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.

Data Types

The data types used by MicroBlaze assembly programs are shown in Table 4-1. Data types such as data8, data16, and data32 are used in place of the usual byte, half-word, and word.register.

Table 4-1: **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[1]	2/4

1. 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 4-2.

Table 4-2: **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/SW	Return address for Exceptions HW, if configured to support hardware exceptions, else SW
R18	Dedicated	SW	Reserved for Assembler/Compiler Temporaries Used for Service ID with attribute svc_table_handler
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
RPVR0-12	Special	HW	Processor Version Register 0 through 12

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 (0-12). 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 4-3.

The shaded area in Table 4-3 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 4-3: **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 4-1. 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 4-1.

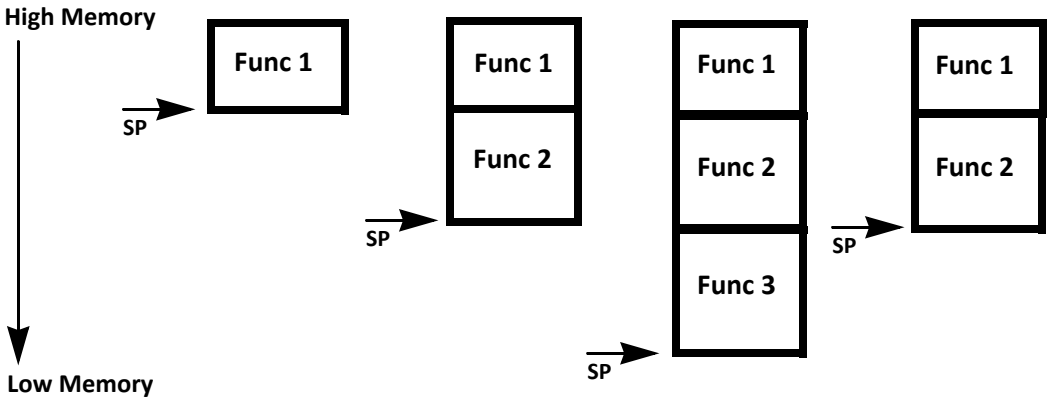


Figure 4-1: **Stack Frame**

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 4-1. The parameters for Func 2 are stored either in the registers R5 through R10 or on the stack frame allocated for Func 1.

If Func 2 has more than six integer parameters, the first six parameters can be passed in registers R5 through R10, whereas all subsequent parameters must be passed on the stack frame allocated for Func 1, starting at offset SP + 28.

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, Break and Exception Handling

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

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

On	Hardware jumps to	Software Labels
Start / Reset	C_BASE_VECTORS + 0x0	_start
User exception	C_BASE_VECTORS + 0x8	_exception_handler
Interrupt	C_BASE_VECTORS + 0x10[1]	_interrupt_handler
Break (HW/SW)	C_BASE_VECTORS + 0x18	-
Hardware exception	C_BASE_VECTORS + 0x20	_hw_exception_handler
Reserved by Xilinx for future use	C_BASE_VECTORS + 0x28 - C_BASE_VECTORS + 0x4F	-

1. With low-latency interrupt mode, the vector address is supplied by the Interrupt Controller.

The code expected at these locations is as shown below. 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.

The following is code for passing control to Exception, Break and Interrupt handlers, assuming the default C_BASE_VECTORS value of 0x00000000:

```
0x00:   bri     _start1
0x04:   nop
0x08:   imm     high bits of address (user exception handler)
0x0c:   bri     _exception_handler
0x10:   imm     high bits of address (interrupt handler)
0x14:   bri     _interrupt_handler
0x18:   imm     high bits of address (break handler)
0x1c:   bri     low bits of address (break handler)
0x20:   imm     high bits of address (HW exception handler
0x24:   bri     _hw_exception_handler
```

With low-latency interrupt mode, control is directly passed to the interrupt handler for each individual interrupt utilizing this mode. In this case, it is the responsibility of each handler to save and restore used registers. The MicroBlaze C compiler (mb-gcc) attribute **`fast_interrupt`** is available to allow this task to be performed by the compiler:

```
void interrupt_handler_name() __attribute__((fast_interrupt));
```

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` for interrupts that do not use low-latency handlers.

In the current MicroBlaze system, there are dummy routines for interrupt, break and user exception handling, which you can change. In order to override these routines and link your own interrupt and exception handlers, you must define the handler code with specific attributes.

The interrupt handler code must be defined with attribute `interrupt_handler` to ensure that the compiler will generate code to save and restore used registers and emit an `rtid` instruction to return from the handler:

```
void function_name() __attribute__((interrupt_handler));
```

The break handler code must be defined with attribute **`break_handler`** to ensure that the compiler will generate code to save and restore used registers and emit an **`rtbd`** instruction to return from the handler:

```
void function_name() __attribute__((break_handler));
```

The user exception handler code must either be defined with either attribute `svc_handler` or attribute `svc_table_handler`:

```
void function_name() __attribute__((svc_handler));
void function_name() __attribute__((svc_table_handler (ID)));
```

The first attribute ensures that the compiler will emit an indirect call to the handler with a `brki rD,0x8` instruction, and emit an `rtbd` instruction to return from the handler. This means that when the MMU is enabled the handler function is executed in privileged mode.

The second attribute is declared with a Service ID. In this case the compiler will emit code to store this ID in register R18 followed by an indirect call to `_exception_handler` using a `brki rD,0x8` instruction, and emit an `rtbd` instruction to return from the handler. It is necessary to override the `_exception_handler` function to call the appropriate handler based on the ID provided in R18.

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 Manual*.

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

MicroBlaze Instruction Set Architecture

This chapter provides a detailed guide to the Instruction Set Architecture of MicroBlaze™.

Notation

The symbols used throughout this chapter are defined in Table 5-1.

Table 5-1: **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
<	Less than comparison
< =	Less than or equal comparison
|	Signal choice

Table 5-1: **Symbol Notation *(Cont'd)***

Symbol	Meaning
sext(*x*)	Sign-extend *x*
Mem(*x*)	Memory location at address *x*
FSLx	AXI4-Stream interface x
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	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.

