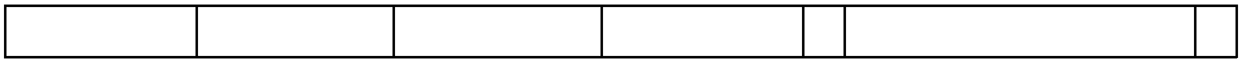
add*x* **add***x*

Add (x'7C00 0214')

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

[POWER mnemonics: **cax**, **cax.**, **caxo**, **caxo.**]



```
rD ← (rA) + (rB)
```

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

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

Other registers altered:

- Condition Register (CR0 field):

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

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

  Affected: SO, OV (if OE = 1)

  Note: The setting of the affected bits in the XER is mode-dependent, and reflects overflow of the 64-bit result in 64-bit mode and overflow of the low-order 32-bit result in 32-bit mode. For further information about 64-bit mode and 32-bit mode in 64-bit implementations, see 4.1.2 , "Computation Modes."

PowerPC Architecture Level	Supervisor Level	32-Bit	64-Bit	64-Bit Bridge	Optional	Form
UISA						XO

addc*x* **addc***x*

Add Carrying (x’7C00 0014’)

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

[POWER mnemonics: **a**, **a.**, **ao**, **ao.**]


```
rD ← (rA) + (rB)
```

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

Other registers altered:

- Condition Register (CR0 field):

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

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

  Affected: CA

  Affected: SO, OV(if OE = 1)

  Note: The setting of the affected bits in the XER is mode-dependent, and reflects overflow of the 64-bit result in 64-bit mode and overflow of the low-order 32-bit result in 32-bit mode. For further information about 64-bit mode and 32-bit mode in 64-bit implementations, see 4.1.2 , “Computation Modes.”

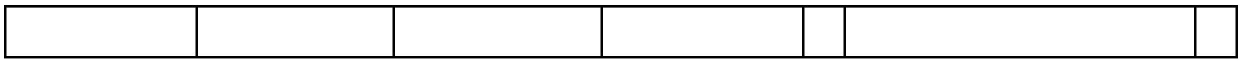
PowerPC Architecture Level	Supervisor Level	32-Bit	64-Bit	64-Bit Bridge	Optional	Form
UISA						XO

adde*x* **adde***x*

Add Extended (x'7C00 0114')

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

[POWER mnemonics: **ae**, **ae.**, **aeo**, **aeo.**]



```
rD ← (rA) + (rB) + XER[CA]
```

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

Other registers altered:

- Condition Register (CR0 field):

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

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

- XER:

  Affected: CA

  Affected: SO, OV (if OE = 1)

  Note: The setting of the affected bits in the XER is mode-dependent, and reflects overflow of the 64-bit result in 64-bit mode and overflow of the low-order 32-bit result in 32-bit mode. For further information about 64-bit mode and 32-bit mode in 64-bit implementations, see 4.1.2 , “Computation Modes.”

PowerPC Architecture Level	Supervisor Level	32-Bit	64-Bit	64-Bit Bridge	Optional	Form
UISA						XO

addi

Add Immediate (x'3800 0000')

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

[POWER mnemonic: **cal**]


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

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

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

Other registers altered:

- None

Simplified mnemonics:

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

PowerPC Architecture Level	Supervisor Level	32-Bit	64-Bit	64-Bit Bridge	Optional	Form
UISA						D

addic

Add Immediate Carrying (x’3000 0000’)

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

[POWER mnemonic: **ai**]


```
rD ← (rA) + EXTS(SIMM)
```

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

Other registers altered:

- XER:

  Affected: CA

  Note: The setting of the affected bits in the XER is mode-dependent, and reflects overflow of the 64-bit result in 64-bit mode and overflow of the low-order 32-bit result in 32-bit mode. For further information about 64-bit mode and 32-bit mode in 64-bit implementations, see 4.1.2 , “Computation Modes.”

Simplified mnemonics:

subic **r**D**,r**A**,**valueequivalent to**addicr**D**,r**A**,**–value

PowerPC Architecture Level	Supervisor Level	32-Bit	64-Bit	64-Bit Bridge	Optional	Form
UISA						D

addic. addic.

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

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

[POWER mnemonic: **ai.**]


```
rD ← (rA) + EXTS(SIMM)
```

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

Other registers altered:

- Condition Register (CR0 field):

  Affected: LT, GT, EQ, SO

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

  Affected: CA

  Note: The setting of the affected bits in the XER is mode-dependent, and reflects overflow of the 64-bit result in 64-bit mode and overflow of the low-order 32-bit result in 32-bit mode. For further information about 64-bit mode and 32-bit mode in 64-bit implementations, see 4.1.2 , “Computation Modes.”

Simplified mnemonics:

subic.rD,**r**A,valueequivalent to**addic.r**D,**r**A,–value

PowerPC Architecture Level	Supervisor Level	32-Bit	64-Bit	64-Bit Bridge	Optional	Form
UISA						D

addis addis

Add Immediate Shifted (x’3C00 0000’)

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

[POWER mnemonic: **cau**]


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

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

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

Other registers altered:

- None

Simplified mnemonics:

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

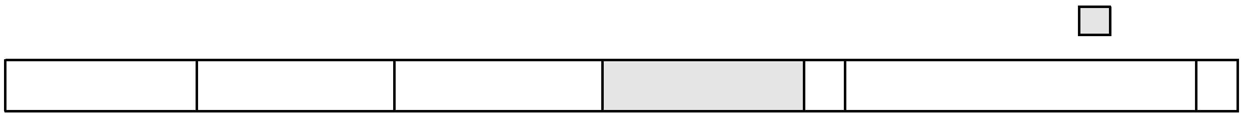
PowerPC Architecture Level	Supervisor Level	32-Bit	64-Bit	64-Bit Bridge	Optional	Form
UISA						D

addme*x* addme*x*

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

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

[POWER mnemonics: **ame**, **ame.**, **ameo**, **ameo.**]



```
rD← (rA) + XER[CA] - 1
```

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

Other registers altered:

- Condition Register (CR0 field):

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

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

  Affected: CA

  Affected: SO, OV(if OE = 1)

  Note: The setting of the affected bits in the XER is mode-dependent, and reflects overflow of the 64-bit result in 64-bit mode and overflow of the low-order 32-bit result in 32-bit mode. For further information about 64-bit mode and 32-bit mode in 64-bit implementations, see 4.1.2 , “Computation Modes.”

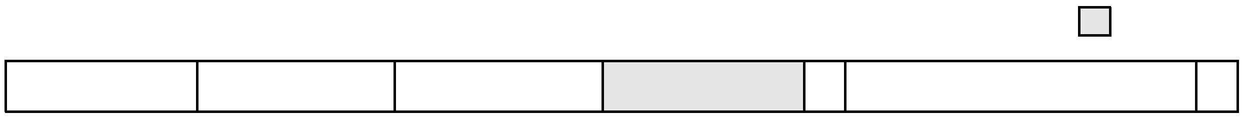
PowerPC Architecture Level	Supervisor Level	32-Bit	64-Bit	64-Bit Bridge	Optional	Form
UISA						XO

addze*x* **addze*x***

Add to Zero Extended (x’7C00 0194’)

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

[POWER mnemonics: **aze**, **aze.**, **azeo**, **azeo.**]



```
rD ← (rA) + XER[CA]
```

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

Other registers altered:

- Condition Register (CR0 field):

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

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

- XER:

  Affected: CA

  Affected: SO, OV(if OE = 1)

  Note: The setting of the affected bits in the XER is mode-dependent, and reflects overflow of the 64-bit result in 64-bit mode and overflow of the low-order 32-bit result in 32-bit mode. For further information about 64-bit mode and 32-bit mode in 64-bit implementations, see 4.1.2 , “Computation Modes.”

PowerPC Architecture Level	Supervisor Level	32-Bit	64-Bit	64-Bit Bridge	Optional	Form
UISA						XO

and*x* and*x*

AND (x'7C00 0038')

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


```
rA ← (rS) & (rB)
```

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

Other registers altered:

- Condition Register (CR0 field):

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

PowerPC Architecture Level	Supervisor Level	32-Bit	64-Bit	64-Bit Bridge	Optional	Form
UISA						X

andc*x* **andc***x*

AND with Complement (x’7C00 0078’)

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


```
rA← (rS) + ¬ (rB)
```

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

Other registers altered:

- Condition Register (CR0 field):

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

PowerPC Architecture Level	Supervisor Level	32-Bit	64-Bit	64-Bit Bridge	Optional	Form
UISA						X

andi. andi.

AND Immediate (x’7000 0000’)

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

[POWER mnemonic: **andil.**]


```
rA← (rS) & ((4816)0 || UIMM)
```

The contents of **r**S are ANDed with 0x0000_0000_0000 || UIMM and the result is placed into **r**A.

Other registers altered:

- Condition Register (CR0 field):

  Affected: LT, GT, EQ, SO

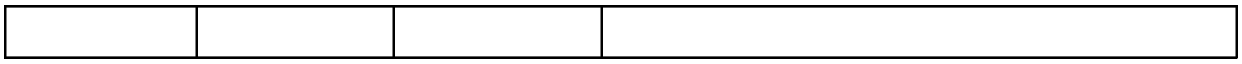
PowerPC Architecture Level	Supervisor Level	32-Bit	64-Bit	64-Bit Bridge	Optional	Form
UISA						D

andis. andis.

AND Immediate Shifted (x’7400 0000’)

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

[POWER mnemonic: **andiu.**]



```
rA← (rS) + ((32)0 || UIMM || (16)0)
```

The contents of **r**S are ANDed with 0x0000_0000 || UIMM || 0x0000 and the result is placed into **r**A.

Other registers altered:

- Condition Register (CR0 field):

  Affected: LT, GT, EQ, SO

PowerPC Architecture Level	Supervisor Level	32-Bit	64-Bit	64-Bit Bridge	Optional	Form
UISA						D

b*x* **b***x*

Branch (x’4800 0000’)

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

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

target_addr specifies the branch target address.

If AA = 0, then the branch target address is the sum of LI || 0b00 sign-extended and the address of this instruction, with the high-order 32 bits of the branch target address cleared in 32-bit mode of 64-bit implementations.

If AA = 1, then the branch target address is the value LI || 0b00 sign-extended, with the high-order 32 bits of the branch target address cleared in 32-bit mode of 64-bit implementations.

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

Other registers altered:

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

PowerPC Architecture Level	Supervisor Level	32-Bit	64-Bit	64-Bit Bridge	Optional	Form
UISA						I

bc*x* **bc***x*

Branch Conditional (x’4000 0000’)

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

0–5	6–10	11–15	16–29	30	31
16	BO	BI	BD	AA	LK

```
if (64-bit implementation) & (64-bit mode)
then m← 0
else m← 32
if ¬ BO[2] then CTR ← CTR - 1
ctr_ok ← BO[2] | ((CTR[m–63] ¦ 0) ⊕ BO[3])
cond_ok ←BO[0] | (CR[BI] ≡ BO[1])
if ctr_ok & cond_ok then
 if AA then NIA ←iea EXTS(BD || 0b00)
 else NIA ←iea CIA + EXTS(BD || 0b00)
 if LK then LR ←iea CIA + 4
```

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

Table 8-6. BO Operand Encodings

BO	Description
0000*y*	Decrement the CTR, then branch if the decremented CTR[M–63] ¦ 0 and the condition is FALSE.
0001*y*	Decrement the CTR, then branch if the decremented CTR[M–63] = 0 and the condition is FALSE.
001*zy*	Branch if the condition is FALSE.
0100*y*	Decrement the CTR, then branch if the decremented CTR[M–63] ¦ 0 and the condition is TRUE.
0101*y*	Decrement the CTR, then branch if the decremented CTR[M–63] = 0 and the condition is TRUE.
011*zy*	Branch if the condition is TRUE.
1*z*00*y*	Decrement the CTR, then branch if the decremented CTR[M–63] ¦ 0.
1*z*01*y*	Decrement the CTR, then branch if the decremented CTR[M–63] = 0.
1*z*1*zz*	Branch always.

M = 32 in 32-bit mode, and M = 0 in the default 64-bit mode. If the BO field specifies that the CTR is to be decremented, the entire 64-bit CTR is decremented regardless of the 32-bit mode or the default 64-bit mode.
In this table, *z* indicates a bit that is ignored.
Note that the *z* bits should be cleared, as they may be assigned a meaning in some future version of the PowerPC architecture.
The *y* bit provides a hint about whether a conditional branch is likely to be taken, and may be used by some PowerPC implementations to improve performance.

target_addr specifies the branch target address.

If AA = 0, the branch target address is the sum of BD || 0b00 sign-extended and the address of this instruction, with the high-order 32 bits of the branch target address cleared in 32-bit mode of 64-bit implementations.

If AA = 1, the branch target address is the value BD || 0b00 sign-extended, with the high-order 32 bits of the branch target address cleared in 32-bit mode of 64-bit implementations.

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

Other registers altered:

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

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

Simplified mnemonics:

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

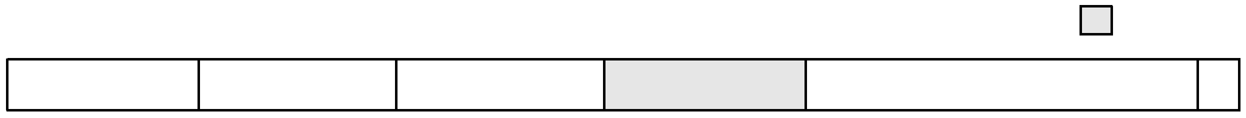
PowerPC Architecture Level	Supervisor Level	32-Bit	64-Bit	64-Bit Bridge	Optional	Form
UISA						B

bcctr*x* bcctr*x*

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

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

[POWER mnemonics: **bcc**, **bccl**]



```
cond_ok ← BO[0] | (CR[BI] ≡ BO[1])
if cond_ok then
  NIA ←iea CTR[0-61] || 0b00
  if LK then LR ←iea CIA + 4
```

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

Table 8-7. BO Operand Encodings

BO	Description
0000*y*	Decrement the CTR, then branch if the decremented CTR[M–63] ¦ 0 and the condition is FALSE.
0001*y*	Decrement the CTR, then branch if the decremented CTR[M–63] = 0 and the condition is FALSE.
001*zy*	Branch if the condition is FALSE.
0100*y*	Decrement the CTR, then branch if the decremented CTR[M–63] ¦ 0 and the condition is TRUE.
0101*y*	Decrement the CTR, then branch if the decremented CTR[M–63] = 0 and the condition is TRUE.
011*zy*	Branch if the condition is TRUE.
1*z*00*y*	Decrement the CTR, then branch if the decremented CTR[M–63] ¦ 0.
1*z*01*y*	Decrement the CTR, then branch if the decremented CTR[M–63] = 0.
1*z*1*zz*	Branch always.
M = 32 in 32-bit mode, and M = 0 in the default 64-bit mode. If the BO field specifies that the CTR is to be decremented, the entire 64-bit CTR is decremented regardless of the 32-bit mode or the default 64-bit mode. In this table, *z* indicates a bit that is ignored. Note that the *z* bits should be cleared, as they may be assigned a meaning in some future version of the PowerPC architecture. The *y* bit provides a hint about whether a conditional branch is likely to be taken, and may be used by some PowerPC implementations to improve performance.	

The branch target address is CTR[0–61] || 0b00, with the high-order 32 bits of the branch target address cleared in 32-bit mode of 64-bit implementations.

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

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

Other registers altered:

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

Simplified mnemonics:

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

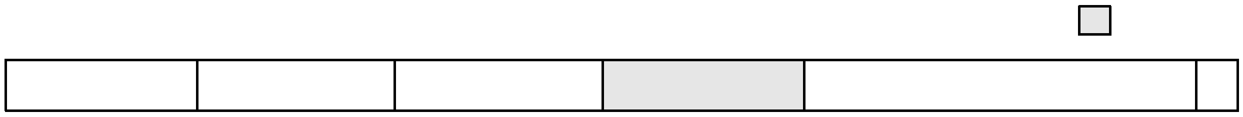
PowerPC Architecture Level	Supervisor Level	32-Bit	64-Bit	64-Bit Bridge	Optional	Form
UISA						XL

bclr*x* bclr*x*

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

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

[POWER mnemonics: **bcr**, **bcrl**]



```
if (64-bit implementation) & (64-bit mode)
then m← 0
else m← 32
if ¬ BO[2] then CTR ← CTR - 1
ctr_ok ← BO[2] | ((CTR[m–63] ¦ 0) ⊕ BO[3])
cond_ok ← BO[0] | (CR[BI] ≡ BO[1])
if ctr_ok & cond_ok then
 NIA ←iea LR[0–61] || 0b00
 if LK then LR ←iea CIA + 4
```

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

Table 8-8. BO Operand Encodings

BO	Description
0000*y*	Decrement the CTR, then branch if the decremented CTR[M–63] ¦ 0 and the condition is FALSE.
0001*y*	Decrement the CTR, then branch if the decremented CTR[M–63] = 0 and the condition is FALSE.
001*zy*	Branch if the condition is FALSE.
0100*y*	Decrement the CTR, then branch if the decremented CTR[M–63] ¦ 0 and the condition is TRUE.
0101*y*	Decrement the CTR, then branch if the decremented CTR[M–63] = 0 and the condition is TRUE.
011*zy*	Branch if the condition is TRUE.
1*z*00*y*	Decrement the CTR, then branch if the decremented CTR[M–63] ¦ 0.
1*z*01*y*	Decrement the CTR, then branch if the decremented CTR[M–63] = 0.
1*z*1*zz*	Branch always.

M = 32 in 32-bit mode, and M = 0 in the default 64-bit mode. If the BO field specifies that the CTR is to be decremented, the entire 64-bit CTR is decremented regardless of the 32-bit mode or the default 64-bit mode.
In this table, *z* indicates a bit that is ignored.
Note that the *z* bits should be cleared, as they may be assigned a meaning in some future version of the PowerPC architecture.
The *y* bit provides a hint about whether a conditional branch is likely to be taken, and may be used by some PowerPC implementations to improve performance.

The branch target address is LR[0–61] || 0b00, with the high-order 32 bits of the branch target address cleared in 32-bit mode of 64-bit implementations.

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

Other registers altered:

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

Simplified mnemonics:

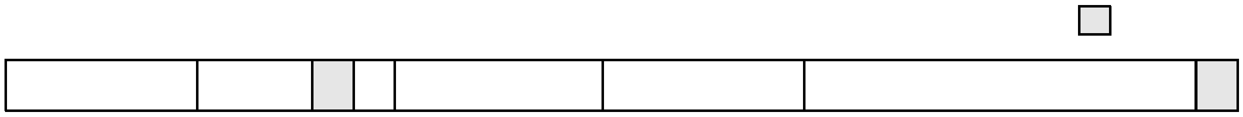
bltlr		equivalent to	**bclr**	**12,0**
bnelr	**cr2**	equivalent to	**bclr**	**4,10**
bdnzlr		equivalent to	**bclr**	**16,0**

PowerPC Architecture Level	Supervisor Level	32-Bit	64-Bit	64-Bit Bridge	Optional	Form
UISA						XL

cmp   cmp

Compare (x’7C00 0000’)

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



```
if L = 0 then a ← EXTS(rA[32-63])
              b ← EXTS(rB[32-63])
else          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 (or the low-order 32 bits of **r**A if L = 0) are compared with the contents of **r**B (or the low-order 32 bits of **r**B if L = 0), treating the operands as signed integers. The result of the comparison is placed into CR field **crf**D.

In 32-bit implementations, if L = 1 the instruction form is invalid.

Other registers altered:

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

  Affected: LT, GT, EQ, SO

Simplified mnemonics:

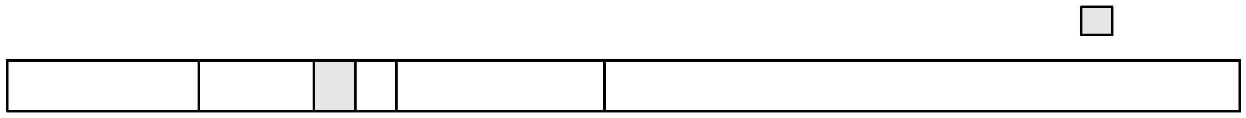
cmpd	**r**A,**r**B	equivalent to	**cmp**	**0,1,r**A,**r**B
cmpw	**cr3,r**A,**r**B	equivalent to	**cmp**	**3,0,r**A,**r**B

PowerPC Architecture Level	Supervisor Level	32-Bit	64-Bit	64-Bit Bridge	Optional	Form
UISA						X

cmpi cmpi

Compare Immediate (x’2C00 0000’)

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



```
if L = 0 then a ← EXTS(rA[32-63])
      elsea ← (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 (or the low-order 32 bits of **r**A sign-extended to 64 bits if L = 0) 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.

In 32-bit implementations, if L = 1 the instruction form is invalid.

Other registers altered:

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

  Affected: LT, GT, EQ, SO

Simplified mnemonics:

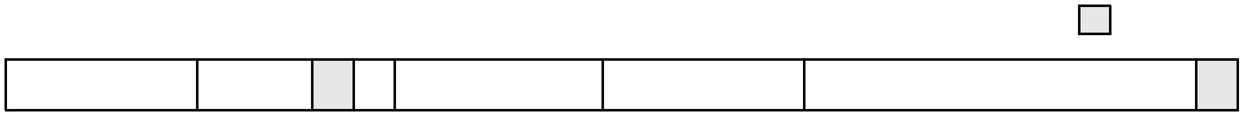
cmpdi	**r**A,value	equivalent to	**cmpi**	**0,1,r**A,value
cmpwi	**cr3,r**A,value	equivalent to	**cmpi**	**3,0,r**A,value

PowerPC Architecture Level	Supervisor Level	32-Bit	64-Bit	64-Bit Bridge	Optional	Form
UISA						D

cmpl cmpl

Compare Logical (x’7C00 0040’)

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



```
if L = 0 then a ← (32)0 || rA[32-63]
        b← (32)0 || rB[32-63]
      elsea ← (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 (or the low-order 32 bits of **r**A if L = 0) are compared with the contents of **r**B (or the low-order 32 bits of **r**B if L = 0), treating the operands as unsigned integers. The result of the comparison is placed into CR field **crf**D.

In 32-bit implementations, if L = 1 the instruction form is invalid.

Other registers altered:

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

  Affected: LT, GT, EQ, SO

Simplified mnemonics:

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

PowerPC Architecture Level	Supervisor Level	32-Bit	64-Bit	64-Bit Bridge	Optional	Form
UISA						X

cmpli

Compare Logical Immediate (x'2800 0000')

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

10	crfD	0	L	A	

```
if L = 0 then a ← (32)0 || rA[32-63]
      else a ← (rA)
if   a <U ((4816)0 || UIMM) then c ← 0b100
else if a >U ((4816)0 || UIMM) then c ← 0b010
else     c ← 0b001
CR[4 * crfD-4 * crfD + 3] ← c || XER[SO]
```

The contents of **r**A (or the low-order 32 bits of **r**A zero-extended to 64-bits if L = 0) are compared with 0x0000_0000_0000 || UIMM, treating the operands as unsigned integers. The result of the comparison is placed into CR field **crf**D.

In 32-bit implementations, if L = 1 the instruction form is invalid.

Other registers altered:

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

  Affected: LT, GT, EQ, SO

Simplified mnemonics:

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

PowerPC Architecture Level	Supervisor Level	32-Bit	64-Bit	64-Bit Bridge	Optional	Form
UISA						D

cntlzd*x* **64-Bit Implementations Only** **cntlzd***x*

Count Leading Zeros Double Word (x’7C00 0074’)

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

▨ Reserved

31	S	A	▨ 0 0 0 0 0	58	Rc
0 5	6 10	11 15	16 20	21 30	31

```
n ← 0
do while n < 64
   if rS[n] = 1 then leave
   n ← n + 1
rA ← n
```

A count of the number of consecutive zero bits starting at bit 0 of register **r**S is placed into **r**A. This number ranges from 0 to 64, inclusive.

This instruction is defined only for 64-bit implementations. Using it on a 32-bit implementation will cause the system illegal instruction error handler to be invoked.

Other registers altered:

- Condition Register (CR0 field):

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

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

PowerPC Architecture Level	Supervisor Level	32-Bit	64-Bit	64-Bit Bridge	Optional	Form
UISA			Ð			X

cntlzw*x* **cntlzw***x*

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

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

[POWER mnemonics: **cntlz**, **cntlz.**]

▨ Reserved

31	S	A	0 0 0 0 0	26	Rc
0 — 5	6 — 10	11 — 15	16 — 20	21 — 30	31

