status Register (ESR)**

Bits	Name	Description	Reset Value
0:18	Reserved		
19	DS	Delay Slot Exception. 0 = not caused by delay slot instruction 1 = caused by delay slot instruction Read-only	0
20:26	ESS	Exception Specific Status For details refer to Table 1-12. Read-only	See Table 1-12
27:31	EC	Exception Cause 00000 = Stream exception 00001 = Unaligned data access exception 00010 = Illegal op-code exception 00011 = Instruction bus error exception 00100 = Data bus error exception 00101 = Divide exception 00110 = Floating point unit exception 00111 = Privileged instruction exception 00111 = Stack protection violation exception 10000 = Data storage exception 10001 = Instruction storage exception 10010 = Data TLB miss exception 10011 = Instruction TLB miss exception Read-only	0

Table 1-12: **Exception Specific Status (ESS)**

Exception Cause	Bits	Name	Description	Reset Value
Unaligned Data Access	20	W	Word Access Exception 0 = unaligned halfword access 1 = unaligned word access	0
	21	S	Store Access Exception 0 = unaligned load access 1 = unaligned store access	0
	22:26	Rx	Source/Destination Register General purpose register used as source (Store) or destination (Load) in unaligned access	0
Illegal Instruction	20:26	Reserved		0
Instruction bus error	20:	ECC	Exception caused by ILMB correctable or uncorrectable error	0
	21:26	Reserved		0
Data bus error	20	ECC	Exception caused by DLMB correctable or uncorrectable error	0
	21:26	Reserved		0
Divide	20	DEC	Divide - Division exception cause 0 = Divide-By-Zero 1 = Division Overflow	0
	21:26	Reserved		0
Floating point unit	20:26	Reserved		0
Privileged instruction	20:26	Reserved		0
Stack protection violation	20:26	Reserved		0
Stream	20:22	Reserved		0
	23:26	FSL	Stream (FSL or AXI) index that caused the exception	0
Data storage	20	DIZ	Data storage - Zone protection 0 = Did not occur 1 = Occurred	0
	21	S	Data storage - Store instruction 0 = Did not occur 1 = Occurred	0
	22:26	Reserved		0

Table 1-12: **Exception Specific Status (ESS) *(Continued)***

Exception Cause	Bits	Name	Description	Reset Value
Instruction storage	20	DIZ	Instruction storage - Zone protection 0 = Did not occur 1 = Occurred	0
	21:26	Reserved		0
Data TLB miss	20	Reserved		0
	21	S	Data TLB miss - Store instruction 0 = Did not occur 1 = Occurred	0
	22:26	Reserved		0
Instruction TLB miss	20:26	Reserved		0

Branch Target Register (BTR)

The Branch Target Register only exists if the MicroBlaze processor is configured to use exceptions. The register stores the branch target address for all delay slot branch instructions executed while MSR[EIP] = 0. If an exception is caused by an instruction in a delay slot (that is, ESR[DS]=1), the exception handler should return execution to the address stored in BTR instead of the normal exception return address stored in R17. When read with the MFS instruction, the BTR is specified by setting Sa = 0x000B. The BTR register is illustrated in Figure 1-7 and Table 1-13 provides bit descriptions and reset values.

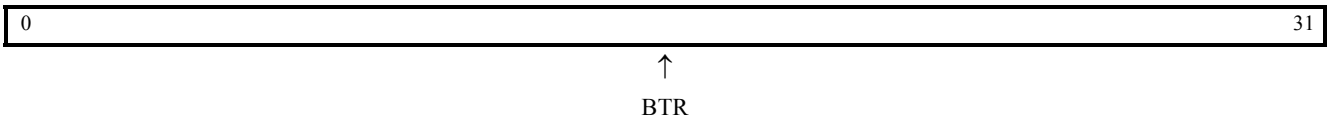


Figure 1-7: **BTR**

Table 1-13: **Branch Target Register (BTR)**

Bits	Name	Description	Reset Value
0:31	BTR	Branch target address used by handler when returning from an exception caused by an instruction in a delay slot. Read-only	0x00000000

Floating Point Status Register (FSR)

The Floating Point Status Register contains status bits for the floating point unit. It can be read with an MFS, and written with an MTS instruction. When read or written, the register is specified by setting Sa = 0x0007. The bits in this register are sticky – floating point instructions can only set bits in the register, and the only way to clear the register is by using the MTS instruction. Figure 1-8 illustrates the FSR register and Table 1-14 provides bit descriptions and reset values.

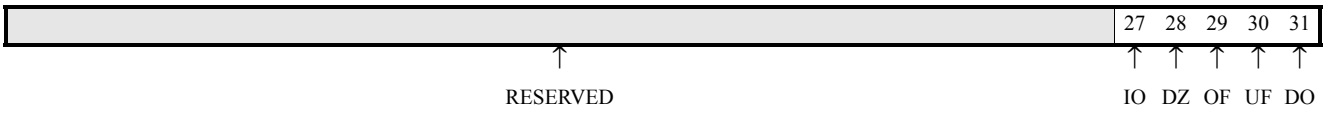


Figure 1-8: **FSR**

Table 1-14: **Floating Point Status Register (FSR)**

Bits	Name	Description	Reset Value
0:26	Reserved		undefined
27	IO	Invalid operation	0
28	DZ	Divide-by-zero	0
29	OF	Overflow	0
30	UF	Underflow	0
31	DO	Denormalized operand error	0

Exception Data Register (EDR)

The Exception Data Register stores data read on a stream link (FSL or AXI) that caused a stream exception.

The contents of this register is undefined for all other exceptions. When read with the MFS instruction, the EDR is specified by setting Sa = 0x000D. Figure 1-9 illustrates the EDR register and Table 1-15 provides bit descriptions and reset values.

Note: The register is only implemented if `C_FSL_LINKS` is greater than 0 and `C_FSL_EXCEPTION` is set to 1.

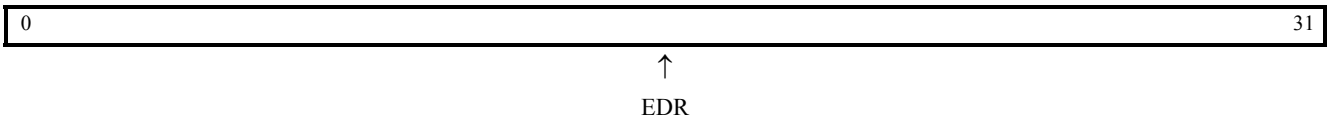


Figure 1-9: **EDR**

Table 1-15: **Exception Data Register (EDR)**

Bits	Name	Description	Reset Value
0:31	EDR	Exception Data Register	0x00000000

Stack Low Register (SLR)

The Stack Low Register stores the stack low limit use to detect stack overflow. When the address of a load or store instruction using the stack pointer (register R1) as rA is less than the Stack Low Register, a stack overflow occurs, causing a Stack Protection Violation exception if exceptions are enabled in MSR.

When read with the MFS instruction, the SLR is specified by setting Sa = 0x0800. Figure 1-10 illustrates the SLR register and Table 1-16 provides bit descriptions and reset values.

Note: The register is only implemented if `C_USE_STACK_PROTECTION` is set to 1.

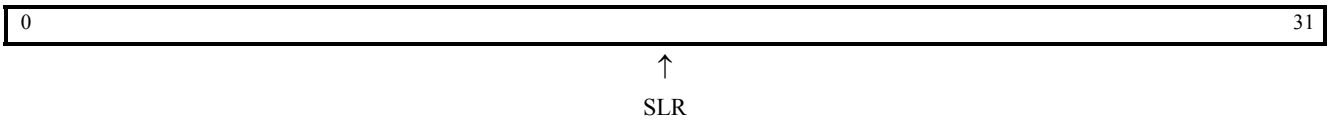


Figure 1-10: **SLR**

Table 1-16: **Stack Low Register (SLR)**

Bits	Name	Description	Reset Value
0:31	SLR	Stack Low Register	0x00000000

Stack High Register (SHR)

The Stack High Register stores the stack high limit use to detect stack underflow. When the address of a load or store instruction using the stack pointer (register R1) as rA is greater than the Stack High Register, a stack underflow occurs, causing a Stack Protection Violation exception if exceptions are enabled in MSR.

When read with the MFS instruction, the SHR is specified by setting Sa = 0x0802. Figure 1-11 illustrates the SHR register and Table 1-17 provides bit descriptions and reset values.

Note: The register is only implemented if `C_USE_STACK_PROTECTION` is set to 1.

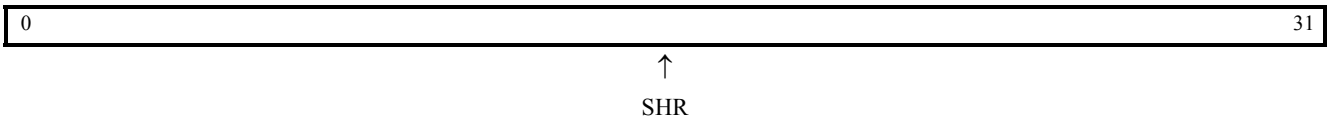


Figure 1-11: **SHR**

Table 1-17: **Stack High Register (SHR)**

Bits	Name	Description	Reset Value
0:31	SHR	Stack High Register	0xFFFFFFFF

Process Identifier Register (PID)

The Process Identifier Register is used to uniquely identify a software process during MMU address translation. It is controlled by the C_USE_MMU configuration option on MicroBlaze. The register is only implemented if C_USE_MMU is greater than 1 and C_AREA_OPTIMIZED is set to 0. When accessed with the MFS and MTS instructions, the PID is specified by setting Sa = 0x1000. The register is accessible according to the memory management special registers parameter C_MMU_TLB_ACCESS.

PID is also used when accessing a TLB entry:

- When writing Translation Look-Aside Buffer High (TLBHI) the value of PID is stored in the TID field of the TLB entry
- When reading TLBHI and MSR[UM] is not set, the value in the TID field is stored in PID

Figure 1-12 illustrates the PID register and Table 1-18 provides bit descriptions and reset values.

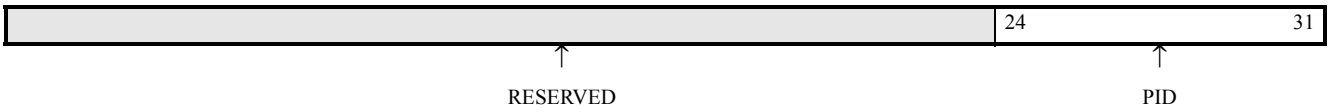


Figure 1-12: **PID**

Table 1-18: **Process Identifier Register (PID)**

Bits	Name	Description	Reset Value
0:23	Reserved		
24:31	PID	Used to uniquely identify a software process during MMU address translation. Read/Write	0x00

Zone Protection Register (ZPR)

The Zone Protection Register is used to override MMU memory protection defined in TLB entries. It is controlled by the C_USE_MMU configuration option on MicroBlaze. The register is only implemented if C_USE_MMU is greater than 1, C_AREA_OPTIMIZED is set to 0, and if the number of specified memory protection zones is greater than zero (C_MMU_ZONES > 0). The implemented register bits depend on the number of specified memory protection zones (C_MMU_ZONES). When accessed with the MFS and MTS instructions, the ZPR is specified by setting Sa = 0x1001. The register is accessible according to the memory management special registers parameter C_MMU_TLB_ACCESS. Figure 1-13 illustrates the ZPR register and Table 1-19 provides bit descriptions and reset values.

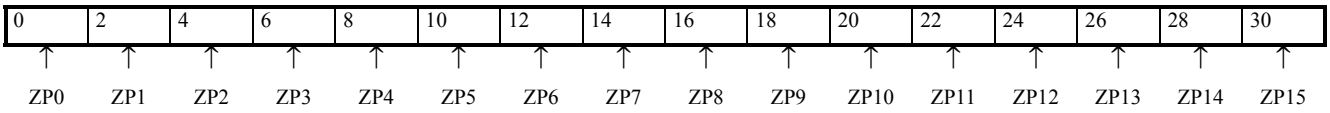


Figure 1-13: **ZPR**

Table 1-19: **Zone Protection Register (ZPR)**

Bits	Name	Description	Reset Value
0:1 2:3 ... 30:31	ZP0 ZP1 ... ZP15	Zone Protect User mode (MSR[UM] = 1): 00 = Override V in TLB entry. No access to the page is allowed 01 = No override. Use V, WR and EX from TLB entry 10 = No override. Use V, WR and EX from TLB entry 11 = Override WR and EX in TLB entry. Access the page as writable and executable Privileged mode (MSR[UM] = 0): 00 = No override. Use V, WR and EX from TLB entry 01 = No override. Use V, WR and EX from TLB entry 10 = Override WR and EX in TLB entry. Access the page as writable and executable 11 = Override WR and EX in TLB entry. Access the page as writable and executable Read/Write	0x00000000

Translation Look-Aside Buffer Low Register (TLBLO)

The Translation Look-Aside Buffer Low Register is used to access MMU Unified Translation Look-Aside Buffer (UTLB) entries. It is controlled by the C_USE_MMU configuration option on MicroBlaze. The register is only implemented if C_USE_MMU is greater than 1, and C_AREA_OPTIMIZED is set to 0. When accessed with the MFS and MTS instructions, the TLBLO is specified by setting Sa = 0x1003. When reading or writing TLBLO, the UTLB entry indexed by the TLBX register is accessed. The register is readable according to the memory management special registers parameter C_MMU_TLB_ACCESS.

The UTLB is reset on bit stream download (reset value is 0x00000000 for all TLBLO entries).

Note: The UTLB is **not** reset by the external reset inputs: Reset and Debug_Rst.

Figure 1-14 illustrates the TLBLO register and Table 1-20 provides bit descriptions and reset values.

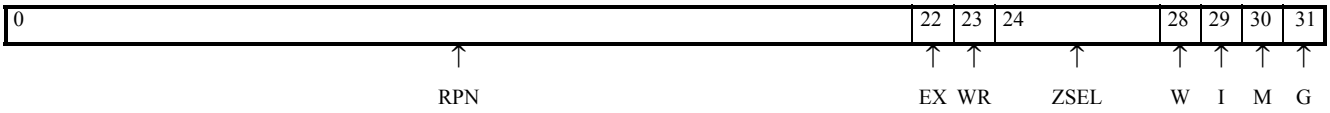


Figure 1-14: **TLBLO**

Table 1-20: **Translation Look-Aside Buffer Low Register (TLBLO)**

Bits	Name	Description	Reset Value
0:21	RPN	Real Page Number or Physical Page Number When a TLB hit occurs, this field is read from the TLB entry and is used to form the physical address. Depending on the value of the SIZE field, some of the RPN bits are not used in the physical address. Software must clear unused bits in this field to zero. Only defined when C_USE_MMU=3. Read/Write	0x000000
22	EX	Executable When bit is set to 1, the page contains executable code, and instructions can be fetched from the page. When bit is cleared to 0, instructions cannot be fetched from the page. Attempts to fetch instructions from a page with a clear EX bit cause an instruction-storage exception. Read/Write	0

Table 1-20: **Translation Look-Aside Buffer Low Register (TLBLO) *(Continued)***

Bits	Name	Description	Reset Value
23	WR	Writable When bit is set to 1, the page is writable and store instructions can be used to store data at addresses within the page. When bit is cleared to 0, the page is read-only (not writable). Attempts to store data into a page with a clear WR bit cause a data storage exception. Read/Write	0
24:27	ZSEL	Zone Select This field selects one of 16 zone fields (Z0-Z15) from the zone-protection register (ZPR). For example, if ZSEL 0x5, zone field Z5 is selected. The selected ZPR field is used to modify the access protection specified by the TLB entry EX and WR fields. It is also used to prevent access to a page by overriding the TLB V (valid) field. Read/Write	0x0
28	W	Write Through When the parameter `C_DCACHE_USE_WRITEBACK` is set to 1, this bit controls caching policy. A write-through policy is selected when set to 1, and a write-back policy is selected otherwise. This bit is fixed to 1, and write-through is always used, when `C_DCACHE_USE_WRITEBACK` is cleared to 0. Read/Write	0/1
29	I	Inhibit Caching When bit is set to 1, accesses to the page are not cached (caching is inhibited). When cleared to 0, accesses to the page are cacheable. Read/Write	0
30	M	Memory Coherent This bit is fixed to 0, because memory coherence is not implemented on MicroBlaze. Read Only	0
31	G	Guarded When bit is set to 1, speculative page accesses are not allowed (memory is guarded). When cleared to 0, speculative page accesses are allowed. The G attribute can be used to protect memory-mapped I/O devices from inappropriate instruction accesses. Read/Write	0

Translation Look-Aside Buffer High Register (TLBHI)

The Translation Look-Aside Buffer High Register is used to access MMU Unified Translation Look-Aside Buffer (UTLB) entries. It is controlled by the C_USE_MMU configuration option on MicroBlaze. The register is only implemented if C_USE_MMU is greater than 1, and C_AREA_OPTIMIZED is set to 0. When accessed with the MFS and MTS instructions, the TLBHI is specified by setting Sa = 0x1004. When reading or writing TLBHI, the UTLB entry indexed by the TLBX register is accessed. The register is readable according to the memory management special registers parameter C_MMU_TLB_ACCESS.

PID is also used when accessing a TLB entry:

- When writing TLBHI the value of PID is stored in the TID field of the TLB entry
- When reading TLBHI and MSR[UM] is not set, the value in the TID field is stored in PID

The UTLB is reset on bit stream download (reset value is 0x00000000 for all TLBHI entries).

Note: The UTLB is **not** reset by the external reset inputs: Reset and Debug_Rst.

Figure 1-15 illustrates the TLBHI register and Table 1-21 provides bit descriptions and reset values.

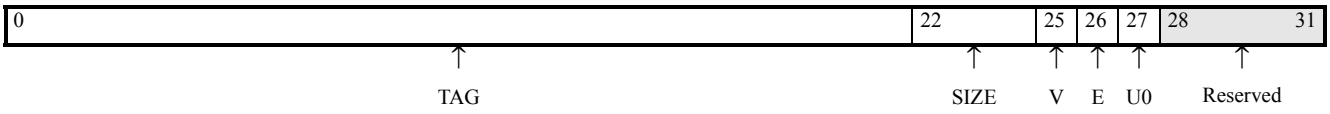


Figure 1-15: **TLBHI**

Table 1-21: **Translation Look-Aside Buffer High Register (TLBHI)**

Bits	Name	Description	Reset Value
0:21	TAG	TLB-entry tag Is compared with the page number portion of the virtual memory address under the control of the SIZE field. Read/Write	0x000000
22:24	SIZE	Size Specifies the page size. The SIZE field controls the bit range used in comparing the TAG field with the page number portion of the virtual memory address. The page sizes defined by this field are listed in Table 1-36. Read/Write	000
25	V	Valid When this bit is set to 1, the TLB entry is valid and contains a page-translation entry. When cleared to 0, the TLB entry is invalid. Read/Write	0

Table 1-21: **Translation Look-Aside Buffer High Register (TLBHI) *(Continued)***

Bits	Name	Description	Reset Value
26	E	Endian When this bit is set to 1, a the page is accessed as a little endian page if `C_ENDIANNESS` is 0, or as a big endian page otherwise. When cleared to 0, the page is accessed as a big endian page if `C_ENDIANNESS` is 0, or as a little endian page otherwise. The E bit only affects data read or data write accesses. Instruction accesses are not affected.. Read/Write	0
27	U0	User Defined This bit is fixed to 0, since there are no user defined storage attributes on MicroBlaze. Read Only	0
28:31	Reserved		

Translation Look-Aside Buffer Index Register (TLBX)

The Translation Look-Aside Buffer Index Register is used as an index to the Unified Translation Look-Aside Buffer (UTLB) when accessing the TLBLO and TLBHI registers. It is controlled by the C_USE_MMU configuration option on MicroBlaze. The register is only implemented if C_USE_MMU is greater than 1, and C_AREA_OPTIMIZED is set to 0. When accessed with the MFS and MTS instructions, the TLBX is specified by setting Sa = 0x1002. Figure 1-16 illustrates the TLBX register and Table 1-22 provides bit descriptions and reset values.

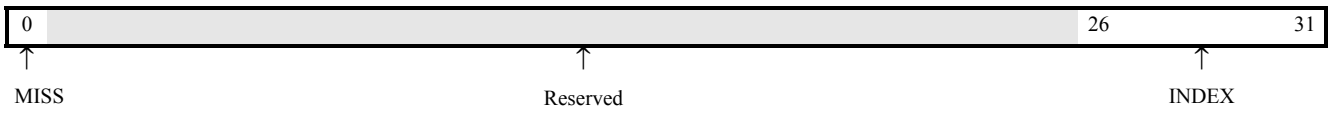


Figure 1-16: **TLBX**

Table 1-22: **Translation Look-Aside Buffer Index Register (TLBX)**

Bits	Name	Description	Reset Value
0	MISS	TLB Miss This bit is cleared to 0 when the TLBSX register is written with a virtual address, and the virtual address is found in a TLB entry. The bit is set to 1 if the virtual address is not found. It is also cleared when the TLBX register itself is written. Read Only Can be read if the memory management special registers parameter C_MMU_TLB_ACCESS > 0.	0
1:25	Reserved		
26:31	INDEX	TLB Index This field is used to index the Translation Look-Aside Buffer entry accessed by the TLBLO and TLBHI registers. The field is updated with a TLB index when the TLBSX register is written with a virtual address, and the virtual address is found in the corresponding TLB entry. Read/Write Can be read and written if the memory management special registers parameter C_MMU_TLB_ACCESS > 0.	000000

Translation Look-Aside Buffer Search Index Register (TLBSX)

The Translation Look-Aside Buffer Search Index Register is used to search for a virtual page number in the Unified Translation Look-Aside Buffer (UTLB). It is controlled by the C_USE_MMU configuration option on MicroBlaze. The register is only implemented if C_USE_MMU is greater than 1, and C_AREA_OPTIMIZED is set to 0. When written with the MTS instruction, the TLBSX is specified by setting Sa = 0x1005. Figure 1-17 illustrates the TLBSX register and Table 1-23 provides bit descriptions and reset values.

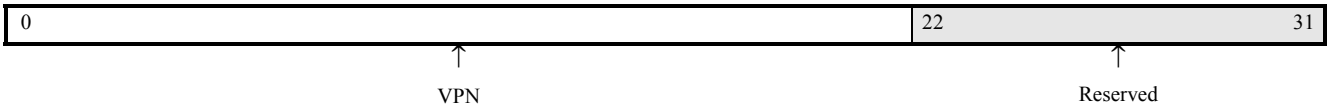


Figure 1-17: **TLBSX**

Table 1-23: **Translation Look-Aside Buffer Index Search Register (TLBSX)**

Bits	Name	Description	Reset Value
0:21	VPN	Virtual Page Number This field represents the page number portion of the virtual memory address. It is compared with the page number portion of the virtual memory address under the control of the SIZE field, in each of the Translation Look-Aside Buffer entries that have the V bit set to 1. If the virtual page number is found, the TLBX register is written with the index of the TLB entry and the MISS bit in TLBX is cleared to 0. If the virtual page number is not found in any of the TLB entries, the MISS bit in the TLBX register is set to 1. Write Only	
22:31	Reserved		

Processor Version Register (PVR)

The Processor Version Register is controlled by the C_PVR configuration option on MicroBlaze.

- When C_PVR is set to 0 the processor does not implement any PVR and MSR[PVR]=0.
- When C_PVR is set to 1, MicroBlaze implements only the first register: PVR0, and if set to 2, all 12 PVR registers (PVR0 to PVR11) are implemented.

When read with the MFS instruction the PVR is specified by setting Sa = 0x200*x*, with *x* being the register number between 0x0 and 0xB.

Table 1-24 through Table 1-35 provide bit descriptions and values.

Table 1-24: **Processor Version Register 0 (PVR0)**

Bits	Name	Description	Value
0	CFG	PVR implementation: 0=basic, 1=full	Based on C_PVR
1	BS	Use barrel shifter	C_USE_BARREL
2	DIV	Use divider	C_USE_DIV
3	MUL	Use hardware multiplier	C_USE_HW_MUL > 0
4	FPU	Use FPU	C_USE_FPU > 0
5	EXC	Use any type of exceptions	Based on C_*_EXCEPTION Also set if C_USE_MMU > 0
6	ICU	Use instruction cache	C_USE_ICACHE
7	DCU	Use data cache	C_USE_DCACHE
8	MMU	Use MMU	C_USE_MMU > 0
9	BTC	Use branch target cache	C_USE_BRANCH_TARGET_CACHE
10	ENDI	Selected endianness: 0 = big endian, 1 = little endian	C_ENDIANNESS
11	FT	Implement fault tolerant features	C_FAULT_TOLERANT
12	SPROT	Use stack protection	C_USE_STACK_PROTECTION
13:15	Reserved		0
16:23	MBV	MicroBlaze release version code 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 0x10 = v7.30.a 0x11 = v7.30.b 0x12 = v8.00.a 0x13 = v8.00.b 0x14 = v8.10.a	Release Specific
24:31	USR1	User configured value 1	C_PVR_USER1

Table 1-25: **Processor Version Register 1 (PVR1)**

Bits	Name	Description	Value
0:31	USR2	User configured value 2	C_PVR_USER2

Table 1-26: **Processor Version Register 2 (PVR2)**

Bits	Name	Description	Value
0	DAXI	Data side AXI4 in use	C_D_AXI
1	DLMB	Data side LMB in use	C_D_LMB
2	IAXI	Instruction side AXI4 in use	C_I_AXI
3	ILMB	Instruction side LMB in use	C_I_LMB
4	IRQEDGE	Interrupt is edge triggered	C_INTERRUPT_IS_EDGE
5	IRQPOS	Interrupt edge is positive	C_EDGE_IS_POSITIVE
6	DPLB	Data side PLB in use	C_D_PLB
7	IPLB	Instruction side PLB in use	C_I_PLB
8	INTERCON	Use PLB interconnect	C_INTERCONNECT
9	STREAM	Use AXI4-Stream interconnect	C_STREAM_INTERCONNECT
10:11	Reserved		
12	FSL	Use extended stream (FSL or AXI) instructions	C_USE_EXTENDED_FSL_INSTR
13	FSLEXC	Generate exception for stream control bit (FSL or AXI) mismatch	C_FSL_EXCEPTION
14	MSR	Use msrset and msrclr instructions	C_USE_MSR_INSTR
15	PCMP	Use pattern compare and CLZ instructions	C_USE_PCMP_INSTR
16	AREA	Select implementation to optimize area with lower instruction throughput	C_AREA_OPTIMIZED
17	BS	Use barrel shifter	C_USE_BARREL
18	DIV	Use divider	C_USE_DIV
19	MUL	Use hardware multiplier	C_USE_HW_MUL > 0
20	FPU	Use FPU	C_USE_FPU > 0
21	MUL64	Use 64-bit hardware multiplier	C_USE_HW_MUL = 2

Table 1-26: **Processor Version Register 2 (PVR2) *(Continued)***

Bits	Name	Description	Value
22	FPU2	Use floating point conversion and square root instructions	C_USE_FPU = 2
23	IPLBEXC	Generate exception for IPLB error	C_IPLB_BUS_EXCEPTION
24	DPLBEXC	Generate exception for DPLB error	C_DPLB_BUS_EXCEPTION
25	OP0EXC	Generate exception for 0x0 illegal opcode	C_OPCODE_0x0_ILLEGAL
26	UNEXC	Generate exception for unaligned data access	C_UNALIGNED_EXCEPTIONS
27	OPEXC	Generate exception for any illegal opcode	C_ILL_OPCODE_EXCEPTION
28	AXIIEXC	Generate exception for M_AXI_I error	C_M_AXI_I_BUS_EXCEPTION
29	AXIDEXC	Generate exception for M_AXI_D error	C_M_AXI_D_BUS_EXCEPTION
30	DIVEXC	Generate exception for division by zero or division overflow	C_DIV_ZERO_EXCEPTION
31	FPUEXC	Generate exceptions from FPU	C_FPU_EXCEPTION

Table 1-27: **Processor Version Register 3 (PVR3)**

Bits	Name	Description	Value
0	DEBUG	Use debug logic	C_DEBUG_ENABLED
1:2	Reserved		
3:6	PCBRK	Number of PC breakpoints	C_NUMBER_OF_PC_BRK
7:9	Reserved		
10:12	RDADDR	Number of read address breakpoints	C_NUMBER_OF_RD_ADDR_BRK
13:15	Reserved		
16:18	WRADDR	Number of write address breakpoints	C_NUMBER_OF_WR_ADDR_BRK
19	Reserved		
20:24	FSL	Number of stream links	C_FSL_LINKS
25:28	Reserved		
29:31	BTC_SIZE	Branch Target Cache size	C_BRANCH_TARGET_CACHE_SIZE

Table 1-28: **Processor Version Register 4 (PVR4)**

Bits	Name	Description	Value
0	ICU	Use instruction cache	C_USE_ICACHE
1:5	ICTS	Instruction cache tag size	C_ADDR_TAG_BITS
6	Reserved		1
7	ICW	Allow instruction cache write	C_ALLOW_ICACHE_WR
8:10	ICLL	The base two logarithm of the instruction cache line length	log2(C_ICACHE_LINE_LEN)
11:15	ICBS	The base two logarithm of the instruction cache byte size	log2(C_CACHE_BYTE_SIZE)
16	IAU	The instruction cache is used for all memory accesses	C_ICACHE_ALWAYS_USED
17	Reserved		0
18	ICI	Instruction cache XCL protocol	C_ICACHE_INTERFACE
19:21	ICV	Instruction cache victims	0-3: C_ICACHE_VICTIMS = 0,2,4,8
22:23	ICS	Instruction cache streams	C_ICACHE_STREAMS
24	IFTL	Instruction cache tag uses distributed RAM	C_ICACHE_FORCE_TAG_LUTRAM
25	ICDW	Instruction cache data width	C_ICACHE_DATA_WIDTH
26:31	Reserved		0

Table 1-29: **Processor Version Register 5 (PVR5)**

Bits	Name	Description	Value
0	DCU	Use data cache	C_USE_DCACHE
1:5	DCTS	Data cache tag size	C_DCACHE_ADDR_TAG
6	Reserved		1
7	DCW	Allow data cache write	C_ALLOW_DCACHE_WR
8:10	DCLL	The base two logarithm of the data cache line length	log2(C_DCACHE_LINE_LEN)
11:15	DCBS	The base two logarithm of the data cache byte size	log2(C_DCACHE_BYTE_SIZE)
16	DAU	The data cache is used for all memory accesses	C_DCACHE_ALWAYS_USED
17	DWB	Data cache policy is write-back	C_DCACHE_USE_WRITEBACK
18	DCI	Data cache XCL protocol	C_DCACHE_INTERFACE

Table 1-29: **Processor Version Register 5 (PVR5) *(Continued)***

Bits	Name	Description	Value
19:21	DCV	Data cache victims	0-3: C_DCACHE_VICTIMS = 0,2,4,8
22:23	Reserved		0
24	DFTL	Data cache tag uses distributed RAM	C_DCACHE_FORCE_TAG_LUTRAM
25	DCDW	Data cache data width	C_DCACHE_DATA_WIDTH
26:31	Reserved		0

Table 1-30: **Processor Version Register 6 (PVR6)**

Bits	Name	Description	Value
0:31	ICBA	Instruction Cache Base Address	C_ICACHE_BASEADDR

Table 1-31: **Processor Version Register 7 (PVR7)**

Bits	Name	Description	Value
0:31	ICHA	Instruction Cache High Address	C_ICACHE_HIGHADDR

Table 1-32: **Processor Version Register 8 (PVR8)**

Bits	Name	Description	Value
0:31	DCBA	Data Cache Base Address	C_DCACHE_BASEADDR

Table 1-33: **Processor Version Register 9 (PVR9)**

Bits	Name	Description	Value
0:31	DCHA	Data Cache High Address	C_DCACHE_HIGHADDR

Table 1-34: **Processor Version Register 10 (PVR10)**

Bits	Name	Description	Value
0:7	ARCH	Target architecture: 0x6 = Spartan®-3, Automotive Spartan-3 0x7 = Virtex-4, Defence Grade Virtex-4 Q Space-Grade Virtex-4 QV 0x8 = Virtex-5, Space-Grade Virtex-5 QV 0x9 = Spartan-3E, Automotive Spartan-3E 0xA = Spartan-3A, Automotive Spartan-3A 0xB = Spartan-3AN 0xC = Spartan-3A DSP, Automotive Spartan-3A DSP 0xD = Spartan-6, Automotive Spartan-6, Defence Grade Spartan-6 Q 0xE = Virtex-6, Defence Grade Virtex-6 Q 0xF = Virtex-7 0x10 = Kintex™-7	Defined by parameter C_FAMILY
8:31	Reserved		0

Table 1-35: **Processor Version Register 11 (PVR11)**

Bits	Name	Description	Value
0:1	MMU	Use MMU: 0 = None 1 = User Mode 2 = Protection 3 = Virtual	C_USE_MMU
2:4	ITLB	Instruction Shadow TLB size	log2(C_MMU_ITLB_SIZE)
5:7	DTLB	Data Shadow TLB size	log2(C_MMU_DTLB_SIZE)
8:9	TLBACC	TLB register access: 0 = Minimal 1 = Read 2 = Write 3 = Full	C_MMU_TLB_ACCESS
10:14	ZONES	Number of memory protection zones	C_MMU_ZONES
15	PRIVINS	Privileged instructions: 0 = Full protection 1 = Allow stream instructions	C_MMU_PRIVILEGED_INSTR
16:16	Reserved	Reserved for future use	0
17:31	RSTMSR	Reset value for MSR	C_RESET_MSR

Pipeline Architecture

MicroBlaze instruction execution is pipelined. For most instructions, each stage takes one clock cycle to complete. Consequently, the number of clock cycles necessary for a specific instruction to complete is equal to the number of pipeline stages, and one instruction is completed on every cycle. A few instructions require multiple clock cycles in the execute stage to complete. This is achieved by stalling the pipeline.

When executing from slower memory, instruction fetches may take multiple cycles. This additional latency directly affects the efficiency of the pipeline. MicroBlaze implements an instruction prefetch buffer that reduces the impact of such multi-cycle instruction memory latency. While the pipeline is stalled by a multi-cycle instruction in the execution stage, the prefetch buffer continues to load sequential instructions. When the pipeline resumes execution, the fetch stage can load new instructions directly from the prefetch buffer instead of waiting for the instruction memory access to complete. If instructions are modified during execution (e.g. with self-modifying code), the prefetch buffer should be emptied before executing the modified instructions, to ensure that it does not contain the old unmodified instructions. The recommended way to do this is using an MBAR instruction, although it is also possible to use a synchronizing branch instruction, for example BRI 4.

Three Stage Pipeline

With `C_AREA_OPTIMIZED` set to 1, the pipeline is divided into three stages to minimize hardware cost: Fetch, Decode, and Execute.

	cycle 1	cycle 2	cycle 3	cycle4	cycle5	cycle6	cycle7
instruction 1	Fetch	Decode	Execute				
instruction 2		Fetch	Decode	Execute	Execute	Execute	
instruction 3			Fetch	Decode	Stall	Stall	Execute

Five Stage Pipeline

With `C_AREA_OPTIMIZED` set to 0, the pipeline is divided into five stages to maximize performance: Fetch (IF), Decode (OF), Execute (EX), Access Memory (MEM), and Writeback (WB).

	cycle 1	cycle 2	cycle 3	cycle4	cycle5	cycle6	cycle7	cycle8	cycle9
instruction 1	IF	OF	EX	MEM	WB				
instruction 2		IF	OF	EX	MEM	MEM	MEM	WB	
instruction 3			IF	OF	EX	Stall	Stall	MEM	WB

Branches

Normally the instructions in the fetch and decode stages (as well as prefetch buffer) are flushed when executing a taken branch. The fetch pipeline stage is then reloaded with a new instruction from the calculated branch address. A taken branch in MicroBlaze takes three clock cycles to execute, two of which are required for refilling the pipeline. To reduce this latency overhead, MicroBlaze supports branches with delay slots.

Delay Slots

When executing a taken branch with delay slot, only the fetch pipeline stage in MicroBlaze is flushed. The instruction in the decode stage (branch delay slot) is allowed to complete. This technique effectively reduces the branch penalty from two clock cycles to one. Branch instructions with delay slots have a D appended to the instruction mnemonic. For example, the BNE instruction does not execute the subsequent instruction (does not have a delay slot), whereas BNED executes the next instruction before control is transferred to the branch location.

A delay slot must not contain the following instructions: IMM, branch, or break. Interrupts and external hardware breaks are deferred until after the delay slot branch has been completed.

Instructions that could cause recoverable exceptions (e.g. unaligned word or halfword load and store) are allowed in the delay slot. If an exception is caused in a delay slot the ESR[DS] bit is set, and the exception handler is responsible for returning the execution to the branch target (stored in the special purpose register BTR). If the ESR[DS] bit is set, register R17 is not valid (otherwise it contains the address following the instruction causing the exception).

Branch Target Cache

To improve branch performance, MicroBlaze provides a Branch Target Cache (BTC) coupled with a branch prediction scheme. With the BTC enabled, a correctly predicted immediate branch or return instruction incurs no overhead.

The BTC operates by saving the target address of each immediate branch and return instruction the first time the instruction is encountered. The next time it is encountered, it is usually found in the Branch Target Cache, and the Instruction Fetch Program Counter is then simply changed to the saved target address, in case the branch should be taken. Unconditional branches and return instructions are always taken, whereas conditional branches use branch prediction, to avoid taking a branch that should not have been taken and vice versa.

The BTC is cleared when a memory barrier (MBAR 0) or synchronizing branch (BRI 4) is executed.

There are three cases where the branch prediction can cause a mispredict, namely:

- A conditional branch that should not have been taken, is actually taken,
- A conditional branch that should actually have been taken, is not taken,
- The target address of a return instruction is incorrect, which may occur when returning from a function called from different places in the code.

All of these cases are detected and corrected when the branch or return instruction reaches the execute stage, and the branch prediction bits or target address are updated in the BTC, to reflect the actual instruction behavior. This correction incurs a penalty of two clock cycles.

The size of the BTC can be selected with `C_BRANCH_TARGET_CACHE_SIZE`. The default recommended setting uses one block RAM, and provides either 512 entries (for Virtex-5 or Virtex-6) or 256 entries (for all other families). When selecting 64 entries or below, distributed RAM is used to implement the BTC, otherwise block RAM is used.

When the BTC uses block RAM, and `C_FAULT_TOLERANT` is set to 1, block RAMs are protected by parity. In case of a parity error, the branch is not predicted. To avoid accumulating errors in this case, the BTC should be cleared periodically by a synchronizing branch.

The Branch Target Cache is available when `C_USE_BRANCH_TARGET_CACHE` is set to 1 and `C_AREA_OPTIMIZED` is set to 0.

Memory Architecture

MicroBlaze is implemented with a Harvard memory architecture; instruction and data accesses are done in separate address spaces. Each address space has a 32-bit range (that is, handles up to 4-GB of instructions and data memory respectively). The instruction and data memory ranges can be made to overlap by mapping them both to the same physical memory. The latter is useful for software debugging.

Both instruction and data interfaces of MicroBlaze are default 32 bits wide and use big endian or little endian, bit-reversed format, depending on the parameter C_ENDIANNESS. MicroBlaze supports word, halfword, and byte accesses to data memory.

Data accesses must be aligned (word accesses must be on word boundaries, halfword on halfword boundaries), unless the processor is configured to support unaligned exceptions. All instruction accesses must be word aligned.

MicroBlaze prefetches instructions to improve performance, using the instruction prefetch buffer and (if enabled) instruction cache streams. To avoid attempts to prefetch instructions beyond the end of physical memory, which may cause an instruction bus error or a processor stall, instructions must not be located too close to the end of physical memory. The instruction prefetch buffer requires 16 bytes margin, and using instruction cache streams adds two additional cache lines (32 or 64 bytes).

MicroBlaze does not separate data accesses to I/O and memory (it uses memory mapped I/O). The processor has up to three interfaces for memory accesses:

- Local Memory Bus (LMB)
- Advanced eXtensible Interface (AXI4) or Processor Local Bus (PLB)
- Advanced eXtensible Interface (AXI4) or Xilinx CacheLink (XCL)

The LMB memory address range must not overlap with AXI4, PLB or XCL ranges.

The C_ENDIANNESS parameter is always automatically set to little endian when using AXI4, and to big endian when using PLB.

MicroBlaze has a single cycle latency for accesses to local memory (LMB) and for cache read hits, except with C_AREA_OPTIMIZED set to 1, when data side accesses and data cache read hits require two clock cycles, and with C_FAULT_TOLERANT set to 1, when byte writes and halfword writes to LMB normally require two clock cycles.

The data cache write latency depends on C_DCACHE_USE_WRITEBACK. When C_DCACHE_USE_WRITEBACK is set to 1, the write latency normally is one cycle (more if the cache needs to do memory accesses). When C_DCACHE_USE_WRITEBACK is cleared to 0, the write latency normally is two cycles (more if the posted-write buffer in the memory controller is full).

The MicroBlaze instruction and data caches can be configured to use 4 or 8 word cache lines. When using a longer cache line, more bytes are prefetched, which generally improves performance for software with sequential access patterns. However, for software with a more random access pattern the performance can instead decrease for a given cache size. This is caused by a reduced cache hit rate due to fewer available cache lines.

For details on the different memory interfaces refer to Chapter 2, “MicroBlaze Signal Interface Description”.

Privileged Instructions

The following MicroBlaze instructions are privileged:

- GET, PUT, NGET, NPUT, CGET, CPUT, NCGET, NCPUT
- WIC, WDC
- MTS
- MSRCLR, MSRSET (except when only the C bit is affected)
- BRK
- RTID, RTBD, RTED
- BRKI (except when jumping to physical address 0x8 or 0x18)

Attempted use of these instructions when running in user mode causes a privileged instruction exception.

There are six ways to leave user mode and virtual mode:

1. Hardware generated reset (including debug reset)
2. Hardware exception
3. Non-maskable break or hardware break
4. Interrupt
5. Executing the instruction "BRALID Re, 0x8" to perform a user vector exception
6. Executing the software break instructions "BRKI" jumping to physical address 0x8 or 0x18

In all of these cases, except hardware generated reset, the user mode and virtual mode status is saved in the MSR UMS and VMS bits.

Application (user-mode) programs transfer control to system-service routines (privileged mode programs) using the BRALID or BRKI instruction, jumping to physical address 0x8. Executing this instruction causes a system-call exception to occur. The exception handler determines which system-service routine to call and whether the calling application has permission to call that service. If permission is granted, the exception handler performs the actual procedure call to the system-service routine on behalf of the application program.

The execution environment expected by the system-service routine requires the execution of prologue instructions to set up that environment. Those instructions usually create the block of storage that holds procedural information (the activation record), update and initialize pointers, and save volatile registers (registers the system-service routine uses). Prologue code can be inserted by the linker when creating an executable module, or it can be included as stub code in either the system-call interrupt handler or the system-library routines.

Returns from the system-service routine reverse the process described above. Epilog code is executed to unwind and deallocate the activation record, restore pointers, and restore volatile registers. The interrupt handler executes a return from exception instruction (RTED) to return to the application.

Virtual-Memory Management

Programs running on MicroBlaze use effective addresses to access a flat 4 GB address space. The processor can interpret this address space in one of two ways, depending on the translation mode:

- In real mode, effective addresses are used to directly access physical memory
- In virtual mode, effective addresses are translated into physical addresses by the virtual-memory management hardware in the processor

Virtual mode provides system software with the ability to relocate programs and data anywhere in the physical address space. System software can move inactive programs and data out of physical memory when space is required by active programs and data.

Relocation can make it appear to a program that more memory exists than is actually implemented by the system. This frees the programmer from working within the limits imposed by the amount of physical memory present in a system. Programmers do not need to know which physical-memory addresses are assigned to other software processes and hardware devices. The addresses visible to programs are translated into the appropriate physical addresses by the processor.

Virtual mode provides greater control over memory protection. Blocks of memory as small as 1 KB can be individually protected from unauthorized access. Protection and relocation enable system software to support multitasking. This capability gives the appearance of simultaneous or near-simultaneous execution of multiple programs.

In MicroBlaze, virtual mode is implemented by the memory-management unit (MMU), available when `C_USE_MMU` is set to 3 and `C_AREA_OPTIMIZED` is set to 0. The MMU controls effective-address to physical-address mapping and supports memory protection. Using these capabilities, system software can implement demand-paged virtual memory and other memory management schemes.

The MicroBlaze MMU implementation is based upon PowerPC™ 405. For details, see the *PowerPC Processor Reference Guide* document.

The MMU features are summarized as follows:

- Translates effective addresses into physical addresses
- Controls page-level access during address translation
- Provides additional virtual-mode protection control through the use of zones
- Provides independent control over instruction-address and data-address translation and protection
- Supports eight page sizes: 1 kB, 4 kB, 16 kB, 64 kB, 256 kB, 1 MB, 4 MB, and 16 MB. Any combination of page sizes can be used by system software
- Software controls the page-replacement strategy

Real Mode

The processor references memory when it fetches an instruction and when it accesses data with a load or store instruction. Programs reference memory locations using a 32-bit effective address calculated by the processor. When real mode is enabled, the physical address is identical to the effective address and the processor uses it to access physical memory. After a processor reset, the processor operates in real mode. Real mode can also be enabled by clearing the VM bit in the MSR.