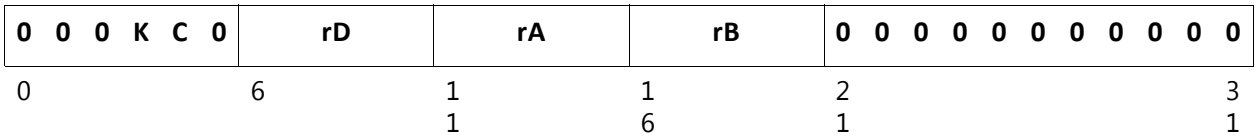
Opcode	Destination Reg	Source Reg A	Immediate Value
0	6	11	16 31

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

0 0 1 K C 0	rD	rA	IMM
0	6	1 1	1 6 3 1

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 218 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	1 1	1 6	2 1										3 1

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	1 1	1 6 3 1

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 218 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	1 1	1 6	2 1 3 1

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

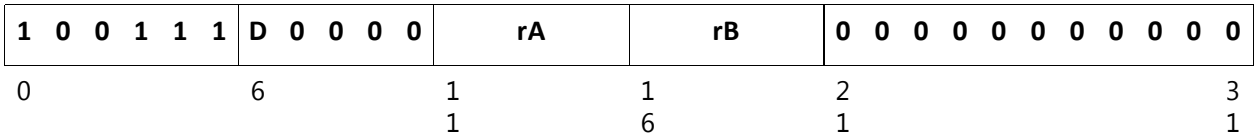
1 cycle

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 218 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

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

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 218 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

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

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

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

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)

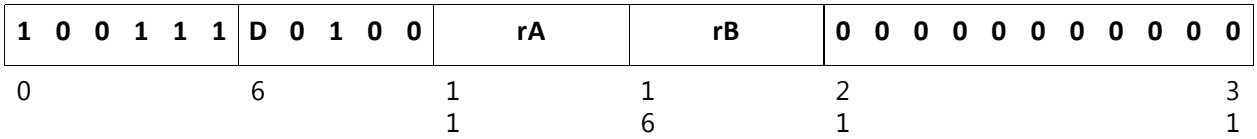
Note

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 218 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

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

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 218 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

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

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)

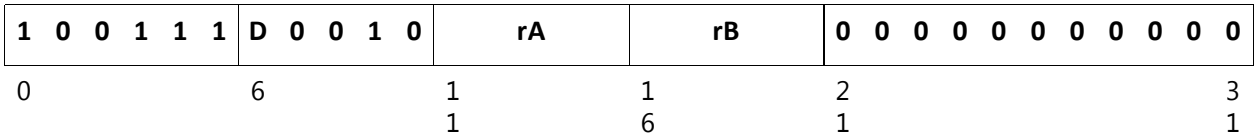
Note

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 218 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

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

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 218 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

1 0 0 1 1 1	D 0 0 0 1	rA	rB	0 0 0 0 0 0 0 0 0 0 0
0	6	1 1	1 6	2 1 3 1

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

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

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)

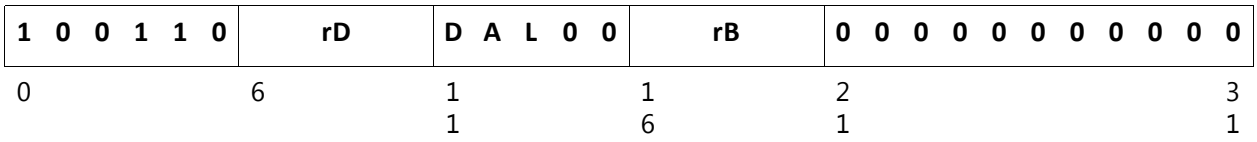
Note

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 218 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

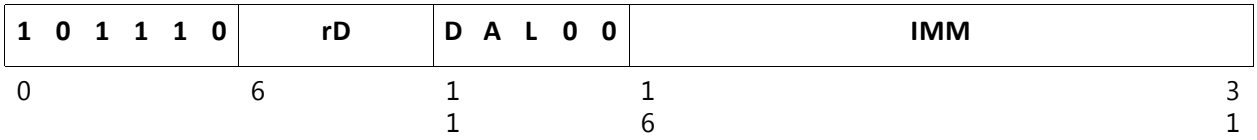
- 2 cycles (if the D bit is set)
- 3 cycles (if the D bit is not set)

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, C_BASE_VECTORS+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 = C_BASE_VECTORS+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 218 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
```

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

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

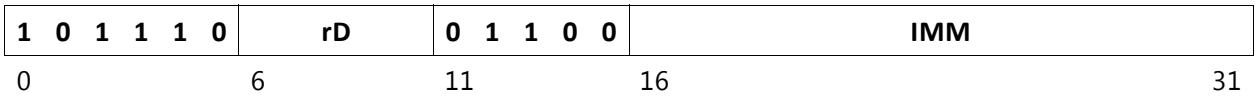
- rD
- PC
- MSR[BIP]
- ESR[EC], in case a privileged instruction exception is generated

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, C_BASE_VECTORS+0x8" or "brki rD, C_BASE_VECTORS+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, C_BASE_VECTORS+0x8" or "brki rD, C_BASE_VECTORS+0x18" is used to perform a Software Break, the Machine Status Register bits User Mode and Virtual Mode are cleared.

Pseudocode

```
if MSR[UM] and IMM ≠ C_BASE_VECTORS+0x8 and IMM ≠ C_BASE_VECTORS+0x18 then
  ESR[EC] ← 00111
else
  (rD) ← PC
  PC ← sext(IMM)
  MSR[BIP] ← 1
  Reservation ← 0
  if IMM = C_BASE_VECTORS+0x8 or IMM = C_BASE_VECTORS+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

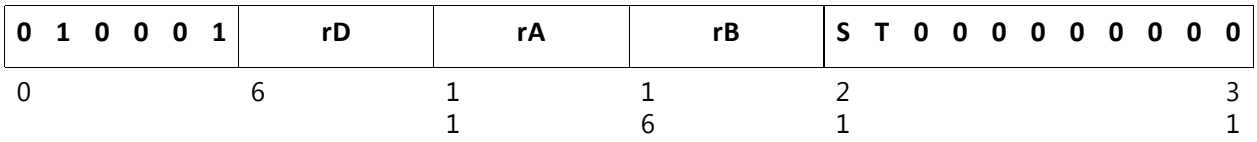
Note

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 218 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, irrespective of the value of C_BASE_VECTORS, 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

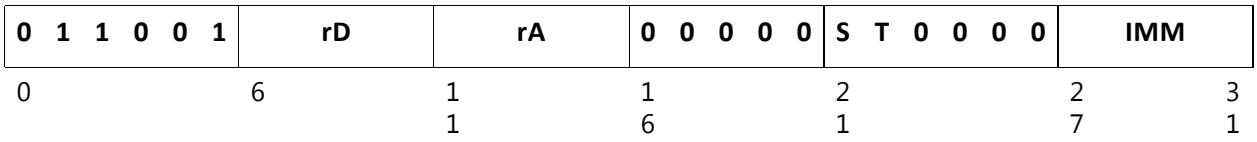
- 1 cycle with `C_AREA_OPTIMIZED=0`
- 2 cycles with `C_AREA_OPTIMIZED=1`