```
n ← 320
do while n < 6432
if rS[n] = 1 then leave
n ← n + 1
rA ← n– 32
```

A count of the number of consecutive zero bits starting at bit 320 of **r**S (bit 0 in 32-bit implementations) is placed into **r**A. This number ranges from 0 to 32, inclusive.

Other registers altered:

- Condition Register (CR0 field):

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

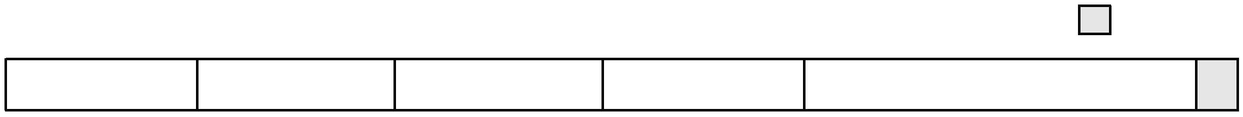
  Note: If Rc = 1, then LT is cleared in the CR0 field.

PowerPC Architecture Level	Supervisor Level	32-Bit	64-Bit	64-Bit Bridge	Optional	Form
UISA						X

crand crand

Condition Register AND (x’4C00 0202’)

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



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

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

Other registers altered:

- Condition Register:

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

PowerPC Architecture Level	Supervisor Level	32-Bit	64-Bit	64-Bit Bridge	Optional	Form
UISA						XL

crandc crandc

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

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

▨ Reserved

19	crbD	crbA	crbB	129	▨ 0
0 5	6 10	11 15	16 20	21 30	31

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

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

Other registers altered:

- Condition Register:

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

PowerPC Architecture Level	Supervisor Level	32-Bit	64-Bit	64-Bit Bridge	Optional	Form
UISA						XL

creqv

creqv

Condition Register Equivalent (x’4C00 0242’)

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

▨ Reserved

19	crbD	crbA	crbB	289	0
0 5	6 10	11 15	16 20	21 30	31

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

Simplified mnemonics:

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

PowerPC Architecture Level	Supervisor Level	32-Bit	64-Bit	64-Bit Bridge	Optional	Form
UISA						XL

crnand crnand

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

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

▨ Reserved

19	crbD	crbA	crbB	225	0
0 5	6 10	11 15	16 20	21 30	31

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

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

Other registers altered:

- Condition Register:

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

PowerPC Architecture Level	Supervisor Level	32-Bit	64-Bit	64-Bit Bridge	Optional	Form
UISA						XL

crnor crnor

Condition Register NOR (x’4C00 0042’)

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

▨ Reserved

19	crbD	crbA	crbB	33	0
0 5	6 10	11 15	16 20	21 30	31

```
CR[crbD] ← ¬ (CR[crbA] | CR[crbB])
```

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

Other registers altered:

- Condition Register:

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

Simplified mnemonics:

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

PowerPC Architecture Level	Supervisor Level	32-Bit	64-Bit	64-Bit Bridge	Optional	Form
UISA						XL

cror cror

Condition Register OR (x’4C00 0382’)

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

▨ Reserved

19	crbD	crbA	crbB	449	0
0 5	6 10	11 15	16 20	21 30	31

CR[**crb**D] $\leftarrow$ 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

Simplified mnemonics:

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

PowerPC Architecture Level	Supervisor Level	32-Bit	64-Bit	64-Bit Bridge	Optional	Form
UISA						XL

crorc

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

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

▨ Reserved

19	crbD	crbA	crbB	417	0
0 5	6 10	11 15	16 20	21 30	31

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

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

Other registers altered:

- Condition Register:

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

PowerPC Architecture Level	Supervisor Level	32-Bit	64-Bit	64-Bit Bridge	Optional	Form
UISA						XL

crxor crxor

Condition Register XOR (x’4C00 0182’)

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

▨ Reserved

19	crbD	crbA	crbB	193	▨ 0
0 5	6 10	11 15	16 20	21 30	31

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

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

Other registers altered:

- Condition Register:

  Affected: Bit specified by **crb**D

Simplified mnemonics:

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

PowerPC Architecture Level	Supervisor Level	32-Bit	64-Bit	64-Bit Bridge	Optional	Form
UISA						XL

dcba dcba

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

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

▨ Reserved

0–5	6–10	11–15	16–20	21–30	31
31	0 0 0 0 0	A	B	758	0

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

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

The **dcba** instruction executes as follows:

- If the cache block containing the byte addressed by EA is in the data cache, the contents of all bytes are made undefined but the cache block is still considered valid. Note that programming errors can occur if the data in this cache block is subsequently read or used inadvertently.
- If the cache block containing the byte addressed by EA is not in the data cache and the corresponding memory page or block is caching-allowed, the cache block is allocated (and made valid) in the data cache without fetching the block from main memory, and the value of all bytes is undefined.
- If the addressed byte corresponds to a caching-inhibited page or block (i.e. if the I bit is set), this instruction is treated as a no-op.
- If the cache block containing the byte addressed by EA is in coherency-required mode, and the cache block exists in the data cache(s) of any other processor(s), it is kept coherent in those caches (i.e. the processor performs the appropriate bus transactions to enforce this).

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

This instruction is optional in the PowerPC architecture.

Other registers altered:

- None

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

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

PowerPC Architecture Level	Supervisor Level	32-Bit	64-Bit	64-Bit Bridge	Optional	Form
VEA					Ð	X

dcbf

Data Cache Block Flush (x’7C00 00AC’)

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

▨ Reserved

31	00 000	A	B	86	0
0 5	6 10	11 15	16 20	21 30	31

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

The **dcbf** instruction invalidates the block in the data cache addressed by EA, copying the block to memory first, if there is any dirty data in it. If the processor is a multiprocessor implementation (for example, the 601, 604,and 604e and 620) and the block is marked coherency-required, the processor will, if necessary, send an address-only broadcast to other processors. The broadcast of the **dcbf** instruction causes another processor to copy the block to memory, if it has dirty data, and then invalidate the block from the cache.

The action taken depends on the memory mode associated with the block containing the byte addressed by EA and on the state of that block. The list below describes the action taken for the various states of the memory coherency attribute (M bit).

- Coherency required
  - Unmodified block—Invalidates copies of the block in the data caches of all processors.
  - Modified block—Copies the block to memory. Invalidates copies of the block in the data caches of all processors.
  - Absent block—If modified copies of the block are in the data caches of other processors, causes them to be copied to memory and invalidated in those data caches. If unmodified copies are in the data caches of other processors, causes those copies to be invalidated in those data caches.
- Coherency not required
  - Unmodified block—Invalidates the block in the processor’s data cache.
  - Modified block—Copies the block to memory. Invalidates the block in the processor’s data cache.
  - Absent block (target block not in cache)—No action is taken.

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

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

Other registers altered:

- None

PowerPC Architecture Level	Supervisor Level	32-Bit	64-Bit	64-Bit Bridge	Optional	Form
VEA						X

dcbi

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

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

▨ Reserved

0–5	6–10	11–15	16–20	21–30	31
31	0 0 0 0 0	A	B	470	0

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

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

- Coherency required
  - Unmodified block—Invalidates copies of the block in the data caches of all processors.
  - Modified block—Invalidates copies of the block in the data caches of all processors. (Discards the modified contents.)
  - Absent block—If copies of the block are in the data caches of any other processor, causes the copies to be invalidated in those data caches. (Discards any modified contents.)
- Coherency not required
  - Unmodified block—Invalidates the block in the processor’s data cache.
  - Modified block—Invalidates the block in the processor’s data cache. (Discards the modified contents.)
  - Absent block (target block not in cache)—No action is taken.

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

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

This is a supervisor-level instruction.

Other registers altered:

- None

PowerPC Architecture Level	Supervisor Level	32-Bit	64-Bit	64-Bit Bridge	Optional	Form
OEA	Ð					X

dcbst

dcbst

Data Cache Block Store (x’7C00 006C’)

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

▨ Reserved

31	00 000	A	B	54	0
0 5	6 10	11 15	16 20	21 30	31

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

The **dcbst** instruction executes as follows:

- If the block containing the byte addressed by EA is in coherency-required mode, and a block containing the byte addressed by EA is in the data cache of any processor and has been modified, the writing of it to main memory is initiated.
- If the block containing the byte addressed by EA is in coherency-not-required mode, and a block containing the byte addressed by EA is in the data cache of this processor and has been modified, the writing of it to main memory is initiated.

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

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

Other registers altered:

- None

PowerPC Architecture Level	Supervisor Level	32-Bit	64-Bit	64-Bit Bridge	Optional	Form
VEA						X

dcbt dcbt

Data Cache Block Touch (x’7C00 022C’)

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

▨ Reserved

31	0 0 0 0 0	A	B	278	0
0 – 5	6 – 10	11 – 15	16 – 20	21 – 30	31

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

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

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

The program uses the **dcbt** instruction to request a cache block fetch before it is actually needed by the program. The program can later execute load instructions to put data into registers. However, the processor is not obliged to load the addressed block into the data cache. Note that this instruction is defined architecturally to perform the same functions as the **dcbtst** instruction. Both are defined in order to allow implementations to differentiate the bus actions when fetching into the cache for the case of a load and for a store.

Other registers altered:

- None

PowerPC Architecture Level	Supervisor Level	32-Bit	64-Bit	64-Bit Bridge	Optional	Form
VEA						X

dcbtst dcbtst

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

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

▨ Reserved

31	0 0 0 0 0	A	B	246	0
0 5	6 10	11 15	16 20	21 30	31

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

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

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

The program uses **dcbtst** to request a cache block fetch to potentially improve performance for a subsequent store to that EA, as that store would then be to a cached location. However, the processor is not obliged to load the addressed block into the data cache. Note that this instruction is defined architecturally to perform the same functions as the **dcbt** instruction. Both are defined in order to allow implementations to differentiate the bus actions when fetching into the cache for the case of a load and for a store.

Other registers altered:

- None

PowerPC Architecture Level	Supervisor Level	32-Bit	64-Bit	64-Bit Bridge	Optional	Form
VEA						X

dcbz dcbz

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

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

[POWER mnemonic: **dclz**]

▨ Reserved

0–5	6–10	11–15	16–20	21–30	31
31	0 0 0 0 0	A	B	1014	0

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

The **dcbz** instruction executes as follows:

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

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

Other registers altered:

- None

The PowerPC OEA describes how the **dcbz** instruction may establish a block in the data cache without verifying that the associated physical address is valid. This scenario can cause a delayed machine check exception; see 6. , “Exceptions,” for a discussion about this type of machine check exception.

PowerPC Architecture Level	Supervisor Level	32-Bit	64-Bit	64-Bit Bridge	Optional	Form
VEA						X

divd*x* 64-Bit Implementations Only divd*x*

Divide Double Word (x'7C00 03D2')

divd	**r**D,**r**A,**r**B	(OE = 0 Rc = 0)
divd.	**r**D,**r**A,**r**B	(OE = 0 Rc = 1)
divdo	**r**D,**r**A,**r**B	(OE = 1 Rc = 0)
divdo.	**r**D,**r**A,**r**B	(OE = 1 Rc = 1)

31	D	A	B	OE	489	Rc
0 5	6 10	11 15	16 20	21	22 30	31

```
dividend[0-63] ← (rA)
divisor[0-63] ← (rB)
rD ← dividend + divisor
```

The 64-bit dividend is the contents of **r**A. The 64-bit divisor is the contents of **r**B. The 64-bit quotient is placed into **r**D. The remainder is not supplied as a result.

Both the operands and the quotient are interpreted as signed integers. The quotient is the unique signed integer that satisfies the equation—dividend = (quotient * divisor) + r—where 0 ð r < |divisor| if the dividend is non-negative, and –|divisor| < r ð 0 if the dividend is negative.

If an attempt is made to perform the divisions—0x8000_0000_0000_0000 ÷ –1 or <anything> ÷ 0—the contents of **r**D are undefined, as are the contents of the LT, GT, and EQ bits of the CR0 field (if Rc = 1). In this case, if OE = 1 then OV is set.

This instruction is defined only for 64-bit implementations. Using it on a 32-bit implementation will cause the system illegal instruction error handler to be invoked.

The 64-bit signed remainder of dividing (**r**A) by (**r**B) can be computed as follows, except in the case that (**r**A) = -2^{63} and (**r**B) = –1:

divd	**r**D,**r**A,**r**B	# **r**D = quotient
mulld	**r**D,**r**D,**r**B	# **r**D = quotient * divisor
subf	**r**D,**r**D,**r**A	# **r**D = remainder

Other registers altered:

- Condition Register (CR0 field):
  Affected: LT, GT, EQ, SO(if Rc = 1)
- XER:
  Affected: SO, OV (if OE = 1)
  Note: The setting of the affected bits in the XER is mode-independent, and reflects overflow of the 64-bit result.

PowerPC Architecture Level	Supervisor Level	32-Bit	64-Bit	64-Bit Bridge	Optional	Form
UISA			Ð			XO

divdu*x* 64-Bit Implementations Only **divdu***x*

Divide Double Word Unsigned (x’7C00 0392’)

divdu	rD,**r**A,**r**B	(OE = 0 Rc = 0)
divdu.	rD,**r**A,**r**B	(OE = 0 Rc = 1)
divduo	rD,**r**A,**r**B	(OE = 1 Rc = 0)
divduo.	rD,**r**A,**r**B	(OE = 1 Rc = 1)

31	D	A	B	OE	457	Rc
0 5	6 10	11 15	16 20	21	22 30	31

```
dividend[0-63] ← (rA)
divisor[0-63] ← (rB)
rD ← dividend + divisor
```

The 64-bit dividend is the contents of **r**A. The 64-bit divisor is the contents of **r**B. The 64-bit quotient of the dividend and divisor is placed into **r**D. The remainder is not supplied as a result.

Both the operands and the quotient are interpreted as unsigned integers, except that if Rc is set to 1 the first three bits of CR0 field are set by signed comparison of the result to zero. The quotient is the unique unsigned integer that satisfies the equation—dividend = (quotient ∗ divisor) + r—where 0 ð r < divisor.

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

This instruction is defined only for 64-bit implementations. Using it on a 32-bit implementation will cause the system illegal instruction error handler to be invoked.

The 64-bit unsigned remainder of dividing (**r**A) by (**r**B) can be computed as follows:

divdu	rD,**r**A,**r**B	# **r**D = quotient
mulld	rD,**r**D,**r**B	# **r**D = quotient * divisor
subf	rD,**r**D,**r**A	# **r**D = remainder

Other registers altered:

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

  Note: The setting of the affected bits in the XER is mode-independent, and reflects overflow of the 64-bit result.

PowerPC Architecture Level	Supervisor Level	32-Bit	64-Bit	64-Bit Bridge	Optional	Form
UISA			Ð			XO

divw*x* divw*x*

Divide Word (x’7C00 03D6’)

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

31	D	A	B	OE	491	Rc
0 5	6 10	11 15	16 20	21	22 30	31

```
dividend[0-63]← EXTS(rA[32-63])
divisor[0-63] ← EXTS(rB[32-63])
rD[32-63] ← dividend ÷ divisor
rD[0-31] ← undefined
```

The 64-bit dividend is the sign-extended value of the contents of the low-order 32 bits of **r**A. The 64-bit divisor is the sign-extended value of the contents of the low-order 32 bits of **r**B. The 6432-bit quotient is formed and placed in **r**D. The low-order 32 bits of the 64-bit quotient are placed into the low-order 32 bits of **r**D. The contents of the high-order 32 bits of **r**D are undefined. The remainder is not supplied as a result.

Both the operands and the quotient are interpreted as signed integers. The quotient is the unique signed integer that satisfies the equation—dividend = (quotient * divisor) + r where 0 ð r < |divisor| (if the dividend is non-negative), and –|divisor| < r ð 0 (if the dividend is negative).

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

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

divw	**r**D,**r**A,**r**B	# **r**D = quotient
mullw	**r**D,**r**D,**r**B	# **r**D = quotient ∗ divisor
subf	**r**D,**r**D,**r**A	# **r**D = remainder

Other registers altered:

- Condition Register (CR0 field):
  Affected: LT, GT, EQ, SO(if Rc = 1)
  LT, GT, EQ undefined(if Rc =1 and 64-bit mode)
- XER:
  Affected: SO, OV(if OE = 1)
  Note: The setting of the affected bits in the XER is mode-independent, and reflects overflow of the low-order 32-bit result.

PowerPC Architecture Level	Supervisor Level	32-Bit	64-Bit	64-Bit Bridge	Optional	Form
UISA						XO

divwu*x* **divwu***x*

Divide Word Unsigned (x’7C00 0396’)

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

31	D	A	B	OE	459	Rc
0 5	6 10	11 15	16 20	21	22 30	31

```
dividend[0-63] ← (32)0 || (rA)[32-63]
divisor[0-63] ← (32)0‖(rB)[32-63]
rD[32-63] ← dividend ÷ divisor
rD[0-31] ← undefined
```

The 64-bit dividend is the zero-extended value of the contents of the low-order 32 bits of **r**A. The 64-bit divisor is the zero-extended value the contents of the low-order 32 bits of **r**B. A 6432-bit quotient is formed. The low-order 32 bits of the 6432-bit quotient areis placed into the low-order 32 bits of **r**D. The contents of the high-order 32 bits of **r**D are undefined. The remainder is not supplied as a result.

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

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

divwurD,**r**A,**r**B# **r**D = quotient
mullw rD,**r**D,**r**B# **r**D = quotient ∗ divisor
subf rD,**r**D,**r**A # **r**D = remainder

Other registers altered:

- Condition Register (CR0 field):
  Affected: LT, GT, EQ, SO(if Rc = 1)
  LT, GT, EQ undefined(if Rc =1 and 64-bit mode)
- XER:
  Affected: SO, OV(if OE = 1)

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

PowerPC Architecture Level	Supervisor Level	32-Bit	64-Bit	64-Bit Bridge	Optional	Form
UISA						XO

eciwx eciwx

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

eciwx rD,rA,rB

▨ Reserved

31	D	A	B	310	0
0 5	6 10	11 15	16 20	21 30	31

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

```
if rA = 0 then b ← 0
else b← (rA)
EA ← b + (rB)
paddr ← address translation of EA
send load word request for paddr to device identified by EAR[RID]
rD ← (32)0 || word from device
```

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

A load word request for the physical address (referred to as real address in the architecture specification) corresponding to EA is sent to the device identified by EAR[RID], bypassing the cache. The word returned by the device is placed in the low-order 32 bits of **r**D. The contents of the high-order 32 bits of **r**D are cleared.

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

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

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

The **eciwx** instruction is supported for EAs that reference memory segments in which SR[T] = 1 (or STE[T] = 1) and for EAs mapped by the DBAT registers. If the EA references a direct-store segment (SR[T] = 1 or STE[T] = 1), either a DSI exception occurs or the results are boundedly undefined. However, note that the direct-store facility is being phased out of the architecture and will not likely be supported in future devices. Thus, software should not depend on its effects.

If this instruction is executed when MSR[DR] = 0 (real addressing mode), the results are boundedly undefined. This instruction is treated as a load from the addressed byte with respect to address translation, memory protection, referenced and changed bit recording, and the ordering performed by **eieio**. This instruction is optional in the PowerPC architecture.

Other registers altered:

- None

PowerPC Architecture Level	Supervisor Level	32-Bit	64-Bit	64-Bit Bridge	Optional	Form
VEA					Ð	X

ecowx ecowx

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

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

▨ Reserved

31	S	A	B	438	0
0 5	6 10	11 15	16 20	21 30	31

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

```
if rA = 0 then b ← 0
else b ← (rA)
EA ← b + (rB)
paddr ← address translation of EA
send store word request for paddr to device identified by EAR[RID]
send rS[32-63] to device
```

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

A store word request for the physical address corresponding to EA and the contents of the low-order 32 bits of **r**S are sent to the device identified by EAR[RID], bypassing the cache.

EAR[E] must be 1, if it is not, a DSI exception is generated. EA must be a multiple of four. If it is not, one of the following occurs:

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

The **ecowx** instruction is supported for effective addresses that reference memory segments in which SR[T] = 0 (or STE[T] = 0), and for EAs mapped by the DBAT registers. If the EA references a direct-store segment (SR[T] = 1 or STE[T] = 1), either a DSI exception occurs or the results are boundedly undefined. However, note that the direct-store facility is being phased out of the architecture and will not likely be supported in future devices. Thus, software should not depend on its effects.

If this instruction is executed when MSR[DR] = 0 (real addressing mode), the results are boundedly undefined. This instruction is treated as a store from the addressed byte with respect to address translation, memory protection, and referenced and changed bit recording, and the ordering performed by **eieio**. Note that software synchronization is required in order to ensure that the data access is performed in program order with respect to data accesses caused by other store or **ecowx** instructions, even though the addressed byte is assumed to be caching-inhibited and guarded. This instruction is optional in the PowerPC architecture.

Other registers altered:

- None

PowerPC Architecture Level	Supervisor Level	32-Bit	64-Bit	64-Bit Bridge	Optional	Form
VEA					Ð	X

eieio

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

▨ Reserved

0–5	6–10	11–15	16–20	21–30	31
31	00 000 ▨	0 0000 ▨	0000 0 ▨	854	0 ▨

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

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

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

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

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

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

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

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

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

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

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

Other registers altered:

- None

PowerPC Architecture Level	Supervisor Level	32-Bit	64-Bit	64-Bit Bridge	Optional	Form
VEA						X

eqv*x*

Equivalent (x’7C00 0238’)

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

31	S	A	**B**	**284**	**Rc**
0 5	6 10	11 15	16 21	22 30	31

```
    rA ← (rS) ≡ (rB)
```

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

Other registers altered:

- Condition Register (CR0 field):

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

PowerPC Architecture Level	Supervisor Level	32-Bit	64-Bit	64-Bit Bridge	Optional	Form
UISA						X

extsb*x* extsb*x*

Extend Sign Byte (x'7C00 0774')

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

▨ Reserved

31	S	A	0 0 0 0 0	954	Rc
0 5	6 10	11 15	16 20	21 30	31

```
S ← rS[5624]
rA[56-6324-31] ← rS[56-6324-31]
rA[0-5523] ← (5624)S
```

The contents of the low-order eight bits of **r**S[24-31] are placed into the low-order eight bits of **r**A[24-31]. Bit 5624 of **r**S is placed into the remaining bits of **r**A[0-23].

Other registers altered:

- Condition Register (CR0 field):

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

PowerPC Architecture Level	Supervisor Level	32-Bit	64-Bit	64-Bit Bridge	Optional	Form
UISA						X

extsh*x* **extsh***x*

Extend Sign Half Word (x’7C00 0734’)

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

[POWER mnemonics: **exts**, **exts.**]

▨ Reserved

31	S	A	0 0 0 0 0	922	Rc
0 5	6 10	11 15	16 20	21 30	31

```
S ← rS[4816]
rA[48-6316-31] ← rS[48-6316-31]
rA[0-470-15] ← (4816)S
```

The contents of the low-order 16 bits of **r**S[16-31] are placed into the low-order 16 bits of **r**A[16-31]. Bit 4816 of **r**S is placed into the remaining bits of **r**A[0–15].

Other registers altered:

- Condition Register (CR0 field):

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

PowerPC Architecture Level	Supervisor Level	32-Bit	64-Bit	64-Bit Bridge	Optional	Form
UISA						X

extsw*x* 64-Bit Implementations Only **extsw***x*

Extend Sign Word (x'7C00 07B4')

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

▨ Reserved

31	S	A	0 0 0 0 0 ▨	986	Rc
0 5	6 10	11 15	16 20	21 30	31

```
S ← rS[32]
rA[32-63] ← rS[32-63]
rA[0-31] ← (32)S
```

The contents of the low-order 32 bits of **r**S are placed into the low-order 32 bits of **r**A. Bit 32 of **r**S is placed into the high-order 32 bits of **r**A.

This instruction is defined only for 64-bit implementations. Using it on a 32-bit implementation will cause the system illegal instruction error handler to be invoked.

Other registers altered:

- Condition Register (CR0 field):

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

PowerPC Architecture Level	Supervisor Level	32-Bit	64-Bit	64-Bit Bridge	Optional	Form
UISA			Ð			X

fabs*x* **fabs***x*

Floating Absolute Value (x’FC00 0210’)

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

▨ Reserved

63	D	0 0 0 0 0	B	264	Rc
0 5	6 10	11 15	16 20	21 30	31

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

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

Other registers altered:

- Condition Register (CR1 field):

  Affected: FX, FEX, VX, OX(if Rc = 1)

PowerPC Architecture Level	Supervisor Level	32-Bit	64-Bit	64-Bit Bridge	Optional	Form
UISA						X

fadd*x* **fadd***x*

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

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

[POWER mnemonics: **fa**, **fa.**]

▨

			B	▨	0 0 0 0 0 21	Rc

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

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

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

Other registers altered:

- Condition Register (CR1 field):

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

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

PowerPC Architecture Level	Supervisor Level	32-Bit	64-Bit	64-Bit Bridge	Optional	Form
UISA						A

fadds*x* **fadds***x*

Floating Add Single (x’EC00 002A’)

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

▨ Reserved

59	D	A	**B**	**0 0 0 0 0**	**21**	**Rc**
0 5	6 10	11 15	16 20	21 25	26 30	31

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

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

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

Other registers altered:

- Condition Register (CR1 field):

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

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

PowerPC Architecture Level	Supervisor Level	32-Bit	64-Bit	64-Bit Bridge	Optional	Form
UISA						A

fcfid*x* 64-Bit Implementations Only fcfid*x*

Floating Convert from Integer Double Word (x’FC00 069C’)

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

▨ Reserved

63	D	0 0000	B	846	Rc
0 5	6 10	11 15	16 20	21 30	31

The 64-bit signed fixed-point operand in register **fr**B is converted to an infinitely precise floating-point integer. The result of the conversion is rounded to double-precision using the rounding mode specified by FPSCR[RN] and placed into register **fr**D.

FPSCR[FPRF] is set to the class and sign of the result. FPSCR[FR] is set if the result is incremented when rounded. FPSCR[FI] is set if the result is inexact.

The conversion is described fully in Section D.4.3 , “Floating-Point Convert from Integer Model.”

This instruction is defined only for 64-bit implementations. Using it on a 32-bit implementation will cause the system illegal instruction error handler to be invoked.

Other registers altered:

- Condition Register (CR1 field):

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

  Affected: FPRF, FR, FI, FX, XX

PowerPC Architecture Level	Supervisor Level	32-Bit	64-Bit	64-Bit Bridge	Optional	Form
UISA			Đ			X

fcmpo

Floating Compare Ordered (x’FC00 0040’)

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

▨ Reserved

63	crfD	0 0	A	B	32	0
0 5	6 8	9 10	11 15	16 20	21 30	31