Physical-memory data accesses (loads and stores) are performed in real mode using the effective address. Real mode does not provide system software with virtual address translation, but the full memory access-protection is available, implemented when C_USE_MMU > 1 and C_AREA_OPTIMIZED = 0. Implementation of a real-mode memory manager is more

straightforward than a virtual-mode memory manager. Real mode is often an appropriate solution for memory management in simple embedded environments, when access-protection is necessary, but virtual address translation is not required.

Virtual Mode

In virtual mode, the processor translates an effective address into a physical address using the process shown in Figure 1-18. Virtual mode can be enabled by setting the VM bit in the MSR..

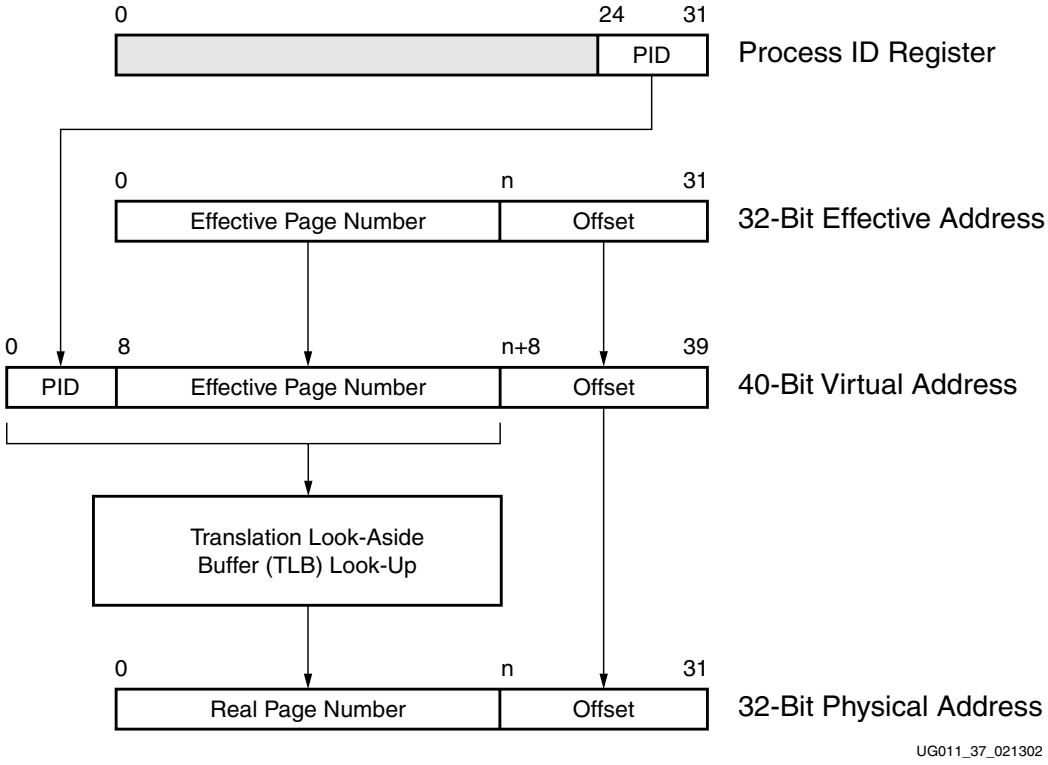


Figure 1-18: **Virtual-Mode Address Translation**

Each address shown in Figure 1-18 contains a page-number field and an offset field. The page number represents the portion of the address translated by the MMU. The offset represents the byte offset into a page and is not translated by the MMU. The virtual address consists of an additional field, called the process ID (PID), which is taken from the PID register (see Process-ID Register, page 39). The combination of PID and effective page number (EPN) is referred to as the virtual page number (VPN). The value n is determined by the page size, as shown in Table 1-36.

System software maintains a page-translation table that contains entries used to translate each virtual page into a physical page. The page size defined by a page translation entry determines the size of the page number and offset fields. For example, when a 4 kB page size is used, the page-number field is 20 bits and the offset field is 12 bits. The VPN in this case is 28 bits.

Then the most frequently used page translations are stored in the translation look-aside buffer (TLB). When translating a virtual address, the MMU examines the page-translation entries for a matching VPN (PID and EPN). Rather than examining all entries in the table, only entries contained in the processor TLB are examined. When a page-translation entry is found with a matching VPN, the corresponding physical-page number is read from the entry and combined with the offset to form the 32-bit physical address. This physical address is used by the processor to reference memory.

System software can use the PID to uniquely identify software processes (tasks, subroutines, threads) running on the processor. Independently compiled processes can operate in effective-address regions that overlap each other. This overlap must be resolved by system software if multitasking is supported. Assigning a PID to each process enables system software to resolve the overlap by relocating each process into a unique region of virtual-address space. The virtual-address space mappings enable independent translation of each process into the physical-address space.

Page-Translation Table

The page-translation table is a software-defined and software-managed data structure containing page translations. The requirement for software-managed page translation represents an architectural trade-off targeted at embedded-system applications. Embedded systems tend to have a tightly controlled operating environment and a well-defined set of application software. That environment enables virtual-memory management to be optimized for each embedded system in the following ways:

- The page-translation table can be organized to maximize page-table search performance (also called table walking) so that a given page-translation entry is located quickly. Most general-purpose processors implement either an indexed page table (simple search method, large page-table size) or a hashed page table (complex search method, small page-table size). With software table walking, any hybrid organization can be employed that suits the particular embedded system. Both the page-table size and access time can be optimized.
- Independent page sizes can be used for application modules, device drivers, system service routines, and data. Independent page-size selection enables system software to more efficiently use memory by reducing fragmentation (unused memory). For example, a large data structure can be allocated to a 16 MB page and a small I/O device-driver can be allocated to a 1 KB page.
- Page replacement can be tuned to minimize the occurrence of missing page translations. As described in the following section, the most-frequently used page translations are stored in the translation look-aside buffer (TLB). Software is responsible for deciding which translations are stored in the TLB and which translations are replaced when a new translation is required. The replacement strategy can be tuned to avoid thrashing, whereby page-translation entries are constantly being moved in and out of the TLB. The replacement strategy can also be tuned to prevent replacement of critical-page translations, a process sometimes referred to as page locking.

The unified 64-entry TLB, managed by software, caches a subset of instruction and data page-translation entries accessible by the MMU. Software is responsible for reading entries from the page-translation table in system memory and storing them in the TLB. The following section describes the unified TLB in more detail. Internally, the MMU also contains shadow TLBs for instructions and data, with sizes configurable by `C_MMU_ITLB_SIZE` and `C_MMU_DTLB_SIZE` respectively.

These shadow TLBs are managed entirely by the processor (transparent to software) and are used to minimize access conflicts with the unified TLB.

Translation Look-Aside Buffer

The translation look-aside buffer (TLB) is used by the MicroBlaze MMU for address translation when the processor is running in virtual mode, memory protection, and storage control. Each entry within the TLB contains the information necessary to identify a virtual page (PID and effective page number), specify its translation into a physical page, determine the protection characteristics of the page, and specify the storage attributes associated with the page.

The MicroBlaze TLB is physically implemented as three separate TLBs:

- Unified TLB—The UTLB contains 64 entries and is pseudo-associative. Instruction-page and data-page translation can be stored in any UTLB entry. The initialization and management of the UTLB is controlled completely by software.
- Instruction Shadow TLB—The ITLB contains instruction page-translation entries and is fully associative. The page-translation entries stored in the ITLB represent the most-recently accessed instruction-page translations from the UTLB. The ITLB is used to minimize contention between instruction translation and UTLB-update operations. The initialization and management of the ITLB is controlled completely by hardware and is transparent to software.
- Data Shadow TLB—The DTLB contains data page-translation entries and is fully associative. The page-translation entries stored in the DTLB represent the most-recently accessed data-page translations from the UTLB. The DTLB is used to minimize contention between data translation and UTLB-update operations. The initialization and management of the DTLB is controlled completely by hardware and is transparent to software.

Figure 1-19 provides the translation flow for TLB.

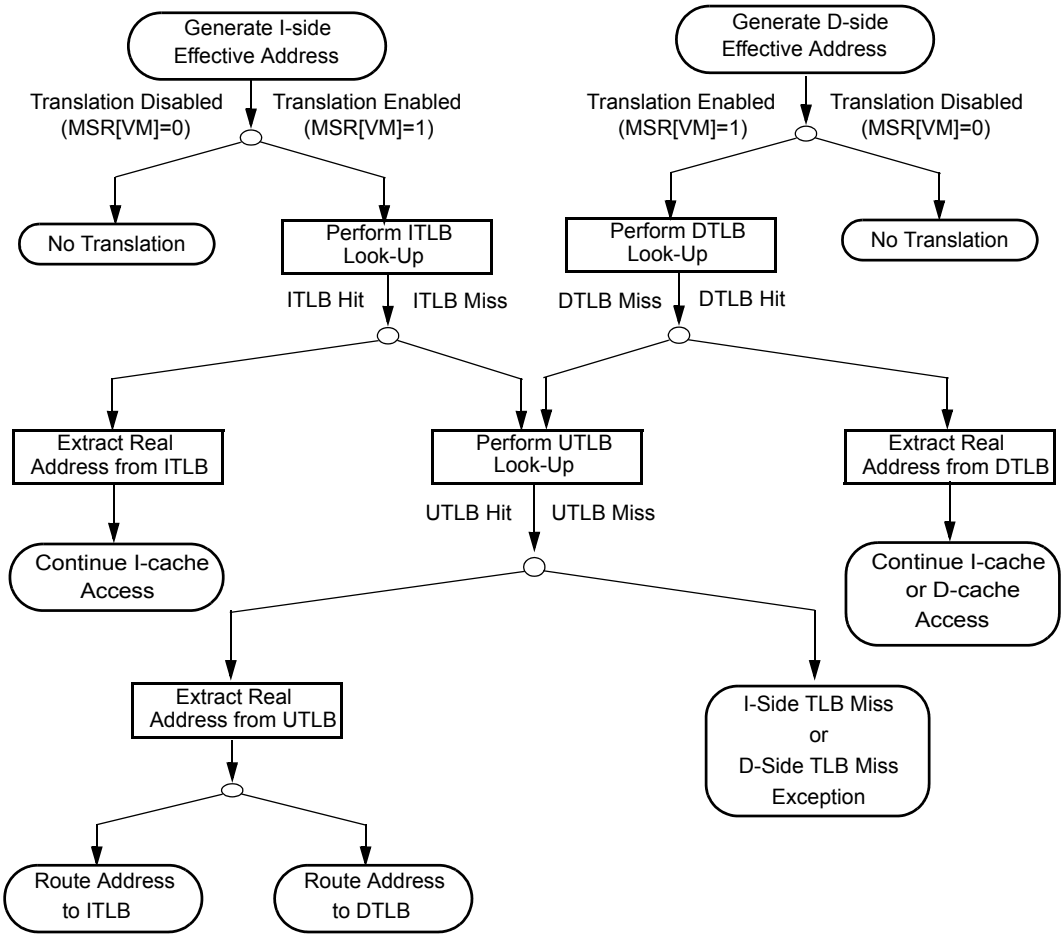


Figure 1-19: **TLB Address Translation Flow**

TLB Entry Format

Figure 1-20 shows the format of a TLB entry. Each TLB entry is 68 bits and is composed of two portions: TLBLO (also referred to as the data entry), and TLBHI (also referred to as the tag entry).

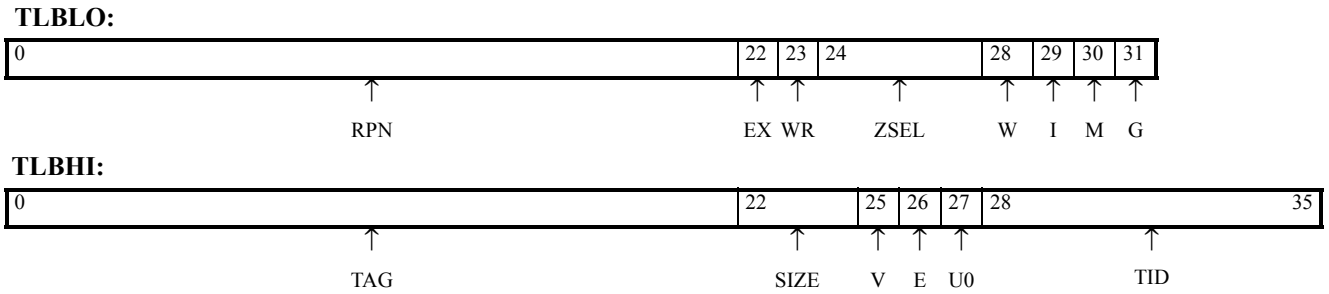


Figure 1-20: **TLB Entry Format**

The TLB entry contents are described in Table 1-20, page 41 and Table 1-21, page 43.

The fields within a TLB entry are categorized as follows:

- Virtual-page identification (TAG, SIZE, V, TID)—These fields identify the page-translation entry. They are compared with the virtual-page number during the translation process.
- Physical-page identification (RPN, SIZE)—These fields identify the translated page in physical memory.
- Access control (EX, WR, ZSEL)—These fields specify the type of access allowed in the page and are used to protect pages from improper accesses.
- Storage attributes (W, I, M, G, E, U0)—These fields specify the storage-control attributes, such as caching policy for the data cache (write-back or write-through), whether a page is cacheable, and how bytes are ordered (endianness).

Table 1-36 shows the relationship between the TLB-entry `SIZE` field and the translated page size. This table also shows how the page size determines which address bits are involved in a tag comparison, which address bits are used as a page offset, and which bits in the physical page number are used in the physical address.

Table 1-36: **Page-Translation Bit Ranges by Page Size**

Page Size	SIZE (TLBHI Field)	Tag Comparison Bit Range	Page Offset	Physical Page Number	RPN Bits Clear to 0
1 KB	000	TAG[0:21] - Address[0:21]	Address[22:31]	RPN[0:21]	-
4 KB	001	TAG[0:19] - Address[0:19]	Address[20:31]	RPN[0:19]	20:21
16 KB	010	TAG[0:17] - Address[0:17]	Address[18:31]	RPN[0:17]	18:21
64 KB	011	TAG[0:15] - Address[0:15]	Address[16:31]	RPN[0:15]	16:21
256 KB	100	TAG[0:13] - Address[0:13]	Address[14:31]	RPN[0:13]	14:21
1 MB	101	TAG[0:11] - Address[0:11]	Address[12:31]	RPN[0:11]	12:21
4 MB	110	TAG[0:9] - Address[0:9]	Address[10:31]	RPN[0:9]	10:21
16 MB	111	TAG[0:7] - Address[0:7]	Address[8:31]	RPN[0:7]	8:21

TLB Access

When the MMU translates a virtual address (the combination of PID and effective address) into a physical address, it first examines the appropriate shadow TLB for the page translation entry. If an entry is found, it is used to access physical memory. If an entry is not found, the MMU examines the UTLB for the entry. A delay occurs each time the UTLB must be accessed due to a shadow TLB miss. The miss latency ranges from 2-32 cycles. The DTLB has priority over the ITLB if both simultaneously access the UTLB.

Figure 1-21, page 63 shows the logical process the MMU follows when examining a page-translation entry in one of the shadow TLBs or the UTLB. All valid entries in the TLB are checked.

A TLB hit occurs when all of the following conditions are met by a TLB entry:

- The entry is valid
- The TAG field in the entry matches the effective address EPN under the control of the SIZE field in the entry
- The TID field in the entry matches the PID

If any of the above conditions are not met, a TLB miss occurs. A TLB miss causes an exception, described as follows:

A TID value of 0x00 causes the MMU to ignore the comparison between the TID and PID. Only the TAG and EA[EPN] are compared. A TLB entry with TID=0x00 represents a process-independent translation. Pages that are accessed globally by all processes should be assigned a TID value of 0x00. A PID value of 0x00 does not identify a process that can access any page. When PID=0x00, a page-translation hit only occurs when TID=0x00. It is possible for software to load the TLB with multiple entries that match an EA[EPN] and PID combination. However, this is considered a programming error and results in undefined behavior.

When a hit occurs, the MMU reads the RPN field from the corresponding TLB entry. Some or all of the bits in this field are used, depending on the value of the SIZE field (see Table 1-36). For example, if the `SIZE` field specifies a 256 kB page size, RPN[0:13] represents the physical page number and is used to form the physical address. RPN[14:21] is not used, and software must clear those bits to 0 when initializing the TLB entry. The remainder of the physical address is taken from the page-offset portion of the EA. If the page size is 256 kB, the 32-bit physical address is formed by concatenating RPN[0:13] with bits14:31 of the effective address.

Prior to accessing physical memory, the MMU examines the TLB-entry access-control fields. These fields indicate whether the currently executing program is allowed to perform the requested memory access.

If access is allowed, the MMU checks the storage-attribute fields to determine how to access the page. The storage-attribute fields specify the caching policy for memory accesses.

TLB Access Failures

A TLB-access failure causes an exception to occur. This interrupts execution of the instruction that caused the failure and transfers control to an interrupt handler to resolve the failure. A TLB access can fail for two reasons:

- A matching TLB entry was not found, resulting in a TLB miss
- A matching TLB entry was found, but access to the page was prevented by either the storage attributes or zone protection

When an interrupt occurs, the processor enters real mode by clearing MSR[VM] to 0. In real mode, all address translation and memory-protection checks performed by the MMU are disabled. After

system software initializes the UTLB with page-translation entries, management of the MicroBlaze UTLB is usually performed using interrupt handlers running in real mode.

Figure 1-21 diagrams the general process for examining a TLB entry.

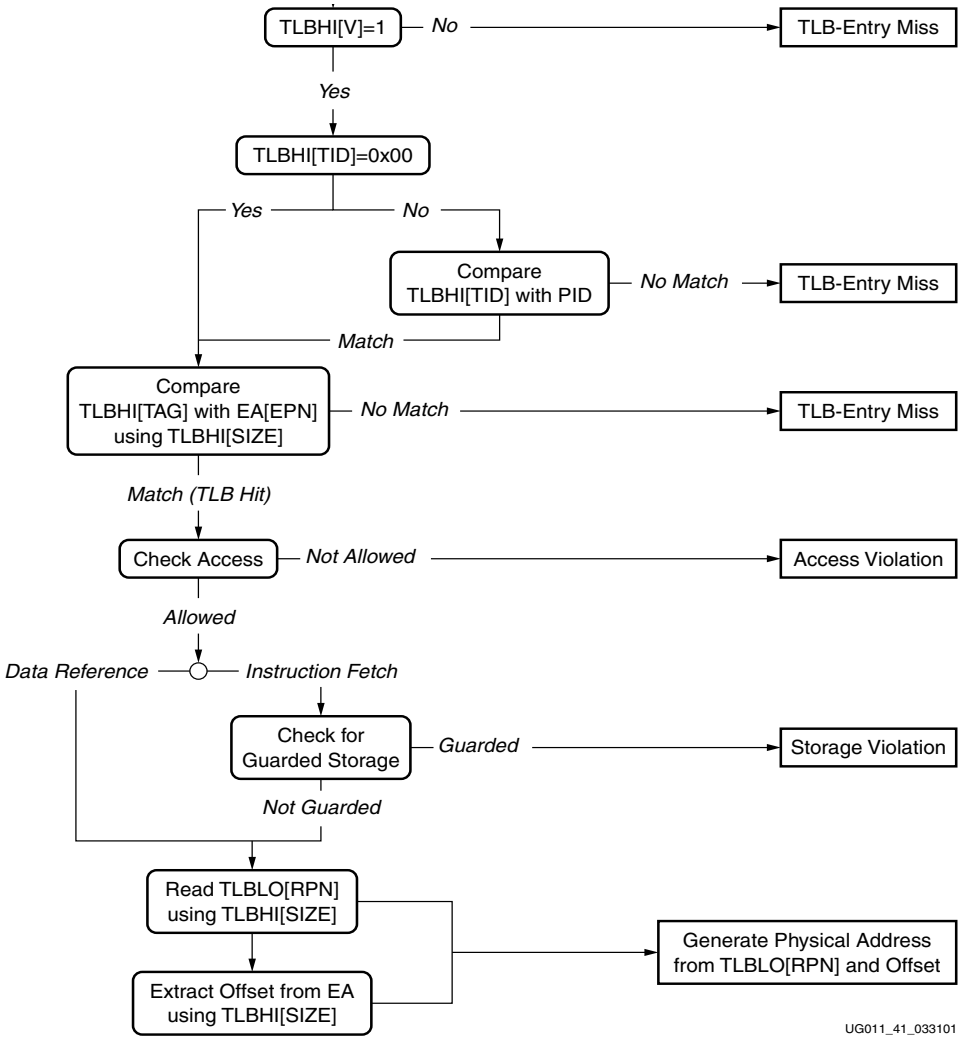


Figure 1-21: **General Process for Examining a TLB Entry**

The following sections describe the conditions under which exceptions occur due to TLB access failures.

Data-Storage Exception

When virtual mode is enabled, (MSR[VM]=1), a data-storage exception occurs when access to a page is not permitted for any of the following reasons:

- From user mode:
  - The TLB entry specifies a zone field that prevents access to the page (ZPR[Zn]=00). This applies to load and store instructions.
  - The TLB entry specifies a read-only page (TLBLO[WR]=0) that is not otherwise overridden by the zone field (ZPR[Zn], 11). This applies to store instructions.

- From privileged mode:
  - The TLB entry specifies a read-only page (TLBLO[WR]=0) that is not otherwise overridden by the zone field (ZPR[Zn], 10 and ZPR[Zn], 11). This applies to store instructions.

Instruction-Storage Exception

When virtual mode is enabled, (MSR[VM]=1), an instruction-storage exception occurs when access to a page is not permitted for any of the following reasons:

- From user mode:
  - The TLB entry specifies a zone field that prevents access to the page (ZPR[Zn]=00).
  - The TLB entry specifies a non-executable page (TLBLO[EX]=0) that is not otherwise overridden by the zone field (ZPR[Zn], 11).
  - The TLB entry specifies a guarded-storage page (TLBLO[G]=1).
- From privileged mode:
  - The TLB entry specifies a non-executable page (TLBLO[EX]=0) that is not otherwise overridden by the zone field (ZPR[Zn], 10 and ZPR[Zn], 11).
  - The TLB entry specifies a guarded-storage page (TLBLO[G]=1).

Data TLB-Miss Exception

When virtual mode is enabled (MSR[VM]=1) a data TLB-miss exception occurs if a valid, matching TLB entry was not found in the TLB (shadow and UTLB). Any load or store instruction can cause a data TLB-miss exception.

Instruction TLB-Miss Exception

When virtual mode is enabled (MSR[VM]=1) an instruction TLB-miss exception occurs if a valid, matching TLB entry was not found in the TLB (shadow and UTLB). Any instruction fetch can cause an instruction TLB-miss exception.

Access Protection

System software uses access protection to protect sensitive memory locations from improper access. System software can restrict memory accesses for both user-mode and privileged-mode software. Restrictions can be placed on reads, writes, and instruction fetches. Access protection is available when virtual protected mode is enabled.

Access control applies to instruction fetches, data loads, and data stores. The TLB entry for a virtual page specifies the type of access allowed to the page. The TLB entry also specifies a zone-protection field in the zone-protection register that is used to override the access controls specified by the TLB entry.

TLB Access-Protection Controls

Each TLB entry controls three types of access:

- Process—Processes are protected from unauthorized access by assigning a unique process ID (PID) to each process. When system software starts a user-mode application, it loads the PID for that application into the PID register. As the application executes, memory addresses are translated using only TLB entries with a TID field in Translation Look-Aside Buffer High (TLBHI) that matches the PID. This enables system software to restrict accesses for an application to a specific area in virtual memory.

A TLB entry with TID=0x00 represents a process-independent translation. Pages that are accessed globally by all processes should be assigned a TID value of 0x00.

- Execution—The processor executes instructions only if they are fetched from a virtual page marked as executable (TLBLO[EX]=1). Clearing TLBLO[EX] to 0 prevents execution of instructions fetched from a page, instead causing an instruction-storage interrupt (ISI) to occur. The ISI does not occur when the instruction is fetched, but instead occurs when the instruction is executed. This prevents speculatively fetched instructions that are later discarded (rather than executed) from causing an ISI.

  The zone-protection register can override execution protection.

- Read/Write—Data is written only to virtual pages marked as writable (TLBLO[WR]=1). Clearing TLBLO[WR] to 0 marks a page as read-only. An attempt to write to a read-only page causes a data-storage interrupt (DSI) to occur.

  The zone-protection register can override write protection.

TLB entries cannot be used to prevent programs from reading pages. In virtual mode, zone protection is used to read-protect pages. This is done by defining a no-access-allowed zone (ZPR[Zn] = 00) and using it to override the TLB-entry access protection. Only programs running in user mode can be prevented from reading a page. Privileged programs always have read access to a page.

Zone Protection

Zone protection is used to override the access protection specified in a TLB entry. Zones are an arbitrary grouping of virtual pages with common access protection. Zones can contain any number of pages specifying any combination of page sizes. There is no requirement for a zone to contain adjacent pages.

The zone-protection register (ZPR) is a 32-bit register used to specify the type of protection override applied to each of 16 possible zones. The protection override for a zone is encoded in the ZPR as a 2-bit field. The 4-bit zone-select field in a TLB entry (TLBLO[ZSEL]) selects one of the 16 zone fields from the ZPR (Z0–Z15). For example, zone Z5 is selected when ZSEL = 0101.

Changing a zone field in the ZPR applies a protection override across all pages in that zone. Without the ZPR, protection changes require individual alterations to each page translation entry within the zone.

UTLB Management

The UTLB serves as the interface between the processor MMU and memory-management software. System software manages the UTLB to tell the MMU how to translate virtual addresses into physical addresses. When a problem occurs due to a missing translation or an access violation, the MMU communicates the problem to system software using the exception mechanism. System software is responsible for providing interrupt handlers to correct these problems so that the MMU can proceed with memory translation.

Software reads and writes UTLB entries using the MFS and MTS instructions, respectively. These instructions use the TLBX register index (numbered 0 to 63) corresponding to one of the 64 entries in the UTLB. The tag and data portions are read and written separately, so software must execute two MFS or MTS instructions to completely access an entry. The UTLB is searched for a specific translation using the TLBSX register. TLBSX locates a translation using an effective address and loads the corresponding UTLB index into the TLBX register.

Individual UTLB entries are invalidated using the MTS instruction to clear the valid bit in the tag portion of a TLB entry (TLBHI[V]).

When C_FAULT_TOLERANT is set to 1, the UTLB block RAM is protected by parity. In case of a parity error, a TLB miss exception occurs. To avoid accumulating errors in this case, each entry in the UTLB should be periodically invalidated.

Recording Page Access and Page Modification

Software management of virtual-memory poses several challenges:

- In a virtual-memory environment, software and data often consume more memory than is physically available. Some of the software and data pages must be stored outside physical memory, such as on a hard drive, when they are not used. Ideally, the most-frequently used pages stay in physical memory and infrequently used pages are stored elsewhere.
- When pages in physical-memory are replaced to make room for new pages, it is important to know whether the replaced (old) pages were modified. If they were modified, they must be saved prior to loading the replacement (new) pages. If the old pages were not modified, the new pages can be loaded without saving the old pages.
- A limited number of page translations are kept in the UTLB. The remaining translations must be stored in the page-translation table. When a translation is not found in the UTLB (due to a miss), system software must decide which UTLB entry to discard so that the missing translation can be loaded. It is desirable for system software to replace infrequently used translations rather than frequently used translations.

Solving the above problems in an efficient manner requires keeping track of page accesses and page modifications. MicroBlaze does not track page access and page modification in hardware. Instead, system software can use the TLB-miss exceptions and the data-storage exception to collect this information. As the information is collected, it can be stored in a data structure associated with the page-translation table.

Page-access information is used to determine which pages should be kept in physical memory and which are replaced when physical-memory space is required. System software can use the valid bit in the TLB entry (TLBHI[V]) to monitor page accesses. This requires page translations be initialized as not valid (TLBHI[V]=0) to indicate they have not been accessed. The first attempt to access a page causes a TLB-miss exception, either because the UTLB entry is marked not valid or because the page translation is not present in the UTLB. The TLB-miss handler updates the UTLB with a valid translation (TLBHI[V]=1). The set valid bit serves as a record that the page and its translation have been accessed. The TLB-miss handler can also record the information in a separate data structure associated with the page-translation entry.

Page-modification information is used to indicate whether an old page can be overwritten with a new page or the old page must first be stored to a hard disk. System software can use the write-protection bit in the TLB entry (TLBLO[WR]) to monitor page modification. This requires page translations be initialized as read-only (TLBLO[WR]=0) to indicate they have not been modified. The first attempt to write data into a page causes a data-storage exception, assuming the page has already been accessed and marked valid as described above. If software has permission to write into the page, the data-storage handler marks the page as writable (TLBLO[WR]=1) and returns. The set write-protection bit serves as a record that a page has been modified. The data-storage handler can also record this information in a separate data structure associated with the page-translation entry.

Tracking page modification is useful when virtual mode is first entered and when a new process is started.

Reset, Interrupts, Exceptions, and Break

MicroBlaze supports reset, interrupt, user exception, break, and hardware exceptions. The following section describes the execution flow associated with each of these events.

The relative priority starting with the highest is:

1. Reset
2. Hardware Exception
3. Non-maskable Break
4. Break
5. Interrupt
6. User Vector (Exception)

Table 1-37 defines the memory address locations of the associated vectors and the hardware enforced register file locations for return addresses. Each vector allocates two addresses to allow full address range branching (requires an `IMM` followed by a `BRAI` instruction). The address range 0x28 to 0x4F is reserved for future software support by Xilinx. Allocating these addresses for user applications is likely to conflict with future releases of EDK support software.

Table 1-37: **Vectors and Return Address Register File Location**

Event	Vector Address	Register File Return Address
Reset	0x00000000 - 0x00000004	-
User Vector (Exception)	0x00000008 - 0x0000000C	Rx
Interrupt	0x00000010 - 0x00000014	R14
Break: Non-maskable hardware	0x00000018 - 0x0000001C	R16
Break: Hardware		
Break: Software		
Hardware Exception	0x00000020 - 0x00000024	R17 or BTR
Reserved by Xilinx for future use	0x00000028 - 0x0000004F	-

All of these events will clear the reservation bit, used together with the LWX and SWX instructions to implement mutual exclusion, such as semaphores and spinlocks.

Reset

When a `Reset` or `Debug_Rst`[1] occurs, MicroBlaze flushes the pipeline and starts fetching instructions from the reset vector (address 0x0). Both external reset signals are active high and should be asserted for a minimum of 16 cycles.

Equivalent Pseudocode

```
PC ← 0x00000000
MSR ← C_RESET_MSR (see "MicroBlaze Core Configurability" in Chapter 2)
EAR ← 0; ESR ← 0; FSR ← 0
PID ← 0; ZPR ← 0; TLBX ← 0
Reservation ← 0
```

1. Reset input controlled by the XMD debugger via MDM.

Hardware Exceptions

MicroBlaze can be configured to trap the following internal error conditions: illegal instruction, instruction and data bus error, and unaligned access. The divide exception can only be enabled if the processor is configured with a hardware divider (`C_USE_DIV=1`). When configured with a hardware floating point unit (`C_USE_FPU>0`), it can also trap the following floating point specific exceptions: underflow, overflow, float division-by-zero, invalid operation, and denormalized operand error.

When configured with a hardware Memory Management Unit, it can also trap the following memory management specific exceptions: Illegal Instruction Exception, Data Storage Exception, Instruction Storage Exception, Data TLB Miss Exception, and Instruction TLB Miss Exception.

A hardware exception causes MicroBlaze to flush the pipeline and branch to the hardware exception vector (address 0x20). The execution stage instruction in the exception cycle is not executed.

The exception also updates the general purpose register R17 in the following manner:

- For the MMU exceptions (Data Storage Exception, Instruction Storage Exception, Data TLB Miss Exception, Instruction TLB Miss Exception) the register R17 is loaded with the appropriate program counter value to re-execute the instruction causing the exception upon return. The value is adjusted to return to a preceding `IMM` instruction, if any. If the exception is caused by an instruction in a branch delay slot, the value is adjusted to return to the branch instruction, including adjustment for a preceding `IMM` instruction, if any.
- For all other exceptions the register R17 is loaded with the program counter value of the subsequent instruction, unless the exception is caused by an instruction in a branch delay slot. If the exception is caused by an instruction in a branch delay slot, the ESR[DS] bit is set. In this case the exception handler should resume execution from the branch target address stored in BTR.

The EE and EIP bits in MSR are automatically reverted when executing the `RTED` instruction.

The VM and UM bits in MSR are automatically reverted from VMS and UMS when executing the `RTED`, `RTBD`, and `RTID` instructions.

Exception Priority

When two or more exceptions occur simultaneously, they are handled in the following order, from the highest priority to the lowest:

- Instruction Bus Exception
- Instruction TLB Miss Exception
- Instruction Storage Exception
- Illegal Opcode Exception
- Privileged Instruction Exception or Stack Protection Violation Exception
- Data TLB Miss Exception
- Data Storage Exception
- Unaligned Exception
- Data Bus Exception
- Divide Exception
- FPU Exception
- Stream Exception

Exception Causes

- Stream Exception

  The stream exception (FSL or AXI) is caused by executing a `get` or `getd` instruction with the 'e' bit set to '1' when there is a control bit mismatch.

- Instruction Bus Exception

  The instruction bus exception is caused by errors when reading data from memory.

  - The instruction peripheral AXI4 interface (M_AXI_IP) exception is caused by an error response on `M_AXI_IP_RRESP`.
  - The instruction cache AXI4 interface (M_AXI_IC) is caused by an error response on `M_AXI_IC_RRESP`. The exception can only occur when `C_ICACHE_ALWAYS_USED` is set to 1 and the cache is turned off. In all other cases the response is ignored.
  - The instruction Processor Local Bus (PLB) exception is caused by an active error signal from the slave (`IPLB_MRdErr`) or timeout signal from the arbiter (`IPLB_MTimeout`).
  - The instructions side local memory (ILMB) can only cause instruction bus exception when `C_FAULT_TOLERANT` is set to 1, and either an uncorrectable error occurs in the LMB memory, as indicated by the `IUE` signal, or `C_ECC_USE_CE_EXCEPTION` is set to 1 and a correctable error occurs in the LMB memory, as indicated by the `ICE` signal.
  - The CacheLink (IXCL) interfaces cannot cause instruction bus exceptions.

- Illegal Opcode Exception

  The illegal opcode exception is caused by an instruction with an invalid major opcode (bits 0 through 5 of instruction). Bits 6 through 31 of the instruction are not checked. Optional processor instructions are detected as illegal if not enabled. If the optional feature `C_OPCODE_0x0_ILLEGAL` is enabled, an illegal opcode exception is also caused if the instruction is equal to 0x00000000.

- Data Bus Exception

  The data bus exception is caused by errors when reading data from memory or writing data to memory.

  - The data peripheral AXI4 interface (M_AXI_DP) exception is caused by an error response on `M_AXI_DP_RRESP` or `M_AXI_DP_BRESP`.
  - The data cache AXI4 interface (M_AXI_DC) exception is caused by:
    - An error response on `M_AXI_DC_RRESP` or `M_AXI_DP_BRESP`,
    - `OKAY` response on `M_AXI_DC_RRESP` in case of an exclusive access using `LWX`.

    The exception can only occur when `C_DCACHE_ALWAYS_USED` is set to 1 and the cache is turned off, or when an exclusive access using `LWX` or `SWX` is performed. In all other cases the response is ignored.
  - The data Processor Local Bus exception is caused by an active error signal from the slave (`DPLB_MRdErr` or `DPLB_MWrErr`) or timeout signal from the arbiter (`DPLB_MTimeout`).
  - The data side local memory (DLMB) can only cause instruction bus exception when `C_FAULT_TOLERANT` is set to 1, and either an uncorrectable error occurs in the LMB memory, as indicated by the `DUE` signal, or `C_ECC_USE_CE_EXCEPTION` is set to 1 and a correctable error occurs in the LMB memory, as indicated by the `DCE` signal. An error can occur for all read accesses, and for byte and halfword write accesses.
  - The CacheLink (DXCL) interfaces cannot cause data bus exceptions.

- Unaligned Exception

  The unaligned exception is caused by a word access where the address to the data bus has bits 30 or 31 set, or a half-word access with bit 31 set.

- Divide Exception

  The divide exception is caused by an integer division (idiv or idivu) where the divisor is zero, or by a signed integer division (idiv) where overflow occurs (-2147483648 / -1).

- FPU Exception

  An FPU exception is caused by an underflow, overflow, divide-by-zero, illegal operation, or denormalized operand occurring with a floating point instruction.

  - Underflow occurs when the result is denormalized.
  - Overflow occurs when the result is not-a-number (NaN).
  - The divide-by-zero FPU exception is caused by the rA operand to fdiv being zero when rB is not infinite.
  - Illegal operation is caused by a signaling NaN operand or by illegal infinite or zero operand combinations.

- Privileged Instruction Exception

  The Privileged Instruction exception is caused by an attempt to execute a privileged instruction in User Mode.

- Stack Protection Violation Exception

  A Stack Protection Violation exception is caused by executing a load or store instruction using the stack pointer (register R1) as rA with an address outside the stack boundaries defined by the special Stack Low and Stack High registers, causing a stack overflow or a stack underflow.

- Data Storage Exception

  The Data Storage exception is caused by an attempt to access data in memory that results in a memory-protection violation.

- Instruction Storage Exception

  The Instruction Storage exception is caused by an attempt to access instructions in memory that results in a memory-protection violation.

- Data TLB Miss Exception

  The Data TLB Miss exception is caused by an attempt to access data in memory, when a valid Translation Look-Aside Buffer entry is not present, and virtual protected mode is enabled.

- Instruction TLB Miss Exception

  The Instruction TLB Miss exception is caused by an attempt to access instructions in memory, when a valid Translation Look-Aside Buffer entry is not present, and virtual protected mode is enabled.

Should an Instruction Bus Exception, Illegal Opcode Exception or Data Bus Exception occur when `C_FAULT_TOLERANT` is set to 1, and an exception is in progress (i.e. MSR[EIP] set and MSR[EE] cleared), the pipeline is halted, and the external signal `MB_Error` is set.

Equivalent Pseudocode

```
ESR[DS] ← exception in delay slot
if ESR[DS] then
    BTR ← branch target PC
    if MMU exception then
        if branch preceded by IMM then
            r17 ← PC - 8
        else
            r17 ← PC - 4
    else
        r17 ← invalid value
else if MMU exception then
    if instruction preceded by IMM then
        r17 ← PC - 4
    else
        r17 ← PC
else
    r17 ← PC + 4
PC ← 0x00000020
MSR[EE]  ← 0, MSR[EIP]← 1
MSR[UMS] ← MSR[UM], MSR[UM] ← 0, MSR[VMS] ← MSR[VM], MSR[VM] ← 0
ESR[EC]  ← exception specific value
ESR[ESS] ← exception specific value
EAR ← exception specific value
FSR ← exception specific value
Reservation ← 0
```

Breaks

There are two kinds of breaks:

- Hardware (external) breaks
- Software (internal) breaks

Hardware Breaks

Hardware breaks are performed by asserting the external break signal (that is, the `Ext_BRK` and `Ext_NM_BRK` input ports). On a break, the instruction in the execution stage completes while the instruction in the decode stage is replaced by a branch to the break vector (address 0x18). The break return address (the PC associated with the instruction in the decode stage at the time of the break) is automatically loaded into general purpose register R16. MicroBlaze also sets the Break In Progress (`BIP`) flag in the Machine Status Register (MSR).

A normal hardware break (that is, the Ext_BRK input port) is only handled when MSR[BIP] and MSR[EIP] are set to 0 (that is, there is no break or exception in progress). The Break In Progress flag disables interrupts. A non-maskable break (that is, the `Ext_NM_BRK` input port) is always handled immediately.

The BIP bit in the MSR is automatically cleared when executing the `RTBD` instruction.

The `Ext_BRK` signal must be kept asserted until the break has occurred, and deasserted before the `RTBD` instruction is executed. The `Ext_NM_BRK` signal must only be asserted one clock cycle.

Software Breaks

To perform a software break, use the `brk` and `brki` instructions. Refer to Chapter 4, "MicroBlaze Instruction Set Architecture" for detailed information on software breaks.

Latency

The time it takes MicroBlaze to enter a break service routine from the time the break occurs depends on the instruction currently in the execution stage and the latency to the memory storing the break vector.

Equivalent Pseudocode

```
r16 ← PC
PC ← 0x00000018
MSR[BIP] ← 1
MSR[UMS] ← MSR[UM], MSR[UM] ← 0, MSR[VMS] ← MSR[VM], MSR[VM] ← 0
Reservation ← 0
```

Interrupt

MicroBlaze supports one external interrupt source (connected to the `Interrupt` input port). The processor only reacts to interrupts if the Interrupt Enable (IE) bit in the Machine Status Register (MSR) is set to 1. On an interrupt, the instruction in the execution stage completes while the instruction in the decode stage is replaced by a branch to the interrupt vector (address 0x10). The interrupt return address (the PC associated with the instruction in the decode stage at the time of the interrupt) is automatically loaded into general purpose register R14. In addition, the processor also disables future interrupts by clearing the IE bit in the MSR. The IE bit is automatically set again when executing the RTID instruction.

Interrupts are ignored by the processor if either of the break in progress (`BIP`) or exception in progress (`EIP`) bits in the MSR are set to 1.

By using the parameter `C_INTERRUPT_IS_EDGE`, the external interrupt can either be set to level-sensitive or edge-sensitive:

- When using level-sensitive interrupts, the `Interrupt` input must remain set until MicroBlaze has taken the interrupt, and jumped to the interrupt vector. Software must clear the interrupt before returning from the interrupt handler. If not, the interrupt is taken again, as soon as interrupts are enabled when returning from the interrupt handler.
- When using edge-sensitive interrupts, MicroBlaze detects and latches the `Interrupt` input edge, which means that the input only needs to be asserted one clock cycle. The interrupt input can remain asserted, but must be deasserted at least one clock cycle before a new interrupt can be detected. The latching of an edge sensitive interrupt is independent of the IE bit in MSR. Should an interrupt occur while the IE bit is 0, it will immediately be serviced when the IE bit is set to 1.

Latency

The time it takes MicroBlaze to enter an Interrupt Service Routine (ISR) from the time an interrupt occurs, depends on the configuration of the processor and the latency of the memory controller storing the interrupt vectors. If MicroBlaze is configured to have a hardware divider, the largest latency happens when an interrupt occurs during the execution of a division instruction.

Equivalent Pseudocode

```
r14 ← PC
PC ← 0x00000010
MSR[IE] ← 0
MSR[UMS] ← MSR[UM], MSR[UM] ← 0, MSR[VMS] ← MSR[VM], MSR[VM] ← 0
Reservation ← 0
```

User Vector (Exception)

The user exception vector is located at address 0x8. A user exception is caused by inserting a ‘BRALID Rx,0x8’ instruction in the software flow. Although R*x* could be any general purpose register, Xilinx recommends using R15 for storing the user exception return address, and to use the RTSD instruction to return from the user exception handler.

Pseudocode

```
rx ← PC
PC ← 0x00000008
MSR[UMS] ← MSR[UM], MSR[UM] ← 0, MSR[VMS] ← MSR[VM], MSR[VM] ← 0
Reservation ← 0
```

Instruction Cache

Overview

MicroBlaze can be used with an optional instruction cache for improved performance when executing code that resides outside the LMB address range.

The instruction cache has the following features:

- Direct mapped (1-way associative)
- User selectable cacheable memory address range
- Configurable cache and tag size
- Caching over AXI4 interface (M_AXI_IC) or CacheLink (XCL) interface
- Option to use 4 or 8 word cache-line
- Cache on and off controlled using a bit in the MSR
- Optional WIC instruction to invalidate instruction cache lines
- Optional stream buffers to improve performance by speculatively prefetching instructions
- Optional victim cache to improve performance by saving evicted cache lines
- Optional parity protection that invalidates cache lines if a Block RAM bit error is detected
- Optional data width selection to either use 32 bits or an entire cache line

General Instruction Cache Functionality

When the instruction cache is used, the memory address space is split into two segments: a cacheable segment and a non-cacheable segment. The cacheable segment is determined by two parameters: `C_ICACHE_BASEADDR` and `C_ICACHE_HIGHADDR`. All addresses within this range correspond to the cacheable address segment. All other addresses are non-cacheable.

The cacheable segment size must be 2^N, where N is a positive integer. The range specified by `C_ICACHE_BASEADDR` and `C_ICACHE_HIGHADDR` must comprise a complete power-of-two range, such that range = 2^N and the N least significant bits of `C_ICACHE_BASEADDR` must be zero.

