SECTION 9
INSTRUCTION SET

This section describes individual instructions, including a description of instruction formats and notation and an alphabetical listing of RCPU instructions by mnemonic.

9.1 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 to 5 always specify the primary opcode. Many instructions also have a secondary 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 set to zero, or if a field that must contain a particular value does not contain that value, the instruction form is invalid.

9.1.1 Split Field Notation

Some instruction fields occupy more than one contiguous sequence of bits or occupy a contiguous sequence of bits used in permuted order. Such a field is called a split field. In the format diagrams and in the individual instruction layouts, the name of a split field is shown in small letters, once for each of the contiguous sequences. In the pseudocode description of an instruction having a split field and in some places where individual bits of a split field are identified, the name of the field in small letters represents the concatenation of the sequences from left to right. Otherwise, the name of the field is capitalized and represents the concatenation of the sequences in some order, which need not be left to right, as described for each affected instruction.

9.1.2 Instruction Fields

Table 9-1 describes the instruction fields used in the various instruction formats.

Table 9-1 Instruction Formats

Field	Bits	Description
AA	30	Absolute address bit 0 The immediate field represents an address relative to the current instruction address. The effective 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 of the branch is the LI field sign-extended to 32 bits or the BD field sign-extended to 32 bits.
crbA	11:15	Field used to specify a bit in the CR to be used as a source.
crbB	16:20	Field used to specify a bit in the CR to be used as a source.
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.
crfD	6:8	Field used to specify one of the CR fields or one of the FPSCR fields as a destination.
crfS	11:13	Field used to specify one of the CR fields or one of the FPSCR fields as a source.
BI	11:15	Field used to specify a bit in the CR to be used as the condition of a branch conditional instruction.
BO	6:10	Field used to specify options for the branch conditional instructions. The encoding is described in **4.6 Flow Control Instructions**.
crbD	6:10	Field used to specify a bit in the CR or in the FPSCR as the destination of the result of an instruction.
CRM	12:19	Field mask 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.
FM	7:14	Field mask used to identify the FPSCR fields that are to be updated by the **mtfsf** instruction.
frA	11:15	Field used to specify an FPR as a source of an operation.
frB	16:20	Field used to specify an FPR as a source of an operation.
frC	21:25	Field used to specify an FPR as a source of an operation.
frS	6:10	Field used to specify an FPR as a source of an operation.
frD	6:10	Field used to specify an FPR as the destination of an operation.
IMM	16:19	Immediate field used as the data to be placed into a field in the FPSCR.
LI	6:29	Immediate field specifying a 24-bit, signed two's complement integer that is concatenated on the right with 0b00 and sign-extended to 32 bits.
LK	31	Link bit. 0 Does not update the link register. 1 Updates the link register. If the instruction is a branch instruction, the address of the instruction following the branch instruction is placed into the link register.
MB, M	21:25, 26:30	Fields used in rotate instructions to specify a 32-bit mask consisting of 1-bits from bit MB+32 through bit ME+32 inclusive, and 0-bits elsewhere, as described in **4.3.4 Integer Rotate and Shift Instructions**.

Table 9-1 Instruction Formats (Continued)

Field	Bits	Description
NB	16:20	Field used to specify the number of bytes to move in an immediate string load or store.
opcode	0:5	Primary opcode field.
OE	21	Used for extended arithmetic to enable setting OV and SO in the XER.
rA	11:15	Field used to specify a GPR to be used as a source or as a destination.
rB	16:20	Field used to specify a GPR to be used as a source.
Rc	31	Record bit 0 Does not update the condition register. 1 Updates the condition register (CR) to reflect the result of the operation. For integer instructions, CR[0:3] are set to reflect the result as a signed quantity. The result as an unsigned quantity or a bit string can be deduced from the EQ bit. For floating-point instructions, CR[4:7] are set to reflect floating-point exception, floating-point enabled exception, floating-point invalid operation exception, and floating-point overflow exception.
rS	6:10	Field used to specify a GPR to be used as a source.
rD	6:10	Field used to specify a GPR to be used as a destination.
SH	16:20	Field used to specify a shift amount.
SIMM	16:31	Immediate field used to specify a 16-bit signed integer.
SPR	11:20	Field used to specify a special purpose register for the **mtspr** and **mfspr** instructions. The encoding is described in **4.7.2 Move to/from Special Purpose Register Instructions**.
TO	6:10	Field used to specify the conditions on which to trap. The encoding is described in**4.6.7 Trap Instructions**.
UIMM	16:31	Immediate field used to specify a 16-bit unsigned integer.
XO	21:30, 22:30, 26:30, or 30	Secondary opcode field.

9.1.3 Notation and Conventions

The operation of some instructions is described by a register transfer language (RTL). See **Table 9-2** for a list of RTL notation and conventions used throughout this chapter.

Table 9-2 RTL Notation and Conventions

Notation/Convention	Meaning
←	Assignment
¬	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
<U,>U	Unsigned comparison relations
?	Unordered comparison relation
&, |	AND, OR logical operators
||	Used to describe the concatenation of two values (i.e., 010 || 111 is the same as 010111)
⊕, ≡	Exclusive-OR, Equivalence logical operators ((a≡b) = (a⊕¬b))
0b*nnnn*	A number expressed in binary format
0x*nnnn*	A number expressed in hexadecimal format
(**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
. (period)	As the last character of an instruction mnemonic, a period (.) means that the instruction updates the condition register field.
CEIL(x)	Least integer Š x
DOUBLE(x)	Result of converting x form floating-point single format to floating-point double format.
EXTS(x)	Result of extending x on the left with sign bits
GPR(x)	General Purpose Register x
MASK(x, y)	Mask having ones in positions x through y (wrapping if x > y) and 0's elsewhere
MEM(x, y)	Contents of y bytes of memory starting at address x
ROTL[32](x, y)	Result of rotating the 64-bit value x||x left y positions, where x is 32 bits long
SINGLE(x)	Result of converting x from floating-point double format to floating-point single format.
SPR(x)	Special Purpose Register x
x(*n*)	x is raised to the *n*th power
(*n*)x	The replication of x, *n* times (i.e., x concatenated to itself *n*-1 times). (*n*)0 and (*n*)1 are special cases
x[*n*]	*n* is a bit or field within x, where x is a register
TRAP	Invoke the system trap handler

Table 9-2 RTL Notation and Conventions (Continued)

Notation/Convention	Meaning
undefined	An undefined value. The value may vary from one implementation to another, and from one execution to another on the same implementation.
characterization	Reference to the setting of status bits, in a standard way that is explained in the text
CIA	Current instruction address, which is 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). Does not correspond to any architected register.
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.
if...then...else...	Conditional execution, indenting shows range, else is optional
do	Do loop, indenting shows range. To and/or by clauses specify incrementing an iteration variable, and while and/or until clauses give termination conditions, in the usual manner.
leave	Leave innermost do loop, or do loop described in leave statement

Precedence rules for RTL operators are summarized in **Table 9-3**.

Table 9-3 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
=,¦,<,ð,>,Š,<U,>U,?	Left to right
&,⊕,≡	Left to right
|	Left to right
– (range)	None
←	None

Note that operators higher in **Table 9-3** 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.

9.2 RCPU Instruction Set

The remainder of this chapter lists and describes the RCPU instruction set. The instructions are listed in alphabetical order by mnemonic. **Figure 9-1** shows the format for each instruction description page.

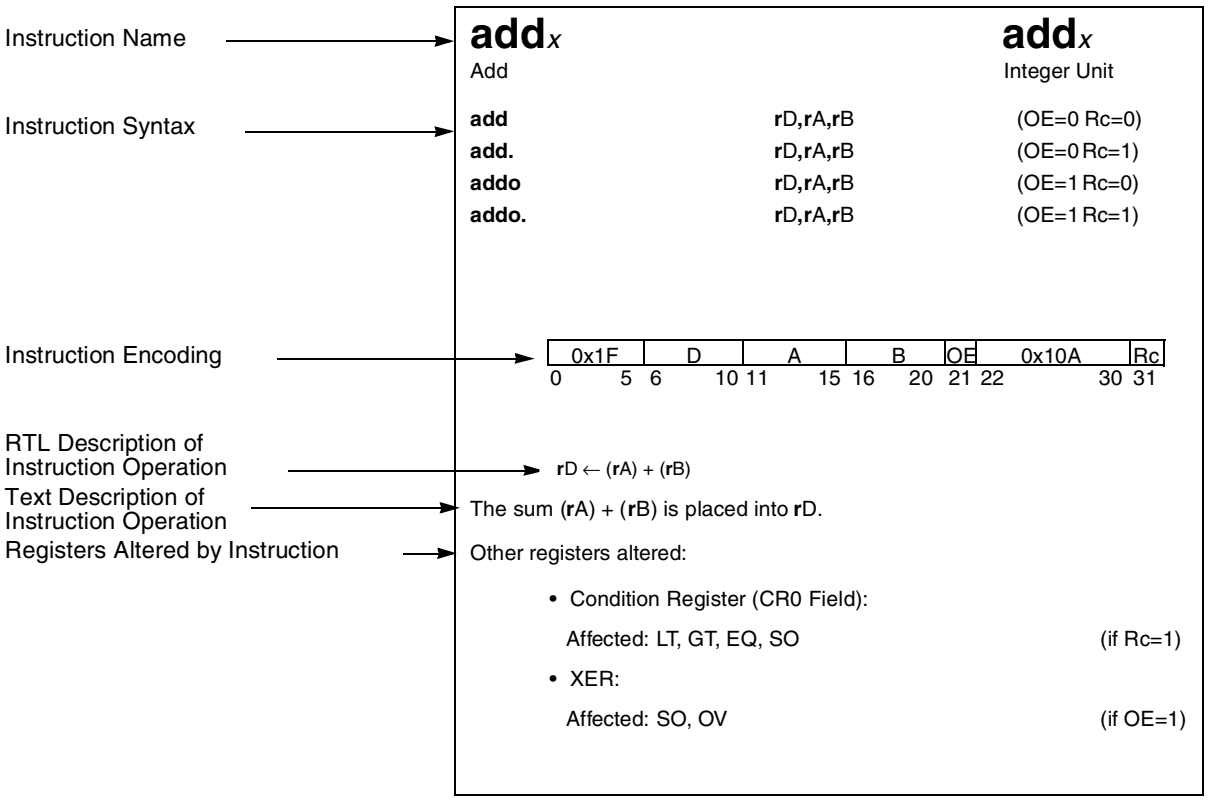
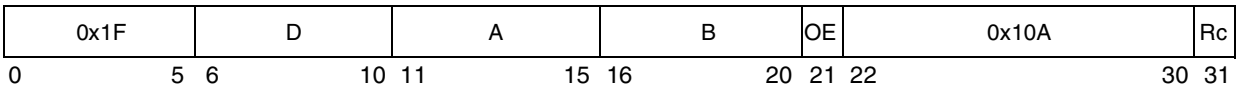


Figure 9-1 Instruction Description

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



rD ← (**r**A) + (**r**B)

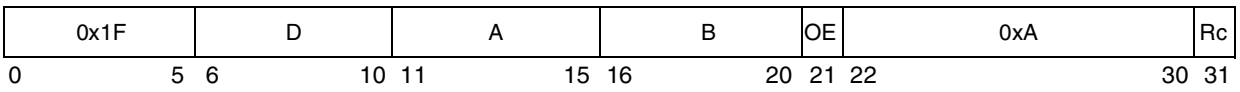
The sum (**r**A) + (**r**B) is placed into **r**D.

Other registers altered:

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

This instruction is defined by the PowerPC UISA.

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



rD ← (**r**A) + (**r**B)

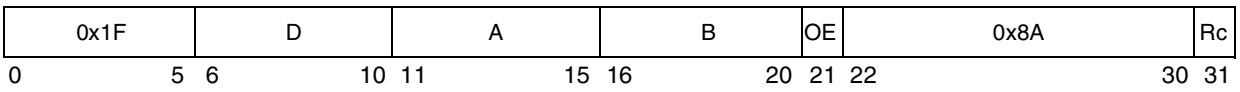
The sum (**r**A) + (**r**B) is placed into **r**D.

Other registers altered:

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

This instruction is defined by the PowerPC UISA.

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 ← (**r**A) + (**r**B) + XER[CA]

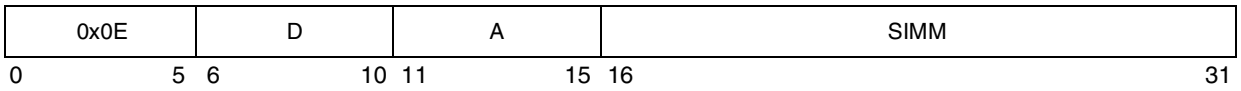
The sum (**r**A) + (**r**B) + XER[CA] is placed into **r**D.

Other registers altered:

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

This instruction is defined by the PowerPC UISA.

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.

Other registers altered:

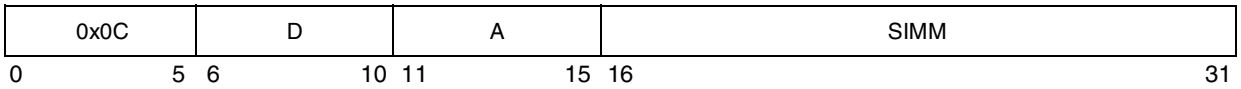
- None

Table 9-4 Simplified Mnemonics for addi Instruction

Simplified Mnemonic	Operands	Equivalent To
la	**r**D, SIMM(**r**A)	**addi r**D,**r**A,SIMM
li	**r**A,value	**addi r**A,**0**,value
subi	**r**D,**r**A,value	**addi** **r**D,**r**A,-value

This instruction is defined by the PowerPC UISA.

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



rD ← (**r**A) + EXTS(SIMM)

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

Other registers altered:

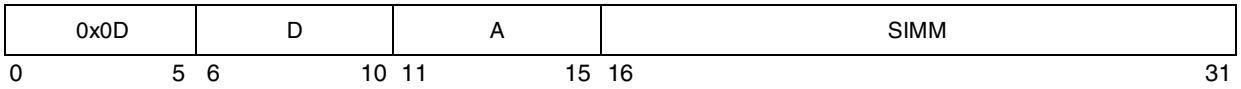
- XER:
  Affected: CA

Table 9-5 Simplified Mnemonics for addic Instruction

Simplified Mnemonic	Operands	Equivalent To
subic	**r**D,**r**A,value	**addic** **r**D,**r**A,-value

This instruction is defined by the PowerPC UISA.

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



rD ← (**r**A) + EXTS(SIMM)

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

Other registers altered:

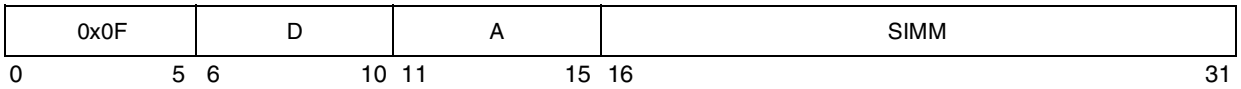
- Condition Register (CR0 Field):
  Affected: LT, GT, EQ, SO
- XER:
  Affected: CA

Table 9-6 Simplified Mnemonics for addic. Instruction

Simplified Mnemonic	Operands	Equivalent To
subic.	**r**D,**r**A,value	**addic.** **r**D,**r**A,-value

This instruction is defined by the PowerPC UISA.

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



if **r**A=0 then
rD←(SIMM || (16)0)
else
rD←(**r**A)+(SIMM || (16)0)

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

Other registers altered:

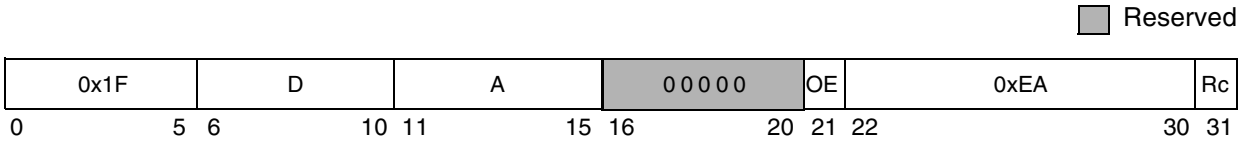
- None

Table 9-7 Simplified Mnemonics for addis Instruction

Simplified Mnemonic	Operands	Equivalent To
lis	**r**A,value	**addi r**A,**0**,value
subis	**r**D,**r**A,value	**addis r**D,**r**A,-value

This instruction is defined by the PowerPC UISA.

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 ← (**r**A) + XER[CA] - 1

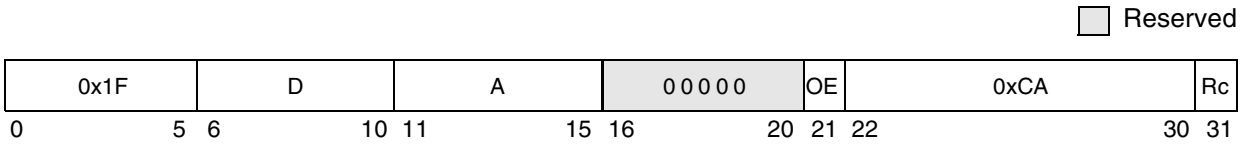
The sum (**r**A)+XER[CA]+0xFFFF FFFFF is placed into **r**D.

Other registers altered:

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

This instruction is defined by the PowerPC UISA.

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 ← (**r**A) + XER[CA]

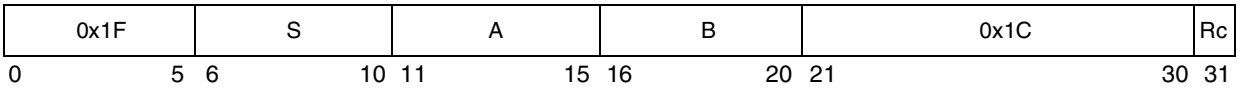
The sum (**r**A)+XER[CA] is placed into **r**D.

Other registers altered:

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

This instruction is defined by the PowerPC UISA.

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



rA ← (**r**S) & (**r**B)

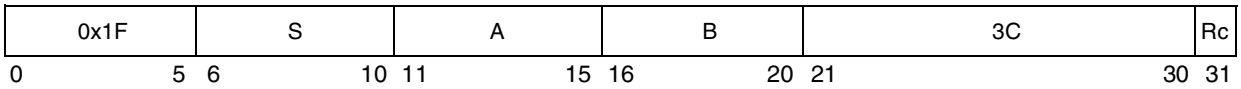
The contents of **r**S is ANDed with the contents of **r**B and the result is placed into **r**A.

Other registers altered:

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

This instruction is defined by the PowerPC UISA.

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



rA←(**r**S)& ¬ (**r**B)

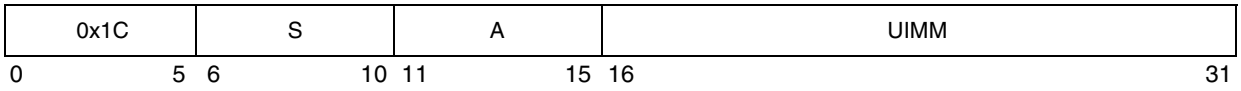
The contents of **r**S is ANDed with the one's complement of the contents of **r**B and the result is placed into **r**A.

Other registers altered:

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

This instruction is defined by the PowerPC UISA.

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



rA←(**r**S) & ((16)0 || UIMM)

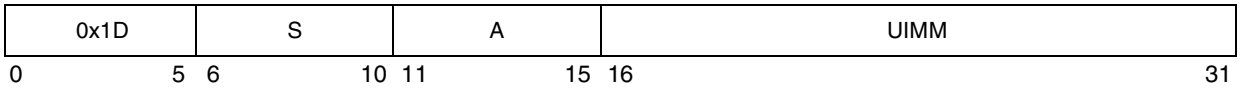
The contents of **r**S are ANDed with 0x0000 || UIMM and the result is placed into **r**A.

Other registers altered:

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

This instruction is defined by the PowerPC UISA.

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



rA←(**r**S)+(UIMM || (16)0)

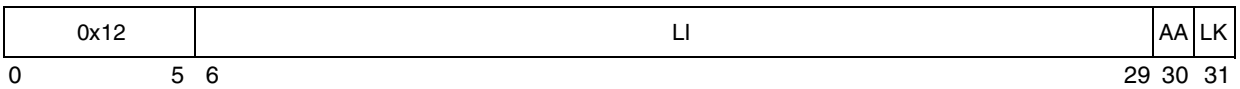
The contents of **r**S are ANDed with UIMM || 0x0000 and the result is placed into **r**A.

Other registers altered:

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

This instruction is defined by the PowerPC UISA.

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




```
if AA then
      NIA←EXTS(LI || 0b00)
else
      NIA←CIA+EXTS(LI || 0b00)
if LK, then
       LR←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.

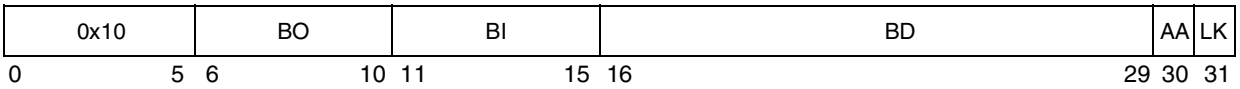
If LK=1, then the effective address of the instruction following the branch instruction is placed into the link register.

Other registers affected:

Link Register (LR) (if LK=1)

This instruction is defined by the PowerPC UISA.

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




```
if ¬ BO[2], then CTR ← CTR-1
ctr_ok ← BO[2] | ((CTR¦0) ⊕ BO[3])
cond_ok ← BO[0] | (CR[BI] ≡ BO[1])
if ctr_ok & cond_ok then
      if AA then
            NIA ← EXTS(BD || 0b00)
      else
            NIA ← CIA+EXTS(BD || 0b00)
 if LK, then
      LR ← 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 used as described above.

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 affected:

Count Register (CTR)	(if BO[2]=0)
Link Register (LR)	(if LK=1)

This instruction is defined by the PowerPC UISA.

