Chapter 8. Instruction Set

This chapter lists the PowerPC instruction set in alphabetical order by mnemonic. Each entry includes the instruction formats and a quick reference ‘legend’ that provides such information as the level(s) of the PowerPC architecture in which the instruction may be found—user instruction set architecture (UISA), virtual environment architecture (VEA), and operating environment architecture (OEA); and the privilege level of the instruction—user- or supervisor-level (an instruction is assumed to be user-level unless the legend specifies that it is supervisor-level); and the instruction formats.

U V O

The format diagrams show, horizontally, all valid combinations of instruction fields; for a graphical representation of these instruction formats, see Appendix A, “PowerPC Instructions Set Listings.” A description of the instruction fields and pseudocode conventions are also provided.

For more information on the PowerPC instruction set, refer to Chapter 4, “Addressing Modes and Instruction Set Summary.”

NOTE: The architecture specification refers to user-level and supervisor-level as problem state and privileged state, respectively.

8

8.1 Instruction Formats

U

Instructions are four bytes long and word-aligned, so when instruction addresses are presented to the processor (as in branch instructions) the two low-order bits are ignored. Similarly, whenever the processor develops an instruction address, its two low-order bits are zero.

Bits 0–5 always specify the primary opcode. Many instructions also have an extended opcode. The remaining bits of the instruction contain one or more fields for the different instruction formats.

Some instruction fields are reserved or must contain a predefined value as shown in the individual instruction layouts. If a reserved field does not have all bits cleared, or if a field that must contain a particular value does not contain that value, the instruction form is invalid and the results are as described in Chapter 4, “Addressing Modes and Instruction set Summary.”

Within the instruction format diagram the instruction operation code and extended operation code (if extended form) are specified in decimal. These fields have been converted to hexadecimal and are shown on line two for each instruction definition.

8.1.1 Split-Field Notation

Some instruction fields occupy more than one contiguous sequence of bits or occupy a contiguous sequence of bits used in permuted order. Such a field is called a split field. Split fields that represent the concatenation of the sequences from left to right are shown in lowercase letters. These split fields— spr, and tbr—are described in Table 8-1.

Table 8-1. Split-Field Notation and Conventions

Field	Description
spr (11–20)	This field is used to specify a special-purpose register for the **mtspr** and **mfspr** instructions. The encoding is described in Section 4.4.2.2, “Move to/from Special-Purpose Register Instructions (OEA)”**.**
tbr (11–20)	This field is used to specify either the time base lower (TBL) or time base upper (TBU).

Split fields that represent the concatenation of the sequences in some order, which need not be left to right (as described for each affected instruction), are shown in uppercase letters. These split fields - MB, ME, and SH- are described in Table 8-2.

8

8.1.2 Instruction Fields

Table 8-2 describes the instruction fields used in the various instruction formats.

Table 8-2. Instruction Syntax Conventions

Field	Description
AA (30)	Absolute address bit. 0 The immediate field represents an address relative to the current instruction address (CIA). (For more information on the CIA, see Table 8-3.) The effective (logical) address of the branch is either the sum of the LI field sign-extended to 32 bits and the address of the branch instruction or the sum of the BD field sign-extended to 32 bits and the address of the branch instruction. 1 The immediate field represents an absolute address. The effective address (EA) of the branch is the LI field sign-extended to 32 bits or the BD field sign-extended to 32 bits. **Note:** The LI and BD fields are sign-extended to 32 bits.
BD (16–29)	Immediate field specifying a 14-bit signed two’s complement branch displacement that is concatenated on the right with 0b00 and sign-extended to 32 bits.
BI (11–15)	This field is used to specify a bit in the CR to be used as the condition of a branch conditional instruction.
BO (6–10)	This field is used to specify options for the branch conditional instructions. The encoding is described in Section 4.2.4.2, “Conditional Branch Control”.
crbA (11–15)	This field is used to specify a bit in the CR to be used as a source.
crbB (16–20)	This field is used to specify a bit in the CR to be used as a source.
crbD (6–10)	This field is used to specify a bit in the CR, or in the FPSCR, as the destination of the result of an instruction.
crfD (6–8)	This field is used to specify one of the CR fields, or one of the FPSCR fields, as a destination.
crfS (11–13)	This field is used to specify one of the CR fields, or one of the FPSCR fields, as a source.

Table 8-2. Instruction Syntax Conventions (Continued)

Field	Description
CRM (12–19)	This field mask is used to identify the CR fields that are to be updated by the **mtcrf** instruction.
d (16–31)	Immediate field specifying a signed two’s complement integer that is sign-extended to 32 bits.
FM (7–14)	This field mask is used to identify the FPSCR fields that are to be updated by the **mtfsf** instruction.
frA (11–15)	This field is used to specify an FPR as a source.
frB (16–20)	This field is used to specify an FPR as a source.
frC (21–25)	This field is used to specify an FPR as a source.
frD (6–10)	This field is used to specify an FPR as the destination.
frS (6–10)	This field is used to specify an FPR as a source.
IMM (16–19)	Immediate field used as the data to be placed into a field in the FPSCR.
LI (6–29)	Immediate field specifying a 24-bit signed two’s complement integer that is concatenated on the right with 0b00 and sign-extended to 32 bits.
LK (31)	Link bit. 0 Does not update the link register (LR). 1 Updates the LR. If the instruction is a branch instruction, the address of the instruction following the branch instruction is placed into the LR.
MB (21–25) and ME (26–30)	These fields are used in rotate instructions to specify a 32-bit mask consisting of 1 bits from bit MB through bit ME inclusive, and 0 bits elsewhere, as described in Section 4.2.1.4, "Integer Rotate and Shift Instructions,".
NB (16–20)	This field is used to specify the number of bytes to move in an immediate string load or store.
OE (21)	This field is used for extended arithmetic to enable setting OV and SO in the XER.
OPCD (0–5)	Primary opcode field
rA (11–15)	This field is used to specify a GPR to be used as a source or destination.
rB (16–20)	This field is used to specify a GPR to be used as a source.
Rc (31)	Record bit. 0 Does not update the condition register (CR). 1 Updates the CR to reflect the result of the operation. For integer instructions, CR bits 0–2 are set to reflect the result as a signed quantity and CR bit 3 receives a copy of the summary overflow bit, XER[SO]. The result as an unsigned quantity or a bit string can be deduced from the EQ bit. For floating-point instructions, CR bits 4–7 are set to reflect floating-point exception, floating-point enabled exception, floating-point invalid operation exception, and floating-point overflow exception. (**Note:** Exceptions are referred to as interrupts in the architecture specification.)
rD (6–10)	This field is used to specify a GPR to be used as a destination.
rS (6–10)	This field is used to specify a GPR to be used as a source.
SH (16–20)	This field is used to specify a shift amount.
SIMM (16–31)	This immediate field is used to specify a 16-bit signed integer.
SR (12–15)	This field is used to specify one of the 16 segment registers.

Table 8-2. Instruction Syntax Conventions (Continued)

Field	Description
TO (6–10)	This field is used to specify the conditions on which to trap. The encoding is described in Section 4.2.4.6, “Trap Instructions.”
UIMM (16–31)	This immediate field is used to specify a 16-bit unsigned integer.
XO (21–30, 22–30, 26–30)	Extended opcode field.

8.1.3 Notation and Conventions

The operation of some instructions is described by a semiformal language (pseudocode). See Table 8-3 for a list of pseudocode notation and conventions used throughout this chapter

Table 8-3. Notation and Conventions

Notation/Convention	Meaning
←	Assignment
$\leftarrow_{iea}$	Assignment of an instruction effective address. .
¬	NOT logical operator
*	Multiplication
÷	Division (yielding quotient)
+	Two’s-complement addition
–	Two’s-complement subtraction, unary minus
=, ≠	Equals and Not Equals relations
<, ≤, ≥, >	Signed comparison relations
. (period)	Update. When used as a character of an instruction mnemonic, a period (.) means that the instruction updates the condition register field.
c	Carry. When used as a character of an instruction mnemonic, a ‘c’ indicates a carry out in XER[CA].
e	Extended Precision. When used as the last character of an instruction mnemonic, an ‘e’ indicates the use of XER[CA] as an operand in the instruction and records a carry out in XER[CA].
o	Overflow. When used as a character of an instruction mnemonic, an ‘o’ indicates the record of an overflow in XER[OV] and CR0[SO] for integer instructions or CR1[SO] for floating-point instructions.
<U, >U	Unsigned comparison relations
?	Unordered comparison relation
&, |	AND, OR logical operators
||	Used to describe the concatenation of two values (that is, 010 || 111 is the same as 010111)
⊕, ≡	Exclusive-OR, Equivalence logical operators (for example, (a ≡ b) = (a ⊕ ¬ b))

Table 8-3. Notation and Conventions (Continued)

Notation/Convention	Meaning
0b*nnnn*	A number expressed in binary format.
0x*nnnn or* x'nnnn nnnn'	A number expressed in hexadecimal format.
(*n*)x	The replication of x, *n* times (that is, x concatenated to itself *n* – 1 times). (*n*)0 and (*n*)1 are special cases. A description of the special cases follows: • (*n*)0 means a field of *n* bits with each bit equal to 0. Thus (5)0 is equivalent to 0b00000. • (*n*)1 means a field of *n* bits with each bit equal to 1. Thus (5)1 is equivalent to 0b11111.
(**r**A|0)	The contents of **r**A if the **r**A field has the value 1–31, or the value 0 if the **r**A field is 0.
(**r**X)	The contents of **r**X
x[*n*]	*n* is a bit or field within x, where x is a register
x^n	x is raised to the *n*th power
ABS(x)	Absolute value of x
CEIL(x)	Least integer ≥ x
Characterization	Reference to the setting of status bits in a standard way that is explained in the text.
CIA	Current instruction address. The 32-bit address of the instruction being described by a sequence of pseudocode. Used by relative branches to set the next instruction address (NIA) and by branch instructions with LK = 1 to set the link register. Does not correspond to any architected register.
Clear	Clear the leftmost or rightmost *n* bits of a register to 0. This operation is used for rotate and shift instructions.
Clear left and shift left	Clear the leftmost *b* bits of a register, then shift the register left by *n* bits. This operation can be used to scale a known non-negative array index by the width of an element. These operations are used for rotate and shift instructions.
Cleared	Bits are set to 0.
Do	Do loop. • Indenting shows range. • "To" and/or "by" clauses specify incrementing an iteration variable. • "While" clauses give termination conditions.
DOUBLE(x)	Result of converting x from floating-point single-precision format to floating-point double-precision format.
Extract	Select a field of *n* bits starting at bit position *b* in the source register, right or left justify this field in the target register, and clear all other bits of the target register to zero. This operation is used for rotate and shift instructions.
EXTS(x)	Result of extending x on the left with sign bits
GPR(x)	General-purpose register x
if...then...else...	Conditional execution, indenting shows range, else is optional.

Table 8-3. Notation and Conventions (Continued)

Notation/Convention	Meaning
Insert	Select a field of *n* bits in the source register, insert this field starting at bit position *b* of the target register, and leave other bits of the target register unchanged. (No simplified mnemonic is provided for insertion of a field when operating on double words; such an insertion requires more than one instruction.) This operation is used for rotate and shift instructions. (**Note:** Simplified mnemonics are referred to as extended mnemonics in the architecture specification.)
Leave	Leave innermost do loop, or the do loop described in leave statement.
MASK(x, y)	Mask having ones in positions x through y (wrapping if x > y) and zeros elsewhere.
MEM(x, y)	Contents of y bytes of memory starting at address x.
NIA	Next instruction address, which is the 32-bit address of the next instruction to be executed (the branch destination) after a successful branch. In pseudocode, a successful branch is indicated by assigning a value to NIA. For instructions which do not branch, the next instruction address is CIA + 4. Does not correspond to any architected register.
OEA	PowerPC operating environment architecture
Rotate	Rotate the contents of a register right or left *n* bits without masking. This operation is used for rotate and shift instructions.
Reserved	An unused field, must be left with zeros.
ROTL(x, y)	Result of rotating the value x left y positions, where x is 32 bits long
Set	Bits are set to 1.
Shift	Shift the contents of a register right or left *n* bits, clearing vacated bits (logical shift). This operation is used for rotate and shift instructions.
SINGLE(x)	Result of converting x from floating-point double-precision format to floating-point single-precision format.
SPR(x)	Special-purpose register x
TRAP	Invoke the system trap handler.
Undefined	An undefined value. The value may vary from one implementation to another, and from one execution to another on the same implementation.
UISA	PowerPC user instruction set architecture
VEA	PowerPC virtual environment architecture

Table 8-4 describes instruction field notation conventions used throughout this chapter.

Table 8-4. Instruction Field Conventions

The Architecture Specification	Equivalent to:
BA, BB, BT	**crb**A, **crb**B, **crb**D (respectively)
BF, BFA	**crf**D, **crf**S (respectively)
D	d
DS	ds
FLM	FM
FRA, FRB, FRC, FRT, FRS	**fr**A, **fr**B, **fr**C, **fr**D, **fr**S (respectively)
FXM	CRM
RA, RB, RT, RS	**r**A, **r**B, **r**D, **r**S (respectively)
SI	SIMM
U	IMM
UI	UIMM
/, *//*, *///*	0...0 (shaded)

Precedence rules for pseudocode operators are summarized in Table 8-5.

Table 8-5. Precedence Rules

Operators	Associativity
x[*n*], function evaluation	Left to right
(*n*)x or replication, x(*n*) or exponentiation	Right to left
unary –, ¬	Right to left
*, ÷	Left to right
+, –	Left to right
||	Left to right
=, ≠, <, ≤, >, ≥, <U, >U, ?	Left to right
&, ⊕, ≡	Left to right
|	Left to right
– (range)	None
←, ←iea	None

Operators higher in Table 8-5 are applied before those lower in the table. Operators at the same level in the table associate from left to right, from right to left, or not at all, as shown. For example, "–" (unary minus) associates from left to right, so a – b – c = (a – b) – c.

Parentheses are used to override the evaluation order implied by Table 8-5, or to increase clarity; parenthesized expressions are evaluated before serving as operands.

8.1.4 Computation Modes

The PowerPC architecture is defined for 32-bit implementations, in which all registers except the FPRs are 32 bits long, and effective addresses are 32 bits long. The FPR registers are 64 bits long. For more information on computation modes see Section 4.1.1, "Computation Modes."

8.2 PowerPC Instruction Set

The remainder of this chapter lists and describes the instruction set for the PowerPC architecture. The instructions are listed in alphabetical order by mnemonic. Figure 8-1 shows the format for each instruction description page.

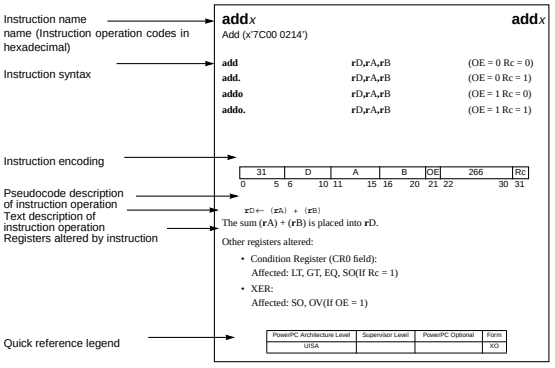


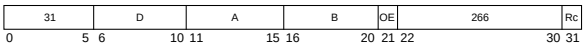
Figure 8-1. Instruction Description

NOTE: The execution unit that executes the instruction may not be the same for all PowerPC processors.

add*x* add*x*

Add (x'7C00 0214')

add **r**D,**r**A,**r**B (OE = 0 Rc = 0)
add. **r**D,**r**A,**r**B (OE = 0 Rc = 1)
addo **r**D,**r**A,**r**B (OE = 1 Rc = 0)
addo. **r**D,**r**A,**r**B (OE = 1 Rc = 1)




```
rD← (rA) + (rB)
```

The sum (**r**A) + (**r**B) is placed into **r**D.

The **add** instruction is preferred for addition because it sets few status bits.

Other registers altered:

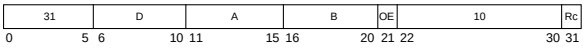
- Condition Register (CR0 field):
  Affected: LT, GT, EQ, SO (If Rc = 1)
  NOTE: CR0 field may not reflect the infinitely precise result if overflow occurs (see next bullet item.
- XER:
  Affected: SO, OV (If OE = 1)
  NOTE: For more information on condition codes see Section 2.1.3, “Condition Register,” and Section 2.1.5, “XER Register.”

PowerPC Architecture Level	Supervisor Level	PowerPC Optional	Form
UISA			XO

addc*x* addc*x*

Add Carrying (x'7C00 0014')

addc **r**D,**r**A,**r**B (OE = 0 Rc = 0)
addc. **r**D,**r**A,**r**B (OE = 0 Rc = 1)
addco **r**D,**r**A,**r**B (OE = 1 Rc = 0)
addco. **r**D,**r**A,**r**B (OE = 1 Rc = 1)




```
rD← (rA) + (rB)
```

The sum (**r**A) + (**r**B) is placed into **r**D.

Other registers altered:

- Condition Register (CR0 field):
  Affected: LT, GT, EQ, SO (If Rc = 1)
  NOTE: CR0 field may not reflect the infinitely precise result if overflow occurs (see next bullet item).
- XER:
  Affected: CA
  Affected: SO, OV (If OE = 1)

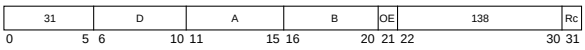
  NOTE: For more information on condition codes see Section 2.1.3, “Condition Register,” and Section 2.1.5, “XER Register.”

PowerPC Architecture Level	Supervisor Level	PowerPC Optional	Form
UISA			XO

adde*x* adde*x*

Add Extended (x'7C00 0114')

adde **r**D,**r**A,**r**B (OE = 0 Rc = 0)
adde. **r**D,**r**A,**r**B (OE = 0 Rc = 1)
addeo **r**D,**r**A,**r**B (OE = 1 Rc = 0)
addeo. **r**D,**r**A,**r**B (OE = 1 Rc = 1)




```
rD← (rA) + (rB) + XER[CA]
```

The sum (**r**A) + (**r**B) + XER[CA] is placed into **r**D.

Other registers altered:

- Condition Register (CR0 field):
  Affected: LT, GT, EQ, SO (If Rc = 1)
  NOTE: CR0 field may not reflect the infinitely precise result if overflow occurs (see next bullet item).
- XER:
  Affected: CA
  Affected: SO, OV (If OE = 1)

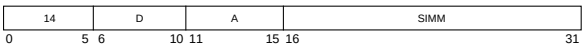
  NOTE: For more information on condition codes see Section 2.1.3, “Condition Register,” and Section 2.1.5, “XER Register.”

PowerPC Architecture Level	Supervisor Level	PowerPC Optional	Form
UISA			XO

addi addi

Add Immediate (x'3800 0000')

addi **r**D,**r**A,SIMM




```
if rA = 0
        then rD ←EXTS(SIMM)
        else rD←(rA) + EXTS(SIMM)
```

The sum (**r**A|0) + sign extended SIMM is placed into **r**D.

The **addi** instruction is preferred for addition because it sets few status bits.

NOTE: **addi** uses the value 0, not the contents of GPR0, if **r**A = 0.

Other registers altered:

- None

Simplified mnemonics:

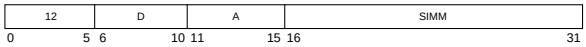
li	**r**D,value	equivalent to	**addi**	**r**D,**0,**value
la	**r**D,disp**(r**A**)**	equivalent to	**addi**	**r**D,**r**A,disp
subi	**r**D,**r**A,value	equivalent to	**addi**	**r**D,**r**A,–value

PowerPC Architecture Level	Supervisor Level	PowerPC Optional	Form
UISA			D

addic addic

Add Immediate Carrying (x'3000 0000')

addic **r**D,**r**A,SIMM




```
rD← (rA) + EXTS(SIMM)
```

The sum (**r**A) + sign extended SIMM is placed into **r**D.

Other registers altered:

- XER:

  Affected: CA

  NOTE: The setting of the affected bits in the XER reflects overflow of the 32-bit result. For more information see Section 2.1.5, "XER Register".

Simplified mnemonics:

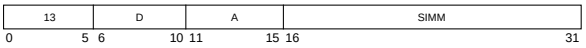
subic **r**D,**r**A,value equivalent to **addic** **r**D,**r**A,–value

PowerPC Architecture Level	Supervisor Level	PowerPC Optional	Form
UISA			D

addic. addic.

Add Immediate Carrying and Record (x'3400 0000')

addic. **r**D,**r**A,SIMM




```
rD← (rA) + EXTS(SIMM)
```

The sum (**r**A) + the sign extended SIMM is placed into **r**D.

Other registers altered:

- Condition Register (CR0 field):

  Affected: LT, GT, EQ, SO

  NOTE: CR0 field may not reflect the infinitely precise result if overflow occurs (see next bullet item).

- XER:

  Affected: CA

  NOTE: For more information on condition codes see Section 2.1.3, "Condition Register," and Section 2.1.5, "XER Register".

Simplified mnemonics:

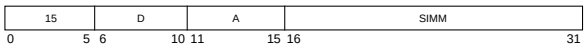
subic. **r**D,**r**A,value equivalent to **addic.** **r**D,**r**A,–value

PowerPC Architecture Level	Supervisor Level	PowerPC Optional	Form
UISA			D

addis addis

Add Immediate Shifted (x'3C00 0000')

addis **r**D,**r**A,SIMM




```
if rA = 0
        then rD← (SIMM || (16)0)
        else rD← (rA) + (SIMM || (16)0)
```

The sum (**r**A|0) + (SIMM || 0x0000) is placed into **r**D.

The **addis** instruction is preferred for addition because it sets few status bits.

NOTE: **addis** uses the value 0, not the contents of GPR0, if **r**A = 0.

Other registers altered:

- None

Simplified mnemonics:

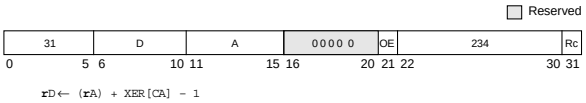
lis **r**D,value equivalent to **addis** **r**D,**0,**value
subis **r**D,**r**A,value equivalent to **addis** **r**D,**r**A,–value

PowerPC Architecture Level	Supervisor Level	PowerPC Optional	Form
UISA			D

addme*x* addme*x*

Add to Minus One Extended (x'7C00 01D4')

addme **r**D,**r**A (OE = 0 Rc = 0)
addme. **r**D,**r**A (OE = 0 Rc = 1)
addmeo **r**D,**r**A (OE = 1 Rc = 0)
addmeo. **r**D,**r**A (OE = 1 Rc = 1)




```
rD← (rA) + XER[CA] - 1
```

The sum (**r**A) + XER[CA] + 0xFFFF_FFFF is placed into **r**D.

Other registers altered:

- Condition Register (CR0 field):

  Affected: LT, GT, EQ, SO (If Rc = 1)

  NOTE: CR0 field may not reflect the infinitely precise result if overflow occurs (see next bullet item).

- XER:

  Affected: CA

  Affected: SO, OV (If OE = 1)

  NOTE: For more information on condition codes see Section 2.1.3, "Condition Register," and Section 2.1.5, "XER Register".

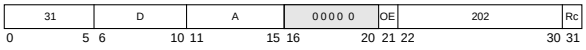
PowerPC Architecture Level	Supervisor Level	PowerPC Optional	Form
UISA			XO

addze*x* addze*x*

Add to Zero Extended (x'7C00 0194')

addze **r**D,**r**A (OE = 0 Rc = 0)
addze. **r**D,**r**A (OE = 0 Rc = 1)
addzeo **r**D,**r**A (OE = 1 Rc = 0)
addzeo. **r**D,**r**A (OE = 1 Rc = 1)

Reserved




```
rD← (rA) + XER[CA]
```

The sum (**r**A) + XER[CA] is placed into **r**D.

Other registers altered:

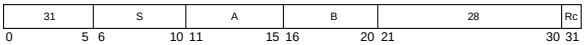
- Condition Register (CR0 field):
  Affected: LT, GT, EQ, SO (If Rc = 1)
  NOTE: CR0 field may not reflect the infinitely precise result if overflow occurs (see next).
- XER:
  Affected: CA
  Affected: SO, OV (If OE = 1)
  NOTE: For more information on condition codes see Section 2.1.3, “Condition Register,” and Section 2.1.5, “XER Register”.

PowerPC Architecture Level	Supervisor Level	PowerPC Optional	Form
UISA			XO

and*x* and*x*

AND (x'7C00 0038')

and **r**A,**r**S,**r**B (Rc = 0)
and. **r**A,**r**S,**r**B (Rc = 1)




```
rA← (rS) & (rB)
```

The contents of **r**S are ANDed with the contents of **r**B and the result is placed into **r**A.

Other registers altered:

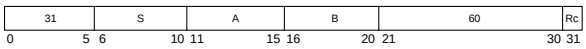
- Condition Register (CR0 field):
  Affected: LT, GT, EQ, SO (If Rc = 1)

PowerPC Architecture Level	Supervisor Level	PowerPC Optional	Form
UISA			X

andc*x* andc*x*

AND with Complement (x'7C00 0078')

andc **r**A,**r**S,**r**B (Rc = 0)
andc. **r**A,**r**S,**r**B (Rc = 1)




```
rA← (rS) & ¬ (rB)
```

The contents of **r**S are ANDed with the one’s complement of the contents of **r**B and the result is placed into **r**A.

Other registers altered:

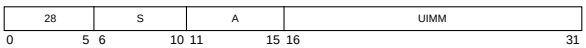
- Condition Register (CR0 field):Affected: LT, GT, EQ, SO(If Rc = 1)

PowerPC Architecture Level	Supervisor Level	PowerPC Optional	Form
UISA			X

andi. andi.

AND Immediate (x'7000 0000')

andi. **r**A,**r**S,UIMM




```
rA← (rS) & ((16)0 || UIMM)
```

The contents of **r**S are ANDed with 0x000 || UIMM and the result is placed into **r**A.

Other registers altered:

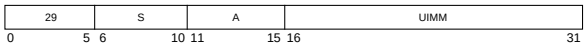
- Condition Register (CR0 field):
  Affected: LT, GT, EQ, SO

PowerPC Architecture Level	Supervisor Level	PowerPC Optional	Form
UISA			D

andis. andis.

AND Immediate Shifted (x'7400 0000')

andis. **r**A,**r**S,UIMM




```
rA← (rS) & (UIMM || (16)0)
```

The contents of **r**S are ANDed with UIMM || 0x0000 and the result is placed into **r**A.

Other registers altered:

- Condition Register (CR0 field):
  Affected: LT, GT, EQ, SO

PowerPC Architecture Level	Supervisor Level	PowerPC Optional	Form
UISA			D

bx bx

Branch (x'4800 0000')

b	target_addr	(AA = 0 LK = 0)
ba	target_addr	(AA = 1 LK = 0)
bl	target_addr	(AA = 0 LK = 1)
bla	target_addr	(AA = 1 LK = 1)


```
if AA = 1
        then NIA←iea EXTS(LI || 0b00)
        else NIA←iea CIA + EXTS(LI || 0b00)
if LK = 1
        then LR←iea CIA + 4
```

target_addr specifies the branch target address.

If AA = 1, then the branch target address is the value LI || 0b00 sign-extended.

If AA = 0, then the branch target address is the sum of LI || 0b00 sign-extended plus the address of this instruction.

If LK = 1, then the effective address of the instruction following the branch instruction is placed into the link register.

Other registers altered:

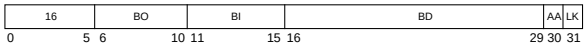
Affected: Link Register (LR) (If LK = 1)

PowerPC Architecture Level	Supervisor Level	PowerPC Optional	Form
UISA			I

bcx bcx

Branch Conditional (x'4000 0000')

bc	BO,BI,target_addr	(AA = 0 LK = 0)
bca	BO,BI,target_addr	(AA = 1 LK = 0)
bcl	BO,BI,target_addr	(AA = 0 LK = 1)
bcla	BO,BI,target_addr	(AA = 1 LK = 1)




```
if ¬ BO[2]
    then CTR ← CTR - 1
ctr_ok ← BO[2] | ((CTR ≠ 0) ⊕ BO[3])
cond_ok ← BO[0] | (CR[BI] ≡ BO[1])
if ctr_ok & cond_ok
    then
        if AA = 1
            then NIA ←iea EXTS(BD || 0b00)
            else NIA ←iea CIA + EXTS(BD || 0b00)
if LK = 1
    then LR ←iea CIA + 4
```

The BI field specifies the bit in the condition register (CR) to be used as the condition of the branch. The BO field is encoded as described in Table 8-6. Additional information about BO field encoding is provided in Section 4.2.4.2, “Conditional Branch Control”.