The cacheable instruction address consists of two parts: the cache address, and the tag address. The MicroBlaze instruction cache can be configured from 64 bytes to 64 kB. This corresponds to a cache address of between 6 and 16 bits. The tag address together with the cache address should match the full address of cacheable memory. When selecting cache sizes below 2 kB, distributed RAM is used to implement the Tag RAM and Instruction RAM. Distributed RAM is always used to implement the Tag RAM, when setting the parameter `C_ICACHE_FORCE_TAG_LUTRAM` to 1. This parameter is only available with cache sizes 8 kB or 16 kB and less, for 4 or 8 word cache-lines, respectively.

For example: in a MicroBlaze configured with `C_ICACHE_BASEADDR= 0x00300000`, `C_ICACHE_HIGHADDR=0x0030ffff`, `C_CACHE_BYTE_SIZE=4096`, `C_ICACHE_LINE_LEN=8`, and `C_ICACHE_FORCE_TAG_LUTRAM=0`; the cacheable memory of 64 kB uses 16 bits of byte address, and the 4 kB cache uses 12 bits of byte address, thus the required address tag width is: 16-12=4 bits. The total number of block RAM primitives required in this configuration is: 2 RAMB16 for storing the 1024 instruction words, and 1 RAMB16 for 128 cache line entries, each consisting of: 4 bits of tag, 8 word-valid bits, 1 line-valid bit. In total 3 RAMB16 primitives.

Figure 1-22, page 75 shows the organization of Instruction Cache.

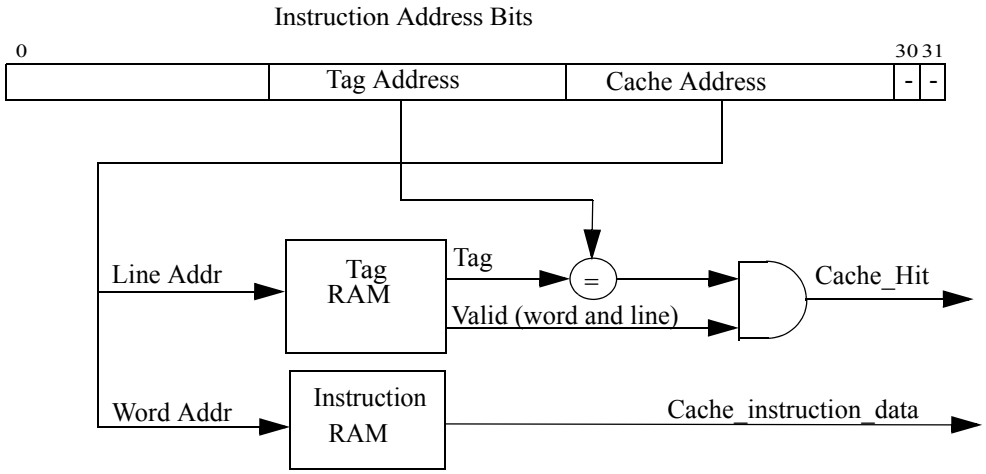


Figure 1-22: **Instruction Cache Organization**

Instruction Cache Operation

For every instruction fetched, the instruction cache detects if the instruction address belongs to the cacheable segment. If the address is non-cacheable, the cache controller ignores the instruction and lets the M_AXI_IP, PLB or LMB complete the request. If the address is cacheable, a lookup is performed on the tag memory to check if the requested address is currently cached. The lookup is successful if: the word and line valid bits are set, and the tag address matches the instruction address tag segment. On a cache miss, the cache controller requests the new instruction over the instruction AXI4 interface (M_AXI_IC) or instruction CacheLink (IXCL) interface, and waits for the memory controller to return the associated cache line.

With the AXI4 interface, `C_ICACHE_DATA_WIDTH` determines the bus data width, either 32 bits or an entire cache line (128 bits or 256 bits).

When `C_FAULT_TOLERANT` is set to 1, a cache miss also occurs if a parity error is detected in a tag or instruction Block RAM.

Stream Buffers

When stream buffers are enabled, by setting the parameter `C_ICACHE_STREAMS` to 1, the cache will speculatively fetch cache lines in advance in sequence following the last requested address, until the stream buffer is full. The stream buffer can hold up to two cache lines. Should the processor subsequently request instructions from a cache line prefetched by the stream buffer, which occurs in linear code, they are immediately available.

The stream buffer often improves performance, since the processor generally has to spend less time waiting for instructions to be fetched from memory.

With the AXI4 interface, `C_ICACHE_DATA_WIDTH` determines the amount of data transferred from the stream buffer each clock cycle, either 32 bits or an entire cache line.

To be able to use instruction cache stream buffers, area optimization must not be enabled.

Victim Cache

The victim cache is enabled by setting the parameter C_ICACHE_VICTIMS to 2, 4 or 8. This defines the number of cache lines that can be stored in the victim cache. Whenever a cache line is evicted from the cache, it is saved in the victim cache. By saving the most recent lines they can be fetched much faster, should the processor request them, thereby improving performance. If the victim cache is not used, all evicted cache lines must be read from memory again when they are needed.

With the AXI4 interface, C_ICACHE_DATA_WIDTH determines the amount of data transferred from/to the victim cache each clock cycle, either 32 bits or an entire cache line.

Note that to be able to use the victim cache, area optimization must not be enabled.

Instruction Cache Software Support

MSR Bit

The ICE bit in the MSR provides software control to enable and disable caches.

The contents of the cache are preserved by default when the cache is disabled. You can invalidate cache lines using the WIC instruction or using the hardware debug logic of MicroBlaze.

WIC Instruction

The optional WIC instruction (C_ALLOW_ICACHE_WR=1) is used to invalidate cache lines in the instruction cache from an application. For a detailed description, refer to Chapter 4, "MicroBlaze Instruction Set Architecture".

The WIC instruction can also be used together with parity protection to periodically invalidate entries the cache, to avoid accumulating errors.

Data Cache

Overview

MicroBlaze can be used with an optional data cache for improved performance. The cached memory range must not include addresses in the LMB address range. The data cache has the following features:

- Direct mapped (1-way associative)
- Write-through or Write-back
- User selectable cacheable memory address range
- Configurable cache size and tag size
- Caching over AXI4 interface (M_AXI_DC) or CacheLink (XCL) interface
- Option to use 4 or 8 word cache-lines
- Cache on and off controlled using a bit in the MSR
- Optional WDC instruction to invalidate or flush data cache lines
- Optional victim cache with write-back to improve performance by saving evicted cache lines
- Optional parity protection for write-through cache that invalidates cache lines if a Block RAM bit error is detected
- Optional data width selection to either use 32 bits or an entire cache line

General Data Cache Functionality

When the data cache is used, the memory address space is split into two segments: a cacheable segment and a non-cacheable segment. The cacheable area is determined by two parameters: C_DCACHE_BASEADDR and C_DCACHE_HIGHADDR. All addresses within this range correspond to the cacheable address space. All other addresses are non-cacheable.

The cacheable segment size must be 2^N, where N is a positive integer. The range specified by C_DCACHE_BASEADDR and C_DCACHE_HIGHADDR must comprise a complete power-of-two range, such that range = 2^N and the N least significant bits of C_DCACHE_BASEADDR must be zero.

Figure 1-23 shows the Data Cache Organization.

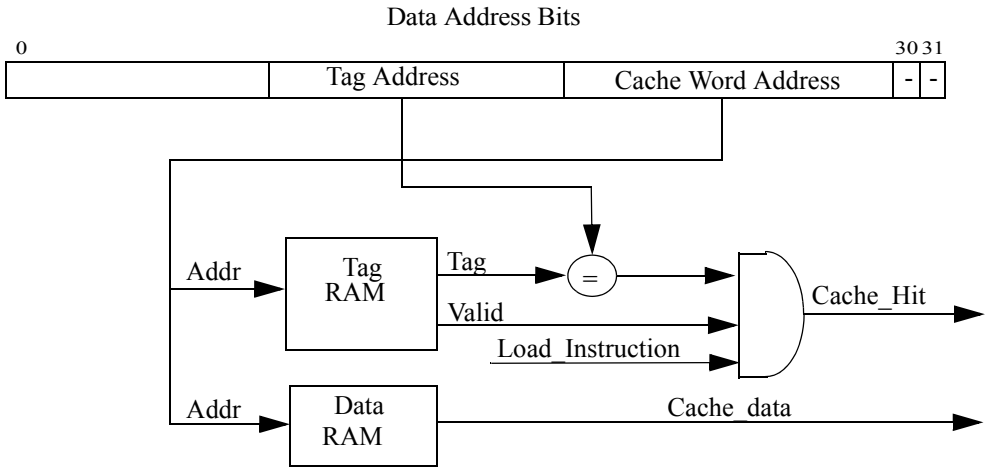


Figure 1-23: **Data Cache Organization**

The cacheable data address consists of two parts: the cache address, and the tag address. The MicroBlaze data cache can be configured from 64 bytes to 64 kB. This corresponds to a cache address of between 6 and 16 bits. The tag address together with the cache address should match the full address of cacheable memory. When selecting cache sizes below 2 kB, distributed RAM is used to implement the Tag RAM and Data RAM, except that block RAM is always used for the Data RAM when C_AREA_OPTIMIZED is set and C_DCACHE_USE_WRITEBACK is not set. Distributed RAM is always used to implement the Tag RAM, when setting the parameter C_DCACHE_FORCE_TAG_LUTRAM to 1. This parameter is only available with cache sizes 8 kB or 16 kB and less, for 4 or 8 word cache-lines, respectively.

For example, in a MicroBlaze configured with C_DCACHE_BASEADDR=0x00400000, C_DCACHE_HIGHADDR=0x00403fff, C_DCACHE_BYTE_SIZE=2048, C_DCACHE_LINE_LEN=4, and C_DCACHE_FORCE_TAG_LUTRAM=0; the cacheable memory of 16 kB uses 14 bits of byte address, and the 2 kB cache uses 11 bits of byte address, thus the required address tag width is 14-11=3 bits. The total number of block RAM primitives required in this configuration is 1 RAMB16 for storing the 512 data words, and 1 RAMB16 for 128 cache line entries, each consisting of 3 bits of tag, 4 word-valid bits, 1 line-valid bit. In total, 2 RAMB16 primitives.

Data Cache Operation

The caching policy used by the MicroBlaze data cache, write-back or write-through, is determined by the parameter C_DCACHE_USE_WRITEBACK. When this parameter is set, a write-back protocol is implemented, otherwise write-through is implemented. However, when configured with an MMU (C_USE_MMU > 1, C_AREA_OPTIMIZED = 0, C_DCACHE_USE_WRITEBACK = 1), the caching policy in virtual mode is determined by the W storage attribute in the TLB entry, whereas write-back is used in real mode.

With the write-back protocol, a store to an address within the cacheable range always updates the cached data. If the target address word is not in the cache (that is, the access is a cache-miss), and the location in the cache contains data that has not yet been written to memory (the cache location is dirty), the old data is written over the data AXI4 interface (M_AXI_DC) or the data CacheLink (DXCL) to external memory before updating the cache with the new data. If an entire cache line needs to be written, a burst cache line write is used, otherwise single word writes are used. For byte or halfword stores, in case of a cache miss, the address is first requested over the data AXI4 interface or the data CacheLink, while a word store only updates the cache.

With the write-through protocol, a store to an address within the cacheable range generates an equivalent byte, halfword, or word write over the data AXI4 interface or the data CacheLink to external memory. The write also updates the cached data if the target address word is in the cache (that is, the write is a cache hit). A write cache-miss does not load the associated cache line into the cache.

Provided that the cache is enabled a load from an address within the cacheable range triggers a check to determine if the requested data is currently cached. If it is (that is, on a cache hit) the requested data is retrieved from the cache. If not (that is, on a cache miss) the address is requested over the data AXI4 interface or data CacheLink, and the processor pipeline stalls until the cache line associated to the requested address is returned from the external memory controller.

With the AXI4 interface, C_DCACHE_DATA_WIDTH determines the bus data width, either 32 bits or an entire cache line (128 bits or 256 bits).

When C_FAULT_TOLERANT is set to 1 and write-through protocol is used, a cache miss also occurs if a parity error is detected in the tag or data Block RAM.

Victim Cache

The victim cache is enabled by setting the parameter C_DCACHE_VICTIMS to 2, 4 or 8. This defines the number of cache lines that can be stored in the victim cache. Whenever a complete cache line is evicted from the cache, it is saved in the victim cache. By saving the most recent lines they can be fetched much faster, should the processor request them, thereby improving performance. If the victim cache is not used, all evicted cache lines must be read from memory again when they are needed.

With the AXI4 interface, C_DCACHE_DATA_WIDTH determines the amount of data transferred from/to the victim cache each clock cycle, either 32 bits or an entire cache line.

Note that to be able to use the victim cache, write-back must be enabled and area optimization must not be enabled.

Data Cache Software Support

MSR Bit

The DCE bit in the MSR controls whether or not the cache is enabled. When disabling caches the user must ensure that all the prior writes within the cacheable range have been completed in external

memory before reading back over M_AXI_DP or PLB. This can be done by writing to a semaphore immediately before turning off caches, and then in a loop poll until it has been written.

The contents of the cache are preserved when the cache is disabled.

WDC Instruction

The optional WDC instruction (`C_ALLOW_DCACHE_WR=1`) is used to invalidate or flush cache lines in the data cache from an application. For a detailed description, please refer to Chapter 4, "MicroBlaze Instruction Set Architecture".

The WDC instruction can also be used together with parity protection to periodically invalidate entries the cache, to avoid accumulating errors.

Floating Point Unit (FPU)

Overview

The MicroBlaze floating point unit is based on the IEEE 754 standard:

- Uses IEEE 754 single precision floating point format, including definitions for infinity, not-a-number (NaN), and zero
- Supports addition, subtraction, multiplication, division, comparison, conversion and square root instructions
- Implements round-to-nearest mode
- Generates sticky status bits for: underflow, overflow, divide-by-zero and invalid operation

For improved performance, the following non-standard simplifications are made:

- Denormalized(1) operands are not supported. A hardware floating point operation on a denormalized number returns a quiet NaN and sets the sticky denormalized operand error bit in FSR; see "Floating Point Status Register (FSR)" on page 37
- A denormalized result is stored as a signed 0 with the underflow bit set in FSR. This method is commonly referred to as Flush-to-Zero (FTZ)
- An operation on a quiet NaN returns the fixed NaN: 0xFFC00000, rather than one of the NaN operands
- Overflow as a result of a floating point operation always returns signed ∞

Format

An IEEE 754 single precision floating point number is composed of the following three fields:

1. 1-bit *sign*
2. 8-bit biased *exponent*
3. 23-bit *fraction* (a.k.a. mantissa or significand)

The fields are stored in a 32 bit word as defined in Figure 1-24:

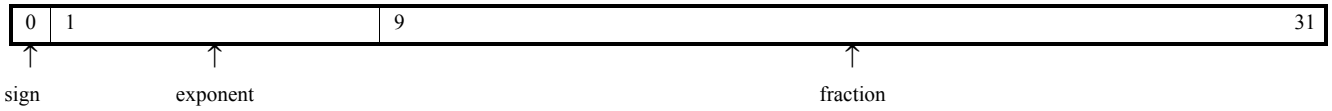


Figure 1-24: **IEEE 754 Single Precision Format**

The value of a floating point number v in MicroBlaze has the following interpretation:

1. If *exponent* = 255 and *fraction* <> 0, then v= NaN, regardless of the *sign* bit
2. If *exponent* = 255 and *fraction* = 0, then $v = (-1)^{sign} * \infty$
3. If 0 < *exponent* < 255, then $v = (-1)^{sign} * 2^{(exponent-127)} * (1.fraction)$
4. If *exponent* = 0 and *fraction* <> 0, then $v = (-1)^{sign} * 2^{-126} * (0.fraction)$
5. If *exponent* = 0 and *fraction* = 0, then $v = (-1)^{sign} * 0$

1. Numbers that are so close to 0, that they cannot be represented with full precision, that is, any number n that falls in the following ranges: ($1.17549*10^{-38} > n > 0$), or ($0 > n > -1.17549 * 10^{-38}$)

For practical purposes only 3 and 5 are useful, while the others all represent either an error or numbers that can no longer be represented with full precision in a 32 bit format.

Rounding

The MicroBlaze FPU only implements the default rounding mode, “Round-to-nearest”, specified in IEEE 754. By definition, the result of any floating point operation should return the nearest single precision value to the infinitely precise result. If the two nearest representable values are equally near, then the one with its least significant bit zero is returned.

Operations

All MicroBlaze FPU operations use the processors general purpose registers rather than a dedicated floating point register file, see “General Purpose Registers”.

Arithmetic

The FPU implements the following floating point operations:

- addition, fadd
- subtraction, fsub
- multiplication, fmul
- division, fdiv
- square root, fsqrt (available if `C_USE_FPU = 2`)

Comparison

The FPU implements the following floating point comparisons:

- compare less-than, fcmp.lt
- compare equal, fcmp.eq
- compare less-or-equal, fcmp.le
- compare greater-than, fcmp.gt
- compare not-equal, fcmp.ne
- compare greater-or-equal, fcmp.ge
- compare unordered, fcmp.un (used for NaN)

Conversion

The FPU implements the following conversions (available if `C_USE_FPU = 2`):

- convert from signed integer to floating point, flt
- convert from floating point to signed integer, fint

Exceptions

The floating point unit uses the regular hardware exception mechanism in MicroBlaze. When enabled, exceptions are thrown for all the IEEE standard conditions: underflow, overflow, divide-by-zero, and illegal operation, as well as for the MicroBlaze specific exception: denormalized operand error.

A floating point exception inhibits the write to the destination register (Rd). This allows a floating point exception handler to operate on the uncorrupted register file.

Software Support

The EDK compiler system, based on GCC, provides support for the Floating Point Unit compliant with the MicroBlaze API. Compiler flags are automatically added to the GCC command line based on the type of FPU present in the system, when using XPS or SDK.

All double-precision operations are emulated in software. Be aware that the xil_printf() function does not support floating-point output. The standard C library printf() and related functions do support floating-point output, but will increase the program code size.

Libraries and Binary Compatibility

The EDK compiler system only includes software floating point C runtime libraries. To take advantage of the hardware FPU, the libraries must be recompiled with the appropriate compiler switches.

For all cases where separate compilation is used, it is very important that you ensure the consistency of FPU compiler flags throughout the build.

Operator Latencies

The latencies of the various operations supported by the FPU are listed in Chapter 4, "MicroBlaze Instruction Set Architecture." The FPU instructions are not pipelined, so only one operation can be ongoing at any time.

C Language Programming

To gain maximum benefit from the FPU without low-level assembly-language programming, it is important to consider how the C compiler will interpret your source code. Very often the same algorithm can be expressed in many different ways, and some are more efficient than others.

Immediate Constants

Floating-point constants in C are double-precision by default. When using a single-precision FPU, careless coding may result in double-precision software emulation routines being used instead of the native single-precision instructions. To avoid this, explicitly specify (by cast or suffix) that immediate constants in your arithmetic expressions are single-precision values.

For example:

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

Note that the GNU C compiler can be instructed to treat all floating-point constants as single-precision (contrary to the ANSI C standard) by supplying the compiler flag -fsingle-precision-constants.

Avoid unnecessary casting

While conversions between floating-point and integer formats are supported in hardware by the FPU, when `C_USE_FPU` is set to 2, it is still best to avoid them when possible.

The following "bad" example calculates the sum of squares of the integers from 1 to 10 using floating-point representation:

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

The above code requires a cast from an integer to a float on each loop iteration. This can be rewritten as:

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

Note that the compiler is not at liberty to perform this optimization in general, as the two code fragments above may give different results in some cases (for example, very large t).

Square root runtime library function

The standard C runtime math library functions operate using double-precision arithmetic. When using a single-precision FPU, calls to the square root functions (sqrt()) result in inefficient emulation routines being used instead of FPU instructions:

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

Here the math.h header is included to avoid a warning message from the compiler.

When used with single-precision data types, the result is a cast to double, a runtime library call is made (which does not use the FPU) and then a truncation back to float is performed.

The solution is to use the non-ANSI function sqrtf() instead, which operates using single precision and can be carried out using the FPU. For example:

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

Note that when compiling this code, the compiler flag -fno-math-errno (in addition to -mhard-float and -mxl-float-sqrt) must be used, to ensure that the compiler does not generate unnecessary code to handle error conditions by updating the errno variable.

Stream Link Interfaces

MicroBlaze can be configured with up to 16 Fast Simplex Link (FSL) or AXI4-Stream interfaces, each consisting of one input and one output port. The channels are dedicated uni-directional point-to-point data streaming interfaces. The parameter `C_STREAM_INTERCONNECT` is used to select FSL or AXI4.

For detailed information on the FSL interface, please refer to the *Fast Simplex Link (FSL) Bus* data-sheet, DS449, in the Xilinx EDK IP Documentation. For detailed information on the AXI4-Stream interface, please refer to the *AMBA®4 AXI4-Stream Protocol Specification, Version 1.0* document.

The interfaces on MicroBlaze are 32 bits wide. A separate bit indicates whether the sent/received word is of control or data type. The get instruction in the MicroBlaze ISA is used to transfer information from a port to a general purpose register. The put instruction is used to transfer data in the opposite direction. Both instructions come in 4 flavors: blocking data, non-blocking data, blocking control, and non-blocking control. For a detailed description of the get and put instructions, please refer to Chapter 4, "MicroBlaze Instruction Set Architecture".

Hardware Acceleration

Each link provides a low latency dedicated interface to the processor pipeline. Thus they are ideal for extending the processors execution unit with custom hardware accelerators. A simple example is illustrated in Figure 1-25. The code uses RFSLx to indicate the used link, independent of whether FSL or AXI4-Stream is used.

Example code:

```
// Configure fx
cput Rc,RFSLx
// Store operands
put Ra, RFSLx // op 1
put Rb, RFSLx // op 2
// Load result
get Rt, RFSLx
```

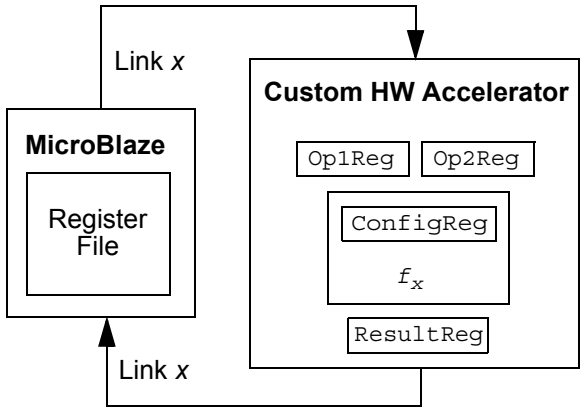


Figure 1-25: **Stream Link Used with HW Accelerated Function f_x**

This method is similar to extending the ISA with custom instructions, but has the benefit of not making the overall speed of the processor pipeline dependent on the custom function. Also, there are no additional requirements on the software tool chain associated with this type of functional extension.

Debug and Trace

Debug Overview

MicroBlaze features a debug interface to support JTAG based software debugging tools (commonly known as BDM or Background Debug Mode debuggers) like the Xilinx Microprocessor Debug (XMD) tool. The debug interface is designed to be connected to the Xilinx Microprocessor Debug Module (MDM) core, which interfaces with the JTAG port of Xilinx FPGAs. Multiple MicroBlaze instances can be interfaced with a single MDM to enable multiprocessor debugging. The debugging features include:

- Configurable number of hardware breakpoints and watchpoints and unlimited software breakpoints
- External processor control enables debug tools to stop, reset, and single step MicroBlaze
- Read from and write to: memory, general purpose registers, and special purpose register, except EAR, EDR, ESR, BTR and PVR0 - PVR11, which can only be read
- Support for multiple processors
- Write to instruction and data caches

Trace Overview

The MicroBlaze trace interface exports a number of internal state signals for performance monitoring and analysis. Xilinx recommends that users only use the trace interface through Xilinx developed analysis cores. This interface is not guaranteed to be backward compatible in future releases of MicroBlaze.

XILINX®

Chapter 2

MicroBlaze Signal Interface Description

This chapter describes the types of signal interfaces that can be used to connect MicroBlaze™. This chapter contains the following sections:

Overview

The MicroBlaze core is organized as a Harvard architecture with separate bus interface units for data and instruction accesses. The following four memory interfaces are supported: Local Memory Bus (LMB), the AMBA® AXI4 interface (AXI4), the IBM Processor Local Bus (PLB), and Xilinx® CacheLink (XCL). The LMB provides single-cycle access to on-chip dual-port block RAM. The AXI4 and PLB interfaces provide a connection to both on-chip and off-chip peripherals and memory. The CacheLink interface is intended for use with specialized external memory controllers. MicroBlaze also supports up to 16 Fast Simplex Link (FSL) or AXI4-Stream interface ports, each with one master and one slave interface.

Features

MicroBlaze can be configured with the following bus interfaces:

- The AMBA AXI4 Interface (see ARM® *AMBA® AXI Protocol Specification, Version 2.0, ARM IHI 0022C*), both for peripheral interfaces and cache interfaces.
- A 32-bit version of the PLB V4.6 interface (see IBM's *128-Bit Processor Local Bus Architectural Specifications, Version 4.6*).
- LMB provides simple synchronous protocol for efficient block RAM transfers
- FSL or AXI4-Stream provides a fast non-arbitrated streaming communication mechanism
- XCL provides a fast slave-side arbitrated streaming interface between caches and external memory controllers
- Debug interface for use with the Microprocessor Debug Module (MDM) core
- Trace interface for performance analysis

MicroBlaze I/O Overview

The core interfaces shown in Figure 2-1 and the following Table 2-1 are defined as follows:

M_AXI_DP: Peripheral Data Interface, AXI4-Lite or AXI4 interface
DPLB: Data interface, Processor Local Bus
DLMB: Data interface, Local Memory Bus (BRAM only)
M_AXI_IP: Peripheral Instruction interface, AXI4-Lite interface
IPLB: Instruction interface, Processor Local Bus
ILMB: Instruction interface, Local Memory Bus (BRAM only)
M0_AXIS..M15_AXIS: AXI4-Stream interface master direct connection interfaces
S0_AXIS..S15_AXIS: AXI4-Stream interface slave direct connection interfaces
MFSL 0..15: FSL master interfaces
DWFSL 0..15: FSL master direct connection interfaces
SFSL 0..15: FSL slave interfaces
DRFSL 0..15: FSL slave direct connection interfaces
DXCL: Data side Xilinx CacheLink interface (FSL master/slave pair)
M_AXI_DC: Data side cache AXI4 interface
IXCL: Instruction side Xilinx CacheLink interface (FSL master/slave pair)
M_AXI_IC: Instruction side cache AXI4 interface
Core: Miscellaneous signals for: clock, reset, debug, and trace

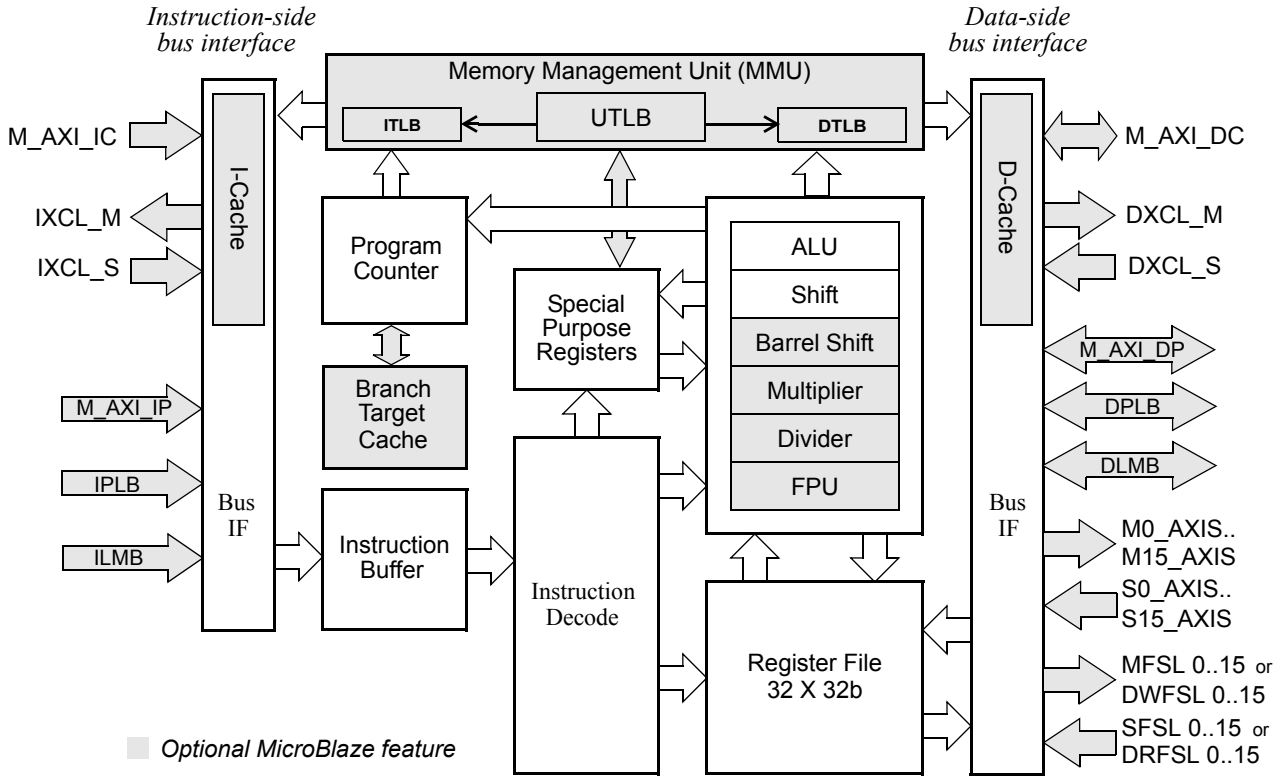


Figure 2-1: **MicroBlaze Core Block Diagram**

Table 2-1: **Summary of MicroBlaze Core I/O**

Signal	Interface	I/O	Description
M_AXI_DP_AWID	M_AXI_DP	O	Master Write address ID
M_AXI_DP_AWADDR	M_AXI_DP	O	Master Write address
M_AXI_DP_AWLEN	M_AXI_DP	O	Master Burst length
M_AXI_DP_AWSIZE	M_AXI_DP	O	Master Burst size
M_AXI_DP_AWBURST	M_AXI_DP	O	Master Burst type
M_AXI_DP_AWLOCK	M_AXI_DP	O	Master Lock type
M_AXI_DP_AWCACHE	M_AXI_DP	O	Master Cache type
M_AXI_DP_AWPROT	M_AXI_DP	O	Master Protection type
M_AXI_DP_AWQOS	M_AXI_DP	O	Master Quality of Service
M_AXI_DP_AWVALID	M_AXI_DP	O	Master Write address valid
M_AXI_DP_AWREADY	M_AXI_DP	I	Slave Write address ready
M_AXI_DP_WDATA	M_AXI_DP	O	Master Write data
M_AXI_DP_WSTRB	M_AXI_DP	O	Master Write strobes
M_AXI_DP_WLAST	M_AXI_DP	O	Master Write last
M_AXI_DP_WVALID	M_AXI_DP	O	Master Write valid
M_AXI_DP_WREADY	M_AXI_DP	I	Slave Write ready
M_AXI_DP_BID	M_AXI_DP	I	Slave Response ID
M_AXI_DP_BRESP	M_AXI_DP	I	Slave Write response
M_AXI_DP_BVALID	M_AXI_DP	I	Slave Write response valid
M_AXI_DP_BREADY	M_AXI_DP	O	Master Response ready
M_AXI_DP_ARID	M_AXI_DP	O	Master Read address ID
M_AXI_DP_ARADDR	M_AXI_DP	O	Master Read address
M_AXI_DP_ARLEN	M_AXI_DP	O	Master Burst length
M_AXI_DP_ARSIZE	M_AXI_DP	O	Master Burst size
M_AXI_DP_ARBURST	M_AXI_DP	O	Master Burst type
M_AXI_DP_ARLOCK	M_AXI_DP	O	Master Lock type
M_AXI_DP_ARCACHE	M_AXI_DP	O	Master Cache type
M_AXI_DP_ARPROT	M_AXI_DP	O	Master Protection type
M_AXI_DP_ARQOS	M_AXI_DP	O	Master Quality of Service
M_AXI_DP_ARVALID	M_AXI_DP	O	Master Read address valid
M_AXI_DP_ARREADY	M_AXI_DP	I	Slave Read address ready

Table 2-1: **Summary of MicroBlaze Core I/O *(Continued)***

Signal	Interface	I/O	Description
M_AXI_DP_RID	M_AXI_DP	I	Slave Read ID tag
M_AXI_DP_RDATA	M_AXI_DP	I	Slave Read data
M_AXI_DP_RRESP	M_AXI_DP	I	Slave Read response
M_AXI_DP_RLAST	M_AXI_DP	I	Slave Read last
M_AXI_DP_RVALID	M_AXI_DP	I	Slave Read valid
M_AXI_DP_RREADY	M_AXI_DP	O	Master Read ready
M_AXI_IP_AWID	M_AXI_IP	O	Master Write address ID
M_AXI_IP_AWADDR	M_AXI_IP	O	Master Write address
M_AXI_IP_AWLEN	M_AXI_IP	O	Master Burst length
M_AXI_IP_AWSIZE	M_AXI_IP	O	Master Burst size
M_AXI_IP_AWBURST	M_AXI_IP	O	Master Burst type
M_AXI_IP_AWLOCK	M_AXI_IP	O	Master Lock type
M_AXI_IP_AWCACHE	M_AXI_IP	O	Master Cache type
M_AXI_IP_AWPROT	M_AXI_IP	O	Master Protection type
M_AXI_IP_AWQOS	M_AXI_IP	O	Master Quality of Service
M_AXI_IP_AWVALID	M_AXI_IP	O	Master Write address valid
M_AXI_IP_AWREADY	M_AXI_IP	I	Slave Write address ready
M_AXI_IP_WDATA	M_AXI_IP	O	Master Write data
M_AXI_IP_WSTRB	M_AXI_IP	O	Master Write strobes
M_AXI_IP_WLAST	M_AXI_IP	O	Master Write last
M_AXI_IP_WVALID	M_AXI_IP	O	Master Write valid
M_AXI_IP_WREADY	M_AXI_IP	I	Slave Write ready
M_AXI_IP_BID	M_AXI_IP	I	Slave Response ID
M_AXI_IP_BRESP	M_AXI_IP	I	Slave Write response
M_AXI_IP_BVALID	M_AXI_IP	I	Slave Write response valid
M_AXI_IP_BREADY	M_AXI_IP	O	Master Response ready
M_AXI_IP_ARID	M_AXI_IP	O	Master Read address ID
M_AXI_IP_ARADDR	M_AXI_IP	O	Master Read address
M_AXI_IP_ARLEN	M_AXI_IP	O	Master Burst length
M_AXI_IP_ARSIZE	M_AXI_IP	O	Master Burst size
M_AXI_IP_ARBURST	M_AXI_IP	O	Master Burst type
M_AXI_IP_ARLOCK	M_AXI_IP	O	Master Lock type

Table 2-1: **Summary of MicroBlaze Core I/O *(Continued)***

Signal	Interface	I/O	Description
M_AXI_IP_ARCACHE	M_AXI_IP	O	Master Cache type
M_AXI_IP_ARPROT	M_AXI_IP	O	Master Protection type
M_AXI_IP_ARQOS	M_AXI_IP	O	Master Quality of Service
M_AXI_IP_ARVALID	M_AXI_IP	O	Master Read address valid
M_AXI_IP_ARREADY	M_AXI_IP	I	Slave Read address ready
M_AXI_IP_RID	M_AXI_IP	I	Slave Read ID tag
M_AXI_IP_RDATA	M_AXI_IP	I	Slave Read data
M_AXI_IP_RRESP	M_AXI_IP	I	Slave Read response
M_AXI_IP_RLAST	M_AXI_IP	I	Slave Read last
M_AXI_IP_RVALID	M_AXI_IP	I	Slave Read valid
M_AXI_IP_RREADY	M_AXI_IP	O	Master Read ready
M_AXI_DC_AWADDR	M_AXI_DC	O	Master Write address
M_AXI_DC_AWLEN	M_AXI_DC	O	Master Burst length
M_AXI_DC_AWSIZE	M_AXI_DC	O	Master Burst size
M_AXI_DC_AWBURST	M_AXI_DC	O	Master Burst type
M_AXI_DC_AWLOCK	M_AXI_DC	O	Master Lock type
M_AXI_DC_AWCACHE	M_AXI_DC	O	Master Cache type
M_AXI_DC_AWPROT	M_AXI_DC	O	Master Protection type
M_AXI_DC_AWQOS	M_AXI_DC	O	Master Quality of Service
M_AXI_DC_AWVALID	M_AXI_DC	O	Master Write address valid
M_AXI_DC_AWREADY	M_AXI_DC	I	Slave Write address ready
M_AXI_DC_AWUSER	M_AXI_DC	O	Master Write address user signals
M_AXI_DC_WDATA	M_AXI_DC	O	Master Write data
M_AXI_DC_WSTRB	M_AXI_DC	O	Master Write strobes
M_AXI_DC_WLAST	M_AXI_DC	O	Master Write last
M_AXI_DC_WVALID	M_AXI_DC	O	Master Write valid
M_AXI_DC_WREADY	M_AXI_DC	I	Slave Write ready
M_AXI_DC_WUSER	M_AXI_DC	O	Master Write user signals
M_AXI_DC_BRESP	M_AXI_DC	I	Slave Write response
M_AXI_DC_BID	M_AXI_DC	I	Slave Response ID
M_AXI_DC_BVALID	M_AXI_DC	I	Slave Write response valid
M_AXI_DC_BREADY	M_AXI_DC	O	Master Response ready

Table 2-1: **Summary of MicroBlaze Core I/O *(Continued)***

Signal	Interface	I/O	Description
M_AXI_DC_BUSER	M_AXI_DC	I	Slave Write response user signals
M_AXI_DC_ARID	M_AXI_DC	O	Master Read address ID
M_AXI_DC_ARADDR	M_AXI_DC	O	Master Read address
M_AXI_DC_ARLEN	M_AXI_DC	O	Master Burst length
M_AXI_DC_ARSIZE	M_AXI_DC	O	Master Burst size
M_AXI_DC_ARBURST	M_AXI_DC	O	Master Burst type
M_AXI_DC_ARLOCK	M_AXI_DC	O	Master Lock type
M_AXI_DC_ARCACHE	M_AXI_DC	O	Master Cache type
M_AXI_DC_ARPROT	M_AXI_DC	O	Master Protection type
M_AXI_DC_ARQOS	M_AXI_DC	O	Master Quality of Service
M_AXI_DC_ARVALID	M_AXI_DC	O	Master Read address valid
M_AXI_DC_ARREADY	M_AXI_DC	I	Slave Read address ready
M_AXI_DC_ARUSER	M_AXI_DC	O	Master Read address user signals
M_AXI_DC_RID	M_AXI_DC	I	Slave Read ID tag
M_AXI_DC_RDATA	M_AXI_DC	I	Slave Read data
M_AXI_DC_RRESP	M_AXI_DC	I	Slave Read response
M_AXI_DC_RLAST	M_AXI_DC	I	Slave Read last
M_AXI_DC_RVALID	M_AXI_DC	I	Slave Read valid
M_AXI_DC_RREADY	M_AXI_DC	O	Master Read ready
M_AXI_DC_RUSER	M_AXI_DC	I	Slave Read user signals
M_AXI_IC_AWID	M_AXI_IC	O	Master Write address ID
M_AXI_IC_AWADDR	M_AXI_IC	O	Master Write address
M_AXI_IC_AWLEN	M_AXI_IC	O	Master Burst length
M_AXI_IC_AWSIZE	M_AXI_IC	O	Master Burst size
M_AXI_IC_AWBURST	M_AXI_IC	O	Master Burst type
M_AXI_IC_AWLOCK	M_AXI_IC	O	Master Lock type
M_AXI_IC_AWCACHE	M_AXI_IC	O	Master Cache type
M_AXI_IC_AWPROT	M_AXI_IC	O	Master Protection type
M_AXI_IC_AWQOS	M_AXI_IC	O	Master Quality of Service
M_AXI_IC_AWVALID	M_AXI_IC	O	Master Write address valid
M_AXI_IC_AWREADY	M_AXI_IC	I	Slave Write address ready
M_AXI_IC_AWUSER	M_AXI_IC	O	Master Write address user signals

Table 2-1: **Summary of MicroBlaze Core I/O *(Continued)***

Signal	Interface	I/O	Description
M_AXI_IC_WDATA	M_AXI_IC	O	Master Write data
M_AXI_IC_WSTRB	M_AXI_IC	O	Master Write strobes
M_AXI_IC_WLAST	M_AXI_IC	O	Master Write last
M_AXI_IC_WVALID	M_AXI_IC	O	Master Write valid
M_AXI_IC_WREADY	M_AXI_IC	I	Slave Write ready
M_AXI_IC_WUSER	M_AXI_IC	O	Master Write user signals
M_AXI_IC_BID	M_AXI_IC	I	Slave Response ID
M_AXI_IC_BRESP	M_AXI_IC	I	Slave Write response
M_AXI_IC_BVALID	M_AXI_IC	I	Slave Write response valid
M_AXI_IC_BREADY	M_AXI_IC	O	Master Response ready
M_AXI_IC_BUSER	M_AXI_IC	I	Slave Write response user signals
M_AXI_IC_ARID	M_AXI_IC	O	Master Read address ID
M_AXI_IC_ARADDR	M_AXI_IC	O	Master Read address
M_AXI_IC_ARLEN	M_AXI_IC	O	Master Burst length
M_AXI_IC_ARSIZE	M_AXI_IC	O	Master Burst size
M_AXI_IC_ARBURST	M_AXI_IC	O	Master Burst type
M_AXI_IC_ARLOCK	M_AXI_IC	O	Master Lock type
M_AXI_IC_ARCACHE	M_AXI_IC	O	Master Cache type
M_AXI_IC_ARPROT	M_AXI_IC	O	Master Protection type
M_AXI_IC_ARQOS	M_AXI_IC	O	Master Quality of Service
M_AXI_IC_ARVALID	M_AXI_IC	O	Master Read address valid
M_AXI_IC_ARREADY	M_AXI_IC	I	Slave Read address ready
M_AXI_IC_ARUSER	M_AXI_IC	O	Master Read address user signals
M_AXI_IC_RID	M_AXI_IC	I	Slave Read ID tag
M_AXI_IC_RDATA	M_AXI_IC	I	Slave Read data
M_AXI_IC_RRESP	M_AXI_IC	I	Slave Read response
M_AXI_IC_RLAST	M_AXI_IC	I	Slave Read last
M_AXI_IC_RVALID	M_AXI_IC	I	Slave Read valid
M_AXI_IC_RREADY	M_AXI_IC	O	Master Read ready
M_AXI_IC_RUSER	M_AXI_IC	I	Slave Read user signals
DPLB_M_ABort	DPLB	O	Data Interface PLB abort bus request indicator

Table 2-1: **Summary of MicroBlaze Core I/O *(Continued)***

Signal	Interface	I/O	Description
DPLB_M_ABus	DPLB	O	Data Interface PLB address bus
DPLB_M_UABus	DPLB	O	Data Interface PLB upper address bus
DPLB_M_BE	DPLB	O	Data Interface PLB byte enables
DPLB_M_busLock	DPLB	O	Data Interface PLB bus lock
DPLB_M_lockErr	DPLB	O	Data Interface PLB lock error indicator
DPLB_M_MSize	DPLB	O	Data Interface PLB master data bus size
DPLB_M_priority	DPLB	O	Data Interface PLB bus request priority
DPLB_M_rdBurst	DPLB	O	Data Interface PLB burst read transfer indicator
DPLB_M_request	DPLB	O	Data Interface PLB bus request
DPLB_M_RNW	DPLB	O	Data Interface PLB read/not write
DPLB_M_size	DPLB	O	Data Interface PLB transfer size
DPLB_M_TAttribute	DPLB	O	Data Interface PLB Transfer Attribute bus
DPLB_M_type	DPLB	O	Data Interface PLB transfer type
DPLB_M_wrBurst	DPLB	O	Data Interface PLB burst write transfer indicator
DPLB_M_wrDBus	DPLB	O	Data Interface PLB write data bus
DPLB_MBusy	DPLB	I	Data Interface PLB slave busy indicator
DPLB_MRdErr	DPLB	I	Data Interface PLB slave read error indicator
DPLB_MWrErr	DPLB	I	Data Interface PLB slave write error indicator
DPLB_MIRQ	DPLB	I	Data Interface PLB slave interrupt indicator
DPLB_MWrBTerm	DPLB	I	Data Interface PLB terminate write burst indicator
DPLB_MWrDAck	DPLB	I	Data Interface PLB write data acknowledge
DPLB_MAddrAck	DPLB	I	Data Interface PLB address acknowledge
DPLB_MRdBTerm	DPLB	I	Data Interface PLB terminate read burst indicator
DPLB_MRdDAck	DPLB	I	Data Interface PLB read data acknowledge
DPLB_MRdDBus	DPLB	I	Data Interface PLB read data bus
DPLB_MRdWdAddr	DPLB	I	Data Interface PLB read word address
DPLB_MRearbitrate	DPLB	I	Data Interface PLB bus rearbitrate indicator
DPLB_MSSize	DPLB	I	Data Interface PLB slave data bus size
DPLB_MTimeout	DPLB	I	Data Interface PLB bus timeout