Table 9-8 Simplified Mnemonics for bc, bca, bcl, and bcla Instructions

Operation	Simplified Mnemonic[1]	Equivalent To
Decrement CTR, branch if CTR non-zero	**bdnz** target	**bc 16,0**,target
Decrement CTR, branch absolute if CTR non-zero	**bdnza** target	**bca 16,0**,target
Decrement CTR, branch and update LR if CTR non-zero	**bdnzl** target	**bcl 16,0**,target
Decrement CTR, branch absolute and update LR if CTR non-zero	**bdnzla** target	**bcla 16,0**,target
Decrement CTR, branch if false and CTR non-zero	**bdnzf** BI,target	**bc 0**,BI,target
Decrement CTR, branch absolute if false and CTR non-zero	**bdnzfa** BI,target	**bca 0**,BI,target
Decrement CTR, branch and update LR if false and CTR non-zero	**bdnzfl** BI,target	**bcl 0**,BI,target
Decrement CTR, branch absolute and update LR if false and CGRnon-zero	**bdnzfla** BI,target	**bcla 0**,BI,target
Decrement CTR, branch if true and CTR non-zero	**bdnzt** BI,target	**bc 8**,BI,target
Decrement CTR, branch absolute if true and CTR non-zero	**bdnzta** BI,target	**bca 8**,BI,target
Decrement CTR, branch and update LR if true and CTR non-zero	**bdnztl** BI,target	**bcl 8**,BI,target
Decrement CTR, branch absolute and update LR if true and CTR non-zero	**bdnztla** BI,target	**bcla 8**,BI,target
Decrement CTR, branch if CTR zero	**bdz** target	**bc 18,0**,target
Decrement CTR, branch absolute if CTR zero	**bdza** target	**bca 18,0**,target
Decrement CTR, branch and update LR if CTR zero	**bdzl** target	**bcl 18,0**,target
Decrement CTR, branch absolute and update LR if CTR zero	**bdzla** target	**bcla 18,0**,target
Decrement CTR, branch if false and CTR zero	**bdzf** BI,target	**bc 2**,BI,target
Decrement CTR, branch absolute if false and CTR zero	**bdzfa** BI,target	**bca 2**,BI,target
Decrement CTR, branch and update LR if false and CTR zero	**bdzfl** BI,target	**bcl 2**,BI,target
Decrement CTR, branch absolute and update LR if false and CTR zero	**bdzfla** BI,target	**bcla 2**,BI,target
Decrement CTR, branch if true and CTR zero	**bdzt** BI,target	**bc 10**,BI,target
Decrement CTR, branch absolute if true and CTR zero	**bdzta** BI,target	**bca 10**,BI,target
Decrement CTR, ranch and update LR if true and CTR zero	**bdztl** BI,target	**bcl 10**,BI,target

Table 9-8 Simplified Mnemonics for bc, bca, bcl, and bcla Instructions (Continued)

Operation	Simplified Mnemonic[1]	Equivalent To
Decrement CTR, branch absolute and update LR if true and CTR zero	**bdztla** BI,target	**bcla 10**,BI,target
Branch if equal	**beq cr**X,target	**bc 12**, **4*cr**X**+2**,target
Branch absolute if equal	**beqa cr**X,target	**bca 12**, **4*cr**X**+2**,target
Branch and update LR if equal	**beql cr**X,target	**bcl 12**, **4*cr**X**+2**,target
Branch absolute and update LR if equal	**beqla cr**X,target	**bcla 12**, **4*cr**X**+2**,target
Branch if false	**bf** BI,target	**bc 4**,BI,target
Branch if false	**bfa** BI,target	**bca 4**,BI,target
Branch and update LR if false	**bfl** BI,target	**bcl 4**,BI,target
Branch absolute and update LR if false	**bfla** BI,target	**bcla 4**,BI,target
Branch if greater than or equal to	**bge cr**X,target	**bc 4**,**4*cr**X,target
Branch absolute if greater than or equal to	**bgea cr**X,target	**bca 4**,**4*cr**X,target
Branch and update LR if greater than or equal to	**bgel cr**X,target	**bcl 4**,**4*cr**X,target
Branch absolute and update LR if greater than or equal to	**bgela cr**X,target	**bcla 4**,**4*cr**X,target
Branch if greater than	**bgt cr**X,target	**bc 12**,**4*cr**X**+1**,target
Branch absolute if greater than	**bgta cr**X,target	**bca 12**,**4*cr**X**+1**,target
Branch and update LR if greater than	**bgtl cr**X,target	**bcl 12**,**4*cr**X**+1**,target
Branch absolute and update LR if greater than	**bgtla cr**X,target	**bcla 12**,**4*cr**X**+1**,target
Branch if less than or equal to	**ble cr**X,target	**bc 4**,**4*cr**X**+1**,target
Branch absolute if less than or equal to	**blea cr**X,target	**bca 4**,**4*cr**X**+1**,target
Branch and update LR if less than or equal to	**blel cr**X,target	**bcl 4**,**4*cr**X**+1**,target
Branch absolute and update LR if less than or equal to	**blela cr**X,target	**bcla 4**,**4*cr**X**+1**,target
Branch if less than	**blt cr**X,target	**bc 12**,**4*cr**X,target
Branch absolute if less than	**blta cr**X,target	**bca 12**,**4*cr**X,target
Branch and update LR if less than	**bltl cr**X,target	**bcl 12**,**4*cr**X,target
Branch absolute and update LR if less than	**bltla cr**X,target	**bcla 12**,**4*cr**X,target
Branch if not equal to	**bne cr**X,target	**bc 4**,**4*cr**X**+2**,target
Branch absolute if not equal to	**bnea cr**X,target	**bca 4**,**4*cr**X**+2**,target
Branch and update LR if not equal to	**bnel cr**X,target	**bcl 4**,**4*cr**X**+2**,target
Branch absolute and update LR if not equal to	**bnela cr**X,target	**bcla 4**,**4*cr**X**+2**,target
Branch if not greater than	**bng cr**X,target	**bc 4**,**4*cr**X**+1**,target

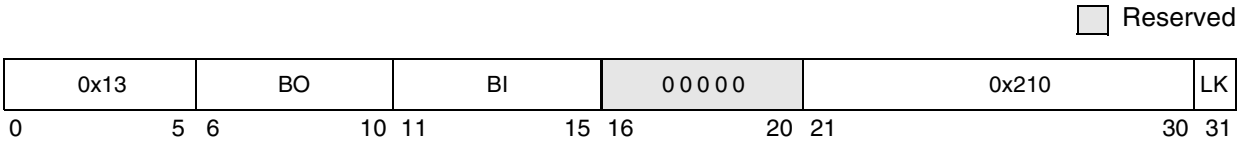
Table 9-8 Simplified Mnemonics for bc, bca, bcl, and bcla Instructions (Continued)

Operation	Simplified Mnemonic[1]	Equivalent To
Branch absolute if not greater than	**bnga crX**,target	**bca 4,4*crX+1**,target
Branch and update LR if not greater than	**bngl crX**,target	**bcl 4,4*crX+1**,target
Branch absolute and update LR if not greater than	**bngla crX**,target	**bcla 4,4*crX+1**,target
Branch if not less than	**bnl crX**,target	**bc 4,4*crX**,target
Branch absolute if not less than	**bnla crX**,target	**bca 4,4*crX**,target
Branch and update LR if not less than	**bnll crX**,target	**bcl 4,4*crX**,target
Branch absolute and update LR if not less than	**bnlla crX**,target	**bcla 4,4*crX**,target
Branch if not summary overflow	**bns crX**,target	**bc 4,4*crX+3**,target
Branch absolute if not summary overflow	**bnsa crX**,target	**bca 4,4*crX+3**,target
Branch and update LR if not summary overflow	**bnsl crX**,target	**bcl 4,4*crX+3**,target
Branch absolute and update LR if not summary overflow	**bnsla crX**,target	**bcla 4,4*crX+3**,target
Branch if not unordered	**bnu crX**,target	**bc 4,4*crX+3**,target
Branch absolute if not unordered	**bnua crX**,target	**bca 4,4*crX+3**,target
Branch and update LR if not unorderd	**bnul crX**,target	**bcl 4,4*crX+3**,target
Branch absolute and update LR if not unordered	**bnula crX**,target	**bcla 4,4*crX+3**,target
Branch if summary overflow	**bso crX**,target	**bc 12,4*crX+3**,target
Branch absolute if summary overflow	**bsoa crX**,target	**bca 12,4*crX+3**,target
Branch and update LR if summary overflow	**bsol crX**,target	**bcl 12,4*crX+3**,target
Branch absolute and update LR if summary overflow	**bsola crX**,target	**bcla 12,4*crX+3**,target
Branch if true	**bt** BI,target	**bc 12**,BI,target
Branch absolute if true	**bta** BI,target	**bca 12**,BI,target
Branch and update LR if true	**btl** BI,target	**bcl 12**,BI,target
Branch absolute and update LR if true	**btla** BI,target	**bcla 12**,BI,target
Branch if unordered	**bun crX**,target	**bc 12,4*crX+3**,target
Branch absolute if unordered	**buna crX**,target	**bca 12,4*crX+3**,target
Branch and update LR if unordered	**bunl crX**,target	**bcl 12,4*crX+3**,target
Branch and update LR if unordered	**bunla crX**,target	**bcla 12,4*crX+3**,target

NOTES:
1. If **cr**X is not included in the operand list (for operations that use a **cr** field), **cr0** is assumed.

Refer to **APPENDIX E SIMPLIFIED MNEMONICS** for more information on simplified mnemonics.

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




```
cond_ok ← BO[0] | (CR[BI] ≡ BO[1])
if cond_ok then
      NIA ← CTR[0:29] || 0b00
 if LK then
      LR ← 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 used as described above, and the branch target address is CTR[0:29] || 0b00.

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

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

Other registers affected:

Link Register (LR) (if LK=1)

This instruction is defined by the PowerPC UISA.

Table 9-9 provides simplified mnemonics for the **bcctr** and **bcctrl** instructions. Refer to **APPENDIX E SIMPLIFIED MNEMONICS** for more information on simplified mnemonics.

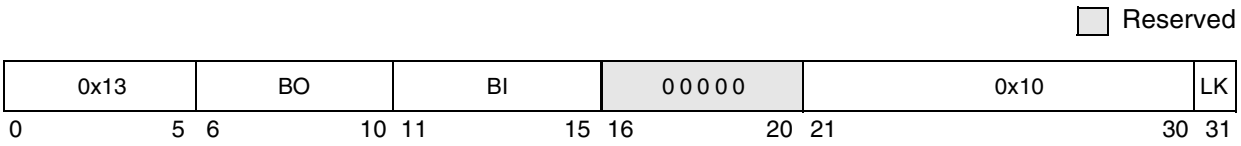
Table 9-9 Simplified Mnemonics for bcctr and bcctrl Instructions

Operation	Simplified Mnemonic[1]	Equivalent To
Branch to CTR	**bctr**	**bcctr 20,0**
Branch to CTR and update LR	**bctrl**	**bcctrl 20,0**
Branch if equal to CTR	**beqctr** crX	**bcctr 12**, **4*cr**X**+2**
Branch if equal to CTR, update LR	**beqctrl** crX	**bcctrl 12**, **4*cr**X**+2**
Branch if false to CTR	**bfctr** BI	**bcctr 4**,BI
Branch if false to CTR, update LR	**bfctrl** BI	**bcctrl 4**,BI
Branch to CTR if greater than or equal to	**bgectr** crX	**bcctr 4**,**4*cr**X
Branch to CTR if greater than or equal to, update LR	**bgectrl** crX	**bcctrl 4**,**4*cr**X
Branch to CTR if greater than	**bgtctr** crX	**bcctr 12**,**4*cr**X**+1**
Branch to CTR if greater than, update LR	**bgtctrl** crX	**bcctrl 12**,**4*cr**X**+1**
Branch to CTR if less than or equal to	**blectr** crX	**bcctr 4**,**4*cr**X**+1**
Branch to CTR if less than or equal to, update LR	**blectrl** crX	**bcctrl 4**,**4*cr**X**+1**
Branch to CTR if less than	**bltctr** crX	**bcctr 12**,**4*cr**X
Branch to CTR if less than, update LR	**bltctrl** crX	**bcctrl 12**,**4*cr**X
Branch to CTR if not equal to	**bnectr** crX	**bcctr 4**,**4*cr**X**+2**
Branch to CTR if not equal to, update LR	**bnectrl** crX	**bcctrl 4**,**4*cr**X**+2**
Branch to CTR if not greater than	**bngctr** crX	**bcctr 4**,**4*cr**X**+1**
Branch to CTR if not greater than, update LR	**bngctrl** crX	**bcctrl 4**,**4*cr**X**+1**
Branch to CTR if not less than	**bnlctr** crX	**bcctr 4**,**4*cr**X
Branch to CTR if not less than, update LR	**bnlctrl** crX	**bcctrl 4**,**4*cr**X
Branch to CTR if not summary overflow	**bnsctr** crX	**bcctr 4**,**4*cr**X**+3**
Branch to CTR if not summary overflow, update LR	**bnsctrl** crX	**bcctrl 4**,**4*cr**X**+3**
Branch to CTR if not unordered	**bnuctr** crX	**bcctrl 4**,**4*cr**X**+3**
Branch to CTR if not unordered, update LR	**bnuctrl** crX	**bcctrl 4**,**4*cr**X**+3**
Branch to CTR if summary overflow	**bsoctr** crX	**bcctr 12**,**4*cr**X**+3**
Branch to CTR if summary overflow, update LR	**bsoctrl** crX	**bcctrl 12**,**4*cr**X**+3**
Branch to CTR if true	**btctr** BI	**bcctr 12**,BI
Branch to CTR if true, update LR	**btctrl** BI	**bcctrl 12**,BI
Branch to CTR if unordered	**bunctr** crX	**bcctr 12**,**4*cr**X**+3**
Branch to CTR if unordered, update LR	**bunctrl** crX	**bcctrl 12**,**4*cr**X**+3**

NOTES:

1. If **cr**X is not included in the operand list (for operations that use a **cr** field), **cr0** is assumed.

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




```
if ¬ BO[2] then
      CTR ← CTR-1
ctr_ok ← BO[2] | ((CTR¦0) ⊕ BO[3])
cond_ok ← BO[0] | (CR[BI] ≡ BO[1])
if ctr_ok & cond_ok then
      NIA ← LR[0:29] || 0b00
 if LK then
      LR ← 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 used as described above, and 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 affected:

Count Register (CTR)	(if BO[2]=0)
Link Register (LR)	(if LK=1)

This instruction is defined by the PowerPC UISA.

Table 9-10 Simplified Mnemonics for bclr and bclrl Instructions

Operation	Simplified Mnemonic[1]	Equivalent To
Decrement CTR, branch to LR if false and CTR non-zero	**bdnzflr** BI	**bclr 0**,BI
Decrement CTR, branch to LR if false and CTR non-zero, update LR	**bdnzflrl** BI	**bclrl 0**,BI
Decrement CTR, branch to LR if CTR non-zero	**bdnzlr**	**bclr 16,0**
Decrement CTR, branch to LR if CTR non-zero, update LR	**bdnzlrl**	**bclrl 16,0**
Decrement CTR, branch to LR if true and CTR non-zero	**bdnztlr** BI	**bclr 8**,BI
Decrement CTR, branch to LR if true and CTR non-zero, update LR	**bdnztlrl** BI	**bclrl 8**,BI
Decrement CTR, branch to LR if false and CTR zero	**bdzflr** BI	**bclr 2**,BI
Decrement CTR, branch to LR if false and CTR zero, update LR	**bdzflrl** BI	**bclrl 2**,BI
Decrement CTR, branch to LR if CTR zero	**bdzlr**	**bclr 18,0**
Decrement CTR, branch to LR if CTR zero, update LR	**bdzlrl**	**bclrl 18,0**
Decrement CTR, branch to LR if true and CTR zero	**bdztlr** BI	**bclr 10**,BI
Decrement CTR, branch to LR if true and CTR zero, update LR	**bdztlrl** BI	**bclrl 10**,BI
Branch to LR if equal	**beqlr cr**X	**bclr 12**, **4*cr**X**+2**
Branch to LR if equal, update LR	**beqlrl cr**X	**bclrl 12**, **4*cr**X**+2**
Branch to LR if false	**bflr** BI	**bclr 4**,BI
Branch to LR if false, update LR	**bflrl** BI	**bclrl 4**,BI
Branch to LR if greater than or equal to	**bgelr cr**X	**bclr 4**,**4*cr**X
Branch to LR if greater than or equal to, update LR	**bgelrl cr**X	**bclrl 4**,**4*cr**X
Branch to LR if greater than	**bgtlr cr**X	**bclr 12**,**4*cr**X**+1**
Branch to LR if greater than, update LR	**bgtlrl cr**X	**bclrl 12**,**4*cr**X**+1**
Branch to LR if less than or equal to	**blelr cr**X	**bclr 4**,**4*cr**X**+1**
Branch to LR if less than or equal to, update LR	**blelrl cr**X	**bclrl 4**,**4*cr**X**+1**
Branch to LR	**blr**	**bclr 20,0**
Branch to LR, update LR	**blrl**	**bclrl 20,0**
Branch to LR if less than	**bltlr cr**X	**bclr 12**,**4*cr**X
Branch to LR if less than, update LR	**bltlrl cr**X	**bclrl 12**,**4*cr**X

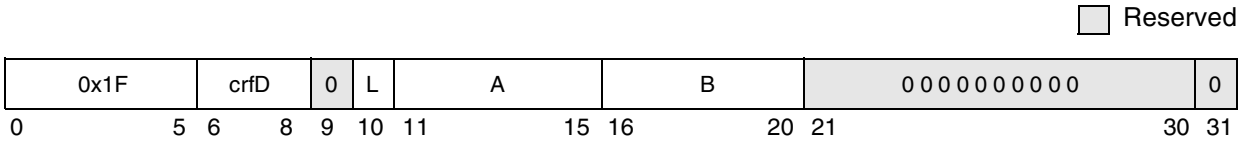
Table 9-10 Simplified Mnemonics for bclr and bclrl Instructions (Continued)

Operation	Simplified Mnemonic[1]	Equivalent To
Branch to LR if not equal to	**bnelr crX**	**bclr 4,4*crX+2**
Branch to LR if not equal to, update LR	**bnelrl crX**	**bclrl 4,4*crX+2**
Branch to LR if not greater than	**bnglr crX**	**bclr 4,4*crX+1**
Branch to LR if not greater than, update LR	**bnglrl crX**	**bclrl 4,4*crX+1**
Branch to LR if not less than	**bnllr crX**	**bclr 4,4*crX**
Branch to LR if not less than, update LR	**bnllrl crX**	**bclrl 4,4*crX**
Branch to LR if not summary overflow	**bnslr crX**	**bclr 4,4*crX+3**
Branch to LR if not summary overflow, update LR	**bnslrl crX**	**bclrl 4,4*crX+3**
Branch to LR if not unordered	**bnulr crX**	**bclr 4,4*crX+3**
Branch to LR if not unordered, update LR	**bnulrl crX**	**bclrl 4,4*crX+3**
Branch to LR if summary overflow	**bsolr crX**	**bclr 12,4*crX+3**
Branch to LR if summary overflow, update LR	**bsolrl crX**	**bclrl 12,4*crX+3**
Branch to LR if true	**btlr** BI	**bclr 12**,BI
Branch to LR if true, update LR	**btlrl** BI	**bclrl 12**,BI
Branch to LR if unordered	**bunlr crX**	**bclr 12,4*crX+3**
Branch to LR if unordered, update LR	**bunlrl crX**	**bclrl 12,4*crX+3**

NOTES:
1. If **cr**X is not included in the operand list (for operations that use a **cr** field), **cr0** is assumed.

Refer to **APPENDIX E SIMPLIFIED MNEMONICS** for more information on simplified mnemonics.

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




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

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

The L operand controls whether **r**A and **r**B are treated as 32-bit operands (L=0) or 64-bit operands (L=1). For 32-bit PowerPC implementations such as the RCPU, if L=1, the instruction form is invalid.

Other registers altered:

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

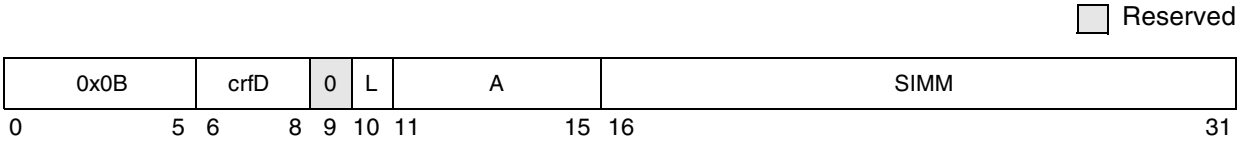
  Affected: LT, GT, EQ, SO

This instruction is defined by the PowerPC UISA.

Table 9-11 Simplified Mnemonics for cmp Instruction

Operation	Simplified Mnemonic	Equivalent To
Compare word	**cmpw crf**D, **r**A,**r**B **cmp crf**D, **r**A,**r**B	**cmp crf**D, **0, r**A,**r**B
Compare word, place result in CR0	**cmpw r**A,**r**B **cmp r**A,**r**B	**cmp 0, 0, r**A,**r**B

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




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

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

The L operand controls whether **r**A and **r**B are treated as 32-bit operands (L=0) or 64-bit operands (L=1). For 32-bit PowerPC implementations such as the RCPU, if L=1, the instruction form is invalid.

Other registers altered:

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

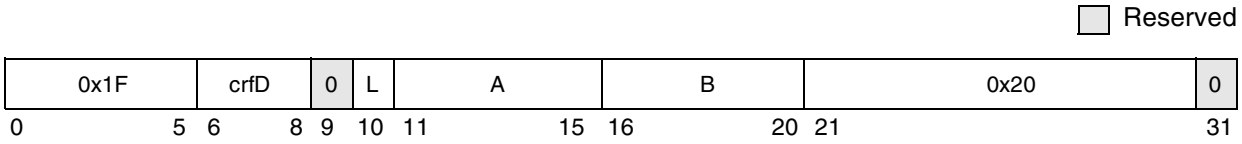
  Affected: LT, GT, EQ, SO

This instruction is defined by the PowerPC UISA.

Table 9-12 Simplified Mnemonics for cmpi Instruction

Operation	Simplified Mnemonic	Equivalent To
Compare word immediate	**cmpwi crf**,**r**A,value **cmpi crf**D, **r**A,value	**cmpi crf**D, **0, r**A,value
Compare word immediate, place result in CR0	**cmpwi r**A,value **cmpi r**A,value	**cmpi 0, 0, r**A,value

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.

The L operand controls whether **r**A and **r**B are treated as 32-bit operands (L=0) or 64-bit operands (L=1). For 32-bit PowerPC implementations such as the RCPU, if L=1, the instruction form is invalid.

Other registers altered:

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

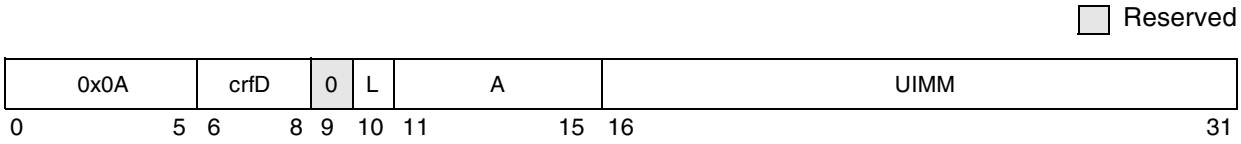
  Affected: LT, GT, EQ, SO

This instruction is defined by the PowerPC UISA.

Table 9-13 Simplified Mnemonics for cmpl Instruction

Operation	Simplified Mnemonic	Equivalent To
Compare word logical	**cmplw crf**D, **r**A,**r**B **cmpl crf**D, **r**A,**r**B	**cmpl crf**D**, 0, r**A,**r**B
Compare word logical, place result in CR0	**cmplw r**A,**r**B **cmpl r**A,**r**B	**cmpl 0, 0, r**A,**r**B