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)

0 0 0 1 0 1	rD	rA	rB	0 0 0 0 0 0 0 0 0 0 U 1
0	6	1 1	1 6	2 1 3 1

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

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

Registers Altered

- rD

Latency

- 1 cycle

fadd Floating Point Arithmetic Add

fadd rD, rA, rB Add

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

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

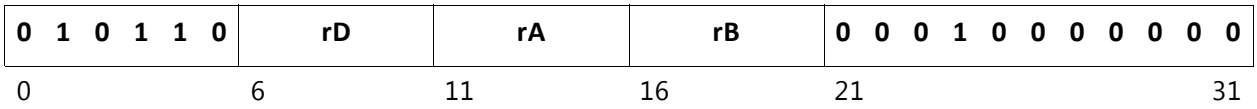
- 4 cycles with `C_AREA_OPTIMIZED=0`
- 6 cycles with `C_AREA_OPTIMIZED=1`

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

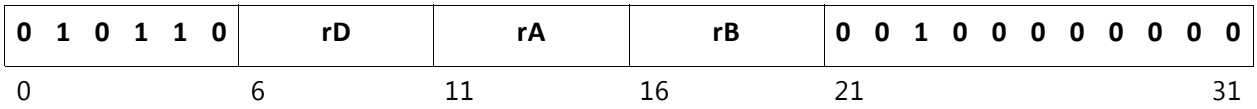
- 4 cycles with `C_AREA_OPTIMIZED=0`
- 6 cycles with `C_AREA_OPTIMIZED=1`

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



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

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

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

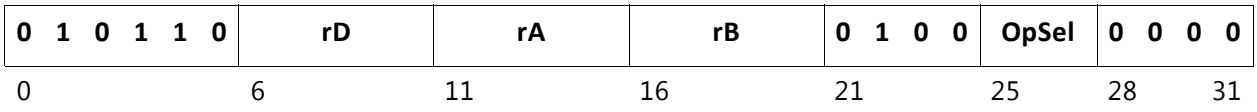
- 28 cycles with `C_AREA_OPTIMIZED=0`, 30 cycles with `C_AREA_OPTIMIZED=1`

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 5-2}
```

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.

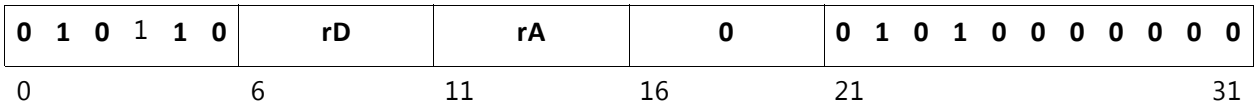
Table 5-2, page 209 lists the floating point comparison operations.

Table 5-2: **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



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

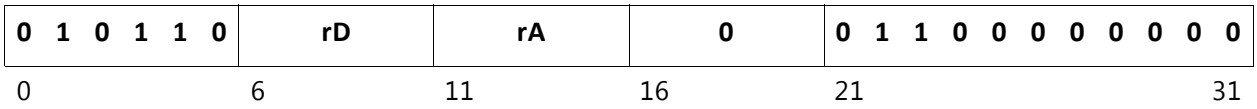
- 4 cycles with `C_AREA_OPTIMIZED=0`
- 6 cycles with `C_AREA_OPTIMIZED=1`

Note

This instruction is only available when the MicroBlaze parameter C_USE_FPU is set to 2 (Extended).

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 (Extended).

fsqrt Floating Point Arithmetic Square Root

fsqrt rD, rA Square Root

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

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

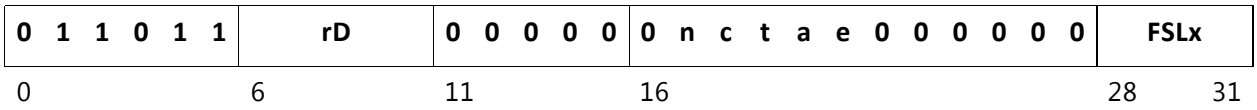
- 27 cycles with `C_AREA_OPTIMIZED=0`
- 29 cycles with `C_AREA_OPTIMIZED=1`

Note

This instruction is only available when the MicroBlaze parameter C_USE_FPU is set to 2 (Extended).

get get from stream interface

tneaget	rD, FSLx	get data from link x t = test-only n = non-blocking e = exception if control bit set a = atomic
tnecaget	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. If the available number of links set by C_FSL_LINKS is less than or equal to FSLx, link 0 is used.

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] 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] 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 ← FSLx
  if x >= C_FSL_LINKS then
    x ← 0
  (rD) ← Sx_AXIS_TDATA
  if (n = 1) then
    MSR[Carry]  ← Sx_AXIS_TVALID
  if  Sx_AXIS_TLAST ≠ c and Sx_AXIS_TVALID then
    MSR[FSL] ← 1
    if (e = 1) then
      ESR[EC] ← 00000
      ESR[ESS]← instruction bits [28:31]
      EDR     ← Sx_AXIS_TDATA
```

Registers Altered

- rD, unless an exception is generated, in which case the register is unchanged
- MSR[FSL]
- 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.

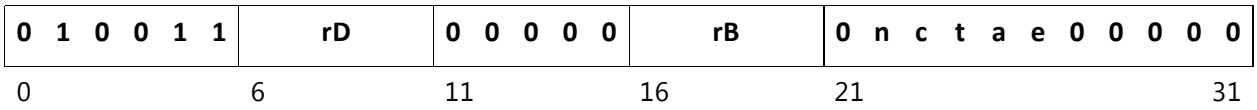
These instructions are only available when the MicroBlaze parameter C_FSL_LINKS is greater than 0.

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

tneagetd	rD, rB	get data from link rB[28:31] t = test-only n = non-blocking e = exception if control bit set a = atomic
tnecagetd	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. If the available number of links set by C_FSL_LINKS is less than or equal to the four least significant bits in rB, link 0 is used.