```
if (frA) is a NaN or
   (frB) is a NaN then      c ← 0b0001
else if (frA)< (frB) then   c ← 0b1000
else if (frA)> (frB) then   c ← 0b0100
else                        c ← 0b0010

FPCC ← c
CR[4 * crfD–4 * crfD + 3] ← c

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

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

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

Other registers altered:

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

  Affected: LT, GT, EQ, UN

- Floating-Point Status and Control Register:

  Affected: FPCC, FX, VXSNAN, VXVC

PowerPC Architecture Level	Supervisor Level	32-Bit	64-Bit	64-Bit Bridge	Optional	Form
UISA						X

fcmpu

Floating Compare Unordered (x'FC00 0000')

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

▨ Reserved

63	crfD	0 0	A	B	0 0 0 0 0 0 0 0 0 0	0
0 5	6 8	9 10	11 15	16 20	21 30	31

```
if (frA) is a NaN or
   (frB) is a NaN then      c ← 0b0001
else if (frA) < (frB) then  c ← 0b1000
else if (frA) > (frB) then  c ← 0b0100
else                        c ← 0b0010

FPCC ← c
CR[4 * crfD–4 * crfD + 3] ← c

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

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

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

Other registers altered:

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

  Affected: LT, GT, EQ, UN

- Floating-Point Status and Control Register:

  Affected: FPCC, FX, VXSNAN

PowerPC Architecture Level	Supervisor Level	32-Bit	64-Bit	64-Bit Bridge	Optional	Form
UISA						X

fctid*x* 64-Bit Implementations Only fctid*x*

Floating Convert to Integer Double Word (x'FC00 065C')

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

▨ Reserved

63	D	0 0000	B	814	Rc
0 5	6 10	11 15	16 20	21 30	31

The floating-point operand in **fr**B is converted to a 64-bit signed fixed-point integer, using the rounding mode specified by FPSCR[RN], and placed into **fr**D.

If the operand in **fr**B is greater than 2^{63}– 1, then **fr**D is set to 0x7FFF_FFFF_FFFF_FFFF. If the operand in **fr**B is less than $–2^{63}$, then **fr**D is set to 0x8000_0000_0000_0000.

Except for 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.

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

This instruction is defined only for 64-bit implementations. Using it on a 32-bit implementation will cause the system illegal instruction error handler to be invoked.

Other registers altered:

- Condition Register (CR1 field):

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

  Affected: FPRF (undefined), FR, FI, FX, XX, VXSNAN, VXCVI

PowerPC Architecture Level	Supervisor Level	32-Bit	64-Bit	64-Bit Bridge	Optional	Form
UISA			Ð			X

fctidz*x* **64-Bit Implementations Only** **fctidz***x*

Floating Convert to Integer Double Word with Round toward Zero (x’FC00 065E’)

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

▨ Reserved

63	D	0 0000	B	815	Rc
0 5	6 10	11 15	16 20	21 30	31

The floating-point operand in **fr**B is converted to a 64-bit signed fixed-point integer, using the rounding mode round toward zero, and placed into **fr**D.

If the operand in **fr**B is greater than $2^{63} - 1$, then **fr**D is set to 0x7FFF_FFFF_FFFF_FFFF. If the operand in **fr**B is less than -2^{63}, then **fr**D is set to 0x8000_0000_0000_0000.

Except for 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.

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

This instruction is defined only for 64-bit implementations. Using it on a 32-bit implementation will cause the system illegal instruction error handler to be invoked.

Other registers altered:

- Condition Register (CR1 field):

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

  Affected: FPRF (undefined), FR, FI, FX, XX, VXSNAN, VXCVI

PowerPC Architecture Level	Supervisor Level	32-Bit	64-Bit	64-Bit Bridge	Optional	Form
UISA			Đ			X

fctiw*x* fctiw*x*

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

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

▨ Reserved

0–5	6–10	11–15	16–20	21–30	31
63	D	0 0000	B	14	Rc

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

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

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

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

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

Other registers altered:

- Condition Register (CR1 field):

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

  Affected: FPRF (undefined), FR, FI, FX, XX, VXSNAN, VXCVI

PowerPC Architecture Level	Supervisor Level	32-Bit	64-Bit	64-Bit Bridge	Optional	Form
UISA						X

fctiwz*x* fctiwz*x*

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

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

▨ Reserved

63	D	0 0000	B	15	Rc
0 5	6 10	11 15	16 20	21 30	31

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

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

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

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

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

Other registers altered:

- Condition Register (CR1 field):

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

  Affected: FPRF (undefined), FR, FI, FX, XX, VXSNAN, VXCVI

PowerPC Architecture Level	Supervisor Level	32-Bit	64-Bit	64-Bit Bridge	Optional	Form
UISA						X

fdiv*x* fdiv*x*

Floating Divide (Double-Precision) (x’FC00 0024’)

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

[POWER mnemonics: **fd**, **fd.**]

▨ Reserved

63	D	A	**B**	**0 0 0 0 0** ▨	**18**	**Rc**
0 5	6 10	11 15	16 20	21 25	26 30	31

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

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

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

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

Other registers altered:

- Condition Register (CR1 field):

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

  Affected: FPRF, FR, FI, FX, OX, UX, ZX, XX, VXSNAN, VXIDI, VXZDZ

PowerPC Architecture Level	Supervisor Level	32-Bit	64-Bit	64-Bit Bridge	Optional	Form
UISA						A

fdivs*x* fdivs*x*

Floating Divide Single (x’EC00 0024’)

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

▨ Reserved

59	D	A	B	▨ 0 0 0 0 0	18	Rc
0 5	6 10	11 15	16 20	21 25	26 30	31

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

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

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

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

Other registers altered:

- Condition Register (CR1 field):

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

  Affected: FPRF, FR, FI, FX, OX, UX, ZX, XX, VXSNAN, VXIDI, VXZDZ

PowerPC Architecture Level	Supervisor Level	32-Bit	64-Bit	64-Bit Bridge	Optional	Form
UISA						A

fmadd*x* **fmadd*x***

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

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

[POWER mnemonics: **fma**, **fma.**]

63	D	A	B	C	29	Rc
0 5	6 10	11 15	16 20	21 25	26 30	31

The following operation is performed:

```
    frD ← (frA * frC) + frB
```

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

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

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

Other registers altered:

- Condition Register (CR1 field):

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

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

PowerPC Architecture Level	Supervisor Level	32-Bit	64-Bit	64-Bit Bridge	Optional	Form
UISA						A

fmadds*x* fmadds*x*

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

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

59	D	A	B	C	29	Rc
0 5	6 10	11 15	16 20	21 25	26 30	31

The following operation is performed:

```
frD ← (frA * frC) + frB
```

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

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

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

Other registers altered:

- Condition Register (CR1 field):

  Affected: FX, FEX, VX, OX(if Rc = 1)

- Floating-Point Status and Control Register:

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

PowerPC Architecture Level	Supervisor Level	32-Bit	64-Bit	64-Bit Bridge	Optional	Form
UISA						A

fmr*x* fmr*x*

Floating Move Register (Double-Precision) (x’FC00 0090’)

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

▨ Reserved

63	D	0 0000	B	72	Rc
0 5	6 10	11 15	16 20	21 30	31

The following operation is performed:

```
frD ← (frB)
```

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

Other registers altered:

- Condition Register (CR1 field):

  Affected: FX, FEX, VX, OX(if Rc = 1)

PowerPC Architecture Level	Supervisor Level	32-Bit	64-Bit	64-Bit Bridge	Optional	Form
UISA						X

fmsub*x* fmsub*x*

Floating Multiply-Subtract (Double-Precision) x’FC00 0038’)

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

[POWER mnemonics: **fms**, **fms.**]

63	D	A	B	C	28	Rc
0 5	6 10	11 15	16 20	21 25	26 30	31

The following operation is performed:

```
frD ← [frA * frC] - frB
```

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

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

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

Other registers altered:

- Condition Register (CR1 field):

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

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

PowerPC Architecture Level	Supervisor Level	32-Bit	64-Bit	64-Bit Bridge	Optional	Form
UISA						A

fmsubs*x* fmsubs*x*

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

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

59	D	A	B	C	28	Rc
0 5	6 10	11 15	16 20	21 25	26 30	31

The following operation is performed:

```
frD ← [frA * frC] - frB
```

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

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

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

Other registers altered:

- Condition Register (CR1 field):

  Affected: FX, FEX, VX, OX(if Rc = 1)

- Floating-Point Status and Control Register:

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

PowerPC Architecture Level	Supervisor Level	32-Bit	64-Bit	64-Bit Bridge	Optional	Form
UISA						A

fmul*x* **fmul***x*

Floating Multiply (Double-Precision) (x’FC00 0032’)

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

[POWER mnemonics: **fm**, **fm.**]

▨ Reserved

63	D	A	0 0 0 0 0	C	25	Rc
0 5	6 10	11 15	16 20	21 25	26 30	31

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

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

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

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

Other registers altered:

- Condition Register (CR1 field):

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

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

PowerPC Architecture Level	Supervisor Level	32-Bit	64-Bit	64-Bit Bridge	Optional	Form
UISA						A

fmuls*x* **fmuls***x*

Floating Multiply Single (x’EC00 0032’)

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

▨ Reserved

59	D	A	0 0 0 0 0	C	25	Rc
0 5	**6 10**	**11 15**	**16 20**	**21 25**	**26 30**	**31**

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

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

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

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

Other registers altered:

- Condition Register (CR1 field):

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

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

PowerPC Architecture Level	Supervisor Level	32-Bit	64-Bit	64-Bit Bridge	Optional	Form
UISA						A

fnabs*x* **fnabs***x*

Floating Negative Absolute Value (x’FC00 0110’)

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

▨ Reserved

63	D	0 0000	B	136	Rc
0 5	6 10	11 15	16 20	21 25	26 30 31

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

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

Other registers altered:

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

PowerPC Architecture Level	Supervisor Level	32-Bit	64-Bit	64-Bit Bridge	Optional	Form
UISA						X

fneg*x* **fneg***x*

Floating Negate (x’FC00 0050’)

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

▨ Reserved

63	D	0 0000	B	40	Rc
0 5	6 10	11 15	16 20	21 30	31

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

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

Other registers altered:

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

PowerPC Architecture Level	Supervisor Level	32-Bit	64-Bit	64-Bit Bridge	Optional	Form
UISA						X

fnmadd*x* **fnmadd***x*

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

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

[POWER mnemonics: **fnma**, **fnma.**]

63	D	A	B	C	31	Rc
0 5	6 10	11 15	16 20	21 25	26 30	31

The following operation is performed:

```
frD ← - ([frA * frC] + frB)
```

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

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

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

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

Other registers altered:

- Condition Register (CR1 field):

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

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

PowerPC Architecture Level	Supervisor Level	32-Bit	64-Bit	64-Bit Bridge	Optional	Form
UISA						A

fnmadds*x* **fnmadds***x*

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

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

59	D	A	B	C	31	Rc
0 5	6 10	11 15	16 20	21 25	26 30	31

The following operation is performed:

```
frD ← - ([frA * frC] + frB)
```

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

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

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

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

Other registers altered:

- Condition Register (CR1 field):

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

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

PowerPC Architecture Level	Supervisor Level	32-Bit	64-Bit	64-Bit Bridge	Optional	Form
UISA						A

fnmsub*x* fnmsub*x*

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

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

[POWER mnemonics: **fnms**, **fnms.**]

63	D	A	B	C	30	Rc
0 5	6 10	11 15	16 20	21 25	26 30	31

The following operation is performed:

```
frD ← - ([frA * frC] - frB)
```

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

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

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

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

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

Other registers altered:

- Condition Register (CR1 field)

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

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

PowerPC Architecture Level	Supervisor Level	32-Bit	64-Bit	64-Bit Bridge	Optional	Form
UISA						A

fnmsubs*x* fnmsubs*x*

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

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

59	D	A	B	C	30	Rc
0 5	6 10	11 15	16 20	21 25	26 30	31

The following operation is performed:

```
frD ← - ([frA * frC] - frB)
```

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

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

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

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

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

Other registers altered:

- Condition Register (CR1 field)

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

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

PowerPC Architecture Level	Supervisor Level	32-Bit	64-Bit	64-Bit Bridge	Optional	Form
UISA						A

fres*x* **fres***x*

Floating Reciprocal Estimate Single (x’EC00 0030’)

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

▨ Reserved

59	D	▨ 0 0000	B	▨ 000 00	24	Rc
0 5	6 10	11 15	16 20	21 25	26 30	31

A single-precision estimate of the reciprocal of the floating-point operand in register **fr**B is placed into register **fr**D. The estimate placed into register **fr**D is correct to a precision of one part in 256 of the reciprocal of **fr**B. That is,

$$\mathrm{ABS}\left(\frac{\mathrm{estimate}-\left(\frac{1}{x}\right)}{\left(\frac{1}{x}\right)}\right) \leq \frac{1}{256}$$

where x is the initial value in **fr**B. Note that the value placed into register **fr**D may vary between implementations, and between different executions on the same implementation.

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

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

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

** No result if FPSCR[VE] = 1

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

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

This instruction is optional in the PowerPC architecture.

Other registers altered:

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

PowerPC Architecture Level	Supervisor Level	32-Bit	64-Bit	64-Bit Bridge	Optional	Form
UISA					Đ	A

frsp*x* **frsp***x*

Floating Round to Single (x'FC00 0018')

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

▨ Reserved

63	D	0 0000	B	12	Rc
0 5	6 10	11 15	16 20	21 30	31

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

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

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

Other registers altered:

- Condition Register (CR1 field):

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

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

PowerPC Architecture Level	Supervisor Level	32-Bit	64-Bit	64-Bit Bridge	Optional	Form
UISA						X

frsqrte*x* **frsqrte***x*

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

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

▨ Reserved

63	D	0 0000	B	000 00	26	Rc
0 5	6 10	11 15	16 20	21 25	26 30	31

A double-precision estimate of the reciprocal of the square root of the floating-point operand in register **fr**B is placed into register **fr**D. The estimate placed into register **fr**D is correct to a precision of one part in 32 of the reciprocal of the square root of **fr**B. That is,

$$\mathrm{ABS}\left(\frac{\text{estimate}-\left(\frac{1}{\sqrt{x}}\right)}{\left(\frac{1}{\sqrt{x}}\right)}\right) \le \frac{1}{32}$$

where x is the initial value in **fr**B. Note that the value placed into register **fr**D may vary between implementations, and between different executions on the same implementation.

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

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

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

** No result if FPSCR[VE] = 1

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

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

This instruction is optional in the PowerPC architecture.

Other registers altered:

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

PowerPC Architecture Level	Supervisor Level	32-Bit	64-Bit	64-Bit Bridge	Optional	Form
UISA					Ð	A

fsel*x* **fsel***x*

Floating Select (x’FC00 002E’)

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

63	D	A	B	C	23	Rc
0 5	6 10	11 15	16 20	21 25	26 30	31

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

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

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

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

This instruction is optional in the PowerPC architecture.

Other registers altered:

- Condition Register (CR1 field):

  Affected: FX, FEX, VX, OX(if Rc = 1)

PowerPC Architecture Level	Supervisor Level	32-Bit	64-Bit	64-Bit Bridge	Optional	Form
UISA					Ð	A

fsqrt*x* fsqrt*x*

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

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

▨ Reserved

0–5	6–10	11–15	16–20	21–25	26–30	31
63	D	0 0000 ▨	B	000 00 ▨	22	Rc

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

If the most-significant bit of the resultant significand is not 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 register **fr**D.

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

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

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

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

This instruction is optional in the PowerPC architecture.

Other registers altered:

- Condition Register (CR1 field):

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

  Affected: FPRF, FR, FI, FX, XX, VXSNAN, VXSQRT

PowerPC Architecture Level	Supervisor Level	32-Bit	64-Bit	64-Bit Bridge	Optional	Form
UISA					Đ	A

fsqrts*x* fsqrts*x*

Floating Square Root Single (x’EC00 002C’)

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

▨ Reserved

59	D	0 0000	B	000 00	22	Rc
0 5	6 10	11 15	16 20	21 25	26 30	31

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

If the most-significant bit of the resultant significand is not 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 register **fr**D.

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

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

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

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

This instruction is optional in the PowerPC architecture.

Other registers altered:

- Condition Register (CR1 field):

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

  Affected: FPRF, FR, FI, FX, XX, VXSNAN, VXSQRT

PowerPC Architecture Level	Supervisor Level	32-Bit	64-Bit	64-Bit Bridge	Optional	Form
UISA					Ð	A

fsub*x*

fsub*x*

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

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

[POWER mnemonics: **fs**, **fs.**]

▨ Reserved

0–5	6–10	11–15	16–20	21–25	26–30	31
63	D	A	B	0 0 0 0 0	20	Rc

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

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

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

Other registers altered:

- Condition Register (CR1 field):

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

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

PowerPC Architecture Level	Supervisor Level	32-Bit	64-Bit	64-Bit Bridge	Optional	Form
UISA						A

fsubs*x* **fsubs***x*

Floating Subtract Single (x’EC00 0028’)

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

▨ Reserved

59	D	A	B	000 00 ▨	20	Rc
0 5	6 10	11 15	16 20	21 25	26 30	31

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

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

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

Other registers altered:

- Condition Register (CR1 field):

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

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

PowerPC Architecture Level	Supervisor Level	32-Bit	64-Bit	64-Bit Bridge	Optional	Form
UISA						A

icbi icbi

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

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

▨ Reserved

0–5	6–10	11–15	16–20	21–30	31
31	0 0 0 0 0	A	B	982	0

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

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

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

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

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

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

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

Other registers altered:

- None

PowerPC Architecture Level	Supervisor Level	32-Bit	64-Bit	64-Bit Bridge	Optional	Form
VEA						X

isync isync

Instruction Synchronize (x’4C00 012C’)

isync

[POWER mnemonic: **ics**]

▨ Reserved

0–5	6–10	11–15	16–20	21–30	31
19	00 000	0 0000	0000 0	150	0

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

This instruction is context synchronizing.

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

Note that all exceptions and the **rfi** and **rfid** instructions are also context synchronizing.

Other registers altered:

- None

PowerPC Architecture Level	Supervisor Level	32-Bit	64-Bit	64-Bit Bridge	Optional	Form
VEA						XL

lbz lbz

Load Byte and Zero (x’8800 0000’)

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

34	D	A	d
0 5	6 10	11 15	16 31

```
if rA = 0 then b ← 0
else      b ← (rA)
EA ← b + EXTS(d)
rD ← (5624)0 || MEM(EA, 1)
```

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

Other registers altered:

- None

PowerPC Architecture Level	Supervisor Level	32-Bit	64-Bit	64-Bit Bridge	Optional	Form
UISA						D

lbzu   lbzu

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

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

35	D	A	d
0 … 5	6 … 10	11 … 15	16 … 31

```
EA ← (rA) + EXTS(d)
rD← (5624)0 || MEM(EA, 1)
rA← EA
```

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

EA is placed into **r**A.

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

Other registers altered:

- None

PowerPC Architecture Level	Supervisor Level	32-Bit	64-Bit	64-Bit Bridge	Optional	Form
UISA						D

lbzux

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

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

▨ Reserved

31	D	A	B	119	▨ 0
0–5	6–10	11–15	16–20	21–30	31

```
EA ← (rA) + (rB)
rD ← (5624)0 || MEM(EA, 1)
rA ← EA
```

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

EA is placed into **r**A.

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

Other registers altered:

- None

PowerPC Architecture Level	Supervisor Level	32-Bit	64-Bit	64-Bit Bridge	Optional	Form
UISA						X

lbzx lbzx

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

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

▨ Reserved

31	D	A	B	87	▨ 0
0 5	6 10	11 15	16 20	21 30	31

```
if rA = 0 then b ← 0
else          b ← (rA)
EA ← b + (rB)
rD ← (5624)0 || MEM(EA, 1)
```

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

Other registers altered:

- None

PowerPC Architecture Level	Supervisor Level	32-Bit	64-Bit	64-Bit Bridge	Optional	Form
UISA						X

ld 64-Bit Implementations Only ld

Load Double Word (x’E800 0000’)

ld **r**D,ds(**r**A)

58	D	A	ds	0 0
0 5	6 10	11 15	16 29	30 31

```
if rA = 0 then b ← 0
else      b ← (rA)
EA ← b + EXTS(ds || 0b00)
rD ← MEM(EA, 8)
```

EA is the sum (**r**A|0) + (ds || 0b00). The double word in memory addressed by EA is loaded into **r**D.

This instruction is defined only for 64-bit implementations. Using it on a 32-bit implementation will cause the system illegal instruction error handler to be invoked.

Other registers altered:

- None

PowerPC Architecture Level	Supervisor Level	32-Bit	64-Bit	64-Bit Bridge	Optional	Form
UISA			Ð			DS

ldarx 64-Bit Implementations Only ldarx

Load Double Word and Reserve Indexed (x’7C00 00A8’)

ldarx **r**D,**r**A,**r**B

▨ Reserved

31	D	A	B	84	▨ 0
0 5	6 10	11 15	16 20	21 30	31

```
if rA = 0 then b ← 0
else      b ← (rA)
EA ← b + (rB)
RESERVE ← 1
RESERVE_ADDR ← physical_addr(EA)
rD ← MEM(EA, 8)
```

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

This instruction creates a reservation for use by a Store Double Word Conditional Indexed (**stdcx.**) instruction. An address computed from the EA is associated with the reservation, and replaces any address previously associated with the reservation.

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

This instruction is defined only for 64-bit implementations. Using it on a 32-bit implementation will cause the system illegal instruction error handler to be invoked.

Other registers altered:

- None

PowerPC Architecture Level	Supervisor Level	32-Bit	64-Bit	64-Bit Bridge	Optional	Form
UISA			Ð			X

ldu 64-Bit Implementations Only ldu

Load Double Word with Update (x’E800 0001’)

ldu **r**D,ds(**r**A)

58	D	A	ds	0 1
0 5	6 10	11 15	16 29	30 31

```
EA ← (rA) + EXTS(ds || 0b00)
rD ← MEM(EA, 8)
rA ← EA
```

EA is the sum (**r**A) + (ds || 0b00). The double 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.

This instruction is defined only for 64-bit implementations. Using it on a 32-bit implementation will cause the system illegal instruction error handler to be invoked.

Other registers altered:

- None

PowerPC Architecture Level	Supervisor Level	32-Bit	64-Bit	64-Bit Bridge	Optional	Form
UISA			Ð			DS

ldux 64-Bit Implementations Only ldux*x*

Load Double Word with Update Indexed (x’7C00 006A’)

ldux **r**D,**r**A,**r**B

▨ Reserved

0–5	6–10	11–15	16–20	21–30	31
31	D	A	B	53	0

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

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

This instruction is defined only for 64-bit implementations. Using it on a 32-bit implementation will cause the system illegal instruction to be invoked.

Other registers altered:

- None

PowerPC Architecture Level	Supervisor Level	32-Bit	64-Bit	64-Bit Bridge	Optional	Form
UISA			Ð			X

ldx 64-Bit Implementations Only ldx

Load Double Word Indexed (x’7C00 002A’)

ldx **r**D,**r**A,**r**B

▨ Reserved

31	D	A	B	21	▨ 0
0 5	6 10	11 15	16 20	21 30	31

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

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

This instruction is defined only for 64-bit implementations. Using it on a 32-bit implementation will cause the system illegal instruction error handler to be invoked.

Other registers altered:

- None

PowerPC Architecture Level	Supervisor Level	32-Bit	64-Bit	64-Bit Bridge	Optional	Form
UISA			Ð			X

lfd lfd

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

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

50	D	A	d
0–5	6–10	11–15	16–31

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

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

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

Other registers altered:

- None

PowerPC Architecture Level	Supervisor Level	32-Bit	64-Bit	64-Bit Bridge	Optional	Form
UISA						D

lfdu lfdu

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

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

51	D	A	d
0 5	6 10	11 15	16 31

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

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

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

EA is placed into **r**A.

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

Other registers altered:

- None

PowerPC Architecture Level	Supervisor Level	32-Bit	64-Bit	64-Bit Bridge	Optional	Form
UISA						D

lfdux lfdux

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

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

▨ Reserved

31	D	A	B	631	0 ▨
0 5	6 10	11 15	16 20	21 30	31

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

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

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

EA is placed into **r**A.

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

Other registers altered:

- None

PowerPC Architecture Level	Supervisor Level	32-Bit	64-Bit	64-Bit Bridge	Optional	Form
UISA						X

lfdx

lfdx

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

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

▨ Reserved

31	D	A	B	599	▨ 0
0 – 5	6 – 10	11 – 15	16 – 20	21 – 30	31

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

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

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

Other registers altered:

- None

PowerPC Architecture Level	Supervisor Level	32-Bit	64-Bit	64-Bit Bridge	Optional	Form
UISA						X

lfs

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

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

48	D	A	d
0 5	6 10	11 15	16 31

```
if rA = 0 then b ← 0
else      b ← (rA)
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 Section D.6 , “Floating-Point Load Instructions”) and placed into **fr**D.

Other registers altered:

- None

PowerPC Architecture Level	Supervisor Level	32-Bit	64-Bit	64-Bit Bridge	Optional	Form
UISA						D

lfsu lfsu

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

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

49	D	A	d
0 5	6 10	11 15	16 31

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

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

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

EA is placed into **r**A.

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

Other registers altered:

- None

PowerPC Architecture Level	Supervisor Level	32-Bit	64-Bit	64-Bit Bridge	Optional	Form
UISA						D

lfsux

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

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

▨ Reserved

31	D	A	B	567	0
0 5	6 10	11 15	16 20	21 30	31

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

EA is the sum (**r**A) + (**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 Section D.6 , “Floating-Point Load Instructions”) and placed into **fr**D.