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




```
a ← (rA)
b ← (rB)
if a <U (0x0000 || UIMM) then
      c ← 0b100
else
      if a >U (0x0000 || 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.

The L operand controls whether **r**A and **r**B are treated as 32-bit operands (L=0) or 64-bit operands (L=1). For 32-bit PowerPC implementations such as the RCPU, if L=1, the instruction form is invalid.

Other registers altered:

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

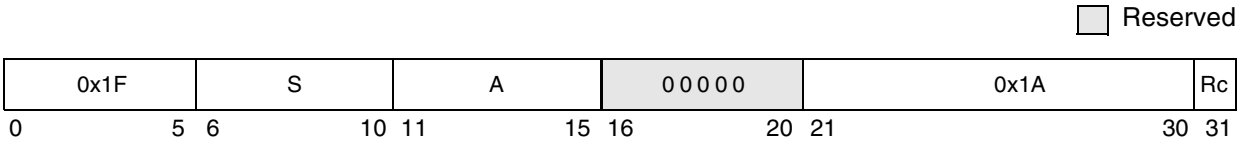
  Affected: LT, GT, EQ, SO

This instruction is defined by the PowerPC UISA.

Table 9-14 Simplified Mnemonics for cmpli Instruction

Operation	Simplified Mnemonic	Equivalent To
Compare word logical immediate	**cmplwi crf**D,**r**A,value **cmpli crf**D,**r**A,value	**cmpli crf**D**,0,r**A,value
Compare word logical immediate, place result in CR0	**cmplwi r**A,value **cmpli r**A,value	**cmpli 0,0,r**A,value

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



n ← 0
do while *n* < 32
if **r**S[*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.

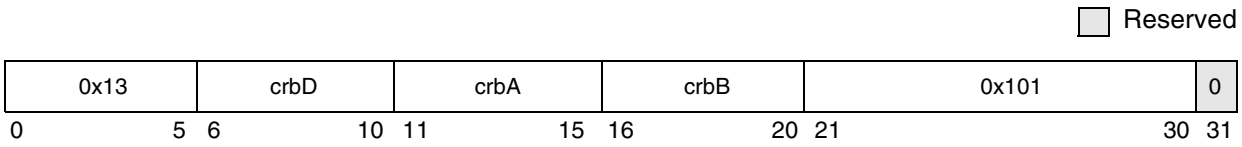
Other registers altered:

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

For count leading zeros instructions, if Rc=1 then LT is cleared to zero in the CR0 field.

This instruction is defined by the PowerPC UISA.

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.

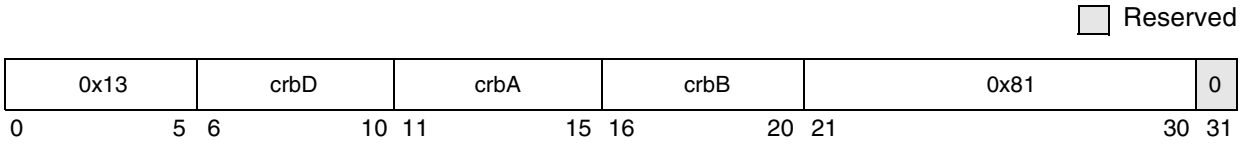
Other registers altered:

- Condition Register:

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

This instruction is defined by the PowerPC UISA.

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.

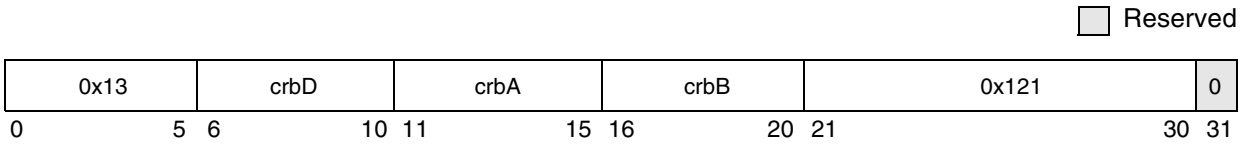
Other registers altered:

- Condition Register:

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

This instruction is defined by the PowerPC UISA.

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:

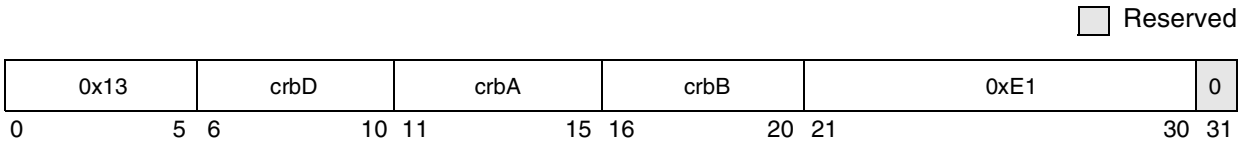
  Affected: Bit specified by operand **crb**D

This instruction is defined by the PowerPC UISA.

Table 9-15 Simplified Mnemonics for creqv Instruction

Operation	Simplified Mnemonic	Equivalent To
Condition register set	**crset crb**D	**creqv crb**D,**crb**D,**crb**D

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.

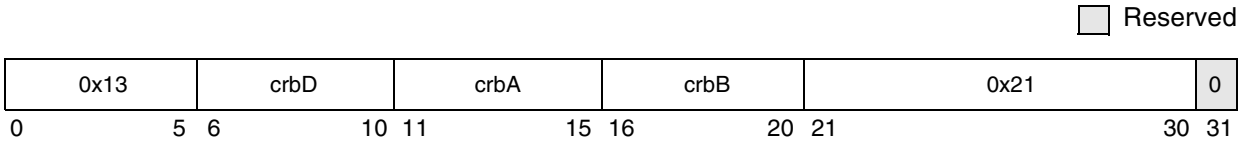
Other registers altered:

- Condition Register:

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

This instruction is defined by the PowerPC UISA.

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:

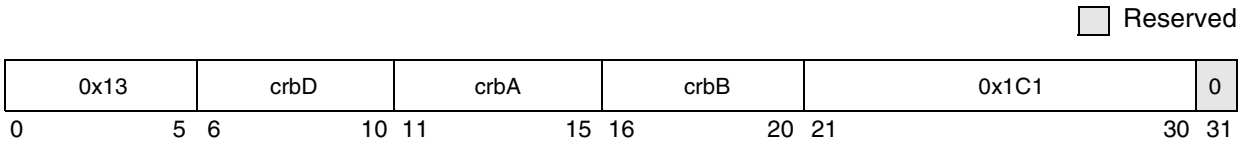
  Affected: Bit specified by operand **crb**D

This instruction is defined by the PowerPC UISA.

Table 9-16 Simplified Mnemonics for crnor Instruction

Operation	Simplified Mnemonic	Equivalent To
Condition register NOT	**crnot crb**D, **crb**A	**crnor crb**D,**crb**A,**crb**A

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:

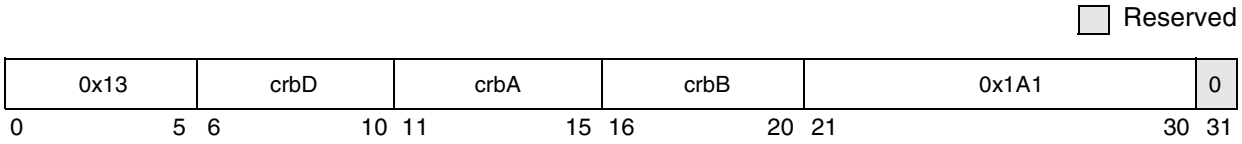
  Affected: Bit specified by operand **crb**D

This instruction is defined by the PowerPC UISA.

Table 9-17 Simplified Mnemonics for cror Instruction

Operation	Simplified Mnemonic	Equivalent To
Condition register move	**crmove crb**D, **crb**A	**cror crb**D,**crb**A,**crb**A

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.

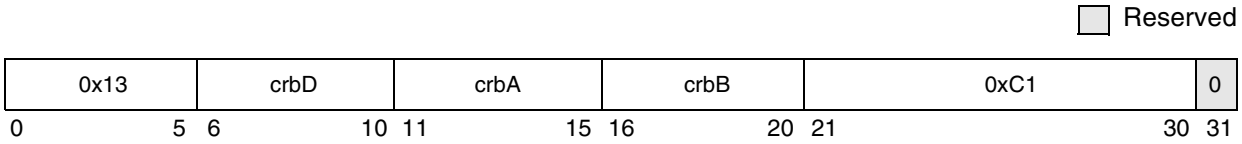
Other registers altered:

- Condition Register:

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

This instruction is defined by the PowerPC UISA.

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 bit specified by **crb**D.

Other registers altered:

- Condition Register:

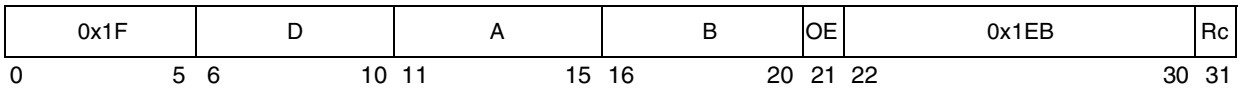
  Affected: Bit specified by **crb**D

This instruction is defined by the PowerPC UISA.

Table 9-18 Simplified Mnemonics for crxor Instruction

Operation	Simplified Mnemonic	Equivalent To
Condition register clear	**crclr crb**D	**crxor crb**D,**crb**D,**crb**D

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 ←(**r**A)
divisor ←(**r**B)
rD ← dividend ÷ divisor

Register **r**A is the 32-bit dividend. Register **r**B is the 32-bit divisor. A 32-bit quotient is formed and placed into **r**D. The remainder is not supplied as a result.

Both operands are interpreted as signed integers. The quotient is the unique signed integer that satisfies the following:

dividend=(quotient times 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 / -1

<anything> / 0

then the following conditons result:

- The contents of **r**D are undefined.
- If Rc = 1, the contents of the LT, GT, and EQ bits of the CR0 field are undefined.
- If OE = 1, then OV is set to 1.

Other registers altered:

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

- XER:

  Affected: SO, OV (if OE=1)

The 32-bit signed remainder of dividing **r**A by **r**B can be computed as follows, except in the case that **r**A= 0x8000 0000 and **r**B=-1:

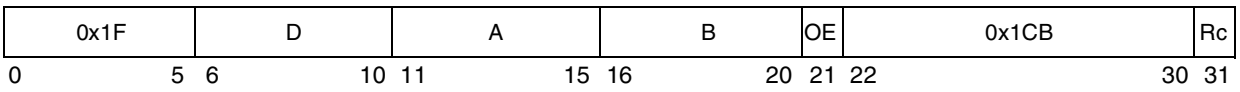
divw **r**D,**r**A,**r**B # **r**D=quotient

mull **r**D,**r**D,**r**B # **r**D=quotient∗divisor

subf **r**D,**r**D,**r**A # **r**D=remainder

This instruction is defined by the PowerPC UISA.

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 ← (**r**A)
divisor ← (**r**B)
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 and placed into **r**D. The remainder is not supplied as a result.

Both operands are interpreted as unsigned integers, except that if Rc = 1 the first three bits of the CR0 field are set by signed comparison of the result to zero. The quotient is the unique unsigned integer that satisfies the following:

dividend=(quotient ∗ divisor)+r

where

0 ≤ r < divisor.

If an attempt is made to divide by zero, then the following conditons result:

- The contents of **r**D are undefined.
- If Rc = 1, the contents of the LT, GT, and EQ bits of the CR0 field are undefined.
- If OE = 1, then OV is set to 1.

Other registers altered:

- Condition Register (CR0 Field):

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

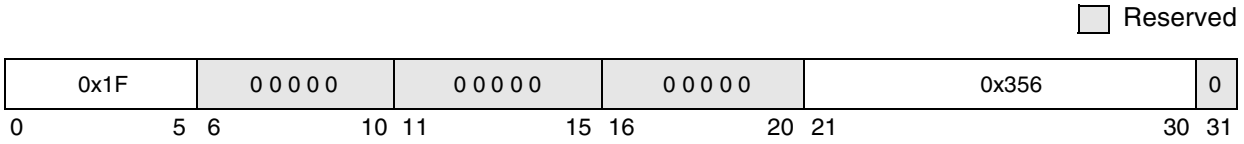
- XER:

  Affected: SO, OV (if OE=1)

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

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

This instruction is defined by the PowerPC UISA.



The **eieio** instruction provides an ordering function for the effects of load and store instructions executed by a given processor. Executing an **eieio** instruction ensures that all memory accesses previously initiated by the given processor are complete with respect to main memory before any memory accesses subsequently initiated by the given processor access main memory.

The **eieio** instruction orders loads from cache-inhibited memory.

Other registers altered:

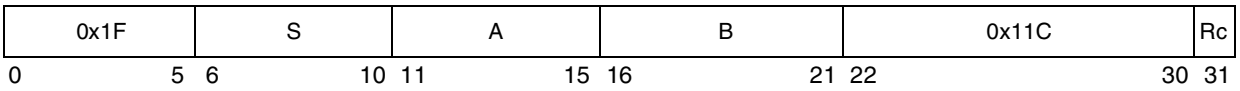
- None

The **eieio** instruction is intended for use only in performing memory-mapped I/O operations and to prevent load/store combining operations in main memory. It 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.

The **eieio** instruction may complete before previously initiated memory accesses have been performed with respect to other processors and mechanisms.

This instruction is defined by the PowerPC VEA.

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



rA ← ((**r**S) ≡ (**r**B))

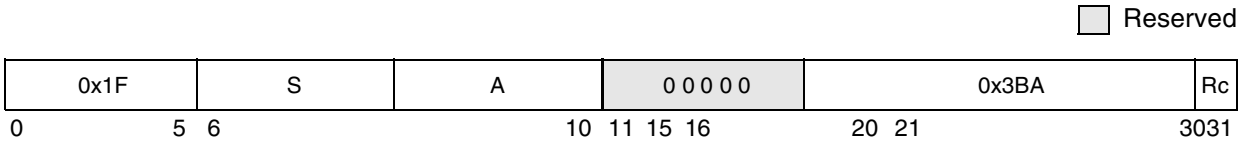
The contents of **r**S are XORed with the contents of **r**B and the complemented result is placed into **r**A.

Other registers altered:

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

This instruction is defined by the PowerPC UISA.

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



S ← **r**S[24]
rA[24:31] ← **r**S[24:31]
rA[0:23] ← (24)S

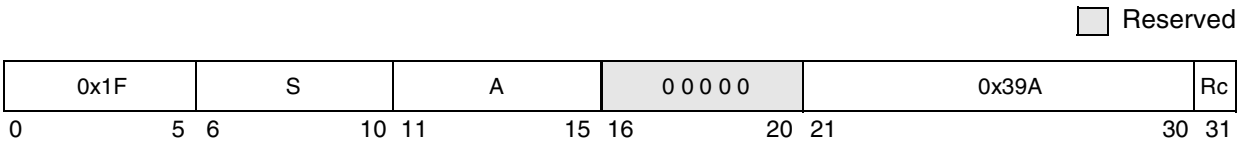
The contents of **r**S[24:31] are placed into **r**A[24:31]. Bit 24 of **r**S is placed into **r**A[0:23].

Other registers altered:

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

This instruction is defined by the PowerPC UISA.

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



S ← **r**S[16]
rA[16:31]← **r**S[16:31]
rA[0:15] ← (16)S

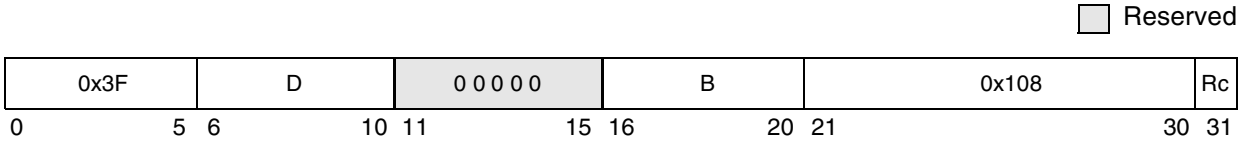
The contents of **r**S[16:31] are placed into **r**A[16:31]. Bit 16 of **r**S is placed into **r**A[0:15].

Other registers altered:

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

This instruction is defined by the PowerPC UISA.

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



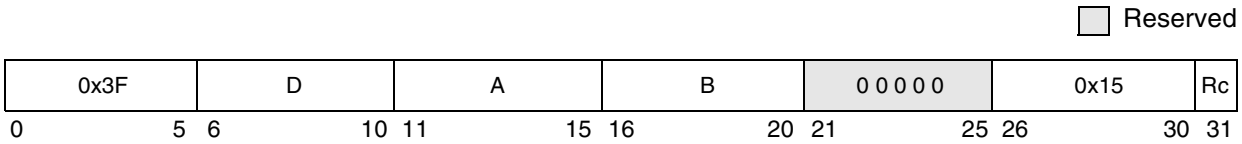
The contents of **fr**B with bit 0 cleared to zero are placed into **fr**D.

Other registers altered:

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

This instruction is defined by the PowerPC UISA.

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



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

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

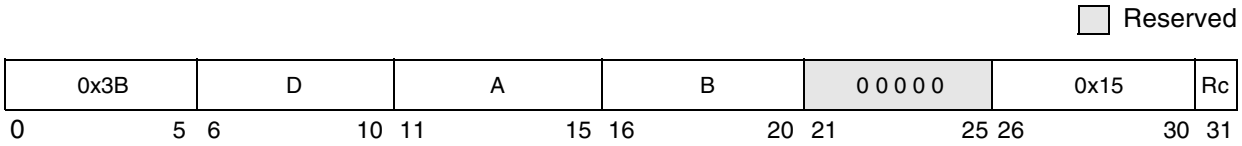
If a carry occurs, the sum's significand is shifted right one bit position and the exponent is increased by one. FPSCR[FPRF] is set to the class and sign of the result, except for invalid operation exceptions when FPSCR[VE]=1.

Other registers altered:

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

This instruction is defined by the PowerPC UISA.

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



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

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

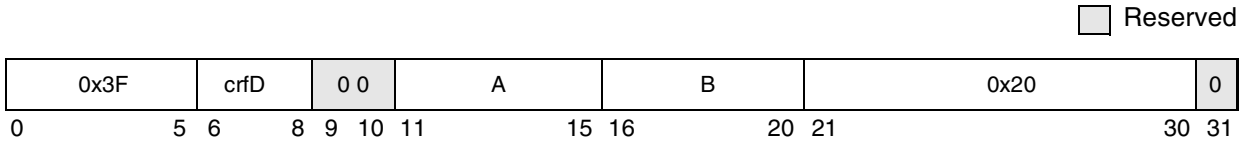
If a carry occurs, the sum's significand is shifted right one bit position and the exponent is increased by one. FPSCR[FPRF] is set to the class and sign of the result, except for invalid operation exceptions when FPSCR[VE]=1.

Other registers altered:

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

This instruction is defined by the PowerPC UISA.

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




