IBM

SPU Assembly Language Specification

Version 1.7

CBEA JSRE Series
Cell Broadband Engine Architecture
Joint Software Reference
Environment Series

July 18, 2008

IBM Systems and Technology Group
2070 Route 52, Bldg. 330
Hopewell Junction, NY 12533-6351

The IBM home page can be found at **ibm.com**

The IBM semiconductor solutions home page can be found at **ibm.com**/chips

July 18, 2008

Table of Contents

List of Tables

About This Document

This document describes the Synergistic Processor Unit (SPU) assembly-language syntax for a processor compliant with the Cell Broadband Engine™ Architecture (CBEA).

Audience

The document is intended for system and application programmers who desire to write assembly language programs for the SPU.

Version History

This section describes significant changes made to each version of this document.

Version Number & Date	Changes
v. 1.7 July 18, 2008	Made editorial changes to mnemonic descriptions (TWG_RFC00125-3: CORRECTION NOTICE).
v. 1.6 September 3, 2007	Corrected miscellaneous documentation errors (TWG_RFC00102-1: CORRECTION NOTICE).
v. 1.5 March 8, 2007	Added several new double precision floating-point instructions (TWG_RFC00071-0). Corrected document version numbers for related documentation (TWG_RGC00093-0).
v. 1.4 October 11, 2006	Changed several operands from `rt` to `rc` in the SPU Assembler Instructions table (TWG_RFC00049-0: CORRECTION NOTICE and jsre-tool messages 00468 and 00488). The description of the `wrch` instruction in the SPU Assembler Instructions table was corrected. Applied changes made in TWG_RFC00061-1 and TWG_RFC00062-0.
v. 1.3 October 20, 2005	Changed “Broadband Processor Architecture” to “Cell Broadband Engine Architecture”, and changed “BPA” to “CBEA” (TWG_RFC00037-0: CORRECTION NOTICE). Deleted several references to BE revisions DD1.0 and DD2.0 (TWG_RFC00040-0: CORRECTION NOTICE).
v. 1.2 July 13, 2005	Deleted several sections in the “About This Document” chapter (TWG_RFC00032-0: CORRECTION NOTICE). Corrected several documentation errors; for example, in several descriptions in the SPU Assembler Instructions table, the phrase “halfword element `rt`” was changed to “halfword element 1 of register `rt`” (TWG_RFC00033-0: CORRECTION NOTICE).
v. 1.1 June 10, 2005	Changed “Broadband Engine” or “BE” to “a processor compliant with the Broadband Processor Architecture” or “a processor compliant with BPA”; and changed Synergistic Processing Unit to Synergistic Processor Unit. Defined a PPU as a PowerPC Processor Unit on first major instance. Corrected several book references and changed the copyright page so that trademark owners were specified. (All changes per TWG_RFC00031-0: CORRECTION NOTICE.) Made miscellaneous changes to the “About This Document” section.
v. 0.9 - 1.0	Not applicable. Version numbers were changed so that JSRE version numbers are in synchrony with those used by IBM in its public

Version Number & Date	Changes
	release.
v. 0.8 May 12, 2005	Changed PU to PPU; changed “PU-to-SPU” (mailboxes) and “SPU-to-PU” to “inbound” and “outbound” respectively (TWG_RFC00028-1: CORRECTION NOTICE). Updated channel names to coincide with BPA channel names (TWG_RFC00029-1).
v. 0.7 July 16, 2004	Removed all branch aliases from table of instruction aliases (TWG_RFC00009-0). Added an additional SPU instruction, `orx` (TWG_RFC00010-0). Added mnemonics for channels that support reading the event mask and tag mask (TWG_RFC00011-0). Removed operands from `hbrp` instruction and provided a new description of this instruction. Also removed it from a table (TWG_RFC00012-0). Made miscellaneous editorial changes.
v. 0.6 March 12, 2004	Made miscellaneous editorial changes.
v. 0.5 February 25, 2004	Changed formatting of document so that it reflects the typographic conventions mentioned in the “About This Document” section. Made minimal editorial changes.
v. 0.4 January 20, 2004	Changed document to new format, including front matter. Made miscellaneous editorial changes.
v. 0.3 August 31, 2003	Corrected PC-relative addressing style. Added low and high halfword address syntax. Added stopd instruction.
v. 0.2 May 13, 2003	Added isolation control channel. Replaced `aci`, `asc`, `sbi`, and `ssb` instructions with `addx`, `cg`, `cgx`, `sfx`, `bg`, `and` `bgx`.
v. 0.1 March 7, 2003	Initial release of this document.

Related Documentation

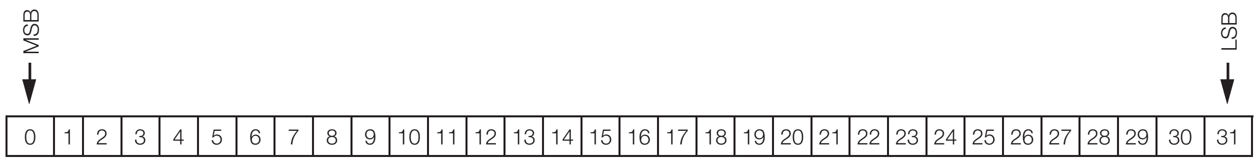
The following table provides a list of references and supporting materials for this document:

Document Title	Version	Date
PowerPC Architecture Book (three volumes) (public IBM documents)	2.02	November 2005
PowerPC Microprocessor Family: The Programming Environments for 32-Bit Microprocessors (PEM) (public IBM document)	1.0	February 2000
Cell Broadband Engine Architecture (public Cell document)	1.01	October 2006
Synergistic Processor Unit Instruction Set Architecture (SPU ISA) (public Cell document)	1.2	January 2007

Conventions Used in This Document

Bit Notation

Standard bit notation is used throughout this document. Bits and bytes are numbered in ascending order from left to right. Thus, for a 4-byte word, bit 0 is the most significant bit and bit 31 is the least significant bit, as shown in the following figure:



MSB = Most significant bit

LSB = Least significant bit

Notation for bit encoding is as follows:

- Hexadecimal values are preceded by `0x`. For example: `0x0A00`.
- Binary values in sentences appear in single quotation marks. For example: ‘1010’.

Other Conventions

The following typographic conventions are used throughout this document:

Convention	Meaning
`courier`	Indicates programming code and literals, such as processing instructions, register names, data types, events, and file names. Also indicates function and macro names. This convention is only used where it facilitates comprehension, especially in narrative descriptions.
`courier + italics`	Indicates arguments, parameters, and variables. This convention is only used where it facilitates comprehension, especially in narrative descriptions.
italics (without courier)	Indicates emphasis. Except when hyperlinked, book references are in italics. When a term is first defined, it is often in italics.
blue	Indicates a hyperlink (color printers or online only).

1. Introduction

This specification describes SPU assembly-language syntax and machine-dependent features for the GNU assembler (as). Although this specification focuses on the GNU assembler, this document might also serve as an example specification for other SPU assemblers.

2. Instruction Set and Instruction Syntax

2.1. Notation and Conventions

In this specification, lower case is used for all instructions, register aliases, and channels names; however, these tokens may also be expressed in upper or mixed case. Table 2-1 describes notations used in this specification.

Table 2-1: Notations and Conventions

Notation/Convention	Meaning
ch	Channel number. Channels are specified as either `$ch` followed by a channel number (for example, `$ch3`) or a specific channel mnemonic. See section "2.4. Channel Mnemonics" for a complete list of channel mnemonics.
ra, rb, rc	Source register. Registers are specified as a dollar symbol ($) followed by a register number from 0 to127. For example, $38 refers to register 38. See Table 2-3 for additional register aliases.
rt	Target register. Registers are specified as a dollar symbol ($) followed by a register number from 0 to127. For example, $38 refers to register 38. See Table 2-3 for additional register aliases.
s3, s6	3-bit or 6-bit signed value, respectively. Encoded as a 7-bit signed immediate in which only a subset of the bits is used.
s7	7-bit sign-extended value.
s10	10-bit sign-extended value.
s11	11-bit sign-extended value.
s14	14-bit sign-extended value.
s16	16-bit sign-extended value.
s18	Relative address computations.
scale7	7-bit scale exponent. Values range from 0 to 127.
spr	Special purpose register.
u3, u5, u6	3-bit, 5-bit, or 6-bit unsigned value, respectively. Encoded as a 7-bit unsigned immediate in which only a subset of the bits is used.
u7	Unsigned 7-bit value.
u14	Unsigned 14-bit value.
u16	Unsigned 16-bit value.
u18	Unsigned 18-bit value.