EA is placed into **r**A.

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

Other registers altered:

- None

PowerPC Architecture Level	Supervisor Level	32-Bit	64-Bit	64-Bit Bridge	Optional	Form
UISA						X

lfsx　　　　　　　　　　　　　　　　　　　　　　　　　　　　lfsx

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

lfsx　　　　　　　　　　　　**fr**D,**r**A,**r**B

▨ Reserved

31	D	A	B	535	0 ▨
0　5	6　10	11　15	16　20	21　30	31

```
if rA = 0 then b ← 0
else       b ← (rA)
EA ← b + (rB)
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 Section D.6 , “Floating-Point Load Instructions”) and placed into **fr**D.

Other registers altered:

- None

PowerPC Architecture Level	Supervisor Level	32-Bit	64-Bit	64-Bit Bridge	Optional	Form
UISA						X

lha lha

Load Half Word Algebraic (x’A800 0000’)

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

42	D	A	d
0 5	6 10	11 15	16 31

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

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

Other registers altered:

- None

PowerPC Architecture Level	Supervisor Level	32-Bit	64-Bit	64-Bit Bridge	Optional	Form
UISA						D

lhau

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

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

43	D	A	d
0 5	6 10	11 15	16 31

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

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

EA is placed into **r**A.

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

Other registers altered:

- None

PowerPC Architecture Level	Supervisor Level	32-Bit	64-Bit	64-Bit Bridge	Optional	Form
UISA						D

lhaux lhaux

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

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

▨ Reserved

31	D	A	B	375	0
0 5	6 10	11 15	16 20	21 30	31

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

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

EA is placed into **r**A.

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

Other registers altered:

- None

PowerPC Architecture Level	Supervisor Level	32-Bit	64-Bit	64-Bit Bridge	Optional	Form
UISA						X

lhax lhax

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

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

▨ Reserved

0–5	6–10	11–15	16–20	21–30	31
31	D	A	B	343	0

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

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

Other registers altered:

- None

PowerPC Architecture Level	Supervisor Level	32-Bit	64-Bit	64-Bit Bridge	Optional	Form
UISA						X

lhbrx   lhbrx

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

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

▨ Reserved

31	D	A	B	790	▨ 0
0 5	6 10	11 15	16 20	21 30	31

```
if rA = 0 then b ← 0
else      b ← (rA)
EA ← b + (rB)
rD ← (4816)0 || MEM(EA + 1, 1) || MEM(EA, 1)
```

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

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

Other registers altered:

- None

PowerPC Architecture Level	Supervisor Level	32-Bit	64-Bit	64-Bit Bridge	Optional	Form
UISA						X

lhz lhz

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

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

40	D	A	d
0 5	6 10	11 15	16 31

```
if rA = 0 then b ← 0
else     b ← (rA)
EA ← b + EXTS(d)
rD ← (4816)0 || MEM(EA, 2)
```

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

Other registers altered:

- None

PowerPC Architecture Level	Supervisor Level	32-Bit	64-Bit	64-Bit Bridge	Optional	Form
UISA						D

lhzu

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

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

41	D	A	d
0 5	6 10	11 15	16 31

```
EA ← rA + EXTS(d)
rD ← (4816)0 || MEM(EA, 2)
rA ← EA
```

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

EA is placed into **r**A.

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

Other registers altered:

- None

PowerPC Architecture Level	Supervisor Level	32-Bit	64-Bit	64-Bit Bridge	Optional	Form
UISA						D

lhzux lhzux

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

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

▨ Reserved

31	D	A	B	311	0
0 5	6 10	11 15	16 20	21 30	31

```
EA ← (rA) + (rB)
rD← (4816)0 || MEM(EA, 2)
rA← EA
```

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

EA is placed into **r**A.

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

Other registers altered:

- None

PowerPC Architecture Level	Supervisor Level	32-Bit	64-Bit	64-Bit Bridge	Optional	Form
UISA						X

lhzx lhzx

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

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

▨ Reserved

31	D	A	B	279	0 ▨
0 5	6 10	11 15	16 20	21 30	31

```
if rA = 0 then b ← 0
else      b ← (rA)
EA ← b + (rB)
rD ← (48)0 || MEM(EA, 2)
```

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

Other registers altered:

- None

PowerPC Architecture Level	Supervisor Level	32-Bit	64-Bit	64-Bit Bridge	Optional	Form
UISA						X

lmw lmw

Load Multiple Word (x'B800 0000')

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

[POWER mnemonic: **lm**]

46	D	A	d
0 5	6 10	11 15	16 31

```
if rA = 0 then b← 0
else     b← (rA)
EA← b + EXTS(d)
r← rD
do while r ð 31
   GPR(r) ← (32)0 || 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 low-order 32 bits of GPRs **r**D through **r31**. The high-order 32 bits of these GPRs are cleared.

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

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

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

Other registers altered:

- None

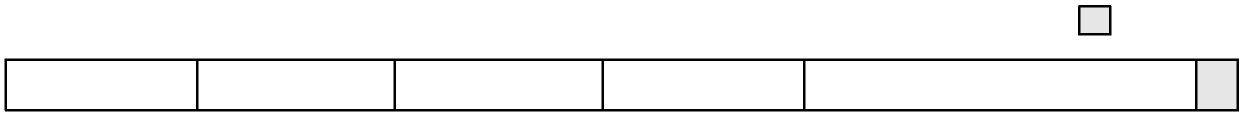
PowerPC Architecture Level	Supervisor Level	32-Bit	64-Bit	64-Bit Bridge	Optional	Form
UISA						D

lswi lswi

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

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

[POWER mnemonic: **lsi**]



```
if rA = 0 then EA ← 0
else EA ← (rA)
if NB = 0 then n ← 32
elsen ← NB
r ← rD - 1
i ← 320
do while n > 0
   if i = 32 then
     r ← r + 1 (mod 32)
     GPR(r) ← 0
   GPR(r)[i–i + 7] ← MEM(EA, 1)
   i ← i + 8
   if i = 6432 then i ← 320
   EA ← EA + 1
   n ← n - 1
```

EA is (**r**A | 0).

Let *n* = NB if NB ¦ 0, *n* = 32 if NB = 0; *n* is the number of bytes to load.
Let *nr* = CEIL(*n* ÷ 4); *nr* is the number of registers to be loaded with data.

n consecutive bytes starting at EA are loaded into GPRs **r**D through **r**D + *nr* – 1. Data is loaded into the low-order four bytes of each GPR; the high-order four bytes are cleared.

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

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

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

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

Other registers altered:

- None

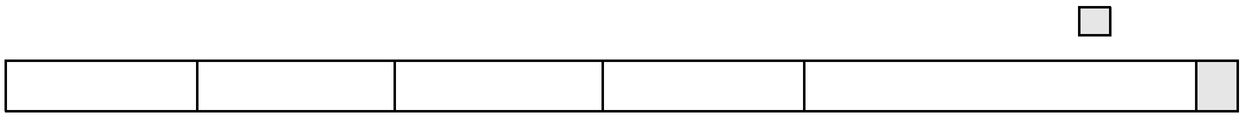
PowerPC Architecture Level	Supervisor Level	32-Bit	64-Bit	64-Bit Bridge	Optional	Form
UISA						X

lswx lswx

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

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

[POWER mnemonic: **lsx**]



```
if rA = 0 then b ← 0
else     b ← (rA)
EA ← b + (rB)
n ← XER[25-31]
r ← rD - 1
i ← 32
rD ← undefined
 do while n > 0
  if i = 32 then
    r ← r + 1 (mod 32)
    GPR(r) ← 0
  GPR(r)[i-i + 7] ← MEM(EA, 1)
  i ← i + 8
  if i = 6432 then i ← 320
  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. Data is loaded into the low-order four bytes of each GPR; the high-order four bytes are cleared.

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

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

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

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

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

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

Other registers altered:

- None

PowerPC Architecture Level	Supervisor Level	32-Bit	64-Bit	64-Bit Bridge	Optional	Form
UISA						X

lwa 64-Bit Implementations Only lwa

Load Word Algebraic (x’E800 0002’)

lwa **r**D,ds(**r**A)

58	D	A	ds	1 0
0 5	6 10	11 15	16 29	30 31

```
if rA = 0 then b ← 0
else     b ← (rA)
EA ← b + EXTS(ds || 0b00)
rD ← EXTS(MEM(EA, 4))
```

EA is the sum (**r**A|0) + (ds || 0b00). The word in memory addressed by EA is loaded into the low-order 32 bits of **r**D. The contents of the high-order 32 bits of **r**D are filled with a copy of bit 0 of the loaded word.

This instruction is defined only for 64-bit implementations. Using it on a 32-bit implementation will cause the system illegal instruction error handler to be invoked.

Other registers altered:

- None

PowerPC Architecture Level	Supervisor Level	32-Bit	64-Bit	64-Bit Bridge	Optional	Form
UISA			Ð			DS

lwarx lwarx

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

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

▨ Reserved

31	D	A	B	20	▨ 0
0 5	6 10	11 15	16 20	21 30	31

```
if rA = 0 then b ← 0
else      b ← (rA)
EA ← b + (rB)
RESERVE ← 1
RESERVE_ADDR ← physical_addr(EA)
rD ← (32)0 || MEM(EA,4)
```

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

The word in memory addressed by EA is loaded into the low-order 32 bits of **r**D. The contents of the high-order 32 bits of **r**D are cleared.

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

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

When the RESERVE bit is set, the processor enables hardware snooping for the block of memory addressed by the RESERVE address. If the processor detects that another processor writes to the block of memory it has reserved, it clears the RESERVE bit. The **stwcx.** instruction will only do a store if the RESERVE bit is set. The **stwcx.** instruction sets the CR0[EQ] bit if the store was successful and clears it if it failed. The **lwarx** and **stwcx.** combination can be used for atomic read-modify-write sequences. Note that the atomic sequence is not guaranteed, but its failure can be detected if CR0[EQ] = 0 after the **stwcx.** instruction.

Other registers altered:

- None

PowerPC Architecture Level	Supervisor Level	32-Bit	64-Bit	64-Bit Bridge	Optional	Form
UISA						X

lwaux 64-Bit Implementations Only lwaux

Load Word Algebraic with Update Indexed (x’7C00 02EA’)

lwaux **r**D,**r**A,**r**B

▨ Reserved

31	D	A	B	373	0
0 – 5	6 – 10	11 – 15	16 – 20	21 – 30	31

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

EA is the sum (**r**A) + (**r**B). The word in memory addressed by EA is loaded into the low-order 32 bits of **r**D. The high-order 32 bits of **r**D are filled with a copy of bit 0 of the loaded word.

EA is placed into **r**A.

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

This instruction is defined only for 64-bit implementations. Using it on a 32-bit implementation will cause the system illegal instruction error handler to be invoked.

Other registers altered:

- None

PowerPC Architecture Level	Supervisor Level	32-Bit	64-Bit	64-Bit Bridge	Optional	Form
UISA			Ð			X

lwax 64-Bit Implementations Only lwax

Load Word Algebraic Indexed (x’7C00 02AA’)

lwax **r**D,**r**A,**r**B

▨ Reserved

31	D	A	B	341	0
0–5	6–10	11–15	16–20	21–30	31

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

EA is the sum (**r**A|0) + (**r**B). The word in memory addressed by EA is loaded into the low-order 32 bits of **r**D. The high-order 32 bits of **r**D are filled with a copy of bit 0 of the loaded word.

This instruction is defined only for 64-bit implementations. Using it on a 32-bit implementation will cause the system illegal instruction error handler to be invoked.

Other registers altered:

- None

PowerPC Architecture Level	Supervisor Level	32-Bit	64-Bit	64-Bit Bridge	Optional	Form
UISA			Đ			X

lwbrx   lwbrx

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

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

[POWER mnemonic: **lbrx**]

▨ Reserved

31	D	A	B	534	▨ 0
0 5	6 10	11 15	16 20	21 30	31

```
if rA = 0 then b ← 0
else     b ← (rA)
EA ← b + (rB)
rD ← (32)0 || MEM(EA + 3, 1) || MEM(EA + 2, 1) || MEM(EA + 1, 1) || MEM(EA, 1)
```

EA is the sum (**r**A|0) + **r**B. Bits 0–7 of the word in memory addressed by EA are loaded into the low-order 8 bits of **r**D. Bits 8–15 of the word in memory addressed by EA are loaded into the subsequent low-order 8 bits of **r**D. Bits 16–23 of the word in memory addressed by EA are loaded into the subsequent low-order eight bits of **r**D. Bits 24–31 of the word in memory addressed by EA are loaded into the subsequent low-order 8 bits of **r**D. The high-order 32 bits of **r**D are cleared.

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

Other registers altered:

- None

PowerPC Architecture Level	Supervisor Level	32-Bit	64-Bit	64-Bit Bridge	Optional	Form
UISA						X

lwz lwz

Load Word and Zero (x’8000 0000’)

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

[POWER mnemonic: **l**]

32	D	A	d
0 5	6 10	11 15	16 31

```
if rA = 0 then b ← 0
else     b ← (rA)
EA ← b + EXTS(d)
rD ← (32)0 || MEM(EA, 4)
```

EA is the sum (**r**A|0) + d. The word in memory addressed by EA is loaded into the low-order 32 bits of **r**D. The high-order 32 bits of **r**D are cleared.

Other registers altered:

- None

PowerPC Architecture Level	Supervisor Level	32-Bit	64-Bit	64-Bit Bridge	Optional	Form
UISA						D

lwzu lwzu

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

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

[POWER mnemonic: **lu**]

33	D	A	d
0 5	6 10	11 15	16 31

```
EA ← rA + EXTS(d)
rD ← (32)0 || MEM(EA, 4)
rA ← EA
```

EA is the sum (**r**A) + d. The word in memory addressed by EA is loaded into the low-order 32 bits of **r**D. The high-order 32 bits of **r**D are cleared.

EA is placed into **r**A.

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

Other registers altered:

- None

PowerPC Architecture Level	Supervisor Level	32-Bit	64-Bit	64-Bit Bridge	Optional	Form
UISA						D

lwzux

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

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

[POWER mnemonic: **lux**]

▨ Reserved

31	D	A	B	55	0
0 5	6 10	11 15	16 20	21 30	31

```
EA ← (rA) + (rB)
rD ← (32)0 || MEM(EA, 4)
rA ← EA
```

EA is the sum (**r**A) + (**r**B). The word in memory addressed by EA is loaded into the low-order 32 bits of **r**D. The high-order 32 bits of **r**D are cleared.

EA is placed into **r**A.

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

Other registers altered:

- None

PowerPC Architecture Level	Supervisor Level	32-Bit	64-Bit	64-Bit Bridge	Optional	Form
UISA						X

lwzx

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

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

[POWER mnemonic: **lx**]

▨ Reserved

31	D	A	B	23	0
0 5	6 10	11 15	16 20	21 30	31

```
if rA = 0 then b ← 0
else      b ← (rA)
EA ← b + rB
rD ← (32)0 || MEM(EA, 4)
```

EA is the sum (**r**A|0) + (**r**B). The word in memory addressed by EA is loaded into the low-order 32 bits of **r**D. The high-order 32 bits of **r**D are cleared.

Other registers altered:

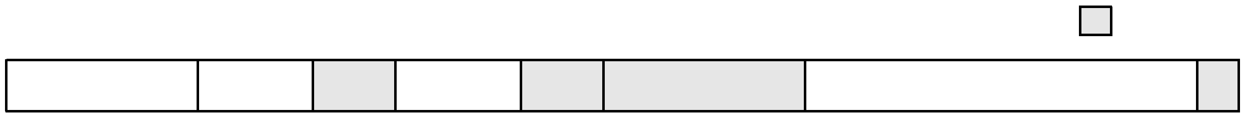
- None

PowerPC Architecture Level	Supervisor Level	32-Bit	64-Bit	64-Bit Bridge	Optional	Form
UISA						X

mcrf

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

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



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

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

Other registers altered:

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

  Affected: LT, GT, EQ, SO

PowerPC Architecture Level	Supervisor Level	32-Bit	64-Bit	64-Bit Bridge	Optional	Form
UISA						XL

mcrfs mcrfs

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

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

▨ Reserved

63	crfD	0 0	crfS	0 0	0 0 0 0 0	64	0
0 5	6 8	9 10	11 13	14 15	16 20	21 30	31

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

Other registers altered:

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

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

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

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

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

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

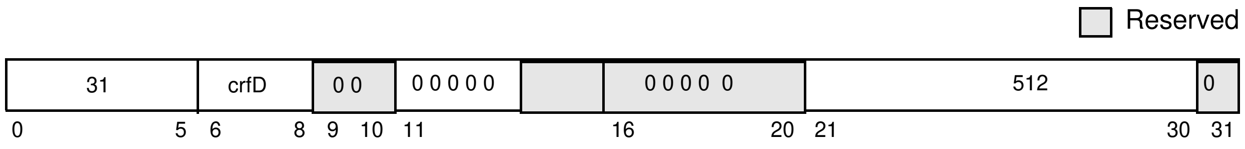
  Affected: VXSOFT, VXSQRT, VXCVI (if **crf**S = 5)

PowerPC Architecture Level	Supervisor Level	32-Bit	64-Bit	64-Bit Bridge	Optional	Form
UISA						X

mcrxr mcrxr

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

mcrxr **crf**D



CR[* **crf**D-4 * **crf**D +3]

The contents of XER[0-3] are copied into the condition register field designated by **crf**D.

All other fields of the condition register remain unchanged. XER[0-3] is cleared.

Other registers altered:

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

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

PowerPC Architecture Level	Supervisor Level	32-Bit	64-Bit	64-Bit Bridge	Optional	Form
UISA						X

mfcr mfcr

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

mfcr **r**D

▨ Reserved

31	D	0 0 0 0 0	0 0 0 0 0	19	0
0 5	6 10	11 15	16 20	21 30	31

```
rD ← (32)0 || CR
```

The contents of the condition register (CR) are placed into the low-order 32 bits of **r**D. The high-order 32 bits of **r**D are cleared.

Other registers altered:

- None

PowerPC Architecture Level	Supervisor Level	32-Bit	64-Bit	64-Bit Bridge	Optional	Form
UISA						X

mffs*x* mffs*x*

Move from FPSCR (x’FC00 048E’)

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

▨ Reserved

63	D	0 0 0 0 0	0 0 0 0 0	583	Rc
0 5	6 10	11 15	16 20	21 30	31

```
frD[32-63] ←FPSCR
```

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

Other registers altered:

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

PowerPC Architecture Level	Supervisor Level	32-Bit	64-Bit	64-Bit Bridge	Optional	Form
UISA						X

mfmsr

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

mfmsr **r**D

░ Reserved

31	D	0 0000	0000 0	83	0
0 5	6 10	11 15	16 20	21 30	31

```
rD ← MSR
```

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

This is a supervisor-level instruction.

Other registers altered:

- None

PowerPC Architecture Level	Supervisor Level	32-Bit	64-Bit	64-Bit Bridge	Optional	Form
OEA	Ð					X

mfspr

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

mfspr	**r**D,SPR

▨ Reserved

31	D	spr*	339	0
0 – 5	6 – 10	11 – 20	21 – 30	31

*Note: This is a split field.

```
n ← spr[5–9] || spr[0–4]
if length (SPR(n)) = 64 then
  rD ← SPR(n)
else
  rD ← (32)0 || SPR(n)
```

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

For special-purpose registers that are 32 bits long, the low-order 32 bits of **r**D receive the contents of the special-purpose register and the high-order 32 bits of **r**D are cleared.

Table 8-9. PowerPC UISA SPR Encodings for mfspr

SPR**			Register Name
Decimal	spr[5–9]	spr[0–4]	
1	00000	00001	XER
8	00000	01000	LR
9	00000	01001	CTR
Note: ** The order of the two 5-bit halves of the SPR number is reversed compared with the actual instruction coding.			

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

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

Other registers altered:

- None

Simplified mnemonics:

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

In the PowerPC OEA, the SPR field denotes a special-purpose register, encoded as shown in Table 8-10. . The contents of the designated SPR are placed into **r**D. For SPRs that are 32 bits long, the low-order 32 bits of **r**D receive the contents of the SPR and the high-order 32 bits of **r**D are cleared.

SPR[0] = 1 if and only if reading the register is supervisor-level. Execution of this instruction specifying a defined and supervisor-level register when MSR[PR] = 1 will result in a privileged instruction type program exception.

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

Other registers altered:

- None

Table 8-10. PowerPC OEA SPR Encodings for mfspr

SPR[1]			Register Name	Access
Decimal	spr[5–9]	spr[0–4]		
1	00000	00001	XER	User
8	00000	01000	LR	User
9	00000	01001	CTR	User
18	00000	10010	DSISR	Supervisor
19	00000	10011	DAR	Supervisor
22	00000	10110	DEC	Supervisor
25	00000	11001	SDR1	Supervisor
26	00000	11010	SRR0	Supervisor
27	00000	11011	SRR1	Supervisor
272	01000	10000	SPRG0	Supervisor
273	01000	10001	SPRG1	Supervisor
274	01000	10010	SPRG2	Supervisor
275	01000	10011	SPRG3	Supervisor
280	01000	11000	ASR[2]	Supervisor
282	01000	11010	EAR	Supervisor
287	01000	11111	PVR	Supervisor
528	10000	10000	IBAT0U	Supervisor
529	10000	10001	IBAT0L	Supervisor
530	10000	10010	IBAT1U	Supervisor
531	10000	10011	IBAT1L	Supervisor
532	10000	10100	IBAT2U	Supervisor
533	10000	10101	IBAT2L	Supervisor
534	10000	10110	IBAT3U	Supervisor

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

SPR[1]			Register Name	Access
Decimal	spr[5–9]	spr[0–4]		
535	10000	10111	IBAT3L	Supervisor
536	10000	11000	DBAT0U	Supervisor
537	10000	11001	DBAT0L	Supervisor
538	10000	11010	DBAT1U	Supervisor
539	10000	11011	DBAT1L	Supervisor
540	10000	11100	DBAT2U	Supervisor
541	10000	11101	DBAT2L	Supervisor
542	10000	11110	DBAT3U	Supervisor
543	10000	11111	DBAT3L	Supervisor
1013	11111	10101	DABR	Supervisor

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

For **mtspr** and **mfspr** instructions, the SPR number coded in assembly language does not appear directly as a 10-bit binary number in the instruction. The number coded is split into two 5-bit halves that are reversed in the instruction, with the high-order five bits appearing in bits 16–20 of the instruction and the low-order five bits in bits 11–15.
[2]64-bit implementations only.

PowerPC Architecture Level	Supervisor Level	32-Bit	64-Bit	64-Bit Bridge	Optional	Form
UISA/OEA	Ð*					XFX

mfsr mfsr

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

mfsr **r**D,SR

▨ Reserved

31	D	0	SR	0 0 0 0 0	595	0
0 5	6 10	11	12 15	16 20	21 30	31

```
rD ← SEGREG(SR)
```

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

This is a supervisor-level instruction.

This instruction is defined only for 32-bit implementations; using it on a 64-bit implementation causes an illegal instruction type program exception.

Other registers altered:

- None

TEMPORARY 64-BIT BRIDGE

```
rD ← SLB(SR)
```

The contents of the SLB entry selected by SR are placed into **r**D; the contents of rD correspond to a segment table entry containing values as shown in *Table 8-11*.

Table 8-11. GPR Content Format Following mfsr

SLB Double Word	Bit(s)	Contents	Description
0	0–31	0x0000_0000	ESID[0–31]
	32–35	SR	ESID[32–35]
	57–59	**r**D[32–34]	T, Ks, Kp
	60–61	**r**D[35–36]	N, reserved bit, or b0
1	0–24	**r**D[7–31]	VSID[0–24] or reserved
	25–51	**r**D[37–63]	VSID[25–51], or b1, CNTLR_SPEC
None	—	**r**D[0–6]	0b0000_000

If the SLB entry selected by SR was not created by an **mtsr**, **mtsrd**, or **mtsrdin** instruction, the contents of **r**D are undefined.

This is a supervisor-level instruction.

Other registers altered:

- None

PowerPC Architecture Level	Supervisor Level	32-Bit	64-Bit	64-Bit Bridge	Optional	Form
OEA	Ð	Ð		Ð		X

mfsrin mfsrin

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

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

▨ Reserved

31	D	0 0000	B	659	0
0 5	6 10	11 15	16 20	21 30	31

```
rD ← SEGREG(rB[0–3])
```

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

This is a supervisor-level instruction.

This instruction is defined only for 32-bit implementations. Using it on a 64-bit implementation causes an illegal instruction type program exception.

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

Other registers altered:

- None

TEMPORARY 64-BIT BRIDGE

```
rD ← SLB(rB[32-35])
```

The contents of the SLB entry selected by rB[32–35] are placed into **r**D; the contents of rD correspond to a segment table entry containing values as shown in *Table 8-12*

Table 8-12. GPR Content Format Following mfsrin

Doubleword	Bit(s)	Contents	Description
0	0-31	0x0000_0000	ESID[0–31]
	32-35	**r**B[32–35]	ESID[32–35]
	57-59	**r**D[32–34]	T, Ks, Kp
	60-61	**r**D[35–36]	N, reserved bit, or b0
1	0-24	**r**D[7–31]	VSID[0–24] or reserved
	25-51	**r**D[37–63]	VSID[25–51], or b1, CNTLR_SPEC
none	$^{7}0$	**r**D[0–6]	0b0000_000

If the SLB entry selected by rB[32–35] was not created by an **mtsr**, **mtsrd**, or **mtsrdin** instruction, the contents of **r**D are undefined.

This is a supervisor-level instruction.

Other registers altered:

- None

PowerPC Architecture Level	Supervisor Level	32-Bit	64-Bit	64-Bit Bridge	Optional	Form
OEA	Ð	Ð		Ð		X

mftb mftb

Move from Time Base (x’7C00 02E6’)

mftb **r**D,TBR

▨ Reserved

31	D	tbr*	371	0
0 5	6 10	11 20	21 30	31

***Note:** This is a split field.