```
if (frA) is a NaN or (frB) is a NaN
      then c←0b001
else if (frA)<(frB) then c ←0b1000
else if (frA)>(frB) then c←0b0100
else c←0b0010
FPSCR[FPCC]←c
CR[4*crfD: 4*crfD+3]←c
if (frA) is an SNaN or (frB) is an SNaN
      then FPSCR[VXSNAN]←1
      if VE=0 then FPSCR[VXVC]←1
else if (frA) is a QNaN or (frB) is a QNaN
      then FPSCR[VXVC]←1
```

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

If at least one of the operands is a NaN, either quiet or signaling, then CR Field **crf**D and FPSCR[FPCC] are set to reflect unordered. If at least one of the operands is a signaling NaN, then FPSCR[VXSNAN] is set, and if invalid operation is disabled (FPSCR[VE]=0) then FPSCR[VXVC] is set. If neither operand is a signaling NaN, but at least one is a QNaN, then FPSCR[VXVC] is set.

Other registers altered:

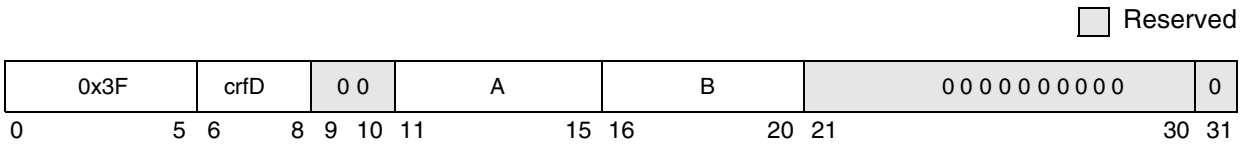
- Condition Register (CR Field specified by operand **crf**D):

  Affected: FX, FEX, VX, OX
- Floating-Point Status and Control Register:

  Affected: FPCC, FX, VXSNAN, VXVC

This instruction is defined by the PowerPC UISA.

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



if (**fr**A) is a NaN or (**fr**B) is a NaN
then c←0b001
else if (**fr**A)<(**fr**B) then c ←0b1000
else if (**fr**A)>(**fr**B) then c←0b0100
else c←0b0010
FPSCR[FPCC]←c
CR[4*crfD: 4*crfD+3]←c
if (**fr**A) is an SNaN or (**fr**B) is an SNaN
then FPSCR[VXSNAN]←1

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

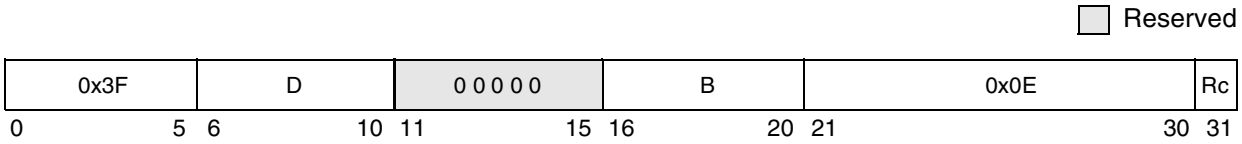
If at least one of the operands is a NaN, either quiet or signaling, then CR Field **crf**D and FPSCR[FPCC] are set to reflect unordered. If at least one of the operands is a signaling NaN, then FPSCR[VXSNAN] is set.

Other registers altered:

- Condition Register (CR Field specified by operand **crf**D):
  Affected: FX, FEX, VX, OX
- Floating-Point Status and Control Register:
  Affected: FPCC, FX, VXSNAN

This instruction is defined by the PowerPC UISA.

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



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

If the contents of **fr**B is greater than 2^{31}-1, **fr**D[32:63] are set to 0x7FFF FFFF.

If the contents of **fr**B is less than -2^{31}, **fr**D[32:63] are set to 0x8000 0000.

The conversion is described fully in **APPENDIX C FLOATING-POINT MODELS AND CONVERSIONS**.

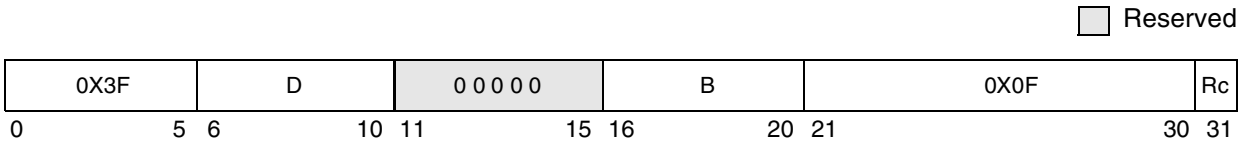
Except for trap-enabled invalid operation exceptions, FPSCR[FPRF] is undefined. FPSCR[FR] is set if the result is incremented when rounded. FPSCR[FI] is set if the result is inexact.

Other registers altered:

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

This instruction is defined by the PowerPC UISA.

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



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

If the operand in **fr**B is greater than 2^{31}-1, **fr**D[32:63] are set to 0x7FFF FFFF.

If the operand in **fr**B is less than -2^{31}, **fr**D[32:63] are set to 0x8000 0000.

The conversion is described fully in **APPENDIX C FLOATING-POINT MODELS AND CONVERSIONS**.

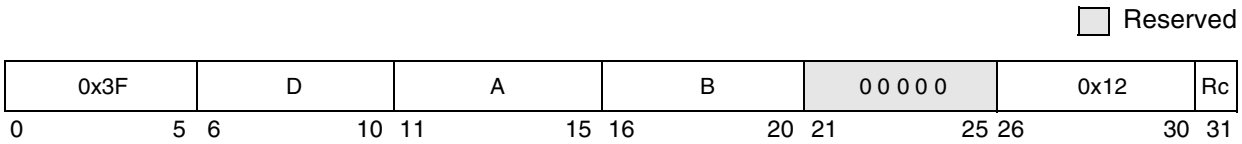
Except for trap-enabled invalid operation exceptions, FPSCR[FPRF] is undefined. FPSCR[FR] is set if the result is incremented when rounded. FPSCR[FI] is set if the result is inexact.

Other registers altered:

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

This instruction is defined by the PowerPC UISA.

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



The floating-point operand in register **fr**A is divided by the floating-point operand in register **fr**B. No remainder is preserved.

If an operand is a denormalized number then it is prenormalized before the operation is started. If the most significant bit of the resultant significand is not a one the result is normalized. The result is rounded to the target precision under control of the floating-point rounding control field RN of the FPSCR and placed into **frD**.

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

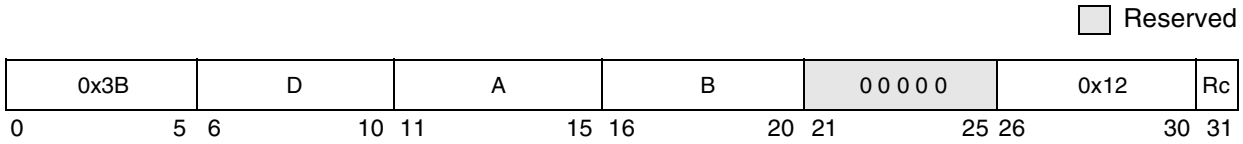
FPSCR[FPRF] is set to the class and sign of the result, except for invalid operation exceptions when FPSCR[VE]=1 and zero divide exceptions when FPSCR[ZE]=1.

Other registers altered:

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

This instruction is defined by the PowerPC UISA.

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



The floating-point operand in register **fr**A is divided by the floating-point operand in register **fr**B. No remainder is preserved.

If an operand is a denormalized number then it is prenormalized before the operation is started. If the most significant bit of the resultant significand is not a one the result is normalized. The result is rounded to the target precision under control of the floating-point rounding control field RN of the FPSCR and placed into **frD**.

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

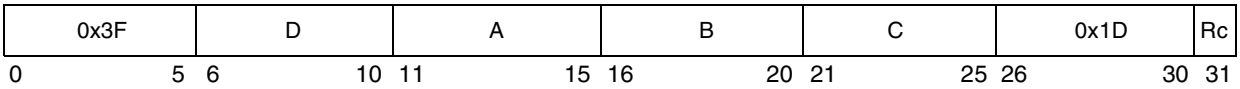
FPSCR[FPRF] is set to the class and sign of the result, except for invalid operation exceptions when FPSCR[VE]=1 and zero divide exceptions when FPSCR[ZE]=1.

Other registers altered:

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

This instruction is defined by the PowerPC UISA.

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



The following operation is performed:

frD ← [(**fr**A)*(**fr**C)]+(**fr**B)

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

If an operand is a denormalized number then it is prenormalized before the operation is started. If the most significant bit of the resultant significand is not a one the result is normalized. The result is rounded to the target precision under control of the floating-point rounding control field RN of the FPSCR and placed into **frD**.

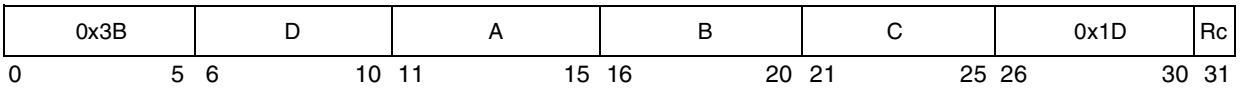
FPSCR[FPRF] is set to the class and sign of the result, except for invalid operation exceptions when FPSCR[VE]=1.

Other registers altered:

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

This instruction is defined by the PowerPC UISA.

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



The following operation is performed:

frD ← [(**fr**A)*(**fr**C)]+(**fr**B)

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

If an operand is a denormalized number then it is prenormalized before the operation is started. If the most significant bit of the resultant significand is not a one the result is normalized. The result is rounded to the target precision under control of the floating-point rounding control field RN of the FPSCR and placed into **frD**.

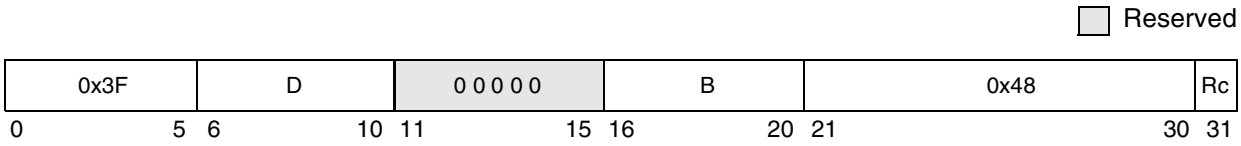
FPSCR[FPRF] is set to the class and sign of the result, except for invalid operation exceptions when FPSCR[VE]=1.

Other registers altered:

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

This instruction is defined by the PowerPC UISA.

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



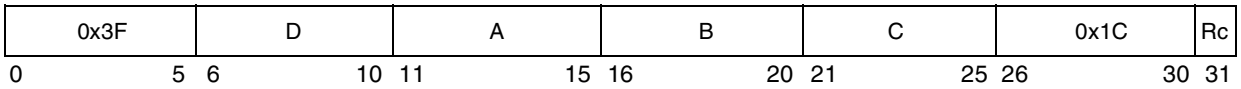
The contents of register **fr**B are placed into **frD**.

Other registers altered:

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

This instruction is defined by the PowerPC UISA.

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



The following operation is performed:

frD ← [(**fr**A)*(**fr**C)] - (**fr**B)

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

If an operand is a denormalized number then it is prenormalized before the operation is started. If the most significant bit of the resultant significand is not a one the result is normalized. The result is rounded to the target precision under control of the floating-point rounding control field RN of the FPSCR and placed into **frD**.

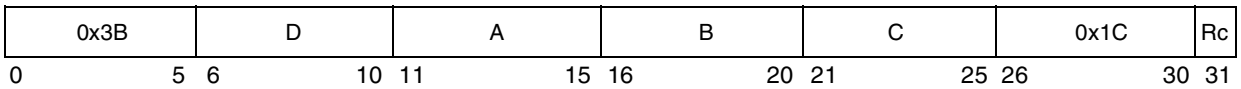
FPSCR[FPRF] is set to the class and sign of the result, except for invalid operation exceptions when FPSCR[VE]=1.

Other registers altered:

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

This instruction is defined by the PowerPC UISA.

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



The following operation is performed:

frD ← [(**fr**A)*(**fr**C)] - (**fr**B)

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

If an operand is a denormalized number then it is prenormalized before the operation is started. If the most significant bit of the resultant significand is not a one the result is normalized. The result is rounded to the target precision under control of the floating-point rounding control field RN of the FPSCR and placed into **fr**D.

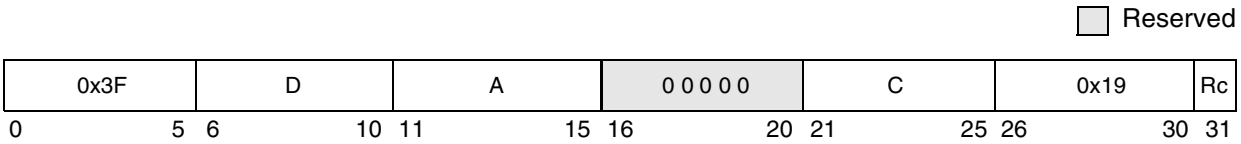
FPSCR[FPRF] is set to the class and sign of the result, except for invalid operation exceptions when FPSCR[VE]=1.

Other registers altered:

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

This instruction is defined by the PowerPC UISA.

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



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

If an operand is a denormalized number then it is prenormalized before the operation is started. If the most significant bit of the resultant significand is not a one the result is normalized. The result is rounded to the target precision under control of the floating-point rounding control field RN of the FPSCR and placed into **fr**D.

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

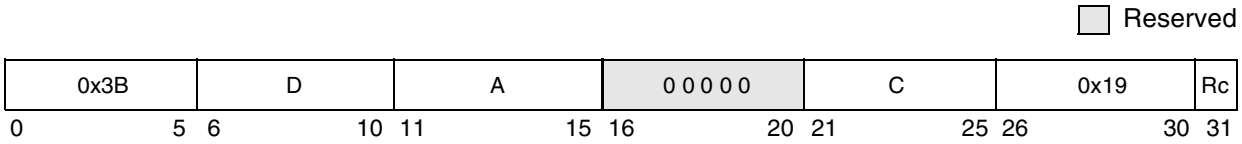
FPSCR[FPRF] is set to the class and sign of the result, except for invalid operation exceptions when FPSCR[VE]=1.

Other registers altered:

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

This instruction is defined by the PowerPC UISA.

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



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

If an operand is a denormalized number then it is prenormalized before the operation is started. If the most significant bit of the resultant significand is not a one the result is normalized. The result is rounded to the target precision under control of the floating-point rounding control field RN of the FPSCR and placed into **fr**D.

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

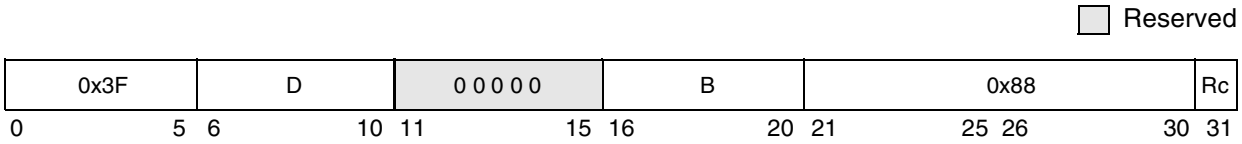
FPSCR[FPRF] is set to the class and sign of the result, except for invalid operation exceptions when FPSCR[VE]=1.

Other registers altered:

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

This instruction is defined by the PowerPC UISA.

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



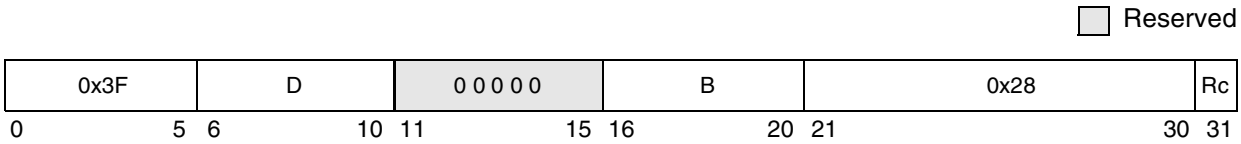
The contents of register **fr**B, with bit 0 set to one, are placed into **frD**.

Other registers altered:

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

This instruction is defined by the PowerPC UISA.

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



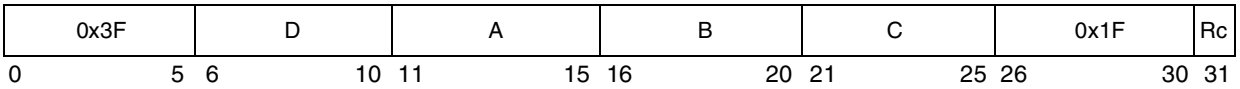
The contents of register **fr**B, with bit 0 inverted, are placed into **frD**.

Other registers altered:

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

This instruction is defined by the PowerPC UISA.

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



The following operation is performed:

frD ← -([(**fr**A)*(**fr**C)]+(**fr**B))

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

This instruction produces the same result as would be obtained by using the floating-point multiply-add instruction and then negating the result, with the following exceptions:

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

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

Other registers altered:

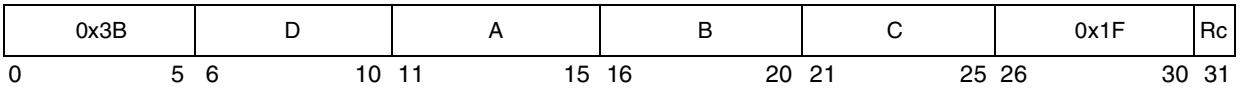
- Condition Register (CR1 Field):

  Affected: FX, FEX, VX, OX (if Rc=1)
- Floating-point Status and Control Register:

  Affected: FPRF, FR, FI, FX, OX, UX, XX, VXSNAN, VXISI, VXIMZ

This instruction is defined by the PowerPC UISA.

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



The following operation is performed:

frD ← -([(**fr**A)*(**fr**C)]+(**fr**B))

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

This instruction produces the same result as would be obtained by using the floating-point multiply-add instruction and then negating the result, with the following exceptions:

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

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

Other registers altered:

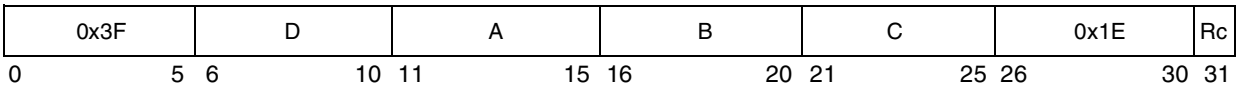
- Condition Register (CR1 Field):

  Affected: FX, FEX, VX, OX (if Rc=1)
- Floating-point Status and Control Register:

  Affected: FPRF, FR, FI, FX, OX, UX, XX, VXSNAN, VXISI, VXIMZ

This instruction is defined by the PowerPC UISA.

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



The following operation is performed:

frD ← -([(**fr**A)*(**fr**C)] - (**fr**B))

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

If an operand is a denormalized number, it is prenormalized before the operation is started. If the most significant bit of the resultant significand is not one, the result is normalized. The result is rounded to the target precision under control of the floating-point rounding control field RN of the FPSCR, then negated and placed into **frD**.

This instruction produces the same result obtained by negating the result of a floating multiply-subtract instruction with the following exceptions:

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

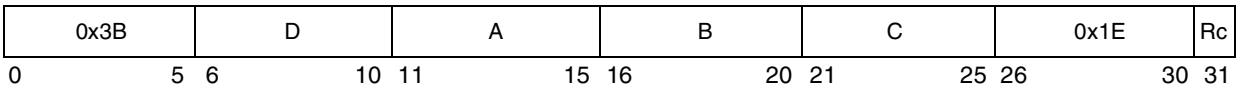
FPSCR[FPRF] is set to the class and sign of the result, except for invalid operation exceptions when FPSCR[VE]=1.

Other registers altered:

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

This instruction is defined by the PowerPC UISA.

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



The following operation is performed:

frD ← -([(**fr**A)*(**fr**C)] - (**fr**B))

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

If an operand is a denormalized number, it is prenormalized before the operation is started. If the most significant bit of the resultant significand is not one, the result is normalized. The result is rounded to the target precision under control of the floating-point rounding control field RN of the FPSCR, then negated and placed into **fr**D.

This instruction produces the same result obtained by negating the result of a floating multiply-subtract instruction with the following exceptions:

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

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

Other registers altered:

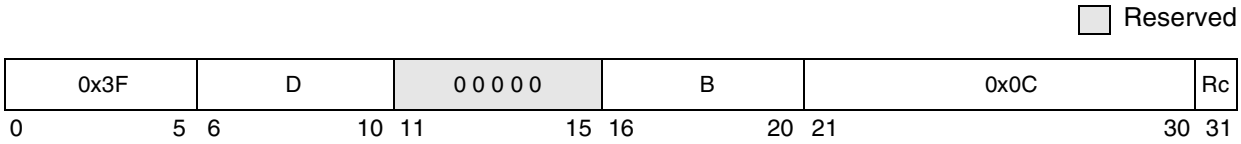
- Condition Register (CR1 Field)

  Affected: FX, FEX, VX, OX (if Rc=1)
- Floating-point Status and Control Register:

  Affected: FPRF, FR, FI, FX, OX, UX, XX, VXSNAN, VXISI, VXIMZ

This instruction is defined by the PowerPC UISA.

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



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

The rounding is described fully in **APPENDIX C FLOATING-POINT MODELS AND CONVERSIONS**.

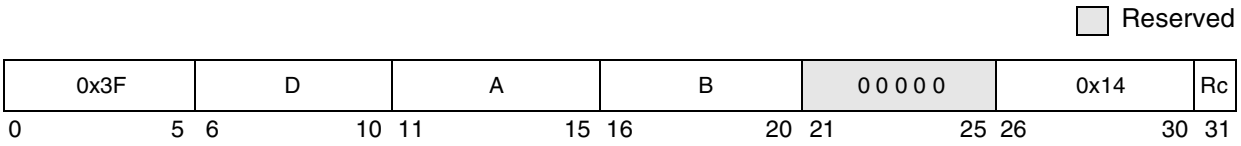
FPSCR[FPRF] is set to the class and sign of the result, except for invalid operation exceptions when FPSCR[VE]=1.

Other registers altered:

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

This instruction is defined by the PowerPC UISA.

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



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

The execution of the floating-point subtract instruction is identical to that of floating-point add, except that the contents of **fr**B participates in the operation with its sign bit (bit 0) inverted.

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

Other registers altered:

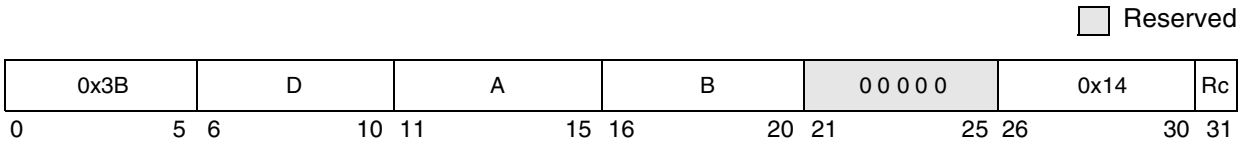
- Condition Register (CR1 Field):

  Affected: FX, FEX, VX, OX (if Rc=1)
- Floating-point Status and Control Register:

  Affected: FPRF, FR, FI, FX, OX, UX, XX, VXSNAN, VXISI

This instruction is defined by the PowerPC UISA.

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



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

The execution of the floating-point subtract instruction is identical to that of floating-point add, except that the contents of **fr**B participates in the operation with its sign bit (bit 0) inverted.

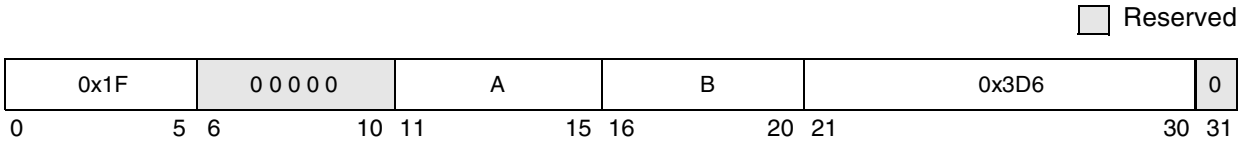
FPSCR[FPRF] is set to the class and sign of the result, except for invalid operation exceptions when FPSCR[VE]=1.

Other registers altered:

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

This instruction is defined by the PowerPC UISA.

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



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

If a block containing the byte addressed by EA is in the instruction cache of this processor, the block is made invalid in the processor. Subsequent references cause the block to be refetched.

NOTE

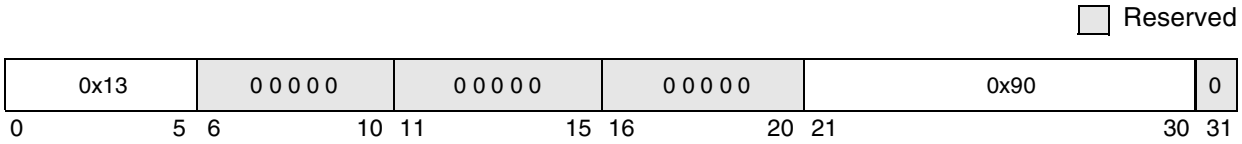
According to the PowerPC architecture, if the addressed block is in coherency-required mode, the block is made invalid in all affected processors. In the RCPU, however, all instruction memory is considered to be in coherency-not-required mode.

Other registers altered:

- None

This instruction is defined by the PowerPC VEA.

isync



Fetch of an **isync** instruction causes fetch serialization: instruction fetch is halted until all instructions currently in the processor (i.e., all issued instructions as well as the pre-fetched instructions waiting to be issued) have completed execution. This instruction causes subsequent instructions to execute in the context established by the previous instructions.

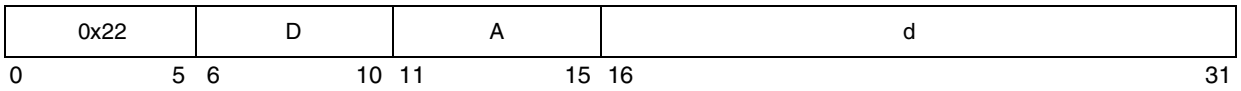
This instruction has no effect on other processors or on their caches.

Other registers altered:

- None

This instruction is defined by the PowerPC VEA.

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



if **r**A=0 then b ← 0
else b ← (**r**A)
EA ← b+EXTS(d)
rD ← (24)0 || MEM(EA, 1)

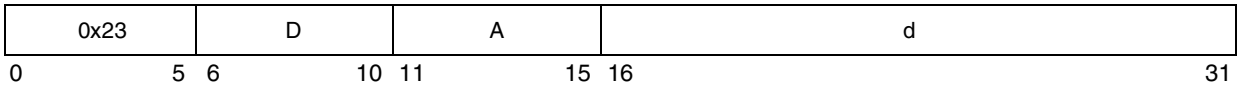
The effective address is the sum (**r**A|0) + d. The byte in memory addressed by EA is loaded into **r**D[24:31]. Bits **r**D[0:23] are cleared to zero.

Other registers altered:

- None

This instruction is defined by the PowerPC UISA.

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



EA ← (**r**A)+EXTS(d)
rD←(24)0 || MEM(EA, 1)
rA←EA

EA is the sum (**r**A|0) + d. The byte in memory addressed by EA is loaded into **r**D[24:31]. Bits **r**D[0:23] are cleared to zero.

EA is placed into **r**A.

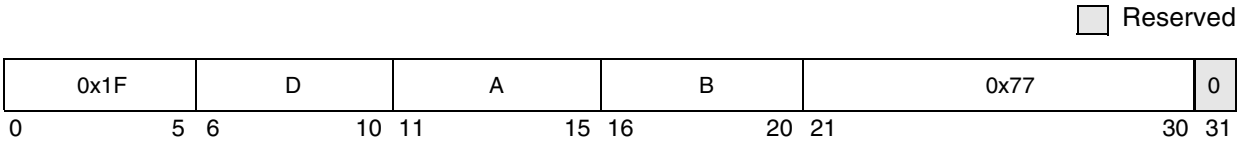
If **r**A=0 or **r**A=**r**D, the instruction form is invalid.

Other registers altered:

- None

This instruction is defined by the PowerPC UISA.

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



EA ← (**r**A)+(**r**B)
rD ← (24)0 || MEM(EA, 1)
rA ← EA

EA is the sum (**r**A|0) + (**r**B). The byte addressed by EA is loaded into **r**D[24:31]. Bits **r**D[0:23] are cleared to zero.

EA is placed into **r**A.

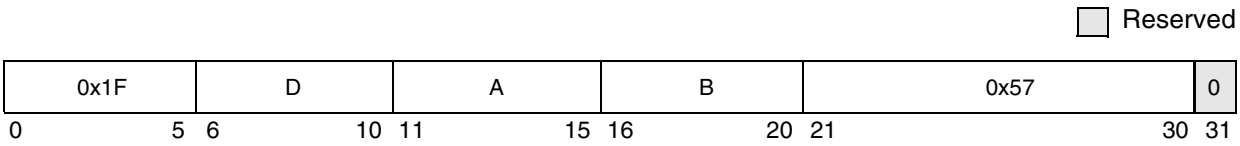
If **r**A=0 or **r**A=**r**D, the instruction form is invalid.

Other registers altered:

- None

This instruction is defined by the PowerPC UISA.

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



if **r**A=0 then b ← 0
else b ← (**r**A)
EA ← b+(**r**B)
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 **r**D[24:31].

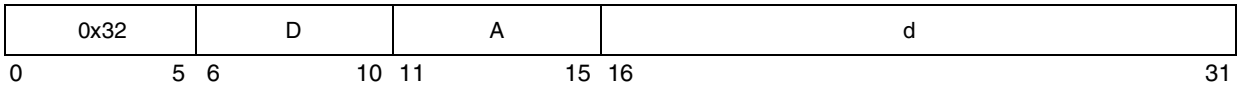
Bits **r**D[0:23] are cleared to zero.

Other registers altered:

- None

This instruction is defined by the PowerPC UISA.

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



if **r**A=0 then b ← 0
else b ← (**r**A)
EA ← b+EXTS(d)
frD ← MEM(EA, 8)

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

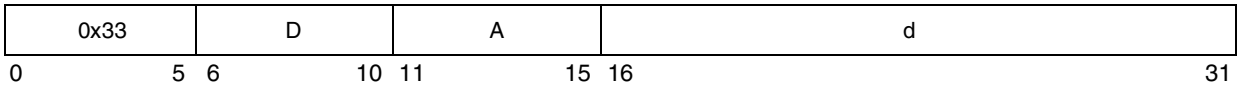
The double word in memory addressed by EA is placed into **frD**.

Other registers altered:

- None

This instruction is defined by the PowerPC UISA.

lfdu **frD,**d**(r**A**)**



EA ← (**r**A)+EXTS(d)
frD ← MEM(EA, 8)
rA ← EA

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

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

EA is placed into **r**A.

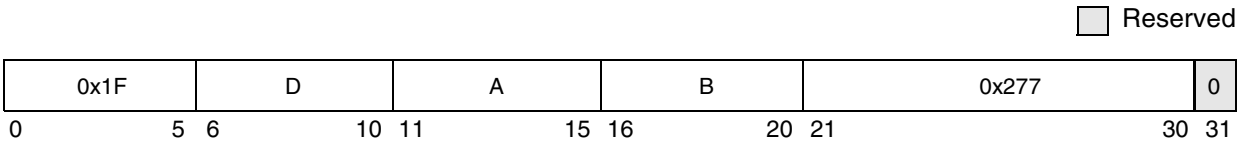
If **r**A=0, the instruction form is invalid.

Other registers altered:

- None

This instruction is defined by the PowerPC UISA.

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



EA ← (**r**A)+(**r**B)
frD ← MEM(EA, 8)
rA ← EA

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

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

EA is placed into **r**A.

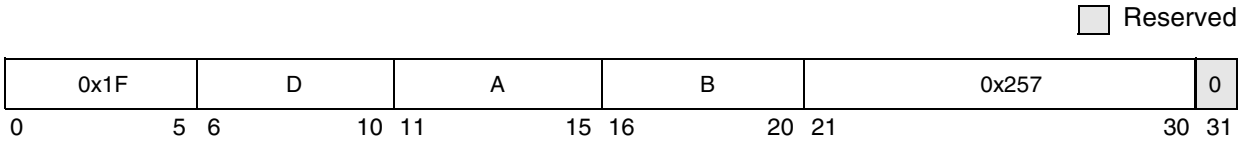
If **r**A=0, the instruction form is invalid.

Other registers altered:

- None

This instruction is defined by the PowerPC UISA.

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



if **r**A=0 then b ← 0
else b ← (**r**A)
EA ← b+(**r**B)
frD ← MEM(EA, 8)

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

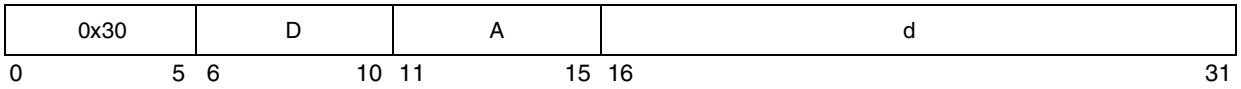
The double word in memory addressed by EA is placed into **frD**.

Other registers altered:

- None

This instruction is defined by the PowerPC UISA.

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



if **r**A=0 then b ← 0
else b ← (**r**A)
EA ← b+EXTS(d)
frD ← DOUBLE(MEM(EA, 4))

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

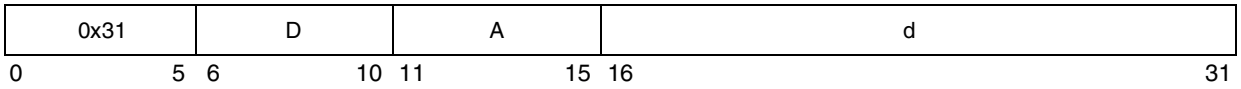
The word in memory addressed by EA is interpreted as a floating-point single-precision operand. This word is converted to floating-point double-precision (see **4.5.8.1 Double-Precision Conversion for Floating-Point Load Instructions**) and placed into **frD**.

Other registers altered:

- None

This instruction is defined by the PowerPC UISA.

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



EA ← (**r**A)+EXTS(d)
frD ← DOUBLE(MEM(EA, 4))
rA ← EA

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

The word in memory addressed by EA is interpreted as a floating-point single-precision operand. This word is converted to floating-point double-precision (see **4.5.8.1 Double-Precision Conversion for Floating-Point Load Instructions**) and placed into **frD**.

EA is placed into **r**A.

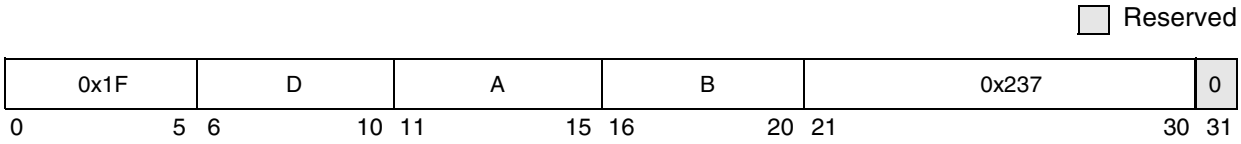
If **r**A=0, the instruction form is invalid.

Other registers altered:

- None

This instruction is defined by the PowerPC UISA.

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



EA ← (**r**A)+(**r**B)
frD ← DOUBLE(MEM(EA, 4))
rA ← EA

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

The word in memory addressed by EA is interpreted as a floating-point single-precision operand. This word is converted to floating-point double-precision (see **4.5.8.1 Double-Precision Conversion for Floating-Point Load Instructions**) and placed into **frD**.

EA is placed into **r**A.

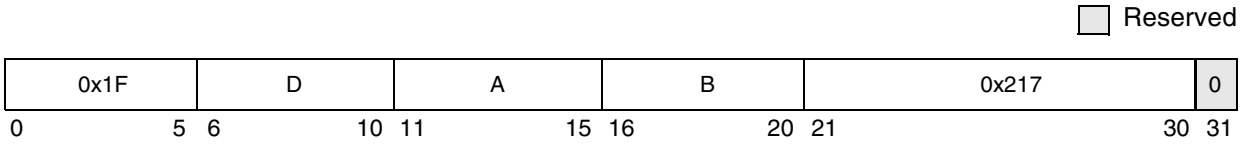
If **r**A=0, the instruction form is invalid.

Other registers altered:

- None

This instruction is defined by the PowerPC UISA.

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



if **r**A=0 then b ← 0
else b ← (**r**A)
EA ← b+(**r**B)
frD ← DOUBLE(MEM(EA, 4))

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

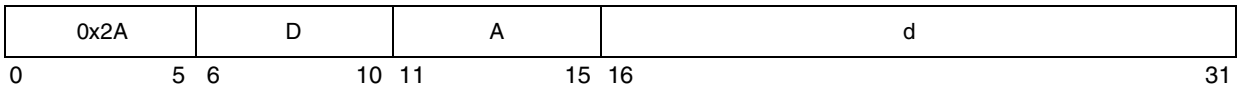
The word in memory addressed by EA is interpreted as a floating-point single-precision operand. This word is converted to floating-point double-precision (see **4.5.8.1 Double-Precision Conversion for Floating-Point Load Instructions**) and placed into **fr**D.

Other registers altered:

- None

This instruction is defined by the PowerPC UISA.

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



if **r**A=0 then b ← 0
else b ← (**r**A)
EA ← b+EXTS(d)
rD ← EXTS(MEM(EA, 2))

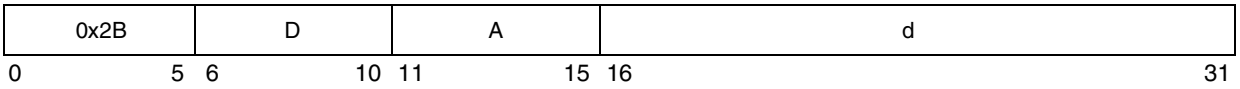
EA is the sum (**r**A|0) + d. The half word in memory addressed by EA is loaded into **r**D[16:31]. Bits **r**D[0:15] are filled with a copy of bit 0 of the loaded half word.

Other registers altered:

- None

This instruction is defined by the PowerPC UISA.

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



EA ← (**r**A)+EXTS(d)
rD ← EXTS(MEM(EA, 2))
rA ← EA

EA is the sum (**r**A|0) + d. The half word in memory addressed by EA is loaded into **r**D[16:31].

Bits **r**D[0:15] are filled with a copy of bit 0 of the loaded half word.

EA is placed into **r**A.

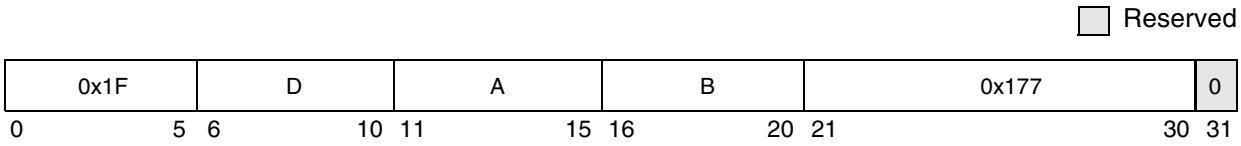
If **r**A=0 or **r**A=**r**D, the instruction form is invalid.

Other registers altered:

- None

This instruction is defined by the PowerPC UISA.

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



EA ← (**r**A)+(**r**B)
rD ← EXTS(MEM(EA, 2))
rA ← EA

EA is the sum (**r**A|0) + (**r**B). The half word in memory addressed by EA is loaded into **r**D[16:31]. Bits **r**D[0:15] are filled with a copy of bit 0 of the loaded half word.

EA is placed into **r**A.

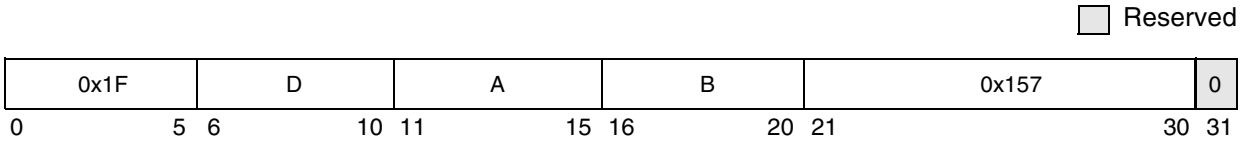
If **r**A=0 or **r**A=**r**D, the instruction form is invalid.

Other registers altered:

- None

This instruction is defined by the PowerPC UISA.

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



if **r**A=0 then b ← 0
else b ← (**r**A)
EA ← b+(**r**B)
rD ← EXTS(MEM(EA, 2))

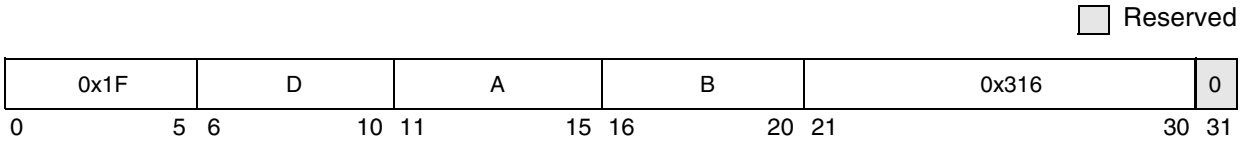
EA is the sum (**r**A|0) + (**r**B). The half word in memory addressed by EA is loaded into **r**D[16:31]. Bits **r**D[0:15] are filled with a copy of bit 0 of the loaded half word.

Other registers altered:

- None

This instruction is defined by the PowerPC UISA.

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



if **r**A=0 then b ← 0
else b ← (**r**A)
EA ← b+(**r**B)
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 **r**D[24:31]. Bits 8:15 of the half word in memory addressed by EA are loaded into **r**D[16:23]. Bits **r**D[0:15] are cleared to zero.

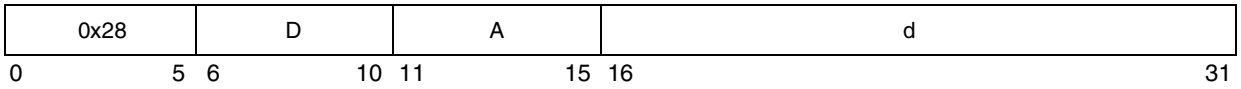
Some PowerPC implementations may run the **lhbrx** instructions with greater latency than other types of load instructions. This is not the case in the RCPU. This instruction operates with the same latency as other load instructions.

Other registers altered:

- None

This instruction is defined by the PowerPC UISA.

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



if **r**A=0 then b←0
else b ← (**r**A)
EA ← b+EXTS(d)
rD ← (16)0 || MEM(EA, 2)

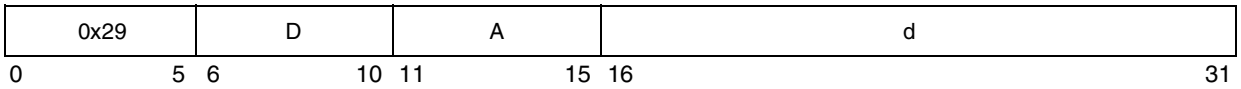
EA is the sum (**r**A|0) + d. The half word in memory addressed by EA is loaded into **r**D[16:31]. Bits **r**D[0:15] are cleared to zero.

Other registers altered:

- None

This instruction is defined by the PowerPC UISA.

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



EA ← (**r**A)+EXTS(d)
rD ← (16)0 || MEM(EA, 2)
rA ← EA

EA is the sum (**r**A|0) + d. The half word in memory addressed by EA is loaded into **r**D[16:31]. Bits **r**D[0:15] are cleared to zero.

EA is placed into **r**A.

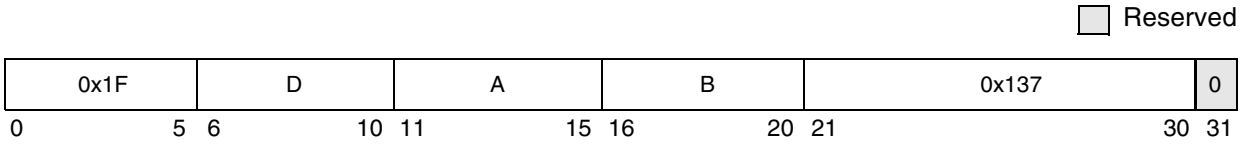
If **r**A=0 or **r**A=**r**D, the instruction form is invalid.

Other registers altered:

- None

This instruction is defined by the PowerPC UISA.

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



EA ← (**r**A)+(**r**B)
rD←(16)0 || MEM(EA, 2)
rA←EA

EA is the sum (**r**A|0) + (**r**B). The half word in memory addressed by EA is loaded into **r**D[16:31]. Bits **r**D[0:15] are cleared to zero.

EA is placed into **r**A.

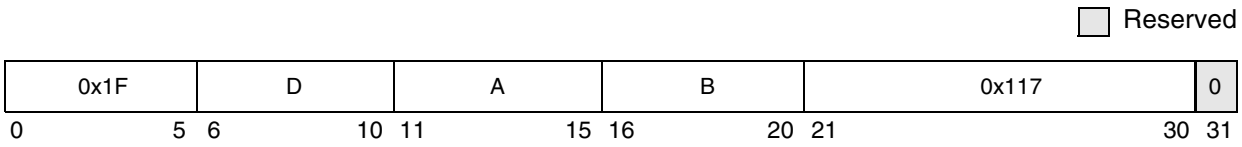
If **r**A=0 or **r**A=**r**D, the instruction form is invalid.

Other registers altered:

- None

This instruction is defined by the PowerPC UISA.

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



if **r**A=0 then b←0
else b←(**r**A)
EA←b+(**r**B)
rD←(16)0 || MEM(EA, 2)

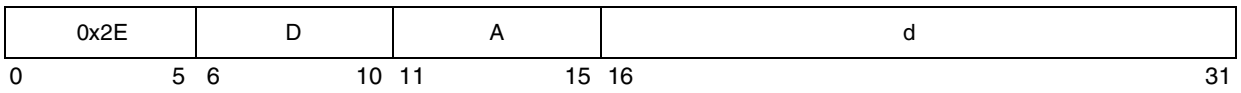
The effective address is the sum (**r**A|0) + (**r**B). The half word in memory addressed by EA is loaded into **r**D[16:31]. Bits **r**D[0:15] are cleared to zero.

Other registers altered:

- None

This instruction is defined by the PowerPC UISA.

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