2.2. Instruction Set

This section provides an overview of the SPU instruction set and its syntax, including:

- Supported instructions and their syntax
- Supported data types
- Supported ranges for instruction parameters

For details about the specific machine instructions, see the *Synergistic Processor Unit Instruction Set Architecture* specification.

In Table 2-2, a short descriptive name is provided for each assembler instruction mnemonic, followed by a full description of the instruction (for example, the `bi ra` instruction is named “Branch indirect”, and this is followed by a complete description of the branch-indirect instruction). In a few instances, the words in a descriptive name do not match the letters in the corresponding mnemonic as they do in the SPU ISA document. The reason for this difference is to improve readability and understandability. In the descriptive names of floating-point instructions, single-precision floating point is abbreviated as “floating”, and double-precision floating point is abbreviated as “double floating”.

Table 2-2: SPU Assembler Instructions

Instruction/Usage	Description
a rt, ra, rb	Add word. Each word element of register `ra` is added to the corresponding word element of register `rb`, and the results are placed in the corresponding word elements of register `rt`.
absdb rt, ra, rb	Absolute difference of bytes. Each byte element of register `ra` is subtracted from the corresponding byte element of register `rb`. The absolute values of the results are placed in the corresponding elements of register `rt`.
addx rt, ra, rb	Add word extended. Each word element of register `ra`, the corresponding word element of register `rb`, and the least significant bit of the corresponding word element of register `rt` are added, and the results are placed in the corresponding word elements of register `rt`.
ah rt, ra, rb	Add halfword. Each halfword element of register `ra` is added to the corresponding halfword element of register `rb`, and the results are placed in the corresponding halfword elements of register `rt`.
ahi rt, ra, s10	Add halfword immediate. The sign-extended immediate value `s10` is added to each halfword element of register `ra`, and the results are placed in the corresponding halfword elements of register `rt`.
ai rt, ra, s10	Add word immediate. The sign-extended immediate value `s10` is added to each word elements of register `ra`, and the results are placed in the corresponding word elements of register `rt`.
and rt, ra, rb	And. The value of register `ra` is logically ANDed with register `rb`, and the result is placed in register `rt`.
andbi rt, ra, s10	And byte immediate. The 8 least significant bits of `s10` are logically ANDed with each byte element of register `ra`, and the results are placed in the corresponding elements of register `rt`.
andc rt, ra, rb	And with complement. The value of register `ra` is logically ANDed with the complement of register `rb`, and the result is placed in register `rt`.
andhi rt, ra, s10	And halfword immediate. The sign-extended immediate value `s10` is logically ANDed with each halfword element of register `ra`, and the results are placed in the corresponding elements of register `rt`.
andi rt, ra, s10	And word immediate. The sign-extended immediate value `s10` is logically ANDed with each word element of register `ra`, and the results are placed in the corresponding elements of register `rt`.
avgb rt, ra, rb	Average bytes. The corresponding byte elements of registers `ra` and `rb` are averaged (`(a+b+1) >> 1`), and the results are placed in the corresponding byte elements of register `rt`.
bg rt, ra, rb	Borrow generate word. Each unsigned word element of register `ra` is compared to the corresponding unsigned word element of `rb`. If the value of `ra` is greater than that of `rb`, a 0 is placed in the corresponding element of `rt`; otherwise, a 1 is placed there.

Instruction/Usage	Description
bgx rt, ra, rb	Borrow generate word extended. Each word element of register `ra` is subtracted from the corresponding word element of register `rb`. An additional 1 is subtracted from the result if the least significant bit of word element `rt` is 0. If the result is less than 0, a 0 is placed in the corresponding element of register `rt`; otherwise, a 1 is placed there.
bi ra	Branch indirect. Execution proceeds with the instruction at the address specified by word element 0 of register `ra`. The 2 least significant bits of the address are ignored.
bid ra	Branch indirect, disable. Execution proceeds with the instruction at the address specified by word element 0 of register `ra`, and interrupts are disabled. The 2 least significant bits of this address are ignored.
bie ra	Branch indirect, enable. Execution proceeds with the instruction at the address specified by word element 0 of register `ra`, and interrupts are enabled. The 2 least significant bits of the address are ignored.
bihnz rc, ra	Branch indirect if not zero halfword. If halfword element 1 of register `rc` is 0, execution proceeds with the next sequential instruction; otherwise, execution proceeds at the address in word element 0 of register `ra`. The 2 least significant bits of this address are ignored.
bihnzd rc, ra	Branch indirect if not zero halfword, disable. If halfword element 1 of register `rc` is 0, execution proceeds with the next sequential instruction; otherwise, the branch is taken, and execution proceeds at the address in word element 0 of register `ra`. The 2 least significant bits of this address are ignored. If the branch is taken, interrupts are disabled; otherwise, the interrupt enable state remains unchanged.
bihnze rc, ra	Branch indirect if not zero halfword, enable. If halfword element 1 of register `rc` is 0, execution proceeds with the next sequential instruction; otherwise, the branch is taken, and execution proceeds at the address in word element 0 of register `ra`. The 2 least significant bits of this address are ignored. If the branch is taken, interrupts are enabled; otherwise, the interrupt enable state remains unchanged.
bihz rc, ra	Branch indirect if zero halfword. If halfword element 1 of register `rc` is 0, execution proceeds at the address in word element 0 of register `ra`. The 2 least significant bits of this address are ignored. If halfword element 1 of register `rc` is nonzero, execution proceeds with the next sequential instruction.
bihzd rc, ra	Branch indirect if zero halfword, disable. If halfword element 1 of register `rc` is 0, the branch is taken, and execution proceeds at the address in word element 0 of register `ra`. The 2 least significant bits of the address in element 0 are ignored. If halfword element 1 of register `rc` is nonzero, execution proceeds with the next sequential instruction. Interrupts are disabled if the branch is taken; otherwise, the interrupt enable state remains unchanged.
bihze rc, ra	Branch indirect if zero halfword, enable. If halfword element 1 of register `rc` is 0, the branch is taken, and execution proceeds at the address in word element 0 of register `ra`. The 2 least significant bits of the address in element 0 are ignored. If halfword element 1 of register `rc` is nonzero, execution proceeds with the next sequential instruction. Interrupts are enabled if the branch is taken; otherwise, the interrupt enable state remains unchanged.
binz rc, ra	Branch indirect if not zero word. If word element 0 of register `rc` is 0, execution proceeds with the next sequential instruction; otherwise, execution proceeds at the address in word element 0 of register `ra`. The 2 least significant bits of this address are ignored.

Instruction/Usage	Description
binzd rc, ra	Branch indirect if not zero word, disable. If word element 0 of register `rc` is 0, execution proceeds with the next sequential instruction; otherwise, the branch is taken, and execution proceeds at the address in word element 0 of register `ra`. The 2 least significant bits of this address are ignored. If the branch is taken, interrupts are disabled; otherwise, the interrupt enable state remains unchanged.
binze rc, ra	Branch indirect if not zero word, enable. If word element 0 of register `rc` is 0, execution proceeds with the next sequential instruction; otherwise, the branch is taken, and execution proceeds at the address in word element 0 of register `ra`. The 2 least significant bits of this address are ignored. If the branch is taken, interrupts are enabled; otherwise, the interrupt enable state remains unchanged.
bisl rt, ra	Branch indirect and set link. The effective address of the next instruction is taken from word element 0 of register `ra`. The 2 least significant bits of this address are ignored. The address of the instruction following this instruction is placed into word element 0 of register `rt`, and all other word elements of `rt` are assigned a value of 0.
bisld rt, ra	Branch indirect and set link, disable. The effective address of the next instruction is taken from word element 0 of register `ra`. The 2 least significant bits of this address are ignored. The address of the instruction following this instruction is placed into word element 0 of register `rt`, and all other word elements of `rt` are assigned a value of 0. Interrupts are also disabled.
bisle rt, ra	Branch indirect and set link, enable. The effective address of the next instruction is taken from word element 0 of register `ra`. The 2 least significant bits of this address are ignored. The address of the instruction following this instruction is placed into word element 0 of register `rt`, and all other word elements of `rt` are assigned a value of 0. Interrupts are also enabled.
bisled rt, ra	Branch indirect and set link on external data. The address of the instruction following this instruction is placed in word element 0 of register `rt`, and all other elements of register `rt` are assigned a value of 0. If the count of channel 0 is nonzero, execution continues at the effective address in word element 0 of register `ra`. The 2 least significant bits of this address are ignored. If the count of channel 0 is zero, execution continues with the next sequential instruction.
bisledd rt, ra	Branch indirect and set link on external data, disable. The address of the instruction following this instruction is placed in word element 0 of register `rt`, and all other elements of register `rt` are assigned a value of 0. If the count of channel 0 is nonzero, the branch is taken, and execution continues at the effective address in word element 0 of register `ra`. The 2 least significant bits of this address are ignored. If the count of channel 0 is zero, execution continues with the next sequential instruction. If the branch is taken, interrupts are disabled; otherwise, the interrupt enable state remains unchanged.
bislede rt, ra	Branch indirect and set link on external data, enable. The address of the instruction following this instruction is placed in word element 0 of register `rt`, and all other elements of register `rt` are assigned a value of 0. If the count of channel 0 is nonzero, the branch is taken, and execution continues at the effective address in word element 0 of register `ra`. The 2 least significant bits of this address are ignored. If the count of channel 0 is zero, execution continues with the next sequential instruction. If the branch is taken, interrupts are enabled; otherwise, the interrupt enable state remains unchanged.
biz rc, ra	Branch indirect if zero word. If word element 0 of register `rc` is zero, execution proceeds at the effective address in word element 0 of register `ra`. The 2 least significant bits of this address are ignored. If word element 0 of `rc` is nonzero, execution proceeds with the next sequential instruction.