```
n ← tbr[5–9] || tbr[0–4]
if n = 268 then
  if (64-bit implementation) then
    rD ← TB
  else
    rD ← TBL
else if n = 269 then
  if (64-bit implementation) then
    rD ← (32)0 || TBU
  else
    rD ← TBU
```

When reading the time base lower (TBL) on a 64-bit implementation, the contents of the entire time base (TBU || TBL) is copied into **r**D. Note that when reading time base upper (TBU) on a 64-bit implementation the high-order 32 bits of **r**D are cleared. The contents of TBL or TBU are copied into rD, as designated by the value in TBR, encoded as shown in The TBR field denotes either the TBL or TBU, encoded as shown in Table 8-13. .

Table 8-13. TBR Encodings for mftb

TBR*			Register Name	Access
Decimal	tbr[5–9]	tbr[0–4]		
268	01000	01100	TBL	User
269	01000	01101	TBU	User
Note: *The order of the two 5-bit halves of the TBR number is reversed.				

If the TBR field contains any value other than one of the values shown in Table 8-13. , then one of the following occurs:

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

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

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

Other registers altered:

- None

Simplified mnemonics:

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

PowerPC Architecture Level	Supervisor Level	32-Bit	64-Bit	64-Bit Bridge	Optional	Form
VEA						XFX

mtcrf mtcrf

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

mtcrf CRM,**r**S

▨ Reserved

31	S	0	CRM	0	144	0
0–5	6–10	11	12–19	20	21–30	31

```
mask ← (4)(CRM[0]) || (4)(CRM[1]) ||... (4)(CRM[7])
CR ← (rS[32-63] & mask) | (CR & ¬ mask)
```

The contents of the low-order 32 bits 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 the low-order 32 bits of **r**S.

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

Other registers altered:

- CR fields selected by mask

Simplified mnemonics:

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

PowerPC Architecture Level	Supervisor Level	32-Bit	64-Bit	64-Bit Bridge	Optional	Form
UISA						XFX

mtfsb0*x*

Move to FPSCR Bit 0 (x’FC00 008C’)

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

▨ Reserved

63	crbD	0 0000	0000 0	70	Rc
0 5	6 10	11 15	16 20	21 30	31

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

Other registers altered:

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

  Note: Bits 1 and 2 (FEX and VX) cannot be explicitly cleared.

PowerPC Architecture Level	Supervisor Level	32-Bit	64-Bit	64-Bit Bridge	Optional	Form
UISA						X

mtfsb1*x* mtfsb1*x*

Move to FPSCR Bit 1 (x’FC00 004C’)

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

▨ Reserved

63	crbD	▨ 0 0000	▨ 0000 0	38	Rc
0 5	6 10	11 15	16 20	21 30	31

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

Other registers altered:

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

  Note: Bits 1 and 2 (FEX and VX) cannot be explicitly set.

PowerPC Architecture Level	Supervisor Level	32-Bit	64-Bit	64-Bit Bridge	Optional	Form
UISA						X

mtfsf*x* mtfsf*x*

Move to FPSCR Fields (x’FC00 058E’)

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

▨ Reserved

63	0	FM	0	B	711	Rc
0 5	6	7 14	15	16 20	21 30	31

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

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

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

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

Other registers altered:

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

PowerPC Architecture Level	Supervisor Level	32-Bit	64-Bit	64-Bit Bridge	Optional	Form
UISA						XFL

mtfsfi*x* mtfsfi*x*

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

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

▨ Reserved

63	crfD	0 0	0 0000	IMM	0	134	Rc
0 5	6 8	9 10	11 12 15	16 19	20	21 30	31

```
FPSCR[crfD] ← IMM
```

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

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

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

Other registers altered:

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

PowerPC Architecture Level	Supervisor Level	32-Bit	64-Bit	64-Bit Bridge	Optional	Form
UISA						X

mtmsr mtmsr

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

mtmsr **r**S

▨ Reserved

31	S	0 0000	0000 0	146	0
0 5	6 10	11 15	16 20	21 30	31

```
MSR ← (rS)
```

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

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

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

This instruction is defined only for 32-bit implementations. Using it on a 64-bit implementation causes an illegal instruction type program exception.

Other registers altered:

- MSR

Temporary 64-Bit Bridge

The **mtmsr** instruction may optionally be provided by a 64-bit implementation. The operation of the **mtmsr** instruction in a 64-bit implementation is identical to operation in a 32-bit implementation, except as described below:

- Bits 32–63 of **r**S are placed into the corresponding bits of the MSR. The high-order 32 bits of the MSR are unchanged.

Note that there is no need for an optional version of the **mfmsr** instruction, as the existing instruction copies the entire contents of the MSR to the selected GPR.

When the optional **mtmsr** instruction is provided in a 64-bit implementation, the optional **rfi** instruction is also provided. Refer to the **rfi** instruction description for additional detail about the operation of the **rfi** instruction in 64-bit implementations.

PowerPC Architecture Level	Supervisor Level	32-Bit	64-Bit	64-Bit Bridge	Optional	Form
OEA	Đ	Đ		Đ		X

mtmsrd 64-Bit Implementations Only mtmsrd

Move to Machine State Register Double Word (x’7C00 0164’)

mtmsrd rS

▨ Reserved

0–5	6–10	11–15	16–20	21–30	31
31	S	0 0000 ▨	0000 0 ▨	178	0

```
MSR ← (rS)
```

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

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

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

This instruction is defined only for 64-bit implementations. Using it on a 32-bit implementation causes an illegal instruction type program exception.

Other registers altered:

- MSR

PowerPC Architecture Level	Supervisor Level	32-Bit	64-Bit	64-Bit Bridge	Optional	Form
OEA	Đ		Đ			X

mtspr

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

mtspr SPR,**r**S

▨ Reserved

0–5	6–10	11–20	21–30	31
31	S	spr*	467	0 ▨

***Note:** This is a split field.

```
n ← spr[5–9] || spr[0–4]
if length (SPR(n)) = 64 then
  SPR(n) ← (rS)
else
  SPR(n) ← rS[32–63]
```

In the PowerPC UISA, the SPR field denotes a special-purpose register, encoded as shown in Table 8-14. . The contents of **r**S are placed into the designated special-purpose register. For special-purpose registers that are 32 bits long, the low-order 32 bits of **r**S are placed into the SPR.

Table 8-14. PowerPC UISA SPR Encodings for mtspr

SPR**			Register Name
Decimal	spr[5–9]	spr[0–4]	
1	00000	00001	XER
8	00000	01000	LR
9	00000	01001	CTR
Note: ** The order of the two 5-bit halves of the SPR number is reversed compared with actual instruction coding.			

If the SPR field contains any value other than one of the values shown in Table 8-14. , and the processor is operating in user mode, one of the following occurs:

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

Other registers altered:

- See Table 8-14. .

Simplified mnemonics:

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

In the PowerPC OEA, the SPR field denotes a special-purpose register, encoded as shown in Table 8-15. . The contents of **r**S are placed into the designated special-purpose register. For special-purpose registers that are 32 bits long, the low-order 32 bits of **r**S are placed into the SPR.

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

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

If MSR[PR] = 1 then the only effect of executing an instruction with an SPR number that is not shown in Table 8-15. and has SPR[0] = 1 is to cause a privileged instruction type program exception or an illegal instruction type program exception. For all other cases, MSR[PR] = 0 or SPR[0] = 0, if the SPR field contains any value that is not shown in Table 8-15. , either an illegal instruction type program exception occurs or the results are boundedly undefined.

Other registers altered:

- See Table 8-15. .

Table 8-15. PowerPC OEA SPR Encodings for mtspr

SPR[1]			Register Name	Access
Decimal	spr[5–9]	spr[0–4]		
1	00000	00001	XER	User
8	00000	01000	LR	User
9	00000	01001	CTR	User
18	00000	10010	DSISR	Supervisor
19	00000	10011	DAR	Supervisor
22	00000	10110	DEC	Supervisor
25	00000	11001	SDR1	Supervisor
26	00000	11010	SRR0	Supervisor
27	00000	11011	SRR1	Supervisor
272	01000	10000	SPRG0	Supervisor
273	01000	10001	SPRG1	Supervisor
274	01000	10010	SPRG2	Supervisor
275	01000	10011	SPRG3	Supervisor
280	01000	11000	ASR[2]	Supervisor
282	01000	11010	EAR	Supervisor
284	01000	11100	TBL	Supervisor
285	01000	11101	TBU	Supervisor
528	10000	10000	IBAT0U	Supervisor
529	10000	10001	IBAT0L	Supervisor
530	10000	10010	IBAT1U	Supervisor
531	10000	10011	IBAT1L	Supervisor
532	10000	10100	IBAT2U	Supervisor
533	10000	10101	IBAT2L	Supervisor
534	10000	10110	IBAT3U	Supervisor

Table 8-15. PowerPC OEA SPR Encodings for mtspr (Continued)

SPR[1]			Register Name	Access
Decimal	spr[5–9]	spr[0–4]		
535	10000	10111	IBAT3L	Supervisor
536	10000	11000	DBAT0U	Supervisor
537	10000	11001	DBAT0L	Supervisor
538	10000	11010	DBAT1U	Supervisor
539	10000	11011	DBAT1L	Supervisor
540	10000	11100	DBAT2U	Supervisor
541	10000	11101	DBAT2L	Supervisor
542	10000	11110	DBAT3U	Supervisor
543	10000	11111	DBAT3L	Supervisor
1013	11111	10101	DABR	Supervisor

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

[2]64-bit implementations only.

PowerPC Architecture Level	Supervisor Level	32-Bit	64-Bit	64-Bit Bridge	Optional	Form
UISA/OEA	Ð*					XFX

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

mtsr

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

mtsr SR,**r**S

▨ Reserved

0–5	6–10	11	12–15	16–20	21–30	31
31	S	0 ▨	SR	0 0 0 0 0 ▨	210	0 ▨

```
SEGREG(SR) ← (rS)
```

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

This is a supervisor-level instruction.

This instruction is defined only for 32-bit implementations. Using it on a 64-bit implementation causes an illegal instruction type program exception.

Other registers altered:

- None

Temporary 64-Bit Bridge

```
SLB(SR) ← (rS[32-63])
```

The SLB entry selected by SR is set as though it were loaded from a segment table entry, as shown in *Table 8-16*.

Table 8-16. SLB Entry Following mtsr

Double Word	Bit(s)	Contents	Description
0	0–31	0x0000_0000	ESID[0–31]
	32–35	SR	ESID[32–35]
	56	0b1	V
	57–59	**r**S[32-34]	T, Ks, Kp
	60–61	**r**S[35-36]	N, reserved bit, or b0
1	0–24	0x0000_00||0b0	VSID[0–24] or reserved
	25–51	**r**S[37-63]	VSID[25–51], or b1, CNTLR_SPEC

This is a supervisor-level instruction.

Note that when creating an ordinary segment (T = 0) using the **mtsr** instruction, rS[36–39] should be set to 0x0, as these bits correspond to the reserved bits in the T = 0 format for a segment register.

Other registers altered:

- None

PowerPC Architecture Level	Supervisor Level	32-Bit	64-Bit	64-Bit Bridge	Optional	Form
OEA	Ð	Ð		Ð		X

mtsrd 64-Bit Implementations Only mtsrd

Move to Segment Register Double Word (x'7C00 00A4')

TEMPORARY 64-BIT BRIDGE

mtsrd SR,**r**S

▨ Reserved

31	S	0	SR	0 0 0 0 0	82	0
0 5	6 10	11	12 15	16 20	21 30	31

```
SLB(SR) ← (rS)
```

The contents of **r**S are placed into the SLB selected by SR. The SLB entry is set as though it were loaded from an STE, as shown in *Table 8-17*.

Table 8-17. SLB Entry Following mtsrd

Double Word	Bit(s)	Contents	Description
0	0–31	0x0000_0000	ESID[0–31]
	32–35	SR	ESID[32–35]
	56	0b1	V
	57–59	**r**S[32–34]	T, Ks, Kp
	60–61	**r**S[35–36]	N, reserved bit, or b0
1	0–24	**r**S[7–31]	VSID[0–24] or reserved
	25–51	**r**S[37–63]	VSID[25–51], or b1, CNTLR_SPEC

This is a supervisor-level instruction.

This instruction is optional, and is defined only for 64-bit implementations. If the **mtsrd** instruction is implemented, the **mtsrdin** instruction will also be implemented. Using it on a 32-bit implementation causes an illegal instruction type program exception.

Other registers altered:

- None

PowerPC Architecture Level	Supervisor Level	32-Bit	64-Bit	64-Bit Bridge	Optional	Form
OEA	Ð		Ð	Ð	Ð	X

mtsrdin 64-Bit Implementations Only mtsrdin

Move to Segment Register Double Word Indirect (x’7C00 00E4’)

TEMPORARY 64-BIT BRIDGE

mtsrdin rS,**r**B

▨ Reserved

31	S	0 0000	B	114	0
0 5	6 10	11 15	16 20	21 30	31

```
SLB(rB[32-35]) ← (rS)
```

The contents of **r**S are copied to the SLB selected by bits 32–35 of **r**B. The SLB entry is set as though it were loaded from an STE, as shown in *Table 8-18*.

Table 8-18. SLB Entry following mtsrdin

Double Word	Bit(s)	Contents	Description
0	0–31	0x0000_0000	ESID[0–31]
	32–35	rB[32–35]	ESID[32–35]
	56	0b1	V
	57–59	**r**S[32–34]	T, Ks, Kp
	60–61	**r**S[35–36]	N, reserved bit, or b0
1	0–24	rS[7–31]	VSID[0-24] or reserved
	25–51	rS[37–63]	VSID[25–51], or b1, CNTLR_SPEC

This is a supervisor-level instruction.

This instruction is optional, and defined only for 64-bit implementations. If the **mtsrdin** instruction is implemented, the **mtsrd** instruction will also be implemented. Using it on a 32-bit implementation causes an illegal instruction exception.

Other registers altered:

- None

PowerPC Architecture Level	Supervisor Level	32-Bit	64-Bit	64-Bit Bridge	Optional	Form
OEA	Ð		Ð	Ð	Ð	X

mtsrin

mtsrin

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

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

[POWER mnemonic: **mtsri**]

▨ Reserved

0–5	6–10	11–15	16–20	21–30	31
31	S	0 0000 ▨	B	242	0 ▨

```
SEGREG(rB[0-3]) ← (rS)
```

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

This is a supervisor-level instruction.

This instruction is defined only for 32-bit implementations. Using it on a 64-bit implementation causes an illegal instruction type program exception.

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

Other registers altered:

- None

TEMPORARY 64-BIT BRIDGE

```
SLB(rB[32-35]) ← (rS[32-63])
```

The SLB entry selected by bits 32-35 of **r**B is set as though it were loaded from a segment table entry, as shown in *Table 8-19*.

Table 8-19. SLB Entry Following mtsrin

Double Word	Bit(s)	Contents	Description
0	0–31	0x0000_0000	ESID[0–31]
	32–35	**r**B[32–35]	ESID[32–35]
	56	0b1	V
	57–59	**r**S[32–34]	T, Ks, Kp
	60–61	**r**S[35–36]	N, reserved bit, or b0
1	0–24	0x0000_00||0b0	VSID[0–24] or reserved
	25–51	**r**S[37–63]	VSID[25–51], or b1, CNTLR_SPEC

This is a supervisor-level instruction.

Note that when creating an ordinary segment (T = 0) using the **mtsrin** instruction, rS[36–39] should be set to 0x0, as these bits correspond to the reserved bits in the T = 0 format for a segment register.

Other registers altered:

- None

PowerPC Architecture Level	Supervisor Level	32-Bit	64-Bit	64-Bit Bridge	Optional	Form
OEA	Ð	Ð		Ð		X

mulhd*x* 64-Bit Implementations Only mulhd*x*

Multiply High Double Word (x'7C00 0092')

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

31	D	A	B	0	73	Rc
0 5	6 10	11 15	16 20	21	22 30	31

```
prod[0-127] ← (rA) * (rB)
rD ← prod[0-63]
```

The 64-bit operands are (**r**A) and (**r**B). The high-order 64 bits of the 128-bit product of the operands are placed into **r**D.

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

This instruction is defined only for 64-bit implementations. Using it on a 32-bit implementation will cause the system illegal instruction error handler to be invoked.

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

Other registers altered:

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

  Note: The setting of CR0 bits LT, GT, and EQ is mode-dependent, and reflects overflow of the 64-bit result.

PowerPC Architecture Level	Supervisor Level	32-Bit	64-Bit	64-Bit Bridge	Optional	Form
UISA			Ð			XO

mulhdu*x* 64-Bit Implementations Only **mulhdu*x***

Multiply High Double Word Unsigned (x’7C00 0012’)

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

31	D	A	B	0	9	Rc
0 5	6 10	11 15	16 20	21	22 30	31

```
prod[0-127] ← (rA) * (rB)
rD ← prod[0-63]
```

The 64-bit operands are (**r**A) and (**r**B). The high-order 64 bits of the 128-bit product of the operands are placed into **r**D.

Both the operands and the product are interpreted as unsigned integers, except that if Rc = 1 the first three bits of CR0 field are set by signed comparison of the result to zero.

This instruction is defined only for 64-bit implementations. Using it on a 32-bit implementation will cause the system illegal instruction error handler to be invoked.

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

Other registers altered:

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

  Note: The setting of CR0 bits LT, GT, and EQ is mode-dependent, and reflects overflow of the 64-bit result.

PowerPC Architecture Level	Supervisor Level	32-Bit	64-Bit	64-Bit Bridge	Optional	Form
UISA			Đ			XO

mulhw*x*

Multiply High Word (x’7C00 0096’)

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

▨ Reserved

31	D	A	B	0 ▨	75	Rc
0–5	6–10	11–15	16–20	21	22–30	31

```
prod[0-63] ← rA[32-63] * rB[32-63]
rD[32-63] ← prod[0-31]
rD[0-31] ← undefined
```

The 6432-bit product is formed from the contents of the low-order 32 bits of **r**A and **r**B. The high-order 32 bits of the 64-bit product of the operands are placed into the low-order 32 bits of **r**D. The high-order 32 bits of **r**D are undefined.

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

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

Other registers altered:

- Condition Register (CR0 field):
  Affected: LT, GT, EQ, SO (if Rc = 1)
  LT, GT, EQ undefined(if Rc =1 and 64-bit mode)

  Note: The setting of CR0 bits LT, GT, and EQ is mode-dependent, and reflects overflow of the 32-bit result.

PowerPC Architecture Level	Supervisor Level	32-Bit	64-Bit	64-Bit Bridge	Optional	Form
UISA						XO

mulhwu*x* mulhwu*x*

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

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

▨ Reserved

31	D	A	B	0	11	Rc
0 5	6 10	11 15	16 20	21	22 30	31

```
prod[0-63] ← rA[32-63] * rB[32-63]
rD[32-63] ← prod[0-31]
rD[0-31] ← undefined
```

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

Both the operands and the product are interpreted as unsigned integers, except that if Rc = 1 the first three bits of CR0 field are set by signed comparison of the result to zero.

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

Other registers altered:

- Condition Register (CR0 field):
  Affected: LT, GT, EQ, SO(if Rc = 1)
  LT, GT, EQ undefined(if Rc =1 and 64-bit mode)

  Note: The setting of CR0 bits LT, GT, and EQ is mode-dependent, and reflects overflow of the 32-bit result.

PowerPC Architecture Level	Supervisor Level	32-Bit	64-Bit	64-Bit Bridge	Optional	Form
UISA						XO

mulld*x* — 64-Bit Implementations Only — mulld*x*

Multiply Low Double Word (x’7C00 01D2’)

mulld	**r**D,**r**A,**r**B	(OE = 0 Rc = 0)
mulld.	**r**D,**r**A,**r**B	(OE = 0 Rc = 1)
mulldo	**r**D,**r**A,**r**B	(OE = 1 Rc = 0)
mulldo.	**r**D,**r**A,**r**B	(OE = 1 Rc = 1)

31	D	A	B	OE	233	Rc
0 5	6 10	11 15	16 20	21	22 30	31

```
prod[0-127] ← (rA) * (rB)
rD ← prod[64-127]
```

The 64-bit operands are the contents of **r**A and **r**B. The low-order 64 bits of the 128-bit product of the operands are placed into **r**D.

Both the operands and the product are interpreted as signed integers. The low-order 64 bits of the product are independent of whether the operands are regarded as signed or unsigned 64-bit integers. If OE = 1, then OV is set if the product cannot be represented in 64 bits.

This instruction is defined only for 64-bit implementations. Using it on a 32-bit implementation will cause the system illegal instruction error handler to be invoked.

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

Other registers altered:

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

  Note: CR0 field may not reflect the infinitely precise result if overflow occurs (see XER below).
- XER:
  Affected: SO, OV(if OE = 1)

  Note: The setting of the affected bits in the XER is mode-independent, and reflects overflow of the 64-bit result.

PowerPC Architecture Level	Supervisor Level	32-Bit	64-Bit	64-Bit Bridge	Optional	Form
UISA			Ð			XO

mulli

Multiply Low Immediate (x’1C00 0000’)

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

[POWER mnemonic: **muli**]

07	D	A	SIMM
0 5	6 10	11 15	16 31

```
prod[0-12748] ← (rA) * EXTS(SIMM)
rD ← prod[64-12716-48]
```

The 6432-bit first operand is (**r**A). The 6416-bit second operand is the sign-extended value of the SIMM field. The low-order 6432-bits of the 12848-bit product of the operands are placed into **r**D.

Both the operands and the product are interpreted as signed integers. The low-order 64 bits (or 32 bits) of the product are calculated independently of whether the operands are treated as signed or unsigned 64-bit (or 32-bit) integers.

This instruction can be used with **mulhd***x* or **mulhw***x* to calculate a full 128-bit (or 64-bit) product.

The low-order 32 bits of the product are the correct 32-bit product for 32-bit implementations and for 32-bit mode in 64-bit implementations.

Other registers altered:

- None

PowerPC Architecture Level	Supervisor Level	32-Bit	64-Bit	64-Bit Bridge	Optional	Form
UISA						D

mullw*x*

Multiply Low Word (x'7C00 01D6')

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

[POWER mnemonics: **muls**, **muls.**, **mulso**, **mulso.**]

31	D	A	B	OE	235	Rc
0 5	6 10	11 15	16 20	21	22 30	31

```
rD ← rA[32-63] * rB[32-63]
```

The 32-bit operands are the contents of the low-order 32 bits of **r**A and **r**B. The low-order 32 bits of the 64-bit product (**r**A) * (**r**B) are placed into **r**D.

The low-order 32 bits of the product are the correct 32-bit product for 32-bit mode of 64-bit implementations and for 32-bit implementations. The low-order 32-bits of the product are independent of whether the operands are regarded as signed or unsigned 32-bit integers.

If OE = 1, then OV is set if the product cannot be represented in 32 bits. Both the operands and the product are interpreted as signed integers.

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

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

Other registers altered:

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

  Note: CR0 field may not reflect the infinitely precise result if overflow occurs (see XER below).
- XER:
  Affected: SO, OV(if OE = 1)

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

PowerPC Architecture Level	Supervisor Level	32-Bit	64-Bit	64-Bit Bridge	Optional	Form
UISA						XO

nand*x* **nand***x*

NAND (x’7C00 03B8’)

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

31	S	A	B	476	Rc
0 5	6 10	11 15	16 20	21 30	31

```
rA ← ¬ ((rS) & (rB))
```

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

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

Other registers altered:

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

PowerPC Architecture Level	Supervisor Level	32-Bit	64-Bit	64-Bit Bridge	Optional	Form
UISA						X

neg*x*

neg*x*

Negate (x’7C00 00D0’)

neg	**r**D,**r**A	(OE = 0 Rc = 0)
neg.	**r**D,**r**A	(OE = 0 Rc = 1)
nego	**r**D,**r**A	(OE = 1 Rc = 0)
nego.	**r**D,**r**A	(OE = 1 Rc = 1)

▨ Reserved

31	D	A	0 0 0 0 0	OE	104	Rc
0 5	6 10	11 15	16 20	21	22 30	31

```
rD ← ¬ (rA) + 1
```

The value 1 is added to the complement of the value in **r**A, and the resulting two’s complement is placed into **r**D.

If executing in the default 64-bit mode and **r**A contains the most negative 6432-bit number (0x8000_0000_0000_0000), the result is the most negative number and, if OE = 1, OV is set. Similarly, if executing in 32-bit mode of a 64-bit implementation (or on a 32-bit implementation) and the low-order 32 bits of **r**A contains the most negative 32-bit number (0x8000_0000), the low-order 32 bits of the result contain 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)

PowerPC Architecture Level	Supervisor Level	32-Bit	64-Bit	64-Bit Bridge	Optional	Form
UISA						XO

nor*x* **nor***x*

NOR (x’7C00 00F8’)

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

0–5	6–10	11–15	16–20	21–30	31
31	S	A	B	124	Rc