```
if rA=0 then b←0
else b←(rA)
EA←b+EXTS(d)
r←rD
do while r ð 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 the 32 bits of GPRs **r**D through **r**31. EA must be a multiple of four; otherwise, the system alignment exception handler is invoked.

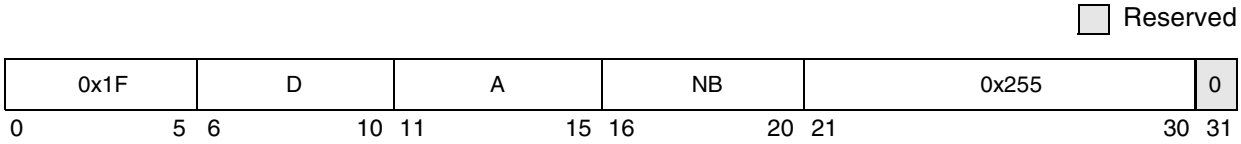
If **r**A is in the range of registers specified to be loaded, including the case in which**r**A = 0, the instruction form is invalid.

Other registers altered:

- None

This instruction is defined by the PowerPC UISA.

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




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

The EA is (rA|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 the 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 **r**0 if required. If the four bytes of register**r**D+*nr*-1 are only partially filled, the unfilled low-order byte(s) of that register are cleared to zero.

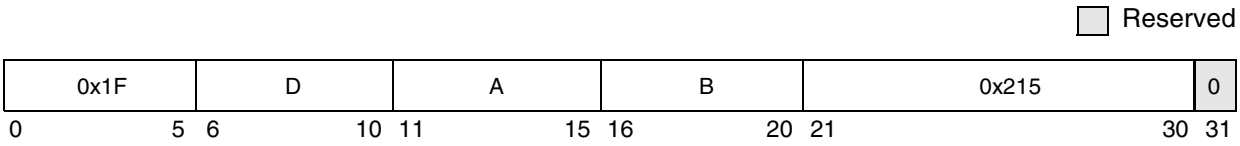
If **r**A is in the range of registers specified to be loaded, including the case in which**r**A = 0, the instruction form is invalid.

Other registers altered:

- None

This instruction is defined by the PowerPC UISA.

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
do while n Š 0
     if i=32 then
          r←r+1 (mod 32)
          GPR(r)←0
     GPR(r)[i:i+7]←MEM(EA, 1)
     i←i+8
     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 to **r**0 if required. If the bytes of **r**D+*nr*-1 are only partially filled, the unfilled low-order byte(s) of that register are cleared to zero.

If *n*=0, the content of **r**D is 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.

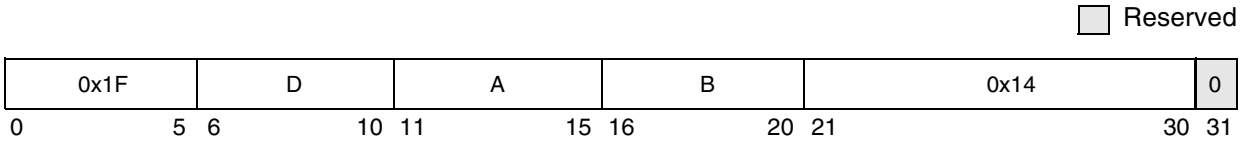
If **r**D = **r**A or **r**D = **r**B, the instruction form is invalid.

Other registers altered:

- None

This instruction is defined by the PowerPC UISA.

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



if **r**A=0 then b←0
else b←(**r**A)
EA←b+(**r**B)
RESERVE←1
RESERVE_ADDR←func(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 instruction. An address computed from the EA is associated with the reservation, and replaces any address previously associated with the reservation: the manner in which the address to be associated with the reservation is computed from the EA is described in **4.1.2 Addressing Modes and Effective Address Calculation**.

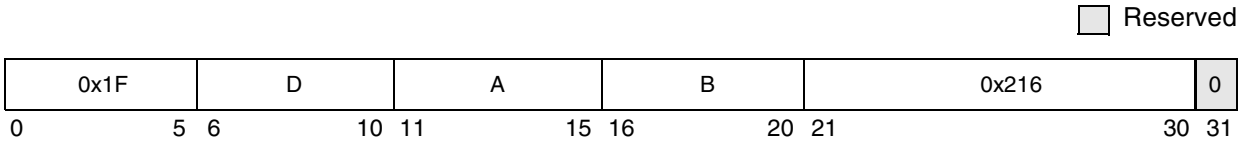
If the EA is not a multiple of four, the alignment exception handler is invoked.

Other registers altered:

- None

This instruction is defined by the PowerPC UISA.

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



if **r**A=0 then b←0
else b←(**r**A)
EA←b+(**r**B)
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 **r**D[24:31]. Bits 8:15 of the word in memory addressed by EA are loaded into **r**D[16:23]. Bits 16:23 of the word in memory addressed by EA are loaded into **r**D[8:15]. Bits 24:31 of the word in memory addressed by EA are loaded into **r**D[0:7].

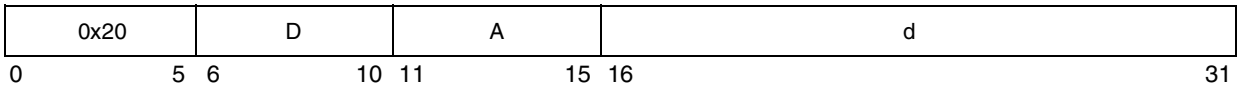
Some PowerPC implementations may run the **lwbrx** instructions with greater latency than other types of load instructions. This is not the case in the RCPU. This instruction operates with the same latency as other load instructions.

Other registers altered:

- None

This instruction is defined by the PowerPC UISA.

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



if **r**A=0 then b←0
else b←(**r**A)
EA←b+EXTS(d)
rD←MEM(EA, 4)

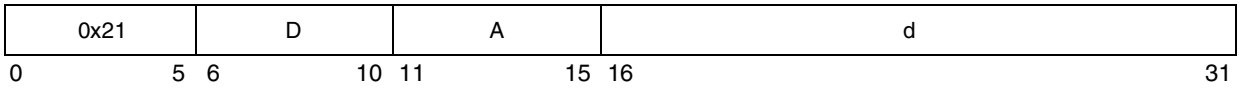
EA is the sum (**r**A|0) + d. The word in memory addressed by EA is loaded into **r**D.

Other registers altered:

- None

This instruction is defined by the PowerPC UISA.

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



EA ← (**r**A)+EXTS(d)
rD←MEM(EA, 4)
rA←EA

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

EA is placed into **r**A.

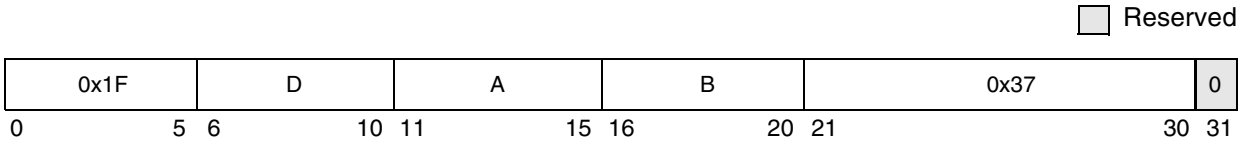
If **r**A=0 or **r**A=**r**D, the instruction form is invalid.

Other registers altered:

- None

This instruction is defined by the PowerPC UISA.

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



EA ← (**r**A)+(**r**B)
rD←MEM(EA, 4)
rA←EA

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

EA is placed into **r**A.

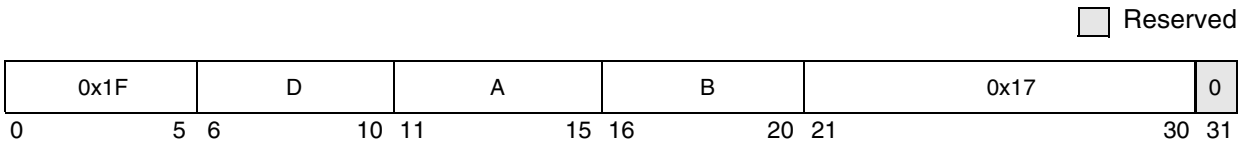
If **r**A=0 or **r**A=**r**D, the instruction form is invalid.

Other registers altered:

- None

This instruction is defined by the PowerPC UISA.

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



if **r**A=0 then b←0
else b←(**r**A)
EA←b+(**r**B)
rD←MEM(EA, 4)

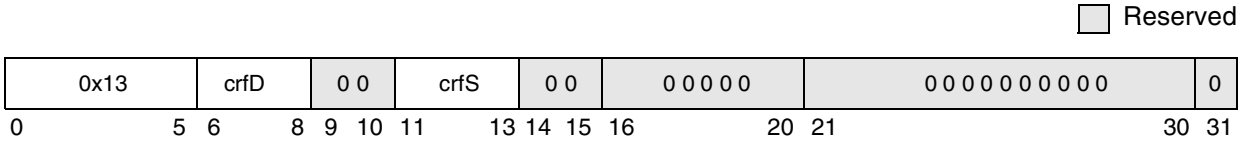
EA is the sum (**r**A|0) + (**r**B). The word in memory addressed by EA is loaded into **r**D.

Other registers altered:

- None

This instruction is defined by the PowerPC UISA.

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



CR[4∗**crf**D:4∗**crf**D+3] ← CR[4∗**crf**S:4∗**crf**S+3]

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

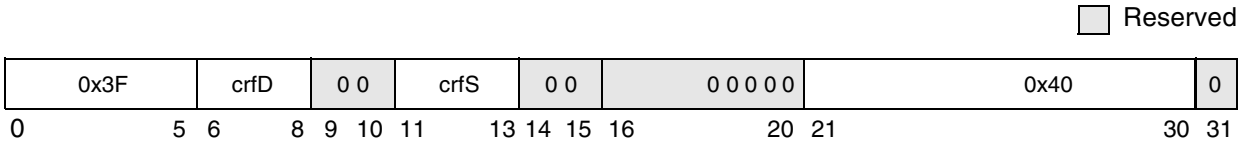
Other registers altered:

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

  Affected: LT, GT, EQ, SO

This instruction is defined by the PowerPC UISA.

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



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

Other registers altered:

- Condition Register (CR Field specified by operand **crf**S):

  Affected: FX, OX (if **crf**S=0)

  Affected: UX, ZX, XX, VXSNAN (if **crf**S=1)

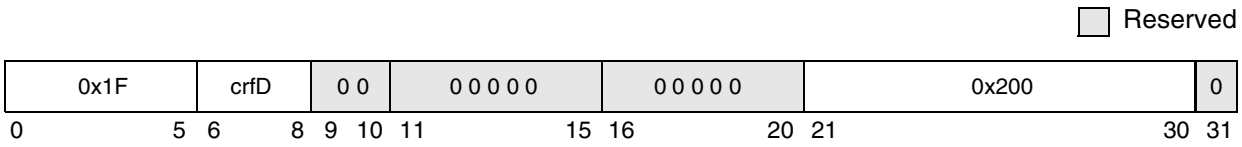
  Affected: VXISI, VXIDI, VXZDZ, VXIMZ (if **crf**S=2)

  Affected: VXVC (if **crf**S=3)

  Affected: VXSOFT, VXSQRT, VXCVI (if **crf**S=5)

This instruction is defined by the PowerPC UISA.

mcrxr **crf**D



CR[4∗**crf**D:4∗**crf**D+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 to zero.

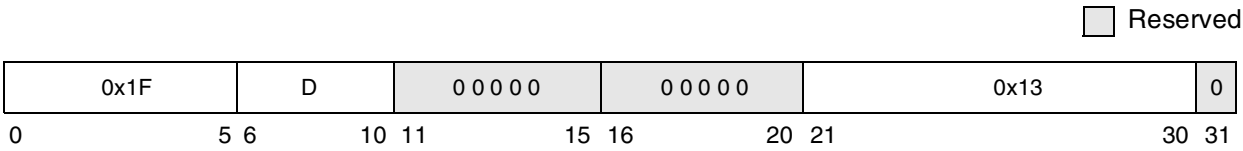
Other registers altered:

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

  Affected: LT, GT, EQ, SO
- XER[0:3]

This instruction is defined by the PowerPC UISA.

mfcr **r**D



rD← CR

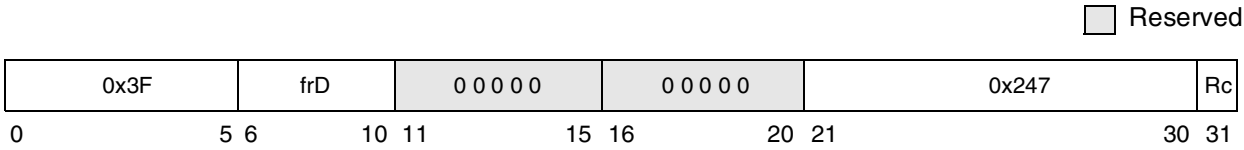
The contents of the condition register are placed into **r**D.

Other registers altered:

- None

This instruction is defined by the PowerPC UISA.

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



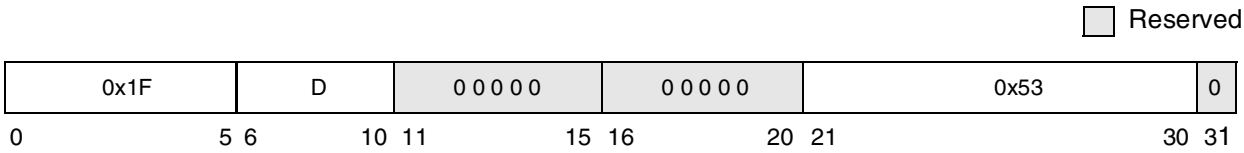
The contents of the FPSCR are placed into **fr**D[32:63]. **fr**D[0:31] are undefined.

Other registers altered:

- Condition Register (CR1 Field):
  Affected: LT, GT, EQ, SO (if Rc=1)

This instruction is defined by the PowerPC UISA.

mfmsr **r**D



rD← MSR

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

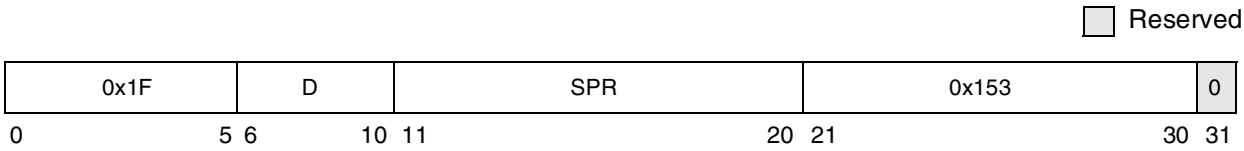
This is a supervisor-level instruction.

Other registers altered:

- None

This instruction is defined by the PowerPC OEA.

mfspr **r**D,SPR



n←SPR[5:9] ||SPR[0:4]
rD← SPR(*n*)

The SPR field denotes a special purpose register, encoded as shown in **Table 4-29**, **Table 4-30**, and **Table 4-31**. The contents of the designated special purpose register are placed into **r**D.

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 5 bits appearing in bits 16 to 20 of the instruction and the low-order 5 bits in bits 11 to 15.

If the SPR field contains any value other than one of the values shown in one of the tables listed above, one of the following occurs:

- The system illegal instruction error handler is invoked.
- The system supervisor-level instruction error handler is invoked.
- The system software emulation exception handler is invoked.