Table 2-1: **Summary of MicroBlaze Core I/O *(Continued)***

Signal	Interface	I/O	Description
IPLB_M_ABort	IPLB	O	Instruction Interface PLB abort bus request indicator
IPLB_M_ABus	IPLB	O	Instruction Interface PLB address bus
IPLB_M_UABus	IPLB	O	Instruction Interface PLB upper address bus
IPLB_M_BE	IPLB	O	Instruction Interface PLB byte enables
IPLB_M_busLock	IPLB	O	Instruction Interface PLB bus lock
IPLB_M_lockErr	IPLB	O	Instruction Interface PLB lock error indicator
IPLB_M_MSize	IPLB	O	Instruction Interface PLB master data bus size
IPLB_M_priority	IPLB	O	Instruction Interface PLB bus request priority
IPLB_M_rdBurst	IPLB	O	Instruction Interface PLB burst read transfer indicator
IPLB_M_request	IPLB	O	Instruction Interface PLB bus request
IPLB_M_RNW	IPLB	O	Instruction Interface PLB read/not write
IPLB_M_size	IPLB	O	Instruction Interface PLB transfer size
IPLB_M_TAttribute	IPLB	O	Instruction Interface PLB Transfer Attribute bus
IPLB_M_type	IPLB	O	Instruction Interface PLB transfer type
IPLB_M_wrBurst	IPLB	O	Instruction Interface PLB burst write transfer indicator
IPLB_M_wrDBus	IPLB	O	Instruction Interface PLB write data bus
IPLB_MBusy	IPLB	I	Instruction Interface PLB slave busy indicator
IPLB_MRdErr	IPLB	I	Instruction Interface PLB slave read error indicator
IPLB_MWrErr	IPLB	I	Instruction Interface PLB slave write error indicator
IPLB_MIRQ	IPLB	I	Instruction Interface PLB slave interrupt indicator
IPLB_MWrBTerm	IPLB	I	Instruction Interface PLB terminate write burst indicator
IPLB_MWrDAck	IPLB	I	Instruction Interface PLB write data acknowledge
IPLB_MAddrAck	IPLB	I	Instruction Interface PLB address acknowledge

Table 2-1: **Summary of MicroBlaze Core I/O *(Continued)***

Signal	Interface	I/O	Description
IPLB_MRdBTerm	IPLB	I	Instruction Interface PLB terminate read burst indicator
IPLB_MRdDAck	IPLB	I	Instruction Interface PLB read data acknowledge
IPLB_MRdDBus	IPLB	I	Instruction Interface PLB read data bus
IPLB_MRdWdAddr	IPLB	I	Instruction Interface PLB read word address
IPLB_MRearbitrate	IPLB	I	Instruction Interface PLB bus rearbitrate indicator
IPLB_MSSize	IPLB	I	Instruction Interface PLB slave data bus size
IPLB_MTimeout	IPLB	I	Instruction Interface PLB bus timeout
Data_Addr[0:31]	DLMB	O	Data interface LMB address bus
Byte_Enable[0:3]	DLMB	O	Data interface LMB byte enables
Data_Write[0:31]	DLMB	O	Data interface LMB write data bus
D_AS	DLMB	O	Data interface LMB address strobe
Read_Strobe	DLMB	O	Data interface LMB read strobe
Write_Strobe	DLMB	O	Data interface LMB write strobe
Data_Read[0:31]	DLMB	I	Data interface LMB read data bus
DReady	DLMB	I	Data interface LMB data ready
DWait	DLMB	I	Data interface LMB data wait
DCE	DLMB	I	Data interface LMB correctable error
DUE	DLMB	I	Data interface LMB uncorrectable error
Instr_Addr[0:31]	ILMB	O	Instruction interface LMB address bus
I_AS	ILMB	O	Instruction interface LMB address strobe
IFetch	ILMB	O	Instruction interface LMB instruction fetch
Instr[0:31]	ILMB	I	Instruction interface LMB read data bus
IReady	ILMB	I	Instruction interface LMB data ready
IWait	ILMB	I	Instruction interface LMB data wait
ICE	ILMB	I	Instruction interface LMB correctable error
IUE	ILMB	I	Instruction interface LMB uncorrectable error
Mn_AXIS_TLAST	M0_AXIS.. M15_AXIS	O	Master interface output AXI4 channels write last
Mn_AXIS_TDATA	M0_AXIS.. M15_AXIS	O	Master interface output AXI4 channels write data

Table 2-1: **Summary of MicroBlaze Core I/O *(Continued)***

Signal	Interface	I/O	Description
Mn_AXIS_TVALID	M0_AXIS.. M15_AXIS	O	Master interface output AXI4 channels write valid
Mn_AXIS_TREADY	M0_AXIS.. M15_AXIS	I	Master interface input AXI4 channels write ready
Sn_AXIS_TLAST	S0_AXIS.. S15_AXIS	I	Slave interface input AXI4 channels write last
Sn_AXIS_TDATA	S0_AXIS.. S15_AXIS	I	Slave interface input AXI4 channels write data
Sn_AXIS_TVALID	S0_AXIS.. S15_AXIS	I	Slave interface input AXI4 channels write valid
Sn_AXIS_TREADY	S0_AXIS.. S15_AXIS	O	Slave interface output AXI4 channels write ready
FSL0_M .. FSL15_M	MFSL or DWFSL	O	Master interface to output FSL channels MFSL is used for FSL bus connections, whereas DWFSL is used for direct connections with FSL slaves
FSL0_S .. FSL15_S	SFSL or DRFSL	I	Slave interface to input FSL channels SFSL is used for FSL bus connections, whereas DRFSL is used for direct connections with FSL masters
ICache_FSL_in...	IXCL_S	IO	Instruction side CacheLink FSL slave interface
ICache_FSL_out...	IXCL_M	IO	Instruction side CacheLink FSL master interface
DCache_FSL_in...	DXCL_S	IO	Data side CacheLink FSL slave interface
DCache_FSL_out...	DXCL_M	IO	Data side CacheLink FSL master interface
Interrupt	Core	I	Interrupt
Reset[1]	Core	I	Core reset, active high. Should be held for at least 1 Clk clock cycle.
MB_Reset[1]	Core	I	Core reset, active high. Should be held for at least 1 Clk clock cycle.
Clk	Core	I	Clock[2]
Ext_BRK	Core	I	Break signal from MDM
Ext_NM_BRK	Core	I	Non-maskable break signal from MDM
MB_Halted	Core	O	Pipeline is halted, either via the Debug Interface or by setting Dbg_Stop

Table 2-1: **Summary of MicroBlaze Core I/O *(Continued)***

Signal	Interface	I/O	Description
Dbg_Stop	Core	I	Unconditionally force pipeline to halt as soon as possible. Rising-edge detected pulse that should be held for at least 1 Clk clock cycle. The signal only has any effect when C_DEBUG_ENABLED is set to 1.
MB_Error	Core	O	Pipeline is halted due to a missed exception, when C_FAULT_TOLERANT is set to 1.
Dbg_...	Core	IO	Debug signals from MDM. See Table 2-9 for details.
Trace_...	Core	O	Trace signals for real time HW analysis. See Table 2-10 for details.

1. The Reset and MB_Reset signals are functionally equivalent. MB_Reset is intended for the AXI4 and PLB interfaces.
2. MicroBlaze is a synchronous design clocked with the Clk signal, except for hardware debug logic, which is clocked with the Dbg_Clk signal. If hardware debug logic is not used, there is no minimum frequency limit for Clk. However, if hardware debug logic is used, there are signals transferred between the two clock regions. In this case Clk must have a higher frequency than Dbg_Clk.

AXI4 Interface Description

Memory Mapped Interfaces

The MicroBlaze AXI4 memory mapped peripheral interfaces (M_AXI_DP, M_AXI_IP) are implemented as 32-bit masters. These interfaces only issue single addresses, and all transactions are completed in order.

- The instruction peripheral interface only performns single word read accesses, and is always set to use the AXI4-Lite subset.
- The data peripheral interface performs single word accesses, and is set to use the AXI4-Lite subset as default, but is set to use AXI4 when enabling exclusive access for LWX and SWX instructions. Halfword and byte writes are performed by setting the appropriate byte strobes.

The AXI4 memory mapped cache interfaces (M_AXI_DC, M_AXI_IC) are implemented as either AXI4 32-bit masters, 128-bit masters, or 256-bit masters, depending on cache line length and data width parameters.

- With a 32-bit master, the instruction cache interface performs 4 word or 8 word burst read accesses, depending on cache line length. With 128-bit or 256-bit masters, only single read accesses are performed. This interface can either issue up to 2 addresses or up to 8 addresses when stream cache is enabled.
- With a 32-bit master, the data cache interface performs single word accesses, as well as 4 word or 8 word burst accesses, depending on cache line length. Burst write accesses are not performed when using write-through cache. With 128-bit or 256-bit masters, only single accesses are performed. Word, halfword and byte writes are performed by setting the appropriate byte strobes. This interface can either issue up to 2 addresses when reading, or up to 32 addresses when writing. Exclusive accesses can be enabled for LWX and SWX instructions.

Please refer to the *AMBA® AXI Protocol Specification, Version 2.0, ARM IHI 0022C* document for details.

Stream Interfaces

The MicroBlaze AXI4-Stream interfaces (M0_AXIS..M15_AXIS, S0_AXIS..S15_AXIS) are implemented as 32-bit masters and slaves. Please refer to the *AMBA®4 AXI4-Stream Protocol Specification, Version 1.0, ARM IHI 0051A* document for further details.

The `Mn_AXIS_TLAST` and `Sn_AXIS_TLAST` signals directly correspond to the equivalent `FSLn_M_Control` and `FSLn_S_Control` signals, respectively.

Write Operation

A write to the stream interface is performed by MicroBlaze using one of the put or putd instructions. A write operation transfers the register contents to an output AXI4 interface. The transfer is completed in a single clock cycle for blocking mode writes (put and cput instructions) as long as the interface is not busy. If the interface is busy, the processor stalls until it becomes available. The non-blocking instructions (with prefix n), always complete in a single clock cycle even if the interface is busy. If the interface was busy, the write is inhibited and the carry bit is set in the MSR.

Read Operation

A read from the stream interface is performed by MicroBlaze using one of the get or getd instructions. A read operations transfers the contents of an input AXI4 interface to a general purpose register. The transfer is typically completed in 2 clock cycles for blocking mode reads as long as data is available. If data is not available, the processor stalls at this instruction until it becomes available. In the non-blocking mode (instructions with prefix n), the transfer is completed in one or two clock cycles irrespective of whether or not data was available. In case data was not available, the transfer of data does not take place and the carry bit is set in the MSR.

Processor Local Bus (PLB) Interface Description

The MicroBlaze PLB interfaces are implemented as byte-enable capable 32-bit masters. Please refer to the *IBM 128-Bit Processor Local Bus Architectural Specification (v4.6)* document for details.

Local Memory Bus (LMB) Interface Description

The LMB is a synchronous bus used primarily to access on-chip block RAM. It uses a minimum number of control signals and a simple protocol to ensure that local block RAM are accessed in a single clock cycle. LMB signals and definitions are shown in the following table. All LMB signals are active high.

LMB Signal Interface

Table 2-2: **LMB Bus Signals**

Signal	Data Interface	Instruction Interface	Type	Description
`Addr[0:31]`	Data_Addr[0:31]	Instr_Addr[0:31]	O	Address bus
`Byte_Enable[0:3]`	Byte_Enable[0:3]	*not used*	O	Byte enables
`Data_Write[0:31]`	Data_Write[0:31]	*not used*	O	Write data bus
`AS`	D_AS	I_AS	O	Address strobe
`Read_Strobe`	Read_Strobe	IFetch	O	Read in progress

Table 2-2: **LMB Bus Signals *(Continued)***

Signal	Data Interface	Instruction Interface	Type	Description
Write_Strobe	Write_Strobe	*not used*	O	Write in progress
Data_Read[0:31]	Data_Read[0:31]	Instr[0:31]	I	Read data bus
Ready	DReady	IReady	I	Ready for next transfer
Wait[1]	DWait	IWait	I	Wait until accepted transfer is ready
CE[1]	DCE	ICE	I	Correctable error
UE[1]	DUE	IUE	I	Uncorrectable error
Clk	Clk	Clk	I	Bus clock

1. Added in LMB for MicroBlaze v8.00

Addr[0:31]

The address bus is an output from the core and indicates the memory address that is being accessed by the current transfer. It is valid only when AS is high. In multicycle accesses (accesses requiring more than one clock cycle to complete), Addr[0:31] is valid only in the first clock cycle of the transfer.

Byte_Enable[0:3]

The byte enable signals are outputs from the core and indicate which byte lanes of the data bus contain valid data. Byte_Enable[0:3] is valid only when AS is high. In multicycle accesses (accesses requiring more than one clock cycle to complete), Byte_Enable[0:3] is valid only in the first clock cycle of the transfer. Valid values for Byte_Enable[0:3] are shown in the following table:

:

Table 2-3: **Valid Values for Byte_Enable[0:3]**

	Byte Lanes Used			
Byte_Enable[0:3]	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]

The write data bus is an output from the core and contains the data that is written to memory. It is valid only when AS is high. Only the byte lanes specified by `Byte_Enable[0:3]` contain valid data.

AS

The address strobe is an output from the core and indicates the start of a transfer and qualifies the address bus and the byte enables. It is high only in the first clock cycle of the transfer, after which it goes low and remains low until the start of the next transfer.

Read_Strobe

The read strobe is an output from the core and indicates that a read transfer is in progress. This signal goes high in the first clock cycle of the transfer, and may remain high until the clock cycle after Ready is sampled high. If a new read transfer is directly started in the next clock cycle, then `Read_Strobe` remains high.

Write_Strobe

The write strobe is an output from the core and indicates that a write transfer is in progress. This signal goes high in the first clock cycle of the transfer, and may remain high until the clock cycle after Ready is sampled high. If a new write transfer is directly started in the next clock cycle, then `Write_Strobe` remains high.

Data_Read[0:31]

The read data bus is an input to the core and contains data read from memory. `Data_Read`[0:31] is valid on the rising edge of the clock when Ready is high.

Ready

The `Ready` signal is an input to the core and indicates completion of the current transfer and that the next transfer can begin in the following clock cycle. It is sampled on the rising edge of the clock. For reads, this signal indicates the `Data_Read[0:31]` bus is valid, and for writes it indicates that the `Data_Write[0:31]` bus has been written to local memory.

Wait

The `Wait` signal is an input to the core and indicates that the current transfer has been accepted, but not yet completed. It is sampled on the rising edge of the clock.

CE

The `CE` signal is an input to the core and indicates that the current transfer had a correctable error. It is valid on the rising edge of the clock when Ready is high. For reads, this signal indicates that an error has been corrected on the `Data_Read[0:31]` bus, and for byte and halfword writes it indicates that the corresponding data word in local memory has been corrected before writing the new data.

UE

The `UE` signal is an input to the core and indicates that the current transfer had an uncorrectable error. It is valid on the rising edge of the clock when Ready is high. For reads, this signal indicates that the value of the `Data_Read[0:31]` bus is erroneous, and for byte and halfword writes it

indicates that the corresponding data word in local memory was erroneous before writing the new data.

Clk

All operations on the LMB are synchronous to the MicroBlaze core clock.

LMB Transactions

The following diagrams provide examples of LMB bus operations.

Generic Write Operations

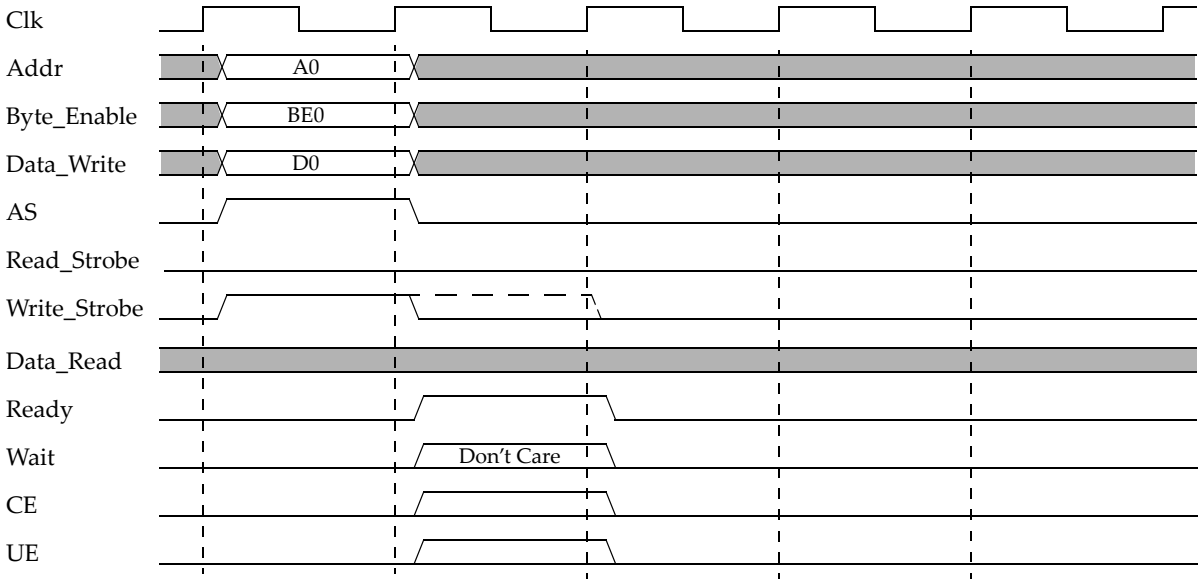


Figure 2-2: **LMB Generic Write Operation, 0 Wait States**

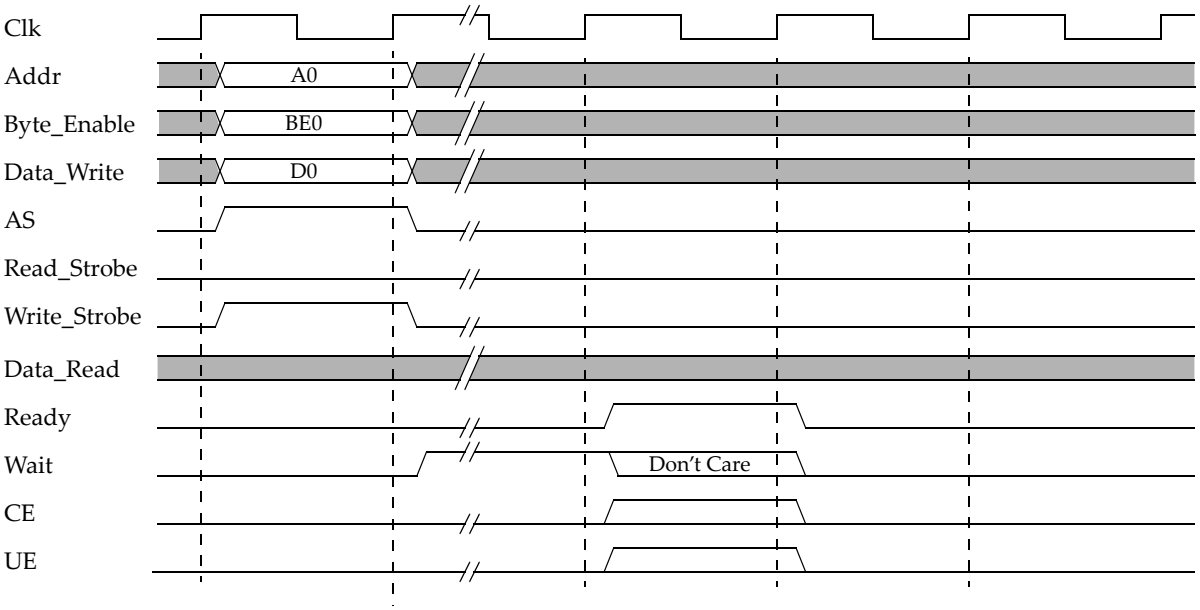


Figure 2-3: **LMB Generic Write Operation, N Wait States**

Generic Read Operations

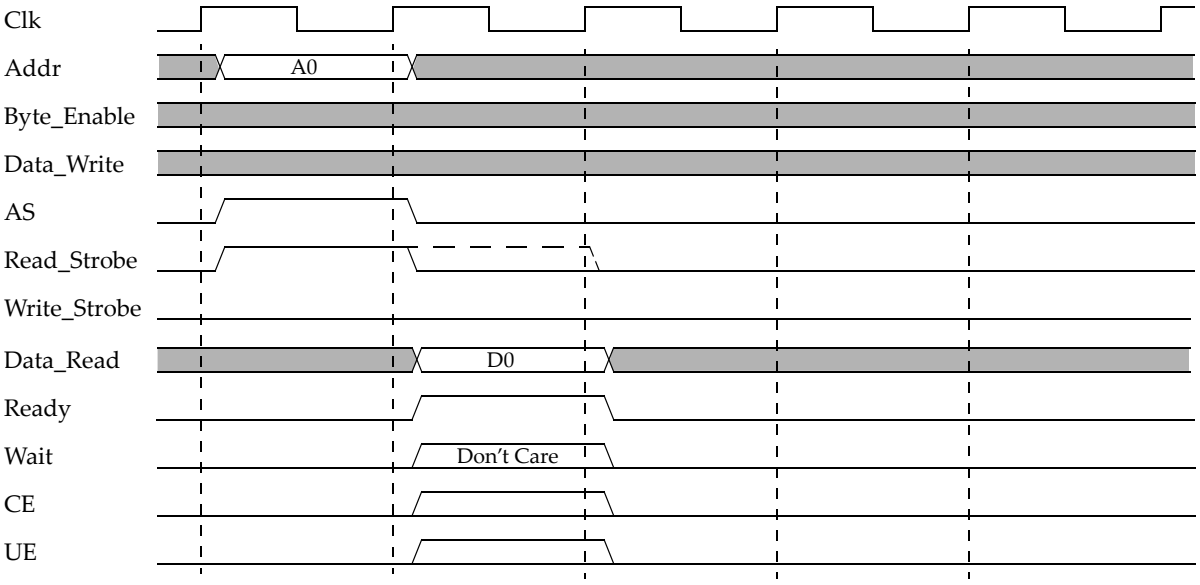


Figure 2-4: **LMB Generic Read Operation, 0 Wait States**

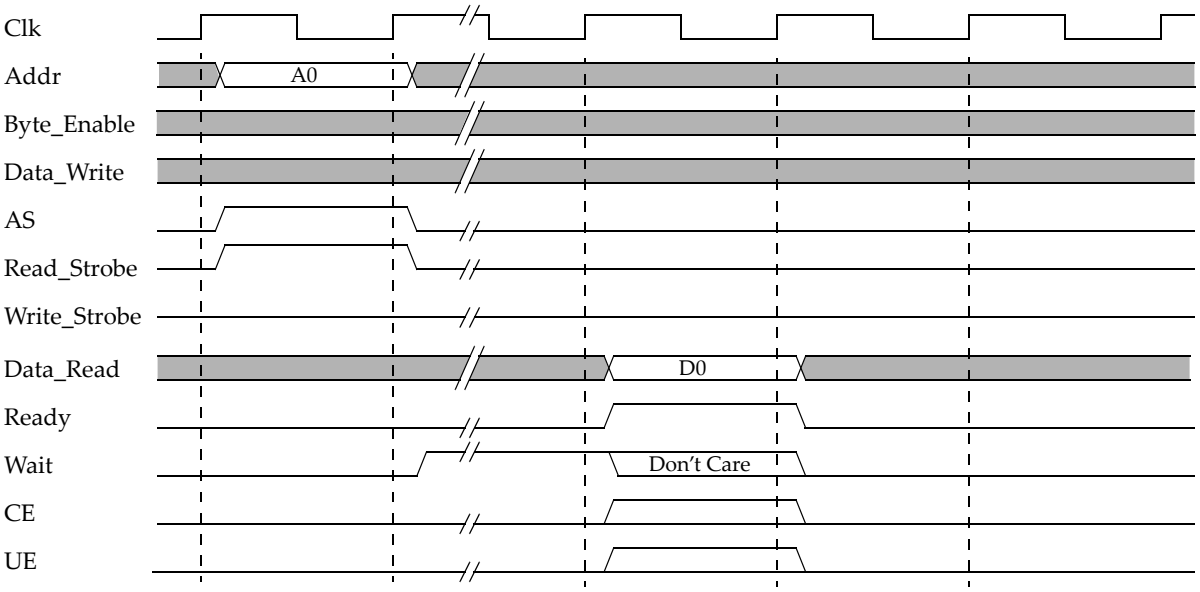


Figure 2-5: **LMB Generic Read Operation, N Wait States**

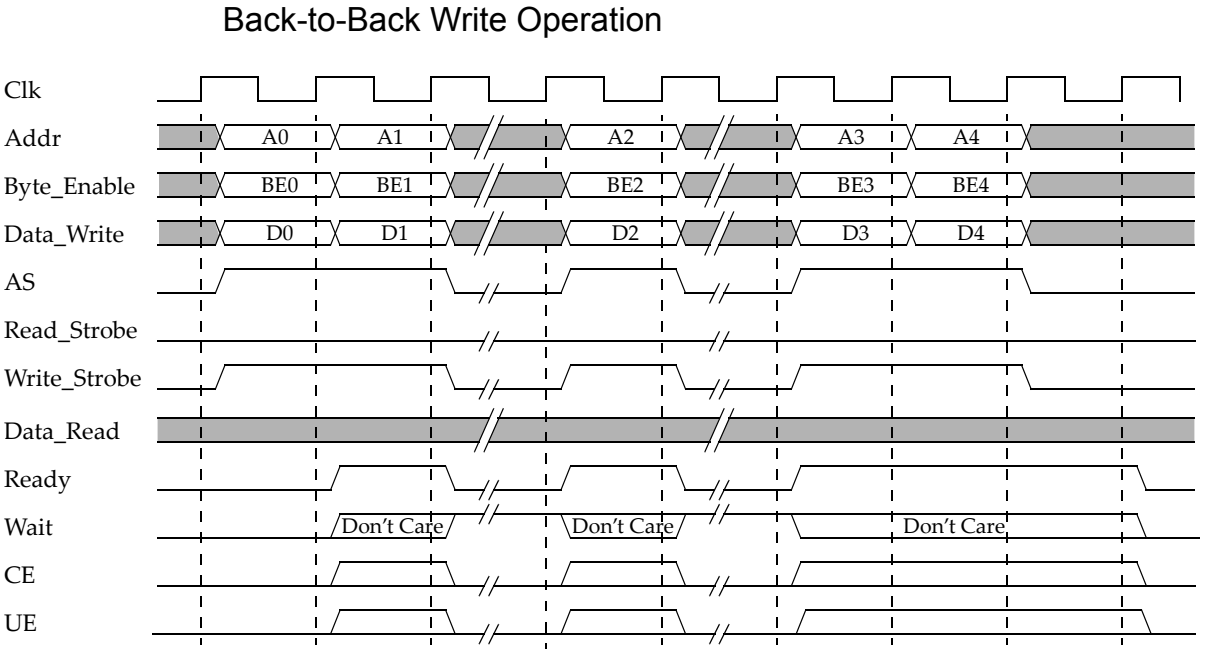


Figure 2-6: **LMB Back-to-Back Write Operation**

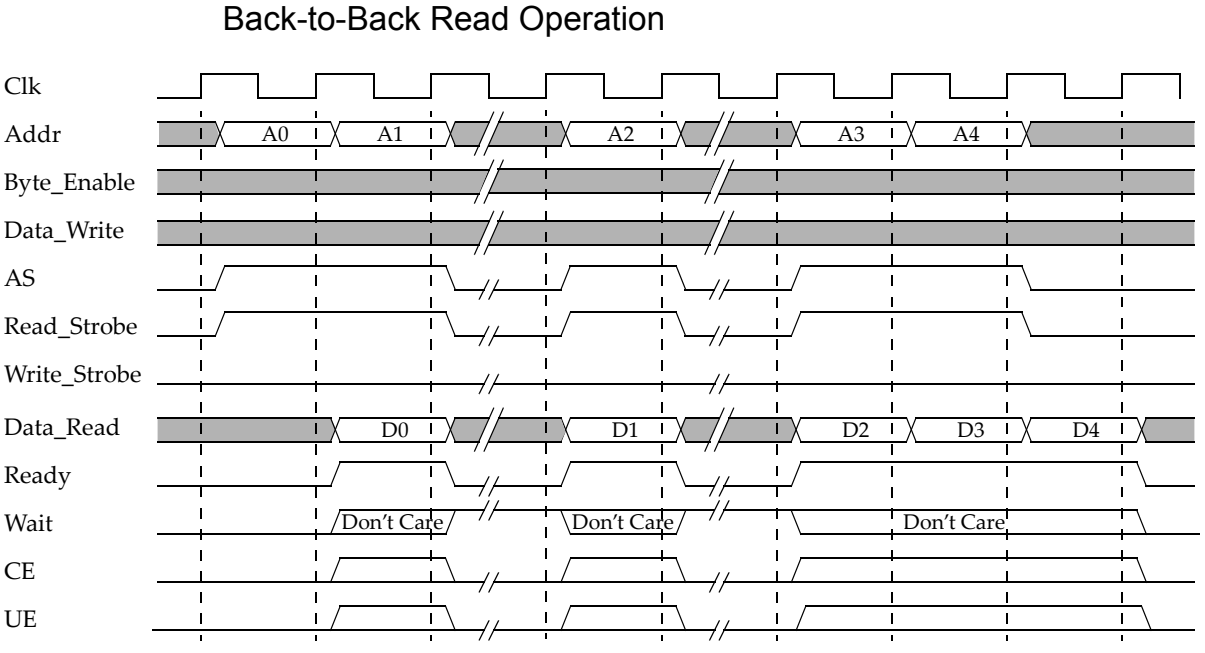


Figure 2-7: **LMB Back-to-Back Read Operation**

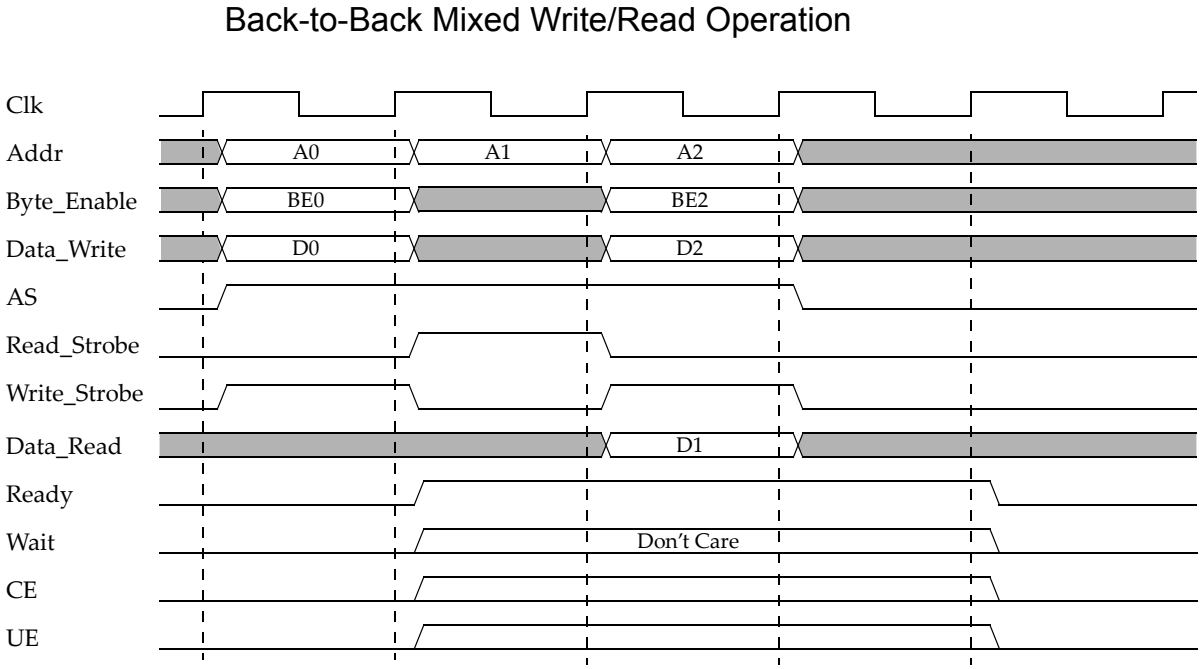


Figure 2-8: **Back-to-Back Mixed Write/Read Operation, 0 Wait States**

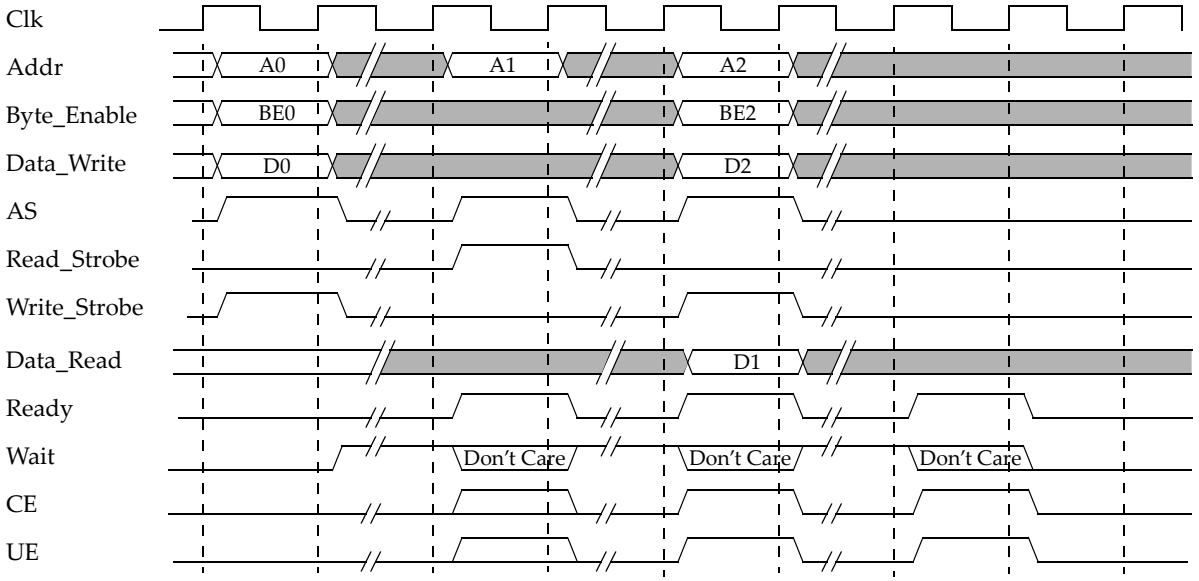


Figure 2-9: **Back-to-Back Mixed Write/Read Operation, N Wait States**

Read and Write Data Steering

The MicroBlaze data-side bus interface performs the read steering and write steering required to support the following transfers:

- byte, halfword, and word transfers to word devices
- byte and halfword transfers to halfword devices
- byte transfers to byte devices

MicroBlaze does not support transfers that are larger than the addressed device. These types of transfers require dynamic bus sizing and conversion cycles that are not supported by the MicroBlaze bus interface. Data steering for read cycles is shown in Table 2-4, and data steering for write cycles is shown in Table 2-5.

Table 2-4: **Read Data Steering (Load to Register rD)**

			Register rD Data			
Address [30:31]	**Byte_Enable [0:3]**	**Transfer Size**	**rD[0:7]**	**rD[8:15]**	**rD[16:23]**	**rD[24:31]**
11	0001	byte				Byte3
10	0010	byte				Byte2
01	0100	byte				Byte1
00	1000	byte				Byte0
10	0011	halfword			Byte2	Byte3
00	1100	halfword			Byte0	Byte1
00	1111	word	Byte0	Byte1	Byte2	Byte3

Table 2-5: **Write Data Steering (Store from Register rD)**

			Write Data Bus Bytes			
Address [30:31]	**Byte_Enable [0:3]**	**Transfer Size**	Byte0	Byte1	Byte2	Byte3
11	0001	byte				**rD[24:31]**
10	0010	byte			**rD[24:31]**	
01	0100	byte		**rD[24:31]**		
00	1000	byte	**rD[24:31]**			
10	0011	halfword			**rD[16:23]**	**rD[24:31]**
00	1100	halfword	**rD[16:23]**	**rD[24:31]**		
00	1111	word	**rD[0:7]**	**rD[8:15]**	**rD[16:23]**	**rD[24:31]**

Note: Other masters may have more restrictive requirements for byte lane placement than those allowed by MicroBlaze. Slave devices are typically attached “left-justified” with byte devices attached to the most-significant byte lane, and halfword devices attached to the most significant halfword lane. The MicroBlaze steering logic fully supports this attachment method.

Fast Simplex Link (FSL) Interface Description

The Fast Simplex Link bus provides a point-to-point communication channel between an output FIFO and an input FIFO. For more information on the generic FSL protocol, see the *Fast Simplex Link (FSL) Bus* data-sheet, DS449, in the Xilinx EDK IP Documentation.

Master FSL Signal Interface

MicroBlaze may contain up to 16 master FSL interfaces. The master signals are depicted in Table 2-6.

Table 2-6: **Master FSL Signals**

Signal Name	Description	VHDL Type	Direction
FSLn_M_Clk	Clock	std_logic	input
FSLn_M_Write	Write enable signal indicating that data is being written to the output FSL	std_logic	output
FSLn_M_Data	Data value written to the output FSL	std_logic_vector	output
FSLn_M_Control	Control bit value written to the output FSL	std_logic	output
FSLn_M_Full	Full Bit indicating output FSL FIFO is full when set	std_logic	input

Slave FSL Signal Interface

MicroBlaze may contain up to 16 slave FSL interfaces. The slave FSL interface signals are depicted in Table 2-7.

Table 2-7: **Slave FSL Signals**

Signal Name	Description	VHDL Type	Direction
FSLn_S_Clk	Clock	std_logic	input
FSLn_S_Read	Read acknowledge signal indicating that data has been read from the input FSL	std_logic	output
FSLn_S_Data	Data value currently available at the top of the input FSL	std_logic_vector	input
FSLn_S_Control	Control Bit value currently available at the top of the input FSL	std_logic	input
FSLn_S_Exists	Flag indicating that data exists in the input FSL	std_logic	input

FSL Transactions

FSL BUS Write Operation

A write to the FSL bus is performed by MicroBlaze using one of the put or putd instructions. A write operation transfers the register contents to an output FSL bus. The transfer is completed in a single clock cycle for blocking mode writes to the FSL (put and cput instructions) as long as the FSL FIFO does not become full. If the FSL FIFO is full, the processor stalls until the FSL full flag is lowered. The non-blocking instructions (with prefix n), always complete in a single clock cycle even if the FSL was full. If the FSL was full, the write is inhibited and the carry bit is set in the MSR.

FSL BUS Read Operation

A read from the FSL bus is performed by MicroBlaze using one of the get or getd instructions. A read operations transfers the contents of an input FSL to a general purpose register. The transfer is typically completed in 2 clock cycles for blocking mode reads from the FSL as long as data exists in the FSL FIFO. If the FSL FIFO is empty, the processor stalls at this instruction until the FSL exists flag is set. In the non-blocking mode (instructions with prefix n), the transfer is completed in one or two clock cycles irrespective of whether or not the FSL was empty. In the case the FSL was empty, the transfer of data does not take place and the carry bit is set in the MSR.

Direct FSL Connections

A direct FSL connection can be used to avoid the need for the FSL bus. This can be useful in case no buffering is needed between the two connected IP cores, since the FSL bus FIFO is not included with a direct connection. No buffering reduces the communication latency and required implementation resources.

Each of the MicroBlaze FSL interfaces can either use a direct FSL connection or an FSL bus.

A MicroBlaze DWFSL interface is the initiator on a direct FSL connection, which can only be connected to a DWFSL target. The DWFSL initiator and target have exactly the same signal names, identical to the MFSL signals, depicted in Table 2-6. MicroBlaze uses the DWFSL interface to write data to the target with one of the put or putd instructions.

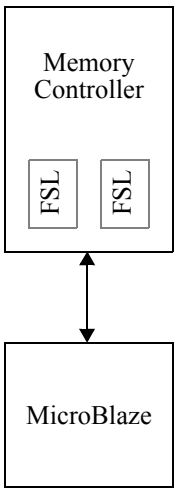
A MicroBlaze DRFSL interface is the target on a direct FSL connection, which can only be connected to a DRFSL initiator. The DRFSL initiator and target have exactly the same signal names, identical to the SFSL signals, depicted in Table 2-7. MicroBlaze uses the DRFSL interface to read data from the initiator with one of the get or getd instructions.

The Xilinx CacheLink (XCL) interface is implemented with direct FSL connections.

Xilinx CacheLink (XCL) Interface Description

Xilinx CacheLink (XCL) is a high performance solution for external memory accesses. The MicroBlaze CacheLink interface is designed to connect directly to a memory controller with integrated FSL buffers, for example, the MPMC. This method has the lowest latency and minimal number of instantiations (see Figure 2-10).

Schematic



Example MHS code

```
BEGIN microblaze
  ...
  BUS_INTERFACE IXCL = myIXCL
  ...
END
BEGIN mpmc
  ...
  BUS_INTERFACE XCL0 = myIXCL
  ...
END
```

Figure 2-10: **CacheLink Connection with Integrated FSL Buffers (Only Instruction Cache Used in this Example)**

The interface is only available on MicroBlaze when caches are enabled. It is legal to use a CacheLink cache on the instruction side or the data side without caching the other.

How memory locations are accessed depend on the parameter C_ICACHE_ALWAYS_USED for the instruction cache and the parameter C_DCACHE_ALWAYS_USED for the data cache. If the parameter is 1, the cached memory range is always accessed via the CacheLink. If the parameter is 0, the cached memory range is accessed over AXI4 or PLB whenever the caches are software disabled (that is, MSR[DCE]=0 or MSR[ICE]=0).

Memory locations outside the cacheable range are accessed over AXI, PLB or LMB.

The CacheLink cache controllers handle 4 or 8-word cache lines, either using critical word first or linear fetch depending on the selected protocol. At the same time the separation from the AXI4 or PLB bus reduces contention for non-cached memory accesses.

CacheLink Signal Interface

The CacheLink signals on MicroBlaze are listed in Table 2-8.

Table 2-8: **MicroBlaze Cache Link Signals**

Signal Name	Description	VHDL Type	Direction
ICACHE_FSL_IN_Clk	Clock output to I-side return read data FSL	std_logic	output
ICACHE_FSL_IN_Read	Read signal to I-side return read data FSL.	std_logic	output
ICACHE_FSL_IN_Data	Read data from I-side return read data FSL	std_logic_ vector (0 to 31)	input
ICACHE_FSL_IN_Control	FSL control-bit from I-side return read data FSL. Reserved for future use	std_logic	input
ICACHE_FSL_IN_Exists	More read data exists in I-side return FSL	std_logic	input
ICACHE_FSL_OUT_Clk	Clock output to I-side read access FSL	std_logic	output
ICACHE_FSL_OUT_Write	Write new cache miss access request to I-side read access FSL	std_logic	output
ICACHE_FSL_OUT_Data	Cache miss access (=address) to I-side read access FSL	std_logic_ vector (0 to 31)	output
ICACHE_FSL_OUT_Control	FSL control-bit to I-side read access FSL. Reserved for future use	std_logic	output
ICACHE_FSL_OUT_Full	FSL access buffer for I-side read accesses is full	std_logic	input
DCACHE_FSL_IN_Clk	Clock output to D-side return read data FSL	std_logic	output
DCACHE_FSL_IN_Read	Read signal to D-side return read data FSL	std_logic	output
DCACHE_FSL_IN_Data	Read data from D-side return read data FSL	std_logic_ vector (0 to 31)	input
DCACHE_FSL_IN_Control	FSL control bit from D-side return read data FSL	std_logic	input
DCACHE_FSL_IN_Exists	More read data exists in D-side return FSL	std_logic	input
DCACHE_FSL_OUT_Clk	Clock output to D-side read access FSL	std_logic	output
DCACHE_FSL_OUT_Write	Write new cache miss access request to D-side read access FSL	std_logic	output

Table 2-8: **MicroBlaze Cache Link Signals**

Signal Name	Description	VHDL Type	Direction
DCACHE_FSL_OUT_Data	Cache miss access (read address or write address + write data + byte write enable + burst write encoding) to D-side read access FSL	std_logic_ vector (0 to 31)	output
DCACHE_FSL_OUT_Control	FSL control-bit to D-side read access FSL. Used with address bits [30 to 31] for read/write, byte enable and burst write encoding.	std_logic	output
DCACHE_FSL_OUT_Full	FSL access buffer for D-side read accesses is full	std_logic	input

CacheLink Transactions

All individual CacheLink accesses follow the FSL FIFO based transaction protocol:

- Access information is encoded over the FSL data and control signals (e.g. DCACHE_FSL_OUT_Data, DCACHE_FSL_OUT_Control, ICACHE_FSL_IN_Data, and ICACHE_FSL_IN_Control)
- Information is sent (stored) by raising the write enable signal (e.g. DCACHE_FSL_OUT_Write)
- The sender is only allowed to write if the full signal from the receiver is inactive (e.g. DCACHE_FSL_OUT_Full = 0). The full signal is not used by the instruction cache controller.
- The use of ICACHE_FSL_IN_Read and DCACHE_FSL_IN_Read depends on the selected interface protocol:
  - With the IXCL and DXCL protocol, information is received (loaded) by raising the read signal. The signal is low, except when the sender signals that new data exists.
  - With the IXCL2 and DXCL2 protocol, lowering the read signal indicates that the receiver is not able to accept new data. New data is only read when the read signal is high, and the sender signals that data exists.
- The receiver is only allowed to read as long as the sender signals that new data exists (e.g. ICACHE_FSL_IN_Exists = 1)

For details on the generic FSL protocol, please see the *Fast Simplex Link (FSL) Bus* data-sheet, DS449, in the Xilinx EDK IP Documentation.

