MPCxxx Instruction Set

This chapter lists the MPCxxx instruction set in alphabetical order by *mnemonic*. Note that 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. The format diagrams show, horizontally, all valid combinations of instruction fields.

Note that the *architecture* specification refers to user-level and supervisor-level as problem state and privileged state, respectively.

Instruction Formats

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.

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 1.

Table 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.
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 2.

Instruction Fields

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

Table 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). 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.
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.
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.
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 16-bit signed two's complement integer that is sign-extended to 32 bits.
frC (21–25)	NOT USED BY MPCxxx.
frD (6–10)	NOT USED BY MPCxxx.
frS (6–10)	NOT USED BY MPCxxx.
IMM (16–19)	NOT USED BY MPCxxx.
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.
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

Table 2. Instruction Syntax Conventions (Continued)

Field	Description
r*A* (11–15)	This field is used to specify a GPR to be used as a source or destination.
r*B* (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. (Note that exceptions are referred to as interrupts in the architecture specification.)
r*D* (6–10)	This field is used to specify a GPR to be used as a destination.
r*S* (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.
TO (6–10)	This field is used to specify the conditions on which to trap.
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.

Notation and Conventions

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

Table 3. Notation and Conventions

Notation/Convention	Meaning
←	Assignment
←$_{iea}$	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.

Table 3. Notation and Conventions (Continued)

Notation/Convention	Meaning
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.
<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)
$\oplus$, $\equiv$	Exclusive-OR, Equivalence logical operators (for example, $(a \equiv b) = (a \oplus \neg b)$)
0b*nnnn*	A number expressed in binary format.
0x*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 $\geq$ 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.

Table 3. Notation and Conventions (Continued)

Notation/Convention	Meaning
Do	Do loop. • Indenting shows range. • “To” and/or “by” clauses specify incrementing an iteration variable. • “While” clauses give termination conditions.
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.
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 that 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.
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.
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 4 describes instruction field notation conventions used throughout this document.

Table 4. Instruction Field Conventions

The Architecture Specification	Equivalent to:
BA, BB, BT	**crb**A, **crb**B, **crb**D (respectively)
D	d
DS	ds
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 5.

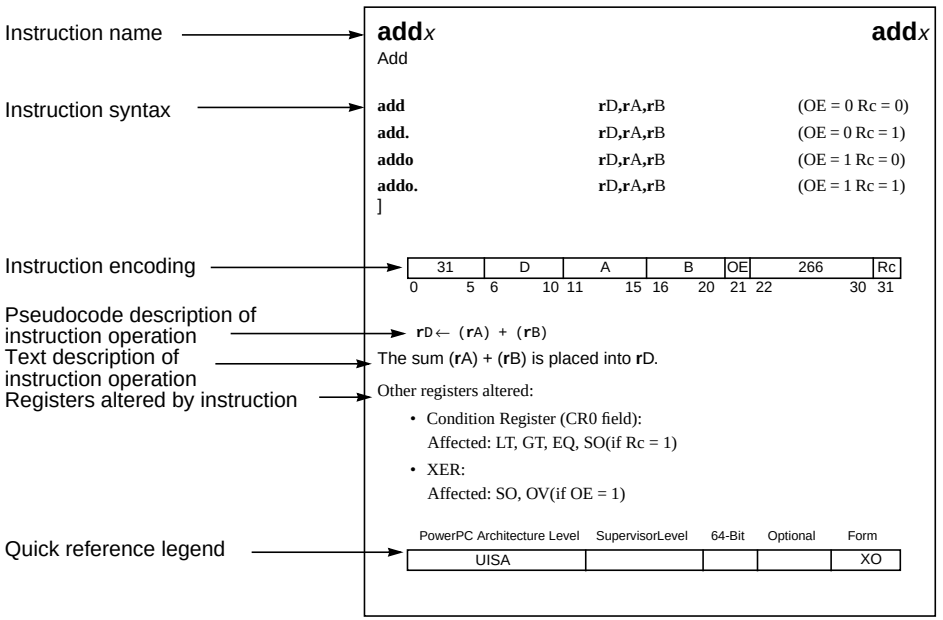
Table 5. Precedence Rules

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
←, $\leftarrow_{iea}$	None

Operators higher in Table 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 5, or to increase clarity; parenthesized expressions are evaluated before serving as operands.

MPCxxx Instruction Set

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

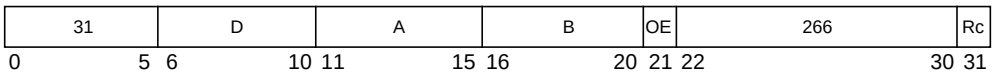


Instruction Description

Note that the execution unit that executes the instruction may not be the same for all PowerPC processors.

Add

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



rD ← (**r**A) + (**r**B)

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)

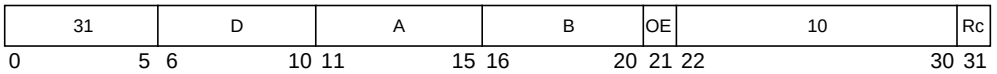
  Note: CR0 field may not reflect the “true” (infinitely precise) result if overflow occurs (see XER below).
- XER:

  Affected: SO, OV (if OE = 1)

PowerPC Architecture Level	Supervisor Level	Optional	Form
UISA			XO

Add Carrying

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 = 1)
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 “true” (infinitely precise) result if overflow occurs (see XER below).

- XER:

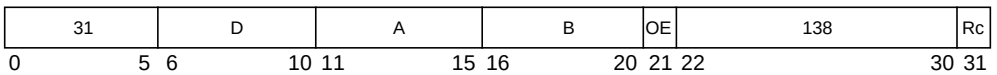
  Affected: CA

  Affected: SO, OV (if OE = 1)

PowerPC Architecture Level	Supervisor Level	Optional	Form
UISA			XO

Add Extended

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 “true” (infinitely precise) result if overflow occurs (see XER below).

- XER:

  Affected: CA

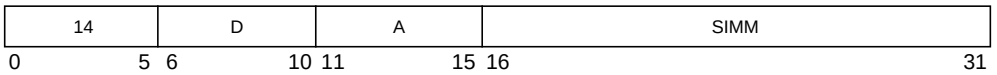
  Affected: SO, OV (if OE = 1)

PowerPC Architecture Level	Supervisor Level	Optional	Form
UISA			XO

addi addi

Add Immediate

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




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

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

The **addi** instruction is preferred for addition because it sets few status bits. Note that **addi** uses the value 0, not the contents of GPR0, if **r**A = 0.

Other registers altered:

- None

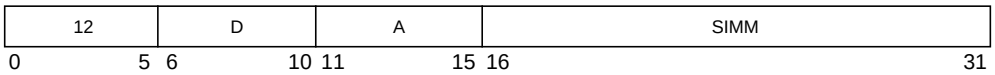
Simplified mnemonics:

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	Optional	Form
UISA			D

Add Immediate Carrying

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




```
rD ¨ (rA) + EXTS(SIMM)
```

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

Other registers altered:

- XER:
  Affected: CA

Simplified mnemonics:

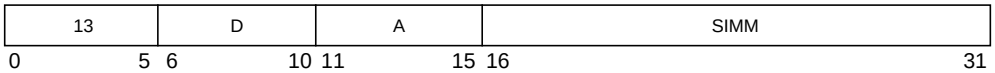
subic **r**D**,r**A**,**value equivalent to **addic** **r**D**,r**A**,**–value

PowerPC Architecture Level	Supervisor Level	Optional	Form
UISA			D

addic. addic.

Add Immediate Carrying and Record

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




```
rD ¨ (rA) + EXTS(SIMM)
```

The sum (**r**A) + 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 “true” (infinitely precise) result if overflow occurs (see XER below).

- XER:

  Affected: CA

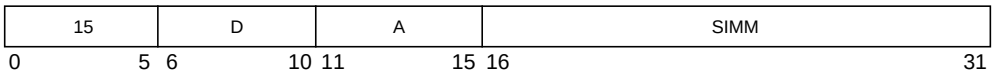
Simplified mnemonics:

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

PowerPC Architecture Level	Supervisor Level	Optional	Form
UISA			D

Add Immediate Shifted

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




```
if rA = 0 then rD ¨ EXTS(SIMM || (16)0)
else   rD ¨ (rA) + EXTS(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 that **addis** uses the value 0, not the contents of GPR0, if **r**A = 0.

Other registers altered:

- None

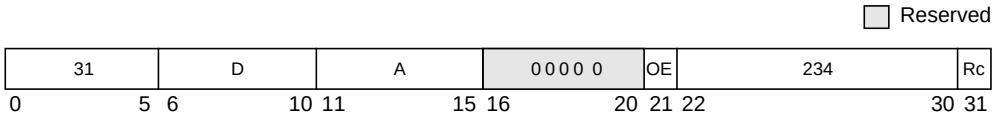
Simplified mnemonics:

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	Optional	Form
UISA			D

Add to Minus One Extended

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_FFFF_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 “true” (infinitely precise) result if overflow occurs (see XER below).

- XER:

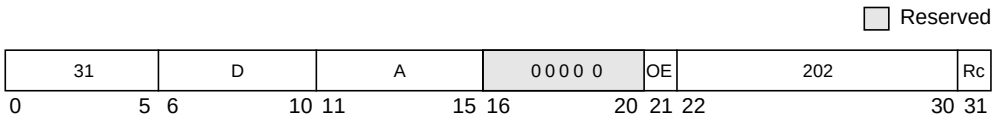
  Affected: CA

  Affected: SO, OV (if OE = 1)

PowerPC Architecture Level	Supervisor Level	Optional	Form
UISA			XO

Add to Zero Extended

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)




```
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 “true” (infinitely precise) result if overflow occurs (see XER below).

- XER:

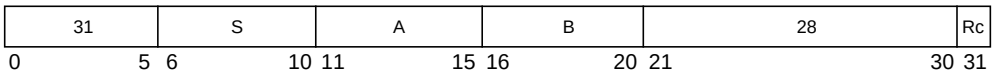
  Affected: CA

  Affected: SO, OV (if OE = 1)

PowerPC Architecture Level	Supervisor Level	Optional	Form
UISA			XO

AND

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



rA ← (**r**S) & (**r**B)

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

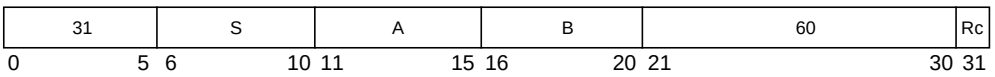
Other registers altered:

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

PowerPC Architecture Level	Supervisor Level	Optional	Form
UISA			X

AND with Complement

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



rA ← (**r**S) + ¬ (**r**B)

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):

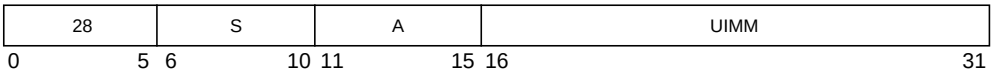
  Affected: LT, GT, EQ, SO (if Rc = 1)

PowerPC Architecture Level	Supervisor Level	Optional	Form
UISA			X

andi. andi.

AND Immediate

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




```
rA← (rS) & ((16)0 || UIMM)
```

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

Other registers altered:

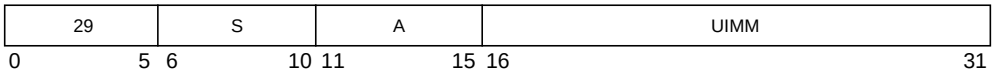
- Condition Register (CR0 field):
  Affected: LT, GT, EQ, SO

PowerPC Architecture Level	Supervisor Level	Optional	Form
UISA			D

andis. andis.

AND Immediate Shifted

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



rA← (**r**S) + (UIMM || (16)0)

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

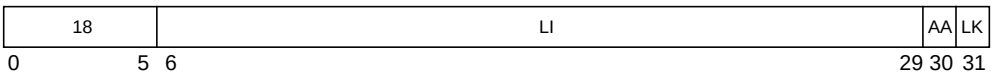
Other registers altered:

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

PowerPC Architecture Level	Supervisor Level	Optional	Form
UISA			D

Branch

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

target_addr specifies the branch target address.

If AA = 0, then the branch target address is the sum of LI || 0b00 sign-extended and the address of this instruction. If AA = 1, then the branch target address is the value LI || 0b00 sign-extended. If LK = 1, then the effective address of the instruction following the branch instruction is placed into the link register.

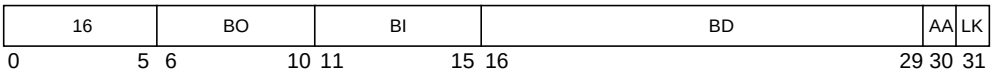
Other registers altered:

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

PowerPC Architecture Level	Supervisor Level	Optional	Form
UISA			I

Branch Conditional

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)




```
m ← 32
if ¬ BO[2] then CTR ← CTR – 1
ctr_ok ← BO[2] | (BO[3])
cond_ok ←BO[0] | (CR[BI] ≡ BO[1])
if ctr_ok & cond_ok then
 if AA then NIA ←iea EXTS(BD || 0b00)
 else NIA ←iea CIA + EXTS(BD || 0b00)
 if LK 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 6.

Table 6. BO Operand Encodings

BO	Description
0000*y*	Decrement the count register (CTR), then branch if the condition is FALSE.
0001*y*	Decrement the CTR, then branch if the condition is FALSE.
001*zy*	Branch if the condition is FALSE.
0100*y*	Decrement the CTR, then branch if the condition is TRUE.
0101*y*	Decrement the CTR, then branch if 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 that the *z* bits should be cleared, as they may be assigned a meaning in some future version of the MPCxxx.

The *y* bit provides a hint about whether a conditional branch is likely to be taken.

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)

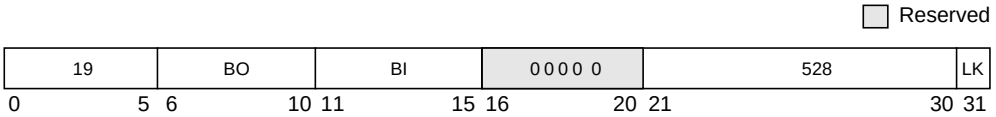
Simplified mnemonics:

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	Optional	Form
UISA			B

Branch Conditional to Count Register

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 || 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 7.

Table 7. BO Operand Encodings

BO	Description
0000*y*	Decrement the count register (CTR), then branch if the condition is FALSE.
0001*y*	Decrement the CTR, then branch if the condition is FALSE.
001*zy*	Branch if the condition is FALSE.
0100*y*	Decrement the CTR, then branch if the condition is TRUE.
0101*y*	Decrement the CTR, then branch if 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 that the *z* bits should be cleared, as they may be assigned a meaning in some future version of the MPCxxx.

The *y* bit provides a hint about whether a conditional branch is likely to be taken.

The branch target address is CTR || 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)

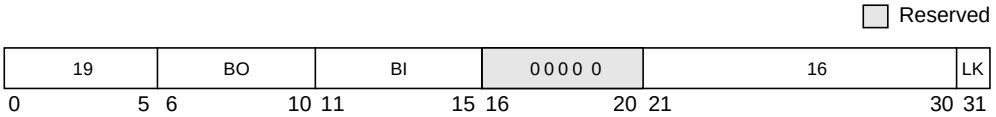
Simplified mnemonics:

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

PowerPC Architecture Level	Supervisor Level	Optional	Form
UISA			XL

Branch Conditional to Link Register

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




```
m ← 32
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 || 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.

Table 8. BO Operand Encodings

BO	Description
0000*y*	Decrement the CTR, then branch if the condition is FALSE.
0001*y*	Decrement the CTR, then branch if the condition is FALSE.
001*zy*	Branch if the condition is FALSE.
0100*y*	Decrement the CTR, then branch if the condition is TRUE.
0101*y*	Decrement the CTR, then branch if 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 that the *z* bits should be cleared, as they may be assigned a meaning in some future version of the MPCxxx.

The *y* bit provides a hint about whether a conditional branch is likely to be taken.

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)

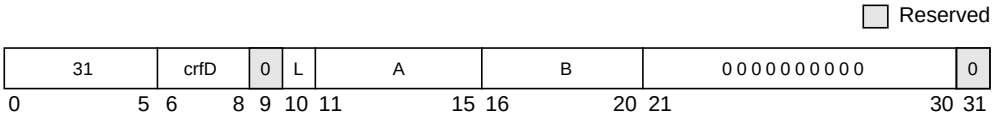
Simplified mnemonics:

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

PowerPC Architecture Level	Supervisor Level	Optional	Form
UISA			XL

Compare

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




```
a ← EXTS(rA)
b ← EXTS(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.

Other registers altered:

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

  Affected: LT, GT, EQ, SO

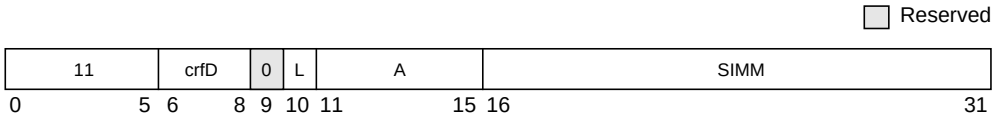
Simplified mnemonics:

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	Optional	Form
UISA			X

Compare Immediate

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




```
a ← (rA)
ifa < 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.

Other registers altered:

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

  Affected: LT, GT, EQ, SO

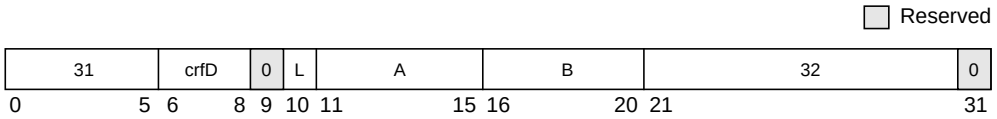
Simplified mnemonics:

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

PowerPC Architecture Level	Supervisor Level	Optional	Form
UISA			D

Compare Logical

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




```
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.

Other registers altered:

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

  Affected: LT, GT, EQ, SO

Simplified mnemonics:

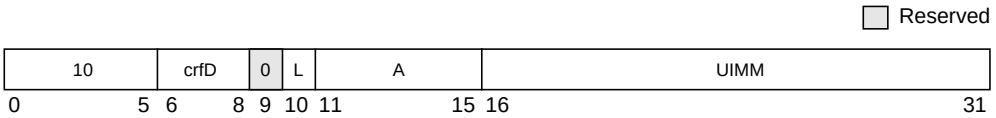
cmpld **r**A,**r**B	equivalent to	**cmpl** **0,1,r**A,**r**B
cmplw **cr3,r**A,**r**B	equivalent to	**cmpl** **3,0,r**A,**r**B

PowerPC Architecture Level	Supervisor Level	Optional	Form
UISA			X

cmpli cmpli

Compare Logical Immediate

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




```
a ← (rA)
ifa <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.