Table 8-6. BO Operand Encodings

BO	Description
0000*y*	Decrement the CTR, then branch if the decremented CTR 0 and the condition is FALSE.
0001*y*	Decrement the CTR, then branch if the decremented CTR = 0 and the condition is FALSE.
001*zy*	Branch if the condition is FALSE.
0100*y*	Decrement the CTR, then branch if the decremented CTR 0 and the condition is TRUE.
0101*y*	Decrement the CTR, then branch if the decremented CTR = 0 and the condition is TRUE.
011*zy*	Branch if the condition is TRUE.
1*z*00*y*	Decrement the CTR, then branch if the decremented CTR 0.
1*z*01*y*	Decrement the CTR, then branch if the decremented CTR = 0.
1*z*1*zz*	Branch always.

In this table, *z* indicates a bit that is ignored.
Note: The *z* bits should be cleared, as they may be assigned a meaning in some future version of the PowerPC architecture.

The *y* bit provides a hint about whether a conditional branch is likely to be taken, and may be used by some PowerPC implementations to improve performance.

target_addr specifies the branch target address.

If AA = 0, the branch target address is the sum of BD || 0b00 sign-extended and the address of this instruction.

If AA = 1, the branch target address is the value BD || 0b00 sign-extended.

If LK = 1, the effective address of the instruction following the branch instruction is placed into the link register.

Other registers altered:

Affected: Count Register (CTR) (If BO[2] = 0)

Affected: Link Register (LR) (If LK = 1)

Simplified mnemonics:

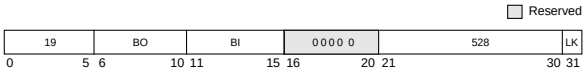
blt	target	equivalent to	**bc**	**12,0,**target
bne	**cr2**,target	equivalent to	**bc**	**4,10,**target
bdnz	target	equivalent to	**bc**	**16,0,**target

PowerPC Architecture Level	Supervisor Level	PowerPC Optional	Form
UISA			D

bcctr*x* bcctr*x*

Branch Conditional to Count Register (x'4C00 0420')

bcctr BO,BI (LK = 0)
bcctrl BO,BI (LK = 1)




```
cond_ok ← BO[0] | (CR[BI] ≡ BO[1])
if cond_ok
      then    NIA ←iea CTR[0–29] || 0b00
if LK
      then    LR ←iea CIA + 4
```

The BI field specifies the bit in the condition register to be used as the condition of the branch. The BO field is encoded as described in Table 8-7. Additional information about BO field encoding is provided in Section 4.2.4.2, "Conditional Branch Control".

8

Table 8-7. BO Operand Encodings

BO	Description
0000*y*	Decrement the CTR, then branch if the decremented CTR 0 and the condition is FALSE.
0001*y*	Decrement the CTR, then branch if the decremented CTR = 0 and the condition is FALSE.
001*zy*	Branch if the condition is FALSE.
0100*y*	Decrement the CTR, then branch if the decremented CTR 0 and the condition is TRUE.
0101*y*	Decrement the CTR, then branch if the decremented CTR = 0 and the condition is TRUE.
011*zy*	Branch if the condition is TRUE.
1*z*00*y*	Decrement the CTR, then branch if the decremented CTR 0.
1*z*01*y*	Decrement the CTR, then branch if the decremented CTR = 0.
1*z*1*zz*	Branch always.

In this table, *z* indicates a bit that is ignored.
Note: The *z* bits should be cleared, as they may be assigned a meaning in some future version of the PowerPC architecture.

The *y* bit provides a hint about whether a conditional branch is likely to be taken, and may be used by some PowerPC implementations to improve performance.

The branch target address is CTR[0–29] || 0b00.

If LK = 1, the effective address of the instruction following the branch instruction is placed into the link register.

If the "decrement and test CTR" option is specified (BO[2] = 0), the instruction form is invalid.

Other registers altered:

Affected: Link Register (LR) (If LK = 1)

Simplified mnemonics:

bltctr equivalent to **bcctr 12,0**
bnectr cr2 equivalent to **bcctr 4,10**

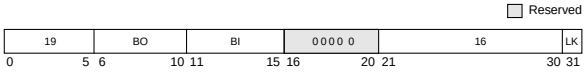
8

PowerPC Architecture Level	Supervisor Level	PowerPC Optional	Form
UISA			XL

bclr*x* bclr*x*

Branch Conditional to Link Register (x'4C00 0020')

bclr BO,BI (LK = 0)
bclrl BO,BI (LK = 1)




```
if ¬ BO[2]
      then CTR ← CTR – 1
ctr_ok ← BO[2] | ((CTR ≠   0)⊕ BO[3])
cond_ok ← BO[0] | (CR[BI] ≡ BO[1])
if ctr_ok & cond_ok
      then   NIA ←iea LR[0–29] || 0b00
if LK
      then LR ←iea CIA + 4
```

The BI field specifies the bit in the condition register to be used as the condition of the branch. The BO field is encoded as described in Table 8-8. Additional information about BO field encoding is provided in Section 4.2.4.2, "Conditional Branch Control".

8

Table 8-8. BO Operand Encodings

BO	Description
0000*y*	Decrement the CTR, then branch if the decremented CTR 0 and the condition is FALSE.
0001*y*	Decrement the CTR, then branch if the decremented CTR = 0 and the condition is FALSE.
001*zy*	Branch if the condition is FALSE.
0100*y*	Decrement the CTR, then branch if the decremented CTR 0 and the condition is TRUE.
0101*y*	Decrement the CTR, then branch if the decremented CTR = 0 and the condition is TRUE.
011*zy*	Branch if the condition is TRUE.
1*z*00*y*	Decrement the CTR, then branch if the decremented CTR 0.
1*z*01*y*	Decrement the CTR, then branch if the decremented CTR = 0.
1*z*1*zz*	Branch always.

If the BO field specifies that the CTR is to be decremented, the entire 32-bit CTR is decremented.

In this table, *z* indicates a bit that is ignored.
Note: The *z* bits should be cleared, as they may be assigned a meaning in some future version of the PowerPC architecture.

The *y* bit provides a hint about whether a conditional branch is likely to be taken, and may be used by some PowerPC implementations to improve performance.

The branch target address is LR[0–29] || 0b00.

If LK = 1, then the effective address of the instruction following the branch instruction is placed into the link register.

Other registers altered:

Affected: Count Register (CTR) (If BO[2] = 0)

Affected: Link Register (LR) (If LK = 1)

Simplified mnemonics:

bltlr equivalent to **bclr 12,0**
bnelr cr2 equivalent to **bclr 4,10**
bdnzlr equivalent to **bclr 16,0**

8

PowerPC Architecture Level	Supervisor Level	PowerPC Optional	Form
UISA			XL

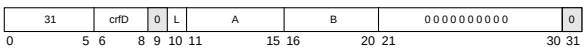
cmp

cmp

Compare (x'7C00 0000')

cmp **crf**D,L,**r**A,**r**B

Reserved




```
a ← (rA)
b ← (rB)
if  a < b
    then c ← 0b100
    else if a > b
            then c ← 0b010
            else c ← 0b001
CR[(4 * crfD)-(4 * crfD + 3)] ← c || XER[SO]
```

The contents of **r**A are compared with the contents of **r**B, treating the operands as signed integers. The result of the comparison is placed into CR field **crf**D.

NOTE: If L = 1, the instruction form is invalid.

8

Other registers altered:

- Condition Register (CR field specified by operand **crf**D):

  Affected: LT, GT, EQ, SO

Simplified mnemonics:

cmpd **r**A,**r**B	equivalent to	**cmp**	**0,1,r**A,**r**B
cmpw **cr3,r**A,**r**B	equivalent to	**cmp**	**3,0,r**A,**r**B

PowerPC Architecture Level	Supervisor Level	PowerPC Optional	Form
UISA			X

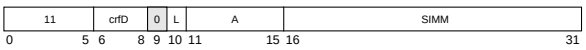
cmpi

cmpi

Compare Immediate (x'2C00 0000')

cmpi **crf**D,L,**r**A,SIMM

Reserved




```
a ← (rA)
if   a < EXTS(SIMM)
     then c ← 0b100
     else if a > EXTS(SIMM)
             then c ← 0b010
             else c ← 0b001
CR[(4 * crfD)-(4 * crfD + 3)] ← c || XER[SO]
```

The contents of **r**A are compared with the sign-extended value of the SIMM field, treating the operands as signed integers. The result of the comparison is placed into CR field **crf**D.

8

NOTE: If L = 1, the instruction form is invalid.

Other registers altered:

- Condition Register (CR field specified by operand **crf**D):

  Affected: LT, GT, EQ, SO

Simplified mnemonics:

cmpdi **r**A,value	equivalent to	**cmpi**	**0,1,r**A,value
cmpwi **cr3,r**A,value	equivalent to	**cmpi**	**3,0,r**A,value

PowerPC Architecture Level	Supervisor Level	PowerPC Optional	Form
UISA			D

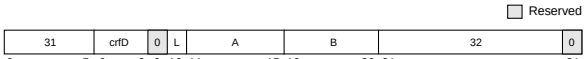
cmpl

cmpl

Compare Logical (x'7C00 0040')

cmpl **crf**D,L,**r**A,**r**B

Reserved




```
a ← (rA)
b ← (rB)
if    a <U b
      then c ← 0b100
      else if a >U b
              then c ← 0b010
              else c ← 0b001
CR[(4 * crfD)-(4 * crfD + 3)] ← c || XER[SO]
```

The contents of **r**A are compared with the contents of **r**B, treating the operands as unsigned integers. The result of the comparison is placed into CR field **crf**D.

NOTE: If L = 1, the instruction form is invalid.

8

Other registers altered:

- Condition Register (CR field specified by operand **crf**D):

  Affected: LT, GT, EQ, SO

Simplified mnemonics:

cmpld **r**A,**r**B	equivalent to	**cmpl**	**0,1,r**A,**r**B
cmplw **cr3,r**A,**r**B	equivalent to	**cmpl**	**3,0,r**A,**r**B

PowerPC Architecture Level	Supervisor Level	PowerPC Optional	Form
UISA			X

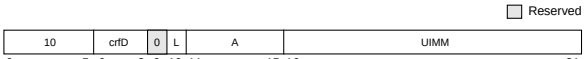
cmpli

cmpli

Compare Logical Immediate (x'2800 0000')

cmpli **crf**D,L,**r**A,UIMM

Reserved




```
a ← (rA)
if   a <U ((16)0 || UIMM)
     then c ← 0b100
     else if a >U ((16)0 || UIMM)
             then c ← 0b010
             else c ← 0b001
CR[(4 * crfD)-(4 * crfD + 3)] ← c || XER[SO]
```

The contents of **r**A are compared with 0x0000 || UIMM, treating the operands as unsigned integers. The result of the comparison is placed into CR field **crf**D.

8

NOTE: If L = 1, the instruction form is invalid.

Other registers altered:

- Condition Register (CR field specified by operand **crf**D):

  Affected: LT, GT, EQ, SO

Simplified mnemonics:

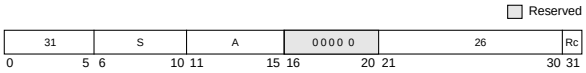
cmpldi **r** A,value	equivalent to	**cmpli**	**0,1,r**A,value
cmplwi **cr3,r**A,value	equivalent to	**cmpli**	**3,0,r**A,value

PowerPC Architecture Level	Supervisor Level	PowerPC Optional	Form
UISA			D

cntlzw*x* cntlzw*x*

Count Leading Zeros Word (x'7C00 0034')

cntlzw **r**A,**r**S (Rc = 0)
cntlzw. **r**A,**r**S (Rc = 1)




```
n ← 0
do while n < 32
     if rS[n] = 1
          then leave
     n ← n + 1
rA ← n
```

A count of the number of consecutive zero bits starting at bit 0 of **r**S is placed into **r**A. This number ranges from 0 to 32, inclusive.

Other registers altered:

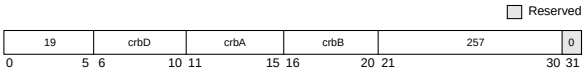
- Condition Register (CR0 field):
  Affected: LT, GT, EQ, SO (If Rc = 1)
  NOTE: If Rc = 1, then LT is cleared in the CR0 field.

PowerPC Architecture Level	Supervisor Level	PowerPC Optional	Form
UISA			X

crand crand

Condition Register AND (x'4C00 0202')

crand **crb**D,**crb**A,**crb**B




```
CR[crbD] ← CR[crbA] & CR[crbB]
```

The bit in the condition register specified by **crb**A is ANDed with the bit in the condition register specified by **crb**B. The result is placed into the condition register bit specified by **crb**D.

Other registers altered:

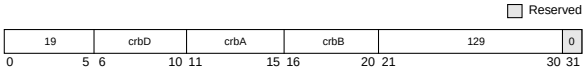
- Condition Register:
  Affected: Bit specified by operand **crb**D

PowerPC Architecture Level	Supervisor Level	PowerPC Optional	Form
UISA			XL

crandc crandc

Condition Register AND with Complement (x'4C00 0102')

crandc **crb**D,**crb**A,**crb**B




```
CR[crbD] ← CR[crbA] & ¬ CR[crbB]
```

The bit in the condition register specified by **crb**A is ANDed with the complement of the bit in the condition register specified by **crb**B and the result is placed into the condition register bit specified by **crb**D.

Other registers altered:

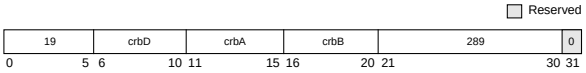
- Condition Register:
  Affected: Bit specified by operand **crb**D

PowerPC Architecture Level	Supervisor Level	PowerPC Optional	Form
UISA			XL

creqv creqv

Condition Register Equivalent (x'4C00 0242')

creqv **crb**D,**crb**A,**crb**B




```
CR[crbD] ← CR[crbA] ≡ CR[crbB]
```

The bit in the condition register specified by **crb**A is XORed with the bit in the condition register specified by **crb**B and the complemented result is placed into the condition register bit specified by **crb**D.

Other registers altered:

- Condition Register:
  Affected: Bit specified by operand **crb**D

Simplified mnemonics:

crse **crb**D equivalent to **creqv** **crb**D,**crb**D,**crb**D

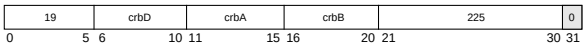
PowerPC Architecture Level	Supervisor Level	PowerPC Optional	Form
UISA			XL

crnand

Condition Register NAND (x'4C00 01C2')

crnand **crb**D,**crb**A,**crb**B

Reserved



CR[**crb**D] ← ¬ (CR[**crb**A] & CR[**crb**B])

The bit in the condition register specified by **crb**A is ANDed with the bit in the condition register specified by **crb**B and the complemented result is placed into the condition register bit specified by **crb**D.

Other registers altered:

- Condition Register:

  Affected: Bit specified by operand **crb**D

8

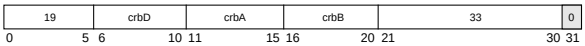
PowerPC Architecture Level	Supervisor Level	PowerPC Optional	Form
UISA			XL

crnor

Condition Register NOR (x'4C00 0042')

crnor **crb**D,**crb**A,**crb**B

Reserved



CR[**crb**D] ← ¬ (CR[**crb**A] | CR[**crb**B])

The bit in the condition register specified by **crb**A is ORed with the bit in the condition register specified by **crb**B and the complemented result is placed into the condition register bit specified by **crb**D.

Other registers altered:

- Condition Register:

  Affected: Bit specified by operand **crb**D

8

Simplified mnemonics:

crnot **crb**D,**crb**A equivalent to **crnor** **crb**D,**crb**A,**crb**A

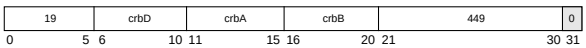
PowerPC Architecture Level	Supervisor Level	PowerPC Optional	Form
UISA			XL

cror

Condition Register OR (x'4C00 0382')

cror **crb**D,**crb**A,**crb**B

Reserved



CR[**crb**D] ← CR[**crb**A] | CR[**crb**B]

The bit in the condition register specified by **crb**A is ORed with the bit in the condition register specified by **crb**B. The result is placed into the condition register bit specified by **crb**D.

Other registers altered:

- Condition Register:

  Affected: Bit specified by operand **crb**D

8

Simplified mnemonics:

crmove **crb**D,**crb**A equivalent to **cror** **crb**D,**crb**A,**crb**A

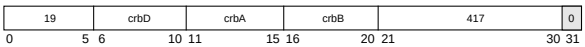
PowerPC Architecture Level	Supervisor Level	PowerPC Optional	Form
UISA			XL

crorc

Condition Register OR with Complement (x'4C00 0342')

crorc **crb**D,**crb**A,**crb**B

Reserved



CR[**crb**D] ← CR[**crb**A] | ¬ CR[**crb**B]

The bit in the condition register specified by **crb**A is ORed with the complement of the condition register bit specified by **crb**B and the result is placed into the condition register bit specified by **crb**D.

Other registers altered:

- Condition Register:

  Affected: Bit specified by operand **crb**D

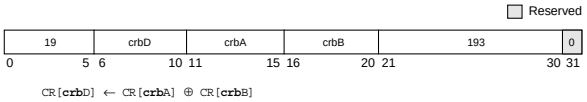
8

PowerPC Architecture Level	Supervisor Level	PowerPC Optional	Form
UISA			XL

crxor

Condition Register XOR (x'4C00 0182')

crxor **crb**D,**crb**A,**crb**B



The bit in the condition register specified by **crb**A is XORed with the bit in the condition register specified by **crb**B and the result is placed into the condition register specified by **crb**D.

Other registers altered:

- Condition Register:

  Affected: Bit specified by **crb**D

Simplified mnemonics:

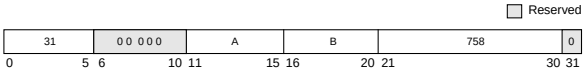
crclr **crb**D equivalent to **crxor** **crb**D,**crb**D,**crb**D

PowerPC Architecture Level	Supervisor Level	PowerPC Optional	Form
UISA			XL

dcba

Data Cache Block Allocate (x'7C00 05EC')

dcba **r**A,**r**B



EA is the sum (**r**A|0) + (**r**B).

The **dcba** instruction allocates the block in the data cache addressed by EA, by marking it valid without reading the contents of the block from memory; the data in the cache block is considered to be undefined after this instruction completes. This instruction is a hint that the program will probably soon store into a portion of the block, but the content of the rest of the block are not meaningful to the program (eliminating the needed to read the block from main memory), and can provide for improved performance in these code sequences.

The **dcba** instruction executes as follows:

- If the cache block containing the byte addressed by EA is in the data cache, the contents of all bytes are made undefined but the cache block is still considered valid.

  NOTE: Programming errors can occur if the data in this cache block is subsequently read or used inadvertently.

- If the cache block containing the byte addressed by EA is **not** in the data cache and the corresponding memory page or block is caching-allowed, the cache block is allocated (and made valid) in the data cache without fetching the block from main memory, and the value of all bytes is undefined.
- If the addressed byte corresponds to a cache-inhibited page or block this instruction is treated as a no-op. (i.e. if the I bit is set),
- If the cache block containing the byte addressed by EA is in coherency-required memory, and the cache block exists in the data cache(s) of any other processor(s), it is kept coherent in those caches (i.e. the processor preforms the appropriate bus transactions to enforce this).

This instruction is treated as a store to the addressed byte with respect to address translation and memory protection, referenced and changed recording and the ordering enforced by **eieio** or by the combination of caching-inhibited and guarded attributes for a page (or block). However, the DSI exception is not invoked for a translation or protection violation, and the referenced and changed bits need not be updated when the page or block is cache-inhibited (causing the instruction to be treated as a no-op).

NOTE: This instruction is optional in the PowerPC architecture.

Other registers altered:

- None

In the PowerPC OEA, the **dcba** instruction is additionally defined to clear all bytes of a newly established block to zero in the case that the block did not already exist in the cache.

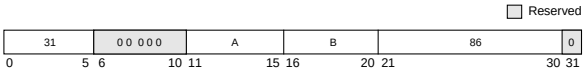
Additionally, as the **dcba** instruction may establish a block in the data cache without verifying that the associated physical address is valid, a delayed machine check exception is possible. See Chapter 6, “Exceptions,” for a discussion about this type of machine check exception.

PowerPC Architecture Level	Supervisor Level	PowerPC Optional	Form
VEA		√	X

dcbf

Data Cache Block Flush (x'7C00 00AC')

dcbf **r**A,**r**B



EA is the sum (**r**A|0) + (**r**B).

The action taken depends on the memory mode associated with the block containing the byte addressed by EA and on the state of that block. If the system is a multiprocessor implementation and the block is marked coherency-required, the processor will, if necessary, send an address-only broadcast to other processors. The broadcast of the **dcbf** instruction causes another processor to copy the block to memory, if it has dirty data, and then invalidate the block from the cache. The list below describes the action taken for the two states of the memory coherency attribute (M bit).

- Coherency required (requires the use of address broadcast)
  - — Unmodified block—Invalidates copies of the block in the data caches of all processor.
  - — Modified block—Copies the block to memory and invalidates it. (In what ever processor it resides, there should be only one modified block)
  - — Absent block —If a modified copy of the block is in the data cache of another processor, causes that processor to copied to memory and invalidated it in it’s data cache. If unmodified copies are in the data caches of other processors, causes those copies to be invalidated in those data caches.
- Coherency not required (no address broadcast required)
  - — Unmodified block—Invalidates the block in the processor’s data cache.
  - — Modified block—Copies the block to memory. Invalidates the block in the processor’s data cache.
  - — Absent block—No action is taken.

The function of this instruction is independent of the write-through, write-back and caching-inhibited/allowed modes of the block containing the byte addressed by EA. This instruction is treated as a load from the addressed byte with respect to address translation and memory protection. It is also treated as a load for referenced and changed bit recording except that referenced and changed bit recording may not occur.

Other registers altered: None

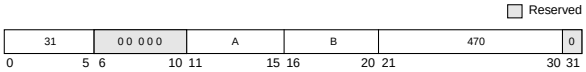
PowerPC Architecture Level	Supervisor Level	PowerPC Optional	Form
VEA			X

dcbi

dcbi

Data Cache Block Invalidate (x'7C00 03AC')

dcbi **r**A,**r**B



EA is the sum (**r**A|0) + (**r**B).

The action taken is dependent on the memory mode associated with the block containing the byte addressed by EA and on the state of that block. The list below describes the action taken if the block containing the byte addressed by EA is or is not in the cache.

- Coherency required (requires the use of address broadcast)
  - — Unmodified block—Invalidates copies of the block in the data caches of all processor.
  - — Modified block—Invalidates the copy of the block in the data cache in the processor(s) where it is found. (Discards any modified contents)
  - — Absent block —If a modified copy of the block is in the data cache of another processor, causes that processor to invalidated it in it's data cache. If unmodified copies are in the data caches of other processors, causes those copies to be invalidated in those data caches.
- Coherency not required (no address broadcast required)
  - — Unmodified block—Invalidates the block in the processor's data cache.
  - — Modified block— Invalidates the block in the processor's data cache. (Discards any modified contents)
  - — Absent block—No action is taken.

When data address translation is enabled, MSR[DR] = 1, and the virtual address has no translation, a DSI exception occurs.

The function of this instruction is independent of the write-through and caching-inhibited/allowed modes of the block containing the byte addressed by EA. This instruction operates as a store to the addressed byte with respect to address translation and protection. The referenced and changed bits are modified appropriately.

This is a supervisor-level instruction.

Other registers altered: None

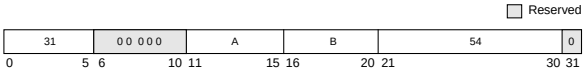
PowerPC Architecture Level	Supervisor Level	PowerPC Optional	Form
VEA	yes		X

dcbst

dcbst

Data Cache Block Store (x'7C00 006C')

dcbst **r**A,**r**B



EA is the sum (**r**A|0) + (**r**B).

The **dcbst** instruction executes as follows:

- Coherency required (requires the use of address broadcast)
  - — Unmodified block—No action in this processor. Signals other processors to copy to memory any modified cache block.
  - — Modified block—The cache block is written to memory. (only one processor should have a copy of a modified block)
  - — Absent block —No action in this processor. If a modified copy of the block is in the data cache of another processor, the cache line is written to memory.
- Coherency not required (no address broadcast required)
  - — Unmodified block—No action is taken.
  - — Modified block— The cache block is written to memory.
  - — Absent block—No action is taken.

NOTE: For modified cache blocks written to memory the architecture does not stipulate whether or not to clear the modified state of the cache block. It is left up to the processor designer to determine the final state of the cache block. Either modified or valid is logically correct.

The function of this instruction is independent of the write-through and caching-inhibited/allowed modes of the block containing the byte addressed by EA.

The processor treats this instruction as a load from the addressed byte with respect to address translation and memory protection. It is also treated as a load for referenced and changed bit recording except that referenced and changed bit recording may not occur.

Other registers altered:

- None

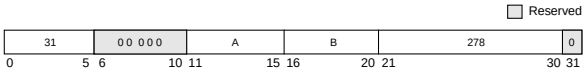
PowerPC Architecture Level	Supervisor Level	PowerPC Optional	Form
VEA			X

dcbt

dcbt

Data Cache Block Touch (x'7C00 022C')

dcbt **r**A,**r**B



EA is the sum (**r**A|0) + (**r**B).

This instruction is a hint that performance will possibly be improved if the block containing the byte addressed by EA is fetched into the data cache, because the program will probably soon load from the addressed byte. If the block is caching-inhibited, the hint is ignored and the instruction is treated as a no-op. Executing **dcbt** does not cause the system alignment error handler to be invoked.

This instruction is treated as a load from the addressed byte with respect to address translation, memory protection, and reference and change recording except that referenced and changed bit recording may not occur. Additionally, no exception occurs in the case of a translation fault or protection violation.

The program uses the **dcbt** instruction to request a cache block fetch before it is actually needed by the program. The program can later execute load instructions to put data into registers. However, the processor is not obliged to load the addressed block into the data cache.

NOTE: This instruction is defined architecturally to perform the same functions as the **dcbtst** instruction. Both are defined in order to allow implementations to differentiate the bus actions when fetching into the cache for the case of a load and for a store.

Other registers altered:

- None

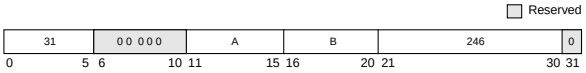
PowerPC Architecture Level	Supervisor Level	PowerPC Optional	Form
VEA			X

dcbtst

dcbtst

Data Cache Block Touch for Store (x'7C00 01EC')

dcbtst **r**A,**r**B



EA is the sum (**r**A|0) + (**r**B).

This instruction is a hint that performance will possibly be improved if the block containing the byte addressed by EA is fetched into the data cache, because the program will probably soon store from the addressed byte. If the block is caching-inhibited, the hint is ignored and the instruction is treated as a no-op. Executing **dcbtst** does not cause the system alignment error handler to be invoked.

This instruction is treated as a load from the addressed byte with respect to address translation, memory protection, and reference and change recording except that referenced and changed bit recording may not occur. Additionally, no exception occurs in the case of a translation fault or protection violation.

The program uses **dcbtst** to request a cache block fetch to potentially improve performance for a subsequent store to that EA, as that store would then be to a cached location. However, the processor is not obliged to load the addressed block into the data cache.

NOTE: This instruction is defined architecturally to perform the same functions as the **dcbt** instruction. Both are defined in order to allow implementations to differentiate the bus actions when fetching into the cache for the case of a load and for a store.

Other registers altered:

- None

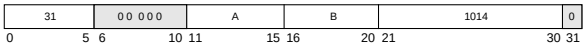
PowerPC Architecture Level	Supervisor Level	PowerPC Optional	Form
VEA			X

dcbz

Data Cache Block Clear to Zero (x'7C00 07EC')

dcbz **r**A,**r**B

Reserved



EA is the sum (**r**A|0) + (**r**B).

This instruction is treated as a store to the addressed byte with respect to address translation, memory protection, referenced and changed recording. It is also treated as a store with respect to the ordering enforced by **eieio** and the ordering enforced by the combination of caching-inhibited and guarded attributes for a page (or block).