The CacheLink solution uses one incoming (slave) and one outgoing (master) FSL per cache controller. The outgoing FSL is used to send access requests, while the incoming FSL is used for receiving the requested cache lines. CacheLink also uses a specific encoding of the transaction information over the FSL data and control signals.

The cache lines used for reads in the CacheLink protocol are 4 or 8 words long. Each cache line is either fetched with the critical word first, or in linear order, depending on the selected interface protocol.

- Critical word first is used by the IXCL and DXCL protocol, selected when C_ICACHE_INTERFACE = 0 and C_DCACHE_INTERFACE = 0, respectively. Each cache

line is expected to start with the critical word first (that is, if an access to address 0x348 is a miss with a 4 word cache line, then the returned cache line should have the following address sequence: 0x348, 0x34c, 0x340, 0x344). The cache controller forwards the first word to the execution unit as well as stores it in the cache memory. This allows execution to resume as soon as the first word is back. The cache controller then follows through by filling up the cache line with the remaining 3 or 7 words as they are received.

- Linear fetch is used by the IXCL2 and DXCL2 protocol, selected when C_ICACHE_INTERFACE = 1 and C_DCACHE_INTERFACE = 1, respectively. The address output on the CacheLink is then aligned to the cache line size (that is, if an access to address 0x348 is a miss with a 4 word cache line, then the address output on the CacheLink is 0x340). The cache controller stores data in the cache memory, and forwards the requested word to the execution unit when it is available.

When the parameter C_DCACHE_USE_WRITEBACK is set to 1, write operations can store an entire cache line using burst write, as well as single-words. Each cache line is always stored in linear order, and the address output on the CacheLink is aligned to the cache line size. When the parameter C_DCACHE_USE_WRITEBACK is cleared to 0, all write operations on the CacheLink are single-word. C_DCACHE_INTERFACE must be set to 1 when write-back is used, since burst write is only available with the DXCL2 protocol.

Instruction Cache Read Miss

On a read miss the cache controller performs the following sequence:

1. Write the word aligned(1) or cache line aligned missed address to ICACHE_FSL_OUT_Data, with the control bit set low (ICACHE_FSL_OUT_Control = 0) to indicate a read access
2. Wait until ICACHE_FSL_IN_Exists goes high to indicate that data is available

Note: There must be at least one clock cycle before ICACHE_FSL_IN_Exists goes high (that is, at least one wait state must be used).

With the IXCL protocol (critical word first):

3. Store the word from ICACHE_FSL_IN_Data to the cache
4. Forward the critical word to the execution unit in order to resume execution
5. Repeat 3 and 4 for the subsequent 3 or 7 words in the cache line

With the IXCL2 protocol (linear fetch):

3. Store words from ICACHE_FSL_IN_Data to the cache
4. Forward the relevant word to the execution unit in order to resume execution
5. Store remaining words from ICACHE_FSL_IN_Data to the cache

Data Cache Read Miss

On a read miss the cache controller will perform the following sequence:

1. If DCACHE_FSL_OUT_Full = 1 then stall until it goes low
2. Write the word aligned[1] or cache line aligned missed address to DCACHE_FSL_OUT_Data, with the control bit set low (DCACHE_FSL_OUT_Control = 0) to indicate a read access
3. Wait until DCACHE_FSL_IN_Exists goes high to indicate that data is available

1. Byte and halfword read misses are naturally expected to return complete words, the cache controller then provides the execution unit with the correct bytes.

Note: There must be at least one clock cycle before `DCACHE_FSL_IN_Exists` goes high (that is, at least one wait state must be used).

With the DXCL protocol (critical word first):

4. Store the word from `DCACHE_FSL_IN_Data` to the cache
5. Forward the critical word to the execution unit in order to resume execution
6. Repeat 4 and 5 for the subsequent 3 or 7 words in the cache line

With the DXCL2 protocol (linear fetch):

4. Store words from `DCACHE_FSL_IN_Data` to the cache
5. Forward the requested word to the execution unit in order to resume execution
6. Store remaining words from `DCACHE_FSL_IN_Data` to the cache

Data Cache Write

When `C_DCACHE_INTERFACE` is set to 1, the CacheLink can either do burst write or single-word write. A burst write is used when `C_DCACHE_USE_WRITEBACK` is set to 1 and an entire cache line is valid.

Note that writes to the data cache always are write-through when `C_DCACHE_USE_WRITEBACK` is cleared to 0, and thus there is a write over the CacheLink regardless of whether there was a hit or miss in the cache.

With the DXCL2 protocol, on a burst cache line write, the cache controller performs the following sequence:

1. If `DCACHE_FSL_OUT_Full` = 1 then stall until it goes low
2. Write the cache aligned address to `DCACHE_FSL_OUT_Data`, with the control bit set high (`DCACHE_FSL_OUT_Control` = 1) to indicate a write access. The two least-significant bits (30:31) of the address are used to encode burst access: 0b10=burst. To separate a burst access from a single byte-write, the control bit for the first data word in step 4 is low for a burst access (`DCACHE_FSL_OUT_Control` = 0).
3. If `DCACHE_FSL_OUT_Full` = 1 then stall until it goes low
4. Write the data to be stored to `DCACHE_FSL_OUT_Data`. The control bit is low (`DCACHE_FSL_OUT_Control` = 0) for a burst access.
5. Repeat 3 and 4 for the subsequent words in the cache line.

With either the DXCL or DXCL2 protocol, on a single-word write, the cache controller performs the following sequence:

1. If `DCACHE_FSL_OUT_Full` = 1 then stall until it goes low
2. Write the missed address to `DCACHE_FSL_OUT_Data`, with the control bit set high (`DCACHE_FSL_OUT_Control` = 1) to indicate a write access. The two least-significant bits (30:31) of the address are used to encode byte and half-word enables: 0b00=byte0, 0b01=byte1 or halfword0, 0x10=byte2, and 0x11=byte3 or halfword1. The selection of half-word or byte access is based on the control bit for the data word in step 4.
3. If `DCACHE_FSL_OUT_Full` = 1 then stall until it goes low
4. Write the data to be stored to `DCACHE_FSL_OUT_Data`. For byte and halfword accesses the data is mirrored onto byte-lanes. Mirroring outputs the byte or halfword to be written on all four byte-lanes or on both halfword-lanes, respectively. The control bit should be low (`DCACHE_FSL_OUT_Control` = 0) for a word or halfword access, and high for a byte access to separate it from a burst access. Word or halfword accesses can be distinguished by the least significant bit of the address (0=word and 1=halfword).

Debug Interface Description

The debug interface on MicroBlaze is designed to work with the Xilinx Microprocessor Debug Module (MDM) IP core. The MDM is controlled by the Xilinx Microprocessor Debugger (XMD) through the JTAG port of the FPGA. The MDM can control multiple MicroBlaze processors at the same time. The debug signals are grouped in the DEBUG bus. The debug signals on MicroBlaze are listed in Table 2-9.

Table 2-9: **MicroBlaze Debug Signals**

Signal Name	Description	VHDL Type	Direction
Dbg_Clk	JTAG clock from MDM	std_logic	input
Dbg_TDI	JTAG TDI from MDM	std_logic	input
Dbg_TDO	JTAG TDO to MDM	std_logic	output
Dbg_Reg_En	Debug register enable from MDM	std_logic	input
Dbg_Shift[1]	JTAG BSCAN shift signal from MDM	std_logic	input
Dbg_Capture	JTAG BSCAN capture signal from MDM	std_logic	input
Dbg_Update	JTAG BSCAN update signal from MDM	std_logic	input
Debug_Rst[1]	Reset signal from MDM, active high. Should be held for at least 1 Clk clock cycle.	std_logic	input

1. Updated for MicroBlaze v7.00: Dbg_Shift added and Debug_Rst included in DEBUG bus

Trace Interface Description

The MicroBlaze core exports a number of internal signals for trace purposes. This signal interface is not standardized and new revisions of the processor may not be backward compatible for signal selection or functionality. It is recommended that you not design custom logic for these signals, but rather to use them via Xilinx provided analysis IP. The trace signals are grouped in the TRACE bus. The current set of trace signals were last updated for MicroBlaze v7.00 and are listed in Table 2-10. The Trace exception types are listed in Table 2-11. All unused Trace exception types are reserved.

Table 2-10: **MicroBlaze Trace Signals**

Signal Name	Description	VHDL Type	Direction
Trace_Valid_Instr	Valid instruction on trace port.	std_logic	output
Trace_Instruction [1]	Instruction code	std_logic_vector (0 to 31)	output
Trace_PC [1]	Program counter	std_logic_vector (0 to 31)	output
Trace_Reg_Write [1]	Instruction writes to the register file	std_logic	output
Trace_Reg_Addr [1]	Destination register address	std_logic_vector (0 to 4)	output
Trace_MSR_Reg[1]	Machine status register	std_logic_vector (0 to 14)[2]	output
Trace_PID_Reg[1,2]	Process identifier register	std_logic_vector (0 to 7)	output

Table 2-10: **MicroBlaze Trace Signals**

Signal Name	Description	VHDL Type	Direction
Trace_New_Reg_Value[1]	Destination register update value	std_logic_vector (0 to 31)	output
Trace_Exception_Taken[1]	Instruction result in taken exception	std_logic	output
Trace_Exception_Kind[1,3]	Exception type. The description for the exception type is documented below.	std_logic_vector (0 to 4)[2]	output
Trace_Jump_Taken[1]	Branch instruction evaluated true, i.e taken	std_logic	output
Trace_Jump_Hit[1,4,5]	Branch Target Cache hit	std_logic	output
Trace_Delay_Slot[1]	Instruction is in delay slot of a taken branch	std_logic	output
Trace_Data_Access[1]	Valid D-side memory access	std_logic	output
Trace_Data_Address[1]	Address for D-side memory access	std_logic_vector (0 to 31)	output
Trace_Data_Write_Value[1]	Value for D-side memory write access	std_logic_vector (0 to 31)	output
Trace_Data_Byte_Enable[1]	Byte enables for D-side memory access	std_logic_vector (0 to 3)	output
Trace_Data_Read[1]	D-side memory access is a read	std_logic	output
Trace_Data_Write[1]	D-side memory access is a write	std_logic	output
Trace_DCache_Req	Data memory address is within D-Cache range	std_logic	output
Trace_DCache_Hit	Data memory address is present in D-Cache	std_logic	output
Trace_ICache_Req	Instruction memory address is in I-Cache range	std_logic	output
Trace_ICache_Hit	Instruction memory address is present in I-Cache	std_logic	output
Trace_OF_PipeRun	Pipeline advance for Decode stage	std_logic	output
Trace_EX_PipeRun[5]	Pipeline advance for Execution stage	std_logic	output
Trace_MEM_PipeRun[5]	Pipeline advance for Memory stage	std_logic	output
Trace_MB_Halted[2]	Pipeline is halted by debug	std_logic	output

1. Valid only when Trace_Valid_Instr = 1
2. Updated for MicroBlaze v7.00: 4 bits added to Trace_MSR_Reg, Trace_PID_Reg added, Trace_MB_Halted added, and 1 bit added to Trace_Exception Kind
3. Valid only when Trace_Exception_Taken = 1
4. Updated for MicroBlaze v7.30: Trace_Jump_Hit added
5. Not used with area optimization feature

Table 2-11: **Type of Trace Exception**

Trace_Exception_Kind [0:4]	Description
00000	Stream exception[1]
00001	Unaligned exception
00010	Illegal Opcode exception
00011	Instruction Bus exception
00100	Data Bus exception
00101	Divide exception
00110	FPU exception
00111	Privileged instruction exception[1]
01010	Interrupt
01011	External non maskable break
01100	External maskable break
10000	Data storage exception[1]
10001	Instruction storage exception[1]
10010	Data TLB miss exception[1]
10011	Instruction TLB miss exception[1]

1. Added for MicroBlaze v7.00

MicroBlaze Core Configurability

The MicroBlaze core has been developed to support a high degree of user configurability. This allows tailoring of the processor to meet specific cost/performance requirements.

Configuration is done via parameters that typically enable, size, or select certain processor features. For example, the instruction cache is enabled by setting the C_USE_ICACHE parameter. The size of the instruction cache, and the cacheable memory range, are all configurable using: C_CACHE_BYTE_SIZE, C_ICACHE_BASEADDR, and C_ICACHE_HIGHADDR respectively.

Parameters valid for MicroBlaze v8.00 are listed in Table 2-12. Not all of these are recognized by older versions of MicroBlaze; however, the configurability is fully backward compatibility.

Note: Shaded rows indicate that the parameter has a fixed value and *cannot* be modified.

Table 2-12: **MPD Parameters**

Parameter Name	Feature/Description	Allowable Values	Default Value	EDK Tool Assigned	VHDL Type
C_FAMILY	Target Family	aspartan3 aspartan3a aspartan3adsp aspartan3e aspartan6 kintex7 qspartan6 qspartan6l spartan3 spartan3a spartan3adsp spartan3an spartan3e spartan6 spartan6l qrvirtex4 qrvirtex5 qvirtex4 qvirtex6 virtex4 virtex5 virtex6 virtex6l virtex7	virtex5	yes	string
C_DATA_SIZE	Data Size	32	32	NA	integer
C_DYNAMIC_BUS_SIZING	Legacy	1	1	NA	integer
C_SCO	Xilinx internal	0	0	NA	integer
C_AREA_OPTIMIZED	Select implementation to optimize area with lower instruction throughput	0, 1	0		integer

Table 2-12: **MPD Parameters *(Continued)***

Parameter Name	Feature/Description	Allowable Values	Default Value	EDK Tool Assigned	VHDL Type
C_OPTIMIZATION	Reserved for future use	0	0	NA	integer
C_INTERCONNECT	Select interconnect 1 = PLBv46 2 = AXI4	1, 2	1		integer
C_ENDIANNESS	Select endianness 0 = Big endian 1= Little endian	0, 1	0	yes	integer
C_FAULT_TOLERANT	Implement fault tolerance	0, 1	0		integer
C_PVR	Processor version register mode selection	0, 1, 2	0		integer
C_PVR_USER1	Processor version register USER1 constant	0x00-0xff	0x00		std_logic_vector (0 to 7)
C_PVR_USER2	Processor version register USER2 constant	0x00000000-0xffffffff	0x00000000		std_logic_vector (0 to 31)
C_RESET_MSR	Reset value for MSR register	0x00, 0x20, 0x80, 0xa0	0x00		std_logic_vector
C_INSTANCE	Instance Name	Any instance name	microblaze	yes	string
C_D_PLB	Data side PLB interface	0, 1	0	yes	integer
C_D_AXI	Data side AXI interface	0, 1	0	yes	integer
C_D_LMB	Data side LMB interface	0, 1	1	yes	integer
C_I_PLB	Instruction side PLB interface	0, 1	0	yes	integer
C_I_AXI	Instruction side AXI interface	0, 1	0	yes	integer
C_I_LMB	Instruction side LMB interface	0, 1	1	yes	integer
C_USE_BARREL	Include barrel shifter	0, 1	0		integer
C_USE_DIV	Include hardware divider	0, 1	0		integer
C_USE_HW_MUL	Include hardware multiplier	0, 1, 2	1		integer

Table 2-12: **MPD Parameters *(Continued)***

Parameter Name	Feature/Description	Allowable Values	Default Value	EDK Tool Assigned	VHDL Type
C_USE_FPU	Include hardware floating point unit	0, 1, 2	0		integer
C_USE_MSR_INSTR	Enable use of instructions: MSRSET and MSRCLR	0, 1	1		integer
C_USE_PCMP_INSTR	Enable use of instructions: CLZ, PCMPBF, PCMPEQ, and PCMPNE	0, 1	1		integer
C_UNALIGNED_EXCEPTIONS	Enable exception handling for unaligned data accesses	0, 1	0		integer
C_ILL_OPCODE_EXCEPTION	Enable exception handling for illegal op-code	0, 1	0		integer
C_IPLB_BUS_EXCEPTION	Enable exception handling for IPLB bus error	0, 1	0		integer
C_DPLB_BUS_EXCEPTION	Enable exception handling for DPLB bus error	0, 1	0		integer
C_M_AXI_I_BUS_EXCEPTION	Enable exception handling for M_AXI_I bus error	0, 1	0		integer
C_M_AXI_D_BUS_EXCEPTION	Enable exception handling for M_AXI_D bus error	0, 1	0		integer
C_DIV_ZERO_EXCEPTION	Enable exception handling for division by zero or division overflow	0, 1	0		integer
C_FPU_EXCEPTION	Enable exception handling for hardware floating point unit exceptions	0, 1	0		integer
C_OPCODE_0x0_ILLEGAL	Detect opcode 0x0 as an illegal instruction	0,1	0		integer
C_FSL_EXCEPTION	Enable exception handling for Stream Links	0,1	0		integer
C_ECC_USE_CE_EXCEPTION	Generate Bus Error Exceptions for correctable errors	0,1	0		integer
C_USE_STACK_PROTECTION	Generate exception for stack overflow or stack underflow	0,1	0		integer

Table 2-12: **MPD Parameters *(Continued)***

Parameter Name	Feature/Description	Allowable Values	Default Value	EDK Tool Assigned	VHDL Type
C_DEBUG_ENABLED	MDM Debug interface	0,1	0		integer
C_NUMBER_OF_PC_BRK	Number of hardware breakpoints	0-8	1		integer
C_NUMBER_OF_RD_ADDR_BRK	Number of read address watchpoints	0-4	0		integer
C_NUMBER_OF_WR_ADDR_BRK	Number of write address watchpoints	0-4	0		integer
C_INTERRUPT_IS_EDGE	Level/Edge Interrupt	0, 1	0	yes	integer
C_EDGE_IS_POSITIVE	Negative/Positive Edge Interrupt	0, 1	1	yes	integer
C_FSL_LINKS[1]	Number of stream interfaces (FSL or AXI)	0-16	0	yes	integer
C_FSL_DATA_SIZE	FSL data bus size	32	32	NA	integer
C_USE_EXTENDED_FSL_INSTR	Enable use of extended stream instructions	0, 1	0		integer
C_ICACHE_BASEADDR	Instruction cache base address	0x00000000 - 0xFFFFFFFF	0x0000 0000		std_logic_vector
C_ICACHE_HIGHADDR	Instruction cache high address	0x00000000 - 0xFFFFFFFF	0x3FFF FFFF		std_logic_vector
C_USE_ICACHE	Instruction cache	0, 1	0		integer
C_ALLOW_ICACHE_WR	Instruction cache write enable	0, 1	1		integer
C_ICACHE_LINE_LEN	Instruction cache line length	4, 8	4		integer
C_ICACHE_ALWAYS_USED	Instruction cache CacheLink used for all memory accesses	0, 1	0		integer
C_ICACHE_INTERFACE	Instruction cache CacheLink interface protocol 0 = IXCL 1 = IXCL2	0, 1	0	yes[2]	integer

Table 2-12: **MPD Parameters *(Continued)***

Parameter Name	Feature/Description	Allowable Values	Default Value	EDK Tool Assigned	VHDL Type
C_ICACHE_FORCE_TAG_LUTRAM	Instruction cache tag always implemented with distributed RAM	0, 1	0		integer
C_ICACHE_STREAMS	Instruction cache streams	0, 1	0		integer
C_ICACHE_VICTIMS	Instruction cache victims	0, 2, 4, 8	0		integer
C_ICACHE_DATA_WIDTH	Instruction cache data width 0 = 32 bits 1 = Full cache line	0, 1	0		integer
C_ADDR_TAG_BITS	Instruction cache address tags	0-25	17	yes	integer
C_CACHE_BYTE_SIZE	Instruction cache size	64, 128, 256, 512, 1024, 2048, 4096, 8192, 16384, 32768, 65536[3]	8192		integer
C_ICACHE_USE_FSL	Cache over CacheLink instead of peripheral bus for instructions	1	1		integer
C_DCACHE_BASEADDR	Data cache base address	0x00000000 - 0xFFFFFFFF	0x0000 0000		std_logic_vector
C_DCACHE_HIGHADDR	Data cache high address	0x00000000 - 0xFFFFFFFF	0x3FFF FFFF		std_logic_vector
C_USE_DCACHE	Data cache	0, 1	0		integer
C_ALLOW_DCACHE_WR	Data cache write enable	0, 1	1		integer
C_DCACHE_LINE_LEN	Data cache line length	4, 8	4		integer
C_DCACHE_ALWAYS_USED	Data cache CacheLink used for all accesses	0, 1	0		integer
C_DCACHE_INTERFACE	Data cache CacheLink interface protocol 0 = DXCL 1 = DXCL2	0, 1	0	yes[2]	integer

Table 2-12: **MPD Parameters *(Continued)***

Parameter Name	Feature/Description	Allowable Values	Default Value	EDK Tool Assigned	VHDL Type
C_DCACHE_FORCE_TAG_LUTRAM	Data cache tag always implemented with distributed RAM	0, 1	0		integer
C_DCACHE_USE_WRITEBACK	Data cache write-back storage policy used	0, 1	0		integer
C_DCACHE_VICTIMS	Data cache victims	0, 2, 4, 8	0		integer
C_DCACHE_DATA_WIDTH	Data cache data width 0 = 32 bits 1 = Full cache line	0, 1	0		integer
C_DCACHE_ADDR_TAG	Data cache address tags	0-25	17	yes	integer
C_DCACHE_BYTE_SIZE	Data cache size	64, 128, 256, 512, 1024, 2048, 4096, 8192, 16384, 32768, 65536[3]	8192		integer
C_DCACHE_USE_FSL	Cache over CacheLink instead of peripheral bus for data	1	1		integer
C_DPLB_DWIDTH	Data side PLB data width	32	32		integer
C_DPLB_NATIVE_DWIDTH	Data side PLB native data width	32	32		integer
C_DPLB_BURST_EN	Data side PLB burst enable	0	0		integer
C_DPLB_P2P	Data side PLB Point-to-point	0, 1	0		integer
C_IPLB_DWIDTH	Instruction side PLB data width	32	32		integer
C_IPLB_NATIVE_DWIDTH	Instruction side PLB native data width	32	32		integer
C_IPLB_BURST_EN	Instruction side PLB burst enable	0	0		integer
C_IPLB_P2P	Instruction side PLB Point-to-point	0, 1	0		integer

Table 2-12: **MPD Parameters *(Continued)***

Parameter Name	Feature/Description	Allowable Values	Default Value	EDK Tool Assigned	VHDL Type
C_USE_MMU[4]	Memory Management: 0 = None 1 = Usermode 2 = Protection 3 = Virtual	0, 1, 2, 3	0		integer
C_MMU_DTLB_SIZE[4]	Data shadow Translation Look-Aside Buffer size	1, 2, 4, 8	4		integer
C_MMU_ITLB_SIZE[4]	Instruction shadow Translation Look-Aside Buffer size	1, 2, 4, 8	2		integer
C_MMU_TLB_ACCESS[4]	Access to memory management special registers: 0 = Minimal 1 = Read 2 = Write 3 = Full	0, 1, 2, 3	3		integer
C_MMU_ZONES[4]	Number of memory protection zones	0-16	16		integer
C_MMU_PRIVILEGED_INSTR[4]	Privileged instructions 0 = Full protection 1 = Allow stream instrs	0,1	0		integer
C_USE_INTERRUPT	Enable interrupt handling	0,1	0	yes	integer
C_USE_EXT_BRK	Enable external break handling	0,1	0	yes	integer
C_USE_EXT_NM_BRK	Enable external non-maskable break handling	0,1	0	yes	integer
C_USE_BRANCH_TARGET_CACHE[4]	Enable Branch Target Cache	0,1	0		integer

Table 2-12: **MPD Parameters *(Continued)***

Parameter Name	Feature/Description	Allowable Values	Default Value	EDK Tool Assigned	VHDL Type
C_BRANCH_TARGET_CACHE_SIZE[4]	Branch Target Cache size: 0 = Default 1 = 8 entries 2 = 16 entries 3 = 32 entries 4 = 64 entries 5 = 512 entries 6 = 1024 entries 7 = 2048 entries	0-7	0		integer
C_M_AXI_DP_THREAD_ID_WIDTH	Data side AXI thread ID width	1	1		integer
C_M_AXI_DP_DATA_WIDTH	Data side AXI data width	32	32		integer
C_M_AXI_DP_ADDR_WIDTH	Data side AXI address width	32	32		integer
C_M_AXI_DP_SUPPORTS_THREADS	Data side AXI uses threads	0	0		integer
C_M_AXI_DP_SUPPORTS_READ	Data side AXI support for read accesses	1	1		integer
C_M_AXI_DP_SUPPORTS_WRITE	Data side AXI support for write accesses	1	1		integer
C_M_AXI_DP_SUPPORTS_NARROW_BURST	Data side AXI narrow burst support	0	0		integer
C_M_AXI_DP_PROTOCOL	Data side AXI protocol	AXI4, AXI4LITE	AXI4 LITE		string
C_M_AXI_DP_EXCLUSIVE_ACCESS	Data side AXI exclusive access support	0,1	0		integer
C_INTERCONNECT_M_AXI_DP_READ_ISSUING	Data side AXI read accesses issued	1	1		integer
C_INTERCONNECT_M_AXI_DP_WRITE_ISSUING	Data side AXI write accesses issued	1	1		integer
C_M_AXI_IP_THREAD_ID_WIDTH	Instruction side AXI thread ID width	1	1		integer
C_M_AXI_IP_DATA_WIDTH	Instruction side AXI data width	32	32		integer

Table 2-12: **MPD Parameters *(Continued)***

Parameter Name	Feature/Description	Allowable Values	Default Value	EDK Tool Assigned	VHDL Type
C_M_AXI_IP_ADDR_WIDTH	Instruction side AXI address width	32	32		integer
C_M_AXI_IP_SUPPORTS_THREADS	Instruction side AXI uses threads	0	0		integer
C_M_AXI_IP_SUPPORTS_READ	Instruction side AXI support for read accesses	1	1		integer
C_M_AXI_IP_SUPPORTS_WRITE	Instruction side AXI support for write accesses	0	0		integer
C_M_AXI_IP_SUPPORTS_NARROW_BURST	Instruction side AXI narrow burst support	0	0		integer
C_M_AXI_IP_PROTOCOL	Instruction side AXI protocol	AXI4LITE	AXI4 LITE		string
C_INTERCONNECT_M_AXI_IP_READ_ISSUING	Instruction side AXI read accesses issued	1	1		integer
C_M_AXI_DC_THREAD_ID_WIDTH	Data cache AXI ID width	1	1		integer
C_M_AXI_DC_DATA_WIDTH	Data cache AXI data width	32,64,128,256	32		integer
C_M_AXI_DC_ADDR_WIDTH	Data cache AXI address width	32	32		integer
C_M_AXI_DC_SUPPORTS_THREADS	Data cache AXI uses threads	0	0		integer
C_M_AXI_DC_SUPPORTS_READ	Data cacheAXI support for read accesses	1	1		integer
C_M_AXI_DC_SUPPORTS_WRITE	Data cache AXI support for write accesses	1	1		integer
C_M_AXI_DC_SUPPORTS_NARROW_BURST	Data cache AXI narrow burst support	0	0		integer
C_M_AXI_DC_SUPPORTS_USER_SIGNALS	Data cache AXI user signal support	1	1		integer
C_M_AXI_DC_PROTOCOL	Data cache AXI protocol	AXI4	AXI4		string
C_M_AXI_DC_AWUSER_WIDTH	Data cache AXI user width	5	5		integer
C_M_AXI_DC_ARUSER_WIDTH	Data cache AXI user width	5	5		integer

Table 2-12: **MPD Parameters *(Continued)***

Parameter Name	Feature/Description	Allowable Values	Default Value	EDK Tool Assigned	VHDL Type
C_M_AXI_DC_WUSER_WIDTH	Data cache AXI user width	1	1		integer
C_M_AXI_DC_RUSER_WIDTH	Data cache AXI user width	1	1		integer
C_M_AXI_DC_BUSER_WIDTH	Data cache AXI user width	1	1		integer
C_M_AXI_DC_EXCLUSIVE_ACCESS	Data cache AXI exclusive access support	0,1	0		integer
C_M_AXI_DC_USER_VALUE	Data cache AXI user value	0-31	31		integer
C_INTERCONNECT_M_AXI_DC_READ_ISSUING	Data cache AXI read accesses issued	1,2	2		integer
C_INTERCONNECT_M_AXI_DC_WRITE_ISSUING	Data cache AXI write accesses issued	1,2,4,8,16,32	32		integer
C_M_AXI_IC_THREAD_ID_WIDTH	Instruction cache AXI ID width	1	1		integer
C_M_AXI_IC_DATA_WIDTH	Instruction cache AXI data width	32,64,128,256	32		integer
C_M_AXI_IC_ADDR_WIDTH	Instruction cache AXI address width	32	32		integer
C_M_AXI_IC_SUPPORTS_THREADS	Instruction cache AXI uses threads	0	0		integer
C_M_AXI_IC_SUPPORTS_READ	Instruction cache AXI support for read accesses	1	1		integer
C_M_AXI_IC_SUPPORTS_WRITE	Instruction cache AXI support for write accesses	0	0		integer
C_M_AXI_IC_SUPPORTS_NARROW_BURST	Instruction cache AXI narrow burst support	0	0		integer
C_M_AXI_IC_SUPPORTS_USER_SIGNALS	Instruction cache AXI user signal support	1	1		integer
C_M_AXI_IC_PROTOCOL	Instruction cache AXI protocol	AXI4	AXI4		string
C_M_AXI_IC_AWUSER_WIDTH	Instruction cache AXI user width	5	5		integer
C_M_AXI_IC_ARUSER_WIDTH	Instruction cache AXI user width	5	5		integer

Table 2-12: **MPD Parameters *(Continued)***

Parameter Name	Feature/Description	Allowable Values	Default Value	EDK Tool Assigned	VHDL Type
C_M_AXI_IC_WUSER_WIDTH	Instruction cache AXI user width	1	1		integer
C_M_AXI_IC_RUSER_WIDTH	Instruction cache AXI user width	1	1		integer
C_M_AXI_IC_BUSER_WIDTH	Instruction cache AXI user width	1	1		integer
C_M_AXI_IC_USER_VALUE	Instruction cache AXI user value	0-31	31		integer
C_INTERCONNECT_ M_AXI_IC_READ_ISSUING	Instruction cache AXI read accesses issued	1,2,4,8	2	yes	integer
C_STREAM_INTERCONNECT	Select AXI4-Stream interconnect	0,1	0		integer
C_Mn_AXIS_PROTOCOL	AXI4-Stream protocol	GENERIC	GENERIC		string
C_Sn_AXIS_PROTOCOL	AXI4-Stream protocol	GENERIC	GENERIC		string
C_Mn_AXIS_DATA_WIDTH	AXI4-Stream master data width	32	32	NA	integer
C_Sn_AXIS_DATA_WIDTH	AXI4-Stream slave data width	32	32	NA	integer

1. The number of Stream Links (FSL or AXI4) is assigned by the tool itself if you are using the co-processor wizard. If you add the IP manually, you must update the parameter manually.
2. EDK tool assigned value can be overridden by explicit assignment.
3. Not all sizes are permitted in all architectures. The cache uses between 0 and 32 RAMB primitives (0 if cache size is less than 2048).
4. Not available when C_AREA_OPTIMIZED is set to 1.

XILINX®

Chapter 3

MicroBlaze Application Binary Interface

This chapter describes MicroBlaze™ Application Binary Interface (ABI), which is important for developing software in assembly language for the soft processor. The MicroBlaze GNU compiler follows the conventions described in this document. Any code written by assembly programmers should also follow the same conventions to be compatible with the compiler generated code. Interrupt and Exception handling is also explained briefly.

This chapter contains the following sections:

Data Types

The data types used by MicroBlaze assembly programs are shown in Table 3-1. Data types such as data8, data16, and data32 are used in place of the usual byte, half-word, and word.register

Table 3-1: **Data Types in MicroBlaze Assembly Programs**

MicroBlaze data types (for assembly programs)	Corresponding ANSI C data types	Size (bytes)
data8	char	1
data16	short	2
data32	int	4
data32	long int	4
data32	float	4
data32	enum	4
data16/data32	pointer[a]	2/4

a. Pointers to small data areas, which can be accessed by global pointers are data16.

Register Usage Conventions

The register usage convention for MicroBlaze is given in Table 3-2.

Table 3-2: **Register Usage Conventions**

Register	Type	Enforcement	Purpose
R0	Dedicated	HW	Value 0
R1	Dedicated	SW	Stack Pointer
R2	Dedicated	SW	Read-only small data area anchor
R3-R4	Volatile	SW	Return Values/Temporaries
R5-R10	Volatile	SW	Passing parameters/Temporaries
R11-R12	Volatile	SW	Temporaries
R13	Dedicated	SW	Read-write small data area anchor
R14	Dedicated	HW	Return address for Interrupt
R15	Dedicated	SW	Return address for Sub-routine
R16	Dedicated	HW	Return address for Trap (Debugger)
R17	Dedicated	HW, if configured to support HW exceptions, else SW	Return address for Exceptions
R18	Dedicated	SW	Reserved for Assembler/Compiler Temporaries
R19	Non-volatile	SW	Must be saved across function calls. Callee-save
R20	Dedicated or Non-volatile	SW	Reserved for storing a pointer to the Global Offset Table (GOT) in Position Independent Code (PIC). Non-volatile in non-PIC code. Must be saved across function calls. Callee-save
R21-R31	Non-volatile	SW	Must be saved across function calls. Callee-save
RPC	Special	HW	Program counter
RMSR	Special	HW	Machine Status Register
REAR	Special	HW	Exception Address Register
RESR	Special	HW	Exception Status Register
RFSR	Special	HW	Floating Point Status Register
RBTR	Special	HW	Branch Target Register
REDR	Special	HW	Exception Data Register
RPID	Special	HW	Process Identifier Register
RZPR	Special	HW	Zone Protection Register
RTLBLO	Special	HW	Translation Look-Aside Buffer Low Register
RTLBHI	Special	HW	Translation Look-Aside Buffer High Register
RTLBX	Special	HW	Translation Look-Aside Buffer Index Register
RTLBSX	Special	HW	Translation Look-Aside Buffer Search Index
RPVR0-RPVR11	Special	HW	Processor Version Register 0 through 11

The architecture for MicroBlaze defines 32 general purpose registers (GPRs). These registers are classified as volatile, non-volatile, and dedicated.

- The volatile registers (also known as caller-save) are used as temporaries and do not retain values across the function calls. Registers R3 through R12 are volatile, of which R3 and R4 are used for returning values to the caller function, if any. Registers R5 through R10 are used for passing parameters between subroutines.
- Registers R19 through R31 retain their contents across function calls and are hence termed as non-volatile registers (a.k.a callee-save). The callee function is expected to save those non-volatile registers, which are being used. These are typically saved to the stack during the prologue and then reloaded during the epilogue.
- Certain registers are used as dedicated registers and programmers are not expected to use them for any other purpose.
  - Registers R14 through R17 are used for storing the return address from interrupts, sub-routines, traps, and exceptions in that order. Subroutines are called using the branch and link instruction, which saves the current Program Counter (PC) onto register R15.
  - Small data area pointers are used for accessing certain memory locations with 16- bit immediate value. These areas are discussed in the memory model section of this document. The read only small data area (SDA) anchor R2 (Read-Only) is used to access the constants such as literals. The other SDA anchor R13 (Read-Write) is used for accessing the values in the small data read-write section.
  - Register R1 stores the value of the stack pointer and is updated on entry and exit from functions.
  - Register R18 is used as a temporary register for assembler operations.
- MicroBlaze includes special purpose registers such as: program counter (rpc), machine status register (rmsr), exception status register (resr), exception address register (rear), floating point status register (rfsr), branch target register (rbtr), exception data register (redr), memory management registers (rpid, rzpr, rtlblo, rtlbhi, rtlbx, rtlbsx), and processor version registers (rpvr0-rpvr11). These registers are not mapped directly to the register file and hence the usage of these registers is different from the general purpose registers. The value of a special purpose registers can be transferred to or from a general purpose register by using **`mts`** and **`mfs`** instructions respectively.

Stack Convention

The stack conventions used by MicroBlaze are detailed in Table 3-3.

The shaded area in Table 3-3 denotes a part of the stack frame for a caller function, while the unshaded area indicates the callee frame function. The ABI conventions of the stack frame define the protocol for passing parameters, preserving non-volatile register values, and allocating space for the local variables in a function.

Functions that contain calls to other subroutines are called as non-leaf functions. These non-leaf functions have to create a new stack frame area for its own use. When the program starts executing, the stack pointer has the maximum value. As functions are called, the stack pointer is decremented by the number of words required by every function for its stack frame. The stack pointer of a caller function always has a higher value as compared to the callee function.

Table 3-3: **Stack Convention**

High Address	
	Function Parameters for called sub-routine (Arg n .. Arg1) (Optional: Maximum number of arguments required for any called procedure from the current procedure).
Old Stack Pointer	Link Register (R15)
	Callee Saved Register (R31....R19) (Optional: Only those registers which are used by the current procedure are saved)
	Local Variables for Current Procedure (Optional: Present only if Locals defined in the procedure)
	Functional Parameters (Arg n .. Arg 1) (Optional: Maximum number of arguments required for any called procedure from the current procedure)
New Stack Pointer	Link Register
Low Address	

Consider an example where Func1 calls Func2, which in turn calls Func3. The stack representation at different instances is depicted in Figure 3-1. After the call from Func 1 to Func 2, the value of the stack pointer (SP) is decremented. This value of SP is again decremented to accommodate the stack frame for Func3. On return from Func 3 the value of the stack pointer is increased to its original value in the function, Func 2.

Details of how the stack is maintained are shown in Figure 3-1.

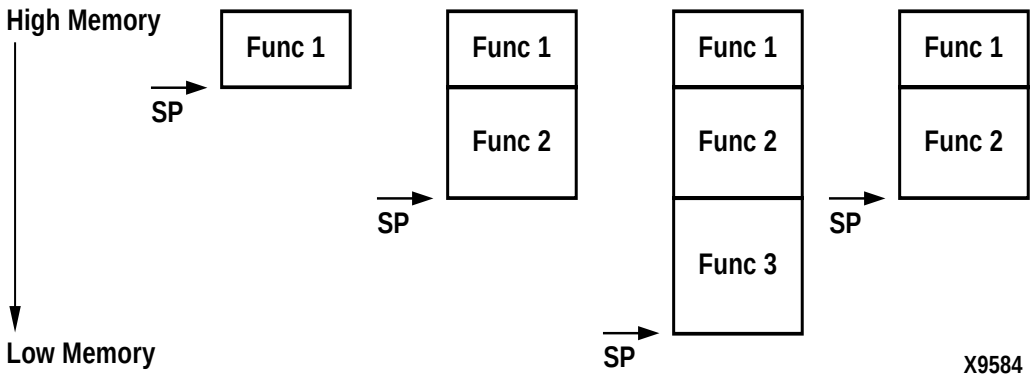


Figure 3-1: **Stack Frame**

Calling Convention

The caller function passes parameters to the callee function using either the registers (R5 through R10) or on its own stack frame. The callee uses the stack area of the caller to store the parameters passed to the callee.

Refer to Figure 3-1. The parameters for Func 2 are stored either in the registers R5 through R10 or on the stack frame allocated for Func 1.

Memory Model

The memory model for MicroBlaze classifies the data into four different parts: Small Data Area, Data Area, Common Un-Initialized Area, and Literals or Constants.

Small Data Area

Global initialized variables which are small in size are stored in this area. The threshold for deciding the size of the variable to be stored in the small data area is set to 8 bytes in the MicroBlaze C compiler (mb-gcc), but this can be changed by giving a command line option to the compiler. Details about this option are discussed in the *GNU Compiler Tools* chapter. 64 kilobytes of memory is allocated for the small data areas. The small data area is accessed using the read-write small data area anchor (R13) and a 16-bit offset. Allocating small variables to this area reduces the requirement of adding **IMM** instructions to the code for accessing global variables. Any variable in the small data area can also be accessed using an absolute address.

Data Area

Comparatively large initialized variables are allocated to the data area, which can either be accessed using the read-write SDA anchor R13 or using the absolute address, depending on the command line option given to the compiler.

Common Un-Initialized Area

Un-initialized global variables are allocated in the common area and can be accessed either using the absolute address or using the read-write small data area anchor R13.

Literals or Constants

Constants are placed into the read-only small data area and are accessed using the read-only small data area anchor R2.

The compiler generates appropriate global pointers to act as base pointers. The actual values of the SDA anchors are decided by the linker, in the final linking stages. For more information on the various sections of the memory please refer to *MicroBlaze Linker Script Sections* in the *Embedded System Tools Reference Manual*.The compiler generates appropriate sections, depending on the command line options. Please refer to the *GNU Compiler Tools* chapter in the *Embedded System Tools Reference Manual* for more information about these options.

Interrupt and Exception Handling

MicroBlaze assumes certain address locations for handling interrupts and exceptions as indicated in Table 3-4. At these locations, code is written to jump to the appropriate handlers.

Table 3-4: **Interrupt and Exception Handling**

On	Hardware jumps to	Software Labels
Start / Reset	0x0	_start
User exception	0x8	_exception_handler
Interrupt	0x10	_interrupt_handler
Break (HW/SW)	0x18	-
Hardware exception	0x20	_hw_exception_handler
Reserved by Xilinx for future use	0x28 - 0x4F	-

The code expected at these locations is as shown below. For programs compiled without the **`-xl-mode-xmdstub`** compiler option, the `crt0.o` initialization file is passed by the **`mb-gcc`** compiler to the **mb-ld** linker for linking. This file sets the appropriate addresses of the exception handlers.

For programs compiled with the **`-xl-mode-xmdstub`** compiler option, the `crt1.o` initialization file is linked to the output program. This program has to be run with the xmdstub already loaded in the memory at address location 0x0. Hence at run-time, the initialization code in `crt1.o` writes the appropriate instructions to location 0x8 through 0x14 depending on the address of the exception and interrupt handlers.