The value of SPR[0] is one if and only if reading the register is at the supervisor level. Execution of this instruction specifying a supervisor-level register when MSR[PR]=1 will result in a supervisor-level instruction type program exception or a software emulation exception. Refer to **SECTION 6 EXCEPTIONS** for details.

If the SPR field contains a value that is not valid for the RCPU, the instruction form is invalid. For an invalid instruction form in which SPR[0]=1, if MSR[PR]=1 a supervisor-level instruction type program exception will occur instead of a no-op.

The execution unit that executes the **mfspr** instruction depends on the SPR. Moves from the XER and from SPRs that are physically implemented outside the processor are handled by the LSU. Moves from the FPSCR and FPECR are executed by the FPU. In all other cases, the BPU executes the **mfspr** instruction.

Other registers altered:

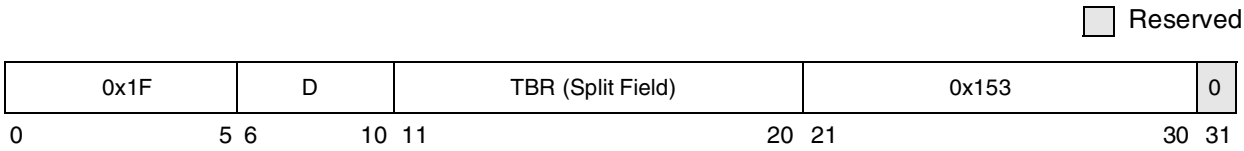
- None

Table 9-19 Simplified Mnemonics for mfspr Instruction

Operation	Simplified Mnemonic	Equivalent To
Move from XER	**mfxer** r**D**	**mfspr** r**D**,**1**
Move from LR	**mflr** r**D**	**mfspr** r**D**,**8**
Move from CTR	**mfctr** r**D**	**mfspr** r**D**,**9**
Move from DSISR	**mfdsisr** r**D**	**mfspr** r**D**,**18**
Move from DAR	**mfdar** r**D**	**mfspr** r**D**,**19**
Move from DEC	**mfdec** r**D**	**mfspr** r**D**,**22**
Move from SRR0	**mfsrr0** r**D**	**mfspr** r**D**,**26**
Move from SRR1	**mfsrr1** r**D**	**mfspr** r**D**,**27**
Move from SPRG	**mfsprg** r**D**, *n*	**mfspr** r**D**,**272+***n*
Move from TBL	**mftb** r**D**	**mftb** r**D**,**268**
Move from TBU	**mftbu** r**D**	**mftb** r**D**,**269**
Move from PVR	**mfpvr** r**D**	**mfspr** r**D**,**287**

This instruction is defined by the PowerPC UISA.

mftb **r**D,TBR



n←TBR[5:9] ||TBR[0:4]
if *n* = 268 then
rD ← TBL
else if *n* = 269 then
rD← TBU

The TBR field denotes either the time base lower (TBL) or time base upper (TBU), encoded as shown in **Table 9-20**. Notice that the order of the two 5-bit halves of the TBR number is reversed in the instruction. The contents of the designated register are copied into **r**D.

Table 9-20 TBR Encodings for mftb

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

If the TBR field contains any value other than one of the values shown in**Table 9-20**, 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

Note that **mftb** serves as both a basic and a simplified mnemonic. The assembler recognized an **mftb** mnemonic with two operands as the basic form and an **mftb** mnemonic with one operand as the simplified form. If **mftb** is coded with one operand, then that operand is assumed to be **r**D, and TBR defaults to the value corresponding to TBL.

Other registers altered:

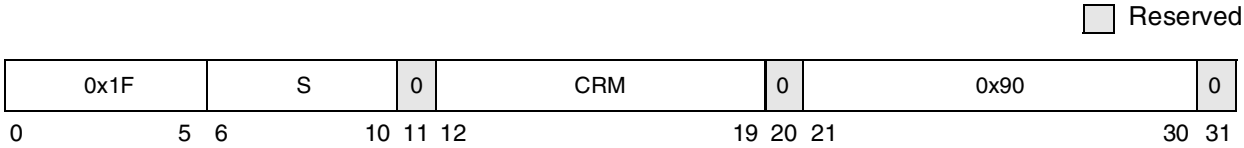
- None

This instruction is defined by the PowerPC VEA.

Table 9-21 Simplified Mnemonics for mfspr Instruction

Operation	Simplified Mnemonic	Equivalent To
Move from TBL	**mftb r**D	**mftb r**D,**268**
Move from TBU	**mftbu r**D	**mftb r**D,**269**

mtcrf CRM,**r**S



mask←(4)(CRM[0]) || (4)(CRM[1]) ||... (4)(CRM[7])
CR←((**r**S) & 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.

Other registers altered:

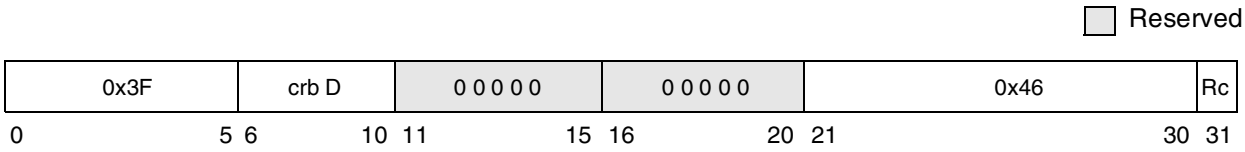
- CR fields selected by mask

Table 9-22 Simplified Mnemonics for mtcrf Instruction

Operation	Simplified Mnemonic	Equivalent To
Move to condition register	**mtcr r**S	**mtcrf 0xFF,r**S

This instruction is defined by the PowerPC UISA.

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



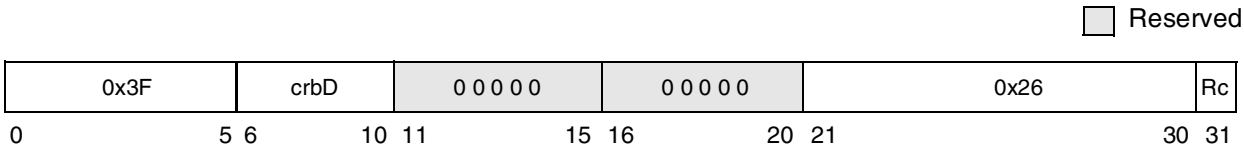
Bit **crb**D of the FPSCR is cleared to zero. All other bits of the FPSCR are unchanged.

Other registers altered:

- Condition Register (CR1 Field):
  Affected: FX, FEX, VX, OX (if Rc=1)
- Floating-point Status and Control Register:
  Affected: FPSCR bit **crb**D
  Note: Bits 1 and 2 (FEX and VX) cannot be explicitly reset.

This instruction is defined by the PowerPC UISA.

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



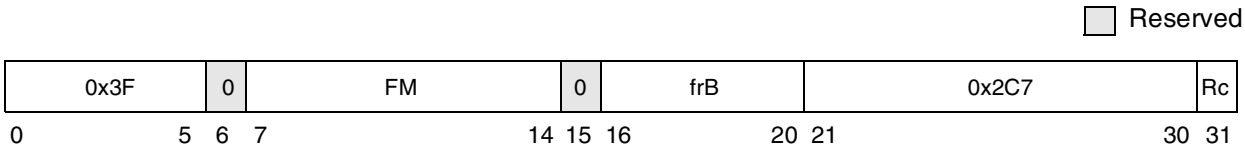
Bit **crb**D of the FPSCR is set to one. All other bits of the FPSCR are unchanged.

Other registers altered:

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

This instruction is defined by the PowerPC UISA.

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



frB[32:63] are placed into the FPSCR under control of the field mask specified by FM. The field mask identifies the 4-bit fields affected. Let i be an integer in the range 0–7. If FM(i)=1, FPSCR Field i (FPSCR bits 4∗i through 4∗i+3) is set to the contents of the corresponding field of the low-order 32 bits of register **fr**B.

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

In some PowerPC implementations, updating fewer than all eight fields of the FPSCR may have substantially poorer performance than updating all the fields. This is not the case with the RCPU.

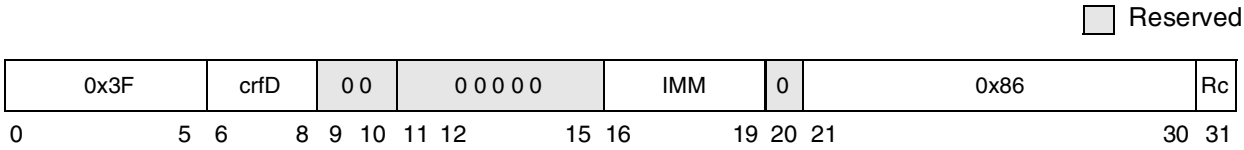
When FPSCR[0:3] is specified, bits 0 (FX) and 3 (OX) are set to the values of**fr**B[32] and **fr**B[35] (i.e., even if this instruction causes OX to change from zero to one, FX is set from **fr**B[32] and not by the usual rule that FX is set to one when an exception bit changes from zero to one). Bits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

Other registers altered:

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

This instruction is defined by the PowerPC UISA.

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



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

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

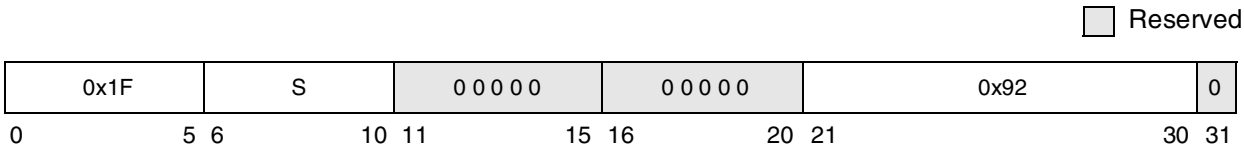
When FPSCR[0:3] is specified, bits 0 (FX) and 3 (OX) are set to the values of IMM[0] and IMM[3] (i.e., even if this instruction causes OX to change from zero to one, FX is set from IMM[0] and not by the usual rule that FX is set to one when an exception bit changes from 0 to 1). Bits 1 and 2 (FEX and VX) are set according to the usual rule, given in **2.2.3 Floating-Point Status and Control Register (FPSCR)** and not from IMM[1:2].

Other registers altered:

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

This instruction is defined by the PowerPC UISA.

mtmsr **r**S



MSR←**r**S

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

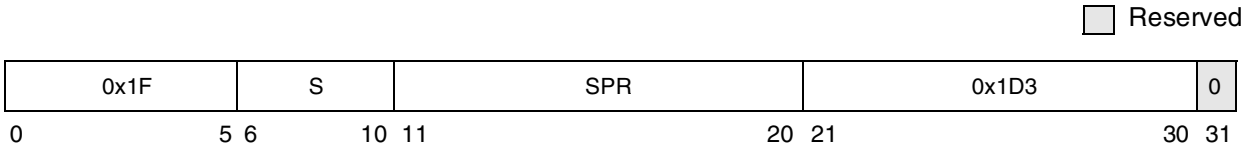
This is a supervisor-level, executing-synchronizing instruction.

Other registers altered:

- MSR

This instruction is defined by the PowerPC OEA.

mtspr SPR,**rS**



n =SPR[5:9] ||SPR[0:4]
SPREG(*n*)←(**r**S)

The SPR field denotes a special purpose register, encoded as shown in **Table 4-29**, **Table 4-30**, and **Table 4-31**. The contents of **r**S are placed into the designated special purpose 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 5 bits appearing in bits 16 to 20 of the instruction and the low-order 5 bits in bits 11 to 15.

If the SPR field contains any value other than one of the values shown in one of the tables listed above, one of the following occurs:

- The system illegal instruction error handler is invoked.
- The system supervisor-level instruction error handler is invoked.
- The software emulation exception handler is invoked.

Note that, for this instruction, SPRs TBL and TBU are treated as separate registers; setting one leaves the other unaltered.

The value of SPR[0] is one if and only if the register is read at the supervisor-level. Execution of this instruction specifying a supervisor-level register when MSR[PR]=1 results in a supervisor-level instruction type program exception or software emulation exception.

If the SPR field contains a value that is not valid for the RCPU, the instruction form is invalid. For an invalid instruction form in which SPR[0]=1, if MSR[PR]=1 a supervisor-level instruction type program exception will occur instead of a no-op.

The execution unit that executes the **mtspr** instruction depends on the SPR. Moves to the XER and to SPRs that are physically implemented outside the processor are handled by the LSU. Moves to the FPSCR and FPECR are executed by the FPU. In all other cases, the BPU executes the **mtspr** instruction.

Other registers altered:

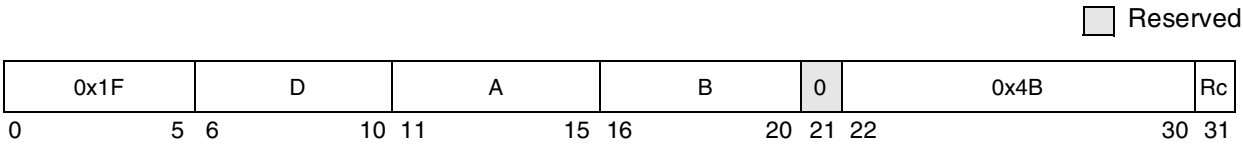
- None

This instruction is defined by the PowerPC UISA.

Table 9-23 Simplified Mnemonics for mtspr Instruction

Operation	Simplified Mnemonic	Equivalent To
Move to XER	**mtxer** r**S**	**mtspr 1**,r**S**
Move to LR	**mtlr** r**S**	**mtspr 8**,r**S**
Move to CTR	**mtctr** r**S**	**mtspr 9**,r**S**
Move to DSISR	**mtdsisr** r**S**	**mtspr 18**,r**S**
Move to DAR	**mtdar** r**S**	**mtspr 19**,r**S**
Move to DEC	**mtdec** r**S**	**mtspr 22**,r**S**
Move to SRR0	**mtsrr0** r**S**	**mtspr 26**,r**S**
Move to SRR1	**mtsrr1** r**S**	**mtspr 27**,r**S**
Move to SPRG	**mtsprg** *n,* r**S**	**mtspr 272+***n*,r**S**
Move to TBL	**mttbl** r**S**	**mtspr 284,**r**S**
Move to TBU	**mttbu** r**S**	**mtspr 285,**r**S**

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



prod[0:63]←(**r**A)*(**r**B)
rD←prod[0:31]

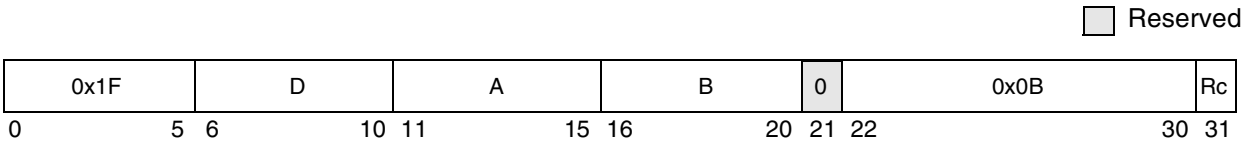
The contents of **r**A and of **r**B are interpreted as 32-bit signed integers. They are multiplied to form a 64-bit signed integer product. The high-order 32 bits of the 64-bit product are placed into **r**D.

Other registers altered:

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

This instruction is defined by the PowerPC UISA.

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



prod[0:63]←(**r**A)*(**r**B)
rD←prod[0:31]

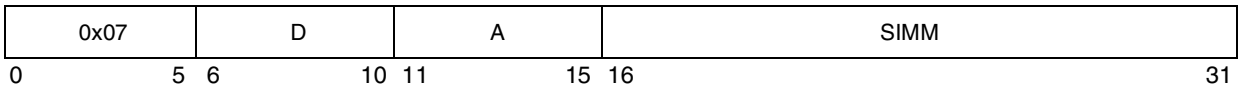
The contents of **r**A and of **r**B are interpreted as 32-bit unsigned integers. They are multiplied to form a 64-bit unsigned integer product. The high-order 32 bits of the 64-bit product are placed into **r**D.

Other registers altered:

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

This instruction is defined by the PowerPC UISA.

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



prod[0:47]←**r**A∗SIMM
rD←prod[16:47]

The low-order 32 bits of the 48-bit product (**r**A)∗SIMM are placed into **r**D. The low-order bits are calculated independently of whether the operands are treated as signed or unsigned 32-bit integers.

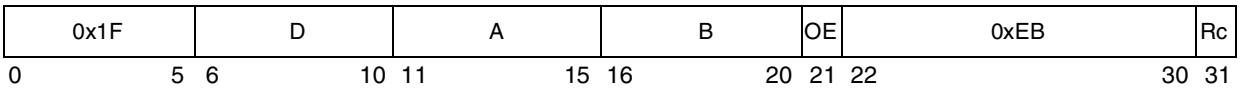
This instruction can be used with **mullhw***x* to calculate a full 64-bit product.

Other registers altered:

- None

This instruction is defined by the PowerPC UISA.

mullw	**r**D,**r**A,**r**B	(OE=0 Rc=0)
mullw.	**r**D,**r**A,**r**B	(OE=0 Rc=1)
mullwo	**r**D, **r**A,**r**B	(OE=1 Rc=0)
mullwo.	**r**D,**r**A,**r**B	(OE=1 Rc=1)



prod[0:63]←(**r**A)*(**r**B)
rD←prod[32:63]

The low-order 32 bits of the 64-bit product (**r**A)*(**r**B) are placed into **r**D. The low-order bits are calculated independently of whether the operands are treated as signed or unsigned integers. However, OV is set based on the result interpreted as a signed integer.

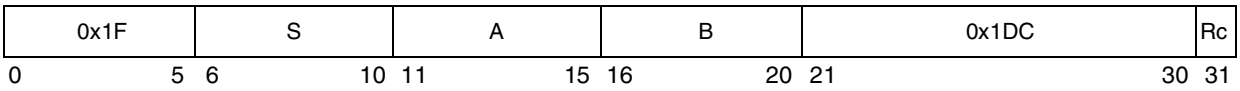
If OE=1, then OV is set to one if the product cannot be represented in 32 bits.

Other registers altered:

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

This instruction is defined by the PowerPC UISA.

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.

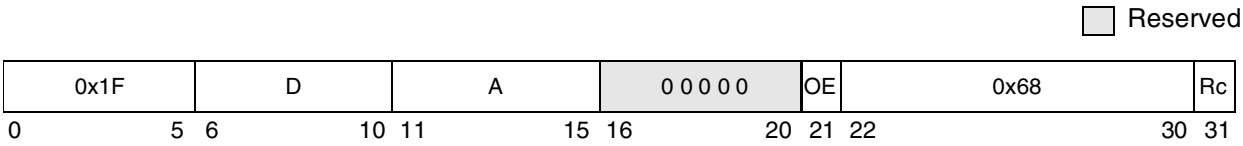
nand with **r**S = **r**B can be used to obtain the one's complement.

Other registers altered:

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

This instruction is defined by the PowerPC UISA.

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

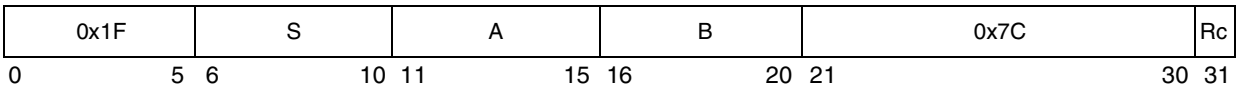
If **r**A contains the most negative 32-bit number (0x8000 0000), the result is the most negative 32-bit number, and if OE=1, OV is set.

Other registers altered:

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

This instruction is defined by the PowerPC UISA.

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 one's complement of the result is placed into **r**A.

nor with **r**S=**r**B can be used to obtain the one's complement.

Other registers altered:

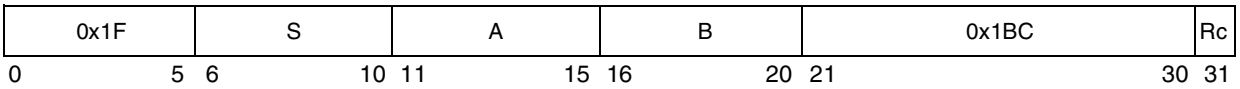
- Condition Register (CR0 Field):
  Affected: LT, GT, EQ, SO (if Rc=1)

This instruction is defined by the PowerPC UISA.

Table 9-24 Simplified Mnemonics for nor Instruction

Operation	Simplified Mnemonic	Equivalent To
Complement register	**not r**A, **r**S **not.** **r**A, **r**S	**nor r**A,**r**S,**r**S **nor.** **r**A,**r**S,**r**S

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 is ORed with the contents of **r**B and the result is placed into **r**A.

Other registers altered:

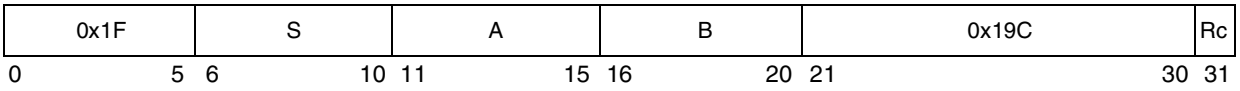
- Condition Register (CR0 Field):
  Affected: LT, GT, EQ, SO (if Rc=1)

This instruction is defined by the PowerPC UISA.

Table 9-25 Simplified Mnemonics for or Instruction

Operation	Simplified Mnemonic	Equivalent To
Move register	**mr** **r**A, **r**S **mr.** **r**A, **r**S	**or** **r**A,**r**S,**r**S **or.** **r**A,**r**S,**r**S

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 is ORed with the complement of the contents of **r**B and the result is placed into **r**A.

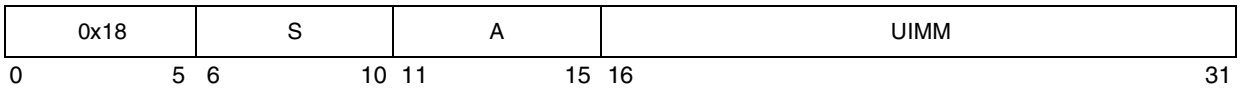
Other registers altered:

- Condition Register (CR0 Field):

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

This instruction is defined by the PowerPC UISA.

ori **r**A,**r**S,UIMM



rA←(**r**S) | ((16)0 || UIMM)

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

The preferred no-op is:

ori 0,0,0

Other registers altered:

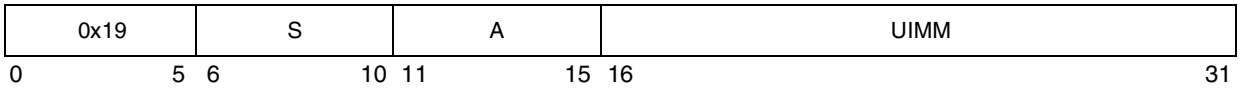
- None

This instruction is defined by the PowerPC UISA.

Table 9-26 Simplified Mnemonics for ori Instruction

Operation	Simplified Mnemonic	Equivalent To
No operation	**nop**	**ori 0,0,0**

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



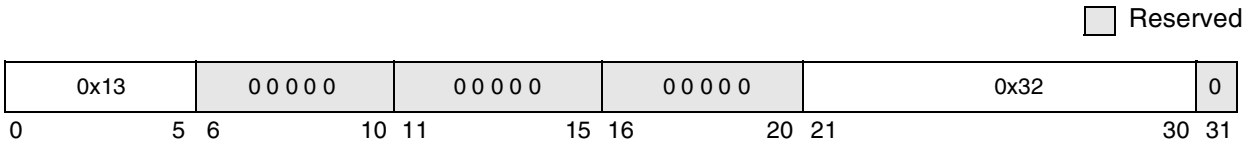
rA←(**r**S) | (UIMM || (16)0)

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

Other registers altered:

- None

This instruction is defined by the PowerPC UISA.



MSR[16:31]←SRR1[16:31]
NIA←SRR0[0:29] || 0b00

SRR1[16:31] are placed into MSR[16:31]. 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.

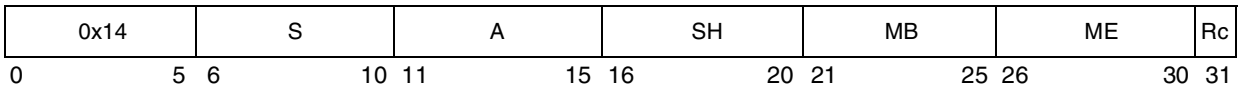
This is a supervisor-level, context-synchronizing instruction.

Other registers altered:

- MSR

This instruction is defined by the PowerPC OEA.

rlwimi **r**A,**r**S,SH,MB,ME (Rc=0)
rlwimi. **r**A,**r**S,SH,MB,ME (Rc=1)



n←SH
r←ROTL(**r**S, *n*)
m←MASK(MB, ME)
rA←(r&M) | (**r**A &¬m)

The contents of **r**S are rotated left SH bits. 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 in **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):

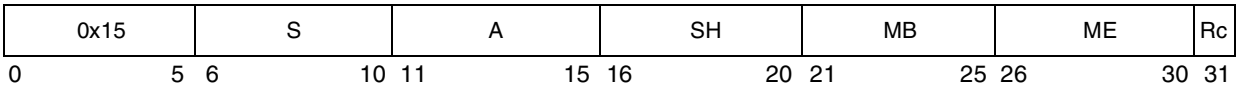
  Affected: LT, GT, EQ, SO (if Rc=1)

This instruction is defined by the PowerPC UISA.