Other registers altered:

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

  Affected: LT, GT, EQ, SO

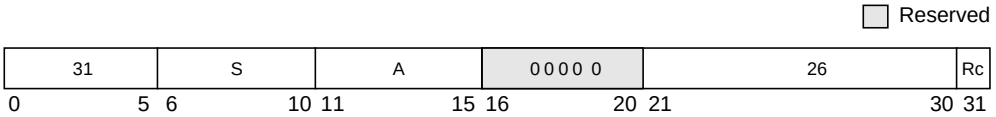
Simplified mnemonics:

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

PowerPC Architecture Level	Supervisor Level	Optional	Form
UISA			D

Count Leading Zeros Word

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):

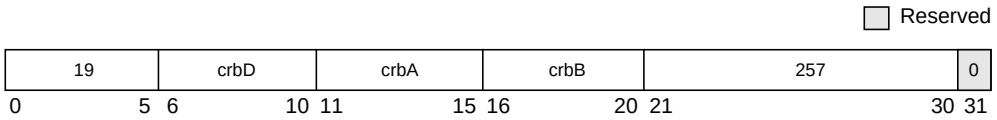
  Affected: LT, GT, EQ, SO (if Rc = 1)

  Note: If Rc = 1, then LT is cleared in the CR0 field.

PowerPC Architecture Level	Supervisor Level	Optional	Form
UISA			X

Condition Register AND

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



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. The result is placed into the condition register bit specified by **crb**D.

Other registers altered:

- Condition Register:

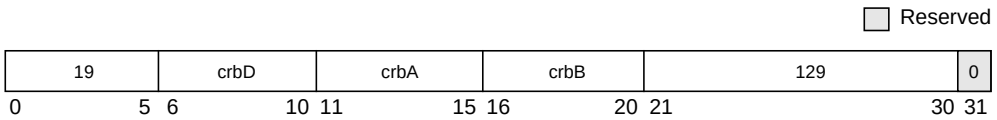
  Affected: Bit specified by operand **crb**D

PowerPC Architecture Level	Supervisor Level	Optional	Form
UISA			XL

crandc crandc

Condition Register AND with Complement

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



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

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:

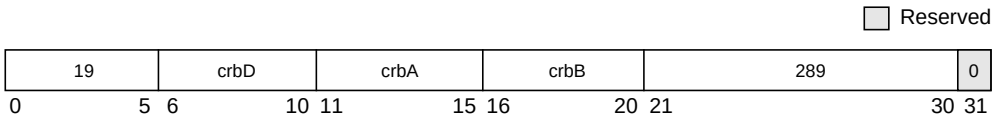
- Condition Register:

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

PowerPC Architecture Level	Supervisor Level	Optional	Form
UISA			XL

Condition Register Equivalent

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



CR[**crb**D] ← CR[**crb**A] ≡ CR[**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 complemented result is placed into the condition register bit specified by **crb**D.

Other registers altered:

- Condition Register:

  Affected: Bit specified by operand crbD

Simplified mnemonics:

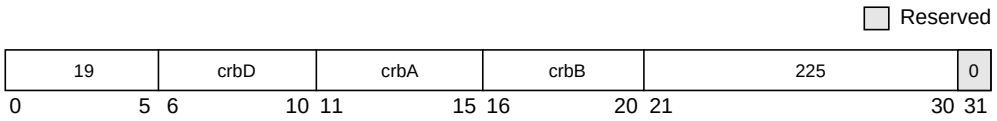
crset **crb**D equivalent to **creqv** **crb**D,**crb**D,**crb**D

PowerPC Architecture Level	Supervisor Level	Optional	Form
UISA			XL

crnand crnand

Condition Register NAND

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



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:

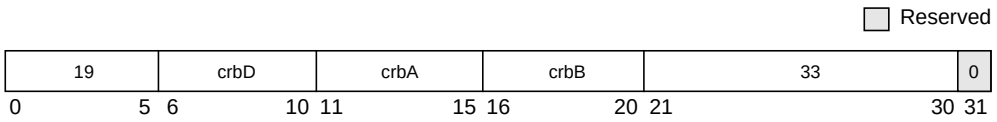
  Affected: Bit specified by operand **crb**D

PowerPC Architecture Level	Supervisor Level	Optional	Form
UISA			XL

crnor crnor

Condition Register NOR

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



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

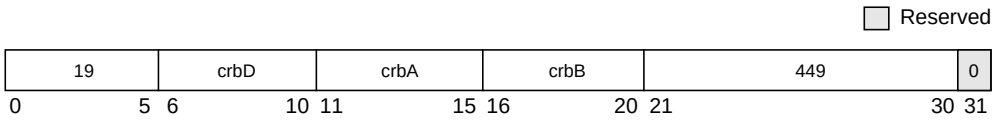
Simplified mnemonics:

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

PowerPC Architecture Level	Supervisor Level	Optional	Form
UISA			XL

Condition Register OR

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



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 crbD

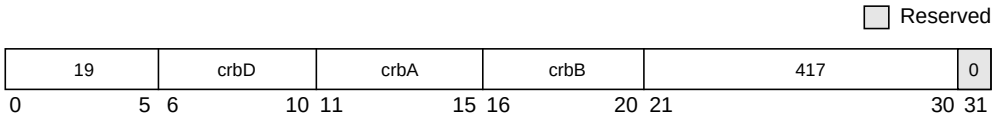
Simplified mnemonics:

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

PowerPC Architecture Level	Supervisor Level	Optional	Form
UISA			XL

Condition Register OR with Complement

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



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:

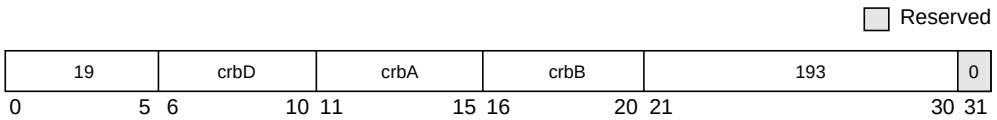
- Condition Register:

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

PowerPC Architecture Level	Supervisor Level	Optional	Form
UISA			XL

Condition Register XOR

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



CR[**crb**D] ← CR[**crb**A] ⊕ CR[**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

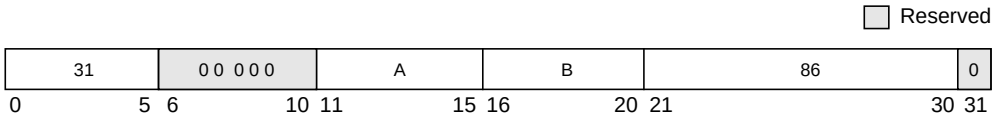
Simplified mnemonics:

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

PowerPC Architecture Level	Supervisor Level	Optional	Form
UISA			XL

Data Cache Block Flush

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



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

The **dcbf** instruction invalidates the *block* in the data *cache* addressed by EA, copying the block to memory first, if there is any dirty data in it. If the processor 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 action taken depends 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 for the various states of the *memory coherency* attribute (M bit).

- Coherency required
  - — Unmodified block—Invalidates copies of the block in the data caches of all processors.
  - — Modified block—Copies the block to memory. Invalidates copies of the block in the data caches of all processors.
  - — Absent block—If modified copies of the block are in the data caches of other processors, causes them to be copied to memory and invalidated in those data caches. If unmodified copies are in the data caches of other processors, causes those copies to be invalidated in those data caches.
- Coherency not 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 (target block not in cache)—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 may be 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.

Other registers altered:

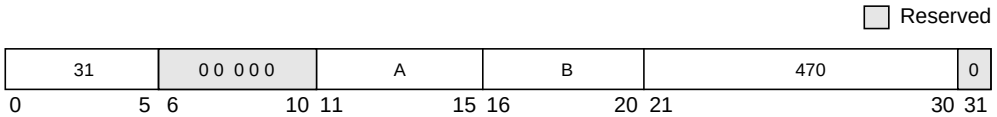
- None

PowerPC Architecture Level	Supervisor Level	Optional	Form
VEA			X

dcbi dcbi

Data Cache Block Invalidate

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
  - — Unmodified block—Invalidates copies of the block in the data caches of all processors.
  - — Modified block—Invalidates copies of the block in the data caches of all processors. (Discards the modified contents.)
  - — Absent block—If copies of the block are in the data caches of any other processor, causes the copies to be invalidated in those data caches. (Discards any modified contents.)
- Coherency not required
  - — Unmodified block—Invalidates the block in the processor's data cache.
  - — Modified block—Invalidates the block in the processor's data cache. (Discards the modified contents.)
  - — Absent block (target block not in cache)—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:

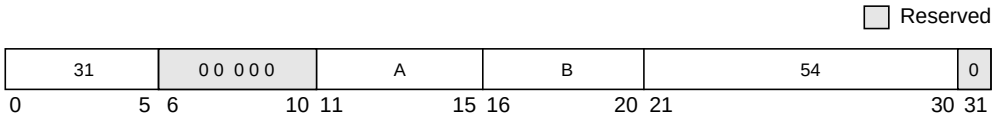
- None

PowerPC Architecture Level	Supervisor Level	Optional	Form
OEA	√		X

dcbst dcbst

Data Cache Block Store

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



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

The **dcbst** instruction executes as follows:

- 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 data cache of any processor and has been modified, the writing of it to main memory is initiated.
- 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 data cache of this processor and has been modified, the writing of it to main memory is initiated.

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 may also be treated as a load for referenced and changed bit recording except that referenced and changed bit recording may not occur.

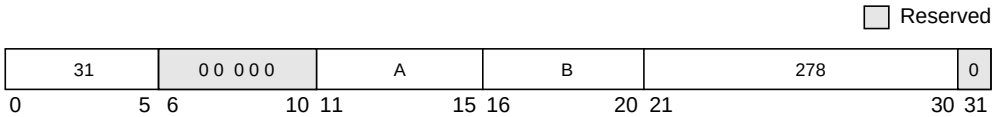
Other registers altered:

- None

PowerPC Architecture Level	Supervisor Level	Optional	Form
VEA			X

Data Cache Block Touch

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



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

This instruction is a hint that performance will probably 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. The hint is ignored if the block is caching-inhibited. Executing **dcbt** does not cause the system alignment error handler to be invoked.

This instruction may be treated as a load from the addressed byte with respect to address translation, memory protection, and reference and change recording, except that 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.

Other registers altered:

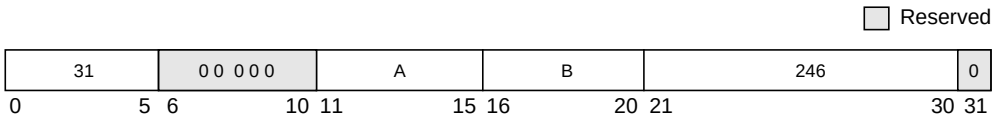
- None

PowerPC Architecture Level	Supervisor Level	Optional	Form
VEA			X

dcbtst dcbtst

Data Cache Block Touch for Store

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



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

This instruction is a hint that performance will be improved if the block containing the byte addressed by EA is fetched into the data cache, because the program will probably soon store into the addressed byte. The hint is ignored if the block is caching-inhibited. Executing **dcbtst** does not cause the system alignment error handler to be invoked.

This instruction operates as a load from the addressed byte with respect to address translation and protection, except that no exception occurs in the case of a translation fault or protection violation. Also, if the referenced and changed bits are recorded, they are recorded as if the access was a load.

The program uses **dcbtst** to request a cache block fetch to guarantee that a subsequent store will be to a cached location. The program can later execute store instructions to put data into memory. However, the processor is not obliged to load the addressed cache block into the data cache.

Other registers altered:

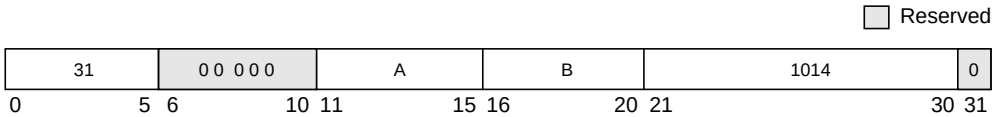
- None

PowerPC Architecture Level	Supervisor Level	Optional	Form
VEA			X

dcbz dcbz

Data Cache Block Set to Zero

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



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

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.
- If the cache block containing the byte addressed by EA is not in the data cache and the corresponding *page* is caching-allowed, the cache block is allocated 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 clears all bytes in main memory that corresponds 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.

This instruction is treated as a store to the addressed byte with respect to address translation, 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.

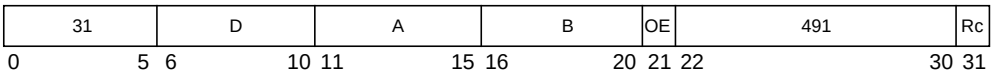
Other registers altered:

- None

PowerPC Architecture Level	Supervisor Level	Optional	Form
VEA			X

Divide Word

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 32-bit quotient is formed and placed in **r**D. 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 $0 \le r < |\text{divisor}|$ (if the dividend is non-negative), and $-|\text{divisor}| < r \le 0$ (if the dividend is negative).

If an attempt is made to perform any of the divisions—0x8000_0000 ÷ –1or <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 = –231 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:

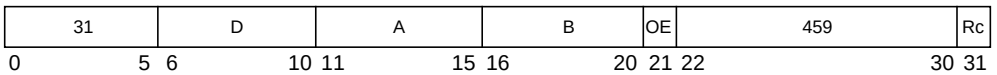
  Affected: SO, OV (if OE = 1)

  Note: The setting of the affected bits in the XER is mode-independent, and reflects overflow of the 32-bit result.

PowerPC Architecture Level	Supervisor Level	Optional	Form
UISA			XO

Divide Word Unsigned

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. A 32-bit quotient is formed. The 32-bit quotient is placed into **r**D. 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:

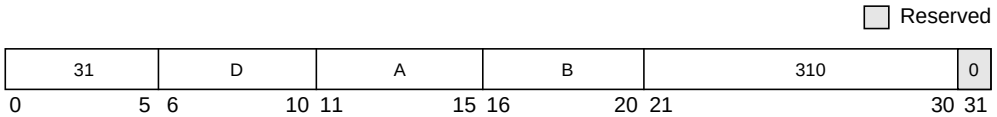
  Affected: SO, OV (if OE = 1)

  Note: The setting of the affected bits in the XER is mode-independent, and reflects overflow of the 32-bit result.

PowerPC Architecture Level	Supervisor Level	Optional	Form
UISA			XO

External Control In Word Indexed

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



The **eciwx** instruction allows the system designer to map special devices in an alternative way. The MMU translation of the EA is not used to select the special device, as it is used in most instructions such as loads and stores. Rather, it is used as an address operand that is passed to the device over the address bus. Four other pins (the burst and size pins on the 60x bus) are used to select the device; these four pins output the 4-bit resource ID (RID) field that is located in the EAR register. The **eciwx** instruction also loads a word from the data bus that is output by the special device.

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. However, note that 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**. This instruction is optional in the PowerPC architecture.

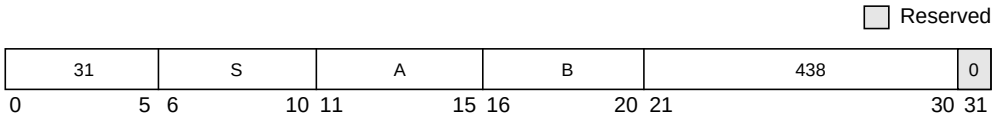
Other registers altered:

- None

PowerPC Architecture Level	Supervisor Level	Optional	Form
VEA		√	X

External Control Out Word Indexed

ecowx **r**S,**r**A,**r**B



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. However, note that 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, nd referenced and changed bit recording, and the ordering performed by **eieio**. This instruction is optional in the PowerPC architecture.

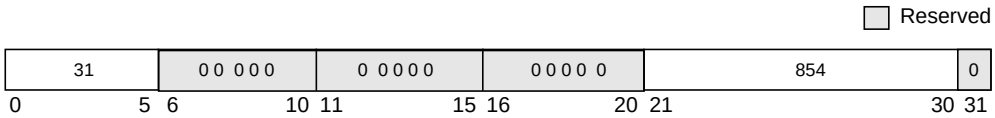
Other registers altered:

- None

PowerPC Architecture Level	Supervisor Level	Optional	Form
VEA		√	X

eieio

Enforce In-Order Execution of I/O



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** 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**, **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). 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, 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 that 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).

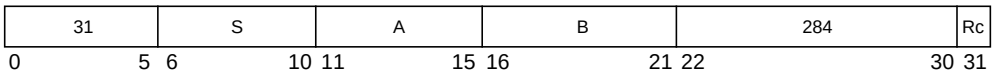
Other registers altered:

- None

PowerPC Architecture Level	Supervisor Level	Optional	Form
VEA			X

Equivalent

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



rA ← (**r**S) ≡ (**r**B)

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

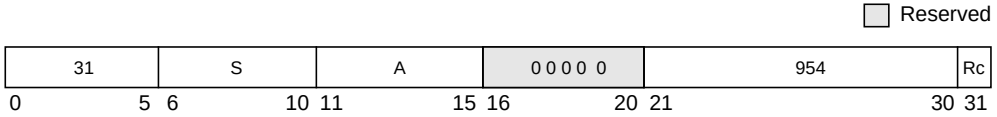
Other registers altered:

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

PowerPC Architecture Level	Supervisor Level	Optional	Form
UISA			X

Extend Sign Byte

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 **r**S[24-31] are placed into **r**A[24-31]. Bit 24 of **r**S is placed into **r**A[0-23].

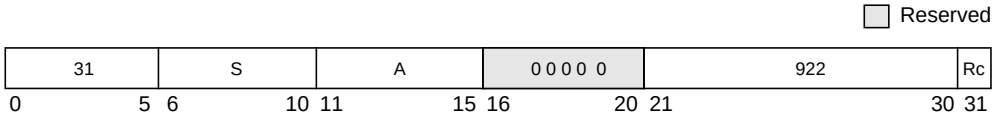
Other registers altered:

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

PowerPC Architecture Level	Supervisor Level	Optional	Form
UISA			X

Extend Sign Half Word

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 **r**S[16-31] are placed into **r**A[16-31]. Bit 16 of **r**S is placed into **r**A[0–15].

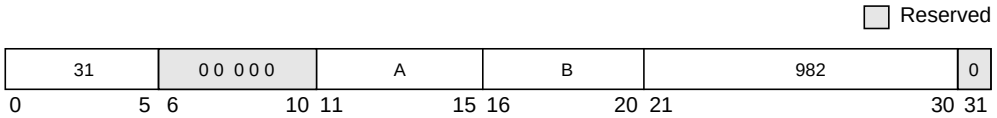
Other registers altered:

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

PowerPC Architecture Level	Supervisor Level	Optional	Form
UISA			X

Instruction Cache Block Invalidate

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 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:

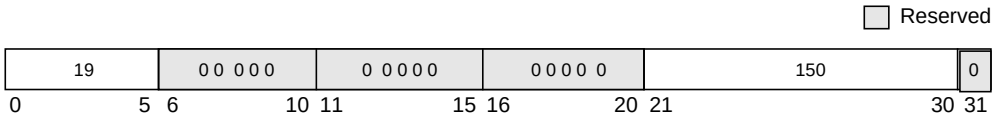
- None

PowerPC Architecture Level	Supervisor Level	Optional	Form
VEA			X

isync isync

Instruction Synchronize

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 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. This instruction is context synchronizing.