The following is code for passing control to Exception and Interrupt handlers:

```
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 allows exception and interrupt handler routines to be located at any address location addressable using 32 bits. The user exception handler code starts with the label **`_exception_handler`**, the hardware exception handler starts with **`_hw_exception_handler`**, while the interrupt handler code starts with the label **`_interrupt_handler`**.

In the current MicroBlaze system, there are dummy routines for interrupt and exception handling, which you can change. In order to override these routines and link your interrupt and exception handlers, you must define the interrupt handler code with an attribute **`interrupt_handler`**. For more details about the use and syntax of the interrupt handler attribute, please refer to the *GNU Compiler Tools* chapter in the *Embedded System Tools Reference Guide*.

When software breakpoints are used in the Xilinx Microprocessor Debug (XMD) tool, the Break (HW/SW) address location is reserved for handling the software breakpoint.

XILINX®

Chapter 4

MicroBlaze Instruction Set Architecture

This chapter provides a detailed guide to the Instruction Set Architecture of MicroBlaze™ and contains the following sections:

- "Notation"
- "Formats"
- "Instructions"

Notation

The symbols used throughout this chapter are defined in Table 4-1.

Table 4-1: **Symbol Notation**

Symbol	Meaning
+	Add
-	Subtract
×	Multiply
/	Divide
∧	Bitwise logical AND
∨	Bitwise logical OR
⊕	Bitwise logical XOR
x	Bitwise logical complement of *x*
←	Assignment
>>	Right shift
<<	Left shift
r*x*	Register *x*
x[*i*]	Bit *i* in register *x*
x[*i*:*j*]	Bits *i* through *j* in register *x*
=	Equal comparison
≠	Not equal comparison
>	Greater than comparison
>=	Greater than or equal comparison

Table 4-1: **Symbol Notation *(Continued)***

Symbol	Meaning
<	Less than comparison
<=	Less than or equal comparison
|	Signal choice
sext(*x*)	Sign-extend *x*
Mem(*x*)	Memory location at address *x*
FSLx	Stream interface x (FSL or AXI)
LSW(*x*)	Least Significant Word of *x*
isDnz(*x*)	Floating point: true if *x* is denormalized
isInfinite(*x*)	Floating point: true if *x* is +∞ or -∞
isPosInfinite(*x*)	Floating point: true if *x* is +∞
isNegInfinite(*x*)	Floating point: true if *x* -∞
isNaN(*x*)	Floating point: true if *x* is a quiet or signalling NaN
isZero(*x*)	Floating point: true if *x* is +0 or -0
isQuietNaN(*x*)	Floating point: true if *x* is a quiet NaN
isSigNaN(*x*)	Floating point: true if *x* is a signaling NaN
signZero(*x*)	Floating point: return +0 for $x > 0$, and -0 if $x < 0$
signInfinite(*x*)	Floating point: return +∞ for $x > 0$, and -∞ if $x < 0$

Formats

MicroBlaze uses two instruction formats: Type A and Type B.

Type A

Type A is used for register-register instructions. It contains the opcode, one destination and two source registers.

Opcode	Destination Reg	Source Reg A	Source Reg B	0	0	0	0	0	0	0	0	0	0	0
0	6	11	16	21										31

Type B

Type B is used for register-immediate instructions. It contains the opcode, one destination and one source registers, and a source 16-bit immediate value.

Opcode	Destination Reg	Source Reg A	Immediate Value
0	6	11	16 31

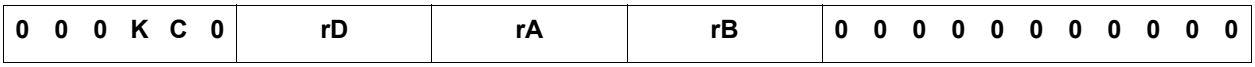
Instructions

This section provides descriptions of MicroBlaze instructions. Instructions are listed in alphabetical order. For each instruction Xilinx® provides the mnemonic, encoding, a description, pseudocode of its semantics, and a list of registers that it modifies.

add

Arithmetic Add

add	rD, rA, rB	Add
addc	rD, rA, rB	Add with Carry
addk	rD, rA, rB	Add and Keep Carry
addkc	rD, rA, rB	Add with Carry and Keep Carry



Description

The sum of the contents of registers rA and rB, is placed into register rD.

Bit 3 of the instruction (labeled as K in the figure) is set to one for the mnemonic addk. Bit 4 of the instruction (labeled as C in the figure) is set to one for the mnemonic addc. Both bits are set to one for the mnemonic addkc.

When an add instruction has bit 3 set (addk, addkc), the carry flag will Keep its previous value regardless of the outcome of the execution of the instruction. If bit 3 is cleared (add, addc), then the carry flag will be affected by the execution of the instruction.

When bit 4 of the instruction is set to one (addc, addkc), the content of the carry flag (MSR[C]) affects the execution of the instruction. When bit 4 is cleared (add, addk), the content of the carry flag does not affect the execution of the instruction (providing a normal addition).

Pseudocode

```
if C = 0 then
  (rD) ← (rA) + (rB)
else
  (rD) ← (rA) + (rB) + MSR[C]
if K = 0 then
  MSR[C] ← CarryOut
```

Registers Altered

- rD
- MSR[C]

Latency

1 cycle

Note

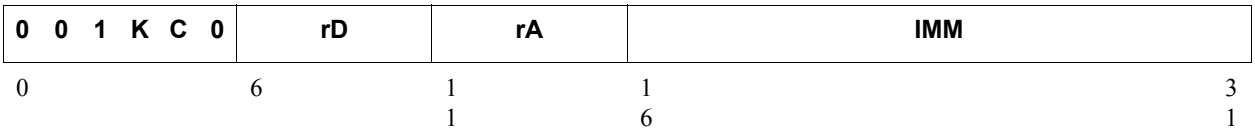
The C bit in the instruction opcode is not the same as the carry bit in the MSR.

The “add r0, r0, r0” (= 0x00000000) instruction is never used by the compiler and usually indicates uninitialized memory. If you are using illegal instruction exceptions you can trap these instructions by setting the MicroBlaze parameter `C_OPCODE_0x0_ILLEGAL=1`.

addi

Arithmetic Add Immediate

addi	rD, rA, IMM	Add Immediate
addic	rD, rA, IMM	Add Immediate with Carry
addik	rD, rA, IMM	Add Immediate and Keep Carry
addikc	rD, rA, IMM	Add Immediate with Carry and Keep Carry



Description

The sum of the contents of registers rA and the value in the IMM field, sign-extended to 32 bits, is placed into register rD. Bit 3 of the instruction (labeled as K in the figure) is set to one for the mnemonic addik. Bit 4 of the instruction (labeled as C in the figure) is set to one for the mnemonic addic. Both bits are set to one for the mnemonic addikc.

When an addi instruction has bit 3 set (addik, addikc), the carry flag will keep its previous value regardless of the outcome of the execution of the instruction. If bit 3 is cleared (addi, addic), then the carry flag will be affected by the execution of the instruction.

When bit 4 of the instruction is set to one (addic, addikc), the content of the carry flag (MSR[C]) affects the execution of the instruction. When bit 4 is cleared (addi, addik), the content of the carry flag does not affect the execution of the instruction (providing a normal addition).

Pseudocode

```
if C = 0 then
  (rD) ← (rA) + sext(IMM)
else
  (rD) ← (rA) + sext(IMM) + MSR[C]
if K = 0 then
  MSR[C] ← CarryOut
```

Registers Altered

- rD
- MSR[C]

Latency

1 cycle

Notes

The C bit in the instruction opcode is not the same as the carry bit in the MSR.

By default, Type B Instructions take the 16-bit IMM field value and sign extend it to 32 bits to use as the immediate operand. This behavior can be overridden by preceding the Type B instruction with an imm instruction. See the instruction “imm,” page 180 for details on using 32-bit immediate values.

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						6	11	16	21										31

Description

The contents of register rA are ANDed with the contents of register rB; the result is placed into register rD.

Pseudocode

$(rD) \leftarrow (rA) \wedge (rB)$

Registers Altered

- rD

Latency

1 cycle

andi

Logial AND with Immediate

andi rD, rA, IMM

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

Description

The contents of register rA are ANDed with the value of the IMM field, sign-extended to 32 bits; the result is placed into register rD.

Pseudocode

```
(rD) ← (rA) ∧ sext(IMM)
```

Registers Altered

- rD

Latency

1 cycle

Note

By default, Type B Instructions will take the 16-bit IMM field value and sign extend it to 32 bits to use as the immediate operand. This behavior can be overridden by preceding the Type B instruction with an imm instruction. See the instruction “imm,” page 180 for details on using 32-bit immediate values.

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						6	11	16	21										31

Description

The contents of register rA are ANDed with the logical complement of the contents of register rB; the result is placed into register rD.

Pseudocode

$(rD) \leftarrow (rA) \wedge (\overline{rB})$

Registers Altered

- rD

Latency

1 cycle

andni

Logical AND NOT with Immediate

andni rD, rA, IMM

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

Description

The IMM field is sign-extended to 32 bits. The contents of register rA are ANDed with the logical complement of the extended IMM field; the result is placed into register rD.

Pseudocode

```
(rD) ← (rA) ∧ (~sext(IMM))
```

Registers Altered

- rD

Latency

1 cycle

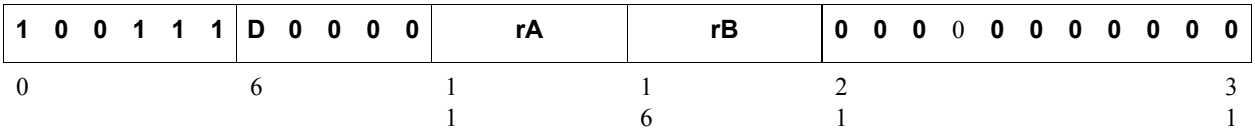
Note

By default, Type B Instructions will take the 16-bit IMM field value and sign extend it to 32 bits to use as the immediate operand. This behavior can be overridden by preceding the Type B instruction with an imm instruction. See the instruction “imm,” page 180 for details on using 32-bit immediate values.

beq

Branch if Equal

beq	rA, rB	Branch if Equal
beqd	rA, rB	Branch if Equal with Delay



Description

Branch if rA is equal to 0, to the instruction located in the offset value of rB. The target of the branch will be the instruction at address PC + rB.

The mnemonic beqd will set the D bit. The D bit determines whether there is a branch delay slot or not. If the D bit is set, it means that there is a delay slot and the instruction following the branch (that is, in the branch delay slot) is allowed to complete execution before executing the target instruction. If the D bit is not set, it means that there is no delay slot, so the instruction to be executed after the branch is the target instruction.

Pseudocode

```
If rA = 0 then
  PC ← PC + rB
else
  PC ← PC + 4
if D = 1 then
  allow following instruction to complete execution
```

Registers Altered

- PC

Latency

1 cycle (if branch is not taken)

2 cycles (if branch is taken and the D bit is set)

3 cycles (if branch is taken and the D bit is not set)

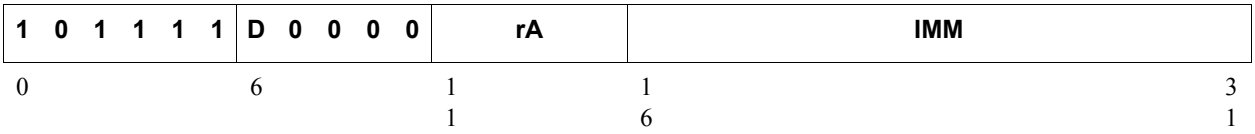
Note

A delay slot must not be used by the following: imm, branch, or break instructions. Interrupts and external hardware breaks are deferred until after the delay slot branch has been completed.

beqi

Branch Immediate if Equal

beqi	rA, IMM	Branch Immediate if Equal
beqid	rA, IMM	Branch Immediate if Equal with Delay



Description

Branch if rA is equal to 0, to the instruction located in the offset value of IMM. The target of the branch will be the instruction at address PC + IMM.

The mnemonic beqid will set the D bit. The D bit determines whether there is a branch delay slot or not. If the D bit is set, it means that there is a delay slot and the instruction following the branch (that is, in the branch delay slot) is allowed to complete execution before executing the target instruction. If the D bit is not set, it means that there is no delay slot, so the instruction to be executed after the branch is the target instruction.

Pseudocode

```
If rA = 0 then
  PC ← PC + sext(IMM)
else
  PC ← PC + 4
if D = 1 then
  allow following instruction to complete execution
```

Registers Altered

- PC

Latency

1 cycle (if branch is not taken, or successful branch prediction occurs)

2 cycles (if branch is taken and the D bit is set)

3 cycles (if branch is taken and the D bit is not set, or a branch prediction mispredict occurs)

Note

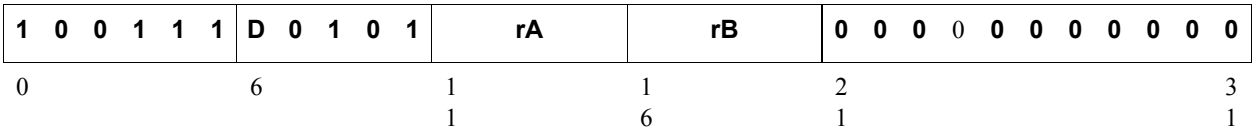
By default, Type B Instructions will take the 16-bit IMM field value and sign extend it to 32 bits to use as the immediate operand. This behavior can be overridden by preceding the Type B instruction with an imm instruction. See the instruction “imm,” page 180 for details on using 32-bit immediate values.

A delay slot must not be used by the following: imm, branch, or break instructions. Interrupts and external hardware breaks are deferred until after the delay slot branch has been completed.

bge

Branch if Greater or Equal

bge	rA, rB	Branch if Greater or Equal
bged	rA, rB	Branch if Greater or Equal with Delay



Description

Branch if rA is greater or equal to 0, to the instruction located in the offset value of rB. The target of the branch will be the instruction at address PC + rB.

The mnemonic bged will set the D bit. The D bit determines whether there is a branch delay slot or not. If the D bit is set, it means that there is a delay slot and the instruction following the branch (that is, in the branch delay slot) is allowed to complete execution before executing the target instruction. If the D bit is not set, it means that there is no delay slot, so the instruction to be executed after the branch is the target instruction.

Pseudocode

```
If rA >= 0 then
  PC ← PC + rB
else
  PC ← PC + 4
if D = 1 then
  allow following instruction to complete execution
```

Registers Altered

- PC

Latency

- 1 cycle (if branch is not taken)
- 2 cycles (if branch is taken and the D bit is set)
- 3 cycles (if branch is taken and the D bit is not set)

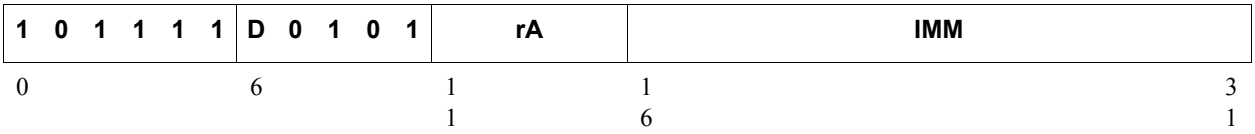
Note

A delay slot must not be used by the following: imm, branch, or break instructions. Interrupts and external hardware breaks are deferred until after the delay slot branch has been completed.

bgei

Branch Immediate if Greater or Equal

bgei	rA, IMM	Branch Immediate if Greater or Equal
bgeid	rA, IMM	Branch Immediate if Greater or Equal with Delay



Description

Branch if rA is greater or equal to 0, to the instruction located in the offset value of IMM. The target of the branch will be the instruction at address PC + IMM.

The mnemonic bgeid will set the D bit. The D bit determines whether there is a branch delay slot or not. If the D bit is set, it means that there is a delay slot and the instruction following the branch (that is, in the branch delay slot) is allowed to complete execution before executing the target instruction. If the D bit is not set, it means that there is no delay slot, so the instruction to be executed after the branch is the target instruction.

Pseudocode

```
If rA >= 0 then
  PC ← PC + sext(IMM)
else
  PC ← PC + 4
if D = 1 then
  allow following instruction to complete execution
```

Registers Altered

- PC

Latency

- 1 cycle (if branch is not taken, or successful branch prediction occurs)
- 2 cycles (if branch is taken and the D bit is set)
- 3 cycles (if branch is taken and the D bit is not set, or a branch prediction mispredict occurs)

Note

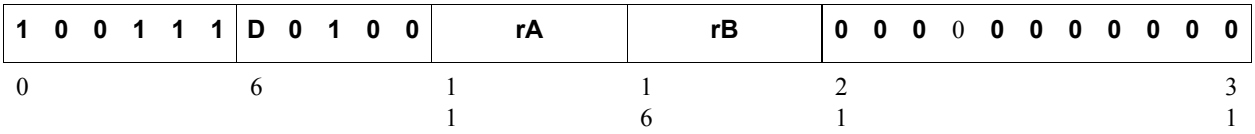
By default, Type B Instructions will take the 16-bit IMM field value and sign extend it to 32 bits to use as the immediate operand. This behavior can be overridden by preceding the Type B instruction with an imm instruction. See the instruction “imm,” page 180 for details on using 32-bit immediate values.

A delay slot must not be used by the following: imm, branch, or break instructions. Interrupts and external hardware breaks are deferred until after the delay slot branch has been completed.

bgt

Branch if Greater Than

bgt	rA, rB	Branch if Greater Than
bgtd	rA, rB	Branch if Greater Than with Delay



Description

Branch if rA is greater than 0, to the instruction located in the offset value of rB. The target of the branch will be the instruction at address PC + rB.

The mnemonic bgtd will set the D bit. The D bit determines whether there is a branch delay slot or not. If the D bit is set, it means that there is a delay slot and the instruction following the branch (that is, in the branch delay slot) is allowed to complete execution before executing the target instruction. If the D bit is not set, it means that there is no delay slot, so the instruction to be executed after the branch is the target instruction.

Pseudocode

```
If rA > 0 then
  PC ← PC + rB
else
  PC ← PC + 4
if D = 1 then
  allow following instruction to complete execution
```

Registers Altered

- PC

Latency

- 1 cycle (if branch is not taken)
- 2 cycles (if branch is taken and the D bit is set)
- 3 cycles (if branch is taken and the D bit is not set)

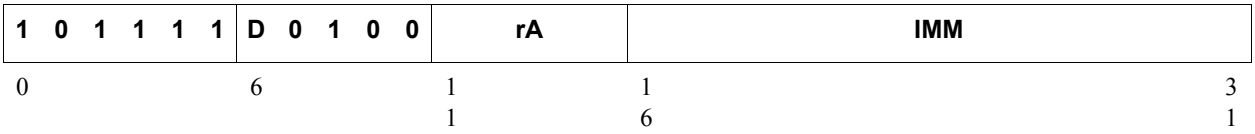
Note

A delay slot must not be used by the following: imm, branch, or break instructions. Interrupts and external hardware breaks are deferred until after the delay slot branch has been completed.

bgti

Branch Immediate if Greater Than

bgti	rA, IMM	Branch Immediate if Greater Than
bgtid	rA, IMM	Branch Immediate if Greater Than with Delay



Description

Branch if rA is greater than 0, to the instruction located in the offset value of IMM. The target of the branch will be the instruction at address PC + IMM.

The mnemonic bgtid will set the D bit. The D bit determines whether there is a branch delay slot or not. If the D bit is set, it means that there is a delay slot and the instruction following the branch (that is, in the branch delay slot) is allowed to complete execution before executing the target instruction. If the D bit is not set, it means that there is no delay slot, so the instruction to be executed after the branch is the target instruction.

Pseudocode

```
If rA > 0 then
  PC ← PC + sext(IMM)
else
  PC ← PC + 4
if D = 1 then
  allow following instruction to complete execution
```

Registers Altered

- PC

Latency

- 1 cycle (if branch is not taken, or successful branch prediction occurs)
- 2 cycles (if branch is taken and the D bit is set)
- 3 cycles (if branch is taken and the D bit is not set, or a branch prediction mispredict occurs)

Note

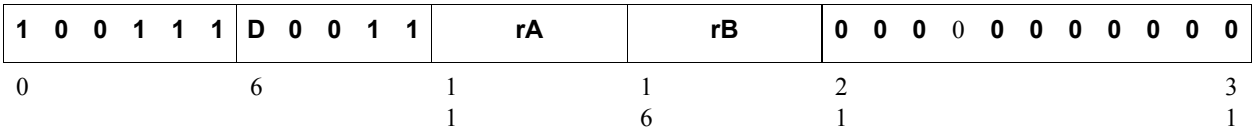
By default, Type B Instructions will take the 16-bit IMM field value and sign extend it to 32 bits to use as the immediate operand. This behavior can be overridden by preceding the Type B instruction with an imm instruction. See the instruction “imm,” page 180 for details on using 32-bit immediate values.

A delay slot must not be used by the following: imm, branch, or break instructions. Interrupts and external hardware breaks are deferred until after the delay slot branch has been completed.

ble

Branch if Less or Equal

ble	rA, rB	Branch if Less or Equal
bled	rA, rB	Branch if Less or Equal with Delay



Description

Branch if rA is less or equal to 0, to the instruction located in the offset value of rB. The target of the branch will be the instruction at address PC + rB.

The mnemonic bled will set the D bit. The D bit determines whether there is a branch delay slot or not. If the D bit is set, it means that there is a delay slot and the instruction following the branch (that is, in the branch delay slot) is allowed to complete execution before executing the target instruction. If the D bit is not set, it means that there is no delay slot, so the instruction to be executed after the branch is the target instruction.

Pseudocode

```
If rA <= 0 then
  PC ← PC + rB
else
  PC ← PC + 4
if D = 1 then
  allow following instruction to complete execution
```

Registers Altered

- PC

Latency

- 1 cycle (if branch is not taken)
- 2 cycles (if branch is taken and the D bit is set)
- 3 cycles (if branch is taken and the D bit is not set)

Note

A delay slot must not be used by the following: imm, branch, or break instructions. Interrupts and external hardware breaks are deferred until after the delay slot branch has been completed.

blei

Branch Immediate if Less or Equal

blei	rA, IMM	Branch Immediate if Less or Equal
bleid	rA, IMM	Branch Immediate if Less or Equal with Delay

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

Description

Branch if rA is less or equal to 0, to the instruction located in the offset value of IMM. The target of the branch will be the instruction at address PC + IMM.

The mnemonic bleid will set the D bit. The D bit determines whether there is a branch delay slot or not. If the D bit is set, it means that there is a delay slot and the instruction following the branch (that is, in the branch delay slot) is allowed to complete execution before executing the target instruction. If the D bit is not set, it means that there is no delay slot, so the instruction to be executed after the branch is the target instruction.

Pseudocode

```
If rA <= 0 then
  PC ← PC + sext(IMM)
else
  PC ← PC + 4
if D = 1 then
  allow following instruction to complete execution
```

Registers Altered

- PC

Latency

- 1 cycle (if branch is not taken, or successful branch prediction occurs)
- 2 cycles (if branch is taken and the D bit is set)
- 3 cycles (if branch is taken and the D bit is not set, or a branch prediction mispredict occurs)

Note

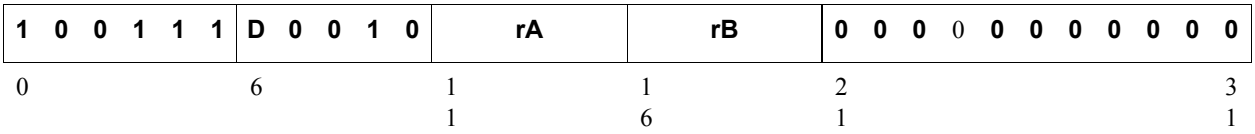
By default, Type B Instructions will take the 16-bit IMM field value and sign extend it to 32 bits to use as the immediate operand. This behavior can be overridden by preceding the Type B instruction with an imm instruction. See the instruction “imm,” page 180 for details on using 32-bit immediate values.

A delay slot must not be used by the following: imm, branch, or break instructions. Interrupts and external hardware breaks are deferred until after the delay slot branch has been completed.

blt

Branch if Less Than

blt	rA, rB	Branch if Less Than
bltd	rA, rB	Branch if Less Than with Delay



Description

Branch if rA is less than 0, to the instruction located in the offset value of rB. The target of the branch will be the instruction at address PC + rB.

The mnemonic bltd will set the D bit. The D bit determines whether there is a branch delay slot or not. If the D bit is set, it means that there is a delay slot and the instruction following the branch (that is, in the branch delay slot) is allowed to complete execution before executing the target instruction. If the D bit is not set, it means that there is no delay slot, so the instruction to be executed after the branch is the target instruction.

Pseudocode

```
If rA < 0 then
  PC ← PC + rB
else
  PC ← PC + 4
if D = 1 then
  allow following instruction to complete execution
```

Registers Altered

- PC

Latency

- 1 cycle (if branch is not taken)
- 2 cycles (if branch is taken and the D bit is set)
- 3 cycles (if branch is taken and the D bit is not set)

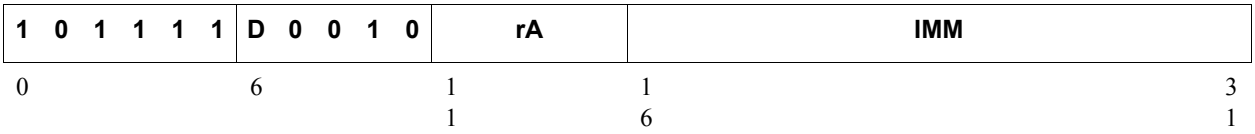
Note

A delay slot must not be used by the following: imm, branch, or break instructions. Interrupts and external hardware breaks are deferred until after the delay slot branch has been completed.

blti

Branch Immediate if Less Than

blti	rA, IMM	Branch Immediate if Less Than
bltid	rA, IMM	Branch Immediate if Less Than with Delay



Description

Branch if rA is less than 0, to the instruction located in the offset value of IMM. The target of the branch will be the instruction at address PC + IMM.

The mnemonic bltid will set the D bit. The D bit determines whether there is a branch delay slot or not. If the D bit is set, it means that there is a delay slot and the instruction following the branch (that is, in the branch delay slot) is allowed to complete execution before executing the target instruction. If the D bit is not set, it means that there is no delay slot, so the instruction to be executed after the branch is the target instruction.

Pseudocode

```
If rA < 0 then
  PC ← PC + sext(IMM)
else
  PC ← PC + 4
if D = 1 then
  allow following instruction to complete execution
```

Registers Altered

- PC

Latency

- 1 cycle (if branch is not taken, or successful branch prediction occurs)
- 2 cycles (if branch is taken and the D bit is set)
- 3 cycles (if branch is taken and the D bit is not set, or a branch prediction mispredict occurs)

Note

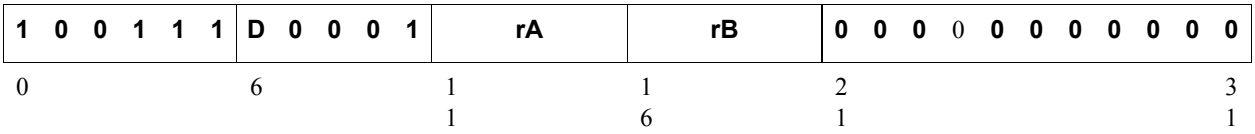
By default, Type B Instructions will take the 16-bit IMM field value and sign extend it to 32 bits to use as the immediate operand. This behavior can be overridden by preceding the Type B instruction with an imm instruction. See the instruction “imm,” page 180 for details on using 32-bit immediate values.

A delay slot must not be used by the following: imm, branch, or break instructions. Interrupts and external hardware breaks are deferred until after the delay slot branch has been completed.

bne

Branch if Not Equal

bne	rA, rB	Branch if Not Equal
bned	rA, rB	Branch if Not Equal with Delay



Description

Branch if rA not equal to 0, to the instruction located in the offset value of rB. The target of the branch will be the instruction at address PC + rB.

The mnemonic bned will set the D bit. The D bit determines whether there is a branch delay slot or not. If the D bit is set, it means that there is a delay slot and the instruction following the branch (that is, in the branch delay slot) is allowed to complete execution before executing the target instruction. If the D bit is not set, it means that there is no delay slot, so the instruction to be executed after the branch is the target instruction.

Pseudocode

```
If rA ≠ 0 then
  PC ← PC + rB
else
  PC ← PC + 4
if D = 1 then
  allow following instruction to complete execution
```

Registers Altered

- PC

Latency

1 cycle (if branch is not taken)

2 cycles (if branch is taken and the D bit is set)

3 cycles (if branch is taken and the D bit is not set)

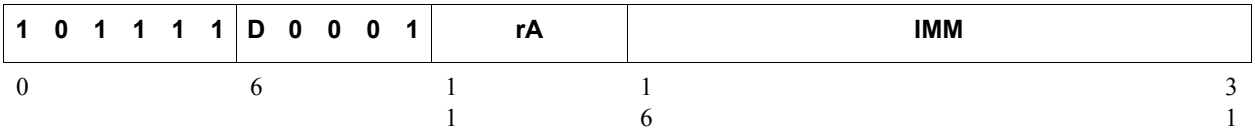
Note

A delay slot must not be used by the following: imm, branch, or break instructions. Interrupts and external hardware breaks are deferred until after the delay slot branch has been completed.

bnei

Branch Immediate if Not Equal

bnei	rA, IMM	Branch Immediate if Not Equal
bneid	rA, IMM	Branch Immediate if Not Equal with Delay



Description

Branch if rA not equal to 0, to the instruction located in the offset value of IMM. The target of the branch will be the instruction at address PC + IMM.

The mnemonic bneid will set the D bit. The D bit determines whether there is a branch delay slot or not. If the D bit is set, it means that there is a delay slot and the instruction following the branch (that is, in the branch delay slot) is allowed to complete execution before executing the target instruction. If the D bit is not set, it means that there is no delay slot, so the instruction to be executed after the branch is the target instruction.

Pseudocode

```
If rA ≠ 0 then
  PC ← PC + sext(IMM)
else
  PC ← PC + 4
if D = 1 then
  allow following instruction to complete execution
```

Registers Altered

- PC

Latency

- 1 cycle (if branch is not taken, or successful branch prediction occurs)
- 2 cycles (if branch is taken and the D bit is set)
- 3 cycles (if branch is taken and the D bit is not set, or a branch prediction mispredict occurs)

Note

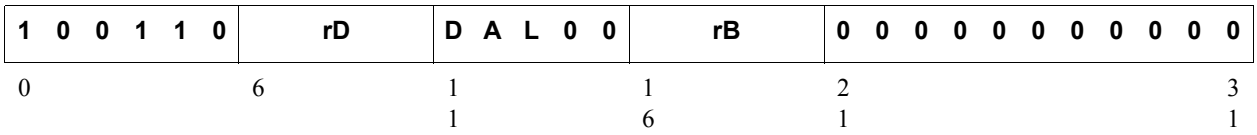
By default, Type B Instructions will take the 16-bit IMM field value and sign extend it to 32 bits to use as the immediate operand. This behavior can be overridden by preceding the Type B instruction with an imm instruction. See the instruction “imm,” page 180 for details on using 32-bit immediate values.

A delay slot must not be used by the following: imm, branch, or break instructions. Interrupts and external hardware breaks are deferred until after the delay slot branch has been completed.

br

Unconditional Branch

br	rB	Branch
bra	rB	Branch Absolute
brd	rB	Branch with Delay
brad	rB	Branch Absolute with Delay
brld	rD, rB	Branch and Link with Delay
brald	rD, rB	Branch Absolute and Link with Delay



Description

Branch to the instruction located at address determined by rB.

The mnemonics brld and brald will set the L bit. If the L bit is set, linking will be performed. The current value of PC will be stored in rD.

The mnemonics bra, brad and brald will set the A bit. If the A bit is set, it means that the branch is to an absolute value and the target is the value in rB, otherwise, it is a relative branch and the target will be PC + rB.

The mnemonics brd, brad, brld and brald will set the D bit. The D bit determines whether there is a branch delay slot or not. If the D bit is set, it means that there is a delay slot and the instruction following the branch (that is, in the branch delay slot) is allowed to complete execution before executing the target instruction.

If the D bit is not set, it means that there is no delay slot, so the instruction to be executed after the branch is the target instruction.

Pseudocode

```
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
```

Registers Altered

- rD
- PC

Latency

- 2 cycles (if the D bit is set)
- 3 cycles (if the D bit is not set)

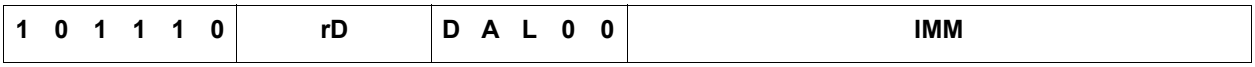
Note

The instructions brl and bral are not available. A delay slot must not be used by the following: imm, branch, or break instructions. Interrupts and external hardware breaks are deferred until after the delay slot branch has been completed.

bri

Unconditional Branch Immediate

bri	IMM	Branch Immediate
brai	IMM	Branch Absolute Immediate
brid	IMM	Branch Immediate with Delay
braid	IMM	Branch Absolute Immediate with Delay
brlid	rD, IMM	Branch and Link Immediate with Delay
bralid	rD, IMM	Branch Absolute and Link Immediate with Delay



Description

Branch to the instruction located at address determined by IMM, sign-extended to 32 bits.

The mnemonics brlid and bralid will set the L bit. If the L bit is set, linking will be performed. The current value of PC will be stored in rD.

The mnemonics brai, braid and bralid will set the A bit. If the A bit is set, it means that the branch is to an absolute value and the target is the value in IMM, otherwise, it is a relative branch and the target will be PC + IMM.

The mnemonics brid, braid, brlid and bralid will set the D bit. The D bit determines whether there is a branch delay slot or not. If the D bit is set, it means that there is a delay slot and the instruction following the branch (that is, in the branch delay slot) is allowed to complete execution before executing the target instruction. If the D bit is not set, it means that there is no delay slot, so the instruction to be executed after the branch is the target instruction.

As a special case, when MicroBlaze is configured to use an MMU (`C_USE_MMU` >= 1) and "`bralid rD, 0x8`"is used to perform a User Vector Exception, the Machine Status Register bits User Mode and Virtual Mode are cleared.

Pseudocode

```
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
```

Registers Altered

- rD
- PC
- MSR[UM], MSR[VM]

Latency

- 1 cycle (if successful branch prediction occurs)
- 2 cycles (if the D bit is set)
- 3 cycles (if the D bit is not set, or a branch prediction mispredict occurs)

Notes

The instructions brli and brali are not available.

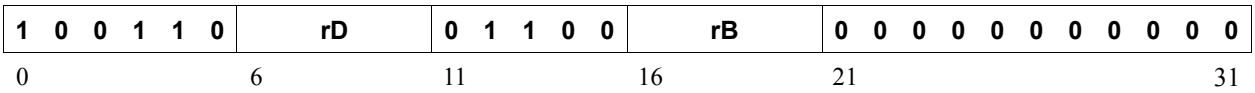
By default, Type B Instructions will take the 16-bit IMM field value and sign extend it to 32 bits to use as the immediate operand. This behavior can be overridden by preceding the Type B instruction with an imm instruction. See the instruction "imm," page 180 for details on using 32-bit immediate values.

A delay slot must not be used by the following: imm, branch, or break instructions. Interrupts and external hardware breaks are deferred until after the delay slot branch has been completed.

brk

Break

brk rD, rB



Description

Branch and link to the instruction located at address value in rB. The current value of PC will be stored in rD. The BIP flag in the MSR will be set, and the reservation bit will be cleared.

When MicroBlaze is configured to use an MMU (C_USE_MMU >= 1) this instruction is privileged. This means that if the instruction is attempted in User Mode (MSR[UM] = 1) a Privileged Instruction exception occurs.

Pseudocode

```
if MSR[UM] = 1 then
  ESR[EC] ← 00111
else
  (rD) ← PC
  PC ← (rB)
  MSR[BIP] ← 1
  Reservation ← 0
```

Registers Altered

- rD
- PC
- MSR[BIP]
- ESR[EC], in case a privileged instruction exception is generated

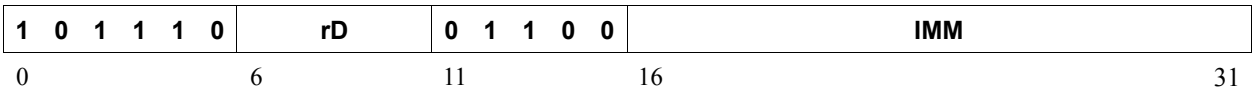
Latency

- 3 cycles

brki

Break Immediate

brki rD, IMM



Description

Branch and link to the instruction located at address value in IMM, sign-extended to 32 bits. The current value of PC will be stored in rD. The BIP flag in the MSR will be set, and the reservation bit will be cleared.

When MicroBlaze is configured to use an MMU (`C_USE_MMU` >= 1) this instruction is privileged, except as a special case when "`brki rD, 0x8`" or "`brki rD, 0x18`" is used to perform a Software Break. This means that, apart from the special case, if the instruction is attempted in User Mode (`MSR[UM]` = 1) a Privileged Instruction exception occurs.

As a special case, when MicroBlaze is configured to use an MMU (`C_USE_MMU` >= 1) and "`brki rD, 0x8`" or "`brki rD, 0x18`" is used to perform a Software Break, the Machine Status Register bits User Mode and Virtual Mode are cleared.

Pseudocode

```
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[UM] ← 0
    MSR[VMS] ← MSR[VM] MSR[VM] ← 0
```

Registers Altered

- rD, unless an exception is generated, in which case the register is unchanged
- PC
- MSR[BIP], MSR[UM], MSR[VM]
- ESR[EC], in case a privileged instruction exception is generated

Latency

- 3 cycles

Note

By default, Type B Instructions will take the 16-bit IMM field value and sign extend it to 32 bits to use as the immediate operand. This behavior can be overridden by preceding the Type B instruction with an imm instruction. See the instruction "imm," page 180 for details on using 32-bit immediate values.

As a special case, the imm instruction does not override a Software Break "`brki rD, 0x18`" when `C_USE_DEBUG` is set, to allow Software Break after an imm instruction.

bs

Barrel Shift

bsrl	rD, rA, rB	Barrel Shift Right Logical
bsra	rD, rA, rB	Barrel Shift Right Arithmetical
bsll	rD, rA, rB	Barrel Shift Left Logical

Description

Shifts the contents of register rA by the amount specified in register rB and puts the result in register rD.

The mnemonic bsll sets the S bit (Side bit). If the S bit is set, the barrel shift is done to the left. The mnemonics bsrl and bsra clear the S bit and the shift is done to the right.

The mnemonic bsra will set the T bit (Type bit). If the T bit is set, the barrel shift performed is Arithmetical. The mnemonics bsrl and bsll clear the T bit and the shift performed is Logical.

Pseudocode

```
if S = 1 then
  (rD) ← (rA) << (rB)[27:31]
else
  if T = 1 then
    if ((rB)[27:31]) ≠ 0 then
      (rD)[0:(rB)[27:31]-1] ← (rA)[0]
      (rD)[(rB)[27:31]:31] ← (rA) >> (rB)[27:31]
    else
      (rD) ← (rA)
  else
    (rD) ← (rA) >> (rB)[27:31]
```

Registers Altered

- rD

Latency

- 1 cycle with `C_AREA_OPTIMIZED=0`
- 2 cycles with `C_AREA_OPTIMIZED=1`

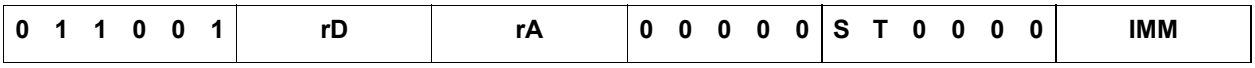
Note

These instructions are optional. To use them, MicroBlaze has to be configured to use barrel shift instructions (`C_USE_BARREL=1`).

bsi

Barrel Shift Immediate

bsrli	rD, rA, IMM	Barrel Shift Right Logical Immediate
bsrai	rD, rA, IMM	Barrel Shift Right Arithmetical Immediate
bslli	rD, rA, IMM	Barrel Shift Left Logical Immediate



Description

Shifts the contents of register rA by the amount specified by IMM and puts the result in register rD.

The mnemonic bsll sets the S bit (Side bit). If the S bit is set, the barrel shift is done to the left. The mnemonics bsrl and bsra clear the S bit and the shift is done to the right.

The mnemonic bsra will set the T bit (Type bit). If the T bit is set, the barrel shift performed is Arithmetical. The mnemonics bsrl and bsll clear the T bit and the shift performed is Logical.

Pseudocode

```
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
```

Registers Altered

- rD

Latency

- 1 cycle with `C_AREA_OPTIMIZED=0`
- 2 cycles with `C_AREA_OPTIMIZED=1`

Notes

These are not Type B Instructions. There is no effect from a preceding imm instruction.

These instructions are optional. To use them, MicroBlaze has to be configured to use barrel shift instructions (`C_USE_BARREL=1`).

clz

Count Leading Zeros

clz	rD, rA	Count leading zeros in rA

1	0	0	1	0	0	rD	rA	0	0	0	0	0	0	0	0	1	1	1	0	0	0	0	0
0						6	1 1	1 6					2 1										3 1

Description

This instruction counts the number of leading zeros in register rA starting from the most significant bit. The result is a number between 0 and 32, stored in register rD.

The result in rD is 32 when rA is 0, and it is 0 if rA is 0xFFFFFFFF.

Pseudocode

```
n ← 0
while (rA)[n] = 0
    n ←n + 1
(rD) ← n
```

Registers Altered

- rD

Latency

- 1 cycle

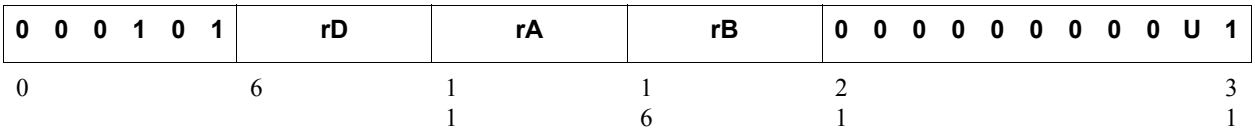
Notes

This instruction is only available when the parameter `C_USE_PCMP_INSTR` is set to 1.

cmp

Integer Compare

cmp	rD, rA, rB	compare rB with rA (signed)
cmpu	rD, rA, rB	compare rB with rA (unsigned)



Description

The contents of register rA is subtracted from the contents of register rB and the result is placed into register rD.

The MSB bit of rD is adjusted to shown true relation between rA and rB. If the U bit is set, rA and rB is considered unsigned values. If the U bit is clear, rA and rB is considered signed values.

Pseudocode

```
(rD) ← (rB) + (rA) + 1
(rD)(MSB) ← (rA) > (rB)
```

Registers Altered

- rD

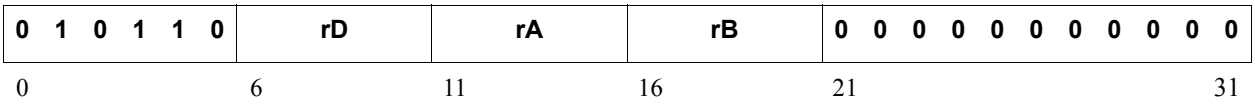
Latency

- 1 cycle

fadd

Floating Point Arithmetic Add

fadd	rD, rA, rB	Add



Description

The floating point sum of registers rA and rB, is placed into register rD.

Pseudocode

```
if isDnz(rA) or isDnz(rB) then
  (rD) ← 0xFFC00000
  FSR[DO] ← 1
  ESR[EC] ← 00110
else if isSigNaN(rA) or isSigNaN(rB)or
     (isPosInfinite(rA) and isNegInfinite(rB)) or
     (isNegInfinite(rA) and isPosInfinite(rB))) then
  (rD) ← 0xFFC00000
  FSR[IO] ← 1
  ESR[EC] ← 00110
else if isQuietNaN(rA) or isQuietNaN(rB) then
  (rD) ← 0xFFC00000
else if isDnz((rA)+(rB)) then
  (rD) ← signZero((rA)+(rB))
  FSR[UF] ← 1
  ESR[EC] ← 00110
else if isNaN((rA)+(rB)) then
  (rD) ← signInfinite((rA)+(rB))
  FSR[OF] ← 1
  ESR[EC] ← 00110
else
  (rD) ← (rA) + (rB)
```

Registers Altered

- rD, unless an FP exception is generated, in which case the register is unchanged
- ESR[EC], if an FP exception is generated
- FSR[IO,UF,OF,DO]

Latency

- 4 cycles with `C_AREA_OPTIMIZED=0`
- 6 cycles with `C_AREA_OPTIMIZED=1`

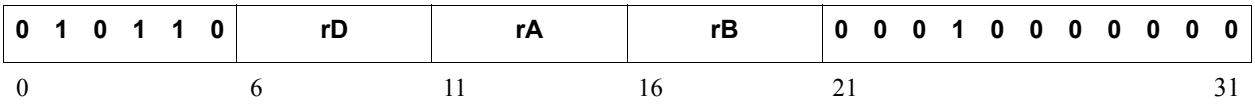
Note

This instruction is only available when the MicroBlaze parameter `C_USE_FPU` is greater than 0.

frsub

Reverse Floating Point Arithmetic Subtraction

frsub	rD, rA, rB	Reverse subtract



Description

The floating point value in rA is subtracted from the floating point value in rB and the result is placed into register rD.

Pseudocode

```
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)
```

Registers Altered

- rD, unless an FP exception is generated, in which case the register is unchanged
- ESR[EC], if an FP exception is generated
- FSR[IO,UF,OF,DO]

Latency

- 4 cycles with `C_AREA_OPTIMIZED=0`
- 6 cycles with `C_AREA_OPTIMIZED=1`

Note

This instruction is only available when the MicroBlaze parameter `C_USE_FPU` is greater than 0.

fmul

Floating Point Arithmetic Multiplication

fmul	rD, rA, rB	Multiply

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

Description

The floating point value in rA is multiplied with the floating point value in rB and the result is placed into register rD.

Pseudocode