The **dcbz** instruction executes as follows:

- If the cache block containing the byte addressed by EA is in the data cache, all bytes are cleared and the cache line is marked "M".
- If the cache block containing the byte addressed by EA is not in the data cache and the corresponding memory page or block is caching-allowed, the cache block is allocated (and made valid) in the data cache without fetching the block from main memory, and all bytes are cleared.
- If the page containing the byte addressed by EA is in caching-inhibited or write-through mode, either all bytes of main memory that correspond to the addressed cache block are cleared or the alignment exception handler is invoked. The exception handler can then clear all bytes in main memory that correspond to the addressed cache block.
- If the cache block containing the byte addressed by EA is in coherency-required mode, and the cache block exists in the data cache(s) of any other processor(s), it is kept coherent in those caches (i.e. the processor performs the appropriate bus transactions to enforce this).

Other registers altered:

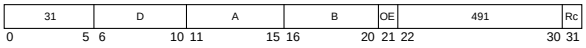
- None

PowerPC Architecture Level	Supervisor Level	PowerPC Optional	Form
VEA			X

divw*x*

Divide Word (x'7C00 03D6')

divw	**r**D,**r**A,**r**B	(OE = 0 Rc = 0)
divw.	**r**D,**r**A,**r**B	(OE = 0 Rc = 1)
divwo	**r**D,**r**A,**r**B	(OE = 1 Rc = 0)
divwo.	**r**D,**r**A,**r**B	(OE = 1 Rc = 1)




```
dividend← (rA)
divisor ← (rB)
rD ← dividend ÷ divisor
```

The dividend is the contents of **r**A. The divisor is the contents of **r**B. The remainder is not supplied as a result. Both the operands and the quotient are interpreted as signed integers. The quotient is the unique signed integer that satisfies the equation—dividend = (quotient * divisor) + r where 0 r < |divisor| (if the dividend is non-negative), and –|divisor| < r 0 (if the dividend is negative).

If an attempt is made to perform either of the divisions—0x8000_0000 ÷ –1 or <anything> ÷ 0, then the contents of **r**D are undefined, as are the contents of the LT, GT, and EQ bits of the CR0 field (if Rc = 1). In this case, if OE = 1 then OV is set.

The 32-bit signed remainder of dividing the contents of **r**A by the contents of **r**B can be computed as follows, except in the case that the contents of **r**A = -2^{31} and the contents of **r**B = –1.

```
divw    rD,rA,rB     # rD = quotient
mullw   rD,rD,rB     # rD = quotient * divisor
subf    rD,rD,rA     # rD = remainder
```

Other registers altered:

- Condition Register (CR0 field):
  Affected: LT, GT, EQ, SO (If Rc = 1)
- XER:
  Affected: SO, OV (If OE = 1)

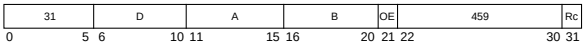
  NOTE: For more information on condition codes see Section 2.1.3, "Condition Register," and Section 2.1.5, "XER Register."

PowerPC Architecture Level	Supervisor Level	PowerPC Optional	Form
UISA			XO

divwu*x*

Divide Word Unsigned (x'7C00 0396')

divwu	**r**D,**r**A,**r**B	(OE = 0 Rc = 0)
divwu.	**r**D,**r**A,**r**B	(OE = 0 Rc = 1)
divwuo	**r**D,**r**A,**r**B	(OE = 1 Rc = 0)
divwuo.	**r**D,**r**A,**r**B	(OE = 1 Rc = 1)




```
dividend ← (rA)
divisor ← (rB)
rD← dividend ÷ divisor
```

The dividend is the contents of **r**A. The divisor is the contents of **r**B. The remainder is not supplied as a result.

Both operands and the quotient are interpreted as unsigned integers, except that if Rc = 1 the first three bits of CR0 field are set by signed comparison of the result to zero. The quotient is the unique unsigned integer that satisfies the equation—dividend = (quotient * divisor) + r (where 0 r < divisor). If an attempt is made to perform the division—<anything> 0—then the contents of **r**D are undefined as are the contents of the LT, GT, and EQ bits of the CR0 field (if Rc = 1). In this case, if OE = 1 then OV is set.

The 32-bit unsigned remainder of dividing the contents of **r**A by the contents of **r**B can be computed as follows:

```
divwu   rD,rA,rB     # rD = quotient
mullw   rD,rD,rB     # rD = quotient * divisor
subf    rD,rD,rA     # rD = remainder
```

Other registers altered:

- Condition Register (CR0 field):
  Affected: LT, GT, EQ, SO (If Rc = 1)
- XER:
  Affected: SO, OV (if OE = 1)

  NOTE: For more information on condition codes see Section 2.1.3, "Condition Register," and Section 2.1.5, "XER Register."

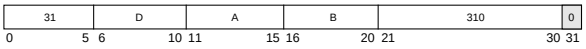
PowerPC Architecture Level	Supervisor Level	PowerPC Optional	Form
UISA			XO

eciwx

External Control In Word Indexed (x'7C00 026C')

eciwx **r**D,**r**A,**r**B

Reserved



The **eciwx** instruction and the EAR register can be very efficient when mapping special devices such as graphics devices that use addresses as pointers.

```
if rA = 0
     then b ← 0
     else b← (rA)
EA ← b + (rB)
paddr ← address translation of EA
send load word request for paddr to device identified by EAR[RID]
rD ← word from device
```

EA is the sum (**r**A|0) + (**r**B).

A load word request for the physical address (referred to as real address in the architecture specification) corresponding to EA is sent to the device identified by EAR[RID], bypassing the cache. The word returned by the device is placed in **r**D.

EAR[E] must be 1. If it is not, a DSI exception is generated.

EA must be a multiple of four. If it is not, one of the following occurs:

- A system alignment exception is generated.
- A DSI exception is generated (possible only if EAR[E] = 0).
- The results are boundedly undefined.

The **eciwx** instruction is supported for EAs that reference memory segments in which SR[T] = 1 and for EAs mapped by the DBAT registers. If the EA references a direct-store segment (SR[T] = 1), either a DSI exception occurs or the results are boundedly undefined.

NOTE: The direct-store facility is being phased out of the architecture and will not likely be supported in future devices. Thus, software should not depend on its effects.

If this instruction is executed when MSR[DR] = 0 (real addressing mode), the results are boundedly undefined.

This instruction is treated as a load from the addressed byte with respect to address translation, memory protection, referenced and changed bit recording, and the ordering performed by **eieio**.

NOTE: This instruction is optional in the PowerPC architecture.

Other registers altered:

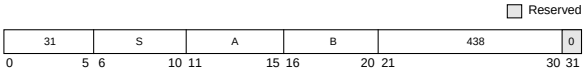
- None

PowerPC Architecture Level	Supervisor Level	PowerPC Optional	Form
VEA		√	X

ecowx

External Control Out Word Indexed (x'7C00 036C')

ecowx **rS,rA,rB**



The **ecowx** instruction and the EAR register can be very efficient when mapping special devices such as graphics devices that use addresses as pointers.

```
if rA = 0
    then b ← 0
    else b ← (rA)
EA ← b + (rB)
paddr ← address translation of EA
send store word request for paddr to device identified by EAR[RID]
send rS to device
```

EA is the sum (**r**A|0) + (**r**B).

A store word request for the physical address corresponding to EA and the contents of **r**S are sent to the device identified by EAR[RID], bypassing the cache.

EAR[E] must be 1, if it is not, a DSI exception is generated.

EA must be a multiple of four. If it is not, one of the following occurs:

- A system alignment exception is generated.
- A DSI exception is generated (possible only if EAR[E] = 0).
- The results are boundedly undefined.

The **ecowx** instruction is supported for effective addresses that reference memory segments in which SR[T] = 0, and for EAs mapped by the DBAT registers. If the EA references a direct-store segment (SR[T] = 1), either a DSI exception occurs or the results are boundedly undefined.

NOTE: The direct-store facility is being phased out of the architecture and will not likely be supported in future devices. Thus, software should not depend on its effects.

If this instruction is executed when MSR[DR] = 0 (real addressing mode), the results are boundedly undefined.

This instruction is treated as a store from the addressed byte with respect to address translation, memory protection, and referenced and changed bit recording, and the ordering performed by **eieio**.

NOTE: Software synchronization is required in order to ensure that the data access is performed in program order with respect to data accesses caused by other store or **ecowx** instructions, even though the addressed byte is assumed to be caching-inhibited and guarded.

NOTE: This instruction is optional in the PowerPC architecture.

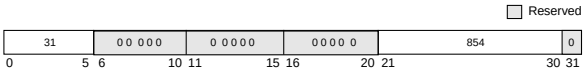
Other registers altered:

- None

PowerPC Architecture Level	Supervisor Level	PowerPC Optional	Form
VEA		√	X

eieio

Enforce In-Order Execution of I/O (x'7C00 06AC')



The **eieio** instruction provides an ordering function for the effects of load and store instructions executed by a processor. These loads and stores are divided into two sets, which are ordered separately. The memory accesses caused by a **dcbz** or a **dcba** instruction are ordered like a store. The two sets follow:

1. Loads and stores to memory that is both caching-inhibited and guarded, and stores to memory that is write-through required.

   The **eieio** instruction controls the order in which the accesses are performed in main memory. It ensures that all applicable memory accesses caused by instructions preceding the **eieio** instruction have completed with respect to main memory before any applicable memory accesses caused by instructions following the **eieio** instruction access main memory. It acts like a barrier that flows through the memory queues and to main memory, preventing the reordering of memory accesses across the barrier. No ordering is performed for **dcbz** if the instruction causes the system alignment error handler to be invoked.

   All accesses in this set are ordered as a single set—that is, there is not one order for loads and stores to caching-inhibited and guarded memory and another order for stores to write-through required memory.

2. Stores to memory that have all of the following attributes—caching-allowed, write-through not required, and memory-coherency required.

   The **eieio** instruction controls the order in which the accesses are performed with respect to coherent memory. It ensures that all applicable stores caused by instructions preceding the **eieio** instruction have completed with respect to coherent memory before any applicable stores caused by instructions following the **eieio** instruction complete with respect to coherent memory.

With the exception of **dcbz** and **dcba**, **eieio** does not affect the order of cache operations (whether caused explicitly by execution of a cache management instruction, or implicitly by the cache coherency mechanism). For more information, refer to Chapter 5, "Cache Model and Memory Coherency." The **eieio** instruction does not affect the order of accesses in one set with respect to accesses in the other set.

The **eieio** instruction may complete before memory accesses caused by instructions preceding the **eieio** instruction have been performed with respect to main memory or coherent memory as appropriate.

The **eieio** instruction is intended for use in managing shared data structures, in accessing memory-mapped I/O, and in preventing load/store combining operations in main memory. For the first use, the shared data structure and the lock that protects it must be altered only by stores that are in the same set (1 or 2; see previous discussion). For the second use, **eieio** can be thought of as placing a barrier into the stream of memory accesses issued by a processor, such that any given memory access appears to be on the same side of the barrier to both the processor and the I/O device.

Because the processor performs store operations in order to memory that is designated as both caching-inhibited and guarded (refer to Section 5.1.1, “Memory Access Ordering”), the **eieio** instruction is needed for such memory only when loads must be ordered with respect to stores or with respect to other loads.

NOTE: The **eieio** instruction does not connect hardware considerations to it such as multiprocessor implementations that send an **eieio** address-only broadcast (useful in some designs).
For example, if a design has an external buffer that re-orders loads and stores for better bus efficiency, the **eieio** broadcast signals to that buffer that previous loads/stores (marked caching-inhibited, guarded, or write-through required) must complete before any following loads/stores (marked caching-inhibited, guarded, or write-through required).

Other registers altered:

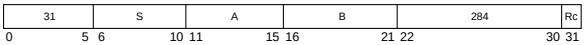
- None

PowerPC Architecture Level	Supervisor Level	PowerPC Optional	Form
VEA			X

eqv*x* eqv*x*

Equivalent (x'7C00 0238')

eqv **r**A,**r**S,**r**B (Rc = 0)
eqv. **r**A,**r**S,**r**B (Rc = 1)




```
rA ← (rS) ≡ (rB)
```

The contents of **r**S are XORed with the contents of **r**B and the complemented result is placed into **r**A.

Other registers altered:

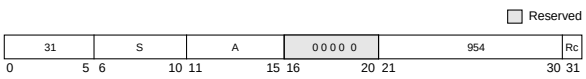
- Condition Register (CR0 field):
  Affected: LT, GT, EQ, SO (If Rc = 1)

PowerPC Architecture Level	Supervisor Level	PowerPC Optional	Form
UISA			X

extsb*x* extsb*x*

Extend Sign Byte (x'7C00 0774')

extsb **r**A,**r**S (Rc = 0)
extsb. **r**A,**r**S (Rc = 1)




```
S ← rS[24]
rA[24-31] ← rS[24-31]
rA[0-23] ← (24)S
```

The contents of the low-order eight bits of **r**S are placed into the low-order eight bits of **r**A. Bit 24of **r**S is placed into the remaining bits of **r**A.

Other registers altered:

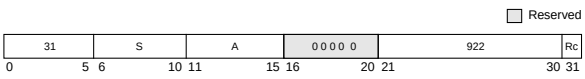
- Condition Register (CR0 field):
  Affected: LT, GT, EQ, SO (If Rc = 1)

PowerPC Architecture Level	Supervisor Level	PowerPC Optional	Form
UISA			X

extsh*x* extsh*x*

Extend Sign Half Word (x'7C00 0734')

extsh **r**A,**r**S (Rc = 0)
extsh. **r**A,**r**S (Rc = 1)




```
S ← rS[16]
rA[16-31] ← rS[16-31]
rA[0-15] ← (16)S
```

The contents of the low-order 16 bits of **r**S are placed into the low-order 16 bits of **r**A. Bit 16 of **r**S is placed into the remaining bits of **r**A.

Other registers altered:

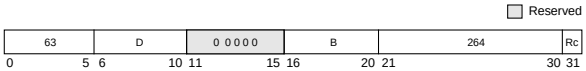
- Condition Register (CR0 field):
  Affected: LT, GT, EQ, SO (If Rc = 1)

PowerPC Architecture Level	Supervisor Level	PowerPC Optional	Form
UISA			X

fabsx

Floating Absolute Value (x'FC00 0210')

fabs **fr**D,**fr**B (Rc = 0)
fabs. **fr**D,**fr**B (Rc = 1)



The contents of **fr**B with bit 0 cleared are placed into **fr**D.

NOTE: The **fabs** instruction treats NaNs just like any other kind of value. That is, the sign bit of a NaN may be altered by **fabs**.
This instruction does not alter the FPSCR.

Other registers altered:

- Condition Register (CR1 field):
  Affected: FX, FEX, VX, OX (If Rc = 1)

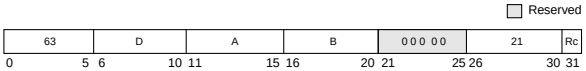
8

PowerPC Architecture Level	Supervisor Level	PowerPC Optional	Form
UISA			X

faddx

Floating Add (Double-Precision) (x'FC00 002A')

fadd **fr**D,**fr**A,**fr**B (Rc = 0)
fadd. **fr**D,**fr**A,**fr**B (Rc = 1)



The following operation is performed:

```
frD ← (frA) + (frB)
```

The floating-point operand in **fr**A is added to the floating-point operand in **fr**B. If the most-significant bit of the resultant significand is not a one, the result is normalized. The result is rounded to double-precision under control of the floating-point rounding control field RN of the FPSCR and placed into **fr**D.

8

Floating-point addition is based on exponent comparison and addition of the two significands. The exponents of the two operands are compared, and the significand accompanying the smaller exponent is shifted right, with its exponent increased by one for each bit shifted, until the two exponents are equal. The two significands are then added or subtracted as appropriate, depending on the signs of the operands. All 53 bits in the significand as well as all three guard bits (G, R, and X) enter into the computation.

If a carry occurs, the sum's significand is shifted right one bit position and the exponent is increased by one. FPSCR[FPRF] is set to the class and sign of the result, except for invalid operation exceptions when FPSCR[VE] = 1.

Other registers altered:

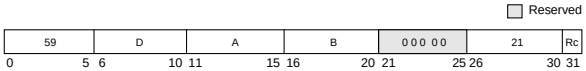
- Condition Register (CR1 field):
  Affected: FX, FEX, VX, OX (If Rc = 1)
- Floating-Point Status and Control Register:
  Affected: FPRF, FR, FI, FX, OX, UX, XX,VXSNAN, VXISI

PowerPC Architecture Level	Supervisor Level	PowerPC Optional	Form
UISA			A

faddsx

Floating Add Single (x'EC00 002A')

fadds **fr**D,**fr**A,**fr**B (Rc = 0)
fadds. **fr**D,**fr**A,**fr**B (Rc = 1)



The following operation is performed:

```
frD ← (frA) + (frB)
```

The floating-point operand in **fr**A is added to the floating-point operand in **fr**B. If the most-significant bit of the resultant significand is not a one, the result is normalized. The result is rounded to the single-precision under control of the floating-point rounding control field RN of the FPSCR and placed into **fr**D.

8

Floating-point addition is based on exponent comparison and addition of the two significands. The exponents of the two operands are compared, and the significand accompanying the smaller exponent is shifted right, with its exponent increased by one for each bit shifted, until the two exponents are equal. The two significands are then added or subtracted as appropriate, depending on the signs of the operands. All 53 bits in the significand as well as all three guard bits (G, R, and X) enter into the computation.

If a carry occurs, the sum's significand is shifted right one bit position and the exponent is increased by one. FPSCR[FPRF] is set to the class and sign of the result, except for invalid operation exceptions when FPSCR[VE] = 1.

Other registers altered:

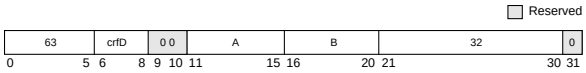
- Condition Register (CR1 field):
  Affected: FX, FEX, VX, OX (If Rc = 1)
- Floating-Point Status and Control Register:
  Affected: FPRF, FR, FI, FX, OX, UX, XX,VXSNAN, VXIS

PowerPC Architecture Level	Supervisor Level	PowerPC Optional	Form
UISA			A

fcmpo

Floating Compare Ordered (x'FC00 0040')

fcmpo **crf**D,**fr**A,**fr**B




```
if ((frA) is a NaN or (frB) is a NaN)
      then c ← 0b0001
      else if (frA)< (frB)
                then c ← 0b1000
                else if (frA)> (frB)
                          then c ← 0b0100
                          else c ← 0b0010
FPCC ← c
CR[(4 * crfD)-(4 * crfD + 3)] ← c

if ((frA) is an SNaN or (frB) is an SNaN)
      then VXSNAN ← 1
if VE = 0
      then VXVC ← 1
      else if ((frA) is a QNaN or (frB) is a QNaN)
                then VXVC ← 1
```

8

The floating-point operand in **fr**A is compared to the floating-point operand in **fr**B. The result of the compare is placed into CR field **crf**D and the FPCC.

If one of the operands is a NaN, either quiet or signaling, then CR field **crf**D and the FPCC are set to reflect unordered. If one of the operands is a signaling NaN, then VXSNAN is set, and if invalid operation is disabled (VE = 0) then VXVC is set. Otherwise, if one of the operands is a QNaN, then VXVC is set.

Other registers altered:

- Condition Register (CR field specified by operand **crf**D):
  Affected: LT, GT, EQ, UN
- Floating-Point Status and Control Register:
  Affected: FPCC, FX, VXSNAN, VXVC

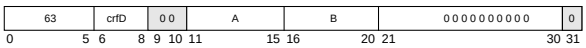
PowerPC Architecture Level	Supervisor Level	PowerPC Optional	Form
UISA			X

fcmpu

Floating Compare Unordered (x'FC00 0000')

fcmpu **crf**D,**fr**A,**fr**B

Reserved




```
if ((frA) is a NaN or (frB) is a NaN)
      then c ← 0b0001
      else if (frA) < (frB)
            then c ← 0b1000
            else if (frA) > (frB)
                  then c ← 0b0100
                  else c ← 0b0010
FPCC ← c
CR[(4 * crfD)-(4 * crfD + 3)] ← c

if ((frA) is an SNaN or (frB) is an SNaN)
      then VXSNAN ← 1
```

The floating-point operand in register **fr**A is compared to the floating-point operand in register **fr**B. The result of the compare is placed into CR field **crf**D and the FPCC.

If one of the operands is a NaN, either quiet or signaling, then CR field **crf**D and the FPCC are set to reflect unordered. If one of the operands is a signaling NaN, then VXSNAN is set.

Other registers altered:

- Condition Register (CR field specified by operand **crf**D):
  Affected: LT, GT, EQ, UN
- Floating-Point Status and Control Register:
  Affected: FPCC, FX, VXSNAN

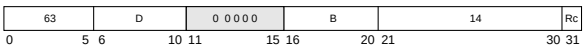
PowerPC Architecture Level	Supervisor Level	PowerPC Optional	Form
UISA			X

fctiw*x*

Floating Convert to Integer Word (x'FC00 001C')

fctiw **fr**D,**fr**B (Rc = 0)
fctiw. **fr**D,**fr**B (Rc = 1)

Reserved



The floating-point operand in register **fr**B is converted to a 32-bit signed integer, using the rounding mode specified by FPSCR[RN], and placed in bits 32–63 of **fr**D. Bits 0–31 of **fr**D are undefined.

If the operand in **fr**B are greater than $2^{31} - 1$, bits 32–63 of **fr**D are set to 0x7FFF_FFFF.

If the operand in **fr**B are less than -2^{31}, bits 32–63 of **fr**D are set to 0x8000_0000.

The conversion is described fully in Section D.4.2, “Floating-Point Convert to Integer Model.”

Except for trap-enabled invalid operation exceptions, FPSCR[FPRF] is undefined. FPSCR[FR] is set if the result is incremented when rounded. FPSCR[FI] is set if the result is inexact.

(Programmers note: A **stfiwz** instruction should be used to store the 32 bit resultant integer because bits 0–31 of **fr**D are undefined. A store double-precision instruction, e.g., **stfd**x, will store the 64 bit result but 4 superfluous bytes are stored (bits **fr**D[0-31]). This may cause wasted bus traffic.)

Other registers altered:

- Condition Register (CR1 field):
  Affected: FX, FEX, VX, OX (If Rc = 1)
- Floating-Point Status and Control Register:
  Affected: FPRF (undefined), FR, FI, FX, XX, VXSNAN, VXCVI

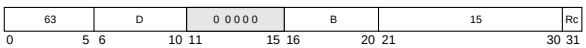
PowerPC Architecture Level	Supervisor Level	PowerPC Optional	Form
UISA			X

fctiwz*x*

Floating Convert to Integer Word with Round toward Zero (x'FC00 001E')

fctiwz **fr**D,**fr**B (Rc = 0)
fctiwz. **fr**D,**fr**B (Rc = 1)

Reserved



The floating-point operand in register **fr**B is converted to a 32-bit signed integer, using the rounding mode round toward zero, and placed in bits 32–63 of **fr**D. Bits 0–31 of **fr**D are undefined.

If the operand in **fr**B is greater than $2^{31} - 1$, bits 32–63 of **fr**D are set to 0x7FFF_FFFF.

If the operand in **fr**B is less than -2^{31}, bits 32–63 of **fr**D are set to 0x 8000_0000.

The conversion is described fully in Section D.4.2, “Floating-Point Convert to Integer Model.”

Except for trap-enabled invalid operation exceptions, FPSCR[FPRF] is undefined. FPSCR[FR] is set if the result is incremented when rounded. FPSCR[FI] is set if the result is inexact.

(Programmers note: A **stfiwx** instruction should be used to store the 32 bit resultant integer because bits 0–31 of **fr**D are undefined. A store double-precision instruction, e.g., **stfd**x, will store the 64 bit result but 4 superfluous bytes are stored (bits **fr**D[0-31]). This may cause wasted bus traffic.)

Other registers altered:

- Condition Register (CR1 field):
  Affected: FX, FEX, VX, OX (If Rc = 1)
- Floating-Point Status and Control Register:
  Affected: FPRF (undefined), FR, FI, FX, XX, VXSNAN, VXCVI

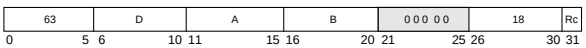
PowerPC Architecture Level	Supervisor Level	PowerPC Optional	Form
UISA			X

fdiv*x*

Floating Divide (Double-Precision),(x'FC00 0024')

fdiv **fr**D,**fr**A,**fr**B (Rc = 0)
fdiv. **fr**D,**fr**A,**fr**B (Rc = 1)

Reserved



The floating-point operand in register **fr**A is divided by the floating-point operand in register **fr**B. The remainder is not supplied as a result.

If the most-significant bit of the resultant significand is not a one, the result is normalized. The result is rounded to double-precision under control of the floating-point rounding control field RN of the FPSCR and placed into **fr**D.

Floating-point division is based on exponent subtraction and division of the significands.

FPSCR[FPRF] is set to the class and sign of the result, except for invalid operation exceptions when FPSCR[VE] = 1 and zero divide exceptions when FPSCR[ZE] = 1.

Other registers altered:

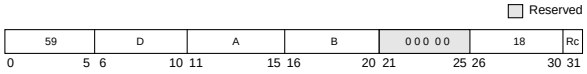
- Condition Register (CR1 field):
  Affected: FX, FEX, VX, OX (If Rc = 1)
- Floating-Point Status and Control Register:
  Affected: FPRF, FR, FI, FX, OX, UX, ZX, XX, VXSNAN, VXIDI, VXZDZ

PowerPC Architecture Level	Supervisor Level	PowerPC Optional	Form
UISA			A

fdivs*x* fdivs*x*

Floating Divide Single (x'EC00 0024')

fdivs **fr**D,**fr**A,**fr**B (Rc = 0)
fdivs. **fr**D,**fr**A,**fr**B (Rc = 1)



The floating-point operand in register **fr**A is divided by the floating-point operand in register **fr**B. The remainder is not supplied as a result.

If the most-significant bit of the resultant significand is not a one, the result is normalized. The result is rounded to single-precision under control of the floating-point rounding control field RN of the FPSCR and placed into **fr**D.

Floating-point division is based on exponent subtraction and division of the significands.

FPSCR[FPRF] is set to the class and sign of the result, except for invalid operation exceptions when FPSCR[VE] = 1 and zero divide exceptions when FPSCR[ZE] = 1.

Other registers altered:

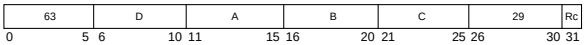
- Condition Register (CR1 field):
  Affected: FX, FEX, VX, OX (if Rc = 1)
- Floating-Point Status and Control Register:
  Affected: FPRF, FR, FI, FX, OX, UX, ZX, XX, VXSNAN, VXIDI, VXZDZ

PowerPC Architecture Level	Supervisor Level	PowerPC Optional	Form
UISA			A

fmadd*x* fmadd*x*

Floating Multiply-Add (Double-Precision),(x'FC00 003A')

fmadd **fr**D,**fr**A,**fr**C,**fr**B (Rc = 0)
fmadd. **fr**D,**fr**A,**fr**C,**fr**B (Rc = 1)



The following operation is performed:

```
frD ← ((fra) * (frC)) + (frB)
```

The floating-point operand in register **fr**A is multiplied by the floating-point operand in register **fr**C. The floating-point operand in register **fr**B is added to this intermediate result.

If the most-significant bit of the resultant significand is not a one, the result is normalized. The result is rounded to double-precision under control of the floating-point rounding control field RN of the FPSCR and placed into **fr**D.

FPSCR[FPRF] is set to the class and sign of the result, except for invalid operation exceptions when FPSCR[VE] = 1.

Other registers altered:

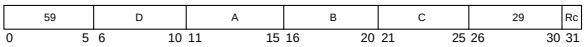
- Condition Register (CR1 field):
  Affected: FX, FEX, VX, OX (if Rc = 1)
- Floating-Point Status and Control Register:
  Affected: FPRF, FR, FI, FX, OX, UX, XX, VXSNAN, VXISI, VXIMZ

PowerPC Architecture Level	Supervisor Level	PowerPC Optional	Form
UISA			A

fmadds*x* fmadds*x*

Floating Multiply-Add Single (x'EC00 003A')

fmadds **fr**D,**fr**A,**fr**C,**fr**B (Rc = 0)
fmadds. **fr**D,**fr**A,**fr**C,**fr**B (Rc = 1)



The following operations are performed:

```
frD ← ((frA) * (frC)) + (frB)
```

The floating-point operand in register **fr**A is multiplied by the floating-point operand in register **fr**C. The floating-point operand in register **fr**B is added to this intermediate result.