Instruction/Usage	Description
bizd rc, ra	Branch indirect if zero word, disable. If word element 0 of register `rc` is zero, the branch is taken, and execution proceeds at the effective address in word element 0 of register `ra`. The 2 least significant bits of this address are ignored. If word element 0 of `rc` is nonzero, execution proceeds with the next sequential instruction. If the branch is taken, interrupts are disabled; otherwise, the interrupt enable state remains unchanged.
bize rc, ra	Branch indirect if zero word, enable. If word element 0 of register `rc` is zero, the branch is taken, and execution proceeds at the effective address in word element 0 of register `ra`. The 2 least significant bits of this address are ignored. If word element 0 of `rc` is nonzero, execution proceeds with the next sequential instruction. If the branch is taken, interrupts are enabled; otherwise, the interrupt enable state remains unchanged.
br s18	Branch relative. Execution proceeds with the instruction addressed by the sum of the current instruction address and the sign-extended value of `s18`. The 2 least significant bits of `s18` are ignored.
bra s18	Branch absolute. Execution proceeds with the instruction addressed by the sign-extended value of `s18`. The 2 least significant bits of `s18` are ignored.
brasl rt, s18	Branch absolute and set link. Execution proceeds with the instruction addressed by the sign-extended value of `s18`. The 2 least significant bits of `s18` are ignored. The instruction following the current instruction is placed in word element 0 of register `rt`, and all other elements of `rt` are assigned a value of 0.
brhnz rc, s18	Branch if not zero halfword. If the halfword element 1 of register `rc` is nonzero, execution proceeds with the instruction addressed by the sum of the current instruction address and the sign-extended value of `s18`. The 2 least significant bits of `s18` are ignored. If halfword element 1 of `rc` is zero, execution proceeds with the next sequential instruction.
brhz rc, s18	Branch if zero halfword. If the halfword element 1 of register `rc` is zero, execution proceeds with the instruction addressed by the sum of the current instruction address and the sign-extended value of `s18`. The 2 least significant bits of `s18` are ignored. If the halfword element 1 of register `rc` is nonzero, execution proceeds with the next sequential instruction.
brnz rc, s18	Branch if not zero word. If the word element 0 of register `rc` is nonzero, execution proceeds with the instruction addressed by the sum of the current instruction address and the sign-extended value of `s18`. The 2 least significant bits of `s18` are ignored. If word element 0 of register `rc` is zero, execution proceeds with the next sequential instruction.
brsl rt, s18	Branch relative and set link. Execution proceeds with the instruction addressed by the sum of the current instruction address and the sign-extended value of `s18`. The 2 least significant bits of `s18` are ignored. The instruction following the current instruction is placed in word element 0 of register `rt`, and all other elements of `rt` are assigned a value of 0.
brz rc, s18	Branch if zero word. If the word element 0 of register `rc` is zero, execution proceeds with the instruction addressed by the sum of the current instruction address and the sign-extended value of `s18`. The 2 least significant bits of `s18` are ignored. If word element 0 of register `rc` is nonzero, execution proceeds with the following instruction.
cbd rt, u7(ra)	Generate controls for byte insertion (d-form). A control mask is generated that can be used by the `shufb` instruction to insert a byte at the effective address computed by the sum of register `ra` and the unsigned value `u7`. The control mask is placed in register `rt`.

Instruction/Usage	Description
cbx rt, ra, rb	Generate controls for byte insertion (x-form). A control mask is generated that can be used by the `shufb` instruction to insert a byte at the effective address computed by the sum of registers `ra` and `rb`. The control mask is placed in register `rt`.
cdd rt, u7(ra)	Generate controls for doubleword insertion (d-form). A control mask is generated that can be used by the `shufb` instruction to insert a doubleword at the effective address computed by the sum of register `ra` and unsigned value `u7`. The control mask is placed in register `rt`.
cdx rt, ra, rb	Generate controls for doubleword insertion (x-form). A control mask is generated that can be used by the `shufb` instruction to insert a doubleword at the effective address computed by the sum of registers `ra` and `rb`. The control mask is placed in register `rt`.
ceq rt, ra, rb	Compare equal word. Each word element of register `ra` is compared with the corresponding word element of register `rb`. If the two elements are equal, `0xFFFFFFFF` is placed in the corresponding word element of register `rt`; otherwise, `0x0` is placed in the corresponding word element.
ceqb rt, ra, rb	Compare equal byte. Each byte element of register `ra` is compared with the corresponding byte element of register `rb`. If the two elements are equal, `0xFF` is placed in the corresponding byte element of register `rt`; otherwise, `0x0` is placed in the corresponding byte element.
ceqbi rt, ra, s10	Compare equal byte immediate. Each byte element of register `ra` is compared with the 8 least significant bits of `s10`. If the two values are equal, `0xFF` is placed in the corresponding byte element of register `rt`; otherwise, `0x0` is placed in the corresponding byte element.
ceqh rt, ra, rb	Compare equal halfword. Each halfword element of register `ra` is compared with the corresponding halfword element of register `rb`. If the two elements are equal, `0xFFFF` is placed in the corresponding halfword element of register `rt`; otherwise, `0x0` is placed in the corresponding halfword element.
ceqhi rt, ra, s10	Compare equal halfword immediate. Each halfword element of register `ra` is compared with the 16-bit sign-extended value `s10`. If the two values are equal, `0xFFFF` is placed in the corresponding halfword element of register `rt`; otherwise, `0x0` is placed in the corresponding halfword element.
ceqi rt, ra, s10	Compare equal word immediate. Each word element of register `ra` is compared with the 32-bit sign-extended value `s10`. If the two values are equal, `0xFFFFFFFF` is placed in the corresponding word element of register `rt`; otherwise, `0x0` is placed in the corresponding word element.
cflts rt, ra, scale7	Convert floating to signed integer. Each floating-point element of register `ra` is multiplied by 2^{scale7}, converted to a signed 32-bit integer, and placed in the corresponding word element of register `rt`. Values outside of the range from -2^{31} to 2^{31}-1 are clamped (saturated to the nearest bound).
cfltu rt, ra, scale7	Convert floating to unsigned integer. Each floating-point element of register `ra` is multiplied by 2^{scale7}, converted to an unsigned 32-bit integer, and placed in the corresponding word element of register `rt`. Values outside of the range from 0 to 2^{32}-1 are clamped (saturated to the nearest bound).
cg rt, ra, rb	Carry generate word. Each word element of register `ra` is added to the corresponding word element of register `rb`. The carry out is placed in the least significant bit of the corresponding word element of register `rt`, and 0 is placed in the remaining bits of `rt`.