```
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)
```

Registers Altered

- rD, unless an FP exception is generated, in which case the register is unchanged
- ESR[EC], if an FP exception is generated
- FSR[IO,UF,OF,DO]

Latency

- 4 cycles with `C_AREA_OPTIMIZED=0`
- 6 cycles with `C_AREA_OPTIMIZED=1`

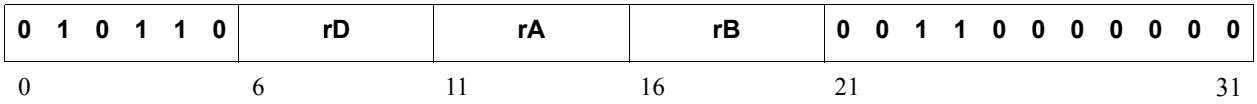
Note

This instruction is only available when the MicroBlaze parameter `C_USE_FPU` is greater than 0.

fdiv

Floating Point Arithmetic Division

fdiv rD, rA, rB Divide



Description

The floating point value in rB is divided by the floating point value in rA and the result is placed into register rD.

Pseudocode

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

Registers Altered

- rD, unless an FP exception is generated, in which case the register is unchanged
- ESR[EC], if an FP exception is generated
- FSR[IO,UF,OF,DO,DZ]

Latency

- 28 cycles with `C_AREA_OPTIMIZED=0`, 30 cycles with `C_AREA_OPTIMIZED=1`

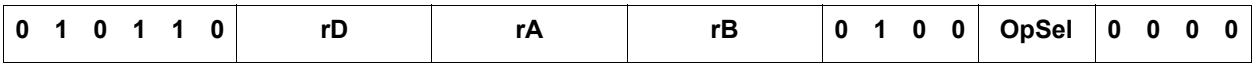
Note

This instruction is only available when the MicroBlaze parameter `C_USE_FPU` is greater than 0.

fcmp

Floating Point Number Comparison

fcmp.un	rD, rA, rB	Unordered floating point comparison
fcmp.lt	rD, rA, rB	Less-than floating point comparison
fcmp.eq	rD, rA, rB	Equal floating point comparison
fcmp.le	rD, rA, rB	Less-or-Equal floating point comparison
fcmp.gt	rD, rA, rB	Greater-than floating point comparison
fcmp.ne	rD, rA, rB	Not-Equal floating point comparison
fcmp.ge	rD, rA, rB	Greater-or-Equal floating point comparison



Description

The floating point value in rB is compared with the floating point value in rA and the comparison result is placed into register rD. The `OpSel` field in the instruction code determines the type of comparison performed.

Pseudocode

```
if isDnz(rA) or isDnz(rB) then
  (rD) ← 0
  FSR[DO] ← 1
  ESR[EC] ← 00110
else
  {read out behavior from Table 4-2}
```

Registers Altered

- rD, unless an FP exception is generated, in which case the register is unchanged
- ESR[EC], if an FP exception is generated
- FSR[IO,DO]

Latency

- 1 cycle with `C_AREA_OPTIMIZED=0`
- 3 cycles with `C_AREA_OPTIMIZED=1`

Note

These instructions are only available when the MicroBlaze parameter `C_USE_FPU` is greater than 0.

Table 4-2, page 171 lists the floating point comparison operations.

Table 4-2: **Floating Point Comparison Operation**

Comparison Type		Operand Relationship				
Description	**OpSel**	**(rB) > (rA)**	**(rB) < (rA)**	**(rB) = (rA)**	**isSigNaN(rA) or isSigNaN(rB)**	**isQuietNaN(rA) or isQuietNaN(rB)**
Unordered	000	(rD) ← 0	(rD) ← 0	(rD) ← 0	(rD) ← 1 FSR[IO] ← 1 ESR[EC] ← 00110	(rD) ← 1
Less-than	001	(rD) ← 0	(rD) ← 1	(rD) ← 0	(rD) ← 0 FSR[IO] ← 1 ESR[EC] ← 00110	(rD) ← 0 FSR[IO] ← 1 ESR[EC] ← 00110
Equal	010	(rD) ← 0	(rD) ← 0	(rD) ← 1	(rD) ← 0 FSR[IO] ← 1 ESR[EC] ← 00110	(rD) ← 0
Less-or-equal	011	(rD) ← 0	(rD) ← 1	(rD) ← 1	(rD) ← 0 FSR[IO] ← 1 ESR[EC] ← 00110	(rD) ← 0 FSR[IO] ← 1 ESR[EC] ← 00110
Greater-than	100	(rD) ← 1	(rD) ← 0	(rD) ← 0	(rD) ← 0 FSR[IO] ← 1 ESR[EC] ← 00110	(rD) ← 0 FSR[IO] ← 1 ESR[EC] ← 00110
Not-equal	101	(rD) ← 1	(rD) ← 1	(rD) ← 0	(rD) ← 1 FSR[IO] ← 1 ESR[EC] ← 00110	(rD) ← 1
Greater-or-equal	110	(rD) ← 1	(rD) ← 0	(rD) ← 1	(rD) ← 0 FSR[IO] ← 1 ESR[EC] ← 00110	(rD) ← 0 FSR[IO] ← 1 ESR[EC] ← 00110

flt

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
0	6	11	16	21 31

Description

Converts the signed integer in register rA to floating point and puts the result in register rD. This is a 32-bit rounding signed conversion that will produce a 32-bit floating point result.

Pseudocode

```
(rD) ← float ((rA))
```

Registers Altered

- rD

Latency

- 4 cycles with `C_AREA_OPTIMIZED=0`
- 6 cycles with `C_AREA_OPTIMIZED=1`

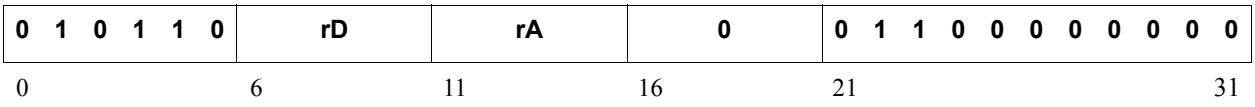
Note

This instruction is only available when the MicroBlaze parameter `C_USE_FPU` is set to 2.

fint

Floating Point Convert Float to Integer

fint rD, rA



Description

Converts the floating point number in register rA to a signed integer and puts the result in register rD. This is a 32-bit signed conversion that will produce a 32-bit integer result.

Pseudocode

```
if isDnz(rA) then
  (rD) ← 0xFFC00000
  FSR[DO] ← 1
  ESR[EC] ← 00110
else if isNaN(rA) then
  (rD) ← 0xFFC00000
  FSR[IO] ← 1
  ESR[EC] ← 00110
else if isInf(rA) or (rA) < -2^31 or (rA) > 2^31 - 1 then
  (rD) ← 0xFFC00000
  FSR[IO] ← 1
  ESR[EC] ← 00110
else
  (rD) ← int ((rA))
```

Registers Altered

- rD, unless an FP exception is generated, in which case the register is unchanged
- ESR[EC], if an FP exception is generated
- FSR[IO,DO]

Latency

- 5 cycles with `C_AREA_OPTIMIZED=0`
- 7 cycles with `C_AREA_OPTIMIZED=1`

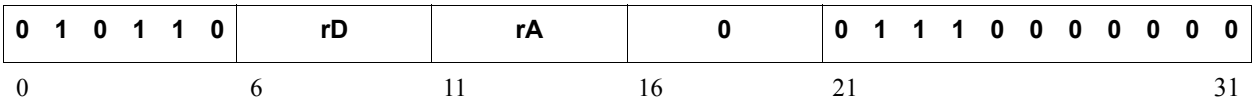
Note

This instruction is only available when the MicroBlaze parameter `C_USE_FPU` is set to 2.

fsqrt

Floating Point Arithmetic Square Root

fsqrt	rD, rA	Square Root



Description

Performs a floating point square root on the value in rA and puts the result in register rD.

Pseudocode

```
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))
```

Registers Altered

- rD, unless an FP exception is generated, in which case the register is unchanged
- ESR[EC], if an FP exception is generated
- FSR[IO,DO]

Latency

- 27 cycles with `C_AREA_OPTIMIZED=0`
- 29 cycles with `C_AREA_OPTIMIZED=1`

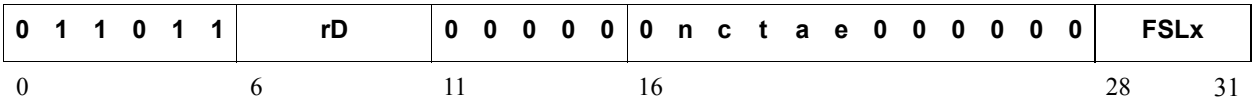
Note

This instruction is only available when the MicroBlaze parameter `C_USE_FPU` is set to 2.

get

get from stream interface

*tnea*get	rD, FSLx	get data from link x *t* = test-only *n* = non-blocking *e* = exception if control bit set *a* = atomic
*tneca*get	rD, FSLx	get control from link x *t* = test-only *n* = non-blocking *e* = exception if control bit not set *a* = atomic



Description

MicroBlaze will read from the link x interface and place the result in register rD.

The get instruction has 32 variants.

The blocking versions (when ‘n’ bit is ‘0’) will stall MicroBlaze until the data from the interface is valid. The non-blocking versions will not stall micro blaze and will set carry to ‘0’ if the data was valid and to ‘1’ if the data was invalid. In case of an invalid access the destination register contents is undefined.

All data get instructions (when ‘c’ bit is ‘0’) expect the control bit from the interface to be ‘0’. If this is not the case, the instruction will set MSR[FSL_Error] to ‘1’. All control get instructions (when ‘c’ bit is ‘1’) expect the control bit from the interface to be ‘1’. If this is not the case, the instruction will set MSR[FSL_Error] to ‘1’.

The exception versions (when ‘e’ bit is ‘1’) will generate an exception if there is a control bit mismatch. In this case ESR is updated with EC set to the exception cause and ESS set to the link index. The target register, rD, is not updated when an exception is generated, instead the data is stored in EDR.

The test versions (when ‘t’ bit is ‘1’) will be handled as the normal case, except that the read signal to the link is not asserted.

Atomic versions (when ‘a’ bit is ‘1’) are not interruptible. This means that a sequence of atomic instructions can be grouped together without an interrupt breaking the program flow. However, note that exceptions may still occur.

When MicroBlaze is configured to use an MMU (`C_USE_MMU` >= 1) and not explicitly allowed by setting `C_MMU_PRIVILEGED_INSTR` to 1 these instructions are privileged. This means that if these instructions are attempted in User Mode (`MSR[UM]` =1) a Privileged Instruction exception occurs.

Pseudocode

```
if MSR[UM] = 1 then
  ESR[EC] ← 00111
else
  (rD) ← FSLx_S_DATA | Sx_AXIS_TDATA
  if (n = 1) then
    MSR[Carry]  ← (FSLx_S_EXISTS | Sx_AXIS_TVALID)
  if (FSLx_S_CONTROL | Sx_AXIS_TLAST ≠ c) and
      (FSLx_S_EXISTS | Sx_AXIS_TVALID) then
    MSR[FSL_Error] ← 1
    if (e = 1) then
      ESR[EC]  ← 00000
      ESR[ESS] ← instruction bits [28:31]
      EDR      ← FSLx_S_DATA | Sx_AXIS_TDATA
```

Registers Altered

- rD, unless an exception is generated, in which case the register is unchanged
- MSR[FSL_Error]
- MSR[Carry]
- ESR[EC], in case a stream exception or a privileged instruction exception is generated
- ESR[ESS], in case a stream exception is generated
- EDR, in case a stream exception is generated

Latency

- 1 cycle with C_AREA_OPTIMIZED=0
- 2 cycles with C_AREA_OPTIMIZED=1

The blocking versions of this instruction will stall the pipeline of MicroBlaze until the instruction can be completed. Interrupts are served when the parameter C_USE_EXTENDED_FSL_INSTR is set to 1, and the instruction is not atomic.

Note

To refer to an FSLx interface in assembly language, use rfsl0, rfsl1, ... rfsl15.

The blocking versions of this instruction should not be placed in a delay slot when the parameter C_USE_EXTENDED_FSL_INSTR is set to 1, since this prevents interrupts from being served.

For non-blocking versions, an rsubc instruction can be used to decrement an index variable.

The ‘e’ bit does not have any effect unless C_FSL_EXCEPTION is set to 1.

These instructions are only available when the MicroBlaze parameter C_FSL_LINKS is greater than 0.

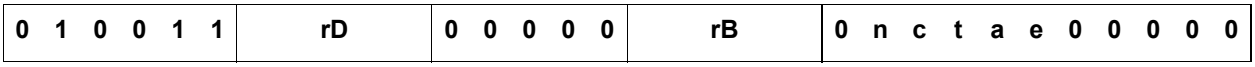
The extended instructions (exception, test and atomic versions) are only available when the MicroBlaze parameter C_USE_EXTENDED_FSL_INSTR is set to 1.

It is not recommended to allow these instructions in user mode, unless absolutely necessary for performance reasons, since that removes all hardware protection preventing incorrect use of a link.

getd

get from stream interface dynamic

*tnea*getd	rD, rB	get data from link rB[28:31] *t* = test-only *n* = non-blocking *e* = exception if control bit set *a* = atomic
*tneca*getd	rD, rB	get control from link rB[28:31] *t* = test-only *n* = non-blocking *e* = exception if control bit not set *a* = atomic



Description

MicroBlaze will read from the interface defined by the four least significant bits in rB and place the result in register rD.

The getd instruction has 32 variants.

The blocking versions (when ‘n’ bit is ‘0’) will stall MicroBlaze until the data from the interface is valid. The non-blocking versions will not stall micro blaze and will set carry to ‘0’ if the data was valid and to ‘1’ if the data was invalid. In case of an invalid access the destination register contents is undefined.

All data get instructions (when ‘c’ bit is ‘0’) expect the control bit from the interface to be ‘0’. If this is not the case, the instruction will set MSR[FSL_Error] to ‘1’. All control get instructions (when ‘c’ bit is ‘1’) expect the control bit from the interface to be ‘1’. If this is not the case, the instruction will set MSR[FSL_Error] to ‘1’.

The exception versions (when ‘e’ bit is ‘1’) will generate an exception if there is a control bit mismatch. In this case ESR is updated with EC set to the exception cause and ESS set to the link index. The target register, rD, is not updated when an exception is generated, instead the data is stored in EDR.

The test versions (when ‘t’ bit is ‘1’) will be handled as the normal case, except that the read signal to the link is not asserted.

Atomic versions (when ‘a’ bit is ‘1’) are not interruptible. This means that a sequence of atomic instructions can be grouped together without an interrupt breaking the program flow. However, note that exceptions may still occur.

When MicroBlaze is configured to use an MMU (`C_USE_MMU >= 1`) and not explicitly allowed by setting `C_MMU_PRIVILEGED_INSTR` to 1 these instructions are privileged. This means that if these instructions are attempted in User Mode (MSR[UM] = 1) a Privileged Instruction exception occurs.

Pseudocode

```
if MSR[UM] = 1 then
  ESR[EC] ← 00111
else
  x ← rB[28:31]
  (rD) ← FSLx_S_DATA | Sx_AXIS_TDATA
  if (n = 1) then
    MSR[Carry]  ← (FSLx_S_EXISTS | Sx_AXIS_TVALID)
  if  (FSLx_S_CONTROL | Sx_AXIS_TLAST ≠ c) and
      (FSLx_S_EXISTS | Sx_AXIS_TVALID) then
    MSR[FSL_Error] ← 1
    if (e = 1) then
      ESR[EC]  ← 00000
      ESR[ESS] ← rB[28:31]
      EDR      ← FSLx_S_DATA | Sx_AXIS_TDATA
```

Registers Altered

- rD, unless an exception is generated, in which case the register is unchanged
- MSR[FSL_Error]
- MSR[Carry]
- ESR[EC], in case a stream exception or a privileged instruction exception is generated
- ESR[ESS], in case a stream exception is generated
- EDR, in case a stream exception is generated

Latency

- 1 cycle with `C_AREA_OPTIMIZED=0`
- 2 cycles with `C_AREA_OPTIMIZED=1`

The blocking versions of this instruction will stall the pipeline of MicroBlaze until the instruction can be completed. Interrupts are served unless the instruction is atomic, which ensures that the instruction cannot be interrupted.

Note

The blocking versions of this instruction should not be placed in a delay slot, since this prevents interrupts from being served.

For non-blocking versions, an rsubc instruction can be used to decrement an index variable.

The ‘e’ bit does not have any effect unless `C_FSL_EXCEPTION` is set to 1.

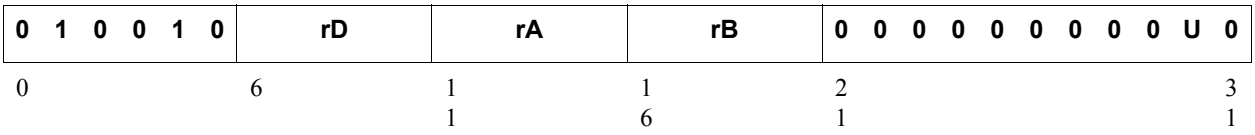
These instructions are only available when the MicroBlaze parameter `C_FSL_LINKS` is greater than 0 and the parameter `C_USE_EXTENDED_FSL_INSTR` is set to 1.

It is not recommended to allow these instructions in user mode, unless absolutely necessary for performance reasons, since that removes all hardware protection preventing incorrect use of a link.

idiv

Integer Divide

idiv	rD, rA, rB	divide rB by rA (signed)
idivu	rD, rA, rB	divide rB by rA (unsigned)



Description

The contents of register rB is divided by the contents of register rA and the result is placed into register rD.

If the U bit is set, rA and rB are considered unsigned values. If the U bit is clear, rA and rB are considered signed values.

If the value of rA is 0, the DZO bit in MSR will be set and the value in rD will be 0, unless an exception is generated.

If the U bit is clear, the value of rA is -1, and the value of rB is -2147483648, the DZO bit in MSR will be set and the value in rD will be -2147483648, unless an exception is generated.

Pseudocode

```
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)
```

Registers Altered

- rD, unless a divide exception is generated, in which case the register is unchanged
- MSR[DZO], if the value in rA is zero
- ESR[EC], if the value in rA is zero

Latency

- 1 cycle if (rA) = 0, otherwise 32 cycles with `C_AREA_OPTIMIZED=0`
- 1 cycle if (rA) = 0, otherwise 34 cycles with `C_AREA_OPTIMIZED=1`

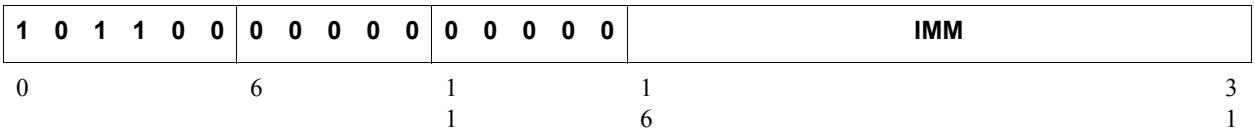
Note

This instruction is only valid if MicroBlaze is configured to use a hardware divider (`C_USE_DIV = 1`).

imm

Immediate

imm IMM



Description

The instruction imm loads the IMM value into a temporary register. It also locks this value so it can be used by the following instruction and form a 32-bit immediate value.

The instruction imm is used in conjunction with Type B instructions. Since Type B instructions have only a 16-bit immediate value field, a 32-bit immediate value cannot be used directly. However, 32-bit immediate values can be used in MicroBlaze. By default, Type B Instructions will take the 16-bit IMM field value and sign extend it to 32 bits to use as the immediate operand. This behavior can be overridden by preceding the Type B instruction with an imm instruction. The imm instruction locks the 16-bit IMM value temporarily for the next instruction. A Type B instruction that immediately follows the imm instruction will then form a 32-bit immediate value from the 16-bit IMM value of the imm instruction (upper 16 bits) and its own 16-bit immediate value field (lower 16 bits). If no Type B instruction follows the imm instruction, the locked value gets unlocked and becomes useless.

Latency

- 1 cycle

Notes

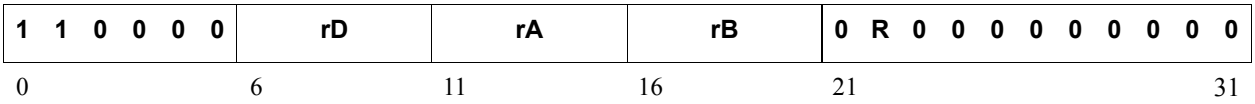
The imm instruction and the Type B instruction following it are atomic; consequently, no interrupts are allowed between them.

The assembler provided by Xilinx automatically detects the need for imm instructions. When a 32-bit IMM value is specified in a Type B instruction, the assembler converts the IMM value to a 16-bit one to assemble the instruction and inserts an imm instruction before it in the executable file.

lbu

Load Byte Unsigned

lbu	rD, rA, rB
lbur	rD, rA, rB



Description

Loads a byte (8 bits) from the memory location that results from adding the contents of registers rA and rB. The data is placed in the least significant byte of register rD and the other three bytes in rD are cleared.

If the R bit is set, a byte reversed memory location is used, loading data with the opposite endianness of the endianness defined by `C_ENDIANNESS` and the E bit (if virtual protected mode is enabled).

A data TLB miss exception occurs if virtual protected mode is enabled, and a valid translation entry corresponding to the address is not found in the TLB.

A data storage exception occurs if access is prevented by a no-access-allowed zone protection. This only applies to accesses with user mode and virtual protected mode enabled.

Pseudocode

```
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
```

Registers Altered

- rD, unless an exception is generated, in which case the register is unchanged
- MSR[UM], MSR[VM], MSR[UMS], MSR[VMS], if an exception is generated
- ESR[EC], ESR[S], if an exception is generated
- ESR[DIZ], if a data storage exception is generated

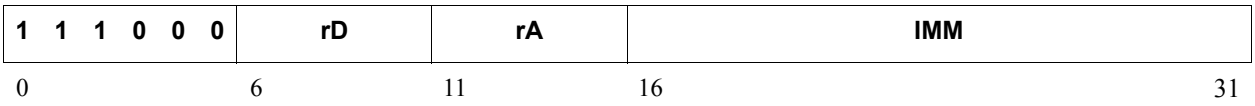
Latency

- 1 cycle with `C_AREA_OPTIMIZED=0`
- 2 cycles with `C_AREA_OPTIMIZED=1`

lbui

Load Byte Unsigned Immediate

lbui rD, rA, IMM



Description

Loads a byte (8 bits) from the memory location that results from adding the contents of register rA with the value in IMM, sign-extended to 32 bits. The data is placed in the least significant byte of register rD and the other three bytes in rD are cleared.

A data TLB miss exception occurs if virtual protected mode is enabled, and a valid translation entry corresponding to the address is not found in the TLB.

A data storage exception occurs if access is prevented by a no-access-allowed zone protection. This only applies to accesses with user mode and virtual protected mode enabled.

Pseudocode

```
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
```

Registers Altered

- rD, unless an exception is generated, in which case the register is unchanged
- MSR[UM], MSR[VM], MSR[UMS], MSR[VMS], if an exception is generated
- ESR[EC], ESR[S], if an exception is generated
- ESR[DIZ], if a data storage exception is generated

Latency

- 1 cycle with `C_AREA_OPTIMIZED=0`
- 2 cycles with `C_AREA_OPTIMIZED=1`

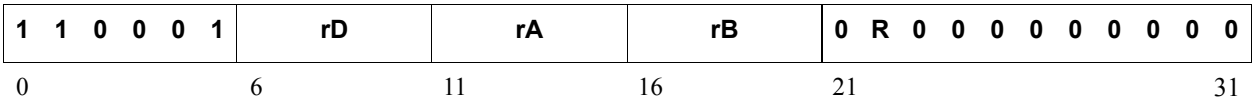
Note

By default, Type B Instructions will take the 16-bit IMM field value and sign extend it to 32 bits to use as the immediate operand. This behavior can be overridden by preceding the Type B instruction with an imm instruction. See the instruction "imm," page 180 for details on using 32-bit immediate values.

lhu

Load Halfword Unsigned

lhu	rD, rA, rB
lhur	rD, rA, rB



Description

Loads a halfword (16 bits) from the halfword aligned memory location that results from adding the contents of registers rA and rB. The data is placed in the least significant halfword of register rD and the most significant halfword in rD is cleared.

If the R bit is set, a halfword reversed memory location is used and the two bytes in the halfword are reversed, loading data with the opposite endianness of the endianness defined by `C_ENDIANNESS` and the E bit (if virtual protected mode is enabled).

A data TLB miss exception occurs if virtual protected mode is enabled, and a valid translation entry corresponding to the address is not found in the TLB.

A data storage exception occurs if access is prevented by a no-access-allowed zone protection. This only applies to accesses with user mode and virtual protected mode enabled.

An unaligned data access exception occurs if the least significant bit in the address is not zero.

Pseudocode

```
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
```

Registers Altered

- rD, unless an exception is generated, in which case the register is unchanged
- MSR[UM], MSR[VM], MSR[UMS], MSR[VMS], if an exception is generated
- ESR[EC], ESR[S], if an exception is generated
- ESR[DIZ], if a data storage exception is generated
- ESR[W], ESR[Rx], if an unaligned data access exception is generated

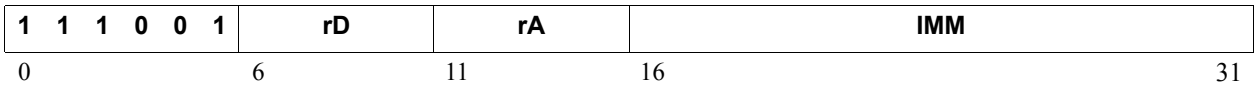
Latency

- 1 cycle with `C_AREA_OPTIMIZED=0`
- 2 cycles with `C_AREA_OPTIMIZED=1`

lhui

Load Halfword Unsigned Immediate

lhui rD, rA, IMM



Description

Loads a halfword (16 bits) from the halfword aligned memory location that results from adding the contents of register rA and the value in IMM, sign-extended to 32 bits. The data is placed in the least significant halfword of register rD and the most significant halfword in rD is cleared.

A data TLB miss exception occurs if virtual protected mode is enabled, and a valid translation entry corresponding to the address is not found in the TLB. A data storage exception occurs if access is prevented by a no-access-allowed zone protection. This only applies to accesses with user mode and virtual protected mode enabled. An unaligned data access exception occurs if the least significant bit in the address is not zero.

Pseudocode

```
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
```

Registers Altered

- rD, unless an exception is generated, in which case the register is unchanged
- MSR[UM], MSR[VM], MSR[UMS], MSR[VMS], if a TLB miss exception or a data storage exception is generated
- ESR[EC], ESR[S], if an exception is generated
- ESR[DIZ], if a data storage exception is generated
- ESR[W], ESR[Rx], if an unaligned data access exception is generated

Latency

- 1 cycle with `C_AREA_OPTIMIZED=0`
- 2 cycles with `C_AREA_OPTIMIZED=1`

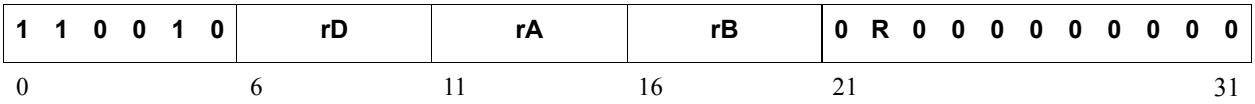
Note

By default, Type B Instructions will take the 16-bit IMM field value and sign extend it to 32 bits to use as the immediate operand. This behavior can be overridden by preceding the Type B instruction with an imm instruction. See the instruction “imm,” page 180 for details on using 32-bit immediate values.

lw

Load Word

lw	rD, rA, rB
lwr	rD, rA, rB



Description

Loads a word (32 bits) from the word aligned memory location that results from adding the contents of registers rA and rB. The data is placed in register rD.

If the R bit is set, the bytes in the loaded word are reversed , loading data with the opposite endianness of the endianness defined by C_ENDIANNESS and the E bit (if virtual protected mode is enabled).

A data TLB miss exception occurs if virtual protected mode is enabled, and a valid translation entry corresponding to the address is not found in the TLB.

A data storage exception occurs if access is prevented by a no-access-allowed zone protection. This only applies to accesses with user mode and virtual protected mode enabled.

An unaligned data access exception occurs if the two least significant bits in the address are not zero.

Pseudocode

```
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)
```

Registers Altered

- rD, unless an exception is generated, in which case the register is unchanged
- MSR[UM], MSR[VM], MSR[UMS], MSR[VMS], if a TLB miss exception or a data storage exception is generated
- ESR[EC], ESR[S], if an exception is generated
- ESR[DIZ], if a data storage exception is generated
- ESR[W], ESR[Rx], if an unaligned data access exception is generated

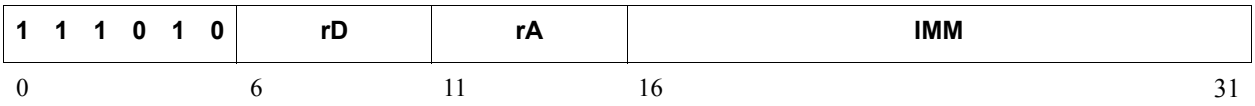
Latency

- 1 cycle with C_AREA_OPTIMIZED=0
- 2 cycles with C_AREA_OPTIMIZED=1

lwi

Load Word Immediate

lwi rD, rA, IMM



Description

Loads a word (32 bits) from the word aligned memory location that results from adding the contents of register rA and the value IMM, sign-extended to 32 bits. The data is placed in register rD. A data TLB miss exception occurs if virtual protected mode is enabled, and a valid translation entry corresponding to the address is not found in the TLB.A data storage exception occurs if access is prevented by a no-access-allowed zone protection. This only applies to accesses with user mode and virtual protected mode enabled. An unaligned data access exception occurs if the two least significant bits in the address are not zero.

Pseudocode

```
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)
```

Registers Altered

- rD, unless an exception is generated, in which case the register is unchanged
- MSR[UM], MSR[VM], MSR[UMS], MSR[VMS], if a TLB miss exception or a data storage exception is generated
- ESR[EC], ESR[S], if an exception is generated
- ESR[DIZ], if a data storage exception is generated
- ESR[W], ESR[Rx], if an unaligned data access exception is generated

Latency

- 1 cycle with `C_AREA_OPTIMIZED=0`
- 2 cycles with `C_AREA_OPTIMIZED=1`

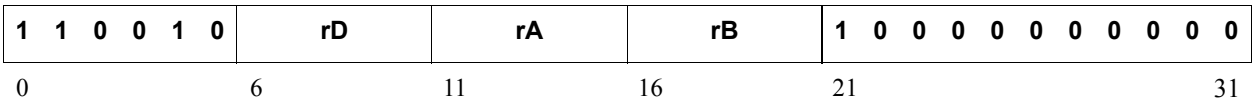
Note

By default, Type B Instructions will take the 16-bit IMM field value and sign extend it to 32 bits to use as the immediate operand. This behavior can be overridden by preceding the Type B instruction with an imm instruction. See the instruction “imm,” page 180 for details on using 32-bit immediate values.

lwx

Load Word Exclusive

lwx rD, rA, rB



Description

Loads a word (32 bits) from the word aligned memory location that results from adding the contents of registers rA and rB. The data is placed in register rD, and the reservation bit is set. If an AXI4 interconnect with exclusive access enabled is used, and the interconnect response is not EXOKAY, the carry flag (MSR[C]) is set; otherwise the carry flag is cleared.

A data TLB miss exception occurs if virtual protected mode is enabled, and a valid translation entry corresponding to the address is not found in the TLB.

A data storage exception occurs if access is prevented by a no-access-allowed zone protection. This only applies to accesses with user mode and virtual protected mode enabled.

An unaligned data access exception will *not* occur, even if the two least significant bits in the address are not zero.

A data bus exception can occur when an AXI4 interconnect with exclusive access enabled is used, and the interconnect response is not EXOKAY, which means that an exclusive access cannot be handled.

Enabling AXI exclusive access ensures that the operation is protected from other bus masters, but requires that the addressed slave supports exclusive access. When exclusive access is not enabled, only the internal reservation bit is used. Exclusive access is enabled using the two parameters `C_M_AXI_DP_EXCLUSIVE_ACCESS` and `C_M_AXI_DC_EXCLUSIVE_ACCESS` for the peripheral and cache interconnect, respectively.

Pseudocode

```
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 AXI_Exclusive_Used(Addr) && AXI_Response /= EXOKAY then
  ESR[EC] ← 00100;ESR[ECC]← 0;
  MSR[UMS] ← MSR[UM]; MSR[VMS] ← MSR[VM]; MSR[UM] ← 0; MSR[VM] ← 0
else
  (rD) ← Mem(Addr); Reservation ← 1;
  if AXI_Exclusive_Used(Addr) && AXI_Response /= EXOKAY then
    MSR[C] ← 1
  else
    MSR[C] ← 0
```

Registers Altered

- rD and MSR[C], unless an exception is generated, in which case they are unchanged
- MSR[UM], MSR[VM], MSR[UMS], MSR[VMS], if a TLB miss exception or a data storage exception is generated
- ESR[EC], ESR[S], if an exception is generated
- ESR[DIZ], if a data storage exception is generated

Latency

- 1 cycle with `C_AREA_OPTIMIZED=0`
- 2 cycles with `C_AREA_OPTIMIZED=1`

Note

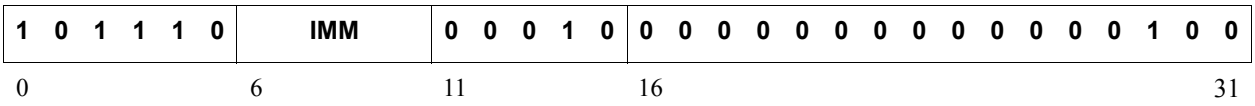
This instruction is used together with SWX to implement exclusive access, such as semaphores and spinlocks.

The carry flag (MSR[C]) may not be set immediately (dependent on pipeline stall behavior). The LWX instruction should not be immediately followed by an SRC instruction, to ensure the correct value of the carry flag is obtained.

mbar

Memory Barrier

mbar	IMM	Memory Barrier



Description

This instruction ensures that outstanding memory accesses on memory interfaces are completed before any subsequent instructions are executed. This is necessary to guarantee that self-modifying code is handled correctly, and that a DMA transfer can be safely started.

With self-modifying code, it is necessary to wait for both instruction reads and data accesses, which can be done by setting IMM to 0. In this case, the instruction also clears the Branch Target Cache, and empties the instruction prefetch buffer.

To ensure that data to be read by a DMA unit has been written to memory, it is only necessary to wait for data accesses, which can be done by setting IMM to 1.

Pseudocode

```
if (IMM & 1) = 0 then
  wait for instruction side memory accesses
if (IMM & 2) = 0 then
  wait for data side memory accesses
PC ← PC + 4
```

Registers Altered

- PC

Latency

- 1 + N cycles, where N is the number of cycles to wait for memory accesses to complete

Notes

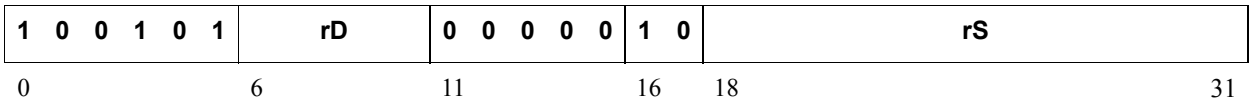
This instruction must not be preceded by an imm instruction.

With XCL, there is no way for this instruction to know when data writes are complete. Hence it is also necessary to read back the last written value in this case, to ensure that the access has completed.

mfs

Move From Special Purpose Register

mfs rD, rS



Description

Copies the contents of the special purpose register rS into register rD. The special purpose registers TLBLO and TLBHI are used to copy the contents of the Unified TLB entry indexed by TLBX.

Pseudocode

```
switch (rS):
case 0x0000 : (rD) ← PC
case 0x0001 : (rD) ← MSR
case 0x0003 : (rD) ← EAR
case 0x0005 : (rD) ← ESR
case 0x0007 : (rD) ← FSR
case 0x000B : (rD) ← BTR
case 0x000D : (rD) ← EDR
case 0x0800 : (rD) ← SLR
case 0x0802 : (rD) ← SHR
case 0x1000 : (rD) ← PID
case 0x1001 : (rD) ← ZPR
case 0x1002 : (rD) ← TLBX
case 0x1003 : (rD) ← TLBLO
case 0x1004 : (rD) ← TLBHI
case 0x200x : (rD) ← PVR[x] (where x = 0 to 11)
default : (rD) ← Undefined
```

Registers Altered

- rD

Latency

- 1 cycle

Notes

To refer to special purpose registers in assembly language, use rpc for PC, rmsr for MSR, rear for EAR, resr for ESR, rfsr for FSR, rbtr for BTR, redr for EDR, rslr for SLR, rshr for SHR, rpid for PID, rzpr for ZPR, rtlblo for TLBLO, rtlbhi for TLBHI, rtlbx for TLBX, and rpvr0 - rpvr11 for PVR0 - PVR11.

The value read from MSR may not include effects of the immediately preceding instruction (dependent on pipeline stall behavior). An instruction that does not affect MSR must precede the MFS instruction to guarantee correct MSR value.

The value read from FSR may not include effects of the immediately preceding instruction (dependent on pipeline stall behavior). An instruction that does not affect FSR must precede the MFS instruction to guarantee correct FSR value.

EAR, ESR and BTR are only valid as operands when at least one of the MicroBlaze C_*_EXCEPTION parameters are set to 1.

EDR is only valid as operand when the parameter C_FSL_EXCEPTION is set to 1 and the parameter C_FSL_LINKS is greater than 0.

FSR is only valid as an operand when the C_USE_FPU parameter is greater than 0.

SLR and SHR are only valid as an operand when the C_USE_STACK_PROTECTION parameter is set to 1.

PID, ZPR, TLBLO and TLBHI are only valid as operands when the parameter C_USE_MMU > 1 and the parameter C_MMU_TLB_ACCESS = 1 or 3.

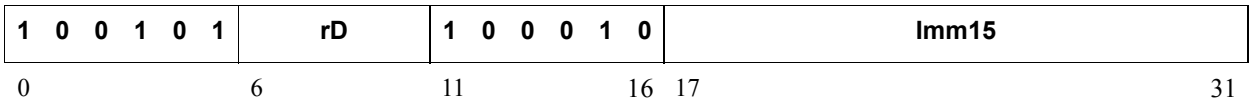
TLBX is only valid as operand when the parameter C_USE_MMU > 1 and the parameter C_MMU_TLB_ACCESS > 0.

PVR0 is only valid as an operand when C_PVR is 1 or 2, and PVR1 - PVR11 are only valid as operands when C_PVR is set to 2.

msrclr

Read MSR and clear bits in MSR

msrclr rD, Imm



Description

Copies the contents of the special purpose register MSR into register rD.
Bit positions in the IMM value that are 1 are cleared in the MSR. Bit positions that are 0 in the IMM value are left untouched.

When MicroBlaze is configured to use an MMU (`C_USE_MMU >= 1`) this instruction is privileged for all IMM values except those only affecting C. This means that if the instruction is attempted in User Mode (`MSR[UM] = 1`) in this case a Privileged Instruction exception occurs.

Pseudocode

```
if MSR[UM] = 1 and IMM ≠ 0x4 then
  ESR[EC] ← 00111
else
  (rD)   ← (MSR)
  (MSR)  ← (MSR) ∧ (~IMM)
```

Registers Altered

- rD
- MSR
- ESR[EC], in case a privileged instruction exception is generated

Latency

- 1 cycle

Notes

MSRCLR will affect the Carry bit immediately while the remaining bits will take effect one cycle after the instruction has been executed. When clearing the IE bit, it is guaranteed that the processor will not react to any interrupt for the subsequent instructions.

The value read from MSR may not include effects of the immediately preceding instruction (dependent on pipeline stall behavior). An instruction that does not affect MSR must precede the MSRCLR instruction to guarantee correct MSR value.

The immediate values has to be less than 2^{15} when `C_USE_MMU >= 1`, and less than 2^{14} otherwise. Only bits 17 to 31 of the MSR can be cleared when `C_USE_MMU >= 1`, and.bits 18 to 31 otherwise.

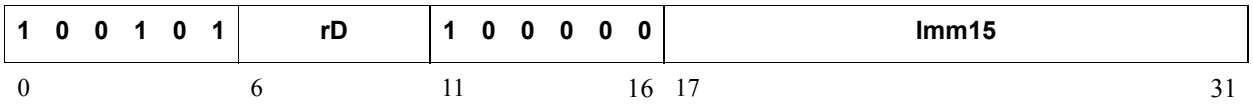
This instruction is only available when the parameter `C_USE_MSR_INSTR` is set to 1.

When clearing MSR[VM] the instruction must always be followed by a synchronizing branch instruction, for example BRI 4.

msrset

Read MSR and set bits in MSR

msrset rD, Imm



Description

Copies the contents of the special purpose register MSR into register rD.
Bit positions in the IMM value that are 1 are set in the MSR. Bit positions that are 0 in the IMM value are left untouched.

When MicroBlaze is configured to use an MMU (`C_USE_MMU` >= 1) this instruction is privileged for all IMM values except those only affecting C. This means that if the instruction is attempted in User Mode (`MSR[UM]` = 1) in this case a Privileged Instruction exception occurs.

Pseudocode

```
if MSR[UM] = 1 and IMM ≠ 0x4 then
  ESR[EC]← 00111
else
  (rD)   ← (MSR)
  (MSR)  ← (MSR) ∨ (IMM)
```

Registers Altered

- rD
- MSR
- ESR[EC], in case a privileged instruction exception is generated

Latency

- 1 cycle

Notes

MSRSET will affect the Carry bit immediately while the remaining bits will take effect one cycle after the instruction has been executed. When setting the EIP or BIP bit, it is guaranteed that the processor will not react to any interrupt or normal hardware break for the subsequent instructions.

The value read from MSR may not include effects of the immediately preceding instruction (dependent on pipeline stall behavior). An instruction that does not affect MSR must precede the MSRSET instruction to guarantee correct MSR value.

The immediate values has to be less than 2^{15} when `C_USE_MMU >= 1`, and less than 2^{14} otherwise. Only bits 17 to 31 of the MSR can be set when `C_USE_MMU >= 1`, and.bits 18 to 31 otherwise.

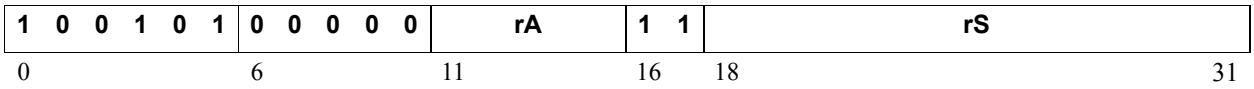
This instruction is only available when the parameter `C_USE_MSR_INSTR` is set to 1.

When setting MSR[VM] the instruction must always be followed by a synchronizing branch instruction, for example BRI 4.

mts

Move To Special Purpose Register

mts rS, rA



Description

Copies the contents of register rD into the special purpose register rS. The special purpose registers TLBLO and TLBHI are used to copy to the Unified TLB entry indexed by TLBX.

When MicroBlaze is configured to use an MMU (`C_USE_MMU` >= 1) this instruction is privileged. This means that if the instruction is attempted in User Mode (`MSR[UM]` = 1) a Privileged Instruction exception occurs.

Pseudocode

```
if MSR[UM] = 1 then
  ESR[EC] ← 00111
else
  switch (rS)
    case 0x0001 : MSR  ← (rA)
    case 0x0007 : FSR  ← (rA)
    case 0x0800 : SLR  ← (rA)
    case 0x0802 : SHR  ← (rA)
    case 0x1000 : PID  ← (rA)
    case 0x1001 : ZPR  ← (rA)
    case 0x1002 : TLBX  ← (rA)
    case 0x1003 : TLBLO ← (rA)
    case 0x1004 : TLBHI ← (rA)
    case 0x1005 : TLBSX ← (rA)
```

Registers Altered

- rS
- ESR[EC], in case a privileged instruction exception is generated

Latency

- 1 cycle

Notes

When writing MSR using MTS, all bits take effect one cycle after the instruction has been executed. An MTS instruction writing MSR should never be followed back-to-back by an instruction that uses the MSR content. When clearing the IE bit, it is guaranteed that the processor will not react to any interrupt for the subsequent instructions. When setting the EIP or BIP bit, it is guaranteed that the processor will not react to any interrupt or normal hardware break for the subsequent instructions.

To refer to special purpose registers in assembly language, use rmsr for MSR, rfsr for FSR, rslr for SLR, rshr for SHR, rpid for PID, rzpr for ZPR, rtlblo for TLBLO, rtlbhi for TLBHI, rtlbx for TLBX, and rtlbsx for TLBSX.

The PC, ESR, EAR, BTR, EDR and PVR0 - PVR11 cannot be written by the MTS instruction.

The FSR is only valid as a destination if the MicroBlaze parameter `C_USE_FPU` is greater than 0.

The SLR and SHR are only valid as a destination if the MicroBlaze parameter C_USE_STACK_PROTECTION is set to 1.

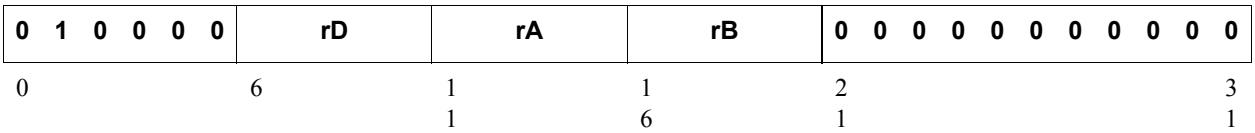
PID, ZPR and TLBSX are only valid as destinations when the parameter `C_USE_MMU` > 1 and the parameter `C_MMU_TLB_ACCESS` > 1. TLBLO, TLBHI and TLBX are only valid as destinations when the parameter `C_USE_MMU` > 1.

When changing MSR[VM] or PID the instruction must always be followed by a synchronizing branch instruction, for example BRI 4.

mul

Multiply

mul rD, rA, rB



Description

Multiplies the contents of registers rA and rB and puts the result in register rD. This is a 32-bit by 32-bit multiplication that will produce a 64-bit result. The least significant word of this value is placed in rD. The most significant word is discarded.

Pseudocode