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] 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] 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]
  if x >= C_FSL_LINKS then
    x ← 0
  (rD) ← Sx_AXIS_TDATA
  if (n = 1) then
    MSR[Carry]  ← Sx_AXIS_TVALID
  if Sx_AXIS_TLAST ≠ c and Sx_AXIS_TVALID then
    MSR[FSL] ← 1
    if (e = 1) then
      ESR[EC] ← 00000
      ESR[ESS]← rB[28:31]
      EDR     ← Sx_AXIS_TDATA
```

Registers Altered

- rD, unless an exception is generated, in which case the register is unchanged
- MSR[FSL]
- 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)

0 1 0 0 1 0	rD	rA	rB	0 0 0 0 0 0 0 0 0 0 U 0
0	6	1 1	1 6	2 1 3 1

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 (divide by zero), 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 (divide overflow), 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 divide by zero or divide overflow occurs
- ESR[EC], if divide by zero or divide overflow occurs

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

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

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

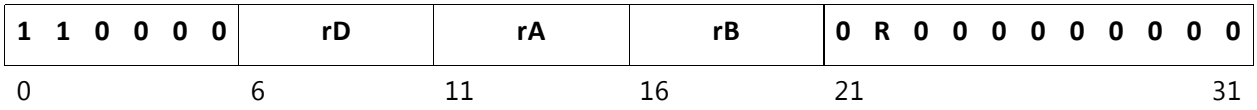
Notes

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 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

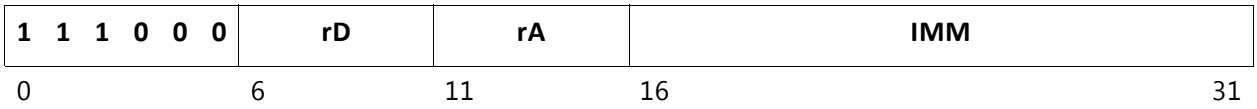
- 1 cycle with `C_AREA_OPTIMIZED=0`
- 2 cycles with `C_AREA_OPTIMIZED=1`

Note

The byte reversed instruction is only valid if MicroBlaze is configured to use reorder instructions (C_USE_REORDER_INSTR = 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

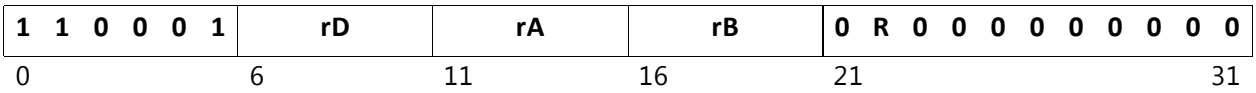
- 1 cycle with `C_AREA_OPTIMIZED=0`
- 2 cycles with `C_AREA_OPTIMIZED=1`

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 218 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 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

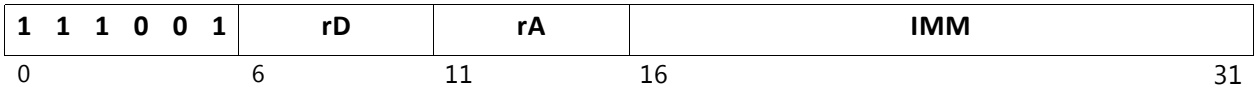
- 1 cycle with `C_AREA_OPTIMIZED=0`
- 2 cycles with `C_AREA_OPTIMIZED=1`

Note

The halfword reversed instruction is only valid if MicroBlaze is configured to use reorder instructions (C_USE_REORDER_INSTR = 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 218 for details on using 32-bit immediate values.

lw Load Word

lw	rD, rA, rB
lwr	rD, rA, rB

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

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 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

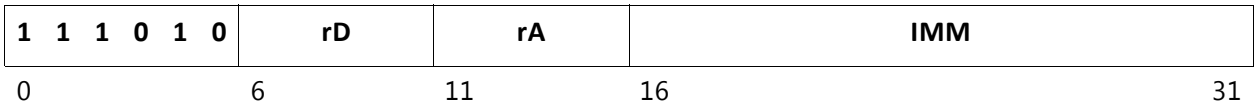
- 1 cycle with `C_AREA_OPTIMIZED=0`
- 2 cycles with `C_AREA_OPTIMIZED=1`

Note

The word reversed instruction is only valid if MicroBlaze is configured to use reorder instructions (C_USE_REORDER_INSTR = 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 218 for details on using 32-bit immediate values.

lwx Load Word Exclusive

lwx rD, rA, rB

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

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(Addr) and AXI_Response ≠ EXOKAY and MSR[EE] 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(Addr) and 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`

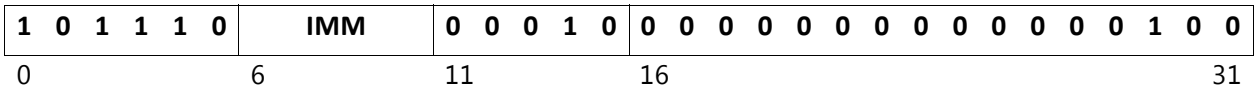
Note

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 MSRCLR, MSRSET, MTS, or 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 first use an MBAR instruction to wait for data accesses, which can be done by setting IMM to 1, and then use another MBAR instruction to clear the Branch Target Cache and empty the instruction prefetch buffer, which can be done by setting IMM to 2.

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.

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

When the most significant bit in IMM is set to 1 and no exception occurs, MicroBlaze enters sleep mode after all outstanding accesses have been completed, and sets the Sleep output signal to indicate this. The pipeline is halted, and MicroBlaze will not continue execution until a bit in the Wakeup input signal is asserted.

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
if (IMM & 16) = 16 then
  enter sleep mode
```

Registers Altered

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

Latency

- 1 + N cycles, where N is the number of cycles to wait for memory accesses to complete

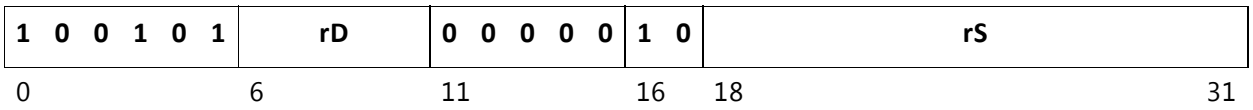
Notes

This instruction must not be preceded by an imm instruction, and must not be placed in a delay slot.

The assembler pseudo-instruction sleep can be used instead of “mbar 16” to enter sleep mode.

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 12)
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 - rpvr12 for PVR0 - PVR12.

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.

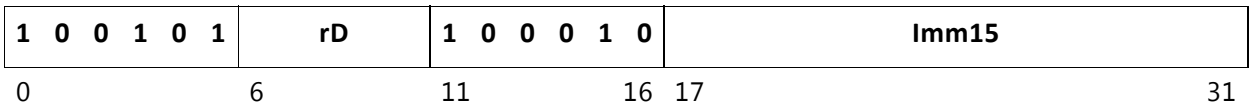
PID, ZPR, TLBLO and TLBHI are only valid as operands when the parameter C_USE_MMU > 1 (User Mode) and the parameter C_MMU_TLB_ACCESS = 1 (Read) or 3 (Full).