Other registers altered:

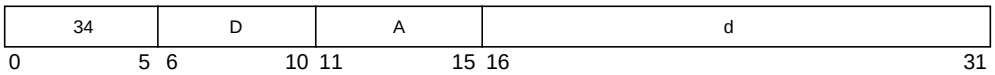
- None

PowerPC Architecture Level	Supervisor Level	Optional	Form
VEA			XL

lbz lbz

Load Byte and Zero

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:

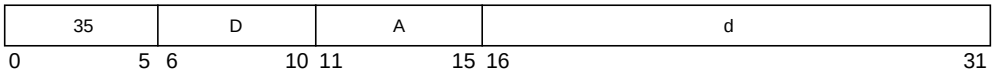
- None

PowerPC Architecture Level	Supervisor Level	Optional	Form
UISA			D

lbzu lbzu

Load Byte and Zero with Update

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:

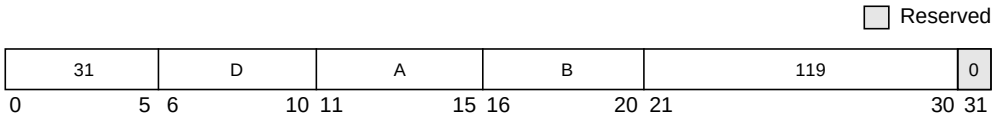
- None

PowerPC Architecture Level	Supervisor Level	Optional	Form
UISA			D

lbzux lbzux

Load Byte and Zero with Update Indexed

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:

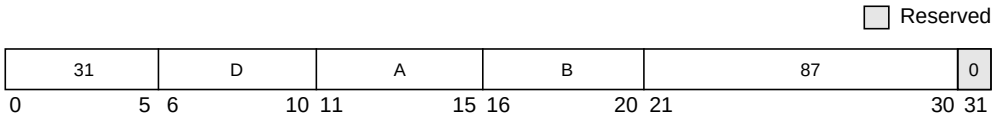
- None

PowerPC Architecture Level	Supervisor Level	Optional	Form
UISA			X

lbzx lbzx

Load Byte and Zero Indexed

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.

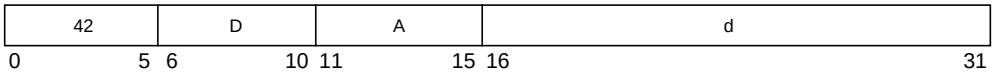
Other registers altered:

- None

PowerPC Architecture Level	Supervisor Level	Optional	Form
UISA			X

Load Half Word Algebraic

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:

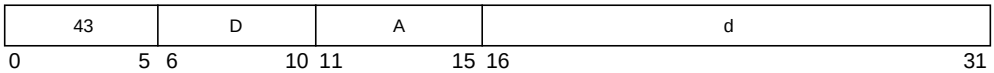
- None

PowerPC Architecture Level	Supervisor Level	Optional	Form
UISA			D

lhau lhau

Load Half Word Algebraic with Update

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:

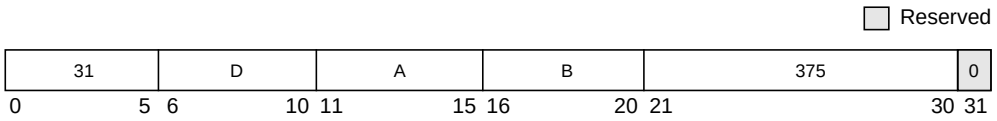
- None

PowerPC Architecture Level	Supervisor Level	Optional	Form
UISA			D

lhaux lhaux

Load Half Word Algebraic with Update Indexed

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:

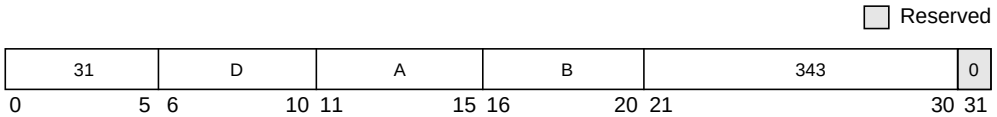
- None

PowerPC Architecture Level	Supervisor Level	Optional	Form
UISA			X

lhax lhax

Load Half Word Algebraic Indexed

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.

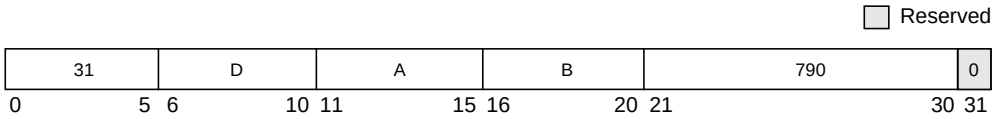
Other registers altered:

- None

PowerPC Architecture Level	Supervisor Level	Optional	Form
UISA			X

Load Half Word Byte-Reverse Indexed

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.

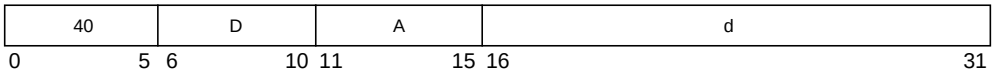
Other registers altered:

- None

PowerPC Architecture Level	Supervisor Level	Optional	Form
UISA			X

Load Half Word and Zero

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.

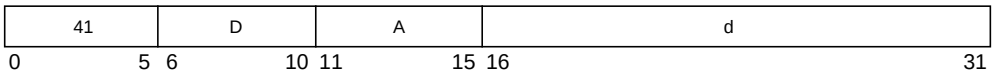
Other registers altered:

- None

PowerPC Architecture Level	Supervisor Level	Optional	Form
UISA			D

Load Half Word and Zero with Update

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:

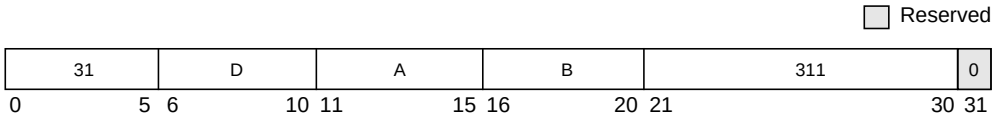
- None

PowerPC Architecture Level	Supervisor Level	Optional	Form
UISA			D

lhzux lhzux

Load Half Word and Zero with Update Indexed

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.

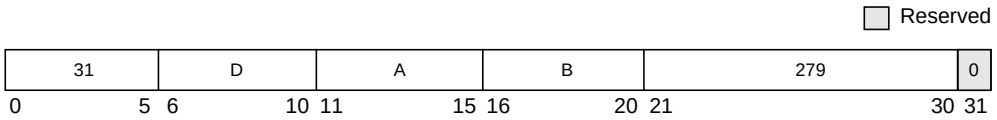
Other registers altered:

- None

PowerPC Architecture Level	Supervisor Level	Optional	Form
UISA			X

Load Half Word and Zero Indexed

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




```
if rA = 0 then b ← 0
elseb ← (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:

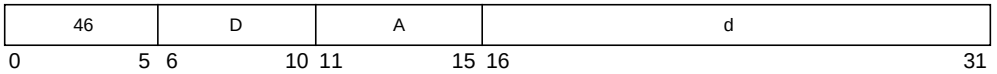
- None

PowerPC Architecture Level	Supervisor Level	Optional	Form
UISA			X

lmw lmw

Load Multiple Word

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




```
if rA = 0 then b ← 0
elseb ← (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. 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 that, 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:

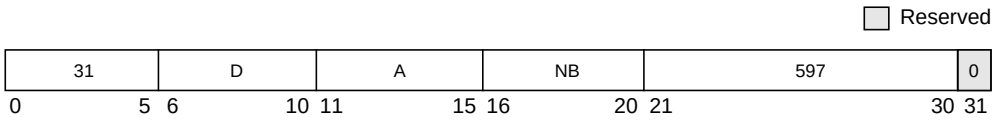
- None

PowerPC Architecture Level	Supervisor Level	Optional	Form
UISA			D

lswi lswi

Load String Word Immediate

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




```
if rA = 0 then EA ← 0
else EA ← (rA)
if NB = 0 then n ← 32
elsen ← NB
r ← rD – 1
i ← 32
do while n > 0
 if i = 32 then
  r ← r + 1 (mod 32)
  GPR(r) ← 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, *n* = 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 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.

Note that, 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.

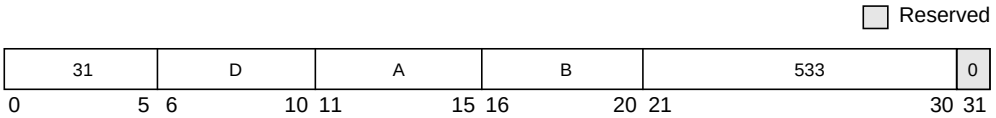
Other registers altered:

- None

PowerPC Architecture Level	Supervisor Level	Optional	Form
UISA			X

Load String Word Indexed

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 ← 32
rD ← undefined
 do while n > 0
 if i = 32 then
  r ← r + 1 (mod 32)
  GPR(r) ← 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 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. Note that, 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.

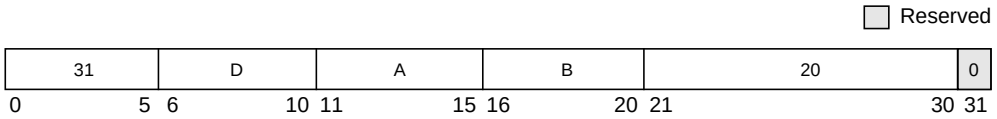
Other registers altered:

- None

PowerPC Architecture Level	Supervisor Level	Optional	Form
UISA			X

Load Word and Reserve Indexed

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.

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

Other registers altered:

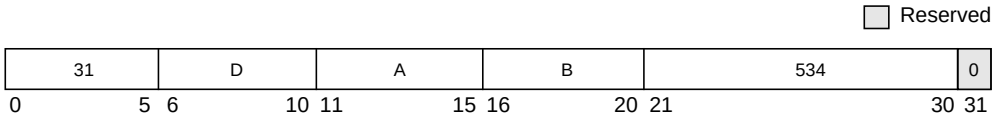
- None

PowerPC Architecture Level	Supervisor Level	Optional	Form
UISA			X

lwbrx lwbrx

Load Word Byte-Reverse Indexed

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




```
if rA = 0 then b ← 0
elseb ← (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 MPCxxx may run the **lwbrx** instructions with greater latency than other types of load instructions.

Other registers altered:

- None

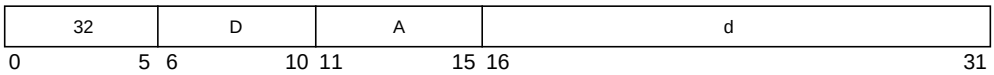
PowerPC Architecture Level	Supervisor Level	Optional	Form
UISA			X

lwz

lwz

Load Word and Zero

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




```
if rA = 0 then b ← 0
elseb ← (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.

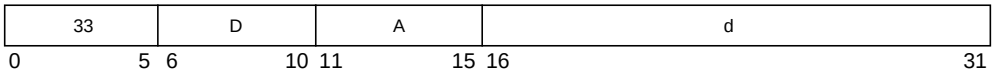
Other registers altered:

- None

PowerPC Architecture Level	Supervisor Level	Optional	Form
UISA			D

Load Word and Zero with Update

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.

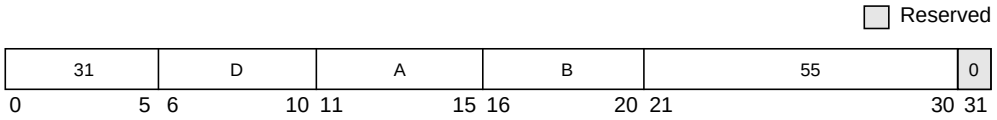
Other registers altered:

- None

PowerPC Architecture Level	Supervisor Level	Optional	Form
UISA			D

Load Word and Zero with Update Indexed

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



EA ← (**r**A) + (**r**B)
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.

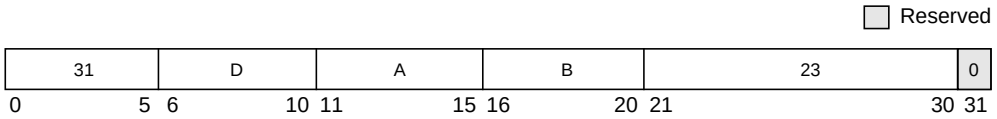
Other registers altered:

- None

PowerPC Architecture Level	Supervisor Level	Optional	Form
UISA			X

Load Word and Zero Indexed

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




```
if rA = 0 then b ← 0
elseb ← (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.

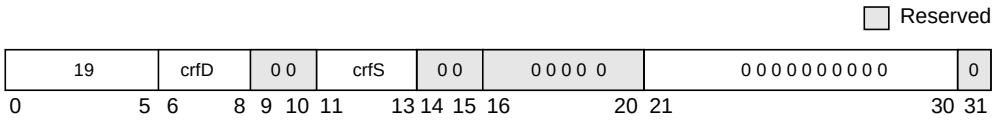
Other registers altered:

- None

PowerPC Architecture Level	Supervisor Level	Optional	Form
UISA			X

Move Condition Register Field

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:

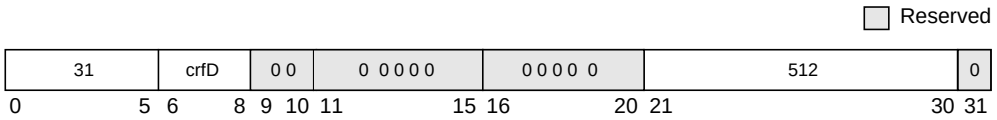
- Condition Register (CR field specified by operand **crf**D):

  Affected: LT, GT, EQ, SO

PowerPC Architecture Level	Supervisor Level	Optional	Form
UISA			XL

Move to Condition Register from XER

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:

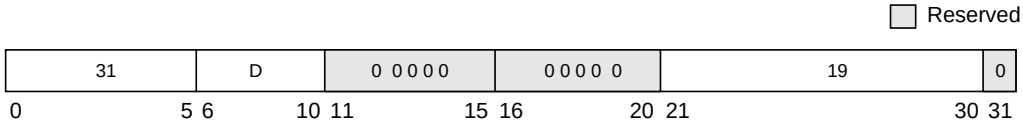
- Condition Register (CR field specified by operand **crf**D):

  Affected: LT, GT, EQ, SO
- XER[0–3]

PowerPC Architecture Level	Supervisor Level	Optional	Form
UISA			X

Move from Condition Register

mfcr **r**D



rD ← CR

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

Other registers altered:

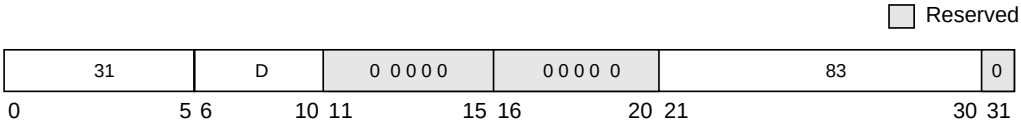
- None

PowerPC Architecture Level	Supervisor Level	Optional	Form
UISA			X

mfmsr mfmsr

Move from Machine State Register

mfmsr **r**D



rD ← MSR

The contents of the MSR are placed into **r**D. This is a supervisor-level instruction.

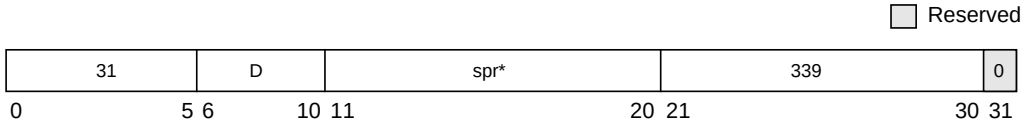
Other registers altered:

- None

PowerPC Architecture Level	Supervisor Level	Optional	Form
OEA	√		X

Move from Special-Purpose Register

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 9. The contents of the designated special-purpose register are placed into **r**D.

Table 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 that 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 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 10. The contents of the designated SPR are placed into **r**D. SPR[0] = 1 if and only if reading the register is supervisor-level. Execution of this instruction specifying a defined and supervisor-level register when MSR[PR] = 1 will result in a priviledged 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 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 10, either an illegal instruction type program exception occurs or the results are boundedly undefined.

Other registers altered:

- None

Table 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
26	00000	11010	SRR0	Supervisor
27	00000	11011	SRR1	Supervisor
80	00010	10000	EIE[2]	Supervisor
81	00010	10001	EID[3]	Supervisor
144	00100	10000	CMPA[4]	Supervisor
145	00100	10001	CMPB[4]	Supervisor
146	00100	10010	CMPC[4]	Supervisor
147	00100	10011	CMPD[4]	Supervisor
148	00100	10100	ICR[4]	Supervisor
149	00100	10101	DER[4]	Supervisor

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

SPR[1]			Register Name	Access
Decimal	spr[5–9]	spr[0–4]		
150	00100	10110	COUNTA[4]	Supervisor
151	00100	10111	COUNTB[4]	Supervisor
152	00100	11000	CMPE[4]	Supervisor
153	00100	11001	CMPF[4]	Supervisor
154	00100	11010	CMPG[4]	Supervisor
155	00100	11011	CMPH[4]	Supervisor
156	00100	11100	LCTRL1[4]	Supervisor
157	00100	11101	LCTRL2[4]	Supervisor
158	00100	11110	ICTRL[4]	Supervisor
159	00100	11111	BAR[4]	Supervisor
272	01000	10000	SPRG0	Supervisor
273	01000	10001	SPRG1	Supervisor
274	01000	10010	SPRG2	Supervisor
275	01000	10011	SPRG3	Supervisor
287	01000	11111	PVR	Supervisor
560	10001	10000	IC_CST	Supervisor
561	10001	10001	IC_ADR	Supervisor
562	10001	10010	IC_DAT	Supervisor
568	10001	11000	DC_CST	Supervisor
569	10001	11001	DC_ADR	Supervisor
570	10001	11010	DC_DAT	Supervisor
630	10011	10110	DPDR[4]	Supervisor
638	10011	11110	IMMR	Supervisor
784	11000	10000	MI_CTR	Supervisor
786	11000	10010	MI_AP	Supervisor
787	11000	10011	MI_EPN	Supervisor
789	11000	10101	MI_TWC	Supervisor
790	11000	10110	MI_RPN	Supervisor
792	11000	11000	MD_CTR	Supervisor
793	11000	11001	M_CASID	Supervisor
794	11000	11010	MD_AP	Supervisor

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

SPR[1]			Register Name	Access
Decimal	spr[5–9]	spr[0–4]		
795	11000	11011	MD_EPN	Supervisor
796	11000	11100	M_TWB	Supervisor
797	11000	11101	MD_TWC	Supervisor
798	11000	11110	MD_RPN	Supervisor
799	11000	11111	M_TW	Supervisor
816	11001	10000	MI_DBCAM	Supervisor
817	11001	10001	MI_DBRAM0	Supervisor
818	11001	10010	MI_DBRAM1	Supervisor
824	11001	11000	MD_DBCAM	Supervisor
825	11001	11001	MI_DBRAM0	Supervisor
826	11001	11010	MI_DBRAM1	Supervisor

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

[2]Sets EE Bit (Bit 16) in MSR.

[3]Clears EE Bit (Bit 16) in MSR.

[4]Development Support (Debug) Register.