```
rA ← ¬ ((rS) | (rB))
```

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

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

Other registers altered:

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

Simplified mnemonics:

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

PowerPC Architecture Level	Supervisor Level	32-Bit	64-Bit	64-Bit Bridge	Optional	Form
UISA						X

or*x* **or***x*

OR (x’7C00 0378’)

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

31	S	A	B	444	Rc
0 5	6 10	11 15	16 20	21 30	31

```
rA ← (rS) | (rB)
```

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

The simplified mnemonic **mr** (shown below) demonstrates the use of the **or** instruction to move register contents.

Other registers altered:

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

Simplified mnemonics:

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

PowerPC Architecture Level	Supervisor Level	32-Bit	64-Bit	64-Bit Bridge	Optional	Form
UISA						X

orc*x* orc*x*

OR with Complement (x’7C00 0338’)

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

31	S	A	B	412	Rc
0 5	6 10	11 15	16 20	21 30	31

```
rA ← (rS) | ¬ (rB)
```

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

Other registers altered:

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

PowerPC Architecture Level	Supervisor Level	32-Bit	64-Bit	64-Bit Bridge	Optional	Form
UISA						X

ori ori

OR Immediate (x’6000 0000’)

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

[POWER mnemonic: **oril**]

24	S	A	UIMM
0 5	6 10	11 15	16 31

```
rA ← (rS) | ((4816)0 || UIMM)
```

The contents of **r**S are ORed with 0x0000_0000_0000 || UIMM and the result is placed into **r**A.

The preferred no-op (an instruction that does nothing) is **ori 0,0,0**.

Other registers altered:

- None

Simplified mnemonics:

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

PowerPC Architecture Level	Supervisor Level	32-Bit	64-Bit	64-Bit Bridge	Optional	Form
UISA						D

oris oris

OR Immediate Shifted (x’6400 0000’)

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

[POWER mnemonic: **oriu**]

25	S	A	UIMM
0 5	6 10	11 15	16 31

```
rA ← (rS) | ((32)0 || UIMM || (16)0)
```

The contents of **r**S are ORed with 0x0000_0000 || UIMM || 0x0000 and the result is placed into **r**A.

Other registers altered:

- None

PowerPC Architecture Level	Supervisor Level	32-Bit	64-Bit	64-Bit Bridge	Optional	Form
UISA						D

rfi

rfi

Return from Interrupt (x’4C00 0064’)

▨ Reserved

19	00 000	0 0000	**0000 0**	**50**	**0**
0 5	6 10	11 15	16 20	21 30	31

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

Bits SRR1[16–23, 25–27, 30–31] are placed into the corresponding bits of the MSR. If the new MSR value does not enable any pending exceptions, then the next instruction is fetched, under control of the new MSR value, from the address SRR0[0–29] || 0b00. If the new MSR value enables one or more pending exceptions, the exception associated with the highest priority pending exception is generated; in this case the value placed into SRR0 by the exception processing mechanism is the address of the instruction that would have been executed next had the exception not occurred. Note that an implementation may define additional MSR bits, and in this case, may also cause them to be saved to SRR1 from MSR on an exception and restored to MSR from SRR1 on an **rfid** (or **rfi**).

This is a supervisor-level, context synchronizing instruction. This instruction is defined only for 32-bit implementations. Using it on a 64-bit implementation causes an illegal instruction type program exception.

Other registers altered:

- MSR

TEMPORARY 64-BIT BRIDGE

The **rfi** instruction may optionally be provided by a 64-bit implementation. The operation of the **rfi** instruction in a 64-bit implementation is identical to the operation in a 32-bit implementation, except as described below:

- The SRR1 bits that are copied to the corresponding bits of the MSR are bits 48–55, 57–59 and 62–63 of SRR1. Note that depending on the implementation, additional bits from SRR1 may be restored to the MSR. The remaining bits of the MSR, including the high-order bits, are unchanged.
- 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[0–61 || 0b00 (when SF = 1 in the new MSR value), or from 0x0000_0000 || SRR[32–61] ||0b00 (when SF = 0 in the new MSR value).

When the optional **rfi** instruction is provided in a 64-bit implementation, the optional **mtmsr** instruction is also provided. Refer to the **mtmsr** instruction description for additional detail about the operation of the **mtmsr** instruction in 64-bit implementations.

PowerPC Architecture Level	Supervisor Level	32-Bit	64-Bit	64-Bit Bridge	Optional	Form
OEA	Ð	Ð		Ð		XL

rfid 64-Bit Implementations Only rfid

Return from Interrupt Double Word (x’4C00 0024’)

▨ Reserved

0–5	6–10	11–15	16–20	21–30	31
19	▨ 00 000	▨ 0 0000	▨ 0000 0	18	▨ 0

```
MSR[0, 48-55, 57-59, 62-63] ← SRR1[0, 48-55, 57-59, 62-63]
NIA ←iea SRR0[0-61] || 0b00
```

Bits SRR1[0, 48–55, 57–59, 62–63] are placed into the corresponding bits of the MSR. If the new MSR value does not enable any pending exceptions, then the next instruction is fetched, under control of the new MSR value, from the address SRR0[0–61] || 0b00 (when
MSR[SF] = 1) or 0x0000_0000 || SRR0[32–61] || 0b00 (when MSR[SF] = 0). If the new MSR value enables one or more pending exceptions, the exception associated with the highest priority pending exception is generated; in this case the value placed into SRR0 by the exception processing mechanism is the address of the instruction that would have been executed next had the exception not occurred. Note that an implementation may define additional MSR bits, and in this case, may also cause them to be saved to SRR1 from MSR on an exception and restored to MSR from SRR1 on an **rfid**.

This is a supervisor-level, context synchronizing instruction.

This instruction is defined only for 64-bit implementations. Using it on a 32-bit implementation causes an illegal instruction type program exception.

Other registers altered:

- MSR

PowerPC Architecture Level	Supervisor Level	32-Bit	64-Bit	64-Bit Bridge	Optional	Form
OEA	Đ		Đ			XL

rldcl*x* 64-Bit Implementations Only rldcl*x*

Rotate Left Double Word then Clear Left (x'7800 0010')

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

30	S	A	B	mb*	8	Rc
0 5	6 10	11 15	16 20	21 26	27 30	31

***Note:** This is a split field.

```
n ← rB[58-63]
r ← ROTL[64](rS, n)
b ← mb[5] || mb[0-4]
m ← MASK(b, 63)
rA ← r & m
```

The contents of **r**S are rotated left the number of bits specified by operand in the low-order six bits of **r**B. A mask is generated having 1 bits from bit MB through bit 63 and 0 bits elsewhere. The rotated data is ANDed with the generated mask and the result is placed into **r**A.

This instruction is defined only for 64-bit implementations. Using it on a 32-bit implementation will cause the system illegal instruction error handler to be invoked.

Note that the **rldcl** instruction can be used to extract and rotate bit fields using the methods shown below:

- To extract an *n*-bit field, that starts at variable bit position *b* in register **r**S, right-justified into **r**A (clearing the remaining 64 – *n* bits of **r**A), set the low-order six bits of **r**B to *b* + *n* and MB = 64 – *n*.
- To rotate the contents of a register left by variable *n* bits, set the low-order six bits of **r**B to *n* and MB = 0, and to shift the contents of a register right, set the low-order six bits of **r**B to(64 – *n*), and MB = 0.

Other registers altered:

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

Simplified mnemonics:

rotld	**r**A,**r**S,**r**B	equivalent to	**rldcl**	**r**A,**r**S,**r**B,**0**

PowerPC Architecture Level	Supervisor Level	32-Bit	64-Bit	64-Bit Bridge	Optional	Form
UISA			Ð			MDS

rldcr*x* 64-Bit Implementations Only **rldcr***x*

Rotate Left Double Word then Clear Right (x’7800 0012’)

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

30	S	A	B	me*	9	Rc
0 5	6 10	11 15	16 20	21 26	27 30	31

***Note:** This is a split field.

```
n ← rB[58-63]
r ← ROTL[64](rS, n)
e ← me[5] || me[0-4]
m ← MASK(0, e)
rA ← r & m
```

The contents of **r**S are rotated left the number of bits specified by the low-order six bits of **r**B. A mask is generated having 1 bits from bit 0 through bit ME and 0 bits elsewhere. The rotated data is ANDed with the generated mask and the result is placed into **r**A.

This instruction is defined only for 64-bit implementations. Using it on a 32-bit implementation will cause the system illegal instruction error handler to be invoked.

Note that **rldcr** can be used to extract and rotate bit fields using the methods shown below:

- To extract an *n*-bit field, that starts at variable bit position *b* in register **r**S, left-justified into **r**A (clearing the remaining 64 – *n* bits of **r**A), set the low-order six bits of **r**B to *b* and ME = *n* – 1.
- To rotate the contents of a register left by variable *n* bits, set the low-order six bits of **r**B to *n* and ME = 63, and to shift the contents of a register right, set the low-order six bits of **r**B to(64 – *n*), and ME = 63.

Other registers altered:

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

For a detailed list of simplified mnemonics for the **rldcr** instruction, refer to Appendix F. , “Simplified Mnemonics.”

PowerPC Architecture Level	Supervisor Level	32-Bit	64-Bit	64-Bit Bridge	Optional	Form
UISA			Ð			MDS

rldic*x* 64-Bit Implementations Only rldic*x*

Rotate Left Double Word Immediate then Clear (x’7800 0008’)

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

30	S	A	sh*	mb*	2	sh*	Rc
0 5	6 10	11 15	16 20	21 26	27 29	30	31

***Note:** This is a split field.

```
n ← sh[5] || sh[0-4]
r ← ROTL[64](rS, n)
b ← mb[5] || mb[0-4]
m ← MASK(b, ¬n)
rA ← r & m
```

The contents of **r**S are rotated left the number of bits specified by operand SH. A mask is generated having 1 bits from bit MB through bit 63 – SH and 0 bits elsewhere. The rotated data is ANDed with the generated mask and the result is placed into **r**A.

This instruction is defined only for 64-bit implementations. Using it on a 32-bit implementation will cause the system illegal instruction error handler to be invoked.

Note that **rldic** can be used to clear and shift bit fields using the methods shown below:

- To clear the high-order *b* bits of the contents of a register and then shift the result left by *n* bits, set SH = *n* and MB = *b* – *n*.
- To clear the high-order *n* bits of a register, set SH = 0 and MB = *n*.

Other registers altered:

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

Simplified mnemonics:

clrlsldi **r**A,**r**S,*b*,*n*equivalent to**rldic****r**A,**r**S,*n*,*b* – *n*

For a more detailed list of simplified mnemonics for the **rldic** instruction, refer to Appendix F. , “Simplified Mnemonics.”

PowerPC Architecture Level	Supervisor Level	32-Bit	64-Bit	64-Bit Bridge	Optional	Form
UISA			Ð			MD

rldicl*x* 64-Bit Implementations Only rldicl*x*

Rotate Left Double Word Immediate then Clear Left (x’7800 0000’)

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

30	S	A	sh*	mb*	0	sh*	Rc
0 5	6 10	11 15	16 20	21 26	27 29	30	31

***Note:** This is a split field.

```
n ← sh[5] || sh[0-4]
r ← ROTL[64](rS, n)
b ← mb[5] || mb[0-4]
m ← MASK(b, 63)
rA ← r & m
```

The contents of **r**S are rotated left the number of bits specified by operand SH. A mask is generated having 1 bits from bit MB through bit 63 and 0 bits elsewhere. The rotated data is ANDed with the generated mask and the result is placed into **r**A.

This instruction is defined only for 64-bit implementations. Using it on a 32-bit implementation will cause the system illegal instruction error handler to be invoked.

Note that **rldicl** can be used to extract, rotate, shift, and clear bit fields using the methods shown below:

- To extract an *n*-bit field, that starts at bit position *b* in **r**S, right-justified into **r**A (clearing the remaining 64 – *n* bits of **r**A), set SH = *b* + *n* and MB = 64 – *n*.
- To rotate the contents of a register left by *n* bits, set SH = *n* and MB = 0; to rotate the contents of a register right by *n* bits, set SH = (64 – *n*), and MB = 0.
- To shift the contents of a register right by *n* bits, set SH = 64 – *n* and MB = *n*.
- To clear the high-order *n* bits of a register, set SH = 0 and MB = *n*.

Other registers altered:

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

Simplified mnemonics:

extrdi rA,**r**S,*n*,*b* (*n* > 0)	equivalent to	**rldicl r**A,**r**S,*b* **+** *n*,64 – *n*
rotldi rA,**r**S,*n*	equivalent to	**rldicl r**A,**r**S,*n*,**0**
rotrdi rA,**r**S,*n*	equivalent to	**rldicl r**A,**r**S,64 – *n*,**0**
srdi rA,**r**S,*n* (*n* < 64)	equivalent to	**rldicl r**A,**r**S,64 – *n*,*n*
clrldi rA,**r**S,*n* (*n* < 64)	equivalent to	**rldicl r**A,**r**S,**0**,*n*

PowerPC Architecture Level	Supervisor Level	32-Bit	64-Bit	64-Bit Bridge	Optional	Form
UISA			Ð			MD

rldicrx 64-Bit Implementations Only rldicr*x*

Rotate Left Double Word Immediate then Clear Right (x'7800 0004')

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

30	S	A	sh*	me*	1	sh*	Rc
0 5	6 10	11 15	16 20	21 26	27 29	30	31

***Note:** This is a split field.

```
n ← sh[5] || sh[0-4]
r ← ROTL[64](rS, n)
e ← me[5] || me[0-4]
m ← MASK(0, e)
rA ← r & m
```

The contents of **r**S are rotated left the number of bits specified by operand SH. A mask is generated having 1 bits from bit 0 through bit ME and 0 bits elsewhere. The rotated data is ANDed with the generated mask and the result is placed into **r**A.

This instruction is defined only for 64-bit implementations. Using it on a 32-bit implementation will cause the system illegal instruction error handler to be invoked.

Note that **rldicr** can be used to extract, rotate, shift, and clear bit fields using the methods shown below:

- To extract an *n*-bit field, that starts at bit position *b* in **r**S, left-justified into **r**A (clearing the remaining 64 – *n* bits of **r**A), set SH = *b* and ME = *n* – 1.
- To rotate the contents of a register left (right) by *n* bits, set SH = *n* (64 – *n*) and ME = 63.
- To shift the contents of a register left by *n* bits, by setting SH = *n* and ME = 63 – *n*.
- To clear the low-order *n* bits of a register, by setting SH = 0 and ME = 63 – *n*.

Other registers altered:

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

Simplified mnemonics:

extldi	**r**A,**r**S,*n*,*b*	equivalent to	**rldicr**	**r**A,**r**S,*b*,*n* – 1
sldi	**r**A,**r**S,*n*	equivalent to	**rldicr**	**r**A,**r**S,*n*,63 – *n*
clrrdi	**r**A,**r**S,*n*	equivalent to	**rldicr**	**r**A,**r**S,**0**,63 – *n*

PowerPC Architecture Level	Supervisor Level	32-Bit	64-Bit	64-Bit Bridge	Optional	Form
UISA			Ð			MD

rldimi*x* 64-Bit Implementations Only rldimi*x*

Rotate Left Double Word Immediate then Mask Insert (x’7800 000C’)

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

30	S	A	sh*	mb*	3	sh*	Rc
0 5	6 10	11 15	16 20	21 26	27 29	30	31

*Note: This is a split field.

```
n ← sh[5] || sh[0-4]
r ← ROTL[64](rS, n)
b ← mb[5] || mb[0-4]
m ← MASK(b, ¬n)
rA ← (r & m) | (rA & ¬ m)
```

The contents of **r**S are rotated left the number of bits specified by operand SH. A mask is generated having 1 bits from bit MB through bit 63 – SH and 0 bits elsewhere. The rotated data is inserted into **r**A under control of the generated mask.

This instruction is defined only for 64-bit implementations. Using it on a 32-bit implementation will cause the system illegal instruction error handler to be invoked.

Note that **rldimi** can be used to insert an *n*-bit field, that is right-justified in **r**S, into **r**A starting at bit position *b*, by setting SH = 64 – (*b* + *n*) and MB = *b*.

Other registers altered:

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

Simplified mnemonics:

insrdi	**r**A,**r**S,*n*,*b*	equivalent to	**rldimi**	**r**A,**r**S,64 – (*b* + *n*),*b*

For a more detailed list of simplified mnemonics for the **rldimi** instruction, refer to Appendix F. , “Simplified Mnemonics.”

PowerPC Architecture Level	Supervisor Level	32-Bit	64-Bit	64-Bit Bridge	Optional	Form
UISA			Ð			MD

rlwimi*x*

Rotate Left Word Immediate then Mask Insert (x'5000 0000')

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

[POWER mnemonics: **rlimi, rlimi**.]

20	S	A	SH	MB	ME	Rc
0 5	6 10	11 15	16 20	21 25	26 30	31

```
n ← SH
r ← ROTL[32](rS[32-63], n)
m ← MASK(MB + 32, ME + 32)
rA ← (r & m) | (rA & ¬ m)
```

The contents of **r**S are rotated left the number of bits specified by operand SH. A mask is generated having 1 bits from bit MB + 32 through bit ME + 32 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 the low-order 32 bits of **r**S, into the high-order 32 bits of **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 the low-order 32 bits of **r**S, into the high-order 32 bits of **r**A starting at bit position *b*, set SH = 32 – (*b* + *n*), MB = *b*, and ME = (*b* + *n*) – 1.

Other registers altered:

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

Simplified mnemonics:

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

PowerPC Architecture Level	Supervisor Level	32-Bit	64-Bit	64-Bit Bridge	Optional	Form
UISA						M

rlwinm*x* rlwinm*x*

Rotate Left Word Immediate then AND with Mask (x'5400 0000')

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

[POWER mnemonics: **rlinm, rlinm.**]

21	S	A	SH	MB	ME	Rc
0 5	6 10	11 15	16 20	21 25	26 30	31

```
n ← SH
r ← ROTL[32](rS[32-63], n)
m ← MASK(MB + 32, ME + 32)
rA ← r & m
```

The contents of **r**S[32-63] are rotated left the number of bits specified by operand SH. A mask is generated having 1 bits from bit MB + 32 through bit ME + 32 and 0 bits elsewhere. The rotated data is ANDed with the generated mask and the result is placed into **r**A. The upper 32 bits of **r**A are cleared.

Note that **rlwinm** can be used to extract, rotate, shift, and clear bit fields using the methods shown below:

- To extract an *n*-bit field, that starts at bit position *b* in the high-order 32 bits of **r**S, right-justified into **r**A (clearing the remaining 32 – *n* bits of **r**A), set SH = *b* + *n*, MB = 32 – *n*, and ME = 31.
- To extract an *n*-bit field, that starts at bit position *b* in the high-order 32 bits of **r**S, left-justified into **r**A (clearing the remaining 32 – *n* bits of **r**A), set SH = *b*, MB = 0, and ME = *n* – 1.
- To rotate the contents of a register left (or right) by *n* bits, set SH = *n* (32 – *n*), MB = 0, and ME = 31.
- To shift the contents of a register right by *n* bits, by setting SH = 32 – *n*, MB = *n*, and ME = 31. It can be used to clear the high-order *b* bits of a register and then shift the result left by *n* bits by setting SH = *n*, MB = *b* – *n* and ME = 31 – *n*.
- To clear the low-order *n* bits of a register, by setting SH = 0, MB = 0, and ME = 31 – *n*.

For all uses mentioned, the high-order 32 bits of **r**A are cleared.

Other registers altered:

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

Simplified mnemonics:

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

clrlwi rA,**r**S,*n* (*n* < 32) equivalent to**rlwinm r**A,**r**S,**0,***n***,31**
clrrwi rA,**r**S,*n* (*n* < 32) equivalent to**rlwinm r**A,**r**S,**0,0,**31 – *n*
clrlslwi rA,**r**S,*b,n* (*n* ð *b* < 32) equivalent to**rlwinm r**A,**r**S,*n,b* – *n*,31 – *n*

PowerPC Architecture Level	Supervisor Level	32-Bit	64-Bit	64-Bit Bridge	Optional	Form
UISA						M

rlwnm*x* rlwnm*x*

Rotate Left Word then AND with Mask (x’5C00 0000’)

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

[POWER mnemonics: **rlnm, rlnm.**]

23	S	A	B	MB	ME	Rc
0 5	6 10	11 15	16 20	21 25	26 30	31

```
n ← rB[59-6327-31]
r ← ROTL[32](rS[32-63], n)
m ← MASK(MB + 32, ME + 32)
rA ← r & m
```

The contents of **r**S are rotated left the number of bits specified by the low-order five bits of **r**B. A mask is generated having 1 bits from bit MB + 32 through bit ME + 32 and 0 bits elsewhere. The rotated data is ANDed with the generated mask and the result is placed into **r**A.

Note that **rlwnm** can be used to extract and rotate bit fields using the methods shown as follows:

- To extract an *n*-bit field, that starts at variable bit position *b* in the high-order 32 bits of **r**S, right-justified into **r**A (clearing the remaining 32 – *n* bits of **r**A), by setting the low-order five bits of **r**B to *b* + *n*, MB = 32 – *n*, and ME = 31.
- To extract an *n*-bit field, that starts at variable bit position *b* in the high-order 32 bits of **r**S, left-justified into **r**A (clearing the remaining 32 – *n* bits of **r**A), by setting the low-order five bits of **r**B to *b*, MB = 0, and ME = *n* – 1.
- To rotate the contents of a register left (or right) by *n* bits, by setting the low-order five bits of **r**B to *n* (32 – *n*), MB = 0, and ME = 31.

For all uses mentioned, the high-order 32 bits of **r**A are cleared.

Other registers altered:

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

Simplified mnemonics:

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

PowerPC Architecture Level	Supervisor Level	32-Bit	64-Bit	64-Bit Bridge	Optional	Form
UISA						M

sc sc

System Call (x’4400 0002’)

[POWER mnemonic: **svca**]

▨ Reserved

17	00 000	0 0000	0000 0000 0000 00	1	0
0 5	6 10	11 15	16 29	30	31

In the PowerPC UISA, the **sc** instruction calls the operating system to perform a service. When control is returned to the program that executed the system call, the content of the registers depends on the register conventions used by the program providing the system service.

This instruction is context synchronizing, as described in Section 4.1.5.1 , “Context Synchronizing Instructions.”

Other registers altered:

- Dependent on the system service

In PowerPC OEA, the **sc** instruction does the following:

```
SRR0 ←iea CIA + 4
SRR1[33-361-4, 42-4710-15] ← 0
SRR1[0, 48-5516-23, 57-5925-27, 62-6330-31] ← MSR[0, 48-5516-23, 57-5925-27, 62-6330-
31]
MSR ← new_value (see below)
NIA ←iea base_ea + 0xC00 (see below)
```

The EA of the instruction following the **sc** instruction is placed into SRR0. Bits 0, 48–5516–23, 57–5925–27, and 62–6330–31 of the MSR are placed into the corresponding bits of SRR1, and bits 33–361-4 and 42–4710-15 of SRR1 are set to undefined values. Note that an implementation may define additional MSR bits, and in this case, may also cause them to be saved to SRR1 from MSR on an exception and restored to MSR from SRR1 on an **rfid** (or **rfi**).

Then a system call exception is generated. The exception causes the MSR to be altered as described in Section 6.4 , “Exception Definitions.”

The exception causes the next instruction to be fetched from offset 0xC00 from the physical base address determined by the new setting of MSR[IP].

Other registers altered:

- SRR0
- SRR1
- MSR

PowerPC Architecture Level	Supervisor Level	32-Bit	64-Bit	64-Bit Bridge	Optional	Form
UISA/OEA						SC

slbia 64-Bit Implementations Only slbia

SLB Invalidate All (x’7C00 03E4’)

▨ Reserved

31	00 000	0 0000	0000 0	498	0
0 5	6 10	11 15	16 20	21 30	31

```
All SLB entries ← invalid
```

The entire segment lookaside buffer (SLB) is made invalid (that is, all entries are removed).

The SLB is invalidated regardless of the settings of MSR[IR] and MSR[DR].

This instruction is supervisor-level.

This instruction is optional in the PowerPC architecture.

This instruction is defined only for 64-bit implementations. Using it on a 32-bit implementation will cause an illegal instruction type program *exception*.

It is not necessary that the ASR point to a valid segment table when issuing **slbia**.

Other registers altered:

- None

PowerPC Architecture Level	Supervisor Level	32-Bit	64-Bit	64-Bit Bridge	Optional	Form
OEA	Ð		Ð		Ð	X

slbie 64-Bit Implementations Only slbie

SLB Invalidate Entry (x’7C00 0364’)

slbie **r**B

▨ Reserved

0–5	6–10	11–15	16–20	21–30	31
31	0 0 0 0 0	0 0 0 0 0	B	434	0

```
EA ← (rB)
if SLB entry exists for EA, then
  SLB entry ← invalid
```

EA is the contents of **r**B. If the segment lookaside buffer (SLB) contains an entry corresponding to EA, that entry is made invalid (that is, removed from the SLB).

The SLB search is done regardless of the settings of MSR[IR] and MSR[DR].

Block address translation for EA, if any, is ignored.

This instruction is supervisor-level and optional in the PowerPC architecture.

This instruction is defined only for 64-bit implementations. Using it on a 32-bit implementation will cause an illegal instruction type program exception.

It is not necessary that the ASR point to a valid segment table when issuing **slbie**.

Note that bits 11–15 of this instruction (ordinarily the position of an **r**A field) must be zero. This provides implementations the option of using (**r**A|0) + **r**B address arithmetic for this instruction.

Other registers altered:

- None

PowerPC Architecture Level	Supervisor Level	32-Bit	64-Bit	64-Bit Bridge	Optional	Form
OEA	Đ		Đ		Đ	X

sld*x* 64-Bit Implementations Only **sld*x***

Shift Left Double Word (x’7C00 0036’)

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

31	S	A	B	27	Rc
0 – 5	6 – 10	11 – 15	16 – 20	21 – 30	31