TLBX is only valid as operand when the parameter C_USE_MMU > 1 (User Mode) and the parameter C_MMU_TLB_ACCESS > 0 (Minimal).

PVR0 is only valid as an operand when C_PVR is 1 (Basic) or 2 (Full), and PVR1 - PVR12 are only valid as operands when C_PVR is set to 2 (Full).

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. This applies to both the value copied to register rD and the changed MSR value itself.

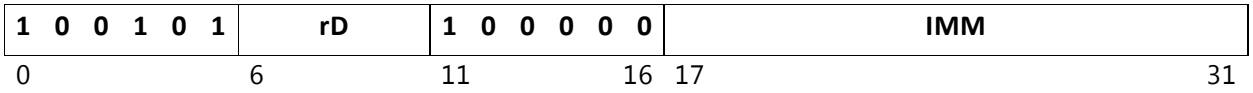
The immediate values has to be less than 215 when C_USE_MMU >= 1 (User Mode), and less than 214 otherwise. Only bits 17 to 31 of the MSR can be cleared when C_USE_MMU >= 1 (User Mode), and.bits 18 to 31 otherwise.

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.

With low-latency interrupt mode (C_USE_INTERRUPT = 2), the Interrupt_Ack output port is set to 11 if the MSR{IE] bit is set by executing this instruction.

Pseudocode

```
if MSR[UM] = 1 and IMM ≠ 0x4 then
  ESR[EC] ← 00111
else
  (rD)   ← (MSR)
  (MSR)  ← (MSR) ∨ (IMM)
  if (IMM) & 2
    Interrupt_Ack ← 11
```

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. This applies to both the value copied to register rD and the changed MSR value itself.

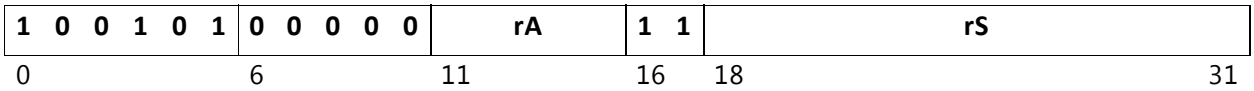
The immediate values has to be less than 215 when C_USE_MMU >= 1 (User Mode), and less than 214 otherwise. Only bits 17 to 31 of the MSR can be set when C_USE_MMU >= 1 (User Mode), and.bits 18 to 31 otherwise.

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.

With low-latency interrupt mode (C_USE_INTERRUPT = 2), the Interrupt_Ack output port is set to 11 if the MSR{IE] bit is set by executing this instruction.

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)
  if (rS) = 0x0001 and (rA) & 2
    Interrupt_Ack  ← 11
```

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 - PVR12 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 (User Mode) and the parameter C_MMU_TLB_ACCESS > 1 (Read). TLBLO, TLBHI and TLBX are only valid as destinations when the parameter C_USE_MMU > 1 (User Mode).

When changing MSR[VM] or PID the instruction must always be followed by a synchronizing branch instruction, for example BRI 4.

After writing to TLBHI in order to invalidate one or more UTLB entries, an MBAR 1 instruction must be issued to ensure that coherency is preserved in a coherent multi-processor system.

mul Multiply

mul rD, rA, rB

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

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
```

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

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 (Mul64).

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
```

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

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 (Mul64).

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
```

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

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 (Mul64).

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 218 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

Note

The assembler pseudo-instruction nop is implemented as “or r0, r0, r0”.

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 218 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	11	16	21 31

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	11	16	21 31

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

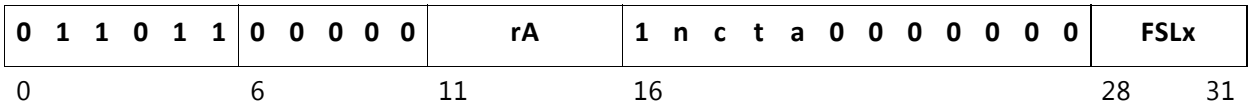
- 1 cycle

Note

This instruction is only available when the parameter C_USE_PCMP_INSTR is set to 1.

put Put to stream interface

naput	rA, FSLx	put data to link x n = non-blocking a = atomic
tnaput	FSLx	put data to link x test-only n = non-blocking a = atomic
ncaput	rA, FSLx	put control to link x n = non-blocking a = atomic
tncaput	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. If the available number of links set by C_FSL_LINKS is less than or equal to FSLx, link 0 is used.

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
  x ← FSLx
  if x >= C_FSL_LINKS then
    x ← 0
  Mx_AXIS_TDATA ← (rA)
  if (n = 1) then
    MSR[Carry] ← Mx_AXIS_TVALID ∧ Mx_AXIS_TREADY
  Mx_AXIS_TLAST ← C
```

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.

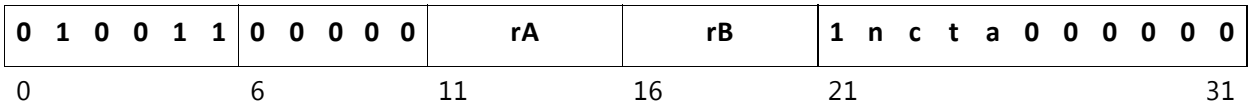
These instructions are only available when the MicroBlaze parameter C_FSL_LINKS is greater than 0.

The extended instructions (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.

putd Put to stream interface dynamic

naputd	rA, rB	put data to link rB[28:31] n = non-blocking a = atomic
tnaputd	rB	put data to link rB[28:31] test-only n = non-blocking a = atomic
ncaputd	rA, rB	put control to link rB[28:31] n = non-blocking a = atomic
tncaputd	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. If the available number of links set by C_FSL_LINKS is less than or equal to the four least significant bits in rB, link 0 is used.

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]
  if x >= C_FSL_LINKS then
    x ← 0
  Mx_AXIS_TDATA ← (rA)
  if (n = 1) then
    MSR[Carry]  ← Mx_AXIS_TVALID ∧ Mx_AXIS_TREADY
  Mx_AXIS_TLAST ← C
```

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

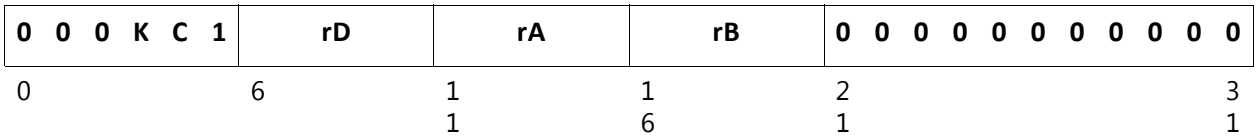
The blocking versions of this instruction should not be placed in a delay slot, since this prevents interrupts from being served.