Instruction/Usage	Description
cgt rt, ra, rb	Compare greater than word. Each word element of register `ra` is compared with the corresponding word element of register `rb`. If the word in `ra` is greater than the corresponding word in `rb`, `0xFFFFFFFF` is placed in the corresponding word element of register `rt`; otherwise, `0x0` is placed in the corresponding word element.
cgtb rt, ra, rb	Compare greater than byte. Each byte element of register `ra` is compared with the corresponding byte element of register `rb`. If the byte in `ra` is greater than the corresponding byte in `rb`, `0xFF` is placed in the corresponding byte element of register `rt`; otherwise, `0x0` is placed in the corresponding byte element.
cgtbi rt, ra, s10	Compare greater than byte immediate. Each byte element of register `ra` is compared with the 8 least significant bits of `s10`. If the byte in `ra` is greater than the 8 least significant bits of `s10`, `0xFF` is placed in the corresponding byte element of register `rt`; otherwise, `0x0` is placed in the corresponding byte element.
cgth rt, ra, rb	Compare greater than halfword. Each halfword element of register `ra` is compared with the corresponding halfword element of register `rb`. If the halfword in `ra` is greater than the corresponding halfword in `rb`, `0xFFFF` is placed in the corresponding halfword element of register `rt`; otherwise, `0x0` is placed in the corresponding halfword element.
cgthi rt, ra, s10	Compare greater than halfword immediate. Each halfword element of register `ra` is compared with the 16-bit sign-extended value `s10`. If the halfword in `ra` is greater than `s10`, `0xFFFF` is placed in the corresponding halfword element of register `rt`; otherwise, `0x0` is placed in the corresponding halfword element.
cgti rt, ra, s10	Compare greater than word immediate. Each word element of register `ra` is compared with the 32-bit sign-extended value `s10`. If the word in `ra` is greater than `s10`, `0xFFFFFFFF` is placed in the corresponding word element of register `rt`; otherwise, `0x0` is placed in the corresponding word element.
cgx rt, ra, rb	Carry generate word extended. For each word element in registers `ra` and `rb`, a carry out is generated by summing the element of register `ra`, the corresponding element of `rb`, and the least significant bit of `rt`. The carry out is placed in the least significant bit of the corresponding word element of `rt`, and zeros are placed in the remaining bits.
chd rt, u7(ra)	Generate controls for halfword insertion (d-form). A control mask is generated that can be used by the `shufb` instruction to insert a halfword at the effective address computed by the sum of register `ra` and the unsigned value `u7`. The control mask is placed in register `rt`.
chx rt, ra, rb	Generate controls for halfword insertion (x-form). A control mask is generated that can be used by the `shufb` instruction to insert a halfword at the effective address computed by the sum of registers `ra` and `rb`. The control mask is placed in register `rt`.
clgt rt, ra, rb	Compare logical greater than word. Each word element of register `ra` is logically compared with the corresponding word element of register `rb`. If the word in `ra` is greater than the corresponding word in `rb`, `0xFFFFFFFF` is placed in the corresponding word element of register `rt`; otherwise, `0x0` is placed in the corresponding word element.
clgtb rt, ra, rb	Compare logical greater than byte. Each byte element of register `ra` is logically compared with the corresponding byte element of register `rb`. If the byte in `ra` is greater than the corresponding byte in `rb`, `0xFF` is placed in the corresponding byte element of register `rt`; otherwise, `0x0` is placed in the corresponding byte element.

Instruction/Usage	Description
clgtbi rt, ra, s10	Compare logical greater than byte immediate. Each byte element of register `ra` is logically compared with the 8 least significant bits of `s10`. If the byte in `ra` is greater than the value in `s10`, `0xFF` is placed in the corresponding byte element of register `rt`; otherwise, `0x0` is placed in the corresponding byte element.
clgth rt, ra, rb	Compare logical greater than halfword. Each halfword element of register `ra` is logically compared with the corresponding halfword element of register `rb`. If the halfword in `ra` is greater than the corresponding halfword in `rb`, `0xFFFF` is placed in the corresponding halfword element of register `rt`; otherwise, `0x0` is placed in the corresponding halfword element.
clgthi rt, ra, s10	Compare logical greater than halfword immediate. Each halfword element of register `ra` is logically compared with the 16-bit sign-extended value `s10`. If the halfword in `ra` is greater than `s10`, `0xFFFF` is placed in the corresponding halfword element of register `rt`; otherwise, `0x0` is placed in the corresponding halfword element.
clgti rt, ra, s10	Compare logical greater than word immediate. Each word element of register `ra` is logically compared with the 32-bit sign-extended value `s10`. If the word in `ra` is greater than `s10`, `0xFFFFFFFF` is placed in the corresponding word element of register `rt`; otherwise, `0x0` is placed in the corresponding word element.
clz rt, ra	Count leading zeros. The number of zeros to the left of the first `1` in each word element of register `ra` is counted, and the resulting count is placed in the corresponding element of register `rt`.
cntb rt, ra	Count ones in bytes. The number of ones in each byte element of register `ra` is counted, and the resulting count is placed in the corresponding element of register `rt`.
csflt rt, ra, scale7	Convert signed integer to floating. Each signed word element of register `ra` is converted to floating-point, multiplied by $2^{-scale7}$, and placed in the corresponding floating-point element of register `rt`.
cuflt rt, ra, scale7	Convert unsigned integer to floating. Each unsigned word element of register `ra` is converted to floating-point, multiplied by $2^{-scale7}$, and placed in the corresponding floating-point element of register `rt`.
cwd rt, u7(ra)	Generate controls for word insertion (d-form). A control mask is generated that can be used by the `shufb` instruction to insert a word at the effective address computed by the sum of register `ra` and the unsigned value `u7`. The control mask is placed in register `rt`.
cwx rt, ra, rb	Generate controls for word insertion (x-form). A control mask is generated that can be used by the `shufb` instruction to insert a word at the effective address computed by the sum of registers `ra` and `rb`. The control mask is placed in register `rt`.
dfa rt, ra, rb	Double floating add. Each double floating-point element of register `ra` is added to the corresponding double floating-point element of register `rb`, and the results are placed in the corresponding elements of register `rt`.
dfceq rt, ra, rb	Double floating compare equal. Each double floating-point element of register `ra` is compared with the corresponding double floating-point element of register `rb`. If the two elements are equal, `0xFFFFFFFFFFFFFFFF` is placed in the corresponding double-word element of register `rt`; otherwise, `0x0` is placed in the corresponding double-word element.

Instruction/Usage	Description
dfcgt rt, ra, rb	Double floating compare greater than. Each double floating-point element of register `ra` is compared with the corresponding double floating-point element of register `rb`. If the element in `ra` is greater than the corresponding element of register `rb`, `0xFFFFFFFFFFFFFFFF` is placed in the corresponding double-word element of register `rt`; otherwise, `0x0` is placed in the corresponding double-word element.
dfcmeq rt, ra, rb	Double floating compare magnitude equal. The absolute value of each double floating-point element of register `ra` is compared with the absolute value of the corresponding double floating-point element of register `rb`. If the two elements are equal, `0xFFFFFFFFFFFFFFFF` is placed in the corresponding double-word element of register `rt`; otherwise, `0x0` is placed in the corresponding double-word element.
dfcmgt rt, ra, rb	Double floating compare magnitude greater than. The absolute value of each double floating-point element of register `ra` is compared with the absolute value of the corresponding double floating-point element of register `rb`. If the element in `ra` is greater than the corresponding element of register `rb`, `0xFFFFFFFFFFFFFFFF` is placed in the corresponding double-word element of register `rt`; otherwise, `0x0` is placed in the corresponding double-word element of register `rt`.
dfm rt, ra, rb	Double floating multiply. Each double floating-point element of register `ra` is multiplied by the corresponding double floating-point element of register `rb`, and the results are placed in the corresponding elements of register `rt`.
dfma rt, ra, rb	Double floating multiply and add. Each double floating-point element of register `ra` is multiplied by the corresponding double floating-point element of register `rb`, and the corresponding double floating-point element of register `rt` is then added to the product. The results are placed in the corresponding elements of register `rt`.
dfms rt, ra, rb	Double floating multiply and subtract. Each double floating-point element of register `ra` is multiplied by the corresponding double floating-point element of register `rb`, and the corresponding double floating-point element of register `rt` is subtracted from the product. The results are placed in the corresponding elements of register `rt`.
dfnma rt, ra, rb	Double floating negative multiply and add. Each double floating-point element of register `ra` is multiplied by the corresponding double floating-point element of register `rb`, and the corresponding double floating-point element of register `rt` is added to the product. Each result is negated and placed in the corresponding element of register `rt`.
dfnms rt, ra, rb	Double floating negative multiply and subtract. Each double floating-point element of register `ra` is multiplied by the corresponding double floating-point element of register `rb`, and the product is subtracted from the corresponding double floating-point element of register `rt`. The results are placed in corresponding elements of register `rt`.
dfs rt, ra, rb	Double floating subtract. Each double floating-point element of register `rb` is subtracted from the corresponding double floating-point element of register `ra`, and the results are placed in the corresponding elements of register `rt`.
dftsv rt, ra, u7	Double floating test special value. Each double floating-point element of register `ra` is tested for the special values specified by the immediate value `u7`. If any of the specified tests are true, `0xFFFFFFFFFFFFFFFF` is placed in the corresponding double-word element of register `rt`; otherwise, `0x0` is placed in the corresponding double-word element.
dsync	Synchronize data. All pending store operations to local storage memory are completed before the processor proceeds to the next instruction.
eqv rt, ra, rb	Equivalent. The value in register `ra` is logically exclusive ORed with the value in