If the most-significant bit of the resultant significand is not a one, the result is normalized. The result is rounded to single-precision under control of the floating-point rounding control field RN of the FPSCR and placed into **fr**D.

FPSCR[FPRF] is set to the class and sign of the result, except for invalid operation exceptions when FPSCR[VE] = 1.

Other registers altered:

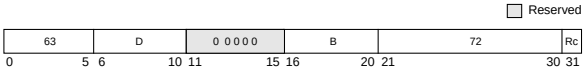
- Condition Register (CR1 field):
  Affected: FX, FEX, VX, OX (if Rc = 1)
- Floating-Point Status and Control Register:
  Affected: FPRF, FR, FI, FX, OX, UX, XX, VXSNAN, VXISI, VXIMZ

PowerPC Architecture Level	Supervisor Level	PowerPC Optional	Form
UISA			A

fmr*x* fmr*x*

Floating Move Register(Double-Precision),(x'FC00 0090')

fmr **fr**D,**fr**B (Rc = 0)
fmr. **fr**D,**fr**B (Rc = 1)



The following operation is performed:

```
frD ← (frB)
```

The content of register **fr**B is placed into **fr**D.

Other registers altered:

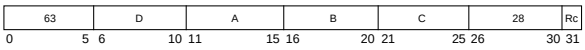
- Condition Register (CR1 field):
  Affected: FX, FEX, VX, OX (if Rc = 1)

PowerPC Architecture Level	Supervisor Level	PowerPC Optional	Form
UISA			X

fmsub*x* fmsub*x*

Floating Multiply-Subtract (Double-Precision),(x'FC00 0038')

fmsub **fr**D,**fr**A,**fr**C,**fr**B (Rc = 0)
fmsub. **fr**D,**fr**A,**fr**C,**fr**B (Rc = 1)



The following operation is performed:

```
frD ← [(frA) * (frC)] - (frB)
```

The floating-point operand in register **fr**A is multiplied by the floating-point operand in register **fr**C. The floating-point operand in register **fr**B is subtracted from this intermediate result.

If the most-significant bit of the resultant significand is not a one, the result is normalized. The result is rounded to double-precision under control of the floating-point rounding control field RN of the FPSCR and placed into **fr**D.

8

FPSCR[FPRF] is set to the class and sign of the result, except for invalid operation exceptions when FPSCR[VE] = 1.

Other registers altered:

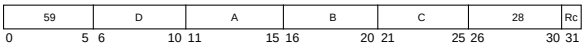
- Condition Register (CR1 field):
  Affected: FX, FEX, VX, OX (if Rc = 1)
- Floating-Point Status and Control Register:
  Affected: FPRF, FR, FI, FX, OX, UX, XX, VXSNAN, VXISI, VXIMZ

PowerPC Architecture Level	Supervisor Level	PowerPC Optional	Form
UISA			A

fmsubs*x* fmsubs*x*

Floating Multiply-Subtract Single (x'EC00 0038')

fmsubs **fr**D,**fr**A,**fr**C,**fr**B (Rc = 0)
fmsubs. **fr**D,**fr**A,**fr**C,**fr**B (Rc = 1)



The following operations are performed:

```
frD ← [(frA) * (frC)] - (frB)
```

The floating-point operand in register **fr**A is multiplied by the floating-point operand in register **fr**C. The floating-point operand in register **fr**B is subtracted from this intermediate result.

If the most-significant bit of the resultant significand is not a one, the result is normalized. The result is rounded to single-precision under control of the floating-point rounding control field RN of the FPSCR and placed into **fr**D.

8

FPSCR[FPRF] is set to the class and sign of the result, except for invalid operation exceptions when FPSCR[VE] = 1.

Other registers altered:

- Condition Register (CR1 field):
  Affected: FX, FEX, VX, OX (if Rc = 1)
- Floating-Point Status and Control Register:
  Affected: FPRF, FR, FI, FX, OX, UX, XX, VXSNAN, VXISI, VXIMZ

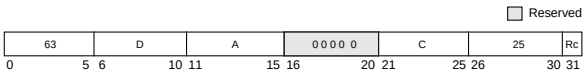
\

PowerPC Architecture Level	Supervisor Level	PowerPC Optional	Form
UISA			A

fmul*x* fmul*x*

Floating Multiply (Double-Precision),(x'FC00 0032')

fmul **fr**D,**fr**A,**fr**C (Rc = 0)
fmul. **fr**D,**fr**A,**fr**C (Rc = 1)



The following operation is performed:

```
frD ← (frA) * (frC)
```

The floating-point operand in register **fr**A is multiplied by the floating-point operand in register **fr**C.

If the most-significant bit of the resultant significand is not a one, the result is normalized. The result is rounded to double-precision under control of the floating-point rounding control field RN of the FPSCR and placed into **fr**D.

8

Floating-point multiplication is based on exponent addition and multiplication of the significands.

FPSCR[FPRF] is set to the class and sign of the result, except for invalid operation exceptions when FPSCR[VE] = 1.

Other registers altered:

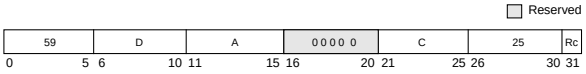
- Condition Register (CR1 field):
  Affected: FX, FEX, VX, OX (if Rc = 1)
- Floating-Point Status and Control Register:
  Affected: FPRF, FR, FI, FX, OX, UX, XX, VXSNAN, VXIMZ

PowerPC Architecture Level	Supervisor Level	PowerPC Optional	Form
UISA			A

fmuls*x* fmuls*x*

Floating Multiply Single (x'EC00 0032')

fmuls **fr**D,**fr**A,**fr**C (Rc = 0)
fmuls. **fr**D,**fr**A,**fr**C (Rc = 1)



The following operation is performed:

```
frD ← (frA) * (frC)
```

The floating-point operand in register **fr**A is multiplied by the floating-point operand in register **fr**C.

If the most-significant bit of the resultant significand is not a one, the result is normalized. The result is rounded to single-precision under control of the floating-point rounding control field RN of the FPSCR and placed into **fr**D.

8

Floating-point multiplication is based on exponent addition and multiplication of the significands.

FPSCR[FPRF] is set to the class and sign of the result, except for invalid operation exceptions when FPSCR[VE] = 1.

Other registers altered:

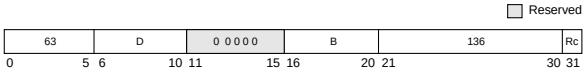
- Condition Register (CR1 field):
  Affected: FX, FEX, VX, OX (if Rc = 1)
- Floating-Point Status and Control Register:
  Affected: FPRF, FR, FI, FX, OX, UX, XX, VXSNAN, VXIMZ

PowerPC Architecture Level	Supervisor Level	PowerPC Optional	Form
UISA			A

fnabs*x* fnabs*x*

Floating Negative Absolute Value (x'FC00 0110')

fnabs **fr**D,**fr**B (Rc = 0)
fnabs. **fr**D,**fr**B (Rc = 1)



The following operation is performed:

```
frD ← 1 || frB[1-63]
```

The contents of register **fr**B with bit 0 set are placed into **fr**D.

NOTE: The **fnabs** instruction treats NaNs just like any other kind of value. That is, the sign bit of a NaN may be altered by **fnabs**. This instruction does not alter the FPSCR.

Other registers altered:

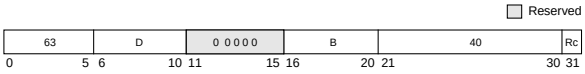
- Condition Register (CR1 field):
  Affected: FX, FEX, VX, OX (if Rc = 1)

PowerPC Architecture Level	Supervisor Level	PowerPC Optional	Form
UISA			X

fneg*x* fneg*x*

Floating Negate (x'FC00 0050')

fneg **fr**D,**fr**B (Rc = 0)
fneg. **fr**D,**fr**B (Rc = 1)



The following operation is performed:

```
frD ← ¬ frB[0] || frB[1-63]
```

The contents of register **fr**B with bit 0 inverted are placed into **fr**D.

NOTE: The **fneg** instruction treats NaNs just like any other kind of value. That is, the sign bit of a NaN may be altered by **fneg**. This instruction does not alter the FPSCR.

Other registers altered:

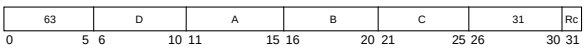
- Condition Register (CR1 field):
  Affected: FX, FEX, VX, OX (if Rc = 1)

PowerPC Architecture Level	Supervisor Level	PowerPC Optional	Form
UISA			X

fnmadd*x* fnmadd*x*

Floating Negative Multiply-Add (Double-Precision),(x'FC00 003E')

fnmadd **fr**D,**fr**A,**fr**C,**fr**B (Rc = 0)
fnmadd. **fr**D,**fr**A,**fr**C,**fr**B (Rc = 1)



The following operations are performed:

```
frD ← – ( [(frA) * (frC)] + (frB) )
```

The floating-point operand in register **fr**A is multiplied by the floating-point operand in register **fr**C. The floating-point operand in register **fr**B is added to this intermediate result. If the most-significant bit of the resultant significand is not a one, the result is normalized. The result is rounded to double-precision under control of the floating-point rounding control field RN of the FPSCR, then negated and placed into **fr**D.

This instruction produces the same result as would be obtained by using the Floating Multiply-Add (**fmadd***x*) instruction and then negating the result, with the following exceptions:

- QNaNs propagate with no effect on their sign bit.
- QNaNs that are generated as the result of a disabled invalid operation exception have a sign bit of zero.
- SNaNs that are converted to QNaNs as the result of a disabled invalid operation exception retain the sign bit of the SNaN.

FPSCR[FPRF] is set to the class and sign of the result, except for invalid operation exceptions when FPSCR[VE] = 1.

Other registers altered:

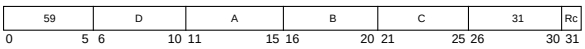
- Condition Register (CR1 field):
  Affected: FX, FEX, VX, OX (if Rc = 1)
- Floating-Point Status and Control Register:
  Affected: FPRF, FR, FI, FX, OX, UX, XX, VXSNAN, VXISI, VXIMZ

PowerPC Architecture Level	Supervisor Level	PowerPC Optional	Form
UISA			A

fnmadds*x* fnmadds*x*

Floating Negative Multiply-Add Single (x'EC00 003E')

fnmadds **fr**D,**fr**A,**fr**C,**fr**B (Rc = 0)
fnmadds. **fr**D,**fr**A,**fr**C,**fr**B (Rc = 1)



The following operations are performed:

```
frD ← – ( [(frA) * (frC)] + (frB) )
```

The floating-point operand in register **fr**A is multiplied by the floating-point operand in register **fr**C. The floating-point operand in register **fr**B is added to this intermediate result. If the most-significant bit of the resultant significand is not a one, the result is normalized. The result is rounded to single-precision under control of the floating-point rounding control field RN of the FPSCR, then negated and placed into **fr**D.

This instruction produces the same result as would be obtained by using the Floating Multiply-Add Single (**fmadds***x*) instruction and then negating the result, with the following exceptions:

- QNaNs propagate with no effect on their sign bit.
- QNaNs that are generated as the result of a disabled invalid operation exception have a sign bit of zero.
- SNaNs that are converted to QNaNs as the result of a disabled invalid operation exception retain the sign bit of the SNaN.

FPSCR[FPRF] is set to the class and sign of the result, except for invalid operation exceptions when FPSCR[VE] = 1.

Other registers altered:

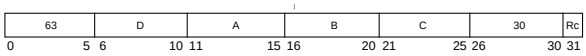
- Condition Register (CR1 field):
  Affected: FX, FEX, VX, OX (if Rc = 1)
- Floating-Point Status and Control Register:
  Affected: FPRF, FR, FI, FX, OX, UX, XX, VXSNAN, VXISI, VXIMZ

PowerPC Architecture Level	Supervisor Level	PowerPC Optional	Form
UISA			A

fnmsub*x* fnmsub*x*

Floating Negative Multiply-Subtract (Double-Precision),(x'FC00 003C')

fnmsub **fr**D,**fr**A,**fr**C,**fr**B (Rc = 0)
fnmsub. **fr**D,**fr**A,**fr**C,**fr**B (Rc = 1)



The following operations are performed:

```
frD ← – ( [(frA) * (frC)] – (frB) )
```

The floating-point operand in register **fr**A is multiplied by the floating-point operand in register **fr**C. The floating-point operand in register **fr**B is subtracted from this intermediate result.

If the most-significant bit of the resultant significand is not one, the result is normalized. The result is rounded to double-precision under control of the floating-point rounding control field RN of the FPSCR, then negated and placed into **fr**D.

This instruction produces the same result obtained by negating the result of a Floating Multiply-Subtract (**fmsub***x*) instruction with the following exceptions:

- QNaNs propagate with no effect on their sign bit.
- QNaNs that are generated as the result of a disabled invalid operation exception have a sign bit of zero.
- SNaNs that are converted to QNaNs as the result of a disabled invalid operation exception retain the sign bit of the SNaN.

FPSCR[FPRF] is set to the class and sign of the result, except for invalid operation exceptions when FPSCR[VE] = 1.

Other registers altered:

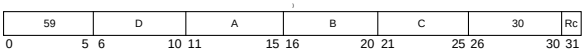
- Condition Register (CR1 field)
  Affected: FX, FEX, VX, OX (if Rc = 1)
- Floating-Point Status and Control Register:
  Affected: FPRF, FR, FI, FX, OX, UX, XX, VXSNAN, VXISI, VXIMZ

PowerPC Architecture Level	Supervisor Level	PowerPC Optional	Form
UISA			A

fnmsubs*x* fnmsubs*x*

Floating Negative Multiply-Subtract Single (x'EC00 003C')

fnmsubs **fr**D,**fr**A,**fr**C,**fr**B (Rc = 0)
fnmsubs. **fr**D,**fr**A,**fr**C,**fr**B (Rc = 1)



The following operations are performed:

```
frD ← – ( [(frA) * (frC)] – (frB) )
```

The floating-point operand in register **fr**A is multiplied by the floating-point operand in register **fr**C. The floating-point operand in register **fr**B is subtracted from this intermediate result.

If the most-significant bit of the resultant significand is not one, the result is normalized. The result is rounded to single-precision under control of the floating-point rounding control field RN of the FPSCR, then negated and placed into **fr**D.

This instruction produces the same result obtained by negating the result of a Floating Multiply-Subtract Single (**fmsubs***x*) instruction with the following exceptions:

- QNaNs propagate with no effect on their sign bit.
- QNaNs that are generated as the result of a disabled invalid operation exception have a sign bit of zero.
- SNaNs that are converted to QNaNs as the result of a disabled invalid operation exception retain the sign bit of the SNaN.

FPSCR[FPRF] is set to the class and sign of the result, except for invalid operation exceptions when FPSCR[VE] = 1.

Other registers altered:

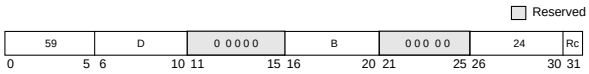
- Condition Register (CR1 field)
  Affected: FX, FEX, VX, OX (if Rc = 1)
- Floating-Point Status and Control Register:
  Affected: FPRF, FR, FI, FX, OX, UX, XX, VXSNAN, VXISI, VXIMZ

PowerPC Architecture Level	Supervisor Level	PowerPC Optional	Form
UISA			A

fres*x* fres*x*

Floating Reciprocal Estimate Single (x'EC00 0030')

fres **fr**D,**fr**B (Rc = 0)
fres. **fr**D,**fr**B (Rc = 1)



The following operation is performed:

```
frD ← estimate[1/(frB)]
```

A single-precision estimate of the reciprocal of the floating-point operand in register **fr**B is placed into register **fr**D. The estimate placed into register **fr**D is correct to a precision of one part in 4096 of the reciprocal of **fr**B. That is,

$$ABS\left(\frac{estimate-\left(\frac{1}{x}\right)}{\left(\frac{1}{x}\right)}\right) \leq \frac{1}{(4096)}$$

where x is the initial value in **fr**B.

NOTE: The value placed into register **fr**D may vary between implementations, and between different executions on the same implementation.

Operation with various special values of the operand is summarized below:

Operand	Result	Exception
–	–0	None
–0	– *	ZX
+0	+ *	ZX
+	+0	None
SNaN	QNaN**	VXSNAN
QNaN	QNaN	None

Notes: * No result if FPSCR[ZE] = 1
** No result if FPSCR[VE] = 1

FPSCR[FPRF] is set to the class and sign of the result, except for invalid operation exceptions when FPSCR[VE] = 1 and zero divide exceptions when FPSCR[ZE] = 1.

NOTE: The PowerPC architecture makes no provision for a double-precision version of the **fres***x* instruction. This is because graphics applications are expected to need only the single-precision version, and no other important performance-critical applications are expected to require a double-precision version of the **fres***x* instruction.

NOTE: This instruction is optional in the PowerPC architecture.

Other registers altered:

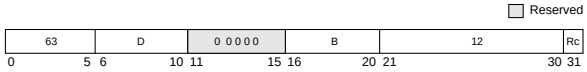
- Condition Register (CR1 field):
  Affected: FX, FEX, VX, OX (if Rc = 1)
- Floating-Point Status and Control Register:
  Affected: FPRF, FR (undefined), FI (undefined), FX, OX, UX, ZX, VXSNAN

PowerPC Architecture Level	Supervisor Level	PowerPC Optional	Form
UISA		√	A

frsp*x* frsp*x*

Floating Round to Single (x'FC00 0018')

frsp	**fr**D,**fr**B	(Rc = 0)
frsp.	**fr**D,**fr**B	(Rc = 1)



The following operation is performed:

```
frD ← Round_single( frB )
```

If it is already in single-precision range, the floating-point operand in register **fr**B is placed into **fr**D. Otherwise, the floating-point operand in register **fr**B is rounded to single-precision using the rounding mode specified by FPSCR[RN] and placed into **fr**D.

The rounding is described fully in Section D.4.1, "Floating-Point Round to Single-Precision Model."

FPSCR[FPRF] is set to the class and sign of the result, except for invalid operation exceptions when FPSCR[VE] = 1.

Other registers altered:

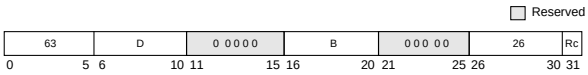
- Condition Register (CR1 field):
  Affected: FX, FEX, VX, OX (if Rc = 1)
- Floating-Point Status and Control Register:
  Affected: FPRF, FR, FI, FX, OX, UX, XX, VXSNAN

PowerPC Architecture Level	Supervisor Level	PowerPC Optional	Form
UISA			X

frsqrte*x* frsqrte*x*

Floating Reciprocal Square Root Estimate (x'FC00 0034')

frsqrte	**fr**D,**fr**B	(Rc = 0)
frsqrte.	**fr**D,**fr**B	(Rc = 1)



A double-precision estimate of the reciprocal of the square root of the floating-point operand in register **fr**B is placed into register **fr**D. The estimate placed into register **fr**D is correct to a precision of one part in 4096 of the reciprocal of the square root of **fr**B. That is,

$$\mathbf{fr}D \leftarrow \quad ABS\left\{\frac{estimate-\left(\frac{1}{\sqrt{x}}\right)}{\left(\frac{1}{\sqrt{x}}\right)}\right\} \leq \frac{1}{4096}$$

where x is the initial value in **fr**B.

NOTE: The value placed into register **fr**D may vary between implementations, and between different executions on the same implementation.

Operation with various special values of the operand is summarized below:

Operand	Result	Exception
–	QNaN**	VXSQRT
<0	QNaN**	VXSQRT
–0	– *	ZX
+0	+ *	ZX
+	+0	None
SNaN	QNaN**	VXSNAN
QNaN	QNaN	None

Notes: * No result if FPSCR[ZE] = 1
** No result if FPSCR[VE] = 1

FPSCR[FPRF] is set to the class and sign of the result, except for invalid operation exceptions when FPSCR[VE] = 1 and zero divide exceptions when FPSCR[ZE] = 1.

NOTE: No single-precision version of the **frsqrte** instruction is provided; however, both **fr**B and **fr**D are representable in single-precision format.

NOTE: This instruction is optional in the PowerPC architecture.

Other registers altered:

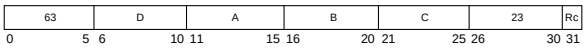
- Condition Register (CR1 field):
  Affected: FX, FEX, VX, OX (if Rc = 1)
- Floating-Point Status and Control Register:
  Affected: FPRF, FR (undefined), FI (undefined), FX, ZX, VXSNAN, VXSQRT

PowerPC Architecture Level	Supervisor Level	PowerPC Optional	Form
UISA		√	A

fsel*x* fsel*x*

Floating Select (x'FC00 002E')

fsel	**fr**D,**fr**A,**fr**C,**fr**B	(Rc = 0)
fsel.	**fr**D,**fr**A,**fr**C,**fr**B	(Rc = 1)




```
if (frA) ≥ 0.0
     then frD ← (frC)
     else frD ← (frB)
```

The floating-point operand in register **fr**A is compared to the value zero. If the operand is greater than or equal to zero, register **fr**D is set to the contents of register **fr**C. If the operand is less than zero or is a NaN, register **fr**D is set to the contents of register **fr**B. The comparison ignores the sign of zero (that is, regards +0 as equal to –0).

Care must be taken in using **fsel** if IEEE compatibility is required, or if the values being tested can be NaNs or infinities.

For examples of uses of this instruction, see Section D.3, "Floating-Point Conversions," and Section D.5, "Floating-Point Selection."

NOTE: This instruction is optional in the PowerPC architecture.

Other registers altered:

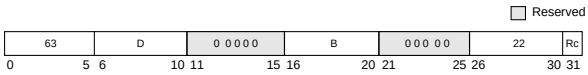
- Condition Register (CR1 field):
  Affected: FX, FEX, VX, OX (if Rc = 1)

PowerPC Architecture Level	Supervisor Level	PowerPC Optional	Form
UISA		√	A

fsqrt*x* fsqrt*x*

Floating Square Root(Double-Precision),(x'FC00 002C')

fsqrt **fr**D,**fr**B (Rc = 0)
fsqrt. **fr**D,**fr**B (Rc = 1)



The following operation is performed:

```
frD ← Square_root( frB )
```

The square root of the floating-point operand in register **fr**B is placed into register **fr**D.

If the most-significant bit of the resultant significand is not one, the result is normalized. The result is rounded to double-precision under control of the floating-point rounding control field RN of the FPSCR and placed into **fr**D.

Operation with various special values of the operand is summarized below:

Operand	Result	Exception
–	QNaN*	VXSQRT
<0	QNaN*	VXSQRT
–0	–0	None
+	+	None
SNaN	QNaN*	VXSNAN
QNaN	QNaN	None

Notes: * No result if FPSCR[VE] = 1

FPSCR[FPRF] is set to the class and sign of the result, except for invalid operation exceptions when FPSCR[VE] = 1.

NOTE: This instruction is optional in the PowerPC architecture.

Other registers altered:

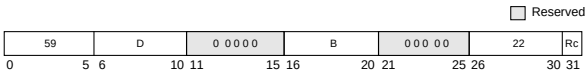
- Condition Register (CR1 field):
  Affected: FX, FEX, VX, OX (if Rc = 1)
- Floating-Point Status and Control Register:
  Affected: FPRF, FR, FI, FX, ZX, VXSNAN, VXSQR

PowerPC Architecture Level	Supervisor Level	PowerPC Optional	Form
UISA		√	A

fsqrts*x* fsqrts*x*

Floating Square Root(Single-Precision),(x'EC00 002C')

fsqrts **fr**D,**fr**B (Rc = 0)
fsqrts. **fr**D,**fr**B (Rc = 1)



The following operation is performed:

```
frD ← Square_root( frB )
```

The square root of the floating-point operand in register **fr**B is placed into register **fr**D.

If the most-significant bit of the resultant significand is not one, the result is normalized. The result is rounded to single-precision under control of the floating-point rounding control field RN of the FPSCR and placed into **fr**D.

Operation with various special values of the operand is summarized below:

Operand	Result	Exception
–	QNaN*	VXSQRT
<0	QNaN*	VXSQRT
–0	–0	None
+	+	None
SNaN	QNaN*	VXSNAN
QNaN	QNaN	None

Notes: * No result if FPSCR[VE] = 1

FPSCR[FPRF] is set to the class and sign of the result, except for invalid operation exceptions when FPSCR[VE] = 1.

NOTE: This instruction is optional in the PowerPC architecture.

Other registers altered:

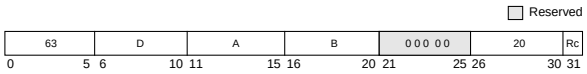
- Condition Register (CR1 field):
  Affected: FX, FEX, VX, OX (if Rc = 1)
- Floating-Point Status and Control Register:
  Affected: FPRF, FR, FI, FX, ZX, VXSNAN, VXSQRT

PowerPC Architecture Level	Supervisor Level	PowerPC Optional	Form
UISA		√	A

fsub*x* fsub*x*

Floating Subtract (Double-Precision),(x'FC00 0028')

fsub **fr**D,**fr**A,**fr**B (Rc = 0)
fsub. **fr**D,**fr**A,**fr**B (Rc = 1)



The following operation is performed:

```
frD ← (frA) – (frB)
```

The floating-point operand in register **fr**B is subtracted from the floating-point operand in register **fr**A. If the most-significant bit of the resultant significand is not a one, the result is normalized. The result is rounded to double-precision under control of the floating-point rounding control field RN of the FPSCR and placed into **fr**D.

The execution of the **fsub** instruction is identical to that of **fadd**, except that the contents of **fr**B participate in the operation with its sign bit (bit 0) inverted.

FPSCR[FPRF] is set to the class and sign of the result, except for invalid operation exceptions when FPSCR[VE] = 1.

Other registers altered:

- Condition Register (CR1 field):
  Affected: FX, FEX, VX, OX (if Rc = 1)
- Floating-Point Status and Control Register:
  Affected: FPRF, FR, FI, FX, OX, UX, XX, VXSNAN, VXISI

PowerPC Architecture Level	Supervisor Level	PowerPC Optional	Form
UISA			A

fsubs*x* fsubs*x*

Floating Subtract Single (x'EC00 0028')

fsubs **fr**D,**fr**A,**fr**B (Rc = 0)
fsubs. **fr**D,**fr**A,**fr**B (Rc = 1)

The following operation is performed:

```
frD ← (frA) – (frB)
```

The floating-point operand in register **fr**B is subtracted from the floating-point operand in register **fr**A. If the most-significant bit of the resultant significand is not a one, the result is normalized. The result is rounded to single-precision under control of the floating-point rounding control field RN of the FPSCR and placed into **fr**D.

The execution of the **fsubs** instruction is identical to that of **fadds**, except that the contents of **fr**B participate in the operation with its sign bit (bit 0) inverted.

FPSCR[FPRF] is set to the class and sign of the result, except for invalid operation exceptions when FPSCR[VE] = 1.

Other registers altered:

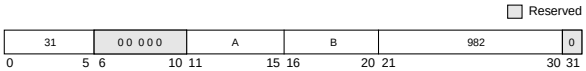
- Condition Register (CR1 field):
  Affected: FX, FEX, VX, OX (if Rc = 1)
- Floating-Point Status and Control Register:
  Affected: FPRF, FR, FI, FX, OX, UX, XX, VXSNAN, VXISI

PowerPC Architecture Level	Supervisor Level	PowerPC Optional	Form
UISA			A

icbi

Instruction Cache Block Invalidate (x'7C00 07AC')

icbi **r**A,**r**B



EA is the sum (**r**A|0) + (**r**B).

If the block containing the byte addressed by EA is in coherency-required mode, and a block containing the byte addressed by EA is in the instruction cache of any processor, the block is made invalid in all such instruction caches, so that subsequent references cause the block to be refetched.

If the block containing the byte addressed by EA is in coherency-not-required mode, and a block containing the byte addressed by EA is in the instruction cache of this processor, the block is made invalid in that instruction cache, so that subsequent references cause the block to be refetched.

The function of this instruction is independent of the write-through, write-back, and caching-inhibited/allowed modes of the block containing the byte addressed by EA.

This instruction is treated as a load from the addressed byte with respect to address translation and memory protection. It may also be treated as a load for referenced and changed bit recording except that referenced and changed bit recording may not occur. Implementations with a combined data and instruction cache treat the **icbi** instruction as a no-op, except that they may invalidate the target block in the instruction caches of other processors if the block is in coherency-required mode.

