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)

31	D	A	B	OE	266	Rc
0 5	6 10	11 15	16 20	21	22 30	31

```
rD← (rA) + (rB)
```

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

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

Other registers altered:

- Condition Register (CR0 field):

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

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

- XER:

  Affected: SO, OV (If OE = 1)

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

PowerPC Architecture Level	Supervisor Level	PowerPC Optional	Form
UISA			XO

addi addi

Add Immediate (x’3800 0000’)

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

14	D	A	SIMM
0 5	6 10	11 15	16 31

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

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

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

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

Other registers altered:

- None

Simplified mnemonics:

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

PowerPC Architecture Level	Supervisor Level	PowerPC Optional	Form
UISA			D

addis

Add Immediate Shifted (x’3C00 0000’)

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

15	D	A	SIMM
0 5	6 10	11 15	16 31

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

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

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

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

Other registers altered:

- None

Simplified mnemonics:

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

PowerPC Architecture Level	Supervisor Level	PowerPC Optional	Form
UISA			D

and*x*

AND (x’7C00 0038’)

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

31	S	A	B	28	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 result is placed into **r**A.

Other registers altered:

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

PowerPC Architecture Level	Supervisor Level	PowerPC Optional	Form
UISA			X

andi.

AND Immediate (x’7000 0000’)

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

28	S	A	UIMM
0 5	6 10	11 15	16 31

```
rA← (rS) & ((16)0 || UIMM)
```

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

Other registers altered:

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

PowerPC Architecture Level	Supervisor Level	PowerPC Optional	Form
UISA			D

andis. andis.

AND Immediate Shifted (x’7400 0000’)

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

29	S	A	UIMM
0 5	6 10	11 15	16 31

```
rA ← (rS) & (UIMM || (16)0)
```

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

Other registers altered:

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

PowerPC Architecture Level	Supervisor Level	PowerPC Optional	Form
UISA			D

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)

18	LI	AA	LK
0 5	6 29	30	31

```
if AA = 1
        then NIA $\leftarrow_{iea}$ EXTS(LI || 0b00)
        else NIA $\leftarrow_{iea}$ CIA + EXTS(LI || 0b00)
if LK = 1
        then LR $\leftarrow_{iea}$ CIA + 4
```

target_addr specifies the branch target address.

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

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

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

Other registers altered:

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

PowerPC Architecture Level	Supervisor Level	PowerPC Optional	Form
UISA			I

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)

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

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

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

Table 8-6. BO Operand Encodings

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

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

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

target_addr specifies the branch target address.

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

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

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

Other registers altered:

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

Simplified mnemonics:

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

PowerPC Architecture Level	Supervisor Level	PowerPC Optional	Form
UISA			D

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

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

▨ Reserved

19	BO	BI	0 0 0 0 0	16	LK
0 5	6 10	11 15	16 20	21 30	31

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

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

Table 8-8. BO Operand Encodings

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

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

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

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

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

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

Other registers altered:

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

Simplified mnemonics:

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

PowerPC Architecture Level	Supervisor Level	PowerPC Optional	Form
UISA			XL

cmp

Compare (x'7C00 0000')

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

▨ Reserved

31	crfD	▨ 0	L	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

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

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

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

Other registers altered:

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

  Affected: LT, GT, EQ, SO

Simplified mnemonics:

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

PowerPC Architecture Level	Supervisor Level	PowerPC Optional	Form
UISA			X

cmpi cmpi

Compare Immediate (x’2C00 0000’)

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

▨ Reserved

11	crfD	▨ 0	L	A	SIMM
0 5	6 8	9	10	11 15	16 31

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

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

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

Other registers altered:

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

  Affected: LT, GT, EQ, SO

Simplified mnemonics:

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

PowerPC Architecture Level	Supervisor Level	PowerPC Optional	Form
UISA			D

cmpl

Compare Logical (x’7C00 0040’)

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

▨ Reserved

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

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

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

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

Other registers altered:

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

  Affected: LT, GT, EQ, SO

Simplified mnemonics:

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

PowerPC Architecture Level	Supervisor Level	PowerPC Optional	Form
UISA			X

cmpli cmpli

Compare Logical Immediate (x’2800 0000’)

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

▨ Reserved

10	crfD	0	L	A	UIMM
0 5	6 8	9	10	11 15	16 31

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

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

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

Other registers altered:

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

  Affected: LT, GT, EQ, SO

Simplified mnemonics:

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

PowerPC Architecture Level	Supervisor Level	PowerPC Optional	Form
UISA			D

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

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

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

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

divw	**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)

- XER:

  Affected: SO, OV (If OE = 1)

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

PowerPC Architecture Level	Supervisor Level	PowerPC Optional	Form
UISA			XO

divwu*x*

Divide Word Unsigned (x’7C00 0396’)

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

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

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

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

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

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

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

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

PowerPC Architecture Level	Supervisor Level	PowerPC Optional	Form
UISA			XO

lwz

Load Word and Zero (x’8000 0000’)

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

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 ← MEM(EA, 4)
```

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

Other registers altered:

- None

PowerPC Architecture Level	Supervisor Level	PowerPC Optional	Form
UISA			D

lwzu lwzu