Instruction/Usage	Description
	register `rb`, and the complement of the result is placed in register `rt`.
fa rt, ra, rb	Floating add. Each floating-point element of register `ra` is added to the corresponding floating-point element of register `rb`, and the results are placed in the corresponding elements of register `rt`.
fceq rt, ra, rb	Floating compare equal. Each single-precision floating-point element of register `ra` is compared with the corresponding single-precision floating-point element of register `rb`. If the two elements are equal, `0xFFFFFFFF` is placed in the corresponding word element of register `rt`; otherwise, `0x0` is placed in the corresponding word element.
fcgt rt, ra, rb	Floating compare greater than. Each single-precision floating-point element of register `ra` is compared with the corresponding single-precision floating-point element of register `rb`. If the element in `ra` is greater than the corresponding element in `rb`, `0xFFFFFFFF` is placed in the corresponding word element of register `rt`; otherwise, `0x0` is placed in the corresponding word element.
fcmeq rt, ra, rb	Floating compare magnitude equal. The absolute value of each single-precision floating-point element of register `ra` is compared with the absolute value of the corresponding single-precision floating-point element of register `rb`. If the elements are equal, all ones are placed in the corresponding word element of register `rt`; otherwise, `0x0` is placed in the corresponding word elements.
fcmgt rt, ra, rb	Floating compare magnitude greater than. The absolute value of each single-precision floating-point element of register `ra` is compared with the absolute value of the corresponding single-precision floating-point element of register `rb`. If the value in `ra` is greater than the corresponding value in `rb`, `0xFFFFFFFF` is placed in the corresponding word element of register `rt`; otherwise, `0x0` is placed in the corresponding word element.
fesd rt, ra	Floating extend single to double. Each even single-precision floating-point element of register `ra` is converted to double precision and then placed in the corresponding element of register `rt`.
fi rt, ra, rb	Floating interpolate. Each floating-point element of register `ra` is interpolated to produce a more accurate estimate, using the base and step contained in the corresponding element of register `rb`, where `rb` is in the output format of a `frest` or `frsqest` instruction. The interpolated result is placed in the corresponding element of register `rt`.
fm rt, ra, rb	Floating multiply. Each floating-point element of register `ra` is multiplied by the corresponding floating-point element of register `rb`, and the products are placed in the corresponding elements of register `rt`.
fma rt, ra, rb, rc	Floating multiply and add. Each floating-point element of register `ra` is multiplied by the corresponding floating-point element of register `rb`, and the corresponding floating-point element of register `rc` is then added to the product. The results are placed in corresponding elements of register `rt`.
fms rt, ra, rb, rc	Floating multiply and subtract. Each floating-point element of register `ra` is multiplied by the corresponding floating-point element of register `rb`, and the corresponding floating-point element of register `rc` is subtracted from the product. The results are placed in the corresponding elements of register `rt`.
fnms rt, ra, rb, rc	Floating negative multiply and subtract. Each floating-point element of register `ra` is multiplied by the corresponding floating-point element of register `rb`, and the product is subtracted from the corresponding floating-point element of register `rc`. The results are placed in the corresponding elements of register `rt`.
frds rt, ra	Floating round double to single. Each double floating-point element of register `ra` is rounded to single precision and placed in the corresponding even element of register `rt`. At the same time, a zero is placed in the corresponding odd element of `rt`.
frest rt, ra	Floating reciprocal estimate. A base and step is computed for estimating the

Instruction/Usage	Description
	reciprocal of each floating-point element of register `ra`, and the result is placed in the corresponding element of register `rt`. The result returned by this instruction is intended as an operand to the `fi` instruction.
frsqest rt, ra	Floating reciprocal square root estimate. A base and step is computed for estimating the reciprocal of the square root for each floating-point element of register `ra`, and the result is placed in the corresponding element of register `rt`. The result returned by this instruction is intended as an operand to the `fi` instruction.
fs rt, ra, rb	Floating subtract. Each floating-point element of register `rb` is subtracted from the corresponding floating-point element of register `ra`, and the results are placed in the corresponding elements of register `rt`.
fscrrd rt	Floating-Point Status and Control Register read. The contents of the Floating-Point Status and Control Register (FPSCR) are read and placed in register `rt`.
fscrwr ra fscrwr rc, ra	Floating-Point Status and Control Register write. The 128-bit register `ra` is written into the Floating-Point Status and Control Register (FPSCR). Register `rc` is a false target and no value is ever written to it. If register `rc` is not specified, register 0 is used as the false target.
fsm rt, ra	Form select mask for words. The 4 least significant bits of word element 0 of register `ra` are used to create a mask by replicating each bit 32 times. The 128-bit result is returned in register `rt`.
fsmb rt, ra	Form select mask for bytes. The 16 least significant bits of word element 0 of register `ra` are used to create a mask by replicating each bit eight times. The 128-bit result is returned in register `rt`.
fsmbi rt, u16	Form select mask for byte immediate. The 16 bits of `u16` are used to create a mask by replicating each bit eight times. The 128-bit result is returned in register `rt`.
fsmh rt, ra	Form select mask for halfwords. The 8 least significant bits of word element 0 of register `ra` are used to create a mask by replicating each bit 16 times. The 128-bit result is returned in register `rt`.
gb rt, ra	Gather bits from words. A 4-bit value is formed by concatenating the least significant bit of each word element of register `ra`. The 4-bit value is then placed in the least significant bits of word element 0 of register `rt`, and zeros are placed in the remaining bits.
gbb rt, ra	Gather bits from bytes. A 16-bit value is formed by concatenating the least significant bit of each byte element of register `ra`. The 16-bit value is then placed in the least significant bits of word element 0 of register `rt`, and zeros are placed in the remaining bits.
gbh rt, ra	Gather bits from halfwords. An 8-bit value is formed by concatenating the least significant bit of each halfword element of register `ra`. The 8-bit value is then placed in the least significant bits of word element 0 of register `rt`, and zeros are placed in the remaining bits.
hbr s11, ra	Hint for branch (r-form). An instruction prefetch is allowed to occur at the branch target address contained in word element 0 of register `ra`, for the branch instruction that is addressed by the sum of the address of this instruction and the sign-extended value `s11`. The 2 least significant bits of `s11` are ignored.
hbra s11, s18	Hint for branch (a-form). An instruction prefetch is allowed to occur at the branch target address specified by the sign-extended value `s18`, for the branch instruction addressed by the sum of the address of this instruction and the sign-extended value `s11`. The 2 least significant bits of `s11` and `s18` are ignored.
hbrp	Hint for branch, prefetch (r-form). A slot in the fetch unit is reserved for an in-line prefetch. This instruction translates to an `hbr` instruction that has the `P` feature