The **icbi** instruction invalidates the block at EA (**r**A|0 + **r**B). If the processor is a multiprocessor implementation (for example, the 601, 604, or 620) and the block is marked coherency-required, the processor will send an address-only broadcast to other processors causing those processors to invalidate the block from their instruction caches.

For faster processing, many implementations will not compare the entire EA (**r**A|0 + **r**B) with the tag in the instruction cache. Instead, they will use the bits in the EA to locate the set that the block is in, and invalidate all blocks in that set.

Other registers altered:

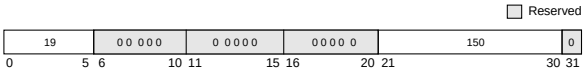
- None

PowerPC Architecture Level	Supervisor Level	PowerPC Optional	Form
VEA			X

isync

Instruction Synchronize (x'4C00 012C')

isync



The **isync** instruction provides an ordering function for the effects of all instructions executed by a processor. Executing an **isync** instruction ensures that all instructions preceding the **isync** instruction have completed before the **isync** instruction completes, except that memory accesses caused by those instructions need not have been performed with respect to other processors and mechanisms. It also ensures that no subsequent instructions are initiated by the processor until after the **isync** instruction completes. Finally, it causes the processor to discard any prefetched instructions, with the effect that subsequent instructions will be fetched and executed in the context established by the instructions preceding the isync instruction. The **isync** instruction has no effect on the other processors or on their caches.

This instruction is context synchronizing.

Context synchronization is necessary after certain code sequences that perform complex operations within the processor. These code sequences are usually operating system tasks that involve memory management. For example, if an instruction A changes the memory translation rules in the memory management unit (MMU), the **isync** instruction should be executed so that the instructions following instruction A will be discarded from the pipeline and refetched according to the new translation rules.

NOTE: All exceptions and **rfi** and **sc** instructions are also context synchronizing.

Other registers altered:

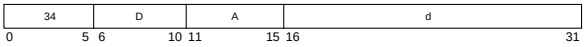
- None

PowerPC Architecture Level	Supervisor Level	PowerPC Optional	Form
VEA			XL

lbz

Load Byte and Zero (x'8800 0000')

lbz **r**D,d(**r**A)




```
if rA = 0
      then b ← 0
      else b ← (rA)
EA ← b + EXTS(d)
rD ← (24)0 || MEM(EA, 1)
```

EA is the sum (**r**A|0) + d. The byte in memory addressed by EA is loaded into the low-order eight bits of **r**D. The remaining bits in **r**D are cleared.

Other registers altered:

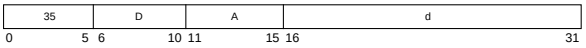
- None

PowerPC Architecture Level	Supervisor Level	PowerPC Optional	Form
UISA			D

lbzu

Load Byte and Zero with Update (x'8C00 0000')

lbzu **r**D,d(**r**A)




```
EA ← (rA) + EXTS(d)
rD← (24)0 || MEM(EA, 1)
rA← EA
```

EA is the sum (**r**A) + d. The byte in memory addressed by EA is loaded into the low-order eight bits of **r**D. The remaining bits in **r**D are cleared.

EA is placed into **r**A.

If **r**A = 0, or **r**A = **r**D, the instruction form is invalid.

Other registers altered:

- None

PowerPC Architecture Level	Supervisor Level	PowerPC Optional	Form
UISA			D

lbzux lbzux

Load Byte and Zero with Update Indexed (x'7C00 00EE')

lbzux **r**D,**r**A,**r**B


```
EA ← (rA) + (rB)
rD ← (24)0 || MEM(EA, 1)
rA ← EA
```

EA is the sum (**r**A) + (**r**B). The byte in memory addressed by EA is loaded into the low-order eight bits of **r**D. The remaining bits in **r**D are cleared.

EA is placed into **r**A.

If **r**A = 0, or **r**A = **r**D, the instruction form is invalid.

Other registers altered:

- None

PowerPC Architecture Level	Supervisor Level	PowerPC Optional	Form
UISA			X

lbzx lbzx

Load Byte and Zero Indexed (x'7C00 00AE')

lbzx **r**D,**r**A,**r**B


```
if rA = 0
     then b ← 0
     else b ← (rA)
EA ← b + (rB)
rD ← (24)0 || MEM(EA, 1)
```

EA is the sum (**r**A|0) + (**r**B). The byte in memory addressed by EA is loaded into the low-order eight bits of **r**D. The remaining bits in **r**D are cleared.

Other registers altered:

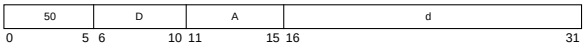
None

PowerPC Architecture Level	Supervisor Level	PowerPC Optional	Form
UISA			X

lfd lfd

Load Floating-Point Double (x'C800 0000')

lfd **fr**D,d(**r**A)




```
if rA = 0
     then b ← 0
     else b ← (rA)
EA ← b + EXTS(d)
frD ← MEM(EA, 8)
```

EA is the sum (**r**A|0) + d.

The double word in memory addressed by EA is placed into **fr**D.

Other registers altered:

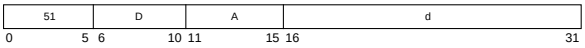
- None

PowerPC Architecture Level	Supervisor Level	PowerPC Optional	Form
UISA			D

lfdu lfdu

Load Floating-Point Double with Update (x'CC00 0000')

lfdu **fr**D,d(**r**A)




```
EA ← (rA) + EXTS(d)
frD ← MEM(EA, 8)
rA ← EA
```

EA is the sum (**r**A) + d.

The double word in memory addressed by EA is placed into **fr**D.

EA is placed into **r**A.

If **r**A = 0, the instruction form is invalid.

Other registers altered:

- None

PowerPC Architecture Level	Supervisor Level	PowerPC Optional	Form
UISA			D

lfdux lfdux

Load Floating-Point Double with Update Indexed (x'7C00 04EE')

lfdux **fr**D,**r**A,**r**B


```
EA ← (rA) + (rB)
frD ←MEM(EA, 8)
rA ← EA
```

EA is the sum (**r**A) + (**r**B).

The double word in memory addressed by EA is placed into **fr**D.

EA is placed into **r**A.

If **r**A = 0, the instruction form is invalid.

Other registers altered:

- None

PowerPC Architecture Level	Supervisor Level	PowerPC Optional	Form
UISA			X

lfdx lfdx

Load Floating-Point Double Indexed (x'7C00 04AE')

lfdx **fr**D,**r**A,**r**B


```
if rA = 0
     then b ← 0
     else b ← (rA)
EA ← b + (rB)
frD ← MEM(EA, 8)
```

EA is the sum (**r**A|0) + (**r**B).

The double word in memory addressed by EA is placed into **fr**D.

Other registers altered:

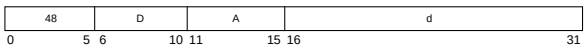
- None

PowerPC Architecture Level	Supervisor Level	PowerPC Optional	Form
UISA			X

lfs lfs

Load Floating-Point Single (x'C000 0000')

lfs **fr**D,d(**r**A)




```
if rA = 0
     then b ← 0
     else b ← (rA)
EA ← b + EXTS(d)
frD ← DOUBLE(MEM(EA, 4))
```

EA is the sum (**r**A) + d.

The word in memory addressed by EA is interpreted as a floating-point single-precision operand. This word is converted to floating-point double-precision and placed into **fr**D. (see Appendix D.6,"Floating-Point Load Instructions").

Other registers altered:

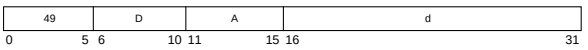
- None

PowerPC Architecture Level	Supervisor Level	PowerPC Optional	Form
UISA			D

lfsu lfsu

Load Floating-Point Single with Update (x'C400 0000')

lfsu **fr**D,d(**r**A)




```
EA ← (rA) + EXTS(d)
frD ← DOUBLE(MEM(EA, 4))
rA ← EA
```

EA is the sum (**r**A) + d.

The word in memory addressed by EA is interpreted as a floating-point single-precision operand. This word is converted to floating-point double-precision and placed into **fr**D. (see Appendix D.6,"Floating-Point Load Instructions").

EA is placed into **r**A.

If **r**A = 0, the instruction form is invalid.

Other registers altered:

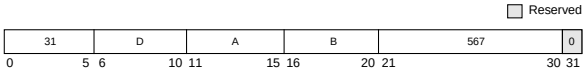
- None

PowerPC Architecture Level	Supervisor Level	PowerPC Optional	Form
UISA			D

lfsux lfsux

Load Floating-Point Single with Update Indexed (x'7C00 046E')

lfsux **fr**D,**r**A,**r**B




```
EA ← (rA) + (rB)
frD ← DOUBLE(MEM(EA, 4))
rA ← EA
```

EA is the sum (**r**A) + d.

The word in memory addressed by EA is interpreted as a floating-point single-precision operand. This word is converted to floating-point double-precision and placed into **fr**D. (see Appendix D.6,"Floating-Point Load Instructions").

EA is placed into **r**A.

If **r**A = 0, the instruction form is invalid.

Other registers altered:

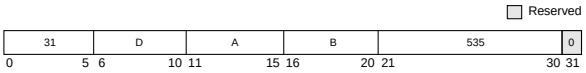
- None

PowerPC Architecture Level	Supervisor Level	PowerPC Optional	Form
UISA			X

lfsx lfsx

Load Floating-Point Single Indexed (x'7C00 042E')

lfsx **fr**D,**r**A,**r**B




```
if rA = 0
      then b ← 0
      else b ← (rA)
EA ← b + (rB)
frD ← DOUBLE(MEM(EA, 4))
```

EA is the sum (**r**A|0) + (rB).

The word in memory addressed by EA is interpreted as a floating-point single-precision operand. This word is converted to floating-point double-precision and placed into **fr**D. (see Appendix D.6,"Floating-Point Load Instructions").

Other registers altered:

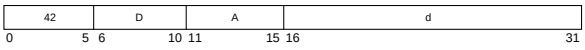
- None

PowerPC Architecture Level	Supervisor Level	PowerPC Optional	Form
UISA			X

lha lha

Load Half Word Algebraic (x'A800 0000')

lha **r**D,d(**r**A)




```
if rA = 0
      then b ← 0
      else b ← (rA)
EA ← b + EXTS(d)
rD ← EXTS(MEM(EA, 2))
```

EA is the sum (**r**A|0) + d. The half word in memory addressed by EA is loaded into the low-order 16 bits of **r**D. The remaining bits in **r**D are filled with a copy of the most-significant bit of the loaded half word.

Other registers altered:

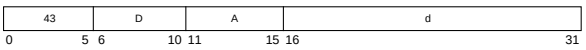
- None

PowerPC Architecture Level	Supervisor Level	PowerPC Optional	Form
UISA			D

lhau lhau

Load Half Word Algebraic with Update (x'AC00 0000')

lhau **r**D,d(**r**A)




```
EA ← (rA) + EXTS(d)
rD ← EXTS(MEM(EA, 2))
rA ← EA
```

EA is the sum (**r**A) + d. The half word in memory addressed by EA is loaded into the low-order 16 bits of **r**D. The remaining bits in **r**D are filled with a copy of the most-significant bit of the loaded half word.

EA is placed into **r**A.

If **r**A = 0 or **r**A = **r**D, the instruction form is invalid.

Other registers altered:

- None

PowerPC Architecture Level	Supervisor Level	PowerPC Optional	Form
UISA			D

lhaux lhaux

Load Half Word Algebraic with Update Indexed (x'7C00 02EE')

lhaux **r**D,**r**A,**r**B


```
EA ← (rA) + (rB)
rD ← EXTS(MEM(EA, 2))
rA ← EA
```

EA is the sum (**r**A) + (**r**B). The half word in memory addressed by EA is loaded into the low-order 16 bits of **r**D. The remaining bits in **r**D are filled with a copy of the most-significant bit of the loaded half word.

EA is placed into **r**A.

If **r**A = 0 or **r**A = **r**D, the instruction form is invalid.

Other registers altered:

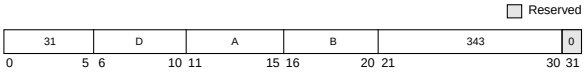
- None

PowerPC Architecture Level	Supervisor Level	PowerPC Optional	Form
UISA			X

lhax lhax

Load Half Word Algebraic Indexed (x'7C00 02AE')

lhax **r**D,**r**A,**r**B




```
if rA = 0
     then b ← 0
     else b ← (rA)
EA ← b + (rB)
rD ← EXTS(MEM(EA, 2))
```

EA is the sum (**r**A|0) + (**r**B). The half word in memory addressed by EA is loaded into the low-order 16 bits of **r**D. The remaining bits in **r**D are filled with a copy of the most-significant bit of the loaded half word.

Other registers altered:

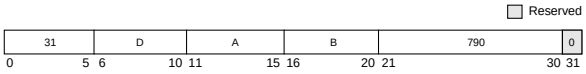
- None

PowerPC Architecture Level	Supervisor Level	PowerPC Optional	Form
UISA			X

lhbrx lhbrx

Load Half Word Byte-Reverse Indexed (x'7C00 062C')

lhbrx **r**D,**r**A,**r**B




```
if rA = 0
     then b ← 0
     else b ← (rA)
EA ← b + (rB)
rD ← (16)0 || MEM(EA + 1, 1) || MEM(EA, 1)
```

EA is the sum (**r**A|0) + (**r**B). Bits 0–7 of the half word in memory addressed by EA are loaded into the low-order eight bits of **r**D. Bits 8–15 of the half word in memory addressed by EA are loaded into the subsequent low-order eight bits of **r**D. The remaining bits in **r**D are cleared.

The PowerPC architecture cautions programmers that some implementations of the architecture may run the **lhbrx** instructions with greater latency than other types of load instructions.

Other registers altered:

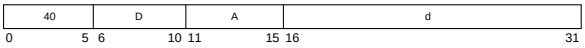
- None

PowerPC Architecture Level	Supervisor Level	PowerPC Optional	Form
UISA			X

lhz lhz

Load Half Word and Zero (x'A000 0000')

lhz **r**D,d(**r**A)




```
if rA = 0
     then b ← 0
     else b ← (rA)
EA ← b + EXTS(d)
rD ← (16)0 || MEM(EA, 2)
```

EA is the sum (**r**A|0) + d. The half word in memory addressed by EA is loaded into the low-order 16 bits of **r**D. The remaining bits in **r**D are cleared.

Other registers altered:

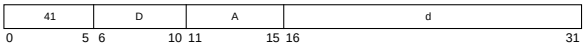
- None

PowerPC Architecture Level	Supervisor Level	PowerPC Optional	Form
UISA			D

lhzu lhzu

Load Half Word and Zero with Update (x'A400 0000')

lhzu **r**D,d(**r**A)




```
EA ← (rA) + EXTS(d)
rD ← (16)0 || MEM(EA, 2)
rA ← EA
```

EA is the sum (**r**A) + d. The half word in memory addressed by EA is loaded into the low-order 16 bits of **r**D. The remaining bits in **r**D are cleared.

EA is placed into **r**A.

If **r**A = 0 or **r**A = **r**D, the instruction form is invalid.

Other registers altered:

- None

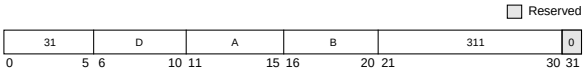
8

PowerPC Architecture Level	Supervisor Level	PowerPC Optional	Form
UISA			D

lhzux lhzux

Load Half Word and Zero with Update Indexed (x'7C00 026E')

lhzux **r**D,**r**A,**r**B




```
EA ← (rA) + (rB)
rD ← (16)0 || MEM(EA, 2)
rA ← EA
```

EA is the sum (**r**A) + (**r**B). The half word in memory addressed by EA is loaded into the low-order 16 bits of **r**D. The remaining bits in **r**D are cleared.

EA is placed into **r**A.

If **r**A = 0 or **r**A = **r**D, the instruction form is invalid.

Other registers altered:

- None

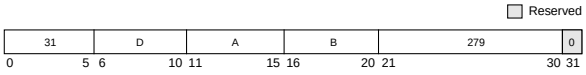
8

PowerPC Architecture Level	Supervisor Level	PowerPC Optional	Form
UISA			X

lhzx lhzx

Load Half Word and Zero Indexed (x'7C00 022E')

lhzx **r**D,**r**A,**r**B




```
if rA = 0
     then b ← 0
     else b ← (rA)
EA ← b + (rB)
rD ← (16)0 || MEM(EA, 2)
```

EA is the sum (**r**A|0) + (**r**B). The half word in memory addressed by EA is loaded into the low-order 16 bits of **r**D. The remaining bits in **r**D are cleared.

Other registers altered:

- None

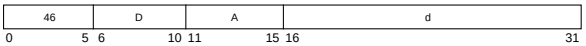
8

PowerPC Architecture Level	Supervisor Level	PowerPC Optional	Form
UISA			X

lmw lmw

Load Multiple Word (x'B800 0000')

lmw **r**D,d(**r**A)




```
if rA = 0
     then b ← 0
     else b ← (rA)
EA ← b + EXTS(d)
r ← rD
do while r ≤ 31
     GPR(r) ← MEM(EA, 4)
     r ← r + 1
     EA ← EA + 4
```

EA is the sum (**r**A|0) + d.

n = (32 – **r**D).

n consecutive words starting at EA are loaded into GPRs **r**D through **r31**.

EA must be a multiple of four. If it is not, either the system alignment exception handler is invoked or the results are boundedly undefined. For additional information about alignment and DSI exceptions, see Section 6.4.3, “DSI Exception (0x00300).”

If **r**A is in the range of registers specified to be loaded, including the case in which **r**A = 0, the instruction form is invalid.

NOTE: In some implementations, this instruction is likely to have a greater latency and take longer to execute, perhaps much longer, than a sequence of individual load or store instructions that produce the same results.

Other registers altered:

- None

8

PowerPC Architecture Level	Supervisor Level	PowerPC Optional	Form
UISA			D

lswi lswi

Load String Word Immediate (x'7C00 04AA')

lswi **r**D,**r**A,NB


```
if rA = 0
     then EA ← 0
     else EA ← (rA)
if NB = 0
     then n ← 32
     else n ← NB
r ← rD – 1
i ← 0
do while n > 0
          if i = 0
               then r ← r + 1 (mod 32)
                    GPR(r) ← (32)0
          GPR(r)[i,i + 7] ← MEM(EA, 1)
          i ← i + 8
          if i = 32 then i ← 0
          EA ← EA + 1
          n ← n – 1
```

EA is (**r**A | 0). Let *n* = NB if NB ≠ 0, v=32 if NB = 0; *n* is the number of bytes to load. Let *nr* = CEIL(*n*, ÷ 4); *nr* is the number of registers to be loaded with data. *n* consecutive bytes starting at EA are loaded into GPRs **r**D through **r**D + *nr* – 1.

Bytes are loaded left to right in each register. The sequence of registers wraps around to **r0** if required. If the low-order 4 bytes of register **r**D + *nr* – 1 are only partially filled, the unfilled low-order byte(s) of that register are cleared.

If **r**A is in the range of registers specified to be loaded, including the case in which **r**A = 0, the instruction form is invalid.

Under certain conditions (for example, segment boundary crossing) the data alignment exception handler may be invoked. For additional information about data alignment exceptions, see Section 6.4.3, “DSI Exception (0x00300).”

NOTE: In some implementations, this instruction is likely to have greater latency and take longer to execute, perhaps much longer, than a sequence of individual load or store instructions that produce the same results.

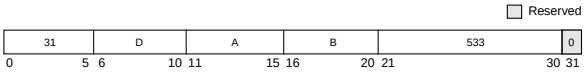
Other registers altered: None

PowerPC Architecture Level	Supervisor Level	PowerPC Optional	Form
UISA			X

lswx lswx

Load String Word Indexed (x'7C00 042A')

lswx **r**D,**r**A,**r**B




```
if rA = 0
     then b ← 0
     else b ← (rA)
EA ← b + (rB)
n ← XER[25–31]
r ← rD – 1
i ← 0
rD ← undefined
 do while n > 0
          if i = 0
               then r ← r + 1 (mod 32)
                    GPR(r) ← (32)0
          GPR(r)[i,i + 7] ← MEM(EA, 1)
          i ← i + 8
          if i = 32
               then i ← 0
          EA ← EA + 1
          n ← n – 1
```

EA is the sum (**r**A|0) + (**r**B). Let *n* = XER[25–31]; *n* is the number of bytes to load. Let *nr* = CEIL(*n* 4); *nr* is the number of registers to receive data. If *n* > 0, *n* consecutive bytes starting at EA are loaded into GPRs **r**D through **r**D + *nr* – 1.

Bytes are loaded left to right in each register. The sequence of registers wraps around through **r0** if required. If the low-order four bytes of **r**D + *nr* – 1 are only partially filled, the unfilled low-order byte(s) of that register are cleared. If *n* = 0, the contents of **r**D are undefined.

If **r**A or **r**B is in the range of registers specified to be loaded, including the case in which **r**A = 0, either the system illegal instruction error handler is invoked or the results are boundedly undefined.

If **r**D = **r**A or **r**D = **r**B, the instruction form is invalid.

If **r**D and **r**A both specify GPR0, the form is invalid.

Under certain conditions (for example, segment boundary crossing) the data alignment exception handler may be invoked. For additional information about data alignment exceptions, see Section 6.4.3, “DSI Exception (0x00300).”

NOTE: In some implementations, this instruction is likely to have a greater latency and take longer to execute, perhaps much longer, than a sequence of individual load or store instructions that produce the same results.

Other registers altered:

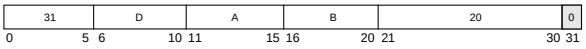
- None

PowerPC Architecture Level	Supervisor Level	PowerPC Optional	Form
UISA			X

lwarx lwarx

Load Word and Reserve Indexed (x'7C00 0028')

lwarx **r**D,**r**A,**r**B




```
if rA = 0
     then b ← 0
     else b ← (rA)
EA ← b + (rB)
RESERVE ← 1
RESERVE_ADDR ← physical_addr(EA)
rD ← MEM(EA,4)
```

EA is the sum (**r**A|0) + (**r**B).

The word in memory addressed by EA is loaded into **r**D.

This instruction creates a reservation for use by a store word conditional indexed (**stwcx.**)instruction. The physical address computed from EA is associated with the reservation, and replaces any address previously associated with the reservation.

EA must be a multiple of four. If it is not, either the system alignment exception handler is invoked or the results are boundedly undefined. For additional information about alignment and DSI exceptions, see Section 6.4.3, “DSI Exception (0x00300).”

When the RESERVE bit is set, the processor enables hardware snooping for the block of memory addressed by the RESERVE address. If the processor detects that another processor writes to the block of memory it has reserved, it clears the RESERVE bit. The **stwcx.** instruction will only do a store if the RESERVE bit is set. The **stwcx.** instruction sets the CR0[EQ] bit if the store was successful and clears it if it failed. The **lwarx** and **stwcx.** combination can be used for atomic read-modify-write sequences.

NOTE: The atomic sequence is not guaranteed, but its failure can be detected if CR0[EQ] = 0 after the **stwcx.** instruction.

Other registers altered:

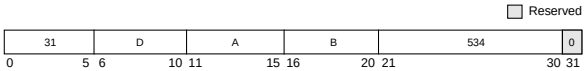
- None

PowerPC Architecture Level	Supervisor Level	PowerPC Optional	Form
UISA			X

lwbrx

Load Word Byte-Reverse Indexed (x'7C00 042C')

lwbrx **r**D,**r**A,**r**B




```
if rA = 0
     then b ← 0
     else b ← (rA)
EA ← b + (rB)
rD ← MEM(EA + 3, 1) || MEM(EA + 2, 1) || MEM(EA + 1, 1) || MEM(EA, 1)
```

EA is the sum (**r**A|0) + **r**B. Bits 0–7 of the word in memory addressed by EA are loaded into the low-order 8 bits of **r**D. Bits 8–15 of the word in memory addressed by EA are loaded into the subsequent low-order 8 bits of **r**D. Bits 16–23 of the word in memory addressed by EA are loaded into the subsequent low-order eight bits of **r**D. Bits 24–31 of the word in memory addressed by EA are loaded into the subsequent low-order 8 bits of **r**D.

The PowerPC architecture cautions programmers that some implementations of the architecture may run the **lwbrx** instructions with greater latency than other types of load instructions.

Other registers altered:

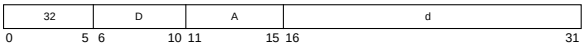
- None

PowerPC Architecture Level	Supervisor Level	PowerPC Optional	Form
UISA			X

lwz

Load Word and Zero (x'8000 0000')

lwz **r**D,d(**r**A)




```
if rA = 0
     then b ← 0
     else b ← (rA)
EA ← b + EXTS(d)
rD ← MEM(EA, 4)
```

EA is the sum (**r**A|0) + d. The word in memory addressed by EA is loaded into **r**D.

Other registers altered:

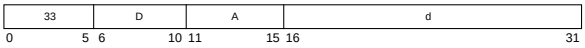
- None

PowerPC Architecture Level	Supervisor Level	PowerPC Optional	Form
UISA			D

lwzu

Load Word and Zero with Update (x'8400 0000')

lwzu **r**D,d(**r**A)




```
EA ← (rA) + EXTS(d)
rD ← MEM(EA, 4)
rA ← EA
```

EA is the sum (**r**A) + d. The word in memory addressed by EA is loaded into **r**D.

EA is placed into **r**A.

If **r**A = 0, or **r**A = **r**D, the instruction form is invalid.

Other registers altered:

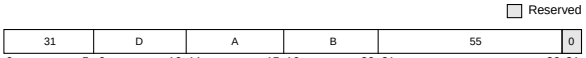
- None

PowerPC Architecture Level	Supervisor Level	PowerPC Optional	Form
UISA			D

lwzux

Load Word and Zero with Update Indexed (x'7C00 006E')

lwzux **r**D,**r**A,**r**B




```
EA ← (rA) + (rB)
rD ← MEM(EA, 4)
rA ← EA
```

EA is the sum (**r**A) + (**r**B). The word in memory addressed by EA is loaded into **r**D.

EA is placed into **r**A.

If **r**A = 0, or **r**A = **r**D, the instruction form is invalid.

Other registers altered:

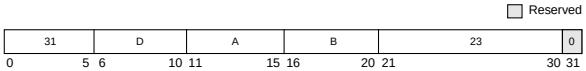
- None

PowerPC Architecture Level	Supervisor Level	PowerPC Optional	Form
UISA			X

lwzx

Load Word and Zero Indexed (x'7C00 002E')

lwzx **r**D,**r**A,**r**B




```
if rA = 0
     then b ← 0
     else b ← (rA)
EA ← b + (rB)
rD ← MEM(EA, 4)
```

EA is the sum (**r**A|0) + (**r**B). The word in memory addressed by EA is loaded into **r**D.

Other registers altered:

- None

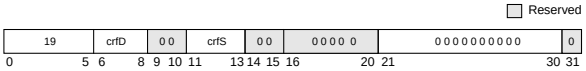
8

PowerPC Architecture Level	Supervisor Level	PowerPC Optional	Form
UISA			X

mcrf

Move Condition Register Field (x'4C00 0000')

mcrf **crf**D,**crf**S




```
CR[(4 * crfD)-(4 * crfD + 3)] ← CR[(4 * crfS)-(4 * crfS + 3)]
```

The contents of condition register field **crf**S are copied into condition register field **crf**D. All other condition register fields remain unchanged.

Other registers altered:

- Condition Register (CR field specified by operand **crf**D):
  Affected: LT, GT, EQ, SO

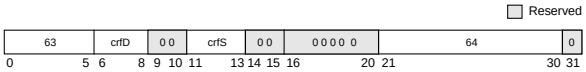
8

PowerPC Architecture Level	Supervisor Level	PowerPC Optional	Form
UISA			XL

mcrfs

Move to Condition Register from FPSCR (x'FC00 0080')

mcrfs **crf**D,**crf**S



The contents of FPSCR field **crf**S are copied to CR field **crf**D. All exception bits copied (except FEX and VX) are cleared in the FPSCR.