```
n ← rB[58-63]
r ← ROTL[64](rS, n)
if rB[57] = 0 then
  m ← MASK(0, 63 - n)
else m ← (64)0
rA ← r & m
```

The contents of **r**S are shifted left the number of bits specified by the low-order seven bits of **r**B. Bits shifted out of position 0 are lost. Zeros are supplied to the vacated positions on the right. The result is placed into **r**A. Shift amounts from 64 to 127 give a zero result.

This instruction is defined only for 64-bit implementations. Using it on a 32-bit implementation will cause the system illegal instruction error handler to be invoked.

Other registers altered:

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

PowerPC Architecture Level	Supervisor Level	32-Bit	64-Bit	64-Bit Bridge	Optional	Form
UISA			Ð			X

slw*x* slw*x*

Shift Left Word (x’7C00 0030’)

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

[POWER mnemonics: **sl, sl.**]

31	S	A	B	24	Rc
0 5	6 10	11 15	16 20	21 30	31

```
n← rB[59-6327-31]
r ← ROTL[32](rS[32-63], n)
if rB[58] = 0 then
m ← MASK(32, 63 - n)
else m ← (64)0
rA ← r & m
```

The contents of the low-order 32 bits of **r**S are shifted left the number of bits specified by the low-order six bits of **r**B. Bits shifted out of position 32 are lost. Zeros are supplied to the vacated positions on the right. The 32-bit result is placed into the low-order 32 bits of **r**A. The high-order 32 bits of **r**A are cleared. Shift amounts from 32 to 63 give a zero result.

If bit 26 of **r**B = 0, the contents of **r**S are shifted left the number of bits specified by **r**B[27–31]. Bits shifted out of position 0 are lost. Zeros are supplied to the vacated positions on the right. The 32-bit result is placed into **r**A. If bit 26 of **r**B = 1, 32 zeros are placed into **r**A.

Other registers altered:

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

PowerPC Architecture Level	Supervisor Level	32-Bit	64-Bit	64-Bit Bridge	Optional	Form
UISA						X

srad*x* 64-Bit Implementations Only **srad***x*

Shift Right Algebraic Double Word (x’7C00 0634’)

srad	**rA,rS,rB**	(Rc = 0)
srad.	**rA,rS,rB**	(Rc = 1)

31	S	A	B	794	Rc
0 5	6 10	11 15	16 20	21 30	31

```
n ← rB[58-63]
r ← ROTL[64](rS, 64 - n)
if rB[57] = 0 then
    m ← MASK(n, 63)
else m ← (64)0
S ← rS[0]
rA ← (r & m) | (((64)S) & ¬ m)
XER[CA] ← S & ((r & ¬ m) ¦ 0)
```

The contents of **r**S are shifted right the number of bits specified by the low-order seven bits of **r**B. Bits shifted out of position 63 are lost. Bit 0 of **r**S is replicated to fill the vacated positions on the left. The result is placed into **r**A. XER[CA] is set if **r**S is negative and any 1 bits are shifted out of position 63; otherwise XER[CA] is cleared. A shift amount of zero causes **r**A to be set equal to **r**S, and XER[CA] to be cleared. Shift amounts from 64 to 127 give a result of 64 sign bits in **r**A, and cause XER[CA] to receive the sign bit of **r**S.

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

This instruction is defined only for 64-bit implementations. Using it on a 32-bit implementation will cause the system illegal instruction error handler to be invoked.

Other registers altered:

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

PowerPC Architecture Level	Supervisor Level	32-Bit	64-Bit	64-Bit Bridge	Optional	Form
UISA			Đ			X

sradi*x* 64-Bit Implementations Only **sradi*x***

Shift Right Algebraic Double Word Immediate (x’7C00 0674’)

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

31	S	A	sh*	413	sh*	Rc
0 5	6 10	11 15	16 20	21	30	31

***Note:** This is a split field.

```
n ← sh[5] || sh[0-4]
r ← ROTL[64](rS, 64 - n)
m ← MASK(n, 63)
S ← rS[0]
rA ← (r & m) | (((64)S) & ¬ m)
XER[CA] ← S & ((r & ¬ m) ¦ 0)
```

The contents of **r**S are shifted right SH bits. Bits shifted out of position 63 are lost. Bit 0 of **r**S is replicated to fill the vacated positions on the left. The result is placed into **r**A. XER[CA] is set if **r**S is negative and any 1 bits are shifted out of position 63; otherwise XER[CA] is cleared. A shift amount of zero causes **r**A to be set equal to **r**S, and XER[CA] to be cleared.

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

This instruction is defined only for 64-bit implementations. Using it on a 32-bit implementation will cause the system illegal instruction error handler to be invoked.

Other registers altered:

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

PowerPC Architecture Level	Supervisor Level	32-Bit	64-Bit	64-Bit Bridge	Optional	Form
UISA			Ð			XS

sraw*x* **sraw***x*

Shift Right Algebraic Word (x’7C00 0630’)

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

[POWER mnemonics: **sra, sra.**]

31	S	A	B	792	Rc
0 5	6 10	11 15	16 20	21 30	31

```
n← rB[59-6327-31]
r ← ROTL[32](rS[32-63], 64 - n)
if rB[5826] = 0 then
m← MASK(n + 32, 63)
else m← (6432)0
S ← rS[32]
rA ← r & m | (64)S & ¬ m
XER[CA] ← S & (r & ¬ m[32-63] ¦ 0
```

The contents of the low-order 32 bits of **r**S are shifted right the number of bits specified by the low-order six bits of **r**B. Bits shifted out of position 63 are lost. Bit 32 of **r**S is replicated to fill the vacated positions on the left. The 32-bit result is placed into the low-order 32 bits of **r**A. Bit 32 of **r**S is replicated to fill the high-order 32 bits of **r**A. XER[CA] is set if the low-order 32 bits of **r**S contain a negative number and any 1 bits are shifted out of position 63; otherwise XER[CA] is cleared. A shift amount of zero causes **r**A to receive the sign-extended value of the low-order 32 bits of **r**S, and XER[CA] to be cleared. Shift amounts from 32 to 63 give a result of 64 sign bits, and cause XER[CA] to receive the sign bit of the low-order 32 bits of **r**S.If **r**B[26] = 0,then the contents of **r**S are shifted right the number of bits specified by **r**B[27–31]. Bits shifted out of position 31 are lost. The result is padded on the left with sign bits before being placed into **r**A. If **r**B[26] = 1, then **r**A is filled with 32 sign bits (bit 0) from **r**S. CR0 is set based on the value written into **r**A. XER[CA] is set if **r**S contains a negative number and any 1 bits are shifted out of position 31; otherwise XER[CA] is cleared. A shift amount of zero causes XER[CA] to be cleared.

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

Other registers altered:

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

PowerPC Architecture Level	Supervisor Level	32-Bit	64-Bit	64-Bit Bridge	Optional	Form
UISA						X

srawi_x_ **srawi**_x_

Shift Right Algebraic Word Immediate (x’7C00 0670’)

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

[POWER mnemonics: **srai, srai.**]

31	S	A	SH	824	Rc
0 5	6 10	11 15	16 20	21 30	31

```
n ← SH
r ← ROTL[32](rS[32-63], 6432 - n)
m← MASK(n + 32, 63)
S ← rS[32]
rA ← r & m | (64)S & ¬ m
XER[CA] ← S & ((r & ¬ m)[32-63] ¦ 0)
```

The contents of the low-order 32 bits of **r**S are shifted right SH bits. Bits shifted out of position 63 are lost. Bit 32 of **r**S is replicated to fill the vacated positions on the left. The 32-bit result is placed into the low-order 32 bits of **r**A. Bit 32 of **r**S is replicated to fill the high-order 32 bits of **r**A. XER[CA] is set if the low-order 32 bits of **r**S contain a negative number and any 1 bits are shifted out of position 63; otherwise XER[CA] is cleared. A shift amount of zero causes **r**A to receive the sign-extended value of the low-order 32 bits of **r**S, and XER[CA] to be cleared.The contents of **r**S are shifted right the number of bits specified by operand SH. Bits shifted out of position 31 are lost. The shifted value is sign-extended before being placed in **r**A. The 32-bit result is placed into **r**A. XER[CA] is set if **r**S contains a negative number and any 1 bits are shifted out of position 31; otherwise XER[CA] is cleared. A shift amount of zero causes XER[CA] to be cleared.

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

Other registers altered:

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

PowerPC Architecture Level	Supervisor Level	32-Bit	64-Bit	64-Bit Bridge	Optional	Form
UISA						X

srd*x* 64-Bit Implementations Only **srd***x*

Shift Right Double Word (x'7C00 0436')

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

31	S	A	B	539	Rc
0 5	6 10	11 15	16 20	21 30	31

```
n ← rB[58-63]
r ← ROTL[64](rS, 64 - n)
if rB[57] = 0 then
  m ← MASK(n, 63)
else m ← (64)0
rA ← r & m
```

The contents of **r**S are shifted right the number of bits specified by the low-order seven bits of **r**B. Bits shifted out of position 63 are lost. Zeros are supplied to the vacated positions on the left. The result is placed into **r**A. Shift amounts from 64 to 127 give a zero result.

This instruction is defined only for 64-bit implementations. Using it on a 32-bit implementation will cause the system illegal instruction error handler to be invoked.

Other registers altered:

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

PowerPC Architecture Level	Supervisor Level	32-Bit	64-Bit	64-Bit Bridge	Optional	Form
UISA			Ð			X

srw*x* **srw***x*

Shift Right Word (x’7C00 0430’)

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

[POWER mnemonics: **sr, sr.**]

31	S	A	B	536	Rc
0 5	6 10	11 15	16 20	21 30	31

```
n ← rB[58-6327-31]
r ← ROTL[32](rS[32-63], 6432 - n)
if rB[58] = 0 then
  m ← MASK(n + 32, 63)
else m ← (64)0
rA ← r & m
```

The contents of the low-order 32 bits of **r**S are shifted right the number of bits specified by the low-order six bits of **r**B. Bits shifted out of position 6331 are lost. Zeros are supplied to the vacated positions on the left. The 32-bit result is placed into the low-order 32 bits of **r**A. The high-order 32 bits of **r**A are cleared. Shift amounts from 32 to 63 give a zero result.

Other registers altered:

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

PowerPC Architecture Level	Supervisor Level	32-Bit	64-Bit	64-Bit Bridge	Optional	Form
UISA						X

stb stb

Store Byte (x’9800 0000’)

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

38	S	A	d
0 5	6 10	11 15	16 31

```
if rA = 0 then b ← 0
else     b ← (rA)
EA ← b + EXTS(d)
MEM(EA, 1) ← rS[56-6324-31]
```

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

Other registers altered:

- None

PowerPC Architecture Level	Supervisor Level	32-Bit	64-Bit	64-Bit Bridge	Optional	Form
UISA						D

stbu

stbu

Store Byte with Update (x’9C00 0000’)

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

39	S	A	d
0 5	6 10	11 15	16 31

```
EA ← (rA) + EXTS(d)
MEM(EA, 1) ← rS[56-6324-31]
rA ← EA
```

EA is the sum (**r**A) + d. The contents of the low-order eight bits of **r**S are stored into the byte in memory addressed by EA.

EA is placed into **r**A.

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

Other registers altered:

- None

PowerPC Architecture Level	Supervisor Level	32-Bit	64-Bit	64-Bit Bridge	Optional	Form
UISA						D

stbux stbux

Store Byte with Update Indexed (x’7C00 01EE’)

stbux **rS,rA,rB**

▨ Reserved

31	S	A	B	247	0
0 5	6 10	11 15	16 21	22 30	31

```
EA ← (rA) + (rB)
MEM(EA, 1) ← rS[56-6324-31]
rA ← EA
```

EA is the sum (**r**A) + (**r**B). The contents of the low-order eight bits of **r**S are stored into the byte in memory addressed by EA.

EA is placed into **r**A.

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

Other registers altered:

- None

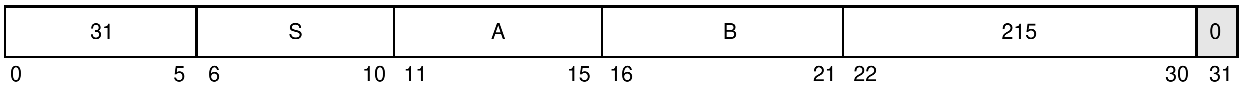
PowerPC Architecture Level	Supervisor Level	32-Bit	64-Bit	64-Bit Bridge	Optional	Form
UISA						X

stbx stbx

Store Byte Indexed (x’7C00 01AE’)

stbx **rS,rA,rB**

▨ Reserved



```
if rA = 0 then b ← 0
else     b ← (rA)
EA ← b + (rB)
MEM(EA, 1) ← rS[56-6324-31]
```

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

Other registers altered:

- None

PowerPC Architecture Level	Supervisor Level	32-Bit	64-Bit	64-Bit Bridge	Optional	Form
UISA						X

std 64-Bit Implementations Only std

Store Double Word (x’F800 0000’)

std **r**S,ds(**r**A)

62	S	A	ds	0 0
0 5	6 10	11 15	16 29	30 31

```
if rA = 0 then b ← 0
else     b ← (rA)
EA ← b + EXTS(ds || 0b00)
(MEM(EA, 8)) ← (rS)
```

EA is the sum (**r**A|0) + (ds || 0b00). The contents of **r**S are stored into the double word in memory addressed by EA.

This instruction is defined only for 64-bit implementations. Using it on a 32-bit implementation will cause the system illegal instruction error handler to be invoked.

Other registers altered:

- None

PowerPC Architecture Level	Supervisor Level	32-Bit	64-Bit	64-Bit Bridge	Optional	Form
UISA			Ð			DS

stdcx. 64-Bit Implementations Only stdcx.

Store Double Word Conditional Indexed (x’7C00 01AD’)

stdcx. **r**S,**r**A,**r**B

31	S	A	B	214	1
0 5	6 10	11 15	16 20	21 30	31

```
if rA = 0 then b ← 0
else      b ← (rA)
EA ← b + (rB)
if RESERVE then
  if RESERVE_ADDR = physical_addr(EA)
    MEM(EA, 8) ← (rS)
    CR0 ← 0b00 || 0b1 || XER[SO]
  else
    u ← undefined 1-bit value
    if u then MEM(EA, 8) ← (rS)
    CR0 ← 0b00 || u || XER[SO]
  RESERVE ← 0
else
  CR0 ← 0b00 || 0b0 || XER[SO]
```

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

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

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

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

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

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

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

Note that, when used correctly, the load and reserve and store conditional instructions can provide an atomic update function for a single aligned word (load word and reserve and store word conditional) or double word (load double word and reserve and store double word conditional) of memory.

In general, correct use requires that load word and reserve be paired with store word conditional, and load double word and reserve with store double word conditional, with the same memory address specified by both instructions of the pair. The only exception is that an unpaired store word conditional or store double word conditional instruction to any (scratch) EA can be used to clear any reservation held by the processor. Examples of correct uses of these instructions, to emulate primitives such as fetch and add, test and set, and compare and swap can be found in Appendix E. , “Synchronization Programming Examples.”

A reservation is cleared if any of the following events occurs:

- The processor holding the reservation executes another load and reserve instruction; this clears the first reservation and establishes a new one.
- The processor holding the reservation executes a store conditional instruction to any address.
- Another processor executes any store instruction to the address associated with the reservation.]
- Any mechanism, other than the processor holding the reservation, stores to the address associated with the reservation.

This instruction is defined only for 64-bit implementations. Using it on a 32-bit implementation will cause the system illegal instruction error handler to be invoked.

Other registers altered:

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

PowerPC Architecture Level	Supervisor Level	32-Bit	64-Bit	64-Bit Bridge	Optional	Form
UISA			Ð			X

stdu 64-Bit Implementations Only stdu

Store Double Word with Update (x’F800 0001’)

stdu **r**S,ds(**r**A)

62	S	A	ds	0 1
0 5	6 10	11 15	16 29	30 31

```
EA ← (rA) + EXTS(ds || 0b00)
(MEM(EA, 8)) ← (rS)
rA ← EA
```

EA is the sum (**r**A) + (ds || 0b00). The contents of **r**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.

This instruction is defined only for 64-bit implementations. Using it on a 32-bit implementation will cause the system illegal instruction error handler to be invoked.

Other registers altered:

- None

PowerPC Architecture Level	Supervisor Level	32-Bit	64-Bit	64-Bit Bridge	Optional	Form
UISA			Ð			DS

stdux 64-Bit Implementations Only stdux

Store Double Word with Update Indexed (x’7C00 016A’)

stdux **rS,rA,rB**

▨ Reserved

31	S	A	B	181	0
0 5	6 10	11 15	16 20	21 30	31

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

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

This instruction is defined only for 64-bit implementations. Using it on a 32-bit implementation will cause the system illegal instruction error handler to be invoked.

Other registers altered:

- None

PowerPC Architecture Level	Supervisor Level	32-Bit	64-Bit	64-Bit Bridge	Optional	Form
UISA			Đ			X

stdx — 64-Bit Implementations Only — stdx

Store Double Word Indexed (x’7C00 012A’)

stdx **r**S,**r**A,**r**B

▨ Reserved

31	S	A	B	149	▨ 0
0 – 5	6 – 10	11 – 15	16 – 20	21 – 30	31

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

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

This instruction is defined only for 64-bit implementations. Using it on a 32-bit implementation will cause the system illegal instruction error handler to be invoked.

Other registers altered:

- None

PowerPC Architecture Level	Supervisor Level	32-Bit	64-Bit	64-Bit Bridge	Optional	Form
UISA			Ð			X

stfd

Store Floating-Point Double (x’D800 0000’)

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

54	S	A	d
0 5	6 10	11 15	16 30 31

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

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

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

Other registers altered:

- None

PowerPC Architecture Level	Supervisor Level	32-Bit	64-Bit	64-Bit Bridge	Optional	Form
UISA						D

stfdu

Store Floating-Point Double with Update (x’DC00 0000’)

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

55	S	A	d
0 5	6 10	11 15	16 31

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

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

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

EA is placed into **r**A.

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

Other registers altered:

- None

PowerPC Architecture Level	Supervisor Level	32-Bit	64-Bit	64-Bit Bridge	Optional	Form
UISA						D

stfdux stfdux

Store Floating-Point Double with Update Indexed (x’7C00 05EE’)

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

▨ Reserved

31	S	A	B	759	0
0 5	6 10	11 15	16 20	21 30	31

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

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

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

EA is placed into **r**A.

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

Other registers altered:

- None

PowerPC Architecture Level	Supervisor Level	32-Bit	64-Bit	64-Bit Bridge	Optional	Form
UISA						X

stfdx stfdx

Store Floating-Point Double Indexed (x’7C00 05AE’)

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

▨ Reserved

31	S	A	B	727	0
0 5	6 10	11 15	16 20	21 30	31

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

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

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

Other registers altered:

- None

PowerPC Architecture Level	Supervisor Level	32-Bit	64-Bit	64-Bit Bridge	Optional	Form
UISA						X

stfiwx stfiwx

Store Floating-Point as Integer Word Indexed (x’7C00 07AE’)

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

▨ Reserved

31	S	A	B	983	▨ 0
0 5	6 10	11 15	16 20	21 30	31

```
if rA = 0 then b ← 0
else     b ← (rA)
EA ← b + (rB)
MEM(EA, 4) ← frS[32-63]
```

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

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

If the contents of register **fr**S were produced, either directly or indirectly, by an **lfs** instruction, a single-precision arithmetic instruction, or **frsp**, then the value stored is 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.

This instruction is defined as optional by the PowerPC architecture to ensure backwards compatibility with earlier processors; however, it will likely be required for subsequent PowerPC processors.

Other registers altered:

- None

PowerPC Architecture Level	Supervisor Level	32-Bit	64-Bit	64-Bit Bridge	Optional	Form
UISA					Ð	X

stfs stfs

Store Floating-Point Single (x’D000 0000’)

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

52	S	A	d
0 5	6 10	11 15	16 31

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

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

The contents of register **fr**S are converted to single-precision and stored into the word in memory addressed by EA. Note that the value to be stored should be in single-precision format prior to the execution of the **stfs** instruction. For a discussion on floating-point store conversions, see Section D.7 , “Floating-Point Store Instructions.”

Other registers altered:

- None

PowerPC Architecture Level	Supervisor Level	32-Bit	64-Bit	64-Bit Bridge	Optional	Form
UISA						D

stfsu

Store Floating-Point Single with Update (x’D400 0000’)

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

53	S	A	d
0–5	6–10	11–15	16–31

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

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

The contents of **fr**S are converted to single-precision and stored into the word in memory addressed by EA. Note that the value to be stored should be in single-precision format prior to the execution of the **stfsu** instruction. For a discussion on floating-point store conversions, see Section D.7 , “Floating-Point Store Instructions.”

EA is placed into **r**A.

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

Other registers altered:

- None

PowerPC Architecture Level	Supervisor Level	32-Bit	64-Bit	64-Bit Bridge	Optional	Form
UISA						D

stfsux stfsux

Store Floating-Point Single with Update Indexed (x’7C00 056E’)

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

▨ Reserved

0–5	6–10	11–15	16–20	21–30	31
31	S	A	B	695	▨ 0

```
EA ← (rA) + (rB)
MEM(EA, 4) ← SINGLE(frS)
rA ← EA
```

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

The contents of **fr**S are converted to single-precision and stored into the word in memory addressed by EA. For a discussion on floating-point store conversions, see Section D.7 , “Floating-Point Store Instructions.”

EA is placed into **r**A.

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

Other registers altered:

- None

PowerPC Architecture Level	Supervisor Level	32-Bit	64-Bit	64-Bit Bridge	Optional	Form
UISA						X

stfsx

Store Floating-Point Single Indexed (x'7C00 052E')

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

▨ Reserved

31	S	A	B	663	▨ 0
0 – 5	6 – 10	11 – 15	16 – 20	21 – 30	31

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

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

The contents of register **fr**S are converted to single-precision and stored into the word in memory addressed by EA. For a discussion on floating-point store conversions, see Section D.7 , "Floating-Point Store Instructions."

Other registers altered:

- None

PowerPC Architecture Level	Supervisor Level	32-Bit	64-Bit	64-Bit Bridge	Optional	Form
UISA						X

sth sth

Store Half Word (x’B000 0000’)

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

44	S	A	d
0 5	6 10	11 15	16 31

```
if rA = 0 then b ← 0
else     b ← (rA)
EA ← b + EXTS(d)
MEM(EA, 2) ← rS[48-6316-31]
```

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

Other registers altered:

- None

PowerPC Architecture Level	Supervisor Level	32-Bit	64-Bit	64-Bit Bridge	Optional	Form
UISA						D

sthbrx sthbrx

Store Half Word Byte-Reverse Indexed (x’7C00 072C’)

sthbrx **rS,rA,rB**

▨ Reserved

31	S	A	B	918	0
0 5	6 10	11 15	16 20	21 30	31

```
if rA = 0 then b ← 0
else     b ← (rA)
EA ← b + (rB)
MEM(EA, 2) ← rS[56-6324-31] || rS[48-5516-23]
```

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

Other registers altered:

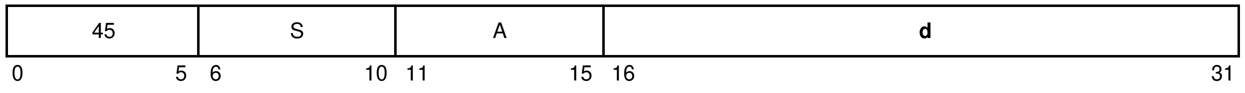
- None

PowerPC Architecture Level	Supervisor Level	32-Bit	64-Bit	64-Bit Bridge	Optional	Form
UISA						X

sthu sthu

Store Half Word with Update (x'B400 0000')

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



```
EA ← (rA) + EXTS(d)
MEM(EA, 2) ← rS[48-6316-31]
rA ← EA
```

EA is the sum (**r**A) + d. The contents of the low-order 16 bits of **r**S are stored into the half word in memory addressed by EA.

EA is placed into **r**A.

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

Other registers altered:

- None

PowerPC Architecture Level	Supervisor Level	32-Bit	64-Bit	64-Bit Bridge	Optional	Form
UISA						D

sthux sthux

Store Half Word with Update Indexed (x'7C00 036E')

sthux **rS,rA,rB**

▨ Reserved

0–5	6–10	11–15	16–20	21–30	31
31	S	A	B	439	▨ 0

```
EA ← (rA) + (rB)
MEM(EA, 2) ← rS[48-6316-31]
rA ← EA
```

EA is the sum (**r**A) + (**r**B). The contents of the low-order 16 bits of **r**S are stored into the half word in memory addressed by EA.

EA is placed into **r**A.

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

Other registers altered:

- None

PowerPC Architecture Level	Supervisor Level	32-Bit	64-Bit	64-Bit Bridge	Optional	Form
UISA						X

sthx sthx

Store Half Word Indexed (x’7C00 032E’)

sthx **rS,rA,rB**

▨ Reserved

0 – 5	6 – 10	11 – 15	16 – 20	21 – 30	31
31	S	A	B	407	0

```
if rA = 0 then b ← 0
else     b ← (rA)
EA ← b + (rB)
MEM(EA, 2) ← rS[48-6316-31]
```

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

Other registers altered:

- None

PowerPC Architecture Level	Supervisor Level	32-Bit	64-Bit	64-Bit Bridge	Optional	Form
UISA						X

stmw

Store Multiple Word (x’BC00 0000’)

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

[POWER mnemonic: **stm**]

47	S	A	d
0 5	6 10	11 15	16 31

```
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)[32-63]
   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 low-order 32 bits of GPRs **r**S through **r31**. For example, if **r**S = 30, 2 words are stored.