Instruction/Usage	Description
	bit set. The field in the `hbr` instruction that contains the offset to the branch instruction is set to zero.
hbrr s11, s18	Hint for branch relative. An instruction prefetch is allowed to occur at the branch target that is addressed by the sum of the address of this instruction and the sign-extended value `s18`, for the branch instruction that is addressed by the sum of the address of this instruction and the sign-extended value `s11`. The 2 least significant bits of `s18 and s11` are ignored.
heq ra, rb heq rt, ra, rb	Halt if equal. If word element 0 of registers `ra` and `rb` are equal, the processor is halted. Register `rt` is a false target and is never written to. If register `rt` is not specified, register 0 is used as the false target.
heqi ra, s10 heqi rt, ra, s10	Halt if equal immediate. If word element 0 of register `ra` equals the sign-extended value of `s10`, the processor is halted. Register `rt` is a false target, and no value is ever written to it. If register `rt` is not specified, register 0 is used as the false target.
hgt ra, rb hgt rt, ra, rb	Halt if greater than. If signed word element 0 of register `ra` is greater than word element 0 of register `rb`, the processor is halted. Register `rt` is a false target, and no value is ever written to it. If register `rt` is not specified, register 0 is used as the false target.
hgti ra, s10 hgti rt, ra, s10	Halt if greater than immediate. If signed word element 0 of register `ra` is greater than the sign-extended value `s10`, the processor is halted. Register `rt` is a false target, and no value is ever written to it. If register `rt` is not specified, register 0 is used as the false target.
hlgt ra, rb hlgt rt, ra, rb	Halt if logically greater than. If unsigned word element 0 of register `ra` is greater than unsigned word element 0 of register `rb`, the processor is halted. Register `rt` is a false target, and no value is ever written to it. If register `rt` is not specified, register 0 is used as the false target.
hlgti ra, s10 hlgti rt, ra, s10	Halt if logically greater than immediate. If unsigned word element 0 of register `ra` is logically greater than the sign-extended value `s10`, the processor is halted. Register `rt` is a false target, and no value is ever written to it. If register `rt` is not specified, register 0 is used as the false target.
il rt, s16	Immediate load word. The sign-extended value `s16` is loaded into each of the word elements of `rt`.
ila rt, u18	Immediate load address. The unsigned value `u18` is loaded into each of the word elements of `rt`.
ilh rt, u16	Immediate load halfword. The value `u16` is loaded into each of the eight halfword elements of `rt`.
ilhu rt, u16	Immediate load halfword upper. The value `u16` is loaded into the 16 most significant bits of each of the four word elements of `rt`.
iohl rt, u16	Immediate OR halfword lower. Immediate OR the value `u16` with each of the word elements of `rt`.
iretd iretd ra	Interrupt return, disable. Execution proceeds with the instruction addressed by machine state save/restore register 0 (`SRR0`). Interrupts are disabled. Register `ra` is a false source, and its contents are ignored. If `ra` is not specified, register 0 is used as a false source.
irete irete ra	Interrupt return, enable. Execution proceeds with the instruction addressed by machine state save/restore register 0 (`SRR0`). Interrupts are enabled. Register `ra` is a false source, and its contents are ignored. If `ra` is not specified, register 0 is used as a false source.
iret iret ra	Interrupt return. Execution proceeds with the instruction addressed by machine state save/restore register 0 (`SRR0`). Register `ra` is a false source, and its contents are ignored. If `ra` is not specified, register 0 is used as a false source.
lnop	No operation (load). A no-operation is performed on the load pipeline.

Instruction/Usage	Description
lqa rt, s18	Load quadword (a-form). A quadword is loaded into register `rt` from the effective address specified by the sign-extended value `s18`. The 2 least significant bits of `s18` are ignored.
lqd rt, s14(ra)	Load quadword (d-form). A quadword is loaded into register `rt` from the effective address computed by the sum of register `ra` and the sign-extended value `s14`. The 4 least significant bits of `s14` are ignored.
lqr rt, s18	Load quadword instruction relative (a-form). A quadword is loaded into register `rt` from the effective address specified by the sum of the current instruction address and `s18`. The 2 least significant bits of `s18` are ignored.
lqx rt, ra, rb	Load quadword (x-form). A quadword is loaded into register `rt` from the effective address computed by the sum of registers `ra` and `rb`.
mfspr rt, spr	Move from special purpose register. The contents of the specified special purpose register `spr` are moved to the word element 0 of register `rt`.
mpy rt, ra, rb	Multiply. The signed 16 least significant bits of the corresponding word elements of registers `ra` and `rb` are multiplied, and the 32-bit products are placed in the corresponding word elements of register `rt`.
mpya rt, ra, rb, rc	Multiply and add. The signed 16 least significant bits of the corresponding word elements of registers `ra` and `rb` are multiplied, and the 32-bit products are then added to the corresponding word elements of register `rc`. The results are placed in the corresponding elements of register `rt`.
mpyh rt, ra, rb	Multiply high. The 16 most significant bits of the word elements of register `ra` are multiplied by the 16 least significant bits of the corresponding elements of register `rb`. The 32-bit products are then shifted left by 16 bits and placed in the corresponding word elements of register `rt`.
mpyhh rt, ra, rb	Multiply high high. The signed 16 most significant bits of the word elements of registers `ra` and `rb` are multiplied, and the 32-bit products are placed in the corresponding word elements of register `rt`.
mpyhha rt, ra, rb	Multiply high high and add. The signed 16 most significant bits of the word elements of registers `ra` and `rb` are multiplied. The 32-bit products are then added to the corresponding word elements of register `rt`, and the sums are placed in register `rt`.
mpyhhau rt, ra, rb	Multiply high high unsigned and add. The unsigned 16 most significant bits of the word elements of registers `ra` and `rb` are multiplied, and the 32-bit products are then added to the corresponding word elements of register `rt`, and the sums are placed in register `rt`.
mpyhhu rt, ra, rb	Multiply high high unsigned. The unsigned 16 most significant bits of the word elements of registers `ra` and `rb` are multiplied, and the 32-bit products are then placed in the corresponding word elements of register `rt`.
mpyi rt, ra, s10	Multiply immediate. The 16 least significant bits of each of the word elements of register `ra` are multiplied by the sign-extended value `s10`. The 32-bit products are then placed in the corresponding word elements of register `rt`.
mpys rt, ra, rb	Multiply and shift right. The 16 most significant bits of the corresponding word elements of registers `ra` and `rb` are multiplied, and the 16 most significant bits of the 32-bit products are placed in the least significant bits of the corresponding word elements of register `rt`.
mpyu rt, ra, rb	Multiply unsigned. The unsigned 16 least significant bits of the corresponding word elements of registers `ra` and `rb` are multiplied, and the 32-bit products are placed in the corresponding word elements of register `rt`.
mpyui rt, ra, s10	Multiply unsigned immediate. The 16 least significant bits of each of the word elements of register `ra` is multiplied by the sign-extended value `s10`. Both operands are treated as unsigned. The 32-bit products are placed in the corresponding word elements of register `rt`.

Instruction/Usage	Description
mtspr spr, ra	Move to special purpose register. The contents of word element 0 of register `ra` are moved to the special purpose register `spr`.
nand rt, ra, rb	Nand. The value of register `ra` is logically ANDed with register `rb`, and the complement of the result is placed in register `rt`.
nop nop rt	No operation (execute). A no-operation is performed on the execute pipeline. Register `rt` is a false target, and no value is ever written to it. If register `rt` is not specified, register 0 is used as the false target.
nor rt, ra, rb	Nor. The value of register `ra` is logically ORed with register `rb`, and the complement of the result is placed in register `rt`.
or rt, ra, rb	Or. The value of register `ra` is logically ORed with register `rb`, and the result is placed in register `rt`.
orbi rt, ra, s10	Or byte immediate. The 8 least significant bits of `s10` are logically ORed with each byte element of register `ra`, and the results are placed in the corresponding elements of register `rt`.
orc rt, ra, rb	Or with complement. The value of register `ra` is logically ORed with the complement of register `rb`, and the result is placed in register `rt`.
orhi rt, ra, s10	Or halfword immediate. The sign-extended value `s10` is logically ORed with each halfword element of register `ra`, and the results are placed in the corresponding elements of register `rt`.
ori rt, ra, s10	Or word immediate. The sign-extended value `s10` is logically ORed with each word element of register `ra`, and the results are placed in the corresponding elements of register `rt`.
orx rt, ra	Or word across. The four word elements of register `ra` are logically ORed, and the result is placed in word element 0 of register `rt`. Word elements 1, 2, and 3 of register `rt` are assigned a value of 0.
rchcnt rt, ch	Read channel count. The channel count of the channel `ch` is read, and the count placed in register `rt`.
rdch rt, ch	Read channel. The contents of the channel `ch` are read, and the contents placed in register `rt`.
rot rt, ra, rb	Rotate word. The contents of each word element of register `ra` are rotated left according to the corresponding word element of register `rb`. The results are placed in the corresponding word elements of register `rt`.
roth rt, ra, rb	Rotate halfword. The contents of each halfword element of register `ra` are rotated left according to the corresponding halfword element of register `rb`. The results are placed in the corresponding halfword elements of register `rt`.
rothi rt, ra, s7	Rotate halfword immediate. The contents of each halfword element of register `ra` are rotated left according to the 4 least significant bits of `s7`. The results are placed in the corresponding halfword elements of register `rt`.
rothm rt, ra, rb	Rotate and mask halfword. The contents of each halfword element of register `ra` are right-shifted according to the two’s complement of the 5 least significant bits of the corresponding halfword element of register `rb`. The results are placed in the corresponding halfword elements of register `rt`.
rothmi rt, ra, s6	Rotate and mask halfword immediate. The contents of each halfword element of register `ra` are right-shifted according to the two’s complement of the signed value `s6`. The results are placed in the corresponding halfword elements of register `rt`.
roti rt, ra, s7	Rotate word immediate. The contents of each word element of register `ra` are rotated left according to the signed value `s7`. The results are placed in the corresponding word elements of register `rt`.
rotm rt, ra, rb	Rotate and mask word. The contents of each word element of register `ra` are right-shifted according to the two’s complement of the 6 least significant bits of