Other registers altered:

- Condition Register (CR field specified by operand **crf**D):
  Affected: FX, FEX, VX, OX
- Floating-Point Status and Control Register:
  Affected: FX, OX (if **crf**S = 0)
  Affected: UX, ZX, XX, VXSNAN (if **crf**S = 1)
  Affected: VXISI, VXIDI, VXZDZ, VXIMZ (if **crf**S = 2)
  Affected: VXVC (if **crf**S = 3)
  Affected: VXSOFT, VXSQRT, VXCVI (if **crf**S = 5)

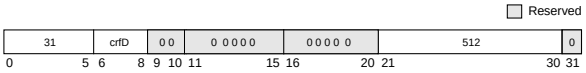
8

PowerPC Architecture Level	Supervisor Level	PowerPC Optional	Form
UISA			X

mcrxr

Move to Condition Register from XER (x'7C00 0400')

mcrxr **crf**D




```
CR[(4 * crfD)-(4 * crfD + 3)] ← XER[0-3]
XER[0-3] ← 0b0000
```

The contents of XER[0–3] are copied into the condition register field designated by **crf**D. All other fields of the condition register remain unchanged. XER[0–3] is cleared.

Other registers altered:

- Condition Register (CR field specified by operand **crf**D):
  Affected: LT, GT, EQ, SO
- XER[0–3]

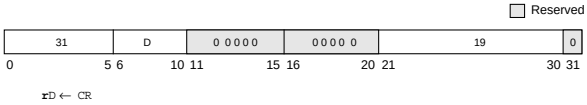
8

PowerPC Architecture Level	Supervisor Level	PowerPC Optional	Form
UISA			X

mfcr

Move from Condition Register ((x'7C00 0026')

mfcr **r**D



rD ← CR

The contents of the condition register (CR) are placed into **r**D.

Other registers altered:

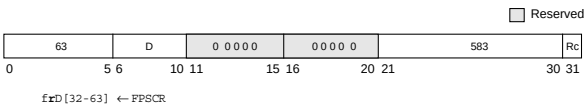
- None

PowerPC Architecture Level	Supervisor Level	PowerPC Optional	Form
UISA			X

mffs*x*

Move from FPSCR (x'FC00 048E')

mffs **fr**D (Rc = 0)
mffs. **fr**D (Rc = 1)



fr**r**D[32-63] ← FPSCR

The contents of the floating-point status and control register (FPSCR) are placed into the low-order bits of register **fr**D. The high-order bits of register **fr**D are undefined.

Other registers altered:

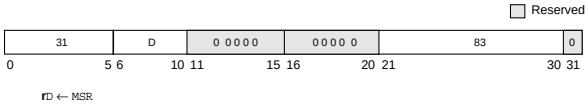
- Condition Register (CR1 field):
  Affected: FX, FEX, VX, OX (If Rc = 1)

PowerPC Architecture Level	Supervisor Level	PowerPC Optional	Form
UISA			X

mfmsr

Move from Machine State Register (x'7C00 00A6')

mfmsr **r**D



rD ← MSR

The contents of the MSR are placed into **r**D.

This is a supervisor-level instruction.

Other registers altered

None

PowerPC Architecture Level	Supervisor Level	PowerPC Optional	Form
OEA	yes		X

mfspr

Move from Special-Purpose Register (x'7C00 02A6')

mfspr **r**D,SPR

NOTE: *This is a split field.

n ← spr[5-9] || spr[0-4]
rD ← SPR(*n*)

In the PowerPC UISA, the SPR field denotes a special-purpose register, encoded as shown in Table 8-9.The contents of the designated special purpose register are placed into **r**D

.

Table 8-9. PowerPC UISA SPR Encodings for mfspr

SPR**			Register Name
Decimal	spr[5–9]	spr[0–4]	
1	00000	00001	XER
8	00000	01000	LR
9	00000	01001	CTR

** **Note:** The order of the two 5-bit halves of the SPR number is reversed compared with the actual instruction coding.

If the SPR field contains any value other than one of the values shown in Table 8-9 (and the processor is in user mode), one of the following occurs:

- The system illegal instruction error handler is invoked.
- The system supervisor-level instruction error handler is invoked.
- The results are boundedly undefined.

Other registers altered:

- None

Simplified mnemonics:

mfxer rD equivalent to **mfspr r**D**,1**
mflr rD equivalent to **mfspr r**D**,8**
mfctr rD equivalent to **mfspr r**D**,9**

In the PowerPC OEA, the SPR field denotes a special-purpose register, encoded as shown in Table 8-10. The contents of the designated SPR are placed into **r**D.

In the PowerPC UISA, the SPR field denotes a special-purpose register, encoded as shown in Table 8-10. If the SPR[0] = 0 (Access type User), the contents of the designated SPR are placed into **r**D.

NOTE: For this instruction (**mfspr**), SPR[0] = 1 is supervisor-level, if and only if reading the register. Execution of this instruction specifying a defined and supervisor-level register when MSR[PR] = 1 results in a privileged instruction type program exception.

If MSR[PR] = 1, the only effect of executing an instruction with an SPR number that is not shown in Table 8-10 and has SPR[0] = 1 is to cause a supervisor-level instruction type program exception or an illegal instruction type program exception. For all other cases, MSR[PR] = 0 or SPR[0] = 0. If the SPR field contains any value that is not shown in Table 8-10, either an illegal instruction type program exception occurs or the results are boundedly undefined.

Other registers altered:

None

8

Table 8-10. PowerPC OEA SPR Encodings for mfspr

SPR[1]			Register Name	Access
Decimal	spr[5–9]	spr[0–4]		
1	00000	00001	XER	User
8	00000	01000	LR	User
9	00000	01001	CTR	User
18	00000	10010	DSISR	Supervisor
19	00000	10011	DAR	Supervisor
22	00000	10110	DEC	Supervisor
25	00000	11001	SDR1	Supervisor
26	00000	11010	SRR0	Supervisor
27	00000	11011	SRR1	Supervisor
272	01000	10000	SPRG0	Supervisor
273	01000	10001	SPRG1	Supervisor
274	01000	10010	SPRG2	Supervisor
275	01000	10011	SPRG3	Supervisor
282	01000	11010	EAR	Supervisor
287	01000	11111	PVR	Supervisor

8

Table 8-10. PowerPC OEA SPR Encodings for mfspr (Continued)

SPR[1]			Register Name	Access
Decimal	spr[5–9]	spr[0–4]		
528	10000	10000	IBAT0U	Supervisor
529	10000	10001	IBAT0L	Supervisor
530	10000	10010	IBAT1U	Supervisor
531	10000	10011	IBAT1L	Supervisor
532	10000	10100	IBAT2U	Supervisor
533	10000	10101	IBAT2L	Supervisor
534	10000	10110	IBAT3U	Supervisor
535	10000	10111	IBAT3L	Supervisor
536	10000	11000	DBAT0U	Supervisor
537	10000	11001	DBAT0L	Supervisor
538	10000	11010	DBAT1U	Supervisor
539	10000	11011	DBAT1L	Supervisor
540	10000	11100	DBAT2U	Supervisor
541	10000	11101	DBAT2L	Supervisor
542	10000	11110	DBAT3U	Supervisor
543	10000	11111	DBAT3L	Supervisor
1013	11111	10101	DABR	Supervisor

[1]**Note**: The order of the two 5-bit halves of the SPR number is reversed compared with actual instruction coding.

For **mtspr** and **mfspr** instructions, the SPR number coded in assembly language does not appear directly as a 10-bit binary number in the instruction. The number coded is split into two 5-bit halves that are reversed in the instruction, with the high-order five bits appearing in bits 16–20 of the instruction and the low-order five bits in bits 11–15.

NOTE: **mfspr** is supervisor-level only if SPR[0] = 1.

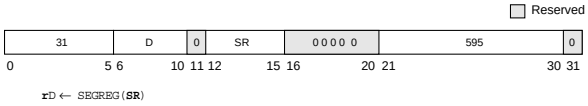
PowerPC Architecture Level	Supervisor Level	PowerPC Optional	Form
UISA/OEA	yes*		XFX

mfsr

mfsr

Move from Segment Register (x'7C00 04A6')

mfsr rD,SR



rD ← SEGREG(**SR**)

The contents of the segment register SR are copied into **r**D.

This is a supervisor-level instruction.

Other registers altered:

- None

8

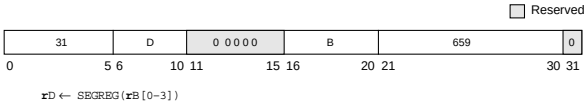
PowerPC Architecture Level	Supervisor Level	PowerPC Optional	Form
OEA	yes		X

mfsrin

mfsrin

Move from Segment Register Indirect (x'7C00 0526')

mfsrin **r**D,**r**B



rD ← SEGREG(**r**B[0–3])

The contents of the segment register selected by bits 0–3 of **r**B are copied into **r**D.

This is a supervisor-level instruction.

The **r**A field is not defined for the **mfsrin** instruction in the PowerPC architecture. However, **mfsrin** performs the same function in the PowerPC architecture as does the **mfsri** instruction in the POWER architecture (if **r**A = 0).

8

Other registers altered:

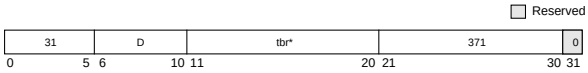
- None

PowerPC Architecture Level	Supervisor Level	PowerPC Optional	Form
OEA	yes		X

mftb

Move from Time Base (x'7C00 02E6')

mftb **r**D,TBR



NOTE: This is a split field.

```
n ← tbr[5–9] || tbr[0–4]
if n = 268
      then rD ← TBL
      else if n = 269
                then rD ← TBU
                else error (invalid TBR field)
```

The contents of the designated register are copied into **r**D. The TBR field denotes either the TBL or TBU, encoded as shown in Table 8-11.

8

Table 8-11. TBR Encodings for mftb

TBR*			Register Name	Access
Decimal	tbr[5–9]	tbr[0–4]		
268	01000	01100	TBL	User
269	01000	01101	TBU	User

*__Note__: The order of the two 5-bit halves of the TBR number is reversed.

If the TBR field contains any value other than one of the values shown in Table 8-11, then one of the following occurs:

- The system illegal instruction error handler is invoked.
- The system supervisor-level instruction error handler is invoked.
- The results are boundedly undefined.

It is important to note that some implementations may implement **mftb** and **mfspr** identically, therefore, a TBR number must not match an SPR number.

For more information on the time base refer to Section 2.2, “PowerPC VEA Register Set—Time Base.”

Other registers altered:

- None

Simplified mnemonics:

mftb **r**D equivalent to **mftb** rD,**268**
mftbu **r**D equivalent to **mftb** rD,**269**

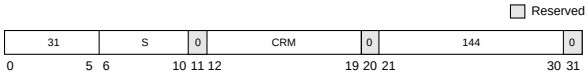
8

PowerPC Architecture Level	Supervisor Level	PowerPC Optional	Form
VEA			XFX

mtcrf

Move to Condition Register Fields (x'7C00 0120')

mtcrf CRM,**r**S




```
mask ← (4)(CRM[0]) || (4)(CRM[1]) ||... (4)(CRM[7])
CR ← (rS & mask) | (CR & ¬ mask)
```

The contents of **r**S are placed into the condition register under control of the field mask specified by CRM. The field mask identifies the 4-bit fields affected. Let i be an integer in the range 0–7. If CRM(i) = 1, CR field i (CR bits 4 ∗ i through 4 ∗ i + 3) is set to the contents of the corresponding field of **r**S.

NOTE: Updating a subset of the eight fields of the condition register may have substantially poorer performance on some implementations than updating all of the fields.

8

Other registers altered:

- CR fields selected by mask

Simplified mnemonics:

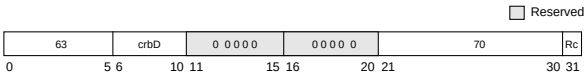
mtcr **r**S equivalent to **mtcrf** 0xFF,**r**S

PowerPC Architecture Level	Supervisor Level	PowerPC Optional	Form
UISA			XFX

mtfsb0*x*

Move to FPSCR Bit 0 (x'FC00 008C')

mtfsb0 **crb**D (Rc = 0)
mtfsb0. **crb**D (Rc = 1)




```
FPSRC[crbD] ← 0
```

Bit **crb**D of the FPSCR is cleared.

Other registers altered:

- Condition Register (CR1 field):
  Affected: FX, FEX, VX, OX (If Rc = 1)
- Floating-Point Status and Control Register:
  Affected: FPSCR bit **crb**D
  NOTE: Bits 1 and 2 (FEX and VX) cannot be explicitly cleared.

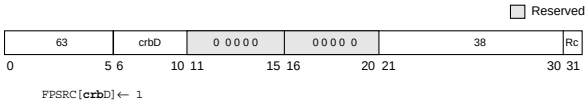
8

PowerPC Architecture Level	Supervisor Level	PowerPC Optional	Form
UISA			X

mtfsb1*x* mtfsb1*x*

Move to FPSCR Bit 1 (x'FC00 004C')

mtfsb1 **crb**D (Rc = 0)
mtfsb1. **crb**D (Rc = 1)




```
FPSRC[crbD]← 1
```

Bit **crb**D of the FPSCR is set.

Other registers altered:

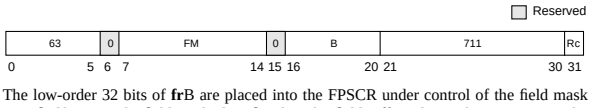
- Condition Register (CR1 field):
  Affected: FX, FEX, VX, OX (If Rc = 1)
- Floating-Point Status and Control Register:
  Affected: FPSCR bit **crb**D and FX
  NOTE: Bits 1 and 2 (FEX and VX) cannot be explicitly set.

PowerPC Architecture Level	Supervisor Level	PowerPC Optional	Form
UISA			X

mtfsf*x* mtfsf*x*

Move to FPSCR Fields (x'FC00 058E')

mtfsf FM,**fr**B (Rc = 0)
mtfsf. FM,**fr**B (Rc = 1)



The low-order 32 bits of **fr**B are placed into the FPSCR under control of the field mask specified by FM. The field mask identifies the 4-bit fields affected. Let i be an integer in the range 0–7. If FM[i] = 1, FPSCR field i (FPSCR bits 4 * i through 4 * i + 3) is set to the contents of the corresponding field of the low-order 32 bits of register **fr**B.

FPSCR[FX] is altered only if FM[0] = 1.

Updating fewer than all eight fields of the FPSCR may have substantially poorer performance on some implementations than updating all the fields.

When FPSCR[0–3] is specified, bits 0 (FX) and 3 (OX) are set to the values of **fr**B[32] and **fr**B[35] (that is, even if this instruction causes OX to change from 0 to 1, FX is set from **fr**B[32] and not by the usual rule that FX is set when an exception bit changes from 0 to 1). Bits 1 and 2 (FEX and VX) are set according to the usual rule and not from **fr**B[33–34].

Other registers altered:

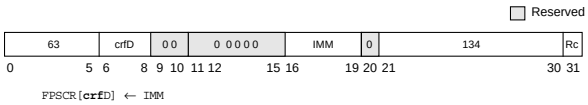
- Condition Register (CR1 field):
  Affected: FX, FEX, VX, OX (If Rc = 1)
- Floating-Point Status and Control Register:
  Affected: FPSCR fields selected by mask

PowerPC Architecture Level	Supervisor Level	PowerPC Optional	Form
UISA			XFL

mtfsfi*x* mtfsfi*x*

Move to FPSCR Field Immediate (x'FC00 010C')

mtfsfi **crf**D,IMM (Rc = 0)
mtfsfi. **crf**D,IMM (Rc = 1)




```
FPSCR[crfD] ← IMM
```

The value of the IMM field is placed into FPSCR field **crf**D.

FPSCR[FX] is altered only if **crf**D = 0.

When FPSCR[0–3] is specified, bits 0 (FX) and 3 (OX) are set to the values of IMM[0] and IMM[3] (that is, even if this instruction causes OX to change from 0 to 1, FX is set from IMM[0] and not by the usual rule that FX is set when an exception bit changes from 0 to 1). Bits 1 and 2 (FEX and VX) are set according to the usual rule and not from IMM[1–2].

Other registers altered:

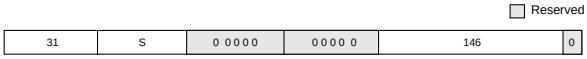
- Condition Register (CR1 field):
  Affected: FX, FEX, VX, OX (If Rc = 1)
- Floating-Point Status and Control Register:
  Affected: FPSCR field **crf**D

PowerPC Architecture Level	Supervisor Level	PowerPC Optional	Form
UISA			X

mtmsr mtmsr

Move to Machine State Register (x'7C00 0124')

mtmsr **r**S




```
MSR ← (rS)
```

The contents of **r**S are placed into the MSR.

This is a supervisor-level instruction. It is also an execution synchronizing instruction except with respect to alterations to the POW and LE bits. Refer to Section 2.3.18, “Synchronization Requirements for Special Registers and for Lookaside Buffers,” for more information.

In addition, alterations to the MSR[EE] and MSR[RI] bits are effective as soon as the instruction completes. Thus if MSR[EE] = 0 and an external or decrementer exception is pending, executing an **mtmsr** instruction that sets MSR[EE] = 1 will cause the external or decrementer exception to be taken before the next instruction is executed, if no higher priority exception exists.

Other registers altered:

- MSR

PowerPC Architecture Level	Supervisor Level	PowerPC Optional	Form
OEA	yes		X

mtspr mtspr

Move to Special-Purpose Register (x'7C00 03A6')

mtspr SPR,**r**S

NOTE: This is a split field.

```
n ← spr[5–9] || spr[0–4]
SPR(n) ← (rS)
```

In the PowerPC UISA, the SPR field denotes a special-purpose register, encoded as shown in Table 8-12. The contents of **r**S are placed into the designated special-purpose register.

Table 8-12. PowerPC UISA SPR Encodings for mtspr

SPR**		Register Name
Decimal	spr[5–9] spr[0–4]	
1	00000 00001	XER
8	00000 01000	LR
9	00000 01001	CTR

** **Note**: The order of the two 5-bit halves of the SPR number is reversed compared with actual instruction coding.

If the SPR field contains any value other than one of the values shown in Table 8-12, and the processor is operating in user mode, one of the following occurs:

- The system illegal instruction error handler is invoked.
- The system supervisor instruction error handler is invoked.
- The results are boundedly undefined.

Other registers altered:

- See Table 8-12.

Simplified mnemonics:

mtxer	**r**D	equivalent to	**mtspr 1,r**D
mtlr	**r**D	equivalent to	**mtspr 8,r**D
mtctr	**r**D	equivalent to	**mtspr 9,r**D

8

In the PowerPC OEA, the SPR field denotes a special-purpose register, encoded as shown in Table 8-13. The contents of **r**S are placed into the designated special-purpose register.

In the PowerPC UISA, if the SPR[0]=0 (Access is User) the contents of **r**S are placed into the designated special-purpose register.

For this instruction, SPRs TBL and TBU are treated as separate 32-bit registers; setting one leaves the other unaltered.

The value of SPR[0] = 1 if and only if writing the register is a supervisor-level operation. Execution of this instruction specifying a defined and supervisor-level register when MSR[PR] = 1 results in a privileged instruction type program exception.

If MSR[PR] = 1 then the only effect of executing an instruction with an SPR number that is not shown in Table 8-13 and has SPR[0] = 1 is to cause a privileged instruction type program exception or an illegal instruction type program exception. For all other cases, MSR[PR] = 0 or SPR[0] = 0, if the SPR field contains any value that is not shown in Table 8-13, either an illegal instruction type program exception occurs or the results are boundedly undefined.

8

Other registers altered:

- See Table 8-13.

Table 8-13. PowerPC OEA SPR Encodings for mtspr

SPR[1]		Register Name	Access
Decimal	spr[5–9] spr[0–4]		
1	00000 00001	XER	User
8	00000 01000	LR	User
9	00000 01001	CTR	User
18	00000 10010	DSISR	Supervisor
19	00000 10011	DAR	Supervisor
22	00000 10110	DEC	Supervisor
25	00000 11001	SDR1	Supervisor
26	00000 11010	SRR0	Supervisor
27	00000 11011	SRR1	Supervisor
272	01000 10000	SPRG0	Supervisor
273	01000 10001	SPRG1	Supervisor
274	01000 10010	SPRG2	Supervisor
275	01000 10011	SPRG3	Supervisor
282	01000 11010	EAR	Supervisor

Table 8-13. PowerPC OEA SPR Encodings for mtspr (Continued)

SPR[1]		Register Name	Access
Decimal	spr[5–9] spr[0–4]		
284	01000 11100	TBL	Supervisor
285	01000 11101	TBU	Supervisor
528	10000 10000	IBAT0U	Supervisor
529	10000 10001	IBAT0L	Supervisor
530	10000 10010	IBAT1U	Supervisor
531	10000 10011	IBAT1L	Supervisor
532	10000 10100	IBAT2U	Supervisor
533	10000 10101	IBAT2L	Supervisor
534	10000 10110	IBAT3U	Supervisor
535	10000 10111	IBAT3L	Supervisor
536	10000 11000	DBAT0U	Supervisor
537	10000 11001	DBAT0L	Supervisor
538	10000 11010	DBAT1U	Supervisor
539	10000 11011	DBAT1L	Supervisor
540	10000 11100	DBAT2U	Supervisor
541	10000 11101	DBAT2L	Supervisor
542	10000 11110	DBAT3U	Supervisor
543	10000 11111	DBAT3L	Supervisor
1013	11111 10101	DABR	Supervisor

[1] **Note**: The order of the two 5-bit halves of the SPR number is reversed. For **mtspr** and **mfspr** instructions, the SPR number coded in assembly language does not appear directly as a 10-bit binary number in the instruction. The number coded is split into two 5-bit halves that are reversed in the instruction, with the high-order five bits appearing in bits 16–20 of the instruction and the low-order five bits in bits 11–15.

NOTE: **mtspr** is supervisor-level only if SPR[0] = 1.

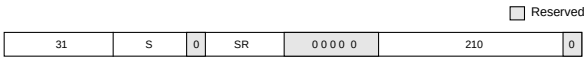
PowerPC Architecture Level	Supervisor Level	PowerPC Optional	Form
UISA/OEA	yes*		XFX

8

mtsr mtsr

Move to Segment Register (x'7C00 01A4')

mtsr SR,**r**S




```
SEGREG(SR) ← (rS)
```

The contents of **r**S are placed into SR.

This is a supervisor-level instruction.

Other registers altered:

- None

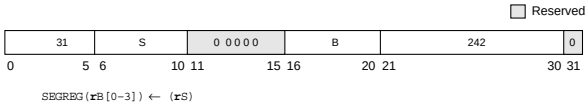
PowerPC Architecture Level	Supervisor Level	PowerPC Optional	Form
OEA	yes		X

mtsrin

mtsrin

Move to Segment Register Indirect (x'7C00 01E4')

mtsrin **r**S,**r**B




```
SEGREG(rB[0-3]) ← (rS)
```

The contents of **r**S are copied to the segment register selected by bits 0–3 of **r**B.

This is a supervisor-level instruction.

NOTE: The PowerPC architecture does not define the **r**A field for the **mtsrin** instruction. However, **mtsrin** performs the same function in the PowerPC architecture as does the **mtsri** instruction in the POWER architecture (if **r**A = 0).

8

Other registers altered:

- None

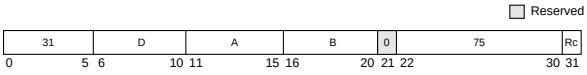
PowerPC Architecture Level	Supervisor Level	PowerPC Optional	Form
OEA	yes		X

mulhw*x*

mulhw*x*

Multiply High Word (x'7C00 0096')

mulhw **r**D,**r**A,**r**B (Rc = 0)
mulhw. **r**D,**r**A,**r**B (Rc = 1)




```
prod[0-63] ← (rA) * (rB)
rD ← prod[0-31]
```

The 64-bit product is formed from the contents of **r**A and **r**B. The high-order 32 bits of the 64-bit product of the operands are placed into **r**D.

Both the operands and the product are interpreted as signed integers.

8

This instruction may execute faster on some implementations if **r**B contains the operand having the smaller absolute value.

Other registers altered:

- Condition Register (CR0 field):
  Affected: LT, GT, EQ, SO (If Rc = 1)

PowerPC Architecture Level	Supervisor Level	PowerPC Optional	Form
UISA			XO

mulhwu*x*

mulhwu*x*

Multiply High Word Unsigned (x'7C00 0016')

mulhwu **r**D,**r**A,**r**B (Rc = 0)
mulhwu. **r**D,**r**A,**r**B (Rc = 1)


```
prod[0-63] ← (rA) * (rB)
rD ← prod[0-31]
```

The 32-bit operands are the contents of **r**A and **r**B. The high-order 32 bits of the 64-bit product of the operands are placed into **r**D.

Both the operands and the product are interpreted as unsigned integers, except that if Rc = 1 the first three bits of CR0 field are set by signed comparison of the result to zero.

8

This instruction may execute faster on some implementations if **r**B contains the operand having the smaller absolute value.

Other registers altered:

- Condition Register (CR0 field):
  Affected: LT, GT, EQ, SO (If Rc = 1)

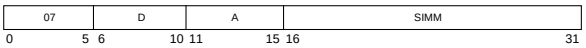
PowerPC Architecture Level	Supervisor Level	PowerPC Optional	Form
UISA			XO

mulli

mulli

Multiply Low Immediate (x'1C00 0000')

mulli **r**D,**r**A,SIMM




```
prod[0-63]← (rA) * EXTS(SIMM)
rD ← prod[32-63]
```

The first operand is (**r**A). The second operand is the sign-extended value of the SIMM field. The low-order 32-bits of the 64-bit product of the operands are placed into **r**D.

Both the operands and the product are interpreted as signed integers. The low-order 32-bits of the product are calculated independently of whether the operands are treated as signed or unsigned 32-bit integers.

8

This instruction can be used with **mulhd***x* or **mulhw***x* to calculate a full 64-bit product.

Other registers altered:

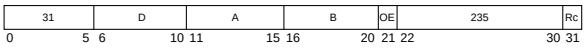
- None

PowerPC Architecture Level	Supervisor Level	PowerPC Optional	Form
UISA			D

mullw*x* mullw*x*

Multiply Low Word (x'7C00 01D6')

mullw	**r**D,**r**A,**r**B	(OE = 0 Rc = 0)
mullw.	**r**D,**r**A,**r**B	(OE = 0 Rc = 1)
mullwo	**r**D,**r**A,**r**B	(OE = 1 Rc = 0)
mullwo.	**r**D,**r**A,**r**B	(OE = 1 Rc = 1)




```
prod[0-63] ← (rA) * (rB)
rD ← prod[32-63]
```

The 32-bit operands are the contents of **r**A and **r**B. The low-order 32-bits of the 64-bit product (**r**A) * (**r**B) are placed into **r**D.

The low-order 32-bits of the product are independent of whether the operands are regarded as signed or unsigned 32-bit integers.

If OE = 1, then OV is set if the product cannot be represented in 32 bits. Both the operands and the product are interpreted as signed integers.

This instruction can be used with **mulhw***x* to calculate a full 64-bit product.

NOTE: This instruction may execute faster on some implementations if **r**B contains the operand having the smaller absolute value.

Other registers altered:

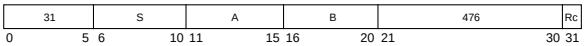
- Condition Register (CR0 field):
  Affected: LT, GT, EQ, SO (If Rc = 1)
  NOTE: CR0 field may not reflect the infinitely precise result if overflow occurs (see next).
- XER:
  Affected: SO, OV (If OE = 1)

PowerPC Architecture Level	Supervisor Level	PowerPC Optional	Form
UISA			XO

nand*x* nand*x*

NAND (x'7C00 03B8')

nand	**r**A,**r**S,**r**B	(Rc = 0)
nand.	**r**A,**r**S,**r**B	(Rc = 1)




```
rA ← ¬ ((rS) & (rB))
```

The contents of **r**S are ANDed with the contents of **r**B and the complemented result is placed into **r**A.

nand with **r**S = **r**B can be used to obtain the one's complement.

Other registers altered:

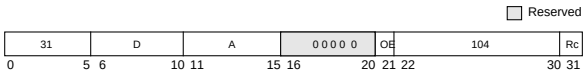
- Condition Register (CR0 field):
  Affected: LT, GT, EQ, SO (If Rc = 1)

PowerPC Architecture Level	Supervisor Level	PowerPC Optional	Form
UISA			X

neg*x* neg*x*

Negate (x'7C00 00D0')

neg	**r**D,**r**A	(OE = 0 Rc = 0)
neg.	**r**D,**r**A	(OE = 0 Rc = 1)
nego	**r**D,**r**A	(OE = 1 Rc = 0)
nego.	**r**D,**r**A	(OE = 1 Rc = 1)




```
rD ← ¬ (rA) + 1
```

The value 1 is added to the one's complement of the value in **r**A, and the resulting two's complement is placed into **r**D.