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

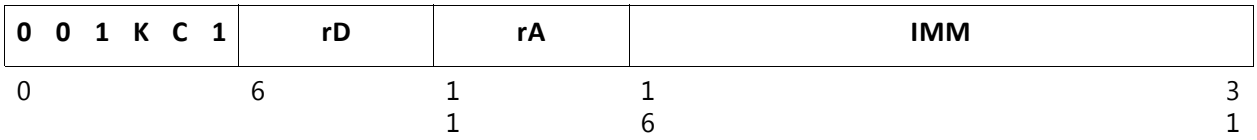
- 1 cycle

Notes

In subtractions, Carry = (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

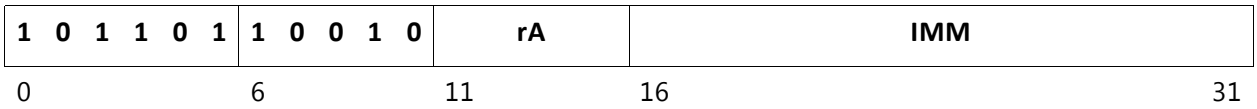
- 1 cycle

Notes

In subtractions, Carry = (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 218 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

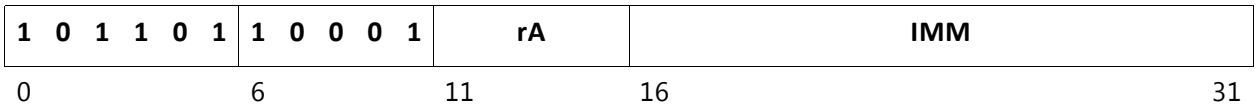
Note

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.

With low-latency interrupt mode (C_USE_INTERRUPT = 2), the Interrupt_Ack output port is set to 10 when this instruction is executed, and subsequently to 11 when the MSR{IE] bit is set.

Pseudocode

```
if MSR[UM] = 1 then
  ESR[EC] ← 00111
else
  PC ← (rA) + sext(IMM)
  Interrupt_Ack ← 10
  allow following instruction to complete execution
  MSR[IE] ← 1
  MSR[UM] ← MSR[UMS]
  MSR[VM] ← MSR[VMS]
  Interrupt_Ack ← 11
```

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

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

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)

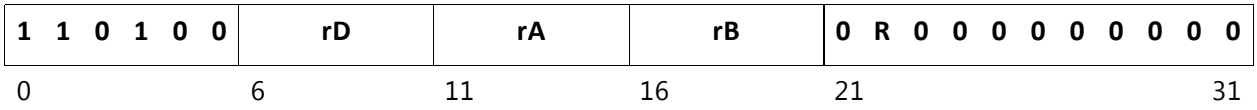
Note

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 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

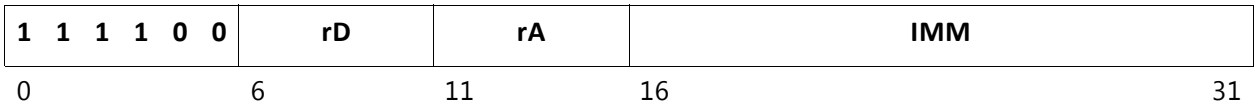
- 1 cycle with `C_AREA_OPTIMIZED=0`
- 2 cycles with `C_AREA_OPTIMIZED=1`

Note

The byte reversed instruction is only valid if MicroBlaze is configured to use reorder instructions (C_USE_REORDER_INSTR = 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 218 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

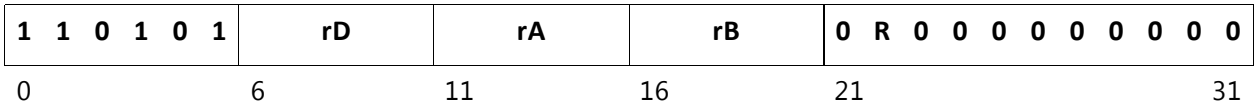
- rD

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 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

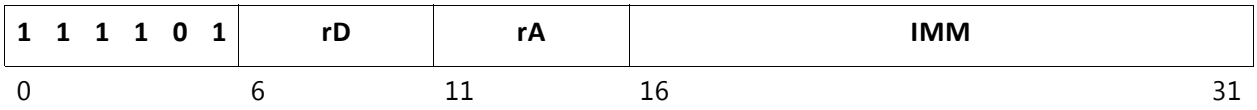
- 1 cycle with `C_AREA_OPTIMIZED=0`
- 2 cycles with `C_AREA_OPTIMIZED=1`

Note

The halfword reversed instruction is only valid if MicroBlaze is configured to use reorder instructions (C_USE_REORDER_INSTR = 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 218 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 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	11	16 31

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	11	16 31

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

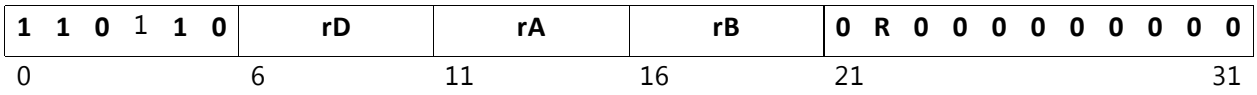
- rD
- MSR[C]

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 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`

Note

The word reversed instruction is only valid if MicroBlaze is configured to use reorder instructions (C_USE_REORDER_INSTR = 1).

swapb Swap Bytes

```
swapb    rD, rA
```

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

Description

Swaps the contents of register rA treated as four bytes, and places the result in rD. This effectively converts the byte sequence in the register between endianness formats, either from little-endian to big-endian or vice versa.

Pseudocode

```
(rD)[24:31] ← (rA)[0:7]
(rD)[16:23] ← (rA)[8:15]
(rD)[8:15]  ← (rA)[16:23]
(rD)[0:7]   ← (rA)[24:31]
```

Registers Altered

- rD

Latency

- 1 cycle

Note

This instruction is only valid if MicroBlaze is configured to use reorder instructions (C_USE_REORDER_INSTR = 1).

swaph Swap Halfwords

```
swaph    rD, rA
```

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

Description

Swaps the contents of register rA treated as two halfwords, and places the result in rD. This effectively converts the two halfwords in the register between endianness formats, either from little-endian to big-endian or vice versa.