Instruction/Usage	Description
	the corresponding word element of register `rb`. The results are placed in the corresponding word elements of register `rt`.
rotma rt, ra, rb	Rotate and mask algebraic word. The contents of each word element of register `ra` are right-shifted according to the two’s complement of the 6 least significant bits of the corresponding word element of register `rb`. Copies of the sign bit are shifted in from the left. The results are placed in the corresponding word elements of register `rt`.
rotmah rt, ra, rb	Rotate and mask algebraic halfword. The contents of each halfword element of register `ra` are right-shifted according to the two’s complement of the 5 least significant bits of the corresponding halfword element of register `rb`. Copies of the sign bit are shifted in from the left. The results are placed in the corresponding halfword element of register `rt`.
rotmahi rt, ra, s6	Rotate and mask algebraic halfword immediate. The contents of each halfword element of register `ra` are right-shifted according to the signed value `s6`. Copies of the sign bit are shifted in from the left. The results are placed in the corresponding halfword elements of register `rt`.
rotmai rt, ra, s7	Rotate and mask algebraic word immediate. The contents of each word element of register `ra` are right-shifted according to the two’s complement of the signed value `s7`. Copies of the sign bit are shifted in from the left. The results are placed in the corresponding word elements of register `rt`.
rotmi rt, ra, s7	Rotate and mask word immediate. The contents of each word element of register `ra` are right-shifted according to the two’s complement of the signed value `s7`. The results are placed in the corresponding word elements of register `rt`.
rotqbi rt, ra, rb	Rotate quadword by bits. The contents of register `ra` are rotated left by the number of bits specified by the 3 least significant bits of word element 0 of register `rb`. The result is placed in register `rt`.
rotqbii rt, ra, u3	Rotate quadword by bits immediate. The contents of register `ra` are rotated left by the number of bits according to the value `u3`. The result is placed in register `rt`.
rotqby rt, ra, rb	Rotate quadword by bytes. The contents of register `ra` are rotated left by the number of bytes specified by the 4 least significant bits of word element 0 of register `rb`. The result is placed in register `rt`.
rotqbybi rt, ra, rb	Rotate quadword by bytes from bit shift count. The contents of register `ra` are rotated left by the number of bytes specified by bits 25–28 of word element 0 of register `rb`. The result is placed in register `rt`.
rotqbyi rt, ra, s7	Rotate quadword by bytes immediate. The contents of register `ra` are rotated left by the number of bytes according to the signed value `s7`. The result is placed in register `rt`.
rotqmbi rt, ra, rb	Rotate and mask quadword by bits. The contents of register `ra` are shifted right by the number of bits specified by the two’s complement of the 3 least significant bits of word element 0 of register `rb`. The result is placed in register `rt`.
rotqmbii rt, ra, s3	Rotate and mask quadword by bits immediate. The contents of register `ra` are shifted right by the number of bits specified by the two’s complement of the signed value `s3`. The result is placed in register `rt`.

Instruction/Usage	Description
rotqmby rt, ra, rb	Rotate and mask quadword by bytes. The contents of register `ra` are shifted right by the number of bytes specified by the two's complement of the 5 least significant bits of word element 0 of register `rb`. The result is placed in register `rt`.
rotqmbybi rt, ra, rb	Rotate and mask quadword by bytes from bit shift count. The contents of register `ra` are shifted right by the number of bytes specified by the two's complement of bits 24–28 of word element 0 of register `rb`. The result is placed in register `rt`.
rotqmbyi rt, ra, s6	Rotate and mask quadword by bytes immediate. The contents of register `ra` are shifted right by the number of bytes specified by the two's complement of the signed value `s6`. The result is placed in register `rt`.
selb rt, ra, rb, rc	Select bits. Each bit of register `rc` whose value is 0 selects the corresponding bit from register `ra`. A bit whose value is 1 selects the corresponding bit from register `rb`. The quadword result is placed in register `rt`.
sf rt, ra, rb	Subtract from word. Each word element of register `ra` is subtracted from the corresponding word element of register `rb`, and the results are placed in the corresponding word elements of register `rt`.
sfh rt, ra, rb	Subtract from halfword. Each halfword element of register `ra` is subtracted from the corresponding halfword element of register `rb`, and the results are placed in the corresponding word elements of register `rt`.
sfhi rt, ra, s10	Subtract from halfword immediate. Each halfword element of register `ra` is subtracted from the sign-extended value `s10`, and the results are placed in the corresponding halfword elements of register `rt`.
sfi rt, ra, s10	Subtract from word immediate. Each word element of register `ra` is subtracted from the sign-extended value `s10`, and the results are placed in the corresponding word elements of register `rt`.
sfx rt, ra, rb	Subtract from word extended. Each word element of register `ra` is subtracted from the corresponding word element of register `rb`. An additional 1 is subtracted from the result if the least significant bit of word element `rt` is 0. The results are placed in the corresponding word elements of register `rt`.
shl rt, ra, rb	Shift left word. The contents of each word element of register `ra` are shifted left according to the 6 least significant bits of the corresponding word element of register `rb`. The results are placed in the corresponding word elements of register `rt`.
shlh rt, ra, rb	Shift left halfword. The contents of each halfword element of register `ra` are shifted left according to the 5 least significant bits of the corresponding halfword element of register `rb`. The results are placed in the corresponding halfword elements of register `rt`.
shlhi rt, ra, u5	Shift left halfword immediate. The contents of each halfword element of register `ra` are shifted left according to unsigned value `u5`. The results are placed in the corresponding halfword elements of register `rt`.
shli rt, ra, u6	Shift left word immediate. The contents of each word element of register `ra` are shifted left according to the unsigned value `u6`. The results are placed in the corresponding word element of register `rt`.
shlqbi rt, ra, rb	Shift left quadword by bits. The contents of register `ra` are shifted left by the number of bits specified by the 3 least significant bits of word element 0 of register `rb`. The result is placed in register `rt`.
shlqbii rt, ra, u3	Shift left quadword by bits immediate. The contents of register `ra` are shifted left by the number of bits specified by the unsigned value `u3`. The result is placed in register `rt`.
shlqby rt, ra, rb	Shift left quadword by bytes. The contents of register `ra` are shifted left by the number of bytes specified by the 5 least significant bits of word element 0 of