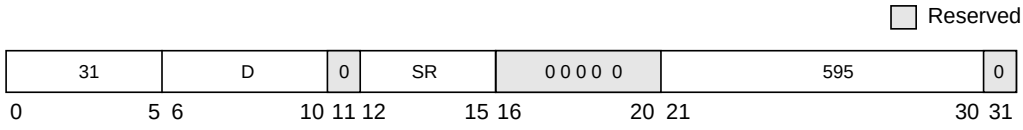
For **mtspr** and **mfspr** instructions, the SPR number coded in assembly language does not appear directly as a 10-bit binary number in the instruction. The number coded is split into two 5-bit halves that are reversed in the instruction, with the high-order five bits appearing in bits 16–20 of the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

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

* Note that **mfspr** is supervisor-level only if SPR[0] = 1.

Move from Segment Register

mfsr **r**D,SR



rD ← SEGREG(SR)

The contents of segment register SR are placed into **r**D. This is a supervisor-level instruction.

Other registers altered:

- None

PowerPC Architecture Level	Supervisor Level	Optional	Form
OEA	√		X

mfsrin mfsrin

Move from Segment Register Indirect

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.

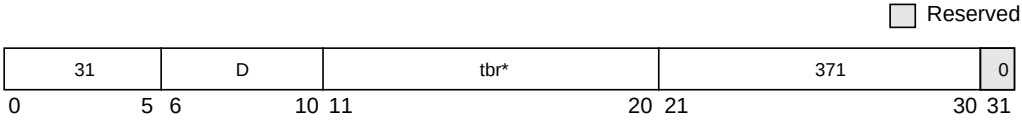
Other registers altered:

- None

PowerPC Architecture Level	Supervisor Level	Optional	Form
OEA	√		X

Move from Time Base

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
```

Table 11. TBR Encodings for mftb

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

*Note that 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 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.

Other registers altered:

- None

Simplified mnemonics:

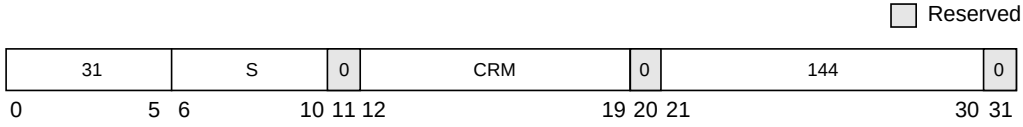
mftb rD	equivalent to	**mftb r**D**,268**
mftbu rD	equivalent to	**mftb r**D**,269**

PowerPC Architecture Level	Supervisor Level	Optional	Form
VEA			XFX

mtcrf mtcrf

Move to Condition Register Fields

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 that 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.

Other registers altered:

- CR fields selected by mask

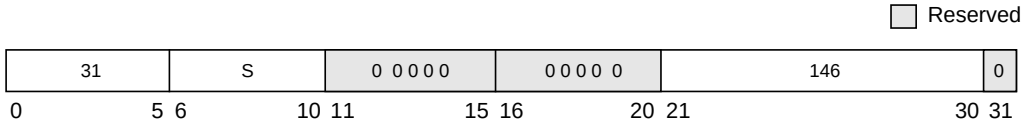
Simplified mnemonics:

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

PowerPC Architecture Level	Supervisor Level	Optional	Form
UISA			XFX

Move to Machine State Register

mtmsr **r**S



MSR ← (**r**S)

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.

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.

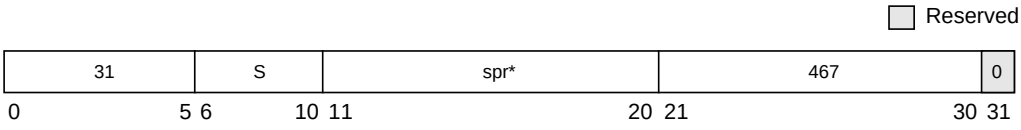
Other registers altered:

- MSR

PowerPC Architecture Level	Supervisor Level	Optional	Form
OEA	√		X

Move to Special-Purpose Register

mtspr SPR**,rS**



***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 12. The contents of **r**S are placed into the designated special-purpose register.

Table 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 that 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 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 12.

Simplified mnemonics:

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

In the PowerPC OEA, the SPR field denotes a special-purpose register, encoded as shown in Table 13. 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 priviledged 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 13 and has SPR[0] = 1 is to cause a priviledged 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 13, either an illegal instruction type program exception occurs or the results are boundedly undefined.

Other registers altered:

- See Table 13.

Table 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
26	00000	11010	SRR0	Supervisor
27	00000	11011	SRR1	Supervisor
80	00010	10000	EIE[2]	Supervisor
81	00010	10001	EID[3]	Supervisor
144	00100	10000	CMPA[4]	Supervisor
145	00100	10001	CMPB[4]	Supervisor
146	00100	10010	CMPC[4]	Supervisor
147	00100	10011	CMPD[4]	Supervisor

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

SPR[1]			Register Name	Access
Decimal	spr[5–9]	spr[0–4]		
148	00100	10100	ICR[4]	Supervisor
149	00100	10101	DER[4]	Supervisor
150	00100	10110	COUNTA[4]	Supervisor
151	00100	10111	COUNTB[4]	Supervisor
152	00100	11000	CMPE[4]	Supervisor
153	00100	11001	CMPF[4]	Supervisor
154	00100	11010	CMPG[4]	Supervisor
155	00100	11011	CMPH[4]	Supervisor
156	00100	11100	LCTRL1[4]	Supervisor
157	00100	11101	LCTRL2[4]	Supervisor
158	00100	11110	ICTRL[4]	Supervisor
159	00100	11111	BAR[4]	Supervisor
272	01000	10000	SPRG0	Supervisor
273	01000	10001	SPRG1	Supervisor
274	01000	10010	SPRG2	Supervisor
275	01000	10011	SPRG3	Supervisor
284	01000	11100	TB Write	Supervisor
285	01000	11101	TBU Write	Supervisor
560	10001	10000	IC_CST	Supervisor
561	10001	10001	IC_ADR	Supervisor
562	10001	10010	IC_DAT	Supervisor
568	10001	11000	DC_CST	Supervisor
569	10001	11001	DC_ADR	Supervisor
570	10001	11010	DC_DAT	Supervisor
630	10011	10110	DPDR[4]	Supervisor
638	10011	11110	IMMR	Supervisor
784	11000	10000	MI_CTR	Supervisor
786	11000	10010	MI_AP	Supervisor
787	11000	10011	MI_EPN	Supervisor
789	11000	10101	MI_TWC	Supervisor
790	11000	10110	MI_RPN	Supervisor

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

SPR[1]			Register Name	Access
Decimal	spr[5–9]	spr[0–4]		
792	11000	11000	MD_CTR	Supervisor
793	11000	11001	M_CASID	Supervisor
794	11000	11010	MD_AP	Supervisor
795	11000	11011	MD_EPN	Supervisor
796	11000	11100	M_TWB	Supervisor
797	11000	11101	MD_TWC	Supervisor
798	11000	11110	MD_RPN	Supervisor
799	11000	11111	M_TW	Supervisor
816	11001	10000	MI_DBCAM	Supervisor
817	11001	10001	MI_DBRAM0	Supervisor
818	11001	10010	MI_DBRAM1	Supervisor
824	11001	11000	MD_DBCAM	Supervisor
825	11001	11001	MI_DBRAM0	Supervisor
826	11001	11010	MI_DBRAM1	Supervisor

[1]Note that 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.

[2]Sets EE Bit (Bit 16) in MSR.

[3]Clears EE Bit (Bit 16) in MSR.

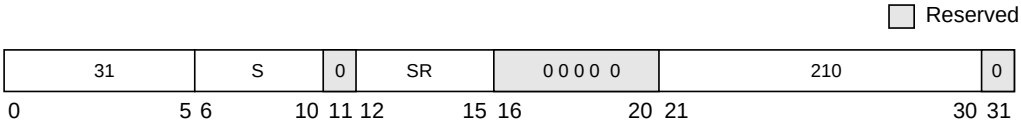
[4]Development Support (Debug) Register.

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

* Note that **mtspr** is supervisor-level only if SPR[0] = 1.

Move to Segment Register

mtsr SR,**r**S



SEGREG(SR) ← (**r**S)

The contents of **r**S are placed into SR. This is a supervisor-level instruction.

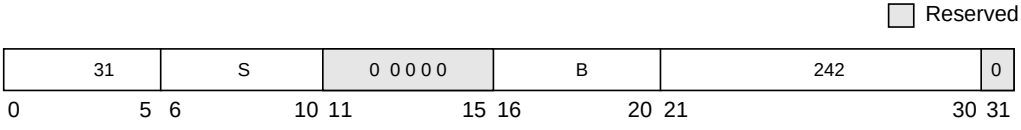
Other registers altered:

- None

PowerPC Architecture Level	Supervisor Level	Optional	Form
OEA	√		X

Move to Segment Register Indirect

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



SEGREG(**r**B[0–3]) ← (**r**S)

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.

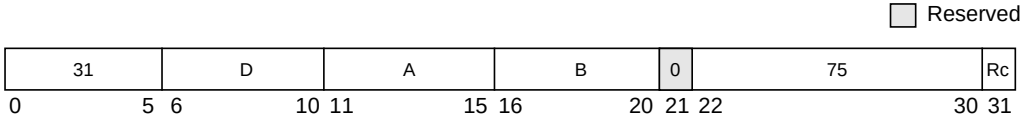
Other registers altered:

- None

PowerPC Architecture Level	Supervisor Level	Optional	Form
OEA	√		X

Multiply High Word

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

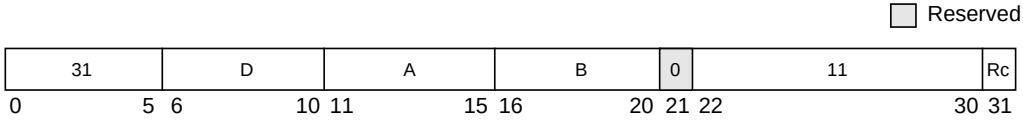
Other registers altered:

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

PowerPC Architecture Level	Supervisor Level	Optional	Form
UISA			XO

Multiply High Word Unsigned

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

Other registers altered:

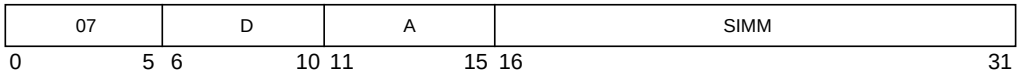
- Condition Register (CR0 field):
  Affected: LT, GT, EQ, SO(if Rc = 1)

PowerPC Architecture Level	Supervisor Level	Optional	Form
UISA			XO

mulli mulli

Multiply Low Immediate

mulli **r**D**,r**A,SIMM




```
prod[0–48] ← (rA) * SIMM
rD ← prod[16-48]
```

The first operand is (**r**A). The 16-bit second operand is the value of the SIMM field. The low-order 32-bits of the 48-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. This instruction can be used with **mulhd***x* or **mulhw***x* to calculate a full 64-bit product.

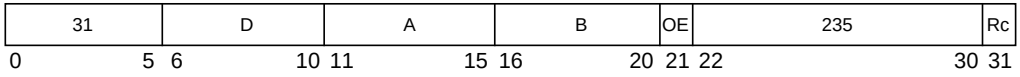
Other registers altered:

- None

PowerPC Architecture Level	Supervisor Level	Optional	Form
UISA			D

Multiply Low Word

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)



rD ← **r**A * **r**B

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 the correct 32-bit product for 32-bit implementations. 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.

Note that 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 “true” (infinitely precise) result if overflow occurs (see XER below).

- XER:

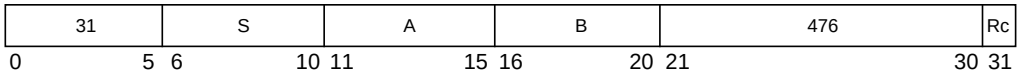
  Affected: SO, OV(if OE = 1)

  Note: The setting of the affected bits in the XER is mode-independent, and reflects overflow of the 32-bit result.

PowerPC Architecture Level	Supervisor Level	Optional	Form
UISA			XO

NAND

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



rA ← ¬ ((**r**S) & (**r**B))

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.

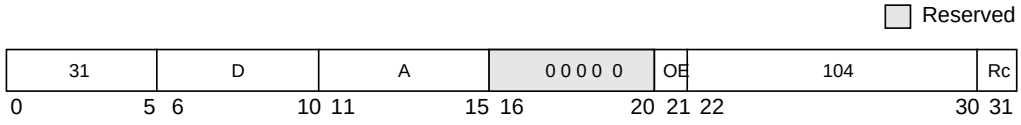
Other registers altered:

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

PowerPC Architecture Level	Supervisor Level	Optional	Form
UISA			X

Negate

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 ← ¬ (**r**A) + 1

The value 1 is added to the 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):

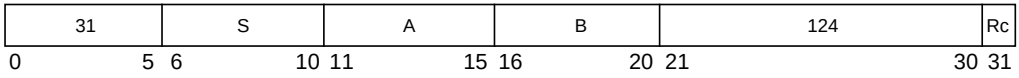
  Affected: LT, GT, EQ, SO(if Rc = 1)
- XER:

  Affected: SO OV(if OE = 1)

PowerPC Architecture Level	Supervisor Level	Optional	Form
UISA			XO

NOR

nor **r**A,**r**S,**r**B (Rc = 0)
nor. **r**A,**r**S,**r**B (Rc = 1)



rA ← ¬ ((**r**S) | (**r**B))

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)

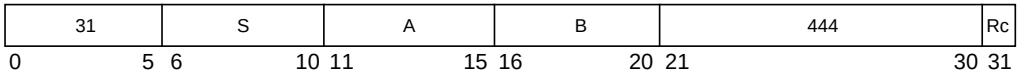
Simplified mnemonics:

not **r**D,**r**S equivalent to **nor** **r**A,**r**S,**r**S

PowerPC Architecture Level	Supervisor Level	Optional	Form
UISA			X

OR

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



rA ← (**r**S) | (**r**B)

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)

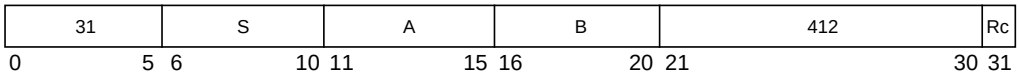
Simplified mnemonics:

mr	**r**A,**r**S	equivalent to	**or**	**r**A,**r**S,**r**S

PowerPC Architecture Level	Supervisor Level	Optional	Form
UISA			X

OR with Complement

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



rA ← (**r**S) | ¬ (**r**B)

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):

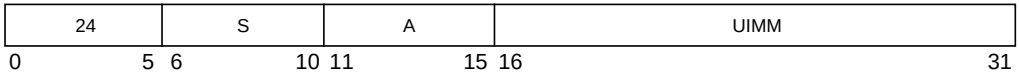
  Affected: LT, GT, EQ, SO(if Rc = 1)

PowerPC Architecture Level	Supervisor Level	Optional	Form
UISA			X

ori ori

OR Immediate

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:

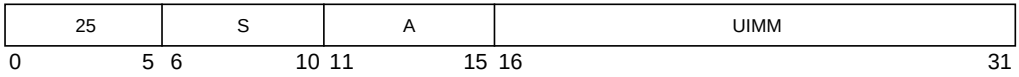
nop equivalent to **ori 0,0,0**

PowerPC Architecture Level	Supervisor Level	Optional	Form
UISA			D

oris oris

OR Immediate Shifted

oris **r**A,**r**S,UIMM




```
rA ← (rS) | (UIMM || (16)0)
```

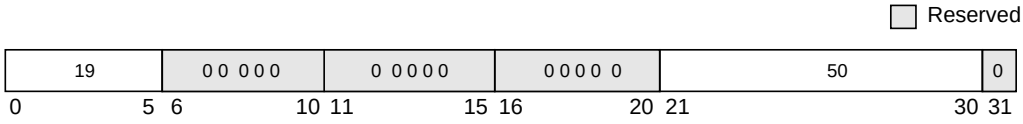
The contents of **r**S are ORed with UIMM || 0x0000 and the result is placed into **r**A.

Other registers altered:

- None

PowerPC Architecture Level	Supervisor Level	Optional	Form
UISA			D

Return from Interrupt




```
MSR[16–23, 25–27, 30–31] ← SRR1[16–23, 25–27, 30–31]
NIA ←iea SRR0[0–29] || 0b00
```

Bits SRR1[0,5-9,16-31] are placed into the corresponding bits of the MSR. 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. Note that an implementation may define addtional 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.

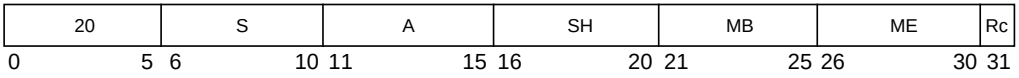
Other registers altered:

- MSR

PowerPC Architecture Level	Supervisor Level	Optional	Form
OEA	√		XL

Rotate Left Word Immediate then Mask Insert

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.

Note that **rlwimi** can be used to insert a bit field into the contents of **r**A using the methods shown below:

- To insert an *n*-bit field, that is left-justified **r**S, into **r**A starting at bit position *b*, set SH = 32 – *b*, MB = *b*, and ME = (*b* + *n*) – 1.
- To insert an *n*-bit field, that is right-justified in **r**S, into **r**A starting at bit position *b*, set SH = 32 – (*b* + *n*), MB = *b*, and ME = (*b* + *n*) – 1.

Other registers altered:

- Condition Register (CR0 field):

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

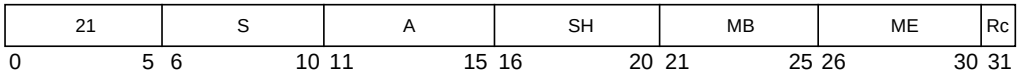
Simplified mnemonics:

inslwi **r**A,**r**S,*n*,*b*	equivalent to **rlwimi**	**r**A,**r**S,32 – *b*,*b*,*b* + *n* – 1
insrwi **r**A,**r**S,*n*,*b* (n > 0)	equivalent to **rlwimi**	**r**A,**r**S,**32 –** (*b* **+** *n*),*b*,(*b* + *n*) – 1

PowerPC Architecture Level	Supervisor Level	Optional	Form
UISA			M

Rotate Left Word Immediate then AND with Mask

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.

Note that **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)

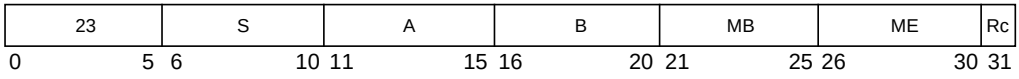
Simplified mnemonics:

extlwi rA**,r**S**,***n***,***b* (*n* > 0)	equivalent to	**rlwinm r**A**,r**S,b**,0,***n* – 1
extrwi rA**,r**S**,***n***,***b* (*n* > 0)	equivalent to	**rlwinm r**A**,r**S,b **+** *n*,32 – *n*,31**
rotlwi rA**,r**S**,***n*	equivalent to	**rlwinm r**A**,r**S**,***n*,0,31**
rotrwi rA**,r**S**,***n*	equivalent to	**rlwinm r**A**,r**S,32** – *n*,0,31**
slwi rA**,r**S**,***n* (*n* < 32)	equivalent to	**rlwinm r**A**,r**S**,***n*,0,**31–*n*
srwi rA**,r**S**,***n* (*n* < 32)	equivalent to	**rlwinm r**A**,r**S,32 – *n*,*n*,31**
clrlwi rA**,r**S**,***n* (*n* < 32)	equivalent to	**rlwinm r**A**,r**S,0,***n*,31**
clrrwi rA**,r**S**,***n* (*n* < 32)	equivalent to	**rlwinm r**A**,r**S,0,0,**31 – *n*
clrlslwi rA**,r**S**,***b*,*n* (*n* ≤ *b* < 32)	equivalent to	**rlwinm r**A**,r**S**,***n*,*b* – *n*,**31 – *n*

PowerPC Architecture Level	Supervisor Level	Optional	Form
UISA			M

Rotate Left Word then AND with Mask

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 that **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)

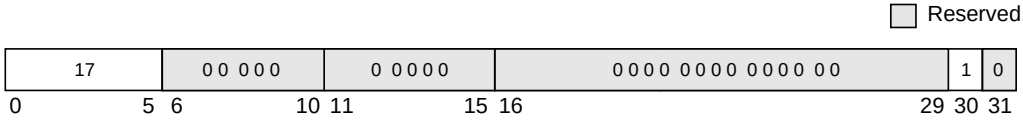
Simplified mnemonics:

rotlw **r**A,**r**S,**r**B equivalent to **rlwnmr**A,**r**S,**r**B,**0,31**

PowerPC Architecture Level	Supervisor Level	Optional	Form
UISA			M

sc sc

System Call



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.

The effective address of the instruction following the **sc** instruction is placed into SRR0. Bits 0, 5-9, and 16-31 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. An **sc** exception is generated. The exception alters the MSR. The exception causes the next instruction to be fetched from offset 0xC00 from the base real address indicated by the new setting of MSR[IP].

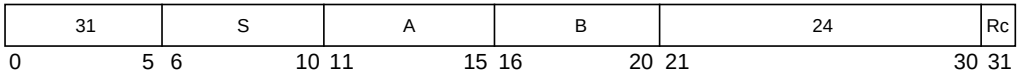
Other registers altered:

- Dependent on the system service
- SRR0
- SRR1
- MSR

PowerPC Architecture Level	Supervisor Level	Optional	Form
UISA/OEA			SC

Shift Left Word

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