If **r**A contains the most negative 32-bit number (0x8000_0000), the result is the most negative number and, if OE = 1, OV is set.

Other registers altered:

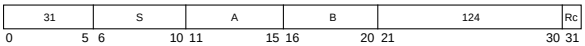
- Condition Register (CR0 field):
  Affected: LT, GT, EQ, SO (If Rc = 1)
- XER:
  Affected: SO OV (If OE = 1)

PowerPC Architecture Level	Supervisor Level	PowerPC Optional	Form
UISA			XO

nor*x* nor*x*

NOR (x'7C00 00F8')

nor	**r**A,**r**S,**r**B	(Rc = 0)
nor.	**r**A,**r**S,**r**B	(Rc = 1)




```
rA ← ¬ ((rS) | (rB))
```

The contents of **r**S are ORed with the contents of **r**B and the complemented result is placed into **r**A.

nor with **r**S = **r**B can be used to obtain the one's complement.

Other registers altered:

- Condition Register (CR0 field):
  Affected: LT, GT, EQ, SO (If Rc = 1)

Simplified mnemonics:

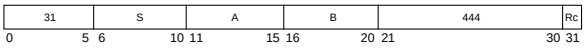
not **r**D,**r**S equivalent to **nor** **r**A,**r**S,**r**S

PowerPC Architecture Level	Supervisor Level	PowerPC Optional	Form
UISA			X

or*x* **or*x***

OR (x'7C00 0378')

or **r**A,**r**S,**r**B (Rc = 0)
or. **r**A,**r**S,**r**B (Rc = 1)




```
rA ← (rS) | (rB)
```

The contents of **r**S are ORed with the contents of **r**B and the result is placed into **r**A.

The simplified mnemonic **mr** (shown below) demonstrates the use of the **or** instruction to move register contents.

Other registers altered:

- Condition Register (CR0 field):
  Affected: LT, GT, EQ, SO (If Rc = 1)

Simplified mnemonics:

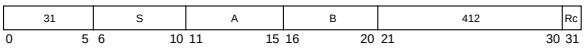
mr **r**A,**r**S equivalent to **or** **r**A,**r**S,**r**S

PowerPC Architecture Level	Supervisor Level	PowerPC Optional	Form
UISA			X

orc*x* **orc*x***

OR with Complement (x'7C00 0338')

orc **r**A,**r**S,**r**B (Rc = 0)
orc. **r**A,**r**S,**r**B (Rc = 1)




```
rA ← (rS) | ¬ (rB)
```

The contents of **r**S are ORed with the complement of the contents of **r**B and the result is placed into **r**A.

Other registers altered:

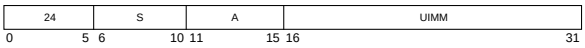
- Condition Register (CR0 field):
  Affected: LT, GT, EQ, SO (If Rc = 1)

PowerPC Architecture Level	Supervisor Level	PowerPC Optional	Form
UISA			X

ori **ori**

OR Immediate (x'6000 0000')

ori **r**A,**r**S,UIMM




```
rA ← (rS) | ((16)0 || UIMM)
```

The contents of **r**S are ORed with 0x0000 || UIMM and the result is placed into **r**A.

The preferred no-op (an instruction that does nothing) is **ori 0,0,0**.

Other registers altered:

- None

Simplified mnemonics:

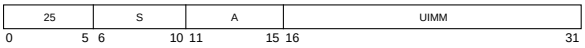
nop equivalent to **ori** **0,0,0**

PowerPC Architecture Level	Supervisor Level	PowerPC Optional	Form
UISA			D

oris **oris**

OR Immediate Shifted (x'6400 0000')

oris **r**A,**r**S,UIMM




```
rA ← (rS) | (UIMM || (16)0)
```

The contents of **r**S are ORed with UIMM || 0x0000 and the result is placed into **r**A.

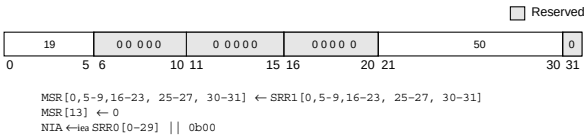
Other registers altered:

- None

PowerPC Architecture Level	Supervisor Level	PowerPC Optional	Form
UISA			D

rfi

Return from Interrupt (x'4C00 0064')




```
MSR[0,5-9,16-23, 25-27, 30-31] ← SRR1[0,5-9,16-23, 25-27, 30-31]
MSR[13] ← 0
NIA ←iea SRR0[0-29] || 0b00
```

Bits SRR1[0,5-9,16–23, 25–27, 30–31] are placed into the corresponding bits of the MSR. MSR[13] is set to 0. If the new MSR value does not enable any pending exceptions, then the next instruction is fetched, under control of the new MSR value, from the address SRR0[0–29] || 0b00. If the new MSR value enables one or more pending exceptions, the exception associated with the highest priority pending exception is generated; in this case the value placed into SRR0 by the exception processing mechanism is the address of the instruction that would have been executed next had the exception not occurred.

8

NOTE: An implementation may define additional MSR bits, and in this case, may also cause them to be saved to SRR1 from MSR on an exception and restored to MSR from SRR1 on an **rfi**.

This is a supervisor-level, context synchronizing instruction.

Other registers altered:

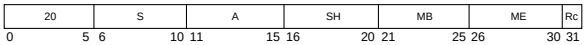
- MSR

PowerPC Architecture Level	Supervisor Level	PowerPC Optional	Form
OEA	YES		XL

rlwimi*x*

Rotate Left Word Immediate then Mask Insert (x'5000 0000')

rlwimi **r**A,**r**S,SH,MB,ME (Rc = 0)
rlwimi. **r**A,**r**S,SH,MB,ME (Rc = 1)




```
n← SH
r ← ROTL(rS, n)
m ← MASK(MB, ME)
rA ← (r & m) | (rA & ¬ m)
```

The contents of **r**S are rotated left the number of bits specified by operand SH. A mask is generated having 1 bits from bit MB through bit ME and 0 bits elsewhere. The rotated data is inserted into **r**A under control of the generated mask.

8

NOTE: **rlwimi** can be used to copy a bit field of any length from register **r**S into the contents of **r**A. This field can start from any bit position in **r**S and be placed into any position in **r**A. The length of the field can range from 0 to 32 bits. The remaining bits in register **r**A remain unchanged:

- To copy byte_0 (bits 0-7) from rS into byte_3 (bits 24-31) of rA, set SH = 8 , MB = *24*, and ME = 31.
- In general, to copy an *n*-bit field that starts in bit position b in register rS into register rA starting a bit position c: set SH = 32 - c + b Mod(32), set MB = *c*, and set ME = (*c* + *n*) – 1 Mod(32).

Other registers altered:

- Condition Register (CR0 field):

  Affected: LT, GT, EQ, SO (if Rc = 1)

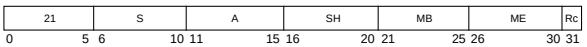
Simplified mnemonics:
insl**w**i rA,rS,n,b equivalent to rlwimirA,rS,32 – b,b,b + n – 1
insr**w**i rA,rS,n,b (n > 0) equivalent to rlwimirA,rS,32 – (b + n),b, (b + n) – 1

PowerPC Architecture Level	Supervisor Level	PowerPC Optional	Form
UISA			M

rlwinm*x*

Rotate Left Word Immediate then AND with Mask (x'5400 0000')

rlwinm **r**A,**r**S,SH,MB,ME (Rc = 0)
rlwinm. **r**A,**r**S,SH,MB,ME (Rc = 1)




```
n← SH
r ← ROTL(rS, n)
m ← MASK(MB , ME)
rA ← r & m
```

The contents of **r**S are rotated left the number of bits specified by operand SH. A mask is generated having 1 bits from bit MB through bit ME and 0 bits elsewhere. The rotated data is ANDed with the generated mask and the result is placed into **r**A.

8

NOTE: **rlwinm** can be used to extract, rotate, shift, and clear bit fields using the methods shown below:

- To extract an *n*-bit field, that starts at bit position *b* in **r**S, right-justified into **r**A (clearing the remaining 32 – *n* bits of **r**A), set SH = *b* + *n*, MB = 32 – *n*, and ME = 31.
- To extract an *n*-bit field, that starts at bit position *b* in **r**S, left-justified into **r**A (clearing the remaining 32 – *n* bits of **r**A), set SH = *b*, MB = 0, and ME = *n* – 1.
- To rotate the contents of a register left (or right) by *n* bits, set SH = *n* (32 – *n*), MB = 0, and ME = 31.
- To shift the contents of a register right by *n* bits, by setting SH = 32 – *n*, MB = *n*, and ME = 31. It can be used to clear the high-order *b* bits of a register and then shift the result left by *n* bits by setting SH = *n*, MB = *b* – *n* and ME = 31 – *n*.
- To clear the low-order *n* bits of a register, by setting SH = 0, MB = 0, and ME = 31 – *n*..

Other registers altered:

- Condition Register (CR0 field):

  Affected: LT, GT, EQ, SO (if Rc = 1)

Simplified mnemonics:

extlwi **r**A,**r**S,*n*,*b* (*n* > 0)	equivalent to	**rlwinm** **r**A,**r**S,b,**0**,*n* – 1
extrwi **r**A,**r**S,*n*,*b* (*n* > 0)	equivalent to	**rlwinm** **r**A,**r**S,b + *n*,32 – *n*,**31**
rotlwi **r**A,**r**S,*n*	equivalent to	**rlwinm** **r**A,**r**S,*n*,**0,31**
rotrwi **r**A,**r**S,*n*	equivalent to	**rlwinm** **r**A,**r**S,**32** – *n*,**0,31**
slwi **r**A,**r**S,*n* (*n* < 32)	equivalent to	**rlwinm** **r**A,**r**S,*n*,**0,**31–*n*
srwi **r**A,**r**S,*n* (*n* < 32)	equivalent to	**rlwinm** **r**A,**r**S,32 – *n*,*n*,**31**
clrlwi **r**A,**r**S,*n* (*n* < 32)	equivalent to	**rlwinm** **r**A,**r**S,**0**,*n*,**31**
clrrwi **r**A,**r**S,*n* (*n* < 32)	equivalent to	**rlwinm** **r**A,**r**S,**0,0**,31 – *n*
clrlslwi **r**A,**r**S,*b*,*n* (*n* ≤ *b* < 32)	equivalent to	**rlwinm** **r**A,**r**S,*n*,*b* – *n*,31 – *n*

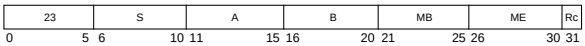
8

PowerPC Architecture Level	Supervisor Level	PowerPC Optional	Form
UISA			M

rlwnm*x* rlwnm*x*

Rotate Left Word then AND with Mask (x'5C00 0000')

rlwnm **r**A,**r**S,**r**B,MB,ME (Rc = 0)
rlwnm. **r**A,**r**S,**r**B,MB,ME (Rc = 1)




```
n ← rB[27-31]
r ← ROTL(rS, n)
m ← MASK(MB, ME)
rA ← r & m
```

The contents of **r**S are rotated left the number of bits specified by the low-order five bits of **r**B. A mask is generated having 1 bits from bit MB through bit ME and 0 bits elsewhere. The rotated data is ANDed with the generated mask and the result is placed into **r**A.

NOTE: **rlwnm** can be used to extract and rotate bit fields using the methods shown as follows:

- To extract an *n*-bit field, that starts at variable bit position *b* in **r**S, right-justified into **r**A (clearing the remaining 32 – *n* bits of **r**A), by setting the low-order five bits of **r**B to *b* + *n*, MB = 32 – *n*, and ME = 31.
- To extract an *n*-bit field, that starts at variable bit position *b* in **r**S, left-justified into **r**A (clearing the remaining 32 – *n* bits of **r**A), by setting the low-order five bits of **r**B to *b*, MB = 0, and ME = *n* – 1.
- To rotate the contents of a register left (or right) by *n* bits, by setting the low-order five bits of **r**B to *n* (32 – *n*), MB = 0, and ME = 31.

Other registers altered:

- Condition Register (CR0 field):

  Affected: LT, GT, EQ, SO (if Rc = 1)

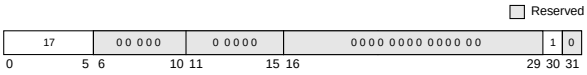
Simplified mnemonics:

rotlw **r**A,**r**S,**r**B equivalent to **rlwnm** **r**A,**r**S,**r**B,**0,31**

PowerPC Architecture Level	Supervisor Level	PowerPC Optional	Form
UISA			M

sc sc

System Call (x'4400 0002')



In the PowerPC UISA, the **sc** instruction calls the operating system to perform a service. When control is returned to the program that executed the system call, the content of the registers depends on the register conventions used by the program providing the system service.

This instruction is context synchronizing, as described in Section 4.1.5.1, "Context Synchronizing Instructions."

Other registers altered:

- Dependent on the system service

In PowerPC OEA, the **sc** instruction does the following:

```
SRR0 ←iea CIA + 4
SRR1[1-4, 10-15] ← 0
SRR1[0,5-9, 16-23, 25-27, 30-31] ← MSR[0,5-9, 16-23, 25-27, 30-31]
MSR ← new_value (see below)
NIA ←iea base_ea + 0xC00 (see below)
```

The EA of the instruction following the **sc** instruction is placed into SRR0. Bits 0, 5-9,16-23, 25-27, and 30-31 of the MSR are placed into the corresponding bits of SRR1, and bits 1-4 and 10-15 of SRR1 are set to undefined values.

NOTE: An implementation may define additional MSR bits, and in this case, may also cause them to be saved to SRR1 from MSR on an exception and restored to MSR from SRR1 on an **rfi**.

Then a system call exception is generated. The exception causes the MSR to be altered as described in Section 6.4, "Exception Definitions."

The exception causes the next instruction to be fetched from offset 0xC00 from the physical base address determined by the new setting of MSR[IP].

Other registers altered:

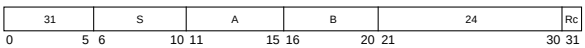
- SRR0
- SRR1
- MSR

PowerPC Architecture Level	Supervisor Level	PowerPC Optional	Form
UISA/OEA			SC

slw*x* slw*x*

Shift Left Word (x'7C00 0030')

slw **r**A,**r**S,**r**B (Rc = 0)
slw. **r**A,**r**S,**r**B (Rc = 1)




```
n ← rB[27-31]
r ← ROTL(rS, n)
if rB[26] = 0
      then m ← MASK(0, 31 – n)
      else m ← (32)0
rA ← r & m
```

The contents of **r**S are shifted left the number of bits specified by the low-order five bits of **r**B. Bits shifted out of position 0 are lost. Zeros are supplied to the vacated positions on the right. The 32-bit result is placed into **r**A. However, shift amounts from 32 to 63 give a zero result.

Other registers altered:

- Condition Register (CR0 field):

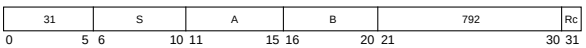
  Affected: LT, GT, EQ, SO (if Rc = 1)

PowerPC Architecture Level	Supervisor Level	PowerPC Optional	Form
UISA			X

sraw*x* sraw*x*

Shift Right Algebraic Word (x'7C00 0630')

sraw **r**A,**r**S,**r**B (Rc = 0)
sraw. **r**A,**r**S,**r**B (Rc = 1)




```
n ← rB[27-31]
r ← ROTL(rS, 32– n)
if rB[26] = 0
      then m ← MASK(n, 31)
      else m ← (32)0
S ← rS(0)
rA ← r & m | (32)S & ¬ m
XER[CA] ← S & ((r & ¬ m) ≠ 0 )
```

The contents of **r**S are shifted right the number of bits specified by the low-order five bits of **r**B (shift amounts between 0-31). Bits shifted out of position 31 are lost. Bit 0 of **r**S is replicated to fill the vacated positions on the left. The 32-bit result is placed into **r**A. XER[CA] is set if **r**S contains a negative number and any 1 bits are shifted out of position 31; otherwise XER[CA] is cleared. A shift amount of zero causes **r**A to receive the 32 bits of **r**S, and XER[CA] to be cleared. However, shift amounts from 32 to 63 give a result of 32 sign bits, and cause XER[CA] to receive the sign bit of **r**S.

NOTE: The **sraw** instruction, followed by **addze**, can be used to divide quickly by 2^n.

Other registers altered:

- Condition Register (CR0 field):

  Affected: LT, GT, EQ, SO (if Rc = 1)
- XER:

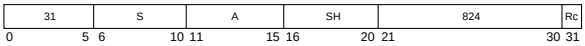
  Affected: CA

PowerPC Architecture Level	Supervisor Level	PowerPC Optional	Form
UISA			X

srawi*x* srawi*x*

Shift Right Algebraic Word Immediate (x'7C00 0670')

srawi **r**A,**r**S,SH (Rc = 0)
srawi. **r**A,**r**S,SH (Rc = 1)




```
n ← SH
r ← ROTL(rS, 32- n)
m← MASK(n, 31)
S ← rS(0)
rA ← r & m | (32)S & ¬ m
XER[CA] ← S & ((r & ¬ m) ≠ 0)
```

The contents of **r**S are shifted right SH bits. Bits shifted out of position 31 are lost. Bit 0 of **r**S is replicated to fill the vacated positions on the left. The result is placed into **r**A. XER[CA] is set if the 32 bits of **r**S contain a negative number and any 1 bits are shifted out of position 31; otherwise XER[CA] is cleared. A shift amount of zero causes **r**A to receive the value of **r**S, and XER[CA] to be cleared.

8

NOTE: The **srawi** instruction, followed by **addze**, can be used to divide quickly by 2^n.

Other registers altered:

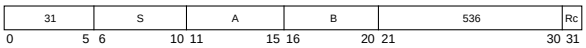
- Condition Register (CR0 field):
  Affected: LT, GT, EQ, SO (if Rc = 1)
- XER:
  Affected: CA

PowerPC Architecture Level	Supervisor Level	PowerPC Optional	Form
UISA			X

srw*x* srw*x*

Shift Right Word (x'7C00 0430')

srw **r**A,**r**S,**r**B (Rc = 0)
srw. **r**A,**r**S,**r**B (Rc = 1)




```
n ← rB[27-31]
r ← ROTL(rS, 32-n)
if rB[26] = 0
     then m ← MASK(n, 31)
     else m ← (32)0
rA ← r & m
```

The contents of **r**S are shifted right the number of bits specified by the low-order five bits of **r**B (shift amounts between 0-31). Bits shifted out of position 31 are lost. Zeros are supplied to the vacated positions on the left. The 32-bit result is placed into **r**A. However, shift amounts from 32 to 63 give a zero result.

8

Other registers altered:

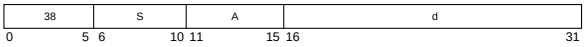
- Condition Register (CR0 field):
  Affected: LT, GT, EQ, SO (if Rc = 1)

PowerPC Architecture Level	Supervisor Level	PowerPC Optional	Form
UISA			X

stb stb

Store Byte (x'9800 0000')

stb **r**S,d(**r**A)




```
if rA = 0
     then b ← 0
     else b ← (rA)
EA ← b + EXTS(d)
MEM(EA, 1) ← rS[24-31]
```

EA is the sum (**r**A|0) + d. The contents of the low-order eight bits of **r**S are stored into the byte in memory addressed by EA.

Other registers altered:

8

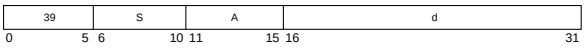
- None

PowerPC Architecture Level	Supervisor Level	PowerPC Optional	Form
UISA			D

stbu stbu

Store Byte with Update (x'9C00 0000')

stbu **r**S,d(**r**A)




```
EA ← (rA) + EXTS(d)
MEM(EA, 1) ← rS[24-31]
rA ← EA
```

EA is the sum (**r**A) + d. The contents of the low-order eight bits of **r**S are stored into the byte in memory addressed by EA.

EA is placed into **r**A.

If **r**A = 0, the instruction form is invalid.

8

Other registers altered:

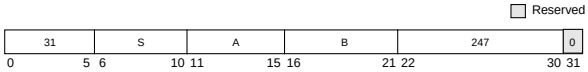
- None

PowerPC Architecture Level	Supervisor Level	PowerPC Optional	Form
UISA			D

stbux stbux

Store Byte with Update Indexed (x'7C00 01EE')

stbux **r**S,**r**A,**r**B




```
EA ← (rA) + (rB)
MEM(EA, 1) ← rS[24-31]
rA ← EA
```

EA is the sum (**r**A) + (**r**B). The contents of the low-order eight bits of **r**S are stored into the byte in memory addressed by EA.

EA is placed into **r**A.

If **r**A = 0, the instruction form is invalid.

Other registers altered:

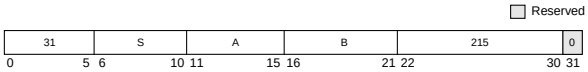
- None

PowerPC Architecture Level	Supervisor Level	PowerPC Optional	Form
UISA			X

stbx stbx

Store Byte Indexed (x'7C00 01AE')

stbx **r**S,**r**A,**r**B




```
if rA = 0
      then b ← 0
      else b ← (rA)
EA ← b + (rB)
MEM(EA, 1) ← rS[24-31]
```

EA is the sum (**r**A|0) + (**r**B). The contents of the low-order eight bits of **r**S are stored into the byte in memory addressed by EA.

Other registers altered:

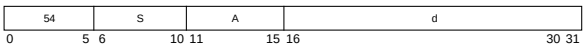
- None

PowerPC Architecture Level	Supervisor Level	PowerPC Optional	Form
UISA			X

stfd stfd

Store Floating-Point Double (x'D800 0000')

stfd **fr**S,d(**r**A)




```
if rA = 0
      then b ← 0
      else b ← (rA)
EA ← b + EXTS(d)
MEM(EA, 8) ← (frS)
```

EA is the sum (**r**A|0) + d.

The contents of register **fr**S are stored into the double word in memory addressed by EA.

Other registers altered:

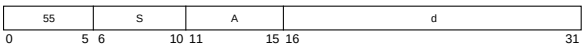
- None

PowerPC Architecture Level	Supervisor Level	PowerPC Optional	Form
UISA			D

stfdu stfdu

Store Floating-Point Double with Update (x'DC00 0000')

stfdu **fr**S,d(**r**A)




```
EA ← (rA) + EXTS(d)
MEM(EA, 8) ← (frS)
rA ← EA
```

EA is the sum (**r**A) + d.

The contents of register **fr**S are stored into the double word in memory addressed by EA.

EA is placed into **r**A.

If **r**A = 0, the instruction form is invalid.

Other registers altered:

- None

PowerPC Architecture Level	Supervisor Level	PowerPC Optional	Form
UISA			D

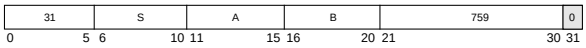
stfdux

stfdux

Store Floating-Point Double with Update Indexed (x'7C00 05EE')

stfdux **fr**S,**r**A,**r**B

Reserved




```
EA ← (rA) + (rB)
MEM(EA, 8) ← (frS)
rA ← EA
```

EA is the sum (**r**A) + (**r**B).

The contents of register **fr**S are stored into the double word in memory addressed by EA.

EA is placed into **r**A.

If **r**A = 0, the instruction form is invalid.

Other registers altered:

- None

PowerPC Architecture Level	Supervisor Level	PowerPC Optional	Form
UISA			X

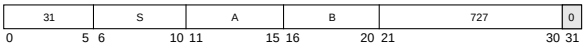
stfdx

stfdx

Store Floating-Point Double Indexed (x'7C00 05AE')

stfdx **fr**S,**r**A,**r**B

Reserved




```
if rA = 0
     then b ← 0
     else b ← (rA)
EA ← b + (rB)
MEM(EA, 8) ← (frS)
```

EA is the sum (**r**A|0) + **r**B.

The contents of register **fr**S are stored into the double word in memory addressed by EA.

Other registers altered:

- None

PowerPC Architecture Level	Supervisor Level	PowerPC Optional	Form
UISA			X

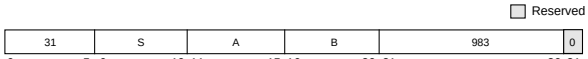
stfiwx

stfiwx

Store Floating-Point as Integer Word Indexed (x'7C00 07AE')

stfiwx **fr**S,**r**A,**r**B

Reserved




```
if rA = 0
     then b ← 0
     else b ← (rA)
EA ← b + (rB)
MEM(EA, 4) ← frS[32–63]
```

EA is the sum (**r**A|0) + (**r**B).

The contents of the low-order 32 bits of register **fr**S are stored, without conversion, into the word in memory addressed by EA.

This instruction when preceded by the floating-point convert to integer word (**fctiwx**) or floating-point convert to integer word with round toward zero (**fctiwzx**) will store the 32-bit integer value of a double-precision floating-point number. (see **fctiwx** and **fctiwzx** instructions)

If the content of register **fr**S is a double-precision floating point number, the low-order 32 bits of the 52 bit mantissa are stored. (without the exponent, this could be a meaningless value)

If the contents of register **fr**S were produced, either directly or indirectly, by an **lfs** instruction, a single-precision arithmetic instruction, or **frsp**, then the value stored is the low-order 32 bits of the 52 bit mantissa of the double-precision number. (all single-precision floating-point numbers are maintained in double precision format in the floating-point register file)

Other registers altered:

- None

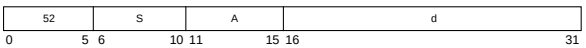
PowerPC Architecture Level	Supervisor Level	PowerPC Optional	Form
UISA		YES	X

stfs

stfs

Store Floating-Point Single (x'D000 0000')

stfs **fr**S,d(**r**A)




```
if rA = 0
     then b ← 0
     else b ← (rA)
EA ← b + EXTS(d)
MEM(EA, 4) ← SINGLE(frS)
```

EA is the sum (**r**A|0) + d.

The contents of register **fr**S are converted to single-precision and stored into the word in memory addressed by EA. For a discussion on floating-point store conversions, see Section D.7, “Floating-Point Store Instructions.”

Other registers altered:

- None

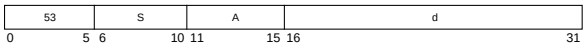
PowerPC Architecture Level	Supervisor Level	PowerPC Optional	Form
UISA			D

stfsu

stfsu

Store Floating-Point Single with Update (x'D400 0000')

stfsu **fr**S,d(**r**A)




```
EA ← (rA) + EXTS(d)
MEM(EA, 4) ← SINGLE(frS)
rA ← EA
```

EA is the sum (**r**A) + d.

The contents of **fr**S are converted to single-precision and stored into the word in memory addressed by EA. For a discussion on floating-point store conversions, see Section D.7, “Floating-Point Store Instructions.”

EA is placed into **r**A.

If **r**A = 0, the instruction form is invalid.

Other registers altered:

- None

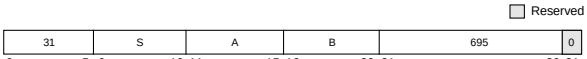
PowerPC Architecture Level	Supervisor Level	PowerPC Optional	Form
UISA			D

stfsux

stfsux

Store Floating-Point Single with Update Indexed (x'7C00 056E')

stfsux **fr**S,**r**A,**r**B




```
EA ← (rA) + (rB)
MEM(EA, 4) ← SINGLE(frS)
rA ← EA
```

EA is the sum (**r**A) + (**r**B).

The contents of **fr**S are converted to single-precision and stored into the word in memory addressed by EA. For a discussion on floating-point store conversions, see Section D.7, “Floating-Point Store Instructions.”

EA is placed into **r**A.

If **r**A = 0, the instruction form is invalid.

Other registers altered:

- None

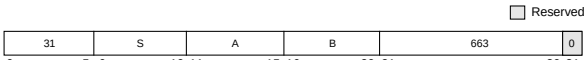
PowerPC Architecture Level	Supervisor Level	PowerPC Optional	Form
UISA			X

stfsx

stfsx

Store Floating-Point Single Indexed (x'7C00 052E')

stfsx **fr**S,**r**A,**r**B




```
if rA = 0
      then b ← 0
      else b ← (rA)
EA ← b + (rB)
MEM(EA, 4) ← SINGLE(frS)
```

EA is the sum (**r**A|0) + (**r**B).