Table 9-27 Simplified Mnemonics for rlwimi Instruction

Operation	Simplified Mnemonic	Equivalent To
Insert from left immediate	**inslwi** **r**A,**r**S,*n*,*b* **inslwi.** **r**A,**r**S,*n*,*b*	**rlwimi** **r**A,**r**S,**32-***b*,*b*,*b*+*n*-**1** **rlwimi.** **r**A,**r**S,**32-***b*,*b*,*b*+*n*-**1**
Insert from right immediate	**insrwi** **r**A,**r**S,*n*,*b* **insrwi.** **r**A,**r**S,*n*,*b*	**rlwimi** **r**A,**r**S,**32-** (*b* + *n*),*b*,*b*+*n*-**1** **rlwimi.** **r**A,**r**S,**32-** (*b* + *n*),*b*,*b*+*n*-**1**

rlwinm **r**A,**r**S,SH,MB,ME (Rc=0)
rlwinm. **r**A,**r**S,SH,MB,ME (Rc=1)



n←SH
r←ROTL(**r**S, *n*)
m←MASK(MB, ME)
rA←r & m

The contents of **r**S are rotated left SH bits. 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, or clear bit fields using the following methods:

- To extract an *n*-bit field that starts at bit position *b* in **r**S[0:31], 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[0–31], 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, set SH=32-N, MB=*n*, and ME=31.
- To clear the high-order *b* bits of a register and then shift the result left by *n* bits, set SH=*n*, MB=*b*-*n* and ME=31-*n*.
- To clear the low-order *n* bits of a register, set SH=0, MB=0, and ME=31-*n*.

Other registers altered:

- Condition Register (CR0 Field):

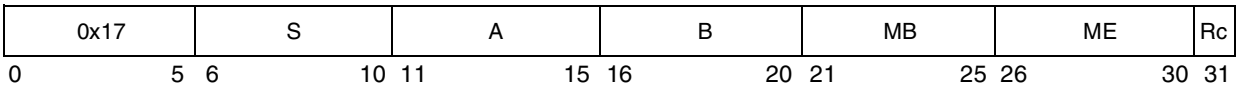
  Affected: LT, GT, EQ, SO (if Rc=1)

This instruction is defined by the PowerPC UISA.

Table 9-28 Simplified Mnemonics for rlwinm Instruction

Operation	Simplified Mnemonic	Equivalent To
Extract and left justify immediate	**extlwi** r**A**,r**S**,*n*,b (n > 0) **extlwi.** r**A**,r**S**,*n*,b (n > 0)	**rlwinm** r**A**,r**S**,b,**0**,*n*-1 **rlwinm.** r**A**,r**S**,b,**0**,*n*-1
Extract and right justify immediate	**extrwi** r**A**,r**S**,*n*,b (n > 0) **extrwi** r**A**,r**S**,*n*,b (n > 0)	**rlwinm** r**A**,r**S**,b + *n*, 32 – *n*, **31** **rlwinm.** r**A**,r**S**,b + *n*, 32 – *n*, **31**
Rotate left immediate	**rotlwi** r**A**,r**S**,*n* **rotlwi.** r**A**,r**S**,*n*	**rlwinm** r**A**,r**S**,*n*,**0,31** **rlwinm.** r**A**,r**S**,*n*,**0,31**
Rotate right immediate	**rotrwi** r**A**,r**S**,*n* **rotrwi.** r**A**,r**S**,*n*	**rlwinm** r**A**,r**S**,**32 –** *n*,**0,31** **rlwinm.** r**A**,r**S**,**32 –** *n*,**0,31**
Shift left immediate	**srwi** r**A**,r**S**,*n* (*n* < 32) **srwi.** r**A**,r**S**,*n* (*n* < 32)	**rlwinm** r**A**,r**S**,*n*,**0**,31–*n* **rlwinm.** r**A**,r**S**,*n*,**0**,31–*n*
Shift right immediate	**srwi** r**A**,r**S**,*n* (*n* < 32) **srwi.** r**A**,r**S**,*n* (*n* < 32)	**rlwinm** r**A**,r**S**,32-*n*,*n*,**31** **rlwinm.** r**A**,r**S**,32-*n*,*n*,**31**
Clear left immediate	**clrlwi** r**A**,r**S**,*n* (*n*<32) **clrlwi.** r**A**,r**S**,*n* (*n*<32)	**rlwinm** r**A**,r**S**,**0**,*n*,31 **rlwinm.** r**A**,r**S**,**0**,*n*,31
Clear right immediate	**clrrwi** r**A**,r**S**,*n* (*n*<32) **clrrwi.** r**A**,r**S**,*n* (*n*<32)	**rlwinm** r**A**,r**S**,**0,0**,31-*n* **rlwinm.** r**A**,r**S**,**0,0**,31-*n*
Clear left and shift left immediate	**clrlslwi** r**A**,r**S**,b,*n* (*n* ð b ð 31) **clrlslwi.** r**A**,r**S**,b,*n* (*n* ð b ð 31)	**rlwinm** r**A**,r**S**,*n*,b-*n*,31-*n* **rlwinm.** r**A**,r**S**,*n*,b-*n*,31-*n*

rlwnm **r**A,**r**S,**r**B,MB,ME (Rc=0)
rlwnm. **r**A,**r**S,**r**B,MB,ME (Rc=1)



n←**r**B[27:31]
r←ROTL(**r**S, *n*)
m←MASK(MB, ME)
rA←r & m

The contents of **r**S are rotated left the number of bits specified by **r**B[27:31]. A mask is generated having 1-bit 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 following methods:

- To extract an *n*-bit field that starts at variable bit position *b* in **r**S[0:31], right-justified into **r**A (clearing the remaining 32-*n* bits of **r**A), set **r**B[27:31]=*b*+*n*, MB=32-*n*, and ME=31.
- To extract an *n*-bit field, that starts at variable bit position *b* in **r**S[0:31], left-justified into **r**A (clearing the remaining 32-*n* bits of **r**A), set **r**B[27:31]=*b*, MB=0, and ME=*n*-1.
- To rotate the contents of a register left (or right) by variable *n* bits, set **r**B[27:31]=*n* (32-N), MB=0, and ME=31.

Other registers altered:

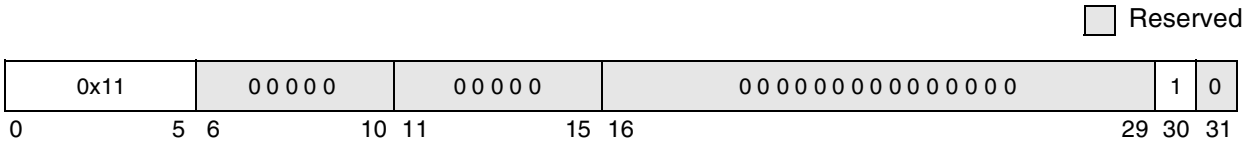
- Condition Register (CR0 Field):

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

This instruction is defined by the PowerPC UISA.

Table 9-29 Simplified Mnemonics for rlwnm Instruction

Operation	Simplified Mnemonic	Equivalent To
Rotate left	**rotlw** **r**A,**r**S,**r**B **rotlw.** **r**A,**r**S,**r**B	**rlwnm** **r**A,**r**S,**r**B,**0,31** **rlwnm.** **r**A,**r**S,**r**B,**0,31**



This 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 system call instruction is placed into SRR0. MSR[16:31] are placed into SRR1[16:31], and SRR1[0:15] are set to undefined values.

Then a system call exception is generated. The exception causes the MSR to be altered as described in **6.11.8 System Call Exception (0x00C00)**.

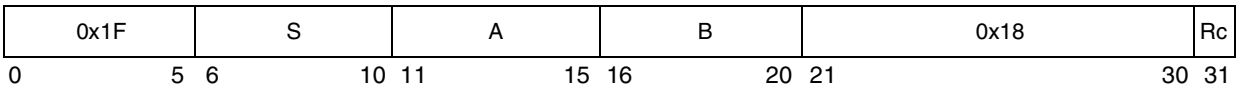
The exception causes the next instruction to be fetched from offset 0xC00 from the physical base address indicated by the new setting of MSR[IP]. This instruction is context-synchronizing.

Other registers altered:

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

This instruction is defined by the PowerPC UISA.

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



n←**r**B[27:31]
r←ROTL((**r**S), *n*)
if **r**B[26]=0 then
    m← MASK(0,31-n)
else
    m←(32)0
rA←r&m

If **r**B[26]=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 **r**B[26]=1, 32 zeros are placed into **r**A.

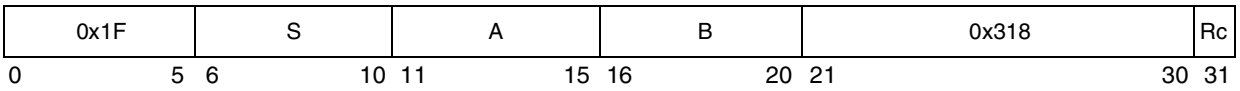
Other registers altered:

- Condition Register (CR0 Field):

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

This instruction is defined by the PowerPC UISA.

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




```
n←rB[27:31]
r←ROTL((rS), 32-n)
if rB[26]=0 then
      m← MASK(n,31)
else
      m←(32)0
s←rS[0]
rA←r&m | (32)s & ¬ m
XER[CA]←s & ((r & ¬ m)¦0)
```

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.

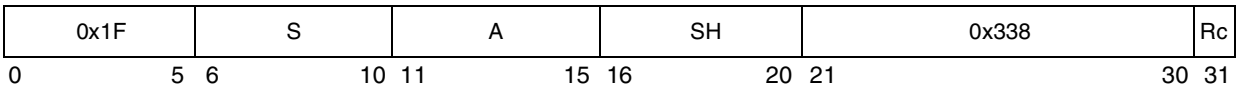
XER[CA] is set to one if **r**S contains a negative number and any 1-bits are shifted out of position 31; otherwise XER[CA] is cleared to zero. A shift amount of zero causes XER[CA] to be cleared.

Other registers altered:

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

This instruction is defined by the PowerPC UISA.

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



n←SH
r←ROTL((**r**S), 32-*n*)
m← MASK(n,31)
s←**r**S[0]
rA←r&m | (32)s & ¬ m
XER[CA]←s & ((r & ¬ m)¦0)

The contents of **r**S are shifted right SH bits. Bits shifted out of position 31 are lost. 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 to one if **r**S contains a negative number and any 1-bits are shifted out of position 31; otherwise XER[CA] is cleared to zero. A shift amount of zero causes XER[CA] to be cleared to zero.

Other registers altered:

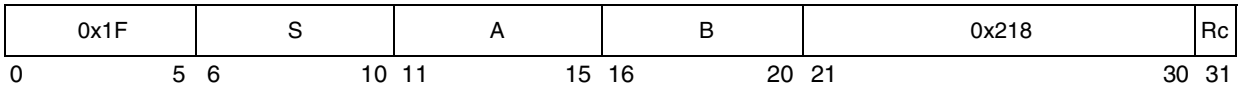
- Condition Register (CR0 Field):

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

  Affected: CA

This instruction is defined by the PowerPC UISA.

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



n←**r**B[27:31]
r←ROTL((**r**S), 32-*n*)
if rB[26]=0 then
    m←MASK(n,31)
else
    m←(32)0
rA←r & m

If **r**B[26]=0, the contents of **r**A are shifted right the number of bits specified by **r**A[27:31]. Bits shifted out of position 31 are lost. Zeros are supplied to the vacated positions on the left. The 32-bit result is placed into **r**A.

If **r**B[26]=1, then **r**A is filled with zeros.

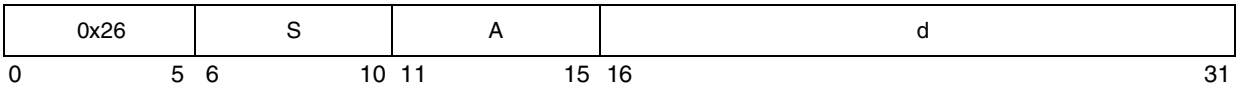
Other registers altered:

- Condition Register (CR0 Field):

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

This instruction is defined by the PowerPC UISA.

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



if **r**A = 0 then b←0
else b←(**r**A)
EA←b + EXTS(d)
MEM(EA, 1)←**r**S[24:31]

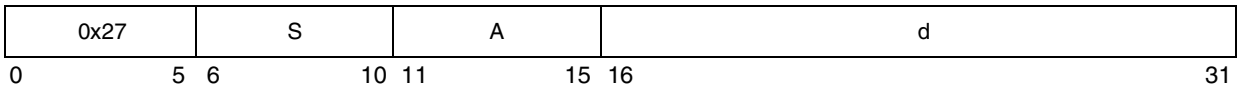
EA is the sum (**r**A|0)+d. The contents of **r**S[24:31] are stored into the byte in memory addressed by EA. Register **r**S is unchanged.

Other registers altered:

- None

This instruction is defined by the PowerPC UISA.

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



EA← (**r**A) + EXTS(d)
MEM(EA, 1)←**r**S[24:31]
rA←EA

EA is the sum (**r**A|0)+d. The contents of **r**S[24:31] are stored into the byte in memory addressed by EA.

EA is placed into **r**A.

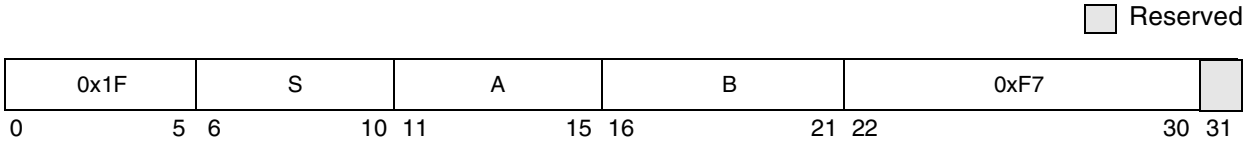
If **r**A=0, the instruction form is invalid.

Other registers altered:

- None

This instruction is defined by the PowerPC UISA.

stbux **r**S,**r**A,**r**B



EA← (**r**A) + (**r**B)
MEM(EA, 1)←**r**S[24:31]
rA←EA

EA is the sum (**r**A|0)+(**r**B). The contents of **r**S[24:31] is stored into the byte in memory addressed by EA.

EA is placed into **r**A.

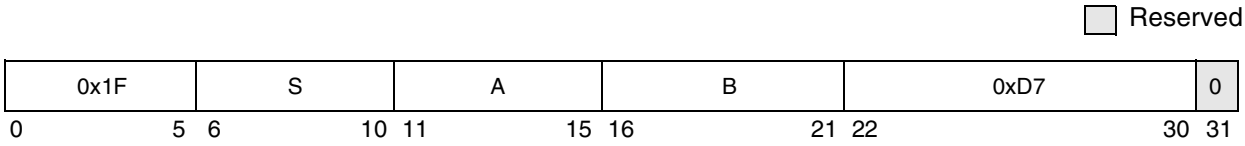
If **r**A=0, the instruction form is invalid.

Other registers altered:

- None

This instruction is defined by the PowerPC UISA.

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



if **r**A = 0 then b←0
else b←(**r**A)
EA←b + (**r**B)
EM(EA, 1) ← **r**S[24:31]

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

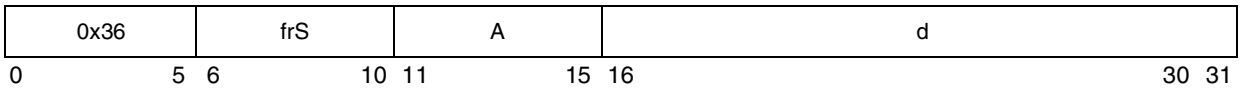
The contents of **r**S[24:31] is stored into the byte in memory addressed by EA. Register **r**S is unchanged.

Other registers altered:

- None

This instruction is defined by the PowerPC UISA.

stfd **fr**S,d**(r**A**)**



if **r**A = 0 then b←0
else b←(**r**A)
EA←b + EXTS(d)
MEM(EA, 8)←(**fr**S)

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

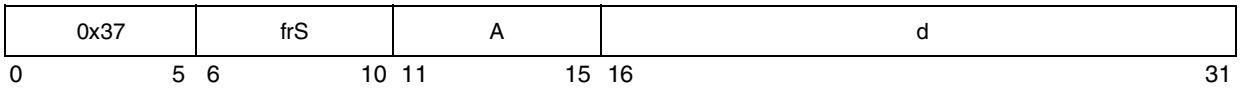
The contents of **fr**S are stored into the double word in memory addressed by EA.

Other registers altered:

- None

This instruction is defined by the PowerPC UISA.

stfdu **fr**S,d**(r**A**)**



EA← (**r**A) + EXTS(d)
MEM(EA, 8)← (**fr**S)
rA←EA

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

The contents of **fr**S are stored into the double word in memory addressed by EA.

EA is placed into **r**A.

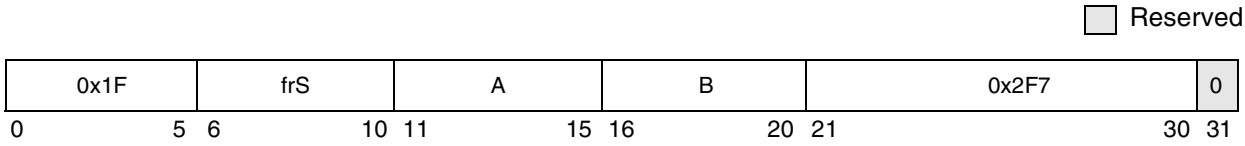
If **r**A=0, the instruction form is invalid.

Other registers altered:

- None

This instruction is defined by the PowerPC UISA.

stfdux **fr**S,**r**A,**r**B



EA← (**r**A) + (**r**B)
MEM(EA, 8)←(**fr**S)
rA←EA

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

The contents of **fr**S are stored into the double word in memory addressed by EA.

EA is placed into **r**A.

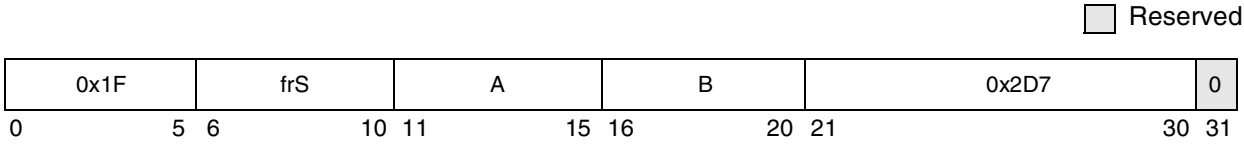
If **r**A=0, the instruction form is invalid.

Other registers altered:

- None

This instruction is defined by the PowerPC UISA.

stfdx **fr**S,**r**A,**r**B



if **r**A + 0 then b ←0
else b←(**r**A)
EA←b + (**r**B)
MEM(EA, 8)←(**fr**S)

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

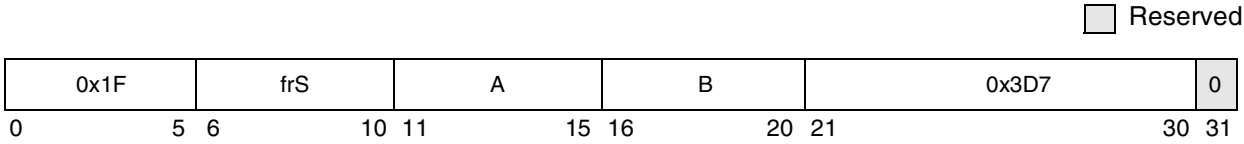
The contents of **fr**S are stored into the double word in memory addressed by EA.

Other registers altered:

- None

This instruction is defined by the PowerPC UISA.

stfiwx **fr**S,**r**A,**r**B



if **r**A =0 then b ←0
else b←(**r**A)
EA←b + (**r**B)
MEM(EA, 4)←**fr**S[32:63]

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

The low-order 32 bits of **fr**S are stored, without conversion, into the word in memory addressed by EA.

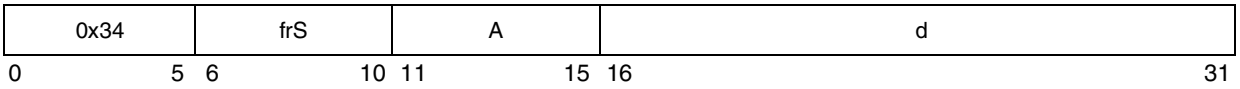
If the contents of **fr**S were produced, either directly or indirectly, by an **lfs** instruction, a single-precision arithmetic instruction, or **frsp**, then the value stored is undefined. The contents of **fr**S are produced directly by such an instruction if **fr**S is the target register for the instruction. The contents of **fr**S are produced indirectly by such an instruction if **fr**S is the final target register of a sequence of one or more floating-point move instructions, with the input to the sequence having been produced directly by such an instruction.

Other registers altered:

- None

This instruction is defined by the PowerPC UISA.

stfs **fr**S**,d(r**A**)**



if **r**A = 0 then b←0
else b←(**r**A)
EA←b + EXTS(d)
MEM(EA, 4)←SINGLE(**fr**S)

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

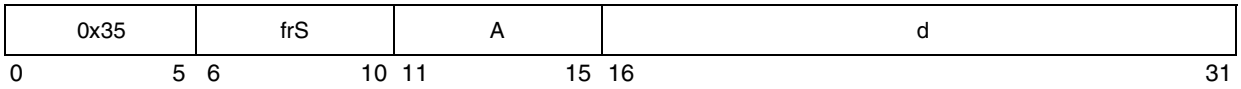
The contents of **fr**S are converted to single-precision and stored into the word in memory addressed by EA.

Other registers altered:

- None

This instruction is defined by the PowerPC UISA.

stfsu **fr**S,d**(r**A**)**



EA← (**r**A) + EXTS(d)
MEM(EA, 4)←SINGLE(**fr**S)
rA←EA

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

The of **fr**S are converted to single-precision and stored into the word in memory addressed by EA.

EA is placed into **r**A.

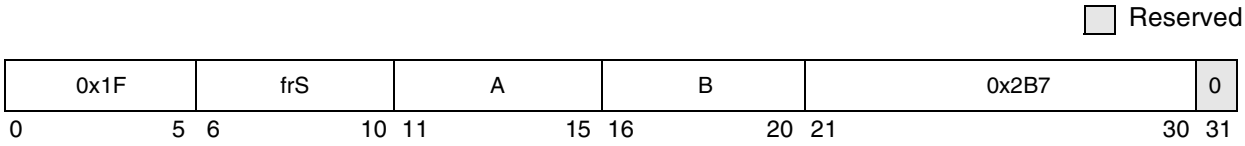
If **r**A=0, the instruction form is invalid.

Other registers altered:

- None

This instruction is defined by the PowerPC UISA.

stfsux **fr**S,**r**A,**r**B



EA← (**r**A) + (**r**B)
MEM(EA, 4)←SINGLE(**fr**S)
rA←EA

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

The contents of **fr**S are converted to single-precision and stored into the word in memory addressed by EA.

EA is placed into **r**A.

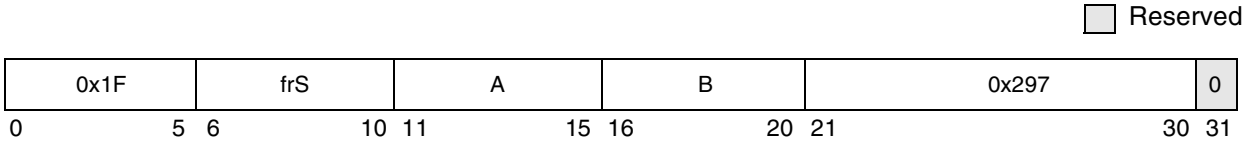
If **r**A=0, the instruction form is invalid.

Other registers altered:

- None

This instruction is defined by the PowerPC UISA.

stfsx **fr**S,**r**A,**r**B



if **r**A=0 then b←0
else b←(**r**A)
EA←b + (**r**B)
MEM(EA, 4)←SINGLE(**fr**S)

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

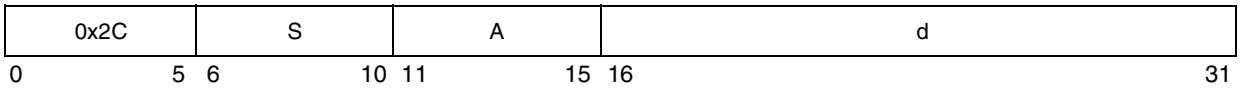
The contents of **fr**S are converted to single-precision and stored into the word in memory addressed by EA.

Other registers altered:

- None

This instruction is defined by the PowerPC UISA.

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



if **r**A = 0 then b←0
else b←(**r**A)
EA←b + EXTS(d)
MEM(EA, 2)←**r**S[16:31]

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

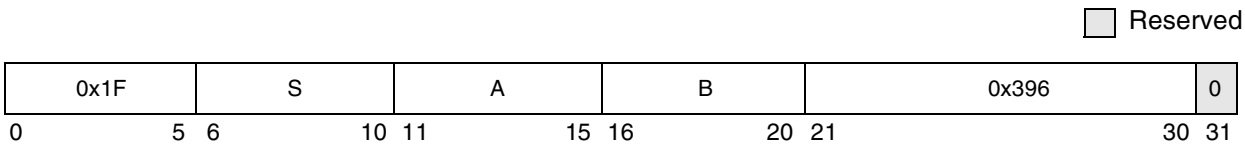
The contents of **r**S[16:31] are stored into the half word in memory addressed by EA.