```
n← rB[27-31]
rA ← ROTL(rS, n)
```

If bit 26 of **r**B = 0, the contents of **r**S are shifted left the number of bits specified by **r**B[27–31]. 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. If bit 26 of **r**B = 1, 32 zeros are placed into **r**A.

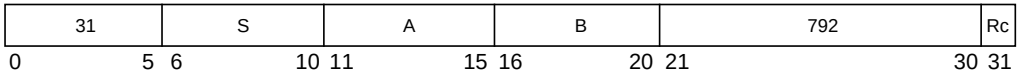
Other registers altered:

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

PowerPC Architecture Level	Supervisor Level	Optional	Form
UISA			X

Shift Right Algebraic Word

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



n← **r**B[27-31]
rA ← ROTL(**r**S, *n*)

If **r**B[26] = 0,then the contents of **r**S are shifted right the number of bits specified by **r**B[27–31]. Bits shifted out of position 31 are lost. The result is padded on the left with sign bits before being placed into **r**A. If **r**B[26] = 1, then **r**A is filled with 32 sign bits (bit 0) from **r**S. CR0 is set based on the value written 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 XER[CA] to be cleared.

Note that the **sraw** instruction, followed by **addze**, can by used to divide quickly by 2^n. The setting of the XER[CA] bit, by **sraw**, is independent of mode.

Other registers altered:

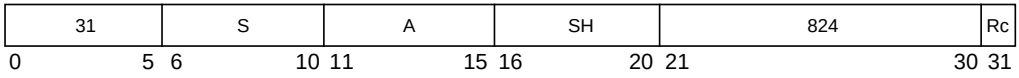
- Condition Register (CR0 field):

  Affected: LT, GT, EQ, SO(if Rc = 1)
- XER:

  Affected: CA

Shift Right Algebraic Word Immediate

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



n ← SH
r ← ROTL(**r**S, 32 − *n*)

The contents of **r**S are shifted right the number of bits specified by operand SH. Bits shifted out of position 31 are lost. The shifted value is sign-extended before being placed in **r**A. 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 XER[CA] to be cleared.

Note that the **srawi** instruction, followed by **addze**, can be used to divide quickly by 2^n. The setting of the CA bit, by **srawi**, is independent of mode.

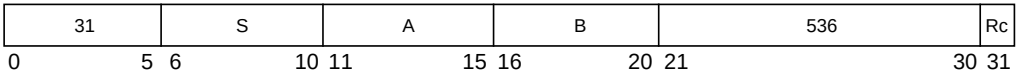
Other registers altered:

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

PowerPC Architecture Level	Supervisor Level	Optional	Form
UISA			X

Shift Right Word

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)
```

The contents of **r**S are shifted right the number of bits specified by the low-order six bits of **r**B. Bits shifted out of position 31 are lost. Zeros are supplied to the vacated positions on the left. The result is placed into **r**A.

Other registers altered:

- Condition Register (CR0 field):

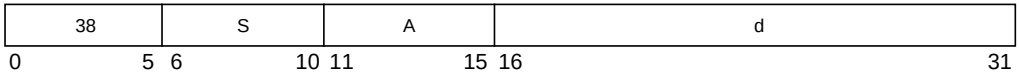
  Affected: LT, GT, EQ, SO(if Rc = 1)

PowerPC Architecture Level	Supervisor Level	Optional	Form
UISA			X

stb stb

Store Byte

stb **r**S,d**(r**A**)**




```
if rA = 0 then b ← 0
elseb ← (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:

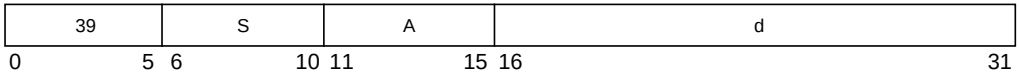
- None

PowerPC Architecture Level	Supervisor Level	Optional	Form
UISA			D

stbu stbu

Store Byte with Update

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.

Other registers altered:

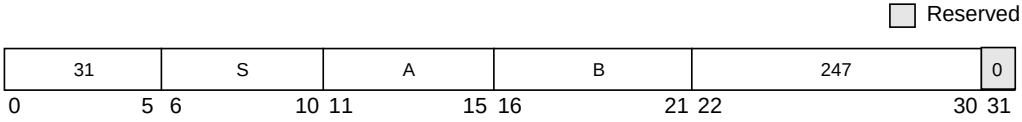
- None

PowerPC Architecture Level	Supervisor Level	Optional	Form
UISA			D

stbux stbux

Store Byte with Update Indexed

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:

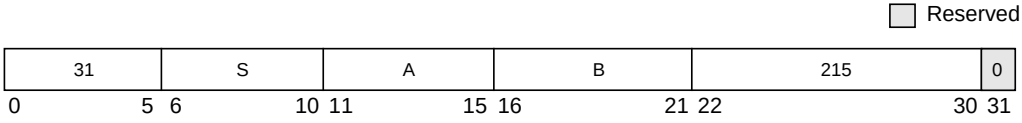
- None

PowerPC Architecture Level	Supervisor Level	Optional	Form
UISA			X

stbx stbx

Store Byte Indexed

stbx **r**S,**r**A,**r**B




```
if rA = 0 then b ← 0
elseb ← (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.

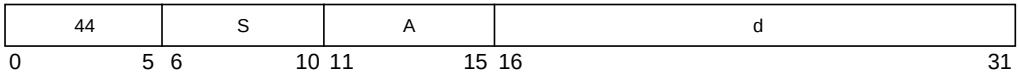
Other registers altered:

- None

PowerPC Architecture Level	Supervisor Level	Optional	Form
UISA			X

Store Half Word

sth **r**S,d**(r**A**)**




```
if rA = 0 then b ← 0
elseb ← (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:

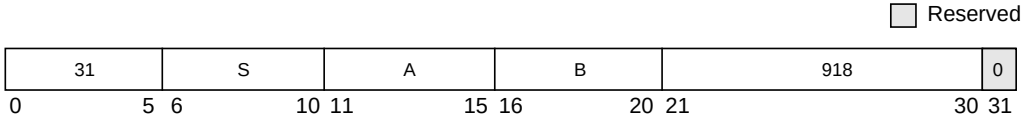
- None

PowerPC Architecture Level	Supervisor Level	Optional	Form
UISA			D

sthbrx sthbrx

Store Half Word Byte-Reverse Indexed

sthbrx **r**S,**r**A,**r**B




```
if rA = 0 then b ← 0
elseb ← (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 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 of **r**S are stored into bits 8–15 of the half word in memory addressed by EA.

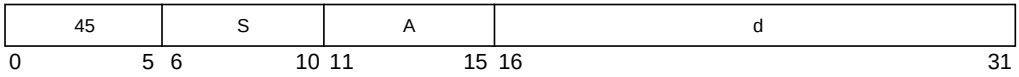
Other registers altered:

- None

PowerPC Architecture Level	Supervisor Level	Optional	Form
UISA			X

Store Half Word with Update

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:

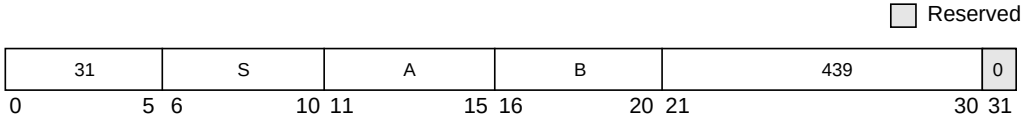
- None

PowerPC Architecture Level	Supervisor Level	Optional	Form
UISA			D

sthux sthux

Store Half Word with Update Indexed

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.

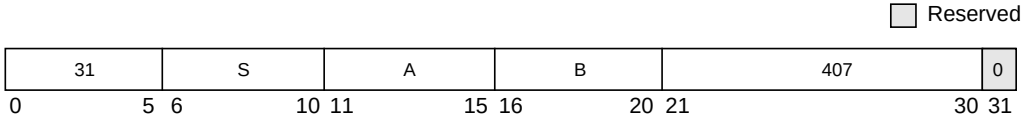
Other registers altered:

- None

PowerPC Architecture Level	Supervisor Level	Optional	Form
UISA			X

Store Half Word Indexed

sthx **r**S,**r**A,**r**B




```
if rA = 0 then b ← 0
elseb ← (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:

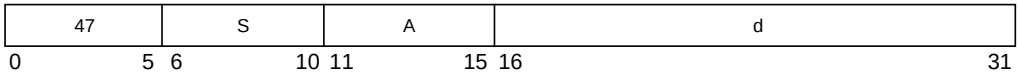
- None

PowerPC Architecture Level	Supervisor Level	Optional	Form
UISA			X

stmw stmw

Store Multiple Word

stmw **r**S,d**(r**A**)**




```
if rA = 0 then b ← 0
elseb ← (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.

Note that 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.

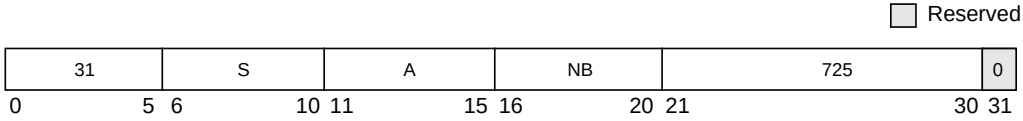
Other registers altered:

- None

PowerPC Architecture Level	Supervisor Level	Optional	Form
UISA			D

Store String Word Immediate

stswi **r**S,**r**A,NB




```
if rA = 0 then EA ← 0
elseEA ← (rA)
if NB = 0 then n ← 32
elsen ← NB
r ← rS – 1
i ← 32
do while n > 0
 if i = 32 then r ← r + 1 (mod 32)
 MEM(EA, 1) ← GPR(r)[i–i + 7]
 i ← i + 8
 if i = 64 then i ← 32
 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.

Note that, 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.

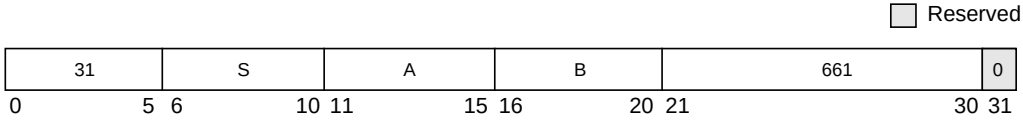
Other registers altered:

- None

PowerPC Architecture Level	Supervisor Level	Optional	Form
UISA			X

Store String Word Indexed

stswx **r**S,**r**A,**r**B




```
if rA = 0 then b ← 0
elseb ← (rA)
EA ← b + (rB)
n ← XER[25–31]
r ← rS – 1
i ← 32
do while n > 0
 if i = 32 then r ← r + 1 (mod 32)
 MEM(EA, 1) ← GPR(r)[i–i + 7]
 i ← i + 8
 if i = 64 then i ← 32
 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.

Note that, 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.

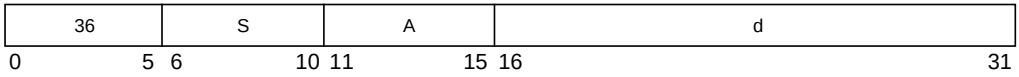
Other registers altered:

- None

PowerPC Architecture Level	Supervisor Level	Optional	Form
UISA			X

Store Word

stw **r**S,d**(r**A**)**




```
if rA = 0 then b ← 0
elseb ← (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:

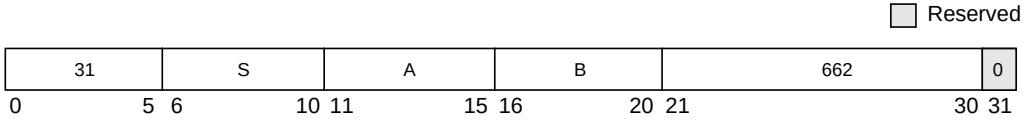
- None

PowerPC Architecture Level	Supervisor Level	Optional	Form
UISA			D

stwbrx stwbrx

Store Word Byte-Reverse Indexed

stwbrx **r**S,**r**A,**r**B




```
if rA = 0 then b ← 0
elseb ← (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 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 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 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 of **r**S are stored into bits 24–31 of the word in memory addressed by EA.

Other registers altered:

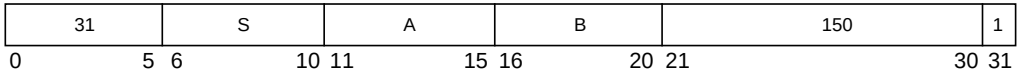
- None

PowerPC Architecture Level	Supervisor Level	Optional	Form
UISA			X

stwcx. stwcx.

Store Word Conditional Indexed

stwcx. **r**S,**r**A,**r**B




```
if rA = 0 then b ← 0
else b ← (rA)
EA ← b + (rB)
if RESERVE then
 if RESERVE_ADDR = physical_addr(EA)
 MEM(EA, 4) ← rS
 CR0 ← 0b00 || 0b1 || XER[SO]
 else
 u ← undefined 1-bit value
 if u then MEM(EA, 4) ← rS
 CR0 ← 0b00 || u || 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 store to the block indicated in the reservation buffer when reserved is set.

If a reservation exists, and the memory address specified by the **stwcx.** instruction is the same as that specified by the load and reserve instruction that established the reservation, the contents of **r**S are stored into the word in memory addressed by EA and the reservation is cleared.

If a reservation exists, but the memory address specified by the **stwcx.** instruction is not the same as that specified by the load and reserve instruction that established the reservation, the reservation is cleared, and it is undefined whether the contents of **r**S are stored into the word in memory addressed by EA.

If no reservation exists, the instruction completes without altering memory.

CR0 field is set to reflect whether the store operation was performed as follows.

```
CR0[LT GT EQ S0] = 0b00 || store_performed || XER[SO]
```

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.

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 allocated by a system library program.

Other registers altered:

- Condition Register (CR0 field):

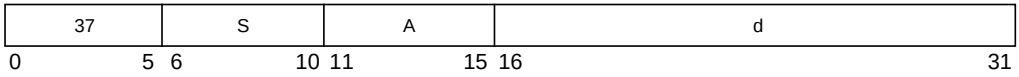
  Affected: LT, GT, EQ, SO

PowerPC Architecture Level	Supervisor Level	Optional	Form
UISA			X

stwu stwu

Store Word with Update

stwu rS,d(rA)




```
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:

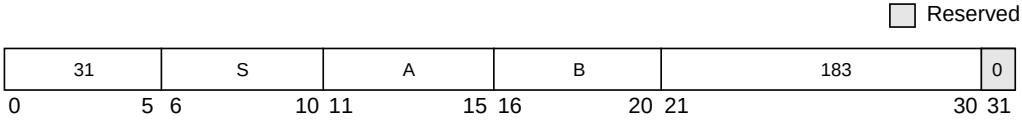
- None

PowerPC Architecture Level	Supervisor Level	Optional	Form
UISA			D

stwux stwux

Store Word with Update Indexed

stwux **r**S,**r**A,**r**B



EA ← (**r**A) + (**r**B)
MEM(EA, 4) ← **r**S
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.

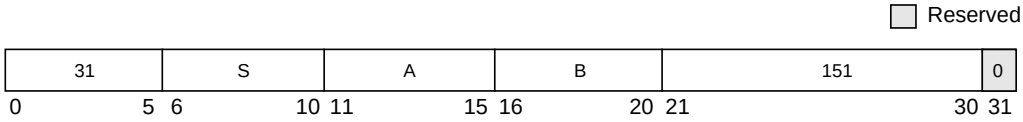
Other registers altered:

- None

PowerPC Architecture Level	Supervisor Level	Optional	Form
UISA			X

Store Word Indexed

stwx **r**S,**r**A,**r**B




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

EA is the sum (**r**A|0) + (**r**B). The contents of **r**S are is stored into the word in memory addressed by EA.

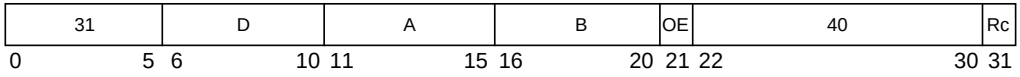
Other registers altered:

- None

PowerPC Architecture Level	Supervisor Level	Optional	Form
UISA			X

Subtract From

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. 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)

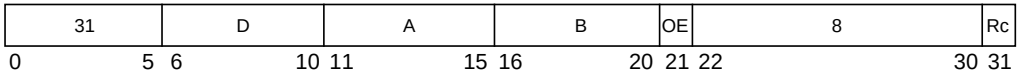
Simplified mnemonics:

sub **r**D**,r**A**,r**B equivalent to **subf** **r**D**,r**B**,r**A

PowerPC Architecture Level	Supervisor Level	Optional	Form
UISA			XO

Subtract from Carrying

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 ← ¬ (**r**A) + (**r**B) + 1

The sum ¬ (**r**A) + (**r**B) + 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 “true” (infinitely precise) result if overflow occurs (see XER below).