The contents of register **fr**S are converted to single-precision and stored into the word in memory addressed by EA. For a discussion on floating-point store conversions, see Section D.7, “Floating-Point Store Instructions.”

Other registers altered:

- None

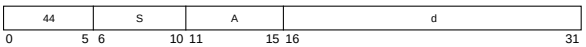
PowerPC Architecture Level	Supervisor Level	PowerPC Optional	Form
UISA			X

sth

sth

Store Half Word (x'B000 0000')

sth **r**S,d(**r**A)




```
if rA = 0
      then b ← 0
      else b ← (rA)
EA ← b + EXTS(d)
MEM(EA, 2) ← rS[16-31]
```

EA is the sum (**r**A|0) + d. The contents of the low-order 16 bits of **r**S are stored into the half word in memory addressed by EA.

Other registers altered:

- None

PowerPC Architecture Level	Supervisor Level	PowerPC Optional	Form
UISA			D

sthbrx sthbrx

Store Half Word Byte-Reverse Indexed (x'7C00 072C')

sthbrx **r**S,**r**A,**r**B


```
if rA = 0
     then b ← 0
     else b ← (rA)
EA ← b + (rB)
MEM(EA, 2) ← rS[24-31] || rS[16-23]
```

EA is the sum (**r**A|0) + (**r**B). The contents of the low-order eight bits (24-31) of **r**S are stored into bits 0–7 of the half word in memory addressed by EA. The contents of the subsequent low-order eight bits (16-23) of **r**S are stored into bits 8–15 of the half word in memory addressed by EA.

Other registers altered:

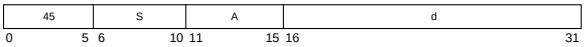
- None

PowerPC Architecture Level	Supervisor Level	PowerPC Optional	Form
UISA			X

sthu sthu

Store Half Word with Update (x'B400 0000')

sthu **r**S,d(**r**A)




```
EA ← (rA) + EXTS(d)
MEM(EA, 2) ← rS[16-31]
rA ← EA
```

EA is the sum (**r**A) + d. The contents of the low-order 16 bits of **r**S are stored into the half word in memory addressed by EA.

EA is placed into **r**A.

If **r**A = 0, the instruction form is invalid.

Other registers altered:

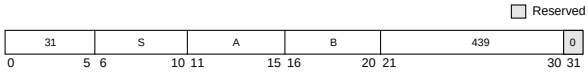
- None

PowerPC Architecture Level	Supervisor Level	PowerPC Optional	Form
UISA			D

sthux sthux

Store Half Word with Update Indexed (x'7C00 036E')

sthux **r**S,**r**A,**r**B




```
EA ← (rA) + (rB)
MEM(EA, 2) ← rS[16-31]
rA ← EA
```

EA is the sum (**r**A) + (**r**B). The contents of the low-order 16 bits of **r**S are stored into the half word in memory addressed by EA.

EA is placed into **r**A.

If **r**A = 0, the instruction form is invalid.

Other registers altered:

- None

PowerPC Architecture Level	Supervisor Level	PowerPC Optional	Form
UISA			X

sthx sthx

Store Half Word Indexed (x'7C00 032E')

sthx **r**S,**r**A,**r**B


```
if rA = 0
     then b ← 0
     else b ← (rA)
EA ← b + (rB)
MEM(EA, 2) ← rS[16-31]
```

EA is the sum (**r**A|0) + (**r**B). The contents of the low-order 16 bits of **r**S are stored into the half word in memory addressed by EA.

Other registers altered:

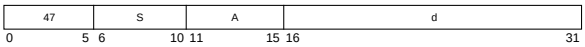
- None

PowerPC Architecture Level	Supervisor Level	PowerPC Optional	Form
UISA			X

stmw stmw

Store Multiple Word (x'BC00 0000')

stmw **r**S,d(**r**A)




```
if rA = 0
      then b ← 0
      else b ← (rA)
EA ← b + EXTS(d)
r ← rS
do while r ≤ 31
          MEM(EA, 4) ← GPR(r)
          r ← r + 1
          EA ← EA + 4
```

EA is the sum (**r**A|0) + d.

n = (32 – **r**S).

n consecutive words starting at EA are stored from the GPRs **r**S through **r31**. For example, if **r**S = 30, 2 words are stored.

EA must be a multiple of four. If it is not, either the system alignment exception handler is invoked or the results are boundedly undefined. For additional information about alignment and DSI exceptions, see Section 6.4.3, "DSI Exception (0x00300)."

NOTE: In some implementations, this instruction is likely to have a greater latency and take longer to execute, perhaps much longer, than a sequence of individual store instructions that produce the same results.

Other registers altered:

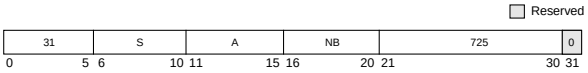
- None

PowerPC Architecture Level	Supervisor Level	PowerPC Optional	Form
UISA			D

stswi stswi

Store String Word Immediate (x'7C00 05AA')

stswi **r**S,**r**A,NB




```
if rA = 0
      then EA ← 0
      else EA ← (rA)
if NB = 0
      then n ← 32
      else n ← NB
r ← rS – 1
i ← 0
do while n > 0
          if i = 0
               then r ← r + 1 (mod 32)
          MEM(EA, 1) ← GPR(r)[i, i+7]
          i ← i + 8
          if i = 32
               then i ← 0
          EA ← EA + 1
          n ← n – 1
```

EA is (**r**A|0). Let *n* = NB if NB ≠ 0, *n* = 32 if NB = 0; *n* is the number of bytes to store. Let *nr* = CEIL(*n* / 4);*nr* is the number of registers to supply data.

n consecutive bytes starting at EA are stored from GPRs **r**S through **r**S + *nr* – 1. Bytes are stored left to right from each register. The sequence of registers wraps around through **r0** if required.

Under certain conditions (for example, segment boundary crossing) the data alignment exception handler may be invoked. For additional information about data alignment exceptions, see Section 6.4.3, "DSI Exception (0x00300)."

NOTE: In some implementations, this instruction is likely to have a greater latency and take longer to execute, perhaps much longer, than a sequence of individual store instructions that produce the same results.

Other registers altered:

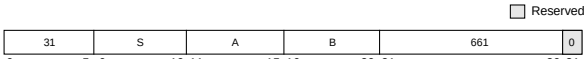
- None

PowerPC Architecture Level	Supervisor Level	PowerPC Optional	Form
UISA			X

stswx stswx

Store String Word Indexed (x'7C00 052A')

stswx **r**S,**r**A,**r**B




```
if rA = 0
      then b ← 0
      else b ← (rA)
EA ← b + (rB)
n ← XER[25–31]
r ← rS – 1
i ← 0
do while n > 0
          if i = 0
               then r ← r + 1 (mod 32)
          MEM(EA, 1) ← GPR(r)[i, i+7]
          i ← i + 8
          if i = 32
               then i ← 0
          EA ← EA + 1
          n ← n – 1
```

EA is the sum (**r**A|0) + (**r**B). Let *n* = XER[25–31]; *n* is the number of bytes to store. Let *nr* = CEIL(*n* / 4);*nr* is the number of registers to supply data.

n consecutive bytes starting at EA are stored from GPRs **r**S through **r**S + *nr* – 1. Bytes are stored left to right from each register. The sequence of registers wraps around through **r0** if required. If *n* = 0, no bytes are stored.

Under certain conditions (for example, segment boundary crossing) the data alignment exception handler may be invoked. For additional information about data alignment exceptions, see Section 6.4.3, "DSI Exception (0x00300)."

NOTE: In some implementations, this instruction is likely to have a greater latency and take longer to execute, perhaps much longer, than a sequence of individual store instructions that produce the same results.

Other registers altered:

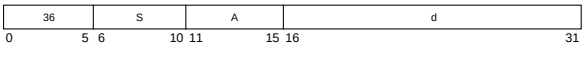
- None

PowerPC Architecture Level	Supervisor Level	PowerPC Optional	Form
UISA			X

stw stw

Store Word (x'9000 0000')

stw **r**S,d(**r**A)




```
if rA = 0
      then b ← 0
      else b ← (rA)
EA ← b + EXTS(d)
MEM(EA, 4) ← rS
```

EA is the sum (**r**A|0) + d. The contents of **r**S are stored into the word in memory addressed by EA.

Other registers altered:

- None

PowerPC Architecture Level	Supervisor Level	PowerPC Optional	Form
UISA			D

stwbrx

Store Word Byte-Reverse Indexed (x'7C00 052C')

stwbrx **r**S,**r**A,**r**B


```
if rA = 0
     then b ← 0
     else b ← (rA)
EA ← b + (rB)
MEM(EA, 4) ← rS[24-31] || rS[16-23] || rS[8-15] || rS[0-7]
```

EA is the sum (**r**A|0) + (**r**B). The contents of the low-order eight bits (24-31) of **r**S are stored into bits 0–7 of the word in memory addressed by EA. The contents of the subsequent eight low-order bits (16-23) of **r**S are stored into bits 8–15 of the word in memory addressed by EA. The contents of the subsequent eight low-order bits (8-15) of **r**S are stored into bits 16–23 of the word in memory addressed by EA. The contents of the subsequent eight low-order bits (0-7) of **r**S are stored into bits 24–31 of the word in memory addressed by EA.

Other registers altered:

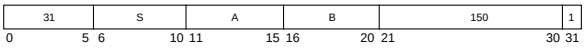
- None

PowerPC Architecture Level	Supervisor Level	PowerPC Optional	Form
UISA			X

stwcx.

Store Word Conditional Indexed (x'7C00 012D')

stwcx. **r**S,**r**A,**r**B




```
if rA = 0
     then b ← 0
     else b ← (rA)
EA ← b + (rB)
if RESERVE
          then
               MEM(EA, 4) ← (rS)
               CR0 ← 0b00 || 0b1 || XER[SO]
               RESERVE ← 0
          else
               CR0 ← 0b00 || 0b0 || XER[SO]
```

EA is the sum (**r**A|0) + (**r**B). If the reserved bit is set, the **stwcx.** instruction stores **r**S to effective address (**r**A + **r**B), clears the reserved bit, and sets CR0[EQ]. If the reserved bit is not set, the **stwcx.** instruction does not do a store; it leaves the reserved bit cleared and clears CR0[EQ]. Software must look at CR0[EQ] to see if the **stwcx.** was successful.

The reserved bit is set by the **lwarx** instruction. The reserved bit is cleared by any **stwcx.** instruction to any address, and also by snooping logic if it detects that another processor does any kind of write or invalidate to the block indicated in the reservation buffer when reserved is set.

EA must be a multiple of four. If it is not, either the system alignment exception handler is invoked or the results are boundedly undefined. For additional information about alignment and DSI exceptions, see Section 6.4.3, "DSI Exception (0x00300)."

The granularity with which reservations are managed is implementation-dependent. Therefore, the memory to be accessed by the load and reserve and store conditional instructions should be controlled by a system library program.

Because the hardware doesn't compare reservation address when executing the stwcx. instruction, operating systems software MUST reset the reservation if an exception or other type of interrupt occurs to insure atomic memory references of **lwarx** and **stwcx. p**airs.

Other registers altered:

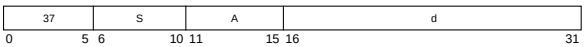
- CR0 field is set to reflect whether the store operation was performed as follows:
  CR0[LT GT EQ S0] = 0b00 || store_performed || XER[SO]
- Condition Register (CR0 field):
  Affected: LT, GT, EQ, SO

PowerPC Architecture Level	Supervisor Level	PowerPC Optional	Form
UISA			X

stwu

Store Word with Update (x'9400 0000')

stwu **r**S,d(**r**A)




```
EA ← (rA) + EXTS(d)
MEM(EA, 4) ← (rS)
rA ← EA
```

EA is the sum (**r**A) + d. The contents of **r**S are stored into the word in memory addressed by EA.

EA is placed into **r**A.

If **r**A = 0, the instruction form is invalid.

Other registers altered:

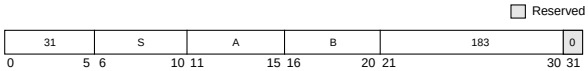
- None

PowerPC Architecture Level	Supervisor Level	PowerPC Optional	Form
UISA			D

stwux

Store Word with Update Indexed (x'7C00 016E')

stwux **r**S,**r**A,**r**B




```
EA ← (rA) + (rB)
MEM(EA, 4) ←(rS)
rA ← EA
```

EA is the sum (**r**A) + (**r**B). The contents of **r**S are stored into the word in memory addressed by EA.

EA is placed into **r**A.

If **r**A = 0, the instruction form is invalid.

Other registers altered:

- None

PowerPC Architecture Level	Supervisor Level	PowerPC Optional	Form
UISA			X

stwx

Store Word Indexed (x'7C00 012E')

stwx **r**S,**r**A,**r**B


```
if rA = 0
     then b ← 0
     else b ← (rA)
EA ← b + (rB)
MEM(EA, 4) ←(rS)
```

EA is the sum (**r**A|0) + (**r**B). The contents of **r**S are stored into the word in memory addressed by EA.

Other registers altered:

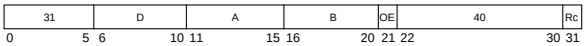
- None

PowerPC Architecture Level	Supervisor Level	PowerPC Optional	Form
UISA			X

subf*x*

Subtract From (x'7C00 0050')

subf	**r**D,**r**A,**r**B	(OE = 0 Rc = 0)
subf.	**r**D,**r**A,**r**B	(OE = 0 Rc = 1)
subfo	**r**D,**r**A,**r**B	(OE = 1 Rc = 0)
subfo.	**r**D,**r**A,**r**B	(OE = 1 Rc = 1)




```
rD ← ¬(rA) + (rB) + 1
```

The sum ¬ (**r**A) + (**r**B) + 1 is placed into **r**D. (equivlent to (rB)--(**r**A))

The **subf** instruction is preferred for subtraction because it sets few status bits.

Other registers altered:

- Condition Register (CR0 field):
  Affected: LT, GT, EQ, SO (if Rc = 1)
- XER:
  Affected: SO, OV (if OE = 1)

Simplified mnemonics:

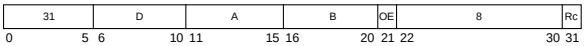
sub **r**D,**r**A,**r**B equivalent to **subf** **r**D,**r**B,**r**A

PowerPC Architecture Level	Supervisor Level	PowerPC Optional	Form
UISA			XO

subfc*x*

Subtract from Carrying (x'7C00 0010')

subfc	**r**D,**r**A,**r**B	(OE = 0 Rc = 0)
subfc.	**r**D,**r**A,**r**B	(OE = 0 Rc = 1)
subfco	**r**D,**r**A,**r**B	(OE = 1 Rc = 0)
subfco.	**r**D,**r**A,**r**B	(OE = 1 Rc = 1)




```
rD ← ¬(rA) + (rB) + 1
```

The sum ¬ (**r**A) + (**r**B) + 1 is placed into **r**D. (equivlent to (rB)--(**r**A))

Other registers altered:

- Condition Register (CR0 field):
  Affected: LT, GT, EQ, SO (if Rc = 1)
  Note: CR0 field may not reflect the infinitely precise result if overflow occurs (see XER below).
- XER:
  Affected: CA
  Affected: SO, OV (if OE = 1)
  Note: The setting of the affected bits in the XER reflects overflow of the 32-bit results. For further information see Chapter 3, “Operand Conventions.”

Simplified mnemonics:

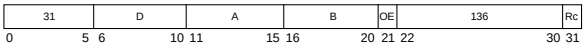
subc **r**D,**r**A,**r**B equivalent to **subfc** **r**D,**r**B,**r**A

PowerPC Architecture Level	Supervisor Level	PowerPC Optional	Form
UISA			XO

subfe*x* subfe*x*

Subtract from Extended (x'7C00 0110')

subfe **r**D,**r**A,**r**B (OE = 0 Rc = 0)
subfe. **r**D,**r**A,**r**B (OE = 0 Rc = 1)
subfeo **r**D,**r**A,**r**B (OE = 1 Rc = 0)
subfeo. **r**D,**r**A,**r**B (OE = 1 Rc = 1)




```
rD ← ¬ (rA) + (rB) + XER[CA]
```

The sum ¬ (**r**A) + (**r**B) + XER[CA] is placed into **r**D.

Other registers altered:

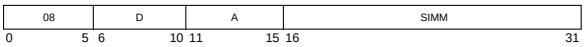
- Condition Register (CR0 field):
  Affected: LT, GT, EQ, SO (if Rc = 1)
  Note: CR0 field may not reflect the infinitely precise result if overflow occurs (**Note:** See Chapter 3, "Operand Conventions" for setting of affected bits).
- XER:
  Affected: CA
  Affected: SO, OV (if OE = 1)

PowerPC Architecture Level	Supervisor Level	PowerPC Optional	Form
UISA			XO

subfic subfic

Subtract from Immediate Carrying (x'2000 0000')

subfic **r**D,**r**A,SIMM




```
rD ← ¬ (rA) + EXTS(SIMM) + 1
```

The sum ¬ (**r**A) + EXTS(SIMM) + 1 is placed into **r**D.(equivlent to EXTS(SIMM)-(**r**A))

Other registers altered:

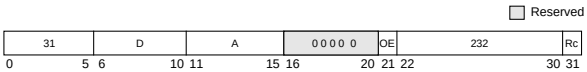
- XER:
  Affected: CA
  Note: See Chapter 3, "Operand Conventions."

PowerPC Architecture Level	Supervisor Level	PowerPC Optional	Form
UISA			D

subfme*x* subfme*x*

Subtract from Minus One Extended (x'7C00 01D0')

subfme **r**D,**r**A (OE = 0 Rc = 0)
subfme. **r**D,**r**A (OE = 0 Rc = 1)
subfmeo **r**D,**r**A (OE = 1 Rc = 0)
subfmeo. **r**D,**r**A (OE = 1 Rc = 1)




```
rD ← ¬ (rA) + XER[CA] – 1
```

The sum ¬ (**r**A) + XER[CA] + (32)1 is placed into **r**D.

Other registers altered:

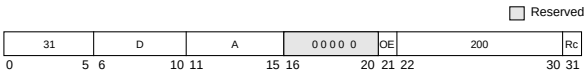
- Condition Register (CR0 field):
  Affected: LT, GT, EQ, SO (if Rc = 1)
  Note: CR0 field may not reflect the infinitely precise result if overflow occurs (See Chapter 3, "Operand Conventions."
- XER:
  Affected: CA
  Affected: SO, OV (if OE = 1)

PowerPC Architecture Level	Supervisor Level	PowerPC Optional	Form
UISA			XO

subfze*x* subfze*x*

Subtract from Zero Extended (x'7C00 0190')

subfze **r**D,**r**A (OE = 0 Rc = 0)
subfze. **r**D,**r**A (OE = 0 Rc = 1)
subfzeo **r**D,**r**A (OE = 1 Rc = 0)
subfzeo. **r**D,**r**A (OE = 1 Rc = 1)




```
rD ← ¬ (rA) + XER[CA]
```

The sum ¬ (**r**A) + XER[CA] is placed into **r**D.

Other registers altered:

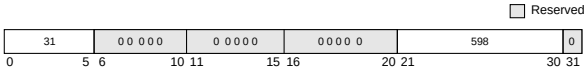
- Condition Register (CR0 field):
  Affected: LT, GT, EQ, SO (if Rc = 1)
  Note: CR0 field may not reflect the infinitely precise result if overflow occurs (see XER below).
- XER:
  Affected: CA
  Affected: SO, OV (if OE = 1)
  Note: See Chapter 3, "Operand Conventions."

PowerPC Architecture Level	Supervisor Level	PowerPC Optional	Form
UISA			XO

sync

sync

Synchronize (x'7C00 04AC')



The **sync** instruction provides an ordering function for the effects of all instructions executed by a given processor. Executing a **sync** instruction ensures that all instructions preceding the **sync** instruction appear to have completed before the **sync** instruction completes, and that no subsequent instructions are initiated by the processor until after the **sync** instruction completes. When the **sync** instruction completes, all external accesses caused by instructions preceding the **sync** instruction will have been performed with respect to all other mechanisms that access memory. For more information on how the **sync** instruction affects the VEA, refer to Chapter 5, "Cache Model and Memory Coherency."

Multiprocessor implementations also send a **sync** address-only broadcast that is useful in some designs. For example, if a design has an external buffer that re-orders loads and stores for better bus efficiency, the **sync** broadcast signals to that buffer that previous loads/stores must be completed before any following loads/stores.

8

The **sync** instruction can be used to ensure that the results of all stores into a data structure, caused by store instructions executed in a "critical section" of a program, are seen by other processors before the data structure is seen as unlocked.

The functions performed by the **sync** instruction will normally take a significant amount of time to complete, so indiscriminate use of this instruction may adversely affect performance. In addition, the time required to execute **sync** may vary from one execution to another.

The **eieio** instruction may be more appropriate than **sync** for many cases.

This instruction is execution synchronizing. For more information on execution synchronization, see Section 4.1.5, "Synchronizing Instructions."

Other registers altered:

- None

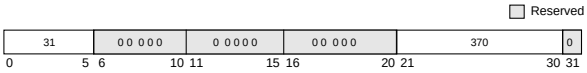
PowerPC Architecture Level	Supervisor Level	PowerPC Optional	Form
UISA			X

tlbia

tlbia

Translation Lookaside Buffer Invalidate All (x'7C00 02E4')

tlbia




```
All TLB entries ← invalid
```

The entire translation lookaside buffer (TLB) is invalidated (that is, all entries are removed).

The TLB is invalidated regardless of the settings of MSR[IR] and MSR[DR]. The invalidation is done without reference to the segment registers.

8

This instruction does not cause the entries to be invalidated in other processors.

This is a supervisor-level instruction and optional in the PowerPC architecture.

Other registers altered:

- None

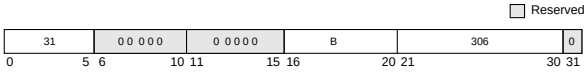
PowerPC Architecture Level	Supervisor Level	PowerPC Optional	Form
OEA	YES	YES	X

tlbie

tlbie

Translation Lookaside Buffer Invalidate Entry (x'7C00 0264')

tlbie **r**B




```
VPS ← rB[4-19]
Identify TLB entries corresponding to VPS
Each such TLB entry ← invalid
```

EA is the contents of **r**B. If the translation lookaside buffer (TLB) contains an entry corresponding to EA, that entry is made invalid (that is, removed from the TLB).

Multiprocessing implementations (for example, the 601, and 604) send a **tlbie** address-only broadcast over the address bus to tell other processors to invalidate the same TLB entry in their TLBs.

8

The TLB search is done regardless of the settings of MSR[IR] and MSR[DR]. The search is done based on a portion of the logical page number within a segment, without reference to the segment registers. All entries matching the search criteria are invalidated.

Block address translation for EA, if any, is ignored. Refer to Section 7.5.3.4, "Synchronization of Memory Accesses and Referenced and Changed Bit Updates," and Section 7.6.3, "Page Table Updates," for other requirements associated with the use of this instruction.

This is a supervisor-level instruction and optional in the PowerPC architecture.

Other registers altered:

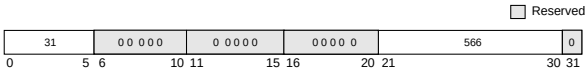
- None

PowerPC Architecture Level	Supervisor Level	PowerPC Optional	Form
OEA	YES	YES	X

tlbsync

tlbsync

TLB Synchronize (x'7C00 046C')



If an implementation sends a broadcast for **tlbie** then it will also send a broadcast for **tlbsync**. Executing a **tlbsync** instruction ensures that all **tlbie** instructions previously executed by the processor executing the **tlbsync** instruction have completed on all other processors.

The operation performed by this instruction is treated as a caching-inhibited and guarded data access with respect to the ordering done by **eieio**.

NOTE: The 601 expands the use of the **sync** instruction to cover **tlbsync** functionality.

8

Refer to Section 7.5.3.4, "Synchronization of Memory Accesses and Referenced and Changed Bit Updates," and Section 7.6.3, "Page Table Updates," for other requirements associated with the use of this instruction.

This instruction is supervisor-level and optional in the PowerPC architecture.

Other registers altered:

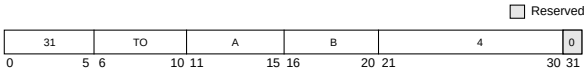
- None

PowerPC Architecture Level	Supervisor Level	PowerPC Optional	Form
OEA	YES	YES	X

tw

Trap Word (x'7C00 0008')

tw TO,**r**A,**r**B




```
a ← EXTS(rA)
b ← EXTS(rB)
if (a < b) & TO[0] then TRAP
if (a > b) & TO[1] then TRAP
if (a = b) & TO[2] then TRAP
if (a <U b) & TO[3] then TRAP
if (a >U b) & TO[4] then TRAP
```

The contents of **r**A are compared arithmetically with the contents of **r**B for TO[0, 1, 2]. The contents of **r**A are compared logically with the contents of **r**B for TO[3, 4]. If any bit in the TO field is set and its corresponding condition is met by the result of the comparison, then the system trap handler is invoked.

Other registers altered:

- None

Simplified mnemonics:

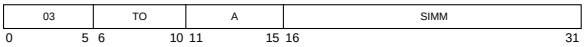
tweq	**r**A,**r**B	equivalent to	**tw**	**4,r**A,**r**B
twlge	**r**A,**r**B	equivalent to	**tw**	**5,r**A,**r**B
trap		equivalent to	**tw**	**31,0,0**

PowerPC Architecture Level	Supervisor Level	PowerPC Optional	Form
UISA			X

twi

Trap Word Immediate (x'0C00 0000')

twi TO,**r**A,SIMM




```
a ← EXTS(rA)
if (a < EXTS(SIMM)) & TO[0] then TRAP
if (a > EXTS(SIMM)) & TO[1] then TRAP
if (a = EXTS(SIMM)) & TO[2] then TRAP
if (a <U EXTS(SIMM)) & TO[3] then TRAP
if (a >U EXTS(SIMM)) & TO[4] then TRAP
```

The contents of **r**A are compared arithmetically with the sign-extended value of the SIMM field for TO[0, 1, 2]. The contents of **r**A are compared logically with the sign-extended value of the SIMM field for TO[3, 4]. If any bit in the TO field is set and its corresponding condition is met by the result of the comparison, then the system trap handler is invoked.

Other registers altered:

- None

Simplified mnemonics:

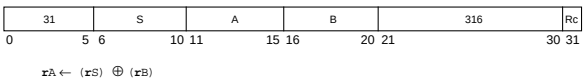
twgti	**r**A,value	equivalent to	**twi**	**8,r**A,value
twllei	**r**A,value	equivalent to	**twi**	**6,r**A,value

PowerPC Architecture Level	Supervisor Level	PowerPC Optional	Form
UISA			D

xor*x*

XOR (x'7C00 0278')

xor	**r**A,**r**S,**r**B	(Rc = 0)
xor.	**r**A,**r**S,**r**B	(Rc = 1)




```
rA ← (rS) ⊕ (rB)
```

The contents of **r**S are XORed with the contents of **r**B and the result is placed into **r**A.

Other registers altered:

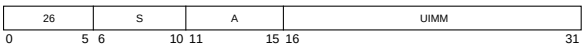
- Condition Register (CR0 field):
  Affected: LT, GT, EQ, SO (if Rc = 1)

PowerPC Architecture Level	Supervisor Level	PowerPC Optional	Form
UISA			X

xori

XOR Immediate (x'6800 0000')

xori **r**A,**r**S,UIMM




```
rA ← (rS) ⊕ ((16)0 || UIMM)
```

The contents of **r**S are XORed with 0x0000 || UIMM and the result is placed into **r**A.

Other registers altered:

- None

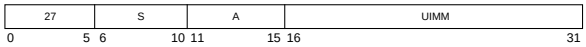
PowerPC Architecture Level	Supervisor Level	PowerPC Optional	Form
UISA			D

xoris

xoris

XOR Immediate Shifted (x'6C00 0000')

xoris **r**A,**r**S,UIMM




```
rA ← (rS) ⊕ (UIMM || (16)0)
```

The contents of **r**S are XORed with UIMM || 0x0000 and the result is placed into **r**A.

Other registers altered:

- None

PowerPC Architecture Level	Supervisor Level	PowerPC Optional	Form
UISA			D

8

This page deliberately left blank.