Pseudocode

```
(rD)[0:15]  ← (rA)[16:31]
(rD)[16:31] ← (rA)[0:15]
```

Registers Altered

- rD

Latency

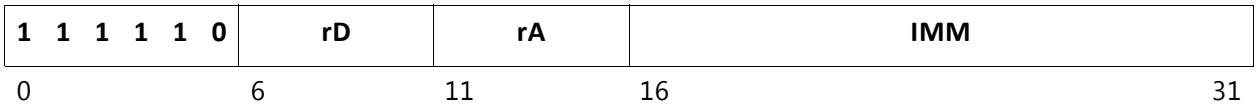
- 1 cycle

Note

This instruction is only valid if MicroBlaze is configured to use reorder instructions (C_USE_REORDER_INSTR = 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 218 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(Addr) and 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`

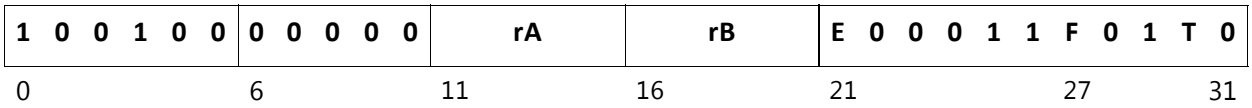
Note

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 MSRCLR, MSRSET, MTS, or 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
wdc.ext.flush	rA,rB
wdc.ext.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, wdc.clear is used to set the T bit, wdc.ext.flush is used to set the E, F and T bits, and wdc.ext.clear is used to set the E and T bits.

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. The E bit is not taken into account.

When C_DCACHE_USE_WRITEBACK is cleared to 0, the instruction will invalidate the cache line if the E bit is not set. Register rA contains the address of the affected cache line, and the register rB value is not used. If the E bit is set to 1, MicroBlaze will request that the matching address in an external cache should be invalidated or flushed, depending on the value of the F bit. The E bit is only taken into account when the parameter C_INTERCONNECT is set to 3 (ACE).

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 E = 0 then
    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 & 0xffffffe0
    if C_DCACHE_LINE_LEN = 16 then
      cacheline_mask ← (1 << log2(C_DCACHE_BYTE_SIZE) - 6) - 1
      cacheline      ← (DCache Line)[(address >> 6) ∧ cacheline_mask]
      cacheline_addr ← address & 0xffffffc0
```

```
    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
  if E = 1 then
    if F = 1 then
      request external cache flush with address
    else
      request external cache invalidate with address
```

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

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

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
  if C_ICACHE_LINE_LEN = 16 then
    cacheline_mask ← (1 << log2(C_CACHE_BYTE_SIZE) - 6) - 1
    (ICache Line)[((Ra) >> 6) ∧ cacheline_mask].Tag ← 0
```

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 the virtual address when the parameter C_USE_MMU = 3 (VIRTUAL) and the MMU is in virtual mode, otherwise it is the physical address.

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 218 for details on using 32-bit immediate values.

Performance and Resource Utilization

Performance

Performance characterization of this core has been done using the margin system methodology. The details of the margin system characterization methodology is described in "IP Characterization and fMAX Margin System Methodology", in the *Vivado Design Suite User Guide: Designing With IP* (UG986).

Maximum Frequencies

The maximum frequencies for the MicroBlaze core are provided in Table A-1.

Note: Zynq®-7000 results are expected to be similar to 7 series results.

Table A-1: **Maximum Frequencies**

Family	F_{max} (MHz)
Virtex®-7	373
Kintex®-7	385
Artix®-7	254
Virtex UltraScale™	458
Kintex UltraScale	430

Resource Utilization

The MicroBlaze core resource utilization for various parameter configurations are measured with Virtex-7 (Table A-2), Kintex-7 (Table A-3), Artix-7 (Table A-4), Virtex UltraScale (Table A-5), and Kintex UltraScale (Table A-6) devices.

Note: Zynq®-7000 results are expected to be similar to 7 series results.

The parameter values for each of the measured configurations are shown in Table A-7. The configurations directly correspond to the predefined templates in the MicroBlaze Configuration Wizard.

Table A-2: **Device Utilization - Virtex-7 FPGAs (XC7VX485T ffg1761-3)**

Configuration	Device Resources		
	LUTs	FFs	F_{max} (MHz)
Minimum Area	663	228	362
Maximum Performance	3863	3063	223
Maximum Frequency	909	546	362
Linux with MMU	3646	3223	225
Low-end Linux with MMU	3121	2588	219
Typical	2066	1777	244

Table A-3: **Device Utilization - Kintex-7 FPGAs (XC7K325T ffg900-3)**

Configuration	Device Resources		
	LUTs	FFs	F_{max} (MHz)
Minimum Area	654	220	355
Maximum Performance	3972	3063	212
Maximum Frequency	912	546	355
Linux with MMU	3668	3213	221
Low-end Linux with MMU	3101	2576	219
Typical	2064	1777	237

Table A-4: **Device Utilization - Artix-7 FPGAs (XC7A200T fbg676-3)**

Configuration	Device Resources		
	LUTs	FFs	F_{max} (MHz)
Minimum Area	648	214	234
Maximum Performance	3879	3064	154
Maximum Frequency	906	545	234
Linux with MMU	3653	3218	142
Low-end Linux with MMU	3108	2582	153
Typical	2070	1777	190

Table A-5: **Device Utilization - Virtex UltraScale FPGAs (XCVU095 ffvd1924-3)**

Configuration	Device Resources		
	LUTs	FFs	F_{max} (MHz)
Minimum Area	584	214	404
Maximum Performance	3923	3061	263
Maximum Frequency	908	545	404
Linux with MMU	3618	3204	251
Low-end Linux with MMU	3086	2575	254
Typical	2052	1776	320

Table A-6: **Device Utilization - Kintex UltraScale FPGAs (XCKU040 ffva1156-3)**

Configuration	Device Resources		
	LUTs	FFs	F_{max} (MHz)
Minimum Area	576	217	410
Maximum Performance	3938	3061	272
Maximum Frequency	906	545	410
Linux with MMU	3613	3204	241
Low-end Linux with MMU	3080	2575	244
Typical	2062	1776	285