Other registers altered:

- None

This instruction is defined by the PowerPC UISA.

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



if **r**A = 0 then b←0
else b←(**r**A)
EA←b + (**r**B)
MEM(EA, 2)←**r**S[24:31] || **r**S[16:23]

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

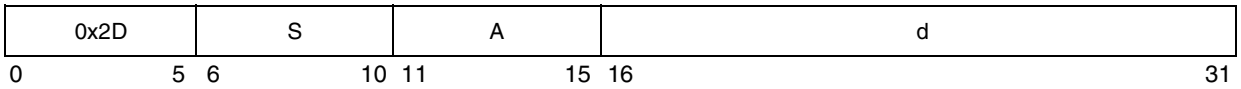
The contents of **r**S[24:31] are stored into bits 0:7 of the half word in memory addressed by EA. Bits **r**S[16:23] are stored into bits 8:15 of the half word in memory addressed by EA.

Other registers altered:

- None

This instruction is defined by the PowerPC UISA.

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



EA← (**r**A) + EXTS(d)
MEM(EA, 2)←**r**S[16:31]
rA←EA

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

The contents of **r**S[16:31] are stored into the half word in memory addressed by EA.

EA is placed into **r**A.

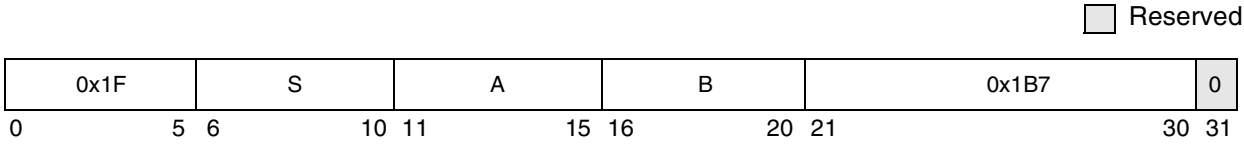
If **r**A=0, the instruction form is invalid.

Other registers altered:

- None

This instruction is defined by the PowerPC UISA.

sthux **r**S,**r**A,**r**B



EA← (**r**A) + (**r**B)
MEM(EA, 2)←**r**S[16:31]
rA←EA

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

The contents of **r**S[16:31] are stored into the half word in memory addressed by EA.

EA is placed into **r**A.

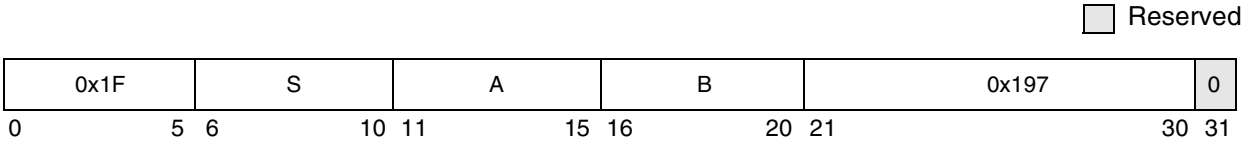
If **r**A=0, the instruction form is invalid.

Other registers altered:

- None

This instruction is defined by the PowerPC UISA.

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



if **r**A = 0 then b←0
else b←(**r**A)
EA←b + (**r**B)
MEM(EA, 2)←**r**S[16:31]

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

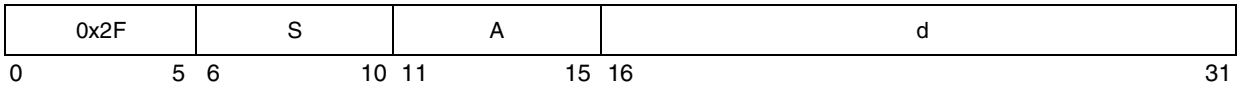
The contents of **r**S[16:31] are stored into the half word in memory addressed by EA.

Other registers altered:

- None

This instruction is defined by the PowerPC UISA.

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




```
if rA = 0 then b←0
else b←(rA)
EA←b + EXTS(d)
r←rS
do while r ð 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 31. For example, if **r**S=30, two words are stored.

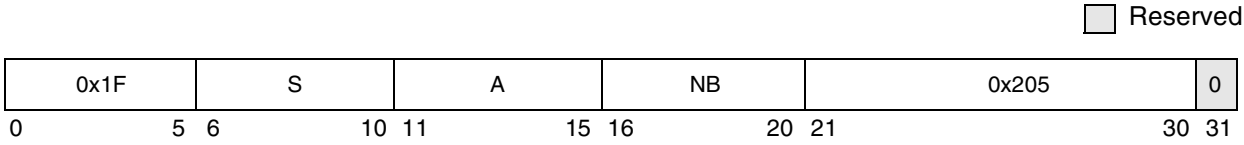
EA must be a multiple of four; otherwise, the system alignment error handler is invoked.

Other registers altered:

- None

This instruction is defined by the PowerPC UISA.

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




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

EA is (**r**A|0). Let *n* = NB if NB¦0, *n* = 32 if NB=0; *n* is the number of bytes to store. Let *nr* = CEIL(*n*/4): *nr* is the number of registers to supply data.

n consecutive bytes starting at EA are stored from GPRs **r**S through **r**S+*nr*-1.

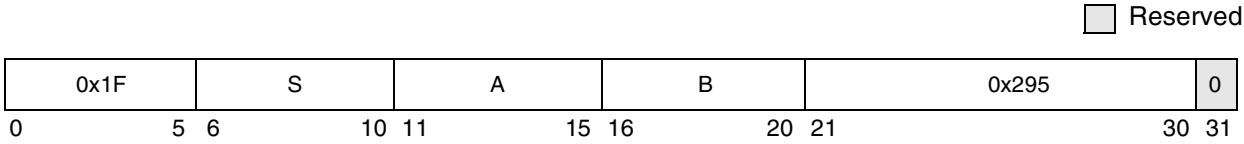
Bytes are stored left to right from each register. The sequence of registers wraps around to GPR0 if required.

Other registers altered:

- None

This instruction is defined by the PowerPC UISA.

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




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

EA is the sum (**r**A|0)+(**r**B). Let *n* = XER[25:31]; *n* is the number of bytes to store.

Let *nr* = CEIL(*n*/4): *nr* is the number of registers to supply data.

n consecutive bytes starting at EA are stored from GPRs **r**S through **r**S+*nr*-1. If *n* = 0, no bytes are stored.

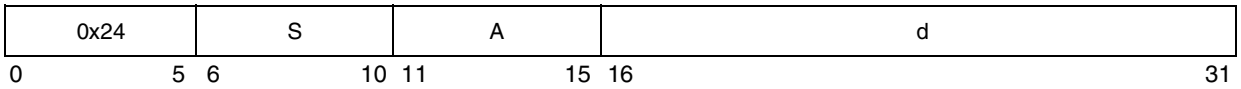
Bytes are stored left to right from each register. The sequence of registers wraps around to GPR0 if required.

Other registers altered:

- None

This instruction is defined by the PowerPC UISA.

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



if **r**A = 0 then b←0
else b←(**r**A)
EA←b + EXTS(d)
MEM(EA, 4)←**r**S

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

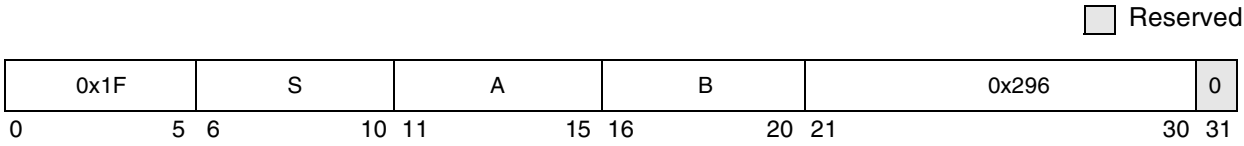
The contents of **r**S are stored into the word in memory addressed by EA.

Other registers altered:

- None

This instruction is defined by the PowerPC UISA.

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



if **r**A = 0 then b←0
else b←(**r**A)
EA←b + (**r**B)
MEM(EA, 4)←**r**S[24:31] || **r**S[16:23] || **r**S[8:15] || **r**S[0:7]

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

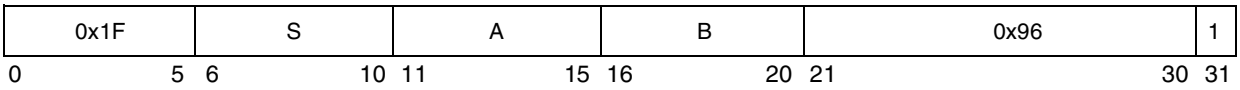
The contents of **r**S[24:31] are stored into bits 0:7 of the word in memory addressed by EA. Bits **r**S[16:23] are stored into bits 8:15 of the word in memory addressed by EA. Bits **r**S[8:15] are stored into bits 16:23 of the word in memory addressed by EA. Bits **r**S[0:7] are stored into bits 24:31 of the word in memory addressed by EA.

Other registers altered:

- None

This instruction is defined by the PowerPC UISA.

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




```
if rA = 0 then b←0
else b←(rA)
EA←b + (rB)
if RESERVE then
      MEM(EA, 4)←rS
      RESERVE←0
      CR0←0b00 || 0b1|| XER[SO]
else
      CR0←0b00 || 0b0 || XER[SO]
```

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

If a reservation exists, the contents of **r**S are stored into the word in memory addressed by EA and the reservation is cleared. If no reservation exists, the instruction completes without altering memory.

CR0 Field is set to reflect whether the store operation was performed (i.e., whether a reservation existed when the **stwcx.** instruction commenced execution) as follows.

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

The EQ bit in the condition register field CR0 is modified to reflect whether the store operation was performed (i.e., whether a reservation existed when the **stwcx.** instruction began execution). If the store was completed successfully, the EQ bit is set to one.

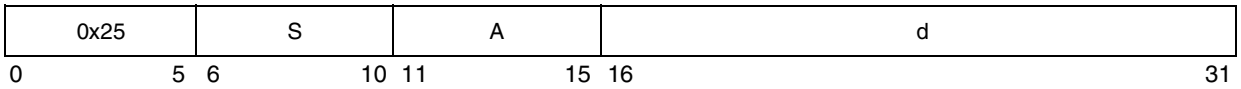
EA must be a multiple of four; otherwise, the system alignment error handler is invoked.

Other registers altered:

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

This instruction is defined by the PowerPC UISA.

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



EA← (**r**A) + EXTS(d)
MEM(EA, 4)←**r**S
rA←EA

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

The contents of **r**S are stored into the word in memory addressed by EA.

EA is placed into **r**A.

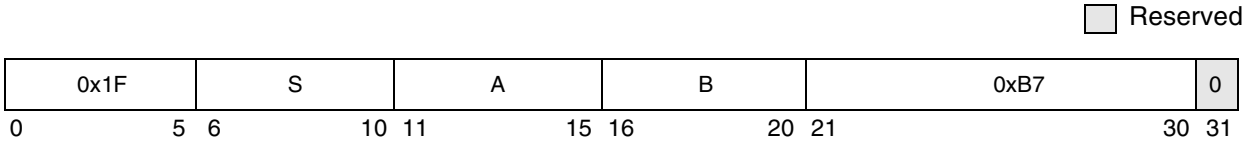
If **r**A=0, the instruction form is invalid.

Other registers altered:

- None

This instruction is defined by the PowerPC UISA.

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|0)+(**r**B).

The contents of **r**S are stored into the word in memory addressed by EA.

EA is placed into **r**A.

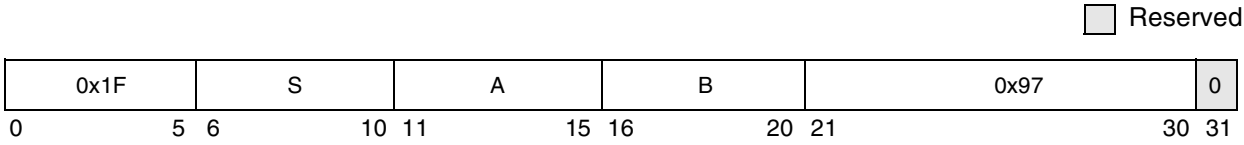
If **r**A=0, the instruction form is invalid.

Other registers altered:

- None

This instruction is defined by the PowerPC UISA.

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



if **r**A = 0 then b←0
else b←(**r**A)
EA←b + (**r**B)
MEM(EA, 4)←**r**S

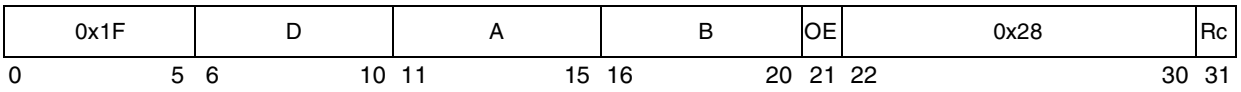
EA is the sum (**r**A|0)+(**r**B). The contents of **r**S are stored into the word in memory addressed by EA.

Other registers altered:

- None

This instruction is defined by the PowerPC UISA.

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← ¬ (**r**A) + (**r**B) + 1

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

Other registers altered:

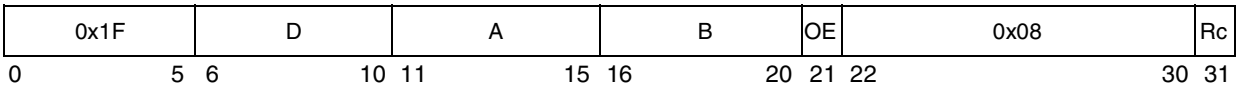
- Condition Register (CR0 Field):
  Affected: LT, GT, EQ, SO (if Rc=1)
- XER:
  Affected: SO, OV (if OE=1)

This instruction is defined by the PowerPC UISA.

Table 9-30 Simplified Mnemonics for subf Instruction

Operation	Simplified Mnemonic	Equivalent To
Subtract	**sub** **r**D,**r**A,**r**B **sub.** **r**D,**r**A,**r**B **subo** **r**D,**r**A,**r**B **subo.** **r**D,**r**A,**r**B	**subf** **r**D,**r**B,**r**A **subf.** **r**D,**r**B,**r**A **subfo** **r**D,**r**B,**r**A **subfo.** **r**D,**r**B,**r**A

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:

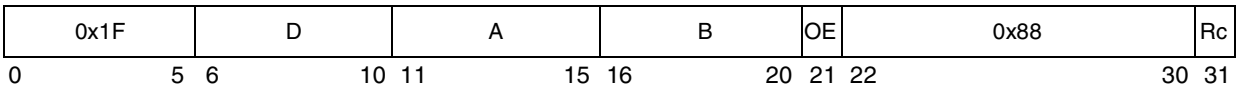
- Condition Register (CR0 Field):
  Affected: LT, GT, EQ, SO (if Rc=1)
- XER:
  Affected: CA
  Affected: SO, OV (if OE=1)

This instruction is defined by the PowerPC UISA.

Table 9-31 Simplified Mnemonics for subfc Instruction

Operation	Simplified Mnemonic	Equivalent To
Subtract	**subc** **r**D,**r**A,**r**B **subc.** **r**D,**r**A,**r**B **subco** **r**D,**r**A,**r**B **subco.** **r**D,**r**A,**r**B	**subfc** **r**D,**r**B,**r**A **subfc.** **r**D,**r**B,**r**A **subfco** **r**D,**r**B,**r**A **subfco.** **r**D,**r**B,**r**A

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]

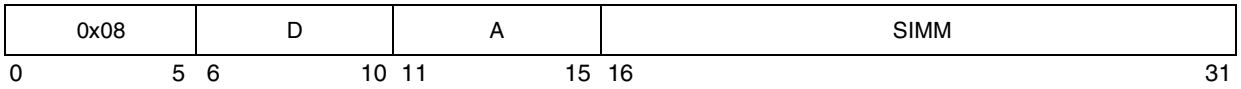
The sum ¬ (**r**A)+(**r**B)+XER[CA] is placed into **r**D.

Other registers altered:

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

This instruction is defined by the PowerPC UISA.

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



rD← ¬ (**r**A) + EXTS(SIMM) + 1

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

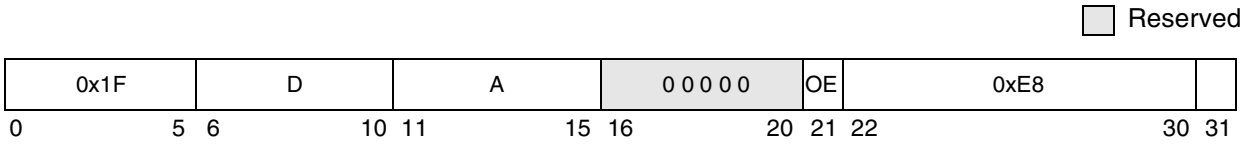
Other registers altered:

- XER:

  Affected: CA

This instruction is defined by the PowerPC UISA.

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← ¬ (**r**A) + XER[CA] - 1

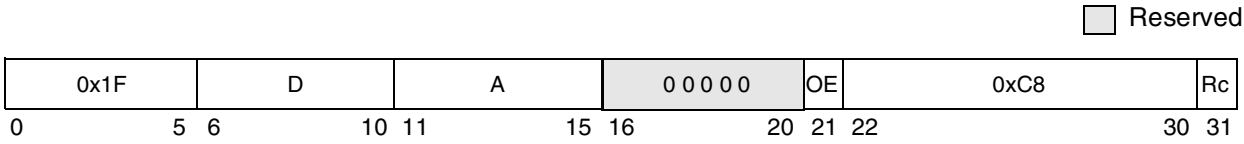
The sum ¬ (**r**A)+XER[CA]+0xFFFF_FFFF is placed into **r**D.

Other registers altered:

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

This instruction is defined by the PowerPC UISA.

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)



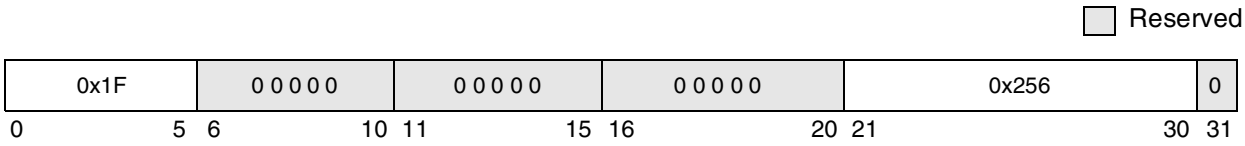
rD← ¬ (**r**A) + XER[CA]

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)
- XER:
  Affected: CA
  Affected: SO, OV (if OE=1)

This instruction is defined by the PowerPC UISA.



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 previously initiated by the given processor appear to have completed before any subsequent instructions are initiated by the given processor. When the **sync** instruction completes, all external accesses initiated by the given processor prior to the **sync** will have been performed with respect to all other mechanisms that access memory.

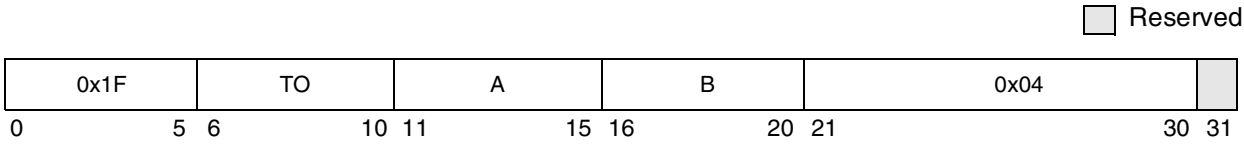
The **sync** instruction can be used to ensure that the results of all stores into a data structure, performed in a “critical section” of a program, are seen by other processors before the data structure is seen as unlocked.

Other registers altered:

- None

This instruction is defined by the PowerPC UISA.

tw TO,**r**A,**r**B



a← (**r**A)
b← (**r**B)
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 to one and its corresponding condition is met by the result of the comparison, then the system trap handler is invoked.

Other registers altered:

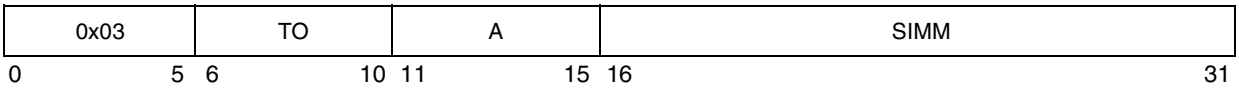
- None

This instruction is defined by the PowerPC UISA.

Table 9-32 Simplified Mnemonics for tw Instruction

Operation	Operands	Equivalent To
Trap unconditionally	**trap**	**tw 31**,**0**,**0**
Trap if equal	**tweq r**A,**r**B	**tw 4**,**r**A,**r**B
Trap if greater than or equal to	**twge r**A,**r**B	**tw 12**,**r**A,**r**B
Trap if greater than	**twgt r**A,**r**B	**tw 8**,**r**A,**r**B
Trap if less than or equal to	**twle r**A,**r**B	**tw 20**,**r**A,**r**B
Trap if logically greater than or equal to	**twlge r**A,**r**B	**tw 5**,**r**A,**r**B
Trap if logically greater than	**twlgt r**A,**r**B	**tw 1**,**r**A,**r**B
Trap if logically less than or equal to	**twlle r**A,**r**B	**tw 6**,**r**A,**r**B
Trap if logically less than	**twllt r**A,**r**B	**tw 2**,**r**A,**r**B
Trap if logically not greater than	**twlng r**A,**r**B	**tw 6**,**r**A,**r**B
Trap if logically not less than	**twlnl r**A,**r**B	**tw 5**,**r**A,**r**B
Trap if less than	**twlt r**A,**r**B	**tw 16**,**r**A,**r**B
Trap if not equal to	**twne r**A,**r**B	**tw 24**,**r**A,**r**B
Trap if not greater than	**twng r**A,**r**B	**tw 20**,**r**A,**r**B
Trap if not less than	**twnl r**A,**r**B	**tw 12**,**r**A,**r**B

twi TO,**r**A,SIMM



a← (**r**A)
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 SIMM field. If any bit in the TO field is set to one and its corresponding condition is met by the result of the comparison, then the system trap handler is invoked.

Other registers altered:

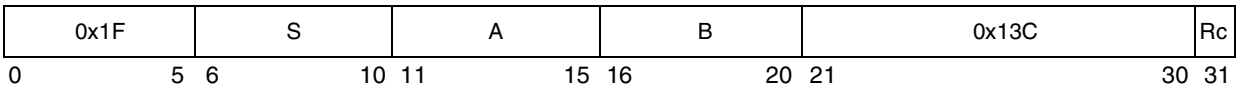
- None

This instruction is defined by the PowerPC UISA.

Table 9-33 Simplified Mnemonics for twi Instruction

Operation	Operands	Equivalent To
Trap if equal	**tweqi r**A,value	**twi 4,r**A,value
Trap if greater than or equal to	**twgei r**A,value	**twi 12,r**A,value
Trap if greater than	**twgti r**A,value	**twi 8,r**A,value
Trap if less than or equal to	**twlei r**A,value	**twi 20,r**A,value
Trap if logically greater than or equal to	**twlgei r**A,value	**twi 5,r**A,value
Trap if logically greater than	**twlgti r**A,value	**twi 1,r**A,value
Trap if logically less than or equal to	**twllei r**A,value	**twi 6,r**A,value
Trap if logically less than	**twllti r**A,value	**twi 2,r**A,value
Trap if logically not greater than	**twlngi r**A,value	**twi 6,r**A,value
Trap if logically not less than	**twlnli r**A,value	**twi 5,r**A,value
Trap if less than	**twlti r**A,value	**twi 16,r**A,value
Trap if not equal to	**twnei r**A,value	**twi 24,r**A,value
Trap if not greater than	**twngi r**A,value	**twi 20,r**A,value
Trap if not less than	**twnli r**A,value	**twi 12,r**A,value

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



rA←(**r**S) ⊕ (**r**B)

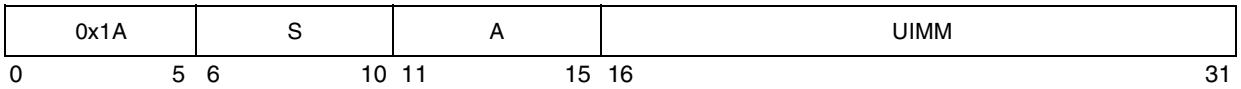
The contents of **r**A is XORed with the contents of **r**B and the result is placed into **r**A.

Other registers altered:

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

This instruction is defined by the PowerPC UISA.

xori **r**A,**r**S,UIMM



rA←(**r**S) ⊕ ((16)0 || UIMM)

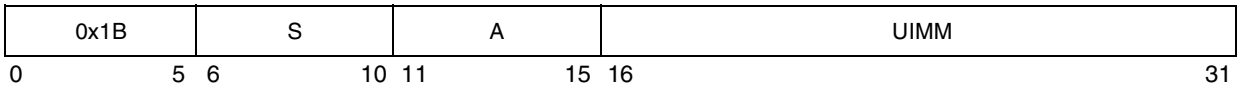
The contents of **r**S is XORed with 0x0000 || UIMM and the result is placed into **r**A.

Other registers altered:

- None

This instruction is defined by the PowerPC UISA.

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



rA←(**r**S) ⊕ (UIMM || (16)0)

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

Other registers altered:

- None

This instruction is defined by the PowerPC UISA.