Instruction/Usage	Description
	register rb. The result is placed in register rt.
shlqbybi rt, ra, rb	Shift left quadword by bytes from bit shift count. The contents of register ra are shifted left by the number of bytes specified by bits 24–28 of word element 0 of register rb. The result is placed in register rt.
shlqbyi rt, ra, u5	Shift left quadword by bytes immediate. The contents of register ra are shifted left by the number of bytes specified by the unsigned value u5. The result is placed in register rt.
shufb rt, ra, rb, rc	Shuffle bytes. Each byte of register rc is used to select a byte from either register ra or register rb or a constant (0, 0x80, or 0xFF). The results are placed in the corresponding bytes of register rt.
stop u14	Stop and signal. Execution is stopped, the current address is written to the SPU NPC register, the value u14 is written to the SPU status register, and an interrupt is sent to the PowerPC® Processor Unit (PPU).
stopd ra, rb, rc	Stop and signal with dependencies. Execution is stopped after register dependencies are met. This involves writing the current address to the SPU NPC register, writing the value 0x3FFF to the SPU status register, and interrupting the PPU.
stqa rc, s18	Store quadword (a-form). The quadword in register rc is stored at the effective address specified by the sign-extended value s18. The 2 least significant bits of s18 are ignored.
stqd rc, s14(ra)	Store quadword (d-form). The quadword in register rc is stored at the effective address computed by the sum of register ra and the sign-extended value s14. The 4 least significant bits of s14 are ignored.
stqr rc, s18	Store quadword instruction relative (a-form). The quadword in register rc is stored at the effective address specified by the sum of the current instruction address and s18. The 2 least significant bits of s18 are ignored.
stqx rc, ra, rb	Store quadword (x-form). The quadword in register rc is stored at the effective address computed by the sum of registers ra and rb.
sumb rt, ra, rb	Sum bytes into halfword. The 4 bytes of each word element of register ra are summed and placed in the corresponding odd halfword elements of register rt, and the 4 bytes of each word element of register rb are summed and placed in the corresponding even halfword elements of register rt.
sync	Synchronize. The processor waits until all pending store instructions have been completed before it fetches the next sequential instruction.
syncc	Synchronize channel. The processor waits until the channel is ready and all pending store instructions have been completed before it fetches the next sequential instruction.
wrch ch, ra	Write channel. The contents of register ra are written to the channel ch.
xor rt, ra, rb	Xor. The value of register ra is logically exclusive ORed with register rb and the result is placed in register rt.
xorbi rt, ra, s10	Exclusive or byte immediate. The 8 least significant bits of s10 are logically exclusive ORed with each byte element of register ra, and the results are placed in the corresponding elements of register rt.
xorhi rt, ra, s10	Exclusive or halfword immediate. The sign-extended 16 least significant bits of s10 are logically exclusive ORed with each halfword element of register ra, and the results are placed in the corresponding elements of register rt.
xori rt, ra, s10	Exclusive or word immediate. The sign-extended value of s10 is logically exclusive ORed with each word element of register ra, and the results are placed in the corresponding elements of register rt.
xsbh rt, ra	Extend sign byte to halfword. The 8 least significant bits of each halfword element of register ra are sign extended to 16 bits and placed in the

Instruction/Usage	Description
	corresponding halfword element of register `rt`.
xshw rt, ra	Extend sign halfword to word. The 16 least significant bits of each word element in register `ra` are sign extended to 32 bits and placed in the corresponding word element of register `rt`.
xswd rt, ra	Extend sign word to doubleword. The 32 least significant bits of each doubleword element in register `ra` are sign extended to 64 bits and placed in the corresponding doubleword element of register `rt`.

2.3. Aliases

For the programmer's convenience, the assembler supports the register and instruction aliases shown in Table 2-3.

Table 2-3: Register and Instruction Aliases

Alias	Is Equivalent To	Description
$LR	$0	Return address / link register
$SP	$1	Stack pointer
lr rt, ra	ori rt, ra, 0	Load register `rt` with the register `ra`

2.4. Channel Mnemonics

Table 2-4 and Table 2-5 specify the supported channel mnemonics. The assembler provides generic channel mnemonics of the form `$ch#` for all possible channels 0–127, where `#` indicates the channel number. For example, `$ch0` is the event status read channel.

All SPU channel mnemonics must be supported. In contrast, only target systems that support the MFC must support the MFC channel mnemonics.

Table 2-4: SPU Channels

Channel Number	Equivalent Mnemonic	Description
0–127	$ch0 – $ch127	Generic channel mnemonics
0	$SPU_RdEventStat	Read event status with mask applied
1	$SPU_WrEventMask	Write event mask
2	$SPU_WrEventAck	Write end of event processing
3	$SPU_RdSigNotify1	Signal notification 1
4	$SPU_RdSigNotify2	Signal notification 2
7	$SPU_WrDec	Write decrementer count
8	$SPU_RdDec	Read decrementer count
11	$SPU_RdEventMask	Read event mask
13	$SPU_RdMachStat	Read SPU run status
14	$SPU_WrSRR0	Write SPU machine state save/restore register 0 (SRR0)
15	$SPU_RdSRR0	Read SPU machine state save/restore register 0 (SRR0)
28	$SPU_WrOutMbox	Write outbound mailbox contents
29	$SPU_RdInMbox	Read inbound mailbox contents
30	$SPU_WrOutIntrMbox	Write outbound interrupt mailbox contents (interrupting PPU)

Table 2-5: MFC Channels

Channel Number	Equivalent Mnemonic	Description
9	$MFC_WrMSSyncReq	Write multisource synchronization request
12	$MFC_RdTagMask	Read tag mask
16	$MFC_LSA	Write local memory address command parameter
17	$MFC_EAH	Write high order DMA effective address command parameter
18	$MFC_EAL	Write low order DMA effective address command parameter
19	$MFC_Size	Write DMA transfer size command parameter
20	$MFC_TagID	Write tag identifier command parameter
21	$MFC_Cmd	Write and enqueue DMA command with associated class ID
22	$MFC_WrTagMask	Write tag mask
23	$MFC_WrTagUpdate	Write request for conditional or unconditional tag status update
24	$MFC_RdTagStat	Read tag status with mask applied
25	$MFC_RdListStallStat	Read DMA list stall-and-notify status
26	$MFC_WrListStallAck	Write DMA list stall-and-notify acknowledge
27	$MFC_RdAtomicStat	Read completion status of last completed immediate MFC atomic update command (see the Synergistic Processor Unit Channels section of *Cell Broadband Engine Architecture*)

2.5. Immediate Values

Many instructions accept signed or unsigned immediate values of various lengths. These values can be encoded in the following ways:

- *An immediate constant value or expression*. For example, the instruction "`ai $3, $3, -32`" subtracts 32 from each of the word elements of register 3.
- *A PC relative address.* The current program counter is expressed by a dot (`.`) symbol. For example, the instruction "`br .-4`" branches to the instruction immediately prior to this instruction.
- *A symbolic label address*. These addresses are resolved during link edit, during which the appropriate instruction value is encoded in the symbol's place. For example, relative addressing instructions are encoded with a relative address. Absolute address instructions are encoded with the address of the label or symbol. Halfword addresses are specified using the `@h` or `@l` to specify the high and lower halfwords, respectively. For example, the following instruction sequence loads the 32-bit address of *`variable`* into register 3:

```
ilhu $3, variable@h    # load high halfword address of variable
iohl $3, variable@l    # logically OR low halfword address of variable
```

2.6. Errors and Warnings

To assist in early identification of coding errors, the assembler will issue a warning or error whenever an immediate value is outside of the range expected by the respective instruction. For some instructions, it is inappropriate to issue a warning or an error for out-of-range values. Table 2-6 shows valid ranges for immediate operands, in addition to any special variances to the valid range of values.

Table 2-6: Valid Immediate Values

Immediate Value	Minimum Value	Maximum Value	Special Variances
s3	-4	3	No limits will be placed on the `rotqmbii`

Immediate Value	Minimum Value	Maximum Value	Special Variances
			instruction. The 7 least significant bits of the specified immediate value will be encoded in the instruction.
s6	-32	31	Warnings may optionally be issued for values outside the range [-31, 0] for the `rothmi`, `rotmahi`, and `rotqmbyi` instructions.
s7	-64	63	No limits will be placed on the `rothi`, `roti`, and `rotqbyi` instructions. The 7 least significant bits of the specified immediate value will be encoded in the instructions. Warnings may optionally be issued for values outside the range [-63, 0] for the `rotmai` and `rotmi` instructions.
s10	-512	511	Warnings may optionally be issued for values outside the range [-128, 255] for the `andbi`, `ceqbi`, `cgtbi`, `clgtbi`, `orbi`, and `xorbi` instructions.
s11	-1024	1023	Warnings may optionally be issued for values whose 2 least significant bits are nonzero, for the `hbr`, `hbra`, and `hbrr` instructions.
s14	-8192	8191	Warnings may optionally be issued for values whose 4 least significant bits are nonzero, for the `lqd` and `stqd` instructions.
s16	-32768	32767	
s18	-131072	131071	Warnings may optionally be issued for values whose 2 least significant bits are nonzero, for the `br`, `bra`, `brasl`, `brhnz`, `brhz`, `brnz`, `brsl`, `brz`, `hbra`, `hbrr`, `lqa`, `lqr`, `stqa`, and `stqr` instructions.
scale7	0	127	
u3	0	7	No limits will be placed on the `rotqbii` instruction. The 7 least significant bits of the specified immediate value will be encoded in the instructions.
u5	0	31	
u6	0	63	
u7	0	127	No limits will be placed on the `cbd`, `cdd`, `chd`, and `cwd` instructions. The assembler will quietly encode the least significant bits of the immediate value as the *u7* parameter.
u14	0	16383	
u16	0	65535	For instructions in which no leading bits are appended, the minimum value will be extended to -32768. This includes the `fsmbi`, `ilh`, `ilhu`, and `iohl` instructions.
u18	0	262143	

End of Document