```
(rD) ← LSW( (rA) × (rB) )
```

Registers Altered

- rD

Latency

- 1 cycle with `C_AREA_OPTIMIZED=0`
- 3 cycles with `C_AREA_OPTIMIZED=1`

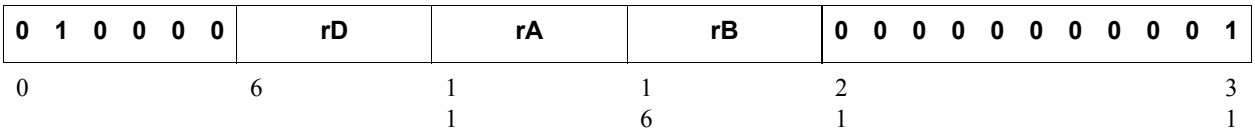
Note

This instruction is only valid if the target architecture has multiplier primitives, and if present, the MicroBlaze parameter `C_USE_HW_MUL` is greater than 0.

mulh

Multiply High

mulh rD, rA, rB



Description

Multiplies the contents of registers rA and rB and puts the result in register rD. This is a 32-bit by 32-bit signed multiplication that will produce a 64-bit result. The most significant word of this value is placed in rD. The least significant word is discarded.

Pseudocode

```
(rD) ← MSW( (rA) × (rB) ), signed
```

Registers Altered

- rD

Latency

- 1 cycle with `C_AREA_OPTIMIZED=0`
- 3 cycles with `C_AREA_OPTIMIZED=1`

Note

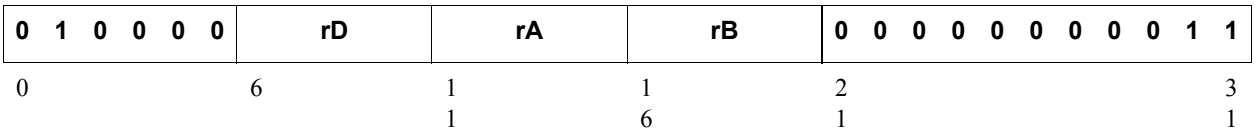
This instruction is only valid if the target architecture has multiplier primitives, and if present, the MicroBlaze parameter `C_USE_HW_MUL` is set to 2.

When MULH is used, bit 30 and 31 in the MUL instruction must be zero to distinguish between the two instructions. In previous versions of MicroBlaze, these bits were defined as zero, but the actual values were not relevant.

mulhu

Multiply High Unsigned

mulhu rD, rA, rB



Description

Multiplies the contents of registers rA and rB and puts the result in register rD. This is a 32-bit by 32-bit unsigned multiplication that will produce a 64-bit unsigned result. The most significant word of this value is placed in rD. The least significant word is discarded.

Pseudocode

```
(rD) ← MSW( (rA) × (rB) ), unsigned
```

Registers Altered

- rD

Latency

- 1 cycle with `C_AREA_OPTIMIZED=0`
- 3 cycles with `C_AREA_OPTIMIZED=1`

Note

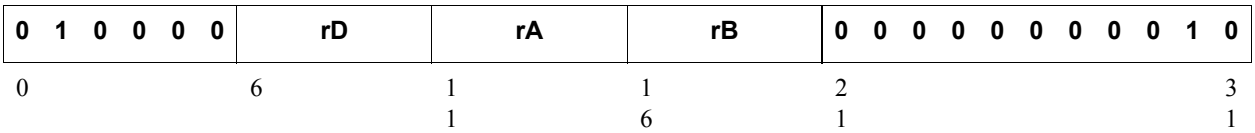
This instruction is only valid if the target architecture has multiplier primitives, and if present, the MicroBlaze parameter `C_USE_HW_MUL` is set to 2.

When MULHU is used, bit 30 and 31 in the MUL instruction must be zero to distinguish between the two instructions. In previous versions of MicroBlaze, these bits were defined as zero, but the actual values were not relevant.

mulhsu

Multiply High Signed Unsigned

mulhsu rD, rA, rB



Description

Multiplies the contents of registers rA and rB and puts the result in register rD. This is a 32-bit signed by 32-bit unsigned multiplication that will produce a 64-bit signed result. The most significant word of this value is placed in rD. The least significant word is discarded.

Pseudocode

```
(rD) ← MSW( (rA), signed × (rB), unsigned ), signed
```

Registers Altered

- rD

Latency

- 1 cycle with `C_AREA_OPTIMIZED=0`
- 3 cycles with `C_AREA_OPTIMIZED=1`

Note

This instruction is only valid if the target architecture has multiplier primitives, and if present, the MicroBlaze parameter `C_USE_HW_MUL` is set to 2.

When MULHSU is used, bit 30 and 31 in the MUL instruction must be zero to distinguish between the two instructions. In previous versions of MicroBlaze, these bits were defined as zero, but the actual values were not relevant.

muli

Multiply Immediate

muli rD, rA, IMM

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

Description

Multiplies the contents of registers rA and the value IMM, sign-extended to 32 bits; and puts the result in register rD. This is a 32-bit by 32-bit multiplication that will produce a 64-bit result. The least significant word of this value is placed in rD. The most significant word is discarded.

Pseudocode

```
(rD) ← LSW( (rA) × sext(IMM) )
```

Registers Altered

- rD

Latency

- 1 cycle with `C_AREA_OPTIMIZED=0`
- 3 cycles with `C_AREA_OPTIMIZED=1`

Notes

By default, Type B Instructions will take the 16-bit IMM field value and sign extend it to 32 bits to use as the immediate operand. This behavior can be overridden by preceding the Type B instruction with an imm instruction. See the instruction "imm," page 180 for details on using 32-bit immediate values.

This instruction is only valid if the target architecture has multiplier primitives, and if present, the MicroBlaze parameter `C_USE_HW_MUL` is greater than 0.

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	6	11	16	21 31

Description

The contents of register rA are ORed with the contents of register rB; the result is placed into register rD.

Pseudocode

```
(rD) ← (rA) ∨ (rB)
```

Registers Altered

- rD

Latency

- 1 cycle

ori

Logical OR with Immediate

ori rD, rA, IMM

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

Description

The contents of register rA are ORed with the extended IMM field, sign-extended to 32 bits; the result is placed into register rD.

Pseudocode

```
(rD) ← (rA) ∨ sext(IMM)
```

Registers Altered

- rD

Latency

- 1 cycle

Note

By default, Type B Instructions will take the 16-bit IMM field value and sign extend it to 32 bits to use as the immediate operand. This behavior can be overridden by preceding the Type B instruction with an imm instruction. See the instruction “imm,” page 180 for details on using 32-bit immediate values.

pcmpbf

Pattern Compare Byte Find

pcmpbf	rD, rA, rB	bytewise comparison returning position of first match

1 0 0 0 0 0	rD	rA	rB	1 0 0 0 0 0 0 0 0 0 0
0	6	1 1	1 6	2 3 1 1

Description

The contents of register rA is bytewise compared with the contents in register rB.

- rD is loaded with the position of the first matching byte pair, starting with MSB as position 1, and comparing until LSB as position 4
- If none of the byte pairs match, rD is set to 0

Pseudocode

```
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
```

Registers Altered

- rD

Latency

- 1 cycle

Note

This instruction is only available when the parameter `C_USE_PCMP_INSTR` is set to 1.

pcmpeq

Pattern Compare Equal

pcmpeq	rD, rA, rB	equality comparison with a positive boolean result

1 0 0 0 1 0	rD	rA	rB	1 0 0 0 0 0 0 0 0 0 0
0	6	1 1	1 6	2 3 1 1

Description

The contents of register rA is compared with the contents in register rB.

- rD is loaded with 1 if they match, and 0 if not

Pseudocode

```
if (rB) = (rA) then
  (rD) ← 1
else
  (rD) ← 0
```

Registers Altered

- rD

Latency

- 1 cycle

Note

This instruction is only available when the parameter `C_USE_PCMP_INSTR` is set to 1.

pcmpne

Pattern Compare Not Equal

pcmpne	rD, rA, rB	equality comparison with a negative boolean result

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

Description

The contents of register rA is compared with the contents in register rB.

- rD is loaded with 0 if they match, and 1 if not

Pseudocode

```
if (rB) = (rA) then
  (rD) ← 0
else
  (rD) ← 1
```

Registers Altered

- rD

Latency

- 1 cycle

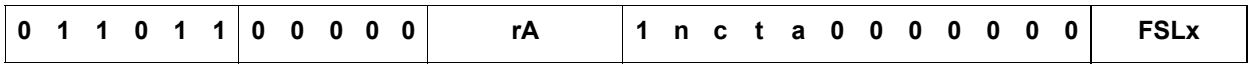
Note

This instruction is only available when the parameter `C_USE_PCMP_INSTR` is set to 1.

put

Put to stream interface

*na*put	rA, FSLx	put data to link x *n* = non-blocking *a* = atomic
t*na*put	FSLx	put data to link x test-only *n* = non-blocking *a* = atomic
*nca*put	rA, FSLx	put control to link x *n* = non-blocking *a* = atomic
t*nca*put	FSLx	put control to link x test-only *n* = non-blocking *a* = atomic



Description

MicroBlaze will write the value from register rA to the link x interface.

The put instruction has 16 variants.

The blocking versions (when ‘n’ is ‘0’) will stall MicroBlaze until there is space available in the interface. The non-blocking versions will not stall MicroBlaze and will set carry to ‘0’ if space was available and to ‘1’ if no space was available.

All data put instructions (when ‘c’ is ‘0’) will set the control bit to the interface to ‘0’ and all control put instructions (when ‘c’ is ‘1’) will set the control bit to ‘1’.

The test versions (when ‘t’ bit is ‘1’) will be handled as the normal case, except that the write signal to the link is not asserted (thus no source register is required).

Atomic versions (when ‘a’ bit is ‘1’) are not interruptible. This means that a sequence of atomic instructions can be grouped together without an interrupt breaking the program flow. However, note that exceptions may still occur.

When MicroBlaze is configured to use an MMU (`C_USE_MMU` >= 1) and not explicitly allowed by setting `C_MMU_PRIVILEGED_INSTR` to 1 these instructions are privileged. This means that if these instructions are attempted in User Mode (MSR[UM] = 1) a Privileged Instruction exception occurs.

Pseudocode

```
if MSR[UM] = 1 then
  ESR[EC] ← 00111
else
  (FSLx_M_DATA | Mx_AXIS_TDATA) ← (rA)
  if (n = 1) then
    MSR[Carry]  ←
      (FSLx_M_FULL | Mx_AXIS_TVALID ∧ ~~Mx_AXIS_TREADY~~)
  (FSLx_M_CONTROL | Mx_AXIS_TLAST) ← C
```

Registers Altered

- MSR[Carry]
- ESR[EC], in case a privileged instruction exception is generated

Latency

- 1 cycle with `C_AREA_OPTIMIZED=0`
- 2 cycles with `C_AREA_OPTIMIZED=1`

The blocking versions of this instruction will stall the pipeline of MicroBlaze until the instruction can be completed. Interrupts are served when the parameter C_USE_EXTENDED_FSL_INSTR is set to 1, and the instruction is not atomic.

Note

To refer to an FSLx interface in assembly language, use rfsl0, rfsl1, ... rfsl15.

The blocking versions of this instruction should not be placed in a delay slot when the parameter C_USE_EXTENDED_FSL_INSTR is set to 1, since this prevents interrupts from being served.

These instructions are only available when the MicroBlaze parameter `C_FSL_LINKS` is greater than 0.

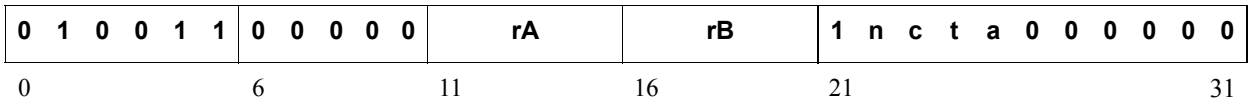
The extended instructions (atomic versions) are only available when the MicroBlaze parameter `C_USE_EXTENDED_FSL_INSTR` is set to 1.

It is not recommended to allow these instructions in user mode, unless absolutely necessary for performance reasons, since that removes all hardware protection preventing incorrect use of a link.

putd

Put to stream interface dynamic

*na*putd	rA, rB	put data to link rB[28:31] *n* = non-blocking *a* = atomic
t*na*putd	rB	put data to link rB[28:31] test-only *n* = non-blocking *a* = atomic
*nca*putd	rA, rB	put control to link rB[28:31] *n* = non-blocking *a* = atomic
t*nca*putd	rB	put control to link rB[28:31] test-only *n* = non-blocking *a* = atomic



Description

MicroBlaze will write the value from register rA to the link interface defined by the four least significant bits in rB. The putd instruction has 16 variants.

The blocking versions (when ‘n’ is ‘0’) will stall MicroBlaze until there is space available in the interface. The non-blocking versions will not stall MicroBlaze and will set carry to ‘0’ if space was available and to ‘1’ if no space was available.

All data putd instructions (when ‘c’ is ‘0’) will set the control bit to the interface to ‘0’ and all control putd instructions (when ‘c’ is ‘1’) will set the control bit to ‘1’.

The test versions (when ‘t’ bit is ‘1’) will be handled as the normal case, except that the write signal to the link is not asserted (thus no source register is required).

Atomic versions (when ‘a’ bit is ‘1’) are not interruptible. This means that a sequence of atomic instructions can be grouped together without an interrupt breaking the program flow. However, note that exceptions may still occur.

When MicroBlaze is configured to use an MMU (`C_USE_MMU` >= 1) and not explicitly allowed by setting `C_MMU_PRIVILEGED_INSTR` to 1 these instructions are privileged. This means that if these instructions are attempted in User Mode (MSR[UM] = 1) a Privileged Instruction exception occurs.

Pseudocode

```
if MSR[UM] = 1 then
  ESR[EC] ← 00111
else
  x ← rB[28:31]
  (FSLx_M_DATA | Mx_AXIS_TDATA) ← (rA)
  if (n = 1) then
    MSR[Carry] ←
      (FSLx_M_FULL | Mx_AXIS_TVALID ∧ ~~Mx_AXIS_TREADY~~)
  (FSLx_M_CONTROL | Mx_AXIS_TLAST) ← C
```

Registers Altered

- MSR[Carry]
- ESR[EC], in case a privileged instruction exception is generated

Latency

- 1 cycle with `C_AREA_OPTIMIZED=0`
- 2 cycles with `C_AREA_OPTIMIZED=1`

The blocking versions of this instruction will stall the pipeline of MicroBlaze until the instruction can be completed. Interrupts are served unless the instruction is atomic, which ensures that the instruction cannot be interrupted.

Note

The blocking versions of this instruction should not be placed in a delay slot, since this prevents interrupts from being served.

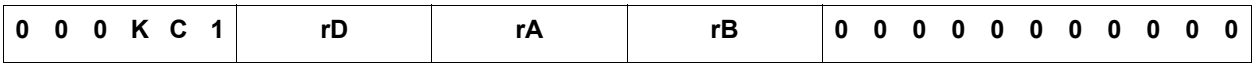
These instructions are only available when the MicroBlaze parameter `C_FSL_LINKS` is greater than 0 and the parameter `C_USE_EXTENDED_FSL_INSTR` is set to 1.

It is not recommended to allow these instructions in user mode, unless absolutely necessary for performance reasons, since that removes all hardware protection preventing incorrect use of a link.

rsub

Arithmetic Reverse Subtract

rsub	rD, rA, rB	Subtract
rsubc	rD, rA, rB	Subtract with Carry
rsubk	rD, rA, rB	Subtract and Keep Carry
rsubkc	rD, rA, rB	Subtract with Carry and Keep Carry



Description

The contents of register rA is subtracted from the contents of register rB and the result is placed into register rD. Bit 3 of the instruction (labeled as K in the figure) is set to one for the mnemonic rsubk. Bit 4 of the instruction (labeled as C in the figure) is set to one for the mnemonic rsubc. Both bits are set to one for the mnemonic rsubkc.

When an rsub instruction has bit 3 set (rsubk, rsubkc), the carry flag will Keep its previous value regardless of the outcome of the execution of the instruction. If bit 3 is cleared (rsub, rsubc), then the carry flag will be affected by the execution of the instruction.

When bit 4 of the instruction is set to one (rsubc, rsubkc), the content of the carry flag (MSR[C]) affects the execution of the instruction. When bit 4 is cleared (rsub, rsubk), the content of the carry flag does not affect the execution of the instruction (providing a normal subtraction).

Pseudocode

```
if C = 0 then
  (rD) ← (rB) + (rA) + 1
else
  (rD) ← (rB) + (rA) + MSR[C]
if K = 0 then
  MSR[C] ← CarryOut
```

Registers Altered

- rD
- MSR[C]

Latency

- 1 cycle

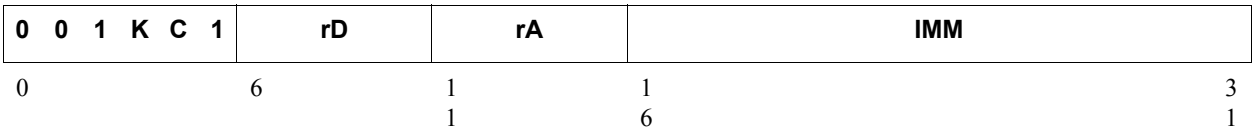
Notes

In subtractions, Carry = ($\overline{\text{Borrow}}$). When the Carry is set by a subtraction, it means that there is no Borrow, and when the Carry is cleared, it means that there is a Borrow.

rsubi

Arithmetic Reverse Subtract Immediate

rsubi	rD, rA, IMM	Subtract Immediate
rsubic	rD, rA, IMM	Subtract Immediate with Carry
rsubik	rD, rA, IMM	Subtract Immediate and Keep Carry
rsubikc	rD, rA, IMM	Subtract Immediate with Carry and Keep Carry



Description

The contents of register rA is subtracted from the value of IMM, sign-extended to 32 bits, and the result is placed into register rD. Bit 3 of the instruction (labeled as K in the figure) is set to one for the mnemonic rsubik. Bit 4 of the instruction (labeled as C in the figure) is set to one for the mnemonic rsubic. Both bits are set to one for the mnemonic rsubikc.

When an rsubi instruction has bit 3 set (rsubik, rsubikc), the carry flag will Keep its previous value regardless of the outcome of the execution of the instruction. If bit 3 is cleared (rsubi, rsubic), then the carry flag will be affected by the execution of the instruction. When bit 4 of the instruction is set to one (rsubic, rsubikc), the content of the carry flag (MSR[C]) affects the execution of the instruction. When bit 4 is cleared (rsubi, rsubik), the content of the carry flag does not affect the execution of the instruction (providing a normal subtraction).

Pseudocode

```
if C = 0 then
  (rD) ← sext(IMM) + (rA) + 1
else
  (rD) ← sext(IMM) + (rA) + MSR[C]
if K = 0 then
  MSR[C] ← CarryOut
```

Registers Altered

- rD
- MSR[C]

Latency

- 1 cycle

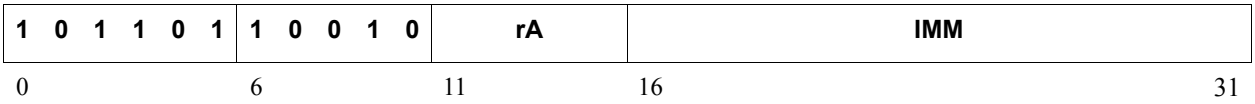
Notes

In subtractions, Carry = ($\overline{\text{Borrow}}$). When the Carry is set by a subtraction, it means that there is no Borrow, and when the Carry is cleared, it means that there is a Borrow. By default, Type B Instructions will take the 16-bit IMM field value and sign extend it to 32 bits to use as the immediate operand. This behavior can be overridden by preceding the Type B instruction with an imm instruction. See the instruction “imm,” page 180 for details on using 32-bit immediate values.

rtbd

Return from Break

rtbd rA, IMM



Description

Return from break will branch to the location specified by the contents of rA plus the IMM field, sign-extended to 32 bits. It will also enable breaks after execution by clearing the BIP flag in the MSR.

This instruction always has a delay slot. The instruction following the RTBD is always executed before the branch target. That delay slot instruction has breaks disabled.

When MicroBlaze is configured to use an MMU (`C_USE_MMU` >= 1) this instruction is privileged. This means that if the instruction is attempted in User Mode (`MSR[UM]` = 1) a Privileged Instruction exception occurs.

Pseudocode

```
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]
```

Registers Altered

- PC
- MSR[BIP], MSR[UM], MSR[VM]
- ESR[EC], in case a privileged instruction exception is generated

Latency

- 2 cycles

Note

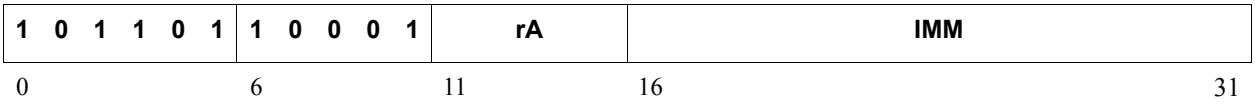
Convention is to use general purpose register r16 as rA.

A delay slot must not be used by the following: imm, branch, or break instructions. Interrupts and external hardware breaks are deferred until after the delay slot branch has been completed.

rtid

Return from Interrupt

rtid rA, IMM



Description

Return from interrupt will branch to the location specified by the contents of rA plus the IMM field, sign-extended to 32 bits. It will also enable interrupts after execution.

This instruction always has a delay slot. The instruction following the RTID is always executed before the branch target. That delay slot instruction has interrupts disabled.

When MicroBlaze is configured to use an MMU (`C_USE_MMU` >= 1) this instruction is privileged. This means that if the instruction is attempted in User Mode (`MSR[UM] = 1`) a Privileged Instruction exception occurs.

Pseudocode

```
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]
```

Registers Altered

- PC
- MSR[IE], MSR[UM], MSR[VM]
- ESR[EC], in case a privileged instruction exception is generated

Latency

- 2 cycles

Note

Convention is to use general purpose register r14 as rA.

A delay slot must not be used by the following: imm, branch, or break instructions. Interrupts and external hardware breaks are deferred until after the delay slot branch has been completed.

rted

Return from Exception

rted rA, IMM

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

Description

Return from exception will branch to the location specified by the contents of rA plus the IMM field, sign-extended to 32 bits. The instruction will also enable exceptions after execution.

This instruction always has a delay slot. The instruction following the RTED is always executed before the branch target.

When MicroBlaze is configured to use an MMU (C_USE_MMU >= 1) this instruction is privileged. This means that if the instruction is attempted in User Mode (MSR[UM] = 1) a Privileged Instruction exception occurs.

Pseudocode

```
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
```

Registers Altered

- PC
- MSR[EE], MSR[EIP], MSR[UM], MSR[VM]
- ESR

Latency

- 2 cycles

Note

Convention is to use general purpose register r17 as rA. This instruction requires that one or more of the MicroBlaze parameters C_*_EXCEPTION are set to 1 or that C_USE_MMU > 0.

A delay slot must not be used by the following: imm, branch, or break instructions. Interrupts and external hardware breaks are deferred until after the delay slot branch has been completed.

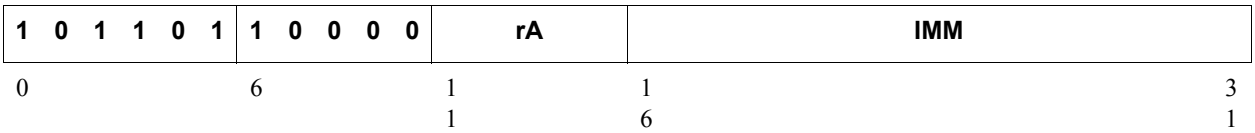
The instruction should normally not be used when MSR[EE] is set, since if the instruction in the delay slot would cause an exception, the exception handler would be entered with exceptions enabled.

Note: Code returning from an exception must first check if MSR[DS] is set, and in that case return to the address in BTR.

rtsd

Return from Subroutine

rtsd rA, IMM



Description

Return from subroutine will branch to the location specified by the contents of rA plus the IMM field, sign-extended to 32 bits.

This instruction always has a delay slot. The instruction following the RTSD is always executed before the branch target.

Pseudocode

```
PC ← (rA) + sext(IMM)
allow following instruction to complete execution
```

Registers Altered

- PC

Latency

- 1 cycle (if successful branch prediction occurs)
- 2 cycles (with Branch Target Cache disabled)
- 3 cycles (if branch prediction mispredict occurs)

Note

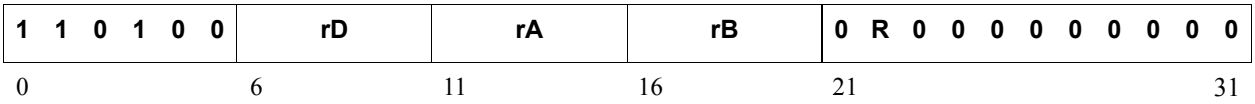
Convention is to use general purpose register r15 as rA.

A delay slot must not be used by the following: imm, branch, or break instructions. Interrupts and external hardware breaks are deferred until after the delay slot branch has been completed.

sb

Store Byte

sb	rD, rA, rB
sbr	rD, rA, rB



Description

Stores the contents of the least significant byte of register rD, into the memory location that results from adding the contents of registers rA and rB.

If the R bit is set, a byte reversed memory location is used, storing data with the opposite endianness of the endianness defined by C_ENDIANNESS and the E bit (if virtual protected mode is enabled).

A data TLB miss exception occurs if virtual protected mode is enabled, and a valid translation entry corresponding to the address is not found in the TLB.

A data storage exception occurs if virtual protected mode is enabled, and access is prevented by no-access-allowed or read-only zone protection. No-access-allowed can only occur in user mode.

Pseudocode

```
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]
```

Registers Altered

- MSR[UM], MSR[VM], MSR[UMS], MSR[VMS], if an exception is generated
- ESR[EC], ESR[S], if an exception is generated
- ESR[DIZ], if a data storage exception is generated

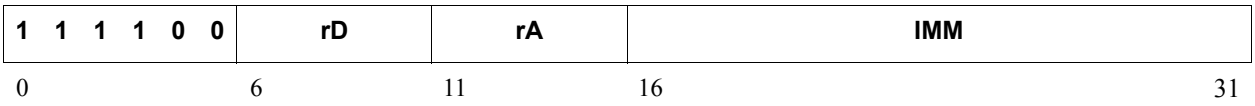
Latency

- 1 cycle with C_AREA_OPTIMIZED=0
- 2 cycles with C_AREA_OPTIMIZED=1

sbi

Store Byte Immediate

sbi rD, rA, IMM



Description

Stores the contents of the least significant byte of register rD, into the memory location that results from adding the contents of register rA and the value IMM, sign-extended to 32 bits.

A data TLB miss exception occurs if virtual protected mode is enabled, and a valid translation entry corresponding to the address is not found in the TLB.

A data storage exception occurs if virtual protected mode is enabled, and access is prevented by no-access-allowed or read-only zone protection. No-access-allowed can only occur in user mode.

Pseudocode

```
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]
```

Registers Altered

- MSR[UM], MSR[VM], MSR[UMS], MSR[VMS], if an exception is generated
- ESR[EC], ESR[S], if an exception is generated
- ESR[DIZ], if a data storage exception is generated

Latency

- 1 cycle with `C_AREA_OPTIMIZED=0`
- 2 cycles with `C_AREA_OPTIMIZED=1`

Note

By default, Type B Instructions will take the 16-bit IMM field value and sign extend it to 32 bits to use as the immediate operand. This behavior can be overridden by preceding the Type B instruction with an imm instruction. See the instruction "imm," page 180 for details on using 32-bit immediate values.

sext16

Sign Extend Halfword

sext16 rD, rA

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

Description

This instruction sign-extends a halfword (16 bits) into a word (32 bits). Bit 16 in rA will be copied into bits 0-15 of rD. Bits 16-31 in rA will be copied into bits 16-31 of rD.

Pseudocode

```
(rD)[0:15] ← (rA)[16]
(rD)[16:31] ← (rA)[16:31]
```

Registers Altered

- rD

Latency

- 1 cycle

sext8

Sign Extend Byte

sext8 rD, rA

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

Description

This instruction sign-extends a byte (8 bits) into a word (32 bits). Bit 24 in rA will be copied into bits 0-23 of rD. Bits 24-31 in rA will be copied into bits 24-31 of rD.

Pseudocode

```
(rD)[0:23] ← (rA)[24]
(rD)[24:31] ← (rA)[24:31]
```

Registers Altered

- rD

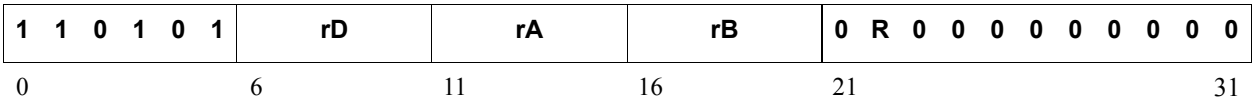
Latency

- 1 cycle

sh

Store Halfword

sh	rD, rA, rB
shr	rD, rA, rB



Description

Stores the contents of the least significant halfword of register rD, into the halfword aligned memory location that results from adding the contents of registers rA and rB.

If the R bit is set, a halfword reversed memory location is used and the two bytes in the halfword are reversed, storing data with the opposite endianness of the endianness defined by C_ENDIANNESS and the E bit (if virtual protected mode is enabled).

A data TLB miss exception occurs if virtual protected mode is enabled, and a valid translation entry corresponding to the address is not found in the TLB.

A data storage exception occurs if virtual protected mode is enabled, and access is prevented by no-access-allowed or read-only zone protection. No-access-allowed can only occur in user mode.

An unaligned data access exception occurs if the least significant bit in the address is not zero.

Pseudocode

```
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]
```

Registers Altered

- MSR[UM], MSR[VM], MSR[UMS], MSR[VMS], if a TLB miss exception or a data storage exception is generated
- ESR[EC], ESR[S], if an exception is generated
- ESR[DIZ], if a data storage exception is generated
- ESR[W], ESR[Rx], if an unaligned data access exception is generated

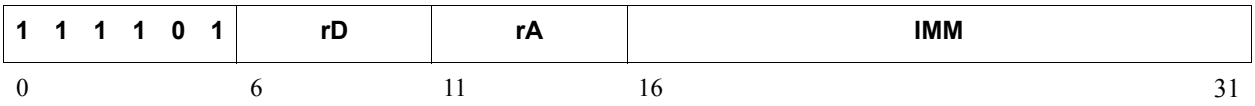
Latency

- 1 cycle with C_AREA_OPTIMIZED=0
- 2 cycles with C_AREA_OPTIMIZED=1

shi

Store Halfword Immediate

shi rD, rA, IMM



Description

Stores the contents of the least significant halfword of register rD, into the halfword aligned memory location that results from adding the contents of register rA and the value IMM, sign-extended to 32 bits.

A data TLB miss exception occurs if virtual protected mode is enabled, and a valid translation entry corresponding to the address is not found in the TLB. A data storage exception occurs if virtual protected mode is enabled, and access is prevented by no-access-allowed or read-only zone protection. No-access-allowed can only occur in user mode. An unaligned data access exception occurs if the least significant bit in the address is not zero.

Pseudocode

```
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]
```

Registers Altered

- MSR[UM], MSR[VM], MSR[UMS], MSR[VMS], if a TLB miss exception or a data storage exception is generated
- ESR[EC], ESR[S], if an exception is generated
- ESR[DIZ], if a data storage exception is generated
- ESR[W], ESR[Rx], if an unaligned data access exception is generated

Latency

- 1 cycle with `C_AREA_OPTIMIZED=0`
- 2 cycles with `C_AREA_OPTIMIZED=1`

Note

By default, Type B Instructions will take the 16-bit IMM field value and sign extend it to 32 bits to use as the immediate operand. This behavior can be overridden by preceding the Type B instruction with an imm instruction. See the instruction “imm,” page 180 for details on using 32-bit immediate values.

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	1
0						6	11	16														31

Description

Shifts arithmetically the contents of register rA, one bit to the right, and places the result in rD. The most significant bit of rA (that is, the sign bit) placed in the most significant bit of rD. The least significant bit coming out of the shift chain is placed in the Carry flag.

Pseudocode

```
(rD)[0] ← (rA)[0]
(rD)[1:31] ← (rA)[0:30]
MSR[C] ← (rA)[31]
```

Registers Altered

- rD
- MSR[C]

Latency

- 1 cycle

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	0	1	0	0	0	0	1
0						6	1 1	1 6															3 1

Description

Shifts the contents of register rA, one bit to the right, and places the result in rD. The Carry flag is shifted in the shift chain and placed in the most significant bit of rD. The least significant bit coming out of the shift chain is placed in the Carry flag.

Pseudocode

```
(rD)[0] ← MSR[C]
(rD)[1:31] ←(rA)[0:30]
MSR[C] ← (rA)[31]
```

Registers Altered

- rD
- MSR[C]

Latency

- 1 cycle

srl

Shift Right Logical

srl rD, rA

1	0	0	1	0	0	rD	rA	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	1
0						6	1 1	1 6															3 1

Description

Shifts logically the contents of register rA, one bit to the right, and places the result in rD. A zero is shifted in the shift chain and placed in the most significant bit of rD. The least significant bit coming out of the shift chain is placed in the Carry flag.

Pseudocode

```
(rD)[0] ← 0
(rD)[1:31] ←(rA)[0:30]
MSR[C] ← (rA)[31]
```

Registers Altered

- rD
- MSR[C]

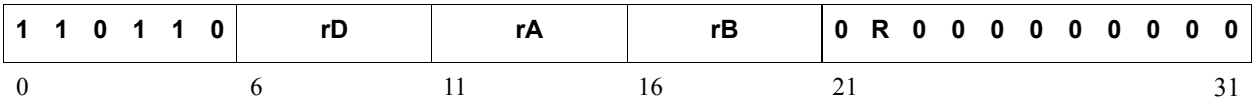
Latency

- 1 cycle

SW

Store Word

sw	rD, rA, rB
swr	rD, rA, rB



Description

Stores the contents of register rD, into the word aligned memory location that results from adding the contents of registers rA and rB.

If the R bit is set, the bytes in the stored word are reversed , storing data with the opposite endianness of the endianness defined by `C_ENDIANNESS` and the E bit (if virtual protected mode is enabled).

A data TLB miss exception occurs if virtual protected mode is enabled, and a valid translation entry corresponding to the address is not found in the TLB.

A data storage exception occurs if virtual protected mode is enabled, and access is prevented by no-access-allowed or read-only zone protection. No-access-allowed can only occur in user mode.

An unaligned data access exception occurs if the two least significant bits in the address are not zero.

Pseudocode

```
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]
```

Registers Altered

- MSR[UM], MSR[VM], MSR[UMS], MSR[VMS], if a TLB miss exception or a data storage exception is generated
- ESR[EC], ESR[S], if an exception is generated
- ESR[DIZ], if a data storage exception is generated
- ESR[W], ESR[Rx], if an unaligned data access exception is generated

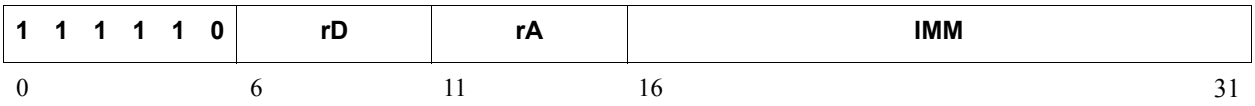
Latency

- 1 cycle with `C_AREA_OPTIMIZED=0`
- 2 cycles with `C_AREA_OPTIMIZED=1`

swi

Store Word Immediate

swi rD, rA, IMM



Description

Stores the contents of register rD, into the word aligned memory location that results from adding the contents of registers rA and the value IMM, sign-extended to 32 bits.

A data TLB miss exception occurs if virtual protected mode is enabled, and a valid translation entry corresponding to the address is not found in the TLB.

A data storage exception occurs if virtual protected mode is enabled, and access is prevented by no-access-allowed or read-only zone protection. No-access-allowed can only occur in user mode.

An unaligned data access exception occurs if the two least significant bits in the address are not zero.

Pseudocode

```
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]
```

Register Altered

- MSR[UM], MSR[VM], MSR[UMS], MSR[VMS], if a TLB miss exception or a data storage exception is generated
- ESR[EC], ESR[S], if an exception is generated
- ESR[DIZ], if a data storage exception is generated
- ESR[W], ESR[Rx], if an unaligned data access exception is generated

Latency

- 1 cycle with `C_AREA_OPTIMIZED=0`
- 2 cycles with `C_AREA_OPTIMIZED=1`

Note

By default, Type B Instructions will take the 16-bit IMM field value and sign extend it to 32 bits to use as the immediate operand. This behavior can be overridden by preceding the Type B instruction with an imm instruction. See the instruction "imm," page 180 for details on using 32-bit immediate values.

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	6	11	16	21 31

Description

Conditionally stores the contents of register rD, into the word aligned memory location that results from adding the contents of registers rA and rB. If an AXI4 interconnect with exclusive access enabled is used, the store occurs if the interconnect response is EXOKAY, and the reservation bit is set; otherwise the store occurs when the reservation bit is set. The carry flag (MSR[C]) is set if the store does not occur, otherwise it is cleared. The reservation bit is cleared.

A data TLB miss exception occurs if virtual protected mode is enabled, and a valid translation entry corresponding to the address is not found in the TLB.

A data storage exception occurs if virtual protected mode is enabled, and access is prevented by no-access-allowed or read-only zone protection. No-access-allowed can only occur in user mode.

An unaligned data access exception will *not* occur even if the two least significant bits in the address are not zero.

Enabling AXI exclusive access ensures that the operation is protected from other bus masters, but requires that the addressed slave supports exclusive access. When exclusive access is not enabled, only the internal reservation bit is used. Exclusive access is enabled using the two parameters `C_M_AXI_DP_EXCLUSIVE_ACCESS` and `C_M_AXI_DC_EXCLUSIVE_ACCESS` for the peripheral and cache interconnect, respectively.

Pseudocode

```
Addr ← (rA) + (rB)
if Reservation = 0 then
  MSR[C] ← 1
else
  if TLB_Miss(Addr) and MSR[VM] = 1 then
    ESR[EC]← 10010;ESR[S]← 1
    MSR[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
    Reservation ← 0
    if AXI_Exclusive_Used(Addr) && AXI_Response /= EXOKAY then
      MSR[C] ← 1
    else
      Mem(Addr) ← (rD)[0:31]
      MSR[C] ← 0
```

Registers Altered

- MSR[C], unless an exception is generated
- MSR[UM], MSR[VM], MSR[UMS], MSR[VMS], if a TLB miss exception or a data storage exception is generated
- ESR[EC], ESR[S], if an exception is generated
- ESR[DIZ], if a data storage exception is generated

Latency

- 1 cycle with `C_AREA_OPTIMIZED=0`
- 2 cycles with `C_AREA_OPTIMIZED=1`

Note

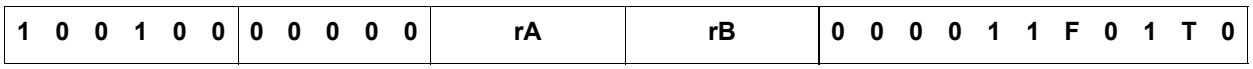
This instruction is used together with LWX to implement exclusive access, such as semaphores and spinlocks .

The carry flag (MSR[C]) may not be set immediately (dependent on pipeline stall behavior). The SWX instruction should not be immediately followed by an SRC instruction, to ensure the correct value of the carry flag is obtained.

wdc

Write to Data Cache

wdc	rA,rB
wdc.flush	rA,rB
wdc.clear	rA,rB



Description

Write into the data cache tag to invalidate or flush a cache line. The mnemonic wdc.flush is used to set the F bit, and wdc.clear is used to set the T bit.

When C_DCACHE_USE_WRITEBACK is set to 1, the instruction will flush the cache line and invalidate it if the F bit is set, otherwise it will only invalidate the cache line and discard any data that has not been written to memory. If the T bit is set, only a cache line with a matching address is invalidated. Register rA added with rB is the address of the affected cache line.

When C_DCACHE_USE_WRITEBACK is cleared to 0, the instruction will always invalidate the cache line. Register rA contains the address of the affected cache line, and the register rB value is not used.

When MicroBlaze is configured to use an MMU (`C_USE_MMU` >= 1) the instruction is privileged. This means that if the instruction is attempted in User Mode (`MSR[UM]` = 1) a Privileged Instruction exception occurs.

Pseudocode

```
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_mask ← (1 << log2(C_DCACHE_BYTE_SIZE) - 4) - 1
    cacheline      ← (DCache Line)[(address >> 4) ∧ cacheline_mask]
    cacheline_addr ← address & 0xfffffff0
  if C_DCACHE_LINE_LEN = 8 then
    cacheline_mask ← (1 << log2(C_DCACHE_BYTE_SIZE) - 5) - 1
    cacheline      ← (DCache Line)[(address >> 5) ∧ cacheline_mask]
    cacheline_addr ← address & 0xfffffff0
  if F = 1 and cacheline.Dirty then
    for i = 0 .. C_DCACHE_LINE_LEN - 1 loop
      if cacheline.Valid[i] then
        Mem(cacheline_addr + i * 4) ← cacheline.Data[i]
  if T = 0 then
    cacheline.Tag ← 0
  else if cacheline.Address = cacheline_addr then
    cacheline.Tag ← 0
```

Registers Altered

- ESR[EC], in case a privileged instruction exception is generated

Latency

- 2 cycles for wdc.clear
- 2 cycles for wdc with C_AREA_OPTIMIZED=1
- 3 cycles for wdc with C_AREA_OPTIMIZED=0
- 2 + N cycles for wdc.flush, where N is the number of clock cycles required to flush the cache line to memory when necessary

Note

The wdc, wdc.flush and wdc.clear instructions are independent of data cache enable (MSR[DCE]), and can be used either with the data cache enabled or disabled.

The wdc.clear instruction is intended to invalidate a specific area in memory, for example a buffer to be written by a Direct Memory Access device. Using this instruction ensures that other cache lines are not inadvertently invalidated, erroneously discarding data that has not yet been written to memory.

The address of the affected cache line is always the physical address, independent of the parameter C_USE_MMU and whether the MMU is in virtual mode or real mode.

When using wdc.flush in a loop to flush the entire cache, the loop can be optimized by using Ra as the cache base address and Rb as the loop counter:

```
        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
```

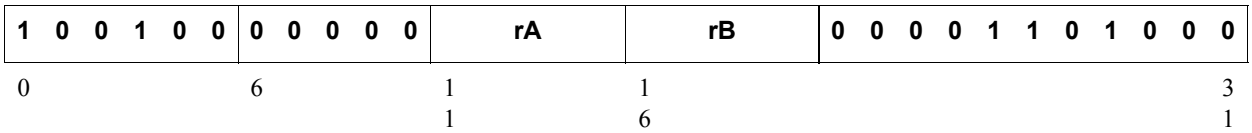
When using wdc.clear in a loop to invalidate a memory area in the cache, the loop can be optimized by using Ra as the memory area base address and Rb as the loop counter:

```
        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



Description

Write into the instruction cache tag to invalidate a cache line. The register rB value is not used. Register rA contains the address of the affected cache line.

When MicroBlaze is configured to use an MMU (`C_USE_MMU` >= 1) this instruction is privileged. This means that if the instruction is attempted in User Mode (`MSR[UM]` = 1) a Privileged Instruction exception occurs.

Pseudocode

```
if MSR[UM] = 1 then
  ESR[EC] ← 00111
else
  if C_ICACHE_LINE_LEN = 4 then
    cacheline_mask ← (1 << log2(C_CACHE_BYTE_SIZE) - 4) - 1
    (ICache Line)[((Ra) >> 4) ∧ cacheline_mask].Tag ← 0
  if C_ICACHE_LINE_LEN = 8 then
    cacheline_mask ← (1 << log2(C_CACHE_BYTE_SIZE) - 5) - 1
    (ICache Line)[((Ra) >> 5) ∧ cacheline_mask].Tag ← 0
```

Registers Altered

- ESR[EC], in case a privileged instruction exception is generated

Latency

- 2 cycles

Note

The WIC instruction is independent of instruction cache enable (MSR[ICE]), and can be used either with the instruction cache enabled or disabled.

The address of the affected cache line is always the physical address, independent of the parameter `C_USE_MMU` and whether the MMU is in virtual mode or real mode.

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	6	11	16	21 31

Description

The contents of register rA are XORed with the contents of register rB; the result is placed into register rD.

Pseudocode

```
(rD) ← (rA) ⊕ (rB)
```

Registers Altered

- rD

Latency

- 1 cycle

xori

Logical Exclusive OR with Immediate

xori rD, rA, IMM

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

Description

The IMM field is extended to 32 bits by concatenating 16 0-bits on the left. The contents of register rA are XOR'ed with the extended IMM field; the result is placed into register rD.

Pseudocode

```
(rD) ← (rA) ⊕ sext(IMM)
```

Registers Altered

- rD

Latency

- 1 cycle

Note

By default, Type B Instructions will take the 16-bit IMM field value and sign extend it to 32 bits to use as the immediate operand. This behavior can be overridden by preceding the Type B instruction with an imm instruction. See the instruction "imm," page 180 for details on using 32-bit immediate values.