Table A-7: **Parameter Configurations**

Parameter	Configuration Parameter Values					
	Minimum Area	Maximum Performance	Maximum Frequency	Linux with MMU	Low-end Linux with MMU	Typical
C_ALLOW_DCACHE_WR	1	1	1	1	1	1
C_ALLOW_ICACHE_WR	1	1	1	1	1	1
C_AREA_OPTIMIZED	1	0	0	0	0	0
C_CACHE_BYTE_SIZE	4096	32768	4096	16384	8192	8192
C_DCACHE_BYTE_SIZE	4096	32768	4096	16384	8192	8192
C_DCACHE_LINE_LEN	4	8	4	4	4	4
C_DCACHE_USE_WRITEBACK	0	1	0	0	0	0
C_DEBUG_ENABLED	0	1	0	1	1	1
C_DIV_ZERO_EXCEPTION	0	0	0	1	0	0
C_M_AXI_D_BUS_EXCEPTION	0	0	0	1	1	1
C_FPU_EXCEPTION	0	0	0	0	0	0
C_FSL_EXCEPTION	0	0	0	0	0	0
C_FSL_LINKS	0	0	1	0	0	0
C_ICACHE_LINE_LEN	4	8	4	8	4	8
C_ILL_OPCODE_EXCEPTION	0	0	0	1	1	0
C_M_AXI_I_BUS_EXCEPTION	0	0	0	1	1	0
C_MMU_DTLB_SIZE	2	4	2	4	4	4
C_MMU_ITLB_SIZE	1	2	1	2	2	2
C_MMU_TLB_ACCESS	3	3	3	3	3	3
C_MMU_ZONES	2	2	2	2	2	2
C_NUMBER_OF_PC_BRK	0	1	1	1	1	2
C_NUMBER_OF_RD_ADDR_BRK	0	0	0	0	0	0
C_NUMBER_OF_WR_ADDR_BRK	0	0	0	0	0	0
C_OPCODE_0x0_ILLEGAL	0	0	0	1	1	0
C_PVR	0	0	0	2	0	0
C_UNALIGNED_EXCEPTIONS	0	0	0	1	1	0
C_USE_BARREL	0	1	0	1	1	1
C_USE_DCACHE	0	1	0	1	1	1
C_USE_DIV	0	1	0	1	0	0
C_USE_EXTENDED_FSL_INSTR	0	0	0	0	0	0

Table A-7: **Parameter Configurations *(Cont'd)***

Parameter	Configuration Parameter Values					
	Minimum Area	Maximum Performance	Maximum Frequency	Linux with MMU	Low-end Linux with MMU	Typical
C_USE_FPU	0	2	0	0	0	0
C_USE_HW_MUL	0	2	0	2	1	1
C_USE_ICACHE	0	1	0	1	1	1
C_USE_MMU	0	0	0	3	3	0
C_USE_MSR_INSTR	0	1	0	1	1	1
C_USE_PCMP_INSTR	0	1	0	1	1	1
C_USE_REORDER_INSTR	0	1	1	1	1	1
C_USE_BRANCH_TARGET_CACHE	0	1	0	0	0	0
C_BRANCH_TARGET_CACHE_SIZE	0	0	0	0	0	0
C_ICACHE_STREAMS	0	1	0	1	0	0
C_ICACHE_VICTIMS	0	8	0	8	0	0
C_DCACHE_VICTIMS	0	8	0	8	0	0
C_ICACHE_FORCE_TAG_LUTRAM	0	0	0	0	0	0
C_DCACHE_FORCE_TAG_LUTRAM	0	0	0	0	0	0
C_ICACHE_ALWAYS_USED	0	1	0	1	1	0
C_DCACHE_ALWAYS_USED	0	1	0	1	1	0
C_D_AXI	0	1	0	1	1	0
C_USE_INTERRUPT	0	0	0	1	1	0

Appendix B

Additional Resources and Legal Notices

Xilinx Resources

For support resources such as Answers, Documentation, Downloads, and Forums, see Xilinx Support.

Solution Centers

See the Xilinx Solution Centers for support on devices, software tools, and intellectual property at all stages of the design cycle. Topics include design assistance, advisories, and troubleshooting tips.

References

The following documents are available via your Vivado installation.

Relevant individual documents are linked below.

1. *Vivado Design Suite User Guide: Designing With IP* (UG896)
2. *Vivado Design Suite User Guide: Designing IP Subsystems Using IP Integrator* (UG994)
3. *Vivado Design Suite User Guide: Embedded Processor Hardware Design* (UG898)
4. *Xilinx Software Development Kit Help* (UG782)
5. *Embedded System Tools Reference Manual* (UG1043)
6. *PowerPC Processor Reference Guide* (UG011)
7. *AMBA 4 AXI4-Stream Protocol Specification, Version 1.0* (ARM IHI 0051A)
8. *AMBA AXI and ACE Protocol Specification* (ARM IHI 0022E)
9. *MicroBlaze Debug Module (MDM) Product Guide* (PG115)

10. *LogiCore IP Soft Error Mitigation Controller* (PG036)
11. *Device Reliability Report* (UG116)
12. *LogiCore IP Processor LMB BRAM Interface Controller* (PG112)
13. *Hierarchical Design Methodology Guide* (UG748)

The following lists additional resources you can access directly using the provided URLs.

14. The entire set of GNU manuals:
    http://www.gnu.org/manual
15. IEEE 754-1985 standard
    http://en.wikipedia.org/wiki/IEEE_754-1985

Training Resources

Xilinx provides a variety of QuickTake videos and training courses to help you learn more about the concepts presented in this document. Use these links to explore related training resources:

1. Vivado Design Suite QuickTake Video: Creating IP Subsystems with Vivado IP Integrator
2. Vivado Design Suite QuickTake Video: IP Integrator Advanced User Tips
3. Vivado Design Suite QuickTake Video Tutorials
4. Essentials of FPGA Design Training Course
5. Vivado Design Suite Tool Flow Training Course
6. Vivado Design Suite Embedded Systems Design
7. Vivado Design Suite Advanced Embedded Systems Design
8. Embedded Systems Software Design

Please Read: Important Legal Notices

terms contained in a license issued to you by Xilinx. Xilinx products are not designed or intended to be fail-safe or for use in any application requiring fail-safe performance; you assume sole risk and liability for use of Xilinx products in such critical applications, please refer to Xilinx's Terms of Sale which can be viewed at http://www.xilinx.com/legal.htm#tos.

Automotive Applications Disclaimer

XILINX PRODUCTS ARE NOT DESIGNED OR INTENDED TO BE FAIL-SAFE, OR FOR USE IN ANY APPLICATION REQUIRING FAIL-SAFE PERFORMANCE, SUCH AS APPLICATIONS RELATED TO: (I) THE DEPLOYMENT OF AIRBAGS, (II) CONTROL OF A VEHICLE, UNLESS THERE IS A FAIL-SAFE OR REDUNDANCY FEATURE (WHICH DOES NOT INCLUDE USE OF SOFTWARE IN THE XILINX DEVICE TO IMPLEMENT THE REDUNDANCY) AND A WARNING SIGNAL UPON FAILURE TO THE OPERATOR, OR (III) USES THAT COULD LEAD TO DEATH OR PERSONAL INJURY. CUSTOMER ASSUMES THE SOLE RISK AND LIABILITY OF ANY USE OF XILINX PRODUCTS IN SUCH APPLICATIONS.