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

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

Other registers altered:

- None

PowerPC Architecture Level	Supervisor Level	32-Bit	64-Bit	64-Bit Bridge	Optional	Form
UISA						D

stswi stswi

Store String Word Immediate (x’7C00 05AA’)

stswi **rS,rA,NB**

[POWER mnemonic: **stsi**]

▨ Reserved

31	S	A	NB	725	0
0 5	6 10	11 15	16 20	21 30	31

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

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

n consecutive bytes starting at EA are stored from GPRs **r**S through **r**S + *nr* – 1. Data is stored from the low-order four bytes of each GPR. Bytes are stored left to right from each register. The sequence of registers wraps around through **r0** if required.

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

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

Other registers altered:

- None

PowerPC Architecture Level	Supervisor Level	32-Bit	64-Bit	64-Bit Bridge	Optional	Form
UISA						X

stswx stswx

Store String Word Indexed (x’7C00 052A’)

stswx **rS,rA,rB**

[POWER mnemonic: **stsx**]

▨ Reserved

31	S	A	B	661	0
0 5	6 10	11 15	16 20	21 30	31

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

EA is the sum (**r**A|0) + (**r**B). Let *n* = XER[25–31]; *n* is the number of bytes to store. Let *nr* = CEIL(*n* ÷ 4); *nr* is the number of registers to supply data.

n consecutive bytes starting at EA are stored from GPRs **r**S through **r**S + *nr* – 1. Data is stored from the low-order four bytes of each GPR. Bytes are stored left to right from each register. The sequence of registers wraps around through **r0** if required. If *n* = 0, no bytes are stored.

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

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

Other registers altered:

- None

PowerPC Architecture Level	Supervisor Level	32-Bit	64-Bit	64-Bit Bridge	Optional	Form
UISA						X

stw stw

Store Word (x’9000 0000’)

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

[POWER mnemonic: **st**]

36	S	A	d
0 5	6 10	11 15	16 31

```
if rA = 0 then b ← 0
else      b ← (rA)
EA ← b + EXTS(d)
MEM(EA, 4) ← rS[32-63]
```

EA is the sum (**r**A|0) + d. The contents of the low-order 32 bits of **r**S are stored into the word in memory addressed by EA.

Other registers altered:

- None

PowerPC Architecture Level	Supervisor Level	32-Bit	64-Bit	64-Bit Bridge	Optional	Form
UISA						D

stwbrx stwbrx

Store Word Byte-Reverse Indexed (x’7C00 052C’)

stwbrx **rS,rA,rB**

[POWER mnemonic: **stbrx**]

▨ Reserved

31	S	A	B	662	0
0 5	6 10	11 15	16 20	21 30	31

```
if rA = 0 then b ← 0
else     b ← (rA)
EA ← b + (rB)
MEM(EA, 4) ← rS[56-6324-31] || rS[48-5516-23] || rS[40-478-15] || rS[32-390-7]
```

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

Other registers altered:

- None

PowerPC Architecture Level	Supervisor Level	32-Bit	64-Bit	64-Bit Bridge	Optional	Form
UISA						X

stwcx. stwcx.

Store Word Conditional Indexed (x’7C00 012D’)

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

31	S	A	B	150	1
0 5	6 10	11 15	16 20	21 30	31

```
if rA = 0 then b ← 0
else      b ← (rA)
EA ← b + (rB)
if RESERVE then
  if RESERVE_ADDR = physical_addr(EA)
    MEM(EA, 4) ← rS[32-63]
    CR0 ← 0b00 || 0b1 || XER[SO]
  else
    u ← undefined 1-bit value
    if u then MEM(EA, 4) ← rS[32-63]
    CR0 ← 0b00 || u || XER[SO]
  RESERVE ← 0
else
  CR0 ← 0b00 || 0b0 || XER[SO]
```

EA is the sum (**r**A|0) + (**r**B). If the reserved bit is set, the **stwcx.** instruction stores **r**S to effective address (**r**A + **r**B), clears the reserved bit, and sets CR0[EQ]. If the reserved bit is not set, the **stwcx.** instruction does not do a store; it leaves the reserved bit cleared and clears CR0[EQ]. Software must look at CR0[EQ] to see if the **stwcx.** was successful.

The reserved bit is set by the **lwarx** instruction. The reserved bit is cleared by any **stwcx.** instruction to any address, and also by snooping logic if it detects that another processor does any kind of store to the block indicated in the reservation buffer when reserved is set.

If a reservation exists, and the memory address specified by the **stwcx.** instruction is the same as that specified by the load and reserve instruction that established the reservation, the contents of the low-order 32 bits of **r**S are stored into the word in memory addressed by EA and the reservation is cleared.

If a reservation exists, but the memory address specified by the **stwcx.** instruction is not the same as that specified by the load and reserve instruction that established the reservation, the reservation is cleared, and it is undefined whether the contents of the low-order 32 bits of **r**S are stored into the word in memory addressed by EA.

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

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

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

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

The granularity with which reservations are managed is implementation-dependent. Therefore, the memory to be accessed by the load and reserve and store conditional instructions should be allocated by a system library program.

Other registers altered:

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

PowerPC Architecture Level	Supervisor Level	32-Bit	64-Bit	64-Bit Bridge	Optional	Form
UISA						X

stwu stwu

Store Word with Update (x’9400 0000’)

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

[POWER mnemonic: **stu**]

37	S	A	**d**
0 5	6 10	11 15	16 31

```
EA ← (rA) + EXTS(d)
MEM(EA, 4) ← rS[32-63]
rA ← EA
```

EA is the sum (**r**A) + d. The contents of the low-order 32 bits of **r**S are stored into the word in memory addressed by EA.

EA is placed into **r**A.

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

Other registers altered:

- None

PowerPC Architecture Level	Supervisor Level	32-Bit	64-Bit	64-Bit Bridge	Optional	Form
UISA						D

stwux stwux

Store Word with Update Indexed (x’7C00 016E’)

stwux **rS,rA,rB**

[POWER mnemonic: **stux**]

▨ Reserved

31	S	A	B	183	▨ 0
0 5	6 10	11 15	16 20	21 30	31

```
EA ← (rA) + (rB)
MEM(EA, 4) ← rS[32-63]
rA ← EA
```

EA is the sum (**r**A) + (**r**B). The contents of the low-order 32 bits of **r**S are stored into the word in memory addressed by EA.

EA is placed into **r**A.

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

Other registers altered:

- None

PowerPC Architecture Level	Supervisor Level	32-Bit	64-Bit	64-Bit Bridge	Optional	Form
UISA						X

stwx stwx

Store Word Indexed (x’7C00 012E’)

stwx **rS,rA,rB**

[POWER mnemonic: **stx**]

▨ Reserved

31	S	A	B	151	0
0 5	6 10	11 15	16 20	21 30	31

```
if rA = 0 then b ← 0
else      b ← (rA)
EA ← b + (rB)
MEM(EA, 4) ← rS[32-63]
```

EA is the sum (**r**A|0) + (**r**B). The contents of the low-order 32 bits of **r**S are is stored into the word in memory addressed by EA.

Other registers altered:

- None

PowerPC Architecture Level	Supervisor Level	32-Bit	64-Bit	64-Bit Bridge	Optional	Form
UISA						X

subf*x* **subf***x*

Subtract From (x’7C00 0050’)

subf	**r**D,**r**A,**r**B	(OE = 0 Rc = 0)
subf.	**r**D,**r**A,**r**B	(OE = 0 Rc = 1)
subfo	**r**D,**r**A,**r**B	(OE = 1 Rc = 0)
subfo.	**r**D,**r**A,**r**B	(OE = 1 Rc = 1)

31	D	A	B	OE	40	Rc
0 5	6 10	11 15	16 20	21	22 30	31

```
rD ← ¬ (rA) + (rB) + 1
```

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

The **subf** instruction is preferred for subtraction because it sets few status bits.

Other registers altered:

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

Simplified mnemonics:

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

PowerPC Architecture Level	Supervisor Level	32-Bit	64-Bit	64-Bit Bridge	Optional	Form
UISA						XO

subfc*x* subfc*x*

Subtract from Carrying (x’7C00 0010’)

subfc	**r**D,**r**A,**r**B	(OE = 0 Rc = 0)
subfc.	**r**D,**r**A,**r**B	(OE = 0 Rc = 1)
subfco	**r**D,**r**A,**r**B	(OE = 1 Rc = 0)
subfco.	**r**D,**r**A,**r**B	(OE = 1 Rc = 1)

[POWER mnemonics: **sf, sf., sfo, sfo.**]

31	D	A	B	OE	8	Rc
0 5	6 10	11 15	16 20	21	22 30	31

```
rD ← ¬ (rA) + (rB) + 1
```

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

Other registers altered:

- Condition Register (CR0 field):
  Affected: LT, GT, EQ, SO (if Rc = 1)
  Note: CR0 field may not reflect the infinitely precise result if overflow occurs (see XER below).
- XER:
  Affected: CA
  Affected: SO, OV (if OE = 1)
  Note: The setting of the affected bits in the XER is mode-dependent, and reflects overflow of the 64-bit result in 64-bit mode and overflow of the low-order 32-bit result in 32-bit mode. For further information about 64-bit mode and 32-bit mode in 64-bit implementations, see 3. , “Operand Conventions.”

Simplified mnemonics:

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

PowerPC Architecture Level	Supervisor Level	32-Bit	64-Bit	64-Bit Bridge	Optional	Form
UISA						XO

subfe*x* **subfe***x*

Subtract from Extended (x'7C00 0110')

subfe	**r**D,**r**A,**r**B	(OE = 0 Rc = 0)
subfe.	**r**D,**r**A,**r**B	(OE = 0 Rc = 1)
subfeo	**r**D,**r**A,**r**B	(OE = 1 Rc = 0)
subfeo.	**r**D,**r**A,**r**B	(OE = 1 Rc = 1)

[POWER mnemonics: **sfe, sfe., sfeo, sfeo.**]

31	D	A	B	OE	136	Rc
0 5	6 10	11 15	16 20	21	22 30	31

```
rD ← ¬ (rA) + (rB) + XER[CA]
```

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

Other registers altered:

- Condition Register (CR0 field):
  Affected: LT, GT, EQ, SO(if Rc = 1)
  Note: CR0 field may not reflect the infinitely precise result if overflow occurs (see XER below).
- XER:
  Affected: CA
  Affected: SO, OV(if OE = 1)
  Note: The setting of the affected bits in the XER is mode-dependent, and reflects overflow of the 64-bit result in 64-bit mode and overflow of the low-order 32-bit result in 32-bit mode. For further information about 64-bit mode and 32-bit mode in 64-bit implementations, see 3. , "Operand Conventions."

PowerPC Architecture Level	Supervisor Level	32-Bit	64-Bit	64-Bit Bridge	Optional	Form
UISA						XO

subfic subfic

Subtract from Immediate Carrying (x’2000 0000’)

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

[POWER mnemonic: **sfi**]

08	D	A	SIMM
0 5	6 10	11 15	16 31

```
rD ← ¬ (rA) + EXTS(SIMM) + 1
```

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

Other registers altered:

- XER:
  Affected: CA
  Note: The setting of the affected bits in the XER is mode-dependent, and reflects overflow of the 64-bit result in 64-bit mode and overflow of the low-order 32-bit result in 32-bit mode. For further information about 64-bit mode and 32-bit mode in 64-bit implementations, see 3. , “Operand Conventions.”

PowerPC Architecture Level	Supervisor Level	32-Bit	64-Bit	64-Bit Bridge	Optional	Form
UISA						D

subfme*x* **subfme***x*

Subtract from Minus One Extended (x’7C00 01D0’)

subfme	**r**D,**r**A	(OE = 0 Rc = 0)
subfme.	**r**D,**r**A	(OE = 0 Rc = 1)
subfmeo	**r**D,**r**A	(OE = 1 Rc = 0)
subfmeo.	**r**D,**r**A	(OE = 1 Rc = 1)

[POWER mnemonics: **sfme, sfme., sfmeo, sfmeo.**]

▨ Reserved

31	D	A	0 0 0 0 0	OE	232	Rc
0 5	6 10	11 15	16 20	21	22 30	31

```
rD ← ¬ (rA) + XER[CA] - 1
```

The sum ¬ (**r**A) + XER[CA] + (6432)1 is placed into **r**D.

Other registers altered:

- Condition Register (CR0 field):
  Affected: LT, GT, EQ, SO(if Rc = 1)
  Note: CR0 field may not reflect the infinitely precise result if overflow occurs (see XER below).
- XER:
  Affected: CA
  Affected: SO, OV(if OE = 1)
  Note: The setting of the affected bits in the XER is mode-dependent, and reflects overflow of the 64-bit result in 64-bit mode and overflow of the low-order 32-bit result in 32-bit mode. For further information about 64-bit mode and 32-bit mode in 64-bit implementations, see 3. , “Operand Conventions.”

PowerPC Architecture Level	Supervisor Level	32-Bit	64-Bit	64-Bit Bridge	Optional	Form
UISA						XO

subfze*x* **subfze***x*

Subtract from Zero Extended (x’7C00 0190’)

subfze	**r**D,**r**A	(OE = 0 Rc = 0)
subfze.	**r**D,**r**A	(OE = 0 Rc = 1)
subfzeo	**r**D,**r**A	(OE = 1 Rc = 0)
subfzeo.	**r**D,**r**A	(OE = 1 Rc = 1)

[POWER mnemonics: **sfze, sfze., sfzeo, sfzeo.**]

▨ Reserved

31	D	A	0 0 0 0 0	OE	200	Rc
0 5	6 10	11 15	16 20	21	22 30	31

```
rD ← ¬ (rA) + XER[CA]
```

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

Other registers altered:

- Condition Register (CR0 field):
  Affected: LT, GT, EQ, SO(if Rc = 1)
  Note: CR0 field may not reflect the infinitely precise result if overflow occurs (see XER below).
- XER:
  Affected: CA
  Affected: SO, OV(if OE = 1)
  Note: The setting of the affected bits in the XER is mode-dependent, and reflects overflow of the 64-bit result in 64-bit mode and overflow of the low-order 32-bit result in 32-bit mode. For further information about 64-bit mode and 32-bit mode in 64-bit implementations, see 3. , “Operand Conventions.”

PowerPC Architecture Level	Supervisor Level	32-Bit	64-Bit	64-Bit Bridge	Optional	Form
UISA						XO

sync

sync

Synchronize (x’7C00 04AC’)

[POWER mnemonic: **dcs**]

▨ Reserved

31	00 000 ▨	0 0000 ▨	0000 0 ▨	598	0 ▨
0 5	6 10	11 15	16 20	21 30	31

The **sync** instruction provides an ordering function for the effects of all instructions executed by a given processor. Executing a **sync** instruction ensures that all instructions preceding the **sync** instruction appear to have completed before the **sync** instruction completes, and that no subsequent instructions are initiated by the processor until after the **sync** instruction completes. When the **sync** instruction completes, all external accesses caused by instructions preceding the **sync** instruction will have been performed with respect to all other mechanisms that access memory. For more information on how the **sync** instruction affects the VEA, refer to 5. , “Cache Model and Memory Coherency.”

Multiprocessor implementations also send a **sync** address-only broadcast that is useful in some designs. For example, if a design has an external buffer that re-orders loads and stores for better bus efficiency, the **sync** broadcast signals to that buffer that previous loads/stores must be completed before any following loads/stores.

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

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

The **eieio** instruction may be more appropriate than **sync** for many cases.

This instruction is execution synchronizing. For more information on execution synchronization, see Section 4.1.5 , “Synchronizing Instructions.”

Other registers altered:

- None

PowerPC Architecture Level	Supervisor Level	32-Bit	64-Bit	64-Bit Bridge	Optional	Form
UISA						X

td 64-Bit Implementations Only td

Trap Double Word (x’7C00 0088’)

td TO,**r**A,**r**B

▨ Reserved

31	TO	A	B	68	0 ▨
0 5	6 10	11 15	16 20	21 30	31

```
a ← (rA)
b ← (rB)
if (a < b) & TO[0] then TRAP
if (a > b) & TO[1] then TRAP
if (a = b) & TO[2] then TRAP
if (a <U b) & TO[3] then TRAP
if (a >U b) & TO[4] then TRAP
```

The contents of **r**A are compared with the contents of **r**B. If any bit in the TO field is set and its corresponding condition is met by the result of the comparison, then the system trap handler is invoked.

This instruction is defined only for 64-bit implementations. Using it on a 32-bit implementation will cause the system illegal instruction error handler to be invoked.

Other registers altered:

- None

Simplified mnemonics:

tdge	**r**A,**r**B	equivalent to	**td**	**12,r**A,**r**B
tdlnl	**r**A,**r**B	equivalent to	**td**	**5,r**A,**r**B

PowerPC Architecture Level	Supervisor Level	32-Bit	64-Bit	64-Bit Bridge	Optional	Form
UISA			Ð			X

tdi 64-Bit Implementations Only tdi

Trap Double Word Immediate (x’0800 0000’)

tdi TO,**r**A,SIMM

02	TO	A	SIMM
0 5	6 10	11 15	16 31

```
a ← (rA)
if (a < EXTS(SIMM)) & TO[0] then TRAP
if (a > EXTS(SIMM)) & TO[1] then TRAP
if (a = EXTS(SIMM)) & TO[2] then TRAP
if (a <U EXTS(SIMM)) & TO[3] then TRAP
if (a >U EXTS(SIMM)) & TO[4] then TRAP
```

The contents of **r**A are compared with the sign-extended value of the SIMM field. If any bit in the TO field is set and its corresponding condition is met by the result of the comparison, then the system trap handler is invoked.

This instruction is defined only for 64-bit implementations. Using it on a 32-bit implementation will cause the system illegal instruction error handler to be invoked.

Other registers altered:

- None

Simplified mnemonics:

tdlti	**r**A,value	equivalent to	**tdi**	**16,r**A,value
tdnei	**r**A,value	equivalent to	**tdi**	**24,r**A,value

PowerPC Architecture Level	Supervisor Level	32-Bit	64-Bit	64-Bit Bridge	Optional	Form
UISA			Ð			D

tlbia

Translation Lookaside Buffer Invalidate All (x’7C00 02E4’)

▨ Reserved

0–5	6–10	11–15	16–20	21–30	31
31	0 0 0 0 0	0 0 0 0 0	0 0 0 0 0	370	0

```
All TLB entries ← invalid
```

The entire translation lookaside buffer (TLB) is invalidated (that is, all entries are removed).

The TLB is invalidated regardless of the settings of MSR[IR] and MSR[DR]. The invalidation is done without reference to the SLB, segment table, or segment registers.

This instruction does not cause the entries to be invalidated in other processors.

This is a supervisor-level instruction and optional in the PowerPC architecture.

Other registers altered:

- None

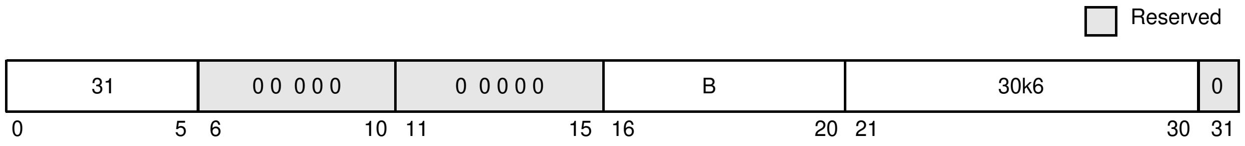
PowerPC Architecture Level	Supervisor Level	32-Bit	64-Bit	64-Bit Bridge	Optional	Form
OEA	Ð				Ð	X

tlbie tlbie

Translation Lookaside Buffer Invalidate Entry (x’7C00 0264’)

tlbie **rB**

[POWER mnemonic: **tlbi**]



```
VPS ← rB[36–514-19]
Identify TLB entries corresponding to VPS
Each such TLB entry ← invalid
```

EA is the contents of **r**B. If the translation lookaside buffer (TLB) contains an entry corresponding to EA, that entry is made invalid (that is, removed from the TLB).

Multiprocessing implementations (for example, the 601, and 604) send a **tlbie** address-only broadcast over the address bus to tell other processors to invalidate the same TLB entry in their TLBs.

The TLB search is done regardless of the settings of MSR[IR] and MSR[DR]. The search is done based on a portion of the logical page number within a segment, without reference to the SLB, segment table, or segment registers. All entries matching the search criteria are invalidated.

Block address translation for EA, if any, is ignored. Refer to Section 7.5.3.4 , “Synchronization of Memory Accesses and Referenced and Changed Bit Updates,” and Section 7.6.3 , “Page Table Updates,” for other requirements associated with the use of this instruction.

This is a supervisor-level instruction and optional in the PowerPC architecture.

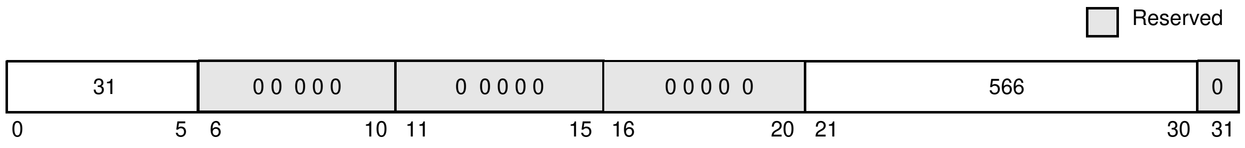
Other registers altered:

- None

PowerPC Architecture Level	Supervisor Level	32-Bit	64-Bit	64-Bit Bridge	Optional	Form
OEA	Ð				Ð	X

tlbsync tlbsync

TLB Synchronize (x’7C00 046C’)



If an implementation sends a broadcast for **tlbie** then it will also send a broadcast for **tlbsync**. Executing a **tlbsync** instruction ensures that all **tlbie** instructions previously executed by the processor executing the **tlbsync** instruction have completed on all other processors.

The operation performed by this instruction is treated as a caching-inhibited and guarded data access with respect to the ordering done by **eieio**.

Note that the 601 expands the use of the **sync** instruction to cover **tlbsync** functionality.

Refer to Section 7.5.3.4 , “Synchronization of Memory Accesses and Referenced and Changed Bit Updates,” and Section 7.6.3 , “Page Table Updates,” for other requirements associated with the use of this instruction.

This instruction is supervisor-level and optional in the PowerPC architecture.

Other registers altered:

- None

PowerPC Architecture Level	Supervisor Level	32-Bit	64-Bit	64-Bit Bridge	Optional	Form
OEA	Ð				Ð	X

tw tw

Trap Word (x’7C00 0008’)

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

[POWER mnemonic: **t**]

▨ Reserved

31	TO	A	B	4	▨ 0
0 5	6 10	11 15	16 20	21 30	31

```
a ← EXTS(rA[32–63])
b ← EXTS(rB[32–63])
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 the low-order 32 bits of **r**A are compared with the contents of the low-order 32 bits of **r**B. If any bit in the TO field is set and its corresponding condition is met by the result of the comparison, then the system trap handler is invoked.

Other registers altered:

- None

Simplified mnemonics:

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

PowerPC Architecture Level	Supervisor Level	32-Bit	64-Bit	64-Bit Bridge	Optional	Form
UISA						X

twi twi

Trap Word Immediate (x’0C00 0000’)

twi TO,**r**A,SIMM

[POWER mnemonic: **ti**]

03	TO	A	SIMM
0 5	6 10	11 15	16 31

```
a ← EXTS(rA[32-63])
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 the low-order 32 bits of **r**A are compared with the sign-extended value of the SIMM field. If any bit in the TO field is set and its corresponding condition is met by the result of the comparison, then the system trap handler is invoked.

Other registers altered:

- None

Simplified mnemonics:

twgti	**r**A,value	equivalent to	**twi**	**8**,**r**A,value
twllei	**r**A,value	equivalent to	**twi**	**6**,**r**A,value

PowerPC Architecture Level	Supervisor Level	32-Bit	64-Bit	64-Bit Bridge	Optional	Form
UISA						D

xor*x* xor*x*

XOR (x'7C00 0278')

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

31	S	A	B	316	Rc
0 5	6 10	11 15	16 20	21 30	31

```
rA ← (rS) ⊕ (rB)
```

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

Other registers altered:

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

PowerPC Architecture Level	Supervisor Level	32-Bit	64-Bit	64-Bit Bridge	Optional	Form
UISA						X

xori xori

XOR Immediate (x’6800 0000’)

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

[POWER mnemonic: **xoril**]

26	S	A	UIMM
0 5	6 10	11 15	16 31

```
rA ← (rS) ⊕ ((4816)0 || UIMM)
```

The contents of **r**S are XORed with 0x0000_0000_0000 || UIMM and the result is placed into **r**A.

Other registers altered:

- None

PowerPC Architecture Level	Supervisor Level	32-Bit	64-Bit	64-Bit Bridge	Optional	Form
UISA						D

xoris

XOR Immediate Shifted (x'6C00 0000')

xoris **rA,rS,UIMM**

[POWER mnemonic: **xoriu**]

27	S	A	UIMM
0 5	6 10	11 15	16 31

```
rA ← (rS) ⊕ ((32)0 || UIMM || (16)0)
```

The contents of **r**S are XORed with 0x0000_0000 || UIMM || 0x0000 and the result is placed into **r**A.

Other registers altered:

- None

PowerPC Architecture Level	Supervisor Level	32-Bit	64-Bit	64-Bit Bridge	Optional	Form
UISA						D