- XER:

  Affected: CA

  Affected: SO, OV (if OE = 1)

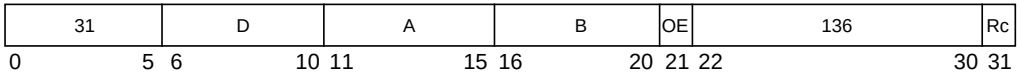
Simplified mnemonics:

subc **r**D**,r**A**,r**B equivalent to **subfc** **r**D**,r**B**,r**A

PowerPC Architecture Level	Supervisor Level	Optional	Form
UISA			XO

Subtract from Extended

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 ← ¬ (**r**A) + (**r**B) + 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 “true” (infinitely precise) result if overflow occurs (see XER below).

- XER:

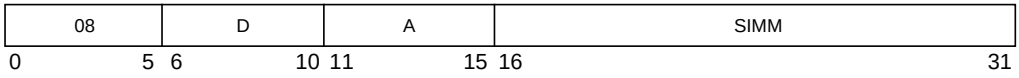
  Affected: CA

  Affected: SO, OV(if OE = 1)

PowerPC Architecture Level	Supervisor Level	Optional	Form
UISA			XO

Subtract from Immediate Carrying

subfic **r**D**,r**A**,**SIMM




```
rD ← ¬ (rA) + EXTS(SIMM) + 1
```

The sum ¬ (**r**A) + EXTS(SIMM) + 1 is placed into **r**D.

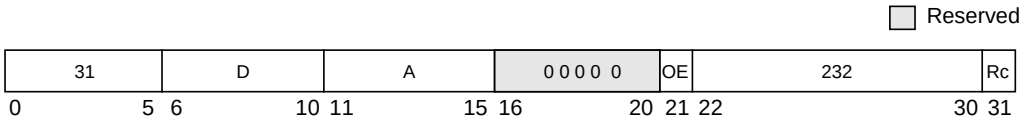
Other registers altered:

- XER:
  Affected: CA

PowerPC Architecture Level	Supervisor Level	Optional	Form
UISA			D

Subtract from Minus One Extended

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 “true” (infinitely precise) result if overflow occurs (see XER below).

- XER:

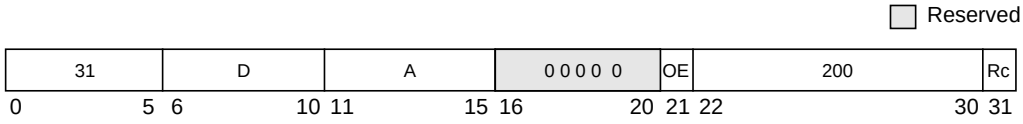
  Affected: CA

  Affected: SO, OV(if OE = 1)

PowerPC Architecture Level	Supervisor Level	Optional	Form
UISA			XO

Subtract from Zero Extended

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 ← ¬ (**r**A) + 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 “true” (infinitely precise) result if overflow occurs (see XER below).

- XER:

  Affected: CA

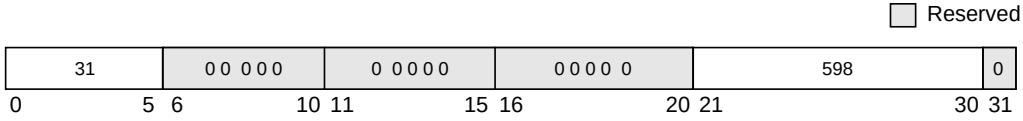
  Affected: SO, OV(if OE = 1)

PowerPC Architecture Level	Supervisor Level	Optional	Form
UISA			XO

sync

Synchronize

sync



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.

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.

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.

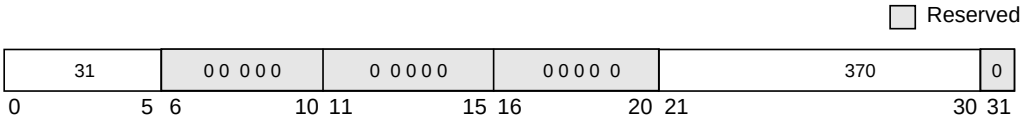
The functions performed by the **sync** instruction will normally take a significant amount of time to complete, so indiscriminate use of this instruction may adversely affect performance. In addition, the time required to execute **sync** may vary from one execution to another. The **eieio** instruction may be more appropriate than **sync** for many cases.

Other registers altered:

- None

PowerPC Architecture Level	Supervisor Level	Optional	Form
UISA			X

Translation Lookaside Buffer Invalidate All




```
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 SLB, segment table, or segment registers. This instruction does not cause the entries to be invalidated in other processors. This is a supervisor-level instructon.

Other registers altered:

- None

PowerPC Architecture Level	Supervisor Level	Optional	Form
OEA	√	√	X

Translation Lookaside Buffer Invalidate Entry

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.

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.

This is a supervisor-level instruction.

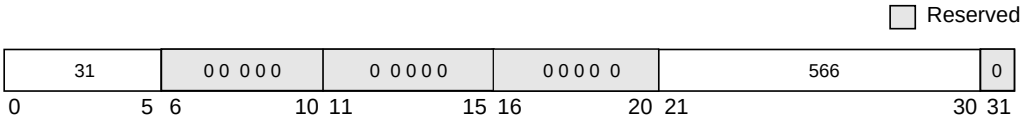
Other registers altered:

- None

PowerPC Architecture Level	Supervisor Level	Optional	Form
OEA	√	√	X

tlbsync tlbsync

TLB Synchronize



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**. This instruction is supervisor-level.

Other registers altered:

- None

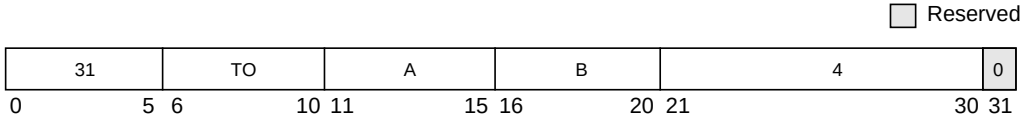
PowerPC Architecture Level	Supervisor Level	Optional	Form
OEA	√	√	X

tw

tw

Trap Word

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 with the contents of **r**B. 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:

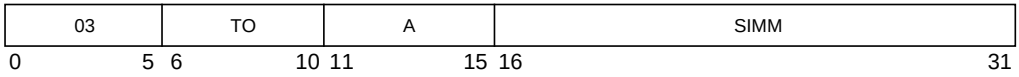
tweq	**r**A,**r**B	equivalent to	**tw**	**4,r**A,**r**B
twlge	**r**A,**r**B	equivalent to	**tw**	**5,r**A,**r**B
trap		equivalent to	**tw**	**31,0,0**

PowerPC Architecture Level	Supervisor Level	Optional	Form
UISA			X

twi twi

Trap Word Immediate

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 with the sign-extended value of the SIMM field. 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

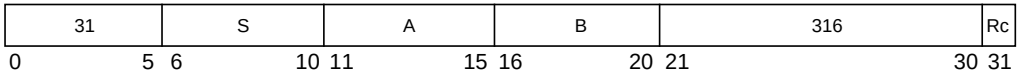
Simplified mnemonics:

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	Optional	Form
UISA			D

XOR

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



$$\mathbf{r}A \leftarrow (\mathbf{r}S) \oplus (\mathbf{r}B)$$

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

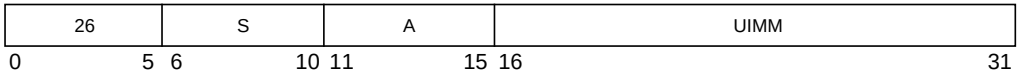
Other registers altered:

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

PowerPC Architecture Level	Supervisor Level	Optional	Form
UISA			X

XOR Immediate

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.

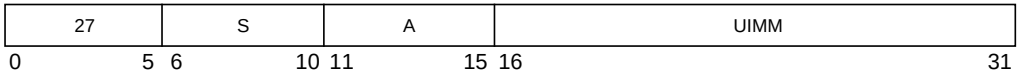
Other registers altered:

- None

PowerPC Architecture Level	Supervisor Level	Optional	Form
UISA			D

XOR Immediate Shifted

xoris **r**A,**r**S,UIMM



rA ← (**r**S) ⊕ (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	Optional	Form
UISA			D

Appendix
MPCxxx Instruction Set Listings

This appendix lists the MPCxxx's instruction set. Instructions are sorted by *mnemonic*, opcode, function, and form. Also included in this appendix is a quick reference table that contains general information, such as the architecture level, privilege level, and form.

Note that split fields, which represent the concatenation of sequences from left to right, are shown in lowercase.

Instructions Sorted by Mnemonic

Table 1 lists the instructions implemented in the MPCxxx in alphabetical order by mnemonic.

Key:

Reserved bits

Table 1. Complete Instruction List Sorted by Mnemonic

Name	0	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
add*x*	31	D					A					B					OE	266								Rc
addc*x*	31	D					A					B					OE	10								Rc
adde*x*	31	D					A					B					OE	138								Rc
addi	14	D					A					SIMM															
addic	12	D					A					SIMM															
addic.	13	D					A					SIMM															
addis	15	D					A					SIMM															
addme*x*	31	D					A					0 0 0 0 0					OE	234								Rc
addze*x*	31	D					A					0 0 0 0 0					OE	202								Rc
and*x*	31	S					A					B					28									Rc
andc*x*	31	S					A					B					60									Rc
andi.	28	S					A					UIMM															
andis.	29	S					A					UIMM															

Name	0	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
b*x*	18	LI																								AA	LK
bc*x*	16	BO					BI					BD														AA	LK
bcctr*x*	19	BO					BI					0 0 0 0 0					528										LK
bclr*x*	19	BO					BI					0 0 0 0 0					16										LK
cmp	31	crfD			0	L	A					B					0										0
cmpi	11	crfD			0	L	A					SIMM															
cmpl	31	crfD			0	L	A					B					32										0
cmpli	10	crfD			0	L	A					UIMM															
cntlzw*x*	31	S					A					0 0 0 0 0					26										Rc
crand	19	crbD					crbA					crbB					257										0
crandc	19	crbD					crbA					crbB					129										0
creqv	19	crbD					crbA					crbB					289										0
crnand	19	crbD					crbA					crbB					225										0
crnor	19	crbD					crbA					crbB					33										0
cror	19	crbD					crbA					crbB					449										0
crorc	19	crbD					crbA					crbB					417										0
crxor	19	crbD					crbA					crbB					193										0
dcbf	31	0 0 0 0 0					A					B					86										0
dcbi [1]	31	0 0 0 0 0					A					B					470										0
dcbst	31	0 0 0 0 0					A					B					54										0
dcbt	31	0 0 0 0 0					A					B					278										0
dcbtst	31	0 0 0 0 0					A					B					246										0
dcbz	31	0 0 0 0 0					A					B					1014										0
divw*x*	31	D					A					B					OE	491									Rc
divwu*x*	31	D					A					B					OE	459									Rc
eciwx	31	D					A					B					310										0
ecowx	31	S					A					B					438										0
eieio	31	0 0 0 0 0					0 0 0 0 0					0 0 0 0 0					854										0
eqv*x*	31	S					A					B					284										Rc
extsb*x*	31	S					A					0 0 0 0 0					954										Rc
extsh*x*	31	S					A					0 0 0 0 0					922										Rc
icbi	31	0 0 0 0 0					A					B					982										0
isync	19	0 0 0 0 0					0 0 0 0 0					0 0 0 0 0					150										0

Name	0	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
lbz	34	D					A					d															
lbzu	35	D					A					d															
lbzux	31	D					A					B					119										0
lbzx	31	D					A					B					87										0
lha	42	D					A					d															
lhau	43	D					A					d															
lhaux	31	D					A					B					375										0
lhax	31	D					A					B					343										0
lhbrx	31	D					A					B					790										0
lhz	40	D					A					d															
lhzu	41	D					A					d															
lhzux	31	D					A					B					311										0
lhzx	31	D					A					B					279										0
lmw [3]	46	D					A					d															
lswi [3]	31	D					A					NB					597										0
lswx [3]	31	D					A					B					533										0
lwarx	31	D					A					B					20										0
lwbrx	31	D					A					B					534										0
lwz	32	D					A					d															
lwzu	33	D					A					d															
lwzux	31	D					A					B					55										0
lwzx	31	D					A					B					23										0
mcrf	19	crfD			0	0	crfS			0	0	0	0	0	0	0	0	0									0
mcrxr	31	crfD			0	0	0	0	0	0	0	0	0	0	0	0	0	0									0
mfcr	31	D					0	0	0	0	0	0	0	0	0	0	0	0									0
mfmsr [1]	31	D					0	0	0	0	0	0	0	0	0	0	0	0									0
mfspr [2]	31	D					spr																				0
mfsr [1]	31	D					0	SR					0	0	0	0	0	0									0
mfsrin [1]	31	D					0	0	0	0	0	B															0
mftb	31	D					tbr																				0
mtcrf	31	S					0	CRM									0										0
mtmsr [1]	31	S					0	0	0	0	0	0	0	0	0	0	0	0									0
mtspr [2]	31	S					spr																				0

Name	0	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
mtsr [1]	31	S					0	SR				0	0	0	0	0	210										0
mtsrin [1]	31	S					0	0	0	0	0	B					242										0
mulhw*x*	31	D					A					B					0	75									Rc
mulhwu*x*	31	D					A					B					0	11									Rc
mulli	7	D					A					SIMM															
mullw*x*	31	D					A					B					OE	235									Rc
nand*x*	31	S					A					B					476										Rc
neg*x*	31	D					A					0	0	0	0	0	OE	104									Rc
nor*x*	31	S					A					B					124										Rc
or*x*	31	S					A					B					444										Rc
orc*x*	31	S					A					B					412										Rc
ori	24	S					A					UIMM															
oris	25	S					A					UIMM															
rfi [1]	19	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	50										0
rlwimi*x*	20	S					A					SH					MB					ME					Rc
rlwinm*x*	21	S					A					SH					MB					ME					Rc
rlwnm*x*	23	S					A					B					MB					ME					Rc
sc	17	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
slw*x*	31	S					A					B					24										Rc
sraw*x*	31	S					A					B					792										Rc
srawi*x*	31	S					A					SH					824										Rc
srw*x*	31	S					A					B					536										Rc
stb	38	S					A					d															
stbu	39	S					A					d															
stbux	31	S					A					B					247										0
stbx	31	S					A					B					215										0
sth	44	S					A					d															
sthbrx	31	S					A					B					918										0
sthu	45	S					A					d															
sthux	31	S					A					B					439										0
sthx	31	S					A					B					407										0
stmw [3]	47	S					A					d															
stswi [3]	31	S					A					NB					725										0

Name	0	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
stswx [3]	31	S					A					B					661										0
stw	36	S					A					d															
stwbrx	31	S					A					B					662										0
stwcx.	31	S					A					B					150										1
stwu	37	S					A					d															
stwux	31	S					A					B					183										0
stwx	31	S					A					B					151										0
subf*x*	31	D					A					B					OE	40									Rc
subfc*x*	31	D					A					B					OE	8									Rc
subfe*x*	31	D					A					B					OE	136									Rc
subfic	08	D					A					SIMM															
subfme*x*	31	D					A					0 0 0 0 0					OE	232									Rc
subfze*x*	31	D					A					0 0 0 0 0					OE	200									Rc
sync	31	0 0 0 0 0					0 0 0 0 0					0 0 0 0 0					598										0
tlbia [1,4]	31	0 0 0 0 0					0 0 0 0 0					0 0 0 0 0					370										0
tlbie [1,4]	31	0 0 0 0 0					0 0 0 0 0					B					306										0
tlbsync[1,4]	31	0 0 0 0 0					0 0 0 0 0					0 0 0 0 0					566										0
tw	31	TO					A					B					4										0
twi	03	TO					A					SIMM															
xor*x*	31	S					A					B					316										Rc
xori	26	S					A					UIMM															
xoris	27	S					A					UIMM															

[1] Supervisor-level instruction
[2] Supervisor- and user-level instruction
[3] Load and store string or multiple instruction
[4] PowerPC Optional instruction

Instructions Sorted by Opcode

Table 2 lists the instructions defined for the MPCxxx in numeric order by opcode.

Key:

Reserved bits

Table 2. Complete Instruction List Sorted by Opcode

Name	0 5	6 7 8	9 10	11 12 13 14 15	16 17 18 19 20	21 22 23 24 25	26 27 28 29 30	31
twi	0 0 0 0 1 1	TO		A	SIMM			
mulli	0 0 0 1 1 1	D		A	SIMM			
subfic	0 0 1 0 0 0	D		A	SIMM			
cmpli	0 0 1 0 1 0	crfD	0 L	A	UIMM			
cmpi	0 0 1 0 1 1	crfD	0 L	A	SIMM			
addic	0 0 1 1 0 0	D		A	SIMM			
addic.	0 0 1 1 0 1	D		A	SIMM			
addi	0 0 1 1 1 0	D		A	SIMM			
addis	0 0 1 1 1 1	D		A	SIMM			
bc*x*	0 1 0 0 0 0	BO		BI	BD		AA	LK
sc	0 1 0 0 0 1	0 0 0 0 0		0 0 0 0 0	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0		1	0
b*x*	0 1 0 0 1 0	LI					AA	LK
mcrf	0 1 0 0 1 1	crfD	0 0	crfS 0 0	0 0 0 0 0	0 0 0 0 0 0 0 0 0 0		0
bclr*x*	0 1 0 0 1 1	BO		BI	0 0 0 0 0	0 0 0 0 0 1 0 0 0 0		LK
crnor	0 1 0 0 1 1	crbD		crbA	crbB	0 0 0 0 1 0 0 0 0 1		0
rfi [1]	0 1 0 0 1 1	0 0 0 0 0		0 0 0 0 0	0 0 0 0 0	0 0 0 0 1 1 0 0 1 0		0
crandc	0 1 0 0 1 1	crbD		crbA	crbB	0 0 1 0 0 0 0 0 0 1		0
isync	0 1 0 0 1 1	0 0 0 0 0		0 0 0 0 0	0 0 0 0 0	0 0 1 0 0 1 0 1 1 0		0
crxor	0 1 0 0 1 1	crbD		crbA	crbB	0 0 1 1 0 0 0 0 0 1		0
crnand	0 1 0 0 1 1	crbD		crbA	crbB	0 0 1 1 1 0 0 0 0 1		0
crand	0 1 0 0 1 1	crbD		crbA	crbB	0 1 0 0 0 0 0 0 0 1		0
creqv	0 1 0 0 1 1	crbD		crbA	crbB	0 1 0 0 1 0 0 0 0 1		0
crorc	0 1 0 0 1 1	crbD		crbA	crbB	0 1 1 0 1 0 0 0 0 1		0
cror	0 1 0 0 1 1	crbD		crbA	crbB	0 1 1 1 0 0 0 0 0 1		0
bcctr*x*	0 1 0 0 1 1	BO		BI	0 0 0 0 0	1 0 0 0 0 1 0 0 0 0		LK
rlwimi*x*	0 1 0 1 0 0	S		A	SH	MB	ME	Rc
rlwinm*x*	0 1 0 1 0 1	S		A	SH	MB	ME	Rc

Name	0	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
rlwnm*x*	0 1 0 1 1 1		S					A					B					MB					ME					Rc
ori	0 1 1 0 0 0		S					A					UIMM															
oris	0 1 1 0 0 1		S					A					UIMM															
xori	0 1 1 0 1 0		S					A					UIMM															
xoris	0 1 1 0 1 1		S					A					UIMM															
andi.	0 1 1 1 0 0		S					A					UIMM															
andis.	0 1 1 1 0 1		S					A					UIMM															
cmp	0 1 1 1 1 1		crfD			0	L	A					B					0 0 0 0 0 0 0 0 0 0										0
tw	0 1 1 1 1 1		TO					A					B					0 0 0 0 0 0 0 1 0 0										0
subfc*x*	0 1 1 1 1 1		D					A					B					OE	0 0 0 0 0 0 0 1 0 0 0									Rc
addc*x*	0 1 1 1 1 1		D					A					B					OE	0 0 0 0 0 0 0 1 0 1 0									Rc
mulhwu*x*	0 1 1 1 1 1		D					A					B					0	0 0 0 0 0 0 0 1 0 1 1									Rc
mfcr	0 1 1 1 1 1		D					0 0 0 0 0					0 0 0 0 0					0 0 0 0 0 0 1 0 0 1 1										0
lwarx	0 1 1 1 1 1		D					A					B					0 0 0 0 0 1 0 1 0 0										0
lwzx	0 1 1 1 1 1		D					A					B					0 0 0 0 0 1 0 1 1 1										0
slw*x*	0 1 1 1 1 1		S					A					B					0 0 0 0 0 1 1 0 0 0										Rc
cntlzw*x*	0 1 1 1 1 1		S					A					0 0 0 0 0					0 0 0 0 0 1 1 0 1 0										Rc
and*x*	0 1 1 1 1 1		S					A					B					0 0 0 0 0 1 1 1 0 0										Rc
cmpl	0 1 1 1 1 1		crfD			0	L	A					B					0 0 0 0 1 0 0 0 0 0										0
subf*x*	0 1 1 1 1 1		D					A					B					OE	0 0 0 0 1 0 1 0 0 0									Rc
dcbst	0 1 1 1 1 1		0 0 0 0 0					A					B					0 0 0 0 1 1 0 1 1 0										0
lwzux	0 1 1 1 1 1		D					A					B					0 0 0 0 1 1 0 1 1 1										0
andc*x*	0 1 1 1 1 1		S					A					B					0 0 0 0 1 1 1 1 0 0										Rc
mulhw*x*	0 1 1 1 1 1		D					A					B					0	0 0 0 1 0 0 1 0 1 1									Rc
mfmsr [1]	0 1 1 1 1 1		D					0 0 0 0 0					0 0 0 0 0					0 0 0 1 0 1 0 0 1 1										0
dcbf	0 1 1 1 1 1		0 0 0 0 0					A					B					0 0 0 1 0 1 0 1 1 0										0
lbzx	0 1 1 1 1 1		D					A					B					0 0 0 1 0 1 0 1 1 1										0
neg*x*	0 1 1 1 1 1		D					A					0 0 0 0 0					OE	0 0 0 1 1 0 1 0 0 0									Rc
lbzux	0 1 1 1 1 1		D					A					B					0 0 0 1 1 1 0 1 1 1										0
nor*x*	0 1 1 1 1 1		S					A					B					0 0 0 1 1 1 1 1 0 0										Rc
subfe*x*	0 1 1 1 1 1		D					A					B					OE	0 0 1 0 0 0 1 0 0 0									Rc
adde*x*	0 1 1 1 1 1		D					A					B					OE	0 0 1 0 0 0 1 0 1 0									Rc
mtcrf	0 1 1 1 1 1		S					0	CRM								0	0 0 1 0 0 1 0 0 0 0										0

Name	0	1 2 3 4 5	6	7 8 9 10	11 12 13 14 15	16 17 18 19 20	21	22 23 24 25 26 27 28 29 30	31
mtmsr [1]	0	1 1 1 1 1	S		0 0 0 0 0	0 0 0 0 0		0 0 1 0 0 1 0 0 1 0	0
stwcx.	0	1 1 1 1 1	S		A	B		0 0 1 0 0 1 0 1 1 0	1
stwx	0	1 1 1 1 1	S		A	B		0 0 1 0 0 1 0 1 1 1	0
stwux	0	1 1 1 1 1	S		A	B		0 0 1 0 1 1 0 1 1 1	0
subfze*x*	0	1 1 1 1 1	D		A	0 0 0 0 0	OE	0 0 1 1 0 0 1 0 0 0	Rc
addze*x*	0	1 1 1 1 1	D		A	0 0 0 0 0	OE	0 0 1 1 0 0 1 0 1 0	Rc
mtsr [1]	0	1 1 1 1 1	S		0 SR	0 0 0 0 0		0 0 1 1 0 1 0 0 1 0	0
stbx	0	1 1 1 1 1	S		A	B		0 0 1 1 0 1 0 1 1 1	0
subfme*x*	0	1 1 1 1 1	D		A	0 0 0 0 0	OE	0 0 1 1 1 0 1 0 0 0	Rc
addme*x*	0	1 1 1 1 1	D		A	0 0 0 0 0	OE	0 0 1 1 1 0 1 0 1 0	Rc
mullw*x*	0	1 1 1 1 1	D		A	B	OE	0 0 1 1 1 0 1 0 1 1	Rc
mtsrin [1]	0	1 1 1 1 1	S		0 0 0 0 0	B		0 0 1 1 1 1 0 0 1 0	0
dcbtst	0	1 1 1 1 1	0 0 0 0 0		A	B		0 0 1 1 1 1 0 1 1 0	0
stbux	0	1 1 1 1 1	S		A	B		0 0 1 1 1 1 0 1 1 1	0
add*x*	0	1 1 1 1 1	D		A	B	OE	0 1 0 0 0 0 1 0 1 0	Rc
dcbt	0	1 1 1 1 1	0 0 0 0 0		A	B		0 1 0 0 0 1 0 1 1 0	0
lhzx	0	1 1 1 1 1	D		A	B		0 1 0 0 0 1 0 1 1 1	0
eqv*x*	0	1 1 1 1 1	S		A	B		0 1 0 0 0 1 1 1 0 0	Rc
tlbie [1,4]	0	1 1 1 1 1	0 0 0 0 0		0 0 0 0 0	B		0 1 0 0 1 1 0 0 1 0	0
eciwx	0	1 1 1 1 1	D		A	B		0 1 0 0 1 1 0 1 1 0	0
lhzux	0	1 1 1 1 1	D		A	B		0 1 0 0 1 1 0 1 1 1	0
xor*x*	0	1 1 1 1 1	S		A	B		0 1 0 0 1 1 1 1 0 0	Rc
mfspr [2]	0	1 1 1 1 1	D		spr			0 1 0 1 0 1 0 0 1 1	0
lhax	0	1 1 1 1 1	D		A	B		0 1 0 1 0 1 0 1 1 1	0
tlbia [1,4]	0	1 1 1 1 1	0 0 0 0 0		0 0 0 0 0	0 0 0 0 0		0 1 0 1 1 1 0 0 1 0	0
mftb	0	1 1 1 1 1	D		tbr			0 1 0 1 1 1 0 0 1 1	0
lhaux	0	1 1 1 1 1	D		A	B		0 1 0 1 1 1 0 1 1 1	0
sthx	0	1 1 1 1 1	S		A	B		0 1 1 0 0 1 0 1 1 1	0
orc*x*	0	1 1 1 1 1	S		A	B		0 1 1 0 0 1 1 1 0 0	Rc
ecowx	0	1 1 1 1 1	S		A	B		0 1 1 0 1 1 0 1 1 0	0
sthux	0	1 1 1 1 1	S		A	B		0 1 1 0 1 1 0 1 1 1	0
or*x*	0	1 1 1 1 1	S		A	B		0 1 1 0 1 1 1 1 0 0	Rc
divwu*x*	0	1 1 1 1 1	D		A	B	OE	0 1 1 1 0 0 1 0 1 1	Rc

Name	0 5	6 7 8 9 10	11 12 13 14 15	16 17 18 19 20	21 22 23 24 25 26 27 28 29 30	31
mtspr [2]	0 1 1 1 1 1	S	spr		0 1 1 1 0 1 0 0 1 1	0
dcbi [1]	0 1 1 1 1 1	0 0 0 0 0	A	B	0 1 1 1 0 1 0 1 1 0	0
nand*x*	0 1 1 1 1 1	S	A	B	0 1 1 1 0 1 1 1 0 0	Rc
divw*x*	0 1 1 1 1 1	D	A	B	OE 0 1 1 1 1 0 1 0 1 1	Rc
mcrxr	0 1 1 1 1 1	crfD 0 0	0 0 0 0 0	0 0 0 0 0	1 0 0 0 0 0 0 0 0 0	0
lswx [3]	0 1 1 1 1 1	D	A	B	1 0 0 0 0 1 0 1 0 1	0
lwbrx	0 1 1 1 1 1	D	A	B	1 0 0 0 0 1 0 1 1 0	0
lfsx	0 1 1 1 1 1	D	A	B	1 0 0 0 0 1 0 1 1 1	0
srw*x*	0 1 1 1 1 1	S	A	B	1 0 0 0 0 1 1 0 0 0	Rc
tlbsync [1,4]	0 1 1 1 1 1	0 0 0 0 0	0 0 0 0 0	0 0 0 0 0	1 0 0 0 1 1 0 1 1 0	0
mfsr [1]	0 1 1 1 1 1	D	0 SR	0 0 0 0 0	1 0 0 1 0 1 0 0 1 1	0
lswi [3]	0 1 1 1 1 1	D	A	NB	1 0 0 1 0 1 0 1 0 1	0
sync	0 1 1 1 1 1	0 0 0 0 0	0 0 0 0 0	0 0 0 0 0	1 0 0 1 0 1 0 1 1 0	0
lfdx	0 1 1 1 1 1	D	A	B	1 0 0 1 0 1 0 1 1 1	0
lfdux	0 1 1 1 1 1	D	A	B	1 0 0 1 1 1 0 1 1 1	0
mfsrin [1]	0 1 1 1 1 1	D	0 0 0 0 0	B	1 0 1 0 0 1 0 0 1 1	0
stswx [3]	0 1 1 1 1 1	S	A	B	1 0 1 0 0 1 0 1 0 1	0
stwbrx	0 1 1 1 1 1	S	A	B	1 0 1 0 0 1 0 1 1 0	0
stswi [3]	0 1 1 1 1 1	S	A	NB	1 0 1 1 0 1 0 1 0 1	0
lhbrx	0 1 1 1 1 1	D	A	B	1 1 0 0 0 1 0 1 1 0	0
sraw*x*	0 1 1 1 1 1	S	A	B	1 1 0 0 0 1 1 0 0 0	Rc
srawi*x*	0 1 1 1 1 1	S	A	SH	1 1 0 0 1 1 1 0 0 0	Rc
eieio	0 1 1 1 1 1	0 0 0 0 0	0 0 0 0 0	0 0 0 0 0	1 1 0 1 0 1 0 1 1 0	0
sthbrx	0 1 1 1 1 1	S	A	B	1 1 1 0 0 1 0 1 1 0	0
extsh*x*	0 1 1 1 1 1	S	A	0 0 0 0 0	1 1 1 0 0 1 1 0 1 0	Rc
extsb*x*	0 1 1 1 1 1	S	A	0 0 0 0 0	1 1 1 0 1 1 1 0 1 0	Rc
icbi	0 1 1 1 1 1	0 0 0 0 0	A	B	1 1 1 1 0 1 0 1 1 0	0
dcbz	0 1 1 1 1 1	0 0 0 0 0	A	B	1 1 1 1 1 1 0 1 1 0	0
lwz	1 0 0 0 0 0	D	A	d		
lwzu	1 0 0 0 0 1	D	A	d		
lbz	1 0 0 0 1 0	D	A	d		
lbzu	1 0 0 0 1 1	D	A	d		
stw	1 0 0 1 0 0	S	A	d		

Name	0 5	6 7	8 9 10 11 12 13 14 15	16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
stwu	1 0 0 1 0 1	S	A	d
stb	1 0 0 1 1 0	S	A	d
stbu	1 0 0 1 1 1	S	A	d
lhz	1 0 1 0 0 0	D	A	d
lhzu	1 0 1 0 0 1	D	A	d
lha	1 0 1 0 1 0	D	A	d
lhau	1 0 1 0 1 1	D	A	d
sth	1 0 1 1 0 0	S	A	d
sthu	1 0 1 1 0 1	S	A	d
lmw [3]	1 0 1 1 1 0	D	A	d
stmw [3]	1 0 1 1 1 1	S	A	d

[1] Supervisor-level instruction
[2] Supervisor- and user-level instruction
[3] Load and store string or multiple instruction
[4] PowerPC Optional instruction

Instructions Grouped by Functional Categories

Tables 3 through 30 list the PowerPC instructions grouped by function.

Key:

Reserved bits

Table 3. Integer Arithmetic Instructions

Name	0 5	6 7 8 9 10	11 12 13 14 15	16 17 18 19 20	21	22 23 24 25 26 27 28 29 30	31
add*x*	31	D	A	B	OE	266	Rc
addc*x*	31	D	A	B	OE	10	Rc
adde*x*	31	D	A	B	OE	138	Rc
addi	14	D	A	SIMM			
addic	12	D	A	SIMM			
addic.	13	D	A	SIMM			
addis	15	D	A	SIMM			
addme*x*	31	D	A	0 0 0 0 0	OE	234	Rc
addze*x*	31	D	A	0 0 0 0 0	OE	202	Rc
divw*x*	31	D	A	B	OE	491	Rc
divwu*x*	31	D	A	B	OE	459	Rc
mulhw*x*	31	D	A	B	0	75	Rc
mulhwu*x*	31	D	A	B	0	11	Rc
mulli	07	D	A	SIMM			
mullw*x*	31	D	A	B	OE	235	Rc
neg*x*	31	D	A	0 0 0 0 0	OE	104	Rc
subf*x*	31	D	A	B	OE	40	Rc
subfc*x*	31	D	A	B	OE	8	Rc
subfic*x*	08	D	A	SIMM			
subfe*x*	31	D	A	B	OE	136	Rc
subfme*x*	31	D	A	0 0 0 0 0	OE	232	Rc
subfze*x*	31	D	A	0 0 0 0 0	OE	200	Rc

Table 4. Integer Compare Instructions

Name	0 – 5	6 – 7	8	9	10	11 – 15	16 – 20	21 – 30	31
cmp	31	crfD	0	L	A	B	0 0 0 0 0 0 0 0 0 0		0
cmpi	11	crfD	0	L	A	SIMM			
cmpl	31	crfD	0	L	A	B	32		0
cmpli	10	crfD	0	L	A	UIMM			

Table 5. Integer Logical Instructions

Name	0 – 5	6 – 7	8 – 9	10	11 – 15	16 – 20	21 – 30	31
and*x*	31	S		A	B	28		Rc
andc*x*	31	S		A	B	60		Rc
andi.	28	S		A	UIMM			
andis.	29	S		A	UIMM			
cntlzw*x*	31	S		A	0 0 0 0 0	26		Rc
eqv*x*	31	S		A	B	284		Rc
extsb*x*	31	S		A	0 0 0 0 0	954		Rc
extsh*x*	31	S		A	0 0 0 0 0	922		Rc
nand*x*	31	S		A	B	476		Rc
nor*x*	31	S		A	B	124		Rc
or*x*	31	S		A	B	444		Rc
orc*x*	31	S		A	B	412		Rc
ori	24	S		A	UIMM			
oris	25	S		A	UIMM			
xor*x*	31	S		A	B	316		Rc
xori	26	S		A	UIMM			
xoris	27	S		A	UIMM			

Table 6. Integer Rotate Instructions

Name	0 – 5	6 – 7	8 – 9	10	11 – 15	16 – 20	21 – 25	26 – 30	31
rlwimi*x*	22	S		A	SH	MB	ME		Rc
rlwinm*x*	20	S		A	SH	MB	ME		Rc
rlwnm*x*	21	S		A	SH	MB	ME		Rc

Table 7. Integer Shift Instructions

Name	0	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
slw*x*	31		S					A					B					24										Rc
sraw*x*	31		S					A					B					792										Rc
srawi*x*	31		S					A					SH					824										Rc
srw*x*	31		S					A					B					536										Rc

Table 8. Integer Load Instructions

Name	0	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
lbz	34		D					A					d															
lbzu	35		D					A					d															
lbzux	31		D					A					B					119										0
lbzx	31		D					A					B					87										0
lha	42		D					A					d															
lhau	43		D					A					d															
lhaux	31		D					A					B					375										0
lhax	31		D					A					B					343										0
lhz	40		D					A					d															
lhzu	41		D					A					d															
lhzux	31		D					A					B					311										0
lhzx	31		D					A					B					279										0
lwz	32		D					A					d															
lwzu	33		D					A					d															
lwzux	31		D					A					B					55										0
lwzx	31		D					A					B					23										0

Table 9. Integer Store Instructions

Name	0 5	6 7 8 9 10	11 12 13 14 15	16 17 18 19 20	21 22 23 24 25 26 27 28 29 30	31
stb	38	S	A	d		
stbu	39	S	A	d		
stbux	31	S	A	B	247	0
stbx	31	S	A	B	215	0
sth	44	S	A	d		
sthu	45	S	A	d		
sthux	31	S	A	B	439	0
sthx	31	S	A	B	407	0
stw	36	S	A	d		
stwu	37	S	A	d		
stwux	31	S	A	B	183	0
stwx	31	S	A	B	151	0

Table 10. Integer Load and Store with Byte Reverse Instructions

Name	0 5	6 7 8 9 10	11 12 13 14 15	16 17 18 19 20	21 22 23 24 25 26 27 28 29 30	31
lhbrx	31	D	A	B	790	0
lwbrx	31	D	A	B	534	0
sthbrx	31	S	A	B	918	0
stwbrx	31	S	A	B	662	0

Table 11. Integer Load and Store Multiple Instructions

Name	0 5	6 7 8 9 10	11 12 13 14 15	16 17 18 19 20	21 22 23 24 25 26 27 28 29 30	31
lmw [3]	46	D	A	d		
stmw [3]	47	S	A	d		

Table 12. Integer Load and Store String Instructions

Name	0 5	6 7 8 9 10	11 12 13 14 15	16 17 18 19 20	21 22 23 24 25 26 27 28 29 30	31
lswi [3]	31	D	A	NB	597	0
lswx [3]	31	D	A	B	533	0
stswi [3]	31	S	A	NB	725	0
stswx [3]	31	S	A	B	661	0

Table 13. Memory Synchronization Instructions

Name	0	5	6 7 8 9 10	11 12 13 14 15	16 17 18 19 20	21 22 23 24 25 26 27 28 29 30	31
eieio	31		0 0 0 0 0	0 0 0 0 0	0 0 0 0 0	854	0
isync	19		0 0 0 0 0	0 0 0 0 0	0 0 0 0 0	150	0
lwarx	31		D	A	B	20	0
stwcx.	31		S	A	B	150	1
sync	31		0 0 0 0 0	0 0 0 0 0	0 0 0 0 0	598	0

Table 14. Branch Instructions

Name	0	5	6 7 8 9 10	11 12 13 14 15	16 17 18 19 20	21 22 23 24 25 26 27 28 29 30	31
b*x*	18		LI			AA	LK
bc*x*	16		BO	BI	BD	AA	LK
bcctr*x*	19		BO	BI	0 0 0 0 0	528	LK
bclr*x*	19		BO	BI	0 0 0 0 0	16	LK

Table 15. Condition Register Logical Instructions

Name	0	5	6 7 8 9	10	11 12 13 14	15	16 17 18 19 20	21 22 23 24 25 26 27 28 29 30	31
crand	19		crbD		crbA		crbB	257	0
crandc	19		crbD		crbA		crbB	129	0
creqv	19		crbD		crbA		crbB	289	0
crnand	19		crbD		crbA		crbB	225	0
crnor	19		crbD		crbA		crbB	33	0
cror	19		crbD		crbA		crbB	449	0
crorc	19		crbD		crbA		crbB	417	0
crxor	19		crbD		crbA		crbB	193	0
mcrf	19		crfD	0 0	crfS	0 0	0 0 0 0 0	0 0 0 0 0 0 0 0 0 0	0

Table 16. System Linkage Instructions

Name	0	5	6 7 8 9 10	11 12 13 14 15	16 17 18 19 20	21 22 23 24 25 26 27 28 29 30	31
rfi [1]	19		0 0 0 0 0	0 0 0 0 0	0 0 0 0 0	50	0
sc	17		0 0 0 0 0	0 0 0 0 0	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	1	0

Table 17. Trap Instructions

Name	0	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
tw	31		TO					A					B					4										0
twi	03		TO					A					SIMM															

Table 18. Processor Control Instructions

Name	0	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
mcrxr	31		crfS			0	0	0	0	0	0	0	0	0	0	0	0	512										0
mfcr	31		D					0	0	0	0	0	0	0	0	0	0	19										0
mfmsr [1]	31		D					0	0	0	0	0	0	0	0	0	0	83										0
mfspr [2]	31		D					spr										339										0
mftb	31		D					tpr										371										0
mtcrf	31		S					0	CRM								0	144										0
mtmsr [1]	31		S					0	0	0	0	0	0	0	0	0	0	146										0
mtspr [2]	31		D					spr										467										0

Table 19. Cache Management Instructions

Name	0	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
dcbf	31		0	0	0	0	0	A					B					86										0
dcbi [1]	31		0	0	0	0	0	A					B					470										0
dcbst	31		0	0	0	0	0	A					B					54										0
dcbt	31		0	0	0	0	0	A					B					278										0
dcbtst	31		0	0	0	0	0	A					B					246										0
dcbz	31		0	0	0	0	0	A					B					1014										0
icbi	31		0	0	0	0	0	A					B					982										0

Table 20. Segment Register Manipulation Instructions

Name	0	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
mfsr [1]	31		D					0	SR				0	0	0	0	0	595										0
mfsrin [1]	31		D					0	0	0	0	0	B					659										0
mtsr [1]	31		S					0	SR				0	0	0	0	0	210										0
mtsrin [1]	31		S					0	0	0	0	0	B					242										0

Table 21. Lookaside Buffer Management Instructions

Name	0 5	6 7 8 9 10	11 12 13 14 15	16 17 18 19 20	21 22 23 24 25 26 27 28 29 30	31
tlbia [1,4]	31	0 0 0 0 0	0 0 0 0 0	0 0 0 0 0	370	0
tlbie [1,4]	31	0 0 0 0 0	0 0 0 0 0	B	306	0
tlbsync[1,4]	31	0 0 0 0 0	0 0 0 0 0	0 0 0 0 0	566	0

Table 22. External Control Instructions

Name	0 5	6 7 8 9 10	11 12 13 14 15	16 17 18 19 20	21 22 23 24 25 26 27 28 29 30	31
eciwx	31	D	A	B	310	0
ecowx	31	S	A	B	438	0

[1] Supervisor-level instruction
[2] Supervisor- and user-level instruction
[3] Load and store string or multiple instruction

Instructions Sorted by Form

Tables 23 through 32 list the MPCxxx instructions grouped by form.

Key:

Reserved bits

Table 23. I-Form

OPCD	LI	AA	LK

Specific Instruction

Name	0	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
b*x*	18	LI																									AA	LK

Table 24. B-Form

OPCD	BO	BI	BD	AA	LK

Specific Instruction

Name	0	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
bc*x*	16	BO									BI																AA	LK

Table 25. SC-Form

OPCD	0 0 0 0 0	0 0 0 0 0	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	1	0

Specific Instruction

Name	0	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
sc	17	0 0 0 0 0									0 0 0 0 0																1	0

Table 26. D-Form

OPCD	D		A	d
OPCD	D		A	SIMM
OPCD	S		A	d
OPCD	S		A	UIMM
OPCD	crfD	0 L	A	SIMM
OPCD	crfD	0 L	A	UIMM
OPCD	TO		A	SIMM

Specific Instructions

Name	0 5	6 7 8	9	10	11 12 13 14 15	16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
addi	14	D			A	SIMM
addic	12	D			A	SIMM
addic.	13	D			A	SIMM
addis	15	D			A	SIMM
andi.	28	S			A	UIMM
andis.	29	S			A	UIMM
cmpi	11	crfD	0	L	A	SIMM
cmpli	10	crfD	0	L	A	UIMM
lbz	34	D			A	d
lbzu	35	D			A	d
lha	42	D			A	d
lhau	43	D			A	d
lhz	40	D			A	d
lhzu	41	D			A	d
lmw [3]	46	D			A	d
lwz	32	D			A	d
lwzu	33	D			A	d
mulli	7	D			A	SIMM
ori	24	S			A	UIMM
oris	25	S			A	UIMM
stb	38	S			A	d
stbu	39	S			A	d
sth	44	S			A	d
sthu	45	S			A	d
stmw [3]	47	S			A	d
stw	36	S			A	d
stwu	37	S			A	d
subfic	08	D			A	SIMM
twi	03	TO			A	SIMM
xori	26	S			A	UIMM
xoris	27	S			A	UIMM

Table 27 DS-Form

Table 28. X-Form

OPCD	D	A	B	XO	0
OPCD	D	A	NB	XO	0
OPCD	D	0 0 0 0 0	B	XO	0
OPCD	D	0 0 0 0 0	0 0 0 0 0	XO	0
OPCD	D	0 SR	0 0 0 0 0	XO	0
OPCD	S	A	B	XO	Rc
OPCD	S	A	B	XO	1
OPCD	S	A	B	XO	0
OPCD	S	A	NB	XO	0
OPCD	S	A	0 0 0 0 0	XO	Rc
OPCD	S	0 0 0 0 0	B	XO	0
OPCD	S	0 0 0 0 0	0 0 0 0 0	XO	0
OPCD	S	0 SR	0 0 0 0 0	XO	0
OPCD	S	A	SH	XO	Rc
OPCD	crfD 0 L	A	B	XO	0
OPCD	crfD 0 0	A	B	XO	0
OPCD	crfD 0 0	crfS 0 0	0 0 0 0 0	XO	0
OPCD	crfD 0 0	0 0 0 0 0	0 0 0 0 0	XO	0
OPCD	crfD 0 0	0 0 0 0 0	IMM 0	XO	Rc
OPCD	TO	A	B	XO	0
OPCD	D	0 0 0 0 0	B	XO	Rc
OPCD	D	0 0 0 0 0	0 0 0 0 0	XO	Rc
OPCD	crbD	0 0 0 0 0	0 0 0 0 0	XO	Rc
OPCD	0 0 0 0 0	A	B	XO	0
OPCD	0 0 0 0 0	0 0 0 0 0	B	XO	0
OPCD	0 0 0 0 0	0 0 0 0 0	0 0 0 0 0	XO	0

Specific Instructions

and*x*	31	S	A	B	28	Rc
andc*x*	31	S	A	B	60	Rc
cmp	31	crfD 0 L	A	B	0	0
cmpl	31	crfD 0 L	A	B	32	0
cntlzw*x*	31	S	A	0 0 0 0 0	26	Rc
dcbf	31	0 0 0 0 0	A	B	86	0

dcbi [1]	31	0 0 0 0 0	A	B	470	0
dcbst	31	0 0 0 0 0	A	B	54	0
dcbt	31	0 0 0 0 0	A	B	278	0
dcbtst	31	0 0 0 0 0	A	B	246	0
dcbz	31	0 0 0 0 0	A	B	1014	0
eciwx	31	D	A	B	310	0
ecowx	31	S	A	B	438	0
eieio	31	0 0 0 0 0	0 0 0 0 0	0 0 0 0 0	854	0
eqv*x*	31	S	A	B	284	Rc
extsb*x*	31	S	A	0 0 0 0 0	954	Rc
extsh*x*	31	S	A	0 0 0 0 0	922	Rc
icbi	31	0 0 0 0 0	A	B	982	0
lbzux	31	D	A	B	119	0
lbzx	31	D	A	B	87	0
lhaux	31	D	A	B	375	0
lhax	31	D	A	B	343	0
lhbrx	31	D	A	B	790	0
lhzux	31	D	A	B	311	0
lhzx	31	D	A	B	279	0
lswi [3]	31	D	A	NB	597	0
lswx [3]	31	D	A	B	533	0
lwarx	31	D	A	B	20	0
lwbrx	31	D	A	B	534	0
lwzux	31	D	A	B	55	0
lwzx	31	D	A	B	23	0
mcrxr	31	crfD 0 0	0 0 0 0 0	0 0 0 0 0	512	0
mfcr	31	D	0 0 0 0 0	0 0 0 0 0	19	0
mfmsr [1]	31	D	0 0 0 0 0	0 0 0 0 0	83	0
mfsr [1]	31	D	0 SR	0 0 0 0 0	595	0
mfsrin [1]	31	D	0 0 0 0 0	B	659	0
mtmsr [1]	31	S	0 0 0 0 0	0 0 0 0 0	146	0
mtsr [1]	31	S	0 SR	0 0 0 0 0	210	0
mtsrin [1]	31	S	0 0 0 0 0	B	242	0
nand*x*	31	S	A	B	476	Rc
nor*x*	31	S	A	B	124	Rc

or*x*	31	S	A	B	444	Rc
orc*x*	31	S	A	B	412	Rc
slw*x*	31	S	A	B	24	Rc
sraw*x*	31	S	A	B	792	Rc
srawi*x*	31	S	A	SH	824	Rc
srw*x*	31	S	A	B	536	Rc
stbux	31	S	A	B	247	0
stbx	31	S	A	B	215	0
sthbrx	31	S	A	B	918	0
sthux	31	S	A	B	439	0
sthx	31	S	A	B	407	0
stswi [3]	31	S	A	NB	725	0
stswx [3]	31	S	A	B	661	0
stwbrx	31	S	A	B	662	0
stwcx.	31	S	A	B	150	1
stwux	31	S	A	B	183	0
stwx	31	S	A	B	151	0
sync	31	0 0 0 0 0	0 0 0 0 0	0 0 0 0 0	598	0
tlbia [1,4]	31	0 0 0 0 0	0 0 0 0 0	0 0 0 0 0	370	0
tlbie [1,4]	31	0 0 0 0 0	0 0 0 0 0	B	306	0
tlbsync[1,4]	31	0 0 0 0 0	0 0 0 0 0	0 0 0 0 0	566	0
tw	31	TO	A	B	4	0
xor*x*	31	S	A	B	316	Rc

Table 29. XL-Form

OPCD	BO		BI		0 0 0 0 0	XO	LK
OPCD	crbD		crbA		crbB	XO	0
OPCD	crfD	0 0	crfS	0 0	0 0 0 0 0	XO	0
OPCD	0 0 0 0 0		0 0 0 0 0		0 0 0 0 0	XO	0

Specific Instructions

Name	0 5	6 7 8 9 10	11 12 13 14 15	16 17 18 19 20	21 22 23 24 25	26 27 28 29 30	31
bcctr*x*	19	BO	BI	0 0 0 0 0	528		LK
bclr*x*	19	BO	BI	0 0 0 0 0	16		LK
crand	19	crbD	crbA	crbB	257		0

crandc	19	crbD	crbA	crbB	129	0
creqv	19	crbD	crbA	crbB	289	0
crnand	19	crbD	crbA	crbB	225	0
crnor	19	crbD	crbA	crbB	33	0
cror	19	crbD	crbA	crbB	449	0
crorc	19	crbD	crbA	crbB	417	0
crxor	19	crbD	crbA	crbB	193	0
isync	19	0 0 0 0 0	0 0 0 0 0	0 0 0 0 0	150	0
mcrf	19	crfD 0 0	crfS 0 0	0 0 0 0 0	0	0
rfi [1]	19	0 0 0 0 0	0 0 0 0 0	0 0 0 0 0	50	0

Table 30. XFX-Form

OPCD					D					spr										XO												0
OPCD					D			0			CRM								0	XO												0
OPCD					S					spr										XO												0
OPCD					D					tbr										XO												0

Specific Instructions

Name	0				5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
mfspr [2]	31				D						spr										339										0
mftb	31				D						tbr										371										0
mtcrf	31				S					0	CRM								0		144										0
mtspr [2]	31				D						spr										467										0

Table 31. XO-Form

OPCD					D					A										B												OE	XO	Rc
OPCD					D					A										B												0	XO	Rc
OPCD					D					A										0 0 0 0 0												OE	XO	Rc

Specific Instructions

Name	0				5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
add*x*	31				D						A										B										OE	266	Rc
addc*x*	31				D						A										B										OE	10	Rc
adde*x*	31				D						A										B										OE	138	Rc
addme*x*	31				D						A										0 0 0 0 0										OE	234	Rc
addze*x*	31				D						A										0 0 0 0 0										OE	202	Rc
divw*x*	31				D						A										B										OE	491	Rc
divwu*x*	31				D						A										B										OE	459	Rc
mulhw*x*	31				D						A										B										0	75	Rc
mulhwu*x*	31				D						A										B										0	11	Rc
mullw*x*	31				D						A										B										OE	235	Rc
neg*x*	31				D						A										0 0 0 0 0										OE	104	Rc
subf*x*	31				D						A										B										OE	40	Rc
subfc*x*	31				D						A										B										OE	8	Rc
subfe*x*	31				D						A										B										OE	136	Rc
subfme*x*	31				D						A										0 0 0 0 0										OE	232	Rc
subfze*x*	31				D						A										0 0 0 0 0										OE	200	Rc

Table 32. M-Form

OPCD	S	A	SH	MB	ME	Rc
OPCD	S	A	B	MB	ME	Rc

Specific Instructions

Name	0 5	6 7 8 9 10	11 12 13 14 15	16 17 18 19 20	21 22 23 24 25	26 27 28 29 30	31
rlwimi*x*	20	S	A	SH	MB	ME	Rc
rlwinm*x*	21	S	A	SH	MB	ME	Rc
rlwnm*x*	23	S	A	B	MB	ME	Rc

[1] Supervisor-level instruction
[2] Supervisor- and user-level instruction
[3] Load and store string or multiple instruction
[4] PowerPC Optional instruction

Instruction Set Legend

Table 33 provides general information on the MPCxxx instruction set (such as the architectural level, privilege level, and form).

Table 33. MPCxxx Instruction Set Legend

	UISA	VEA	OEA	Supervisor Level			Optional	Form
add*x*	√							XO
addc*x*	√							XO
adde*x*	√							XO
addi	√							D
addic	√							D
addic.	√							D
addis	√							D
addme*x*	√							XO
addze*x*	√							XO
and*x*	√							X
andc*x*	√							X
andi.	√							D
andis.	√							D
b*x*	√							I
bc*x*	√							B
bcctr*x*	√							XL
bclr*x*	√							XL
cmp	√							X
cmpi	√							D
cmpl	√							X
cmpli	√							D
cntlzw*x*	√							X
crand	√							XL
crandc	√							XL
creqv	√							XL
crnand	√							XL
crnor	√							XL

	UISA	VEA	OEA	Supervisor Level			Optional	Form
cror	√							XL
crorc	√							XL
crxor	√							XL
dcbf		√						X
dcbi			√	√				X
dcbst		√						X
dcbt		√						X
dcbtst		√						X
dcbz		√						X
divw*x*	√							XO
divwu*x*	√							XO
eciwx		√					√	X
ecowx		√					√	X
eieio		√						X
eqv*x*	√							X
extsb*x*	√							X
extsh*x*	√							X

	UISA	VEA	OEA	Supervisor Level			Optional	Form
icbi		√						X
isync		√						XL
lbz	√							D
lbzu	√							D
lbzux	√							X
lbzx	√							X

	UISA	VEA	OEA	Supervisor Level			Optional	Form
lha	√							D
lhau	√							D
lhaux	√							X
lhax	√							X
lhbrx	√							X
lhz	√							D
lhzu	√							D

lhzux	√							X
lhzx	√							X
lmw [2]	√							D
lswi [2]	√							X
lswx [2]	√							X
lwarx	√							X
lwbrx	√							X
lwz	√							D
lwzu	√							D
lwzux	√							X
lwzx	√							X
mcrf	√							XL
mcrxr	√							X
mfcr	√							X
mfmsr			√	√				X
mfspr [1]	√		√	√				XFX
mfsr			√	√				X
mfsr [3]			√	√			√	X
mfsrin			√	√				X
mfsrin [3]			√	√			√	X
	UISA	VEA	OEA	Supervisor Level			Optional	Form
mftb		√						XFX
mtcrf	√							XFX
mtmsr			√	√				X
mtmsr [3]			√	√			√	X
mtspr [1]	√		√	√				XFX
mtsr			√	√				X
mtsr [3]			√	√			√	X
mtsrin			√	√				X
mtsrin [3]			√	√			√	X
mulhw*x*	√							XO
mulhwu*x*	√							XO
mulli	√							D
mullw*x*	√							XO

nand*x*	√							X
neg*x*	√							XO
nor*x*	√							X
or*x*	√							X
orc*x*	√							X
ori	√							D
oris	√							D
rfi			√	√				XL
rfi [3]			√	√			√	XL
rlwimi*x*	√							M
rlwinm*x*	√							M
rlwnm*x*	√							M
	UISA	VEA	OEA	Supervisor Level			Optional	Form
sc	√		√					SC
slw*x*	√							X
sraw*x*	√							X
srawi*x*	√							X
srw*x*	√							X
stb	√							D
stbu	√							D
stbux	√							X
stbx	√							X
sth	√							D
sthbrx	√							X
sthu	√							D
sthux	√							X
sthx	√							X
	UISA	VEA	OEA	Supervisor Level			Optional	Form
stmw [2]	√							D
stswi [2]	√							X
stswx [2]	√							X
stw	√							D
stwbrx	√							X
stwcx.	√							X

stwu	√							D
stwux	√							X
stwx	√							X
subf*x*	√							XO
subfc*x*	√							XO
subfe*x*	√							XO
subfic	√							D
subfme*x*	√							XO
subfze*x*	√							XO
sync	√							X
tlbia			√	√			√	X
tlbie			√	√			√	X
tlbsync			√	√				X
tw	√							X
twi	√							D
xor*x*	√							X
xori	√							D
xoris	√							D

[1] Supervisor- and user-level instruction
[2] Load and store string or multiple instruction
[3] PowerPC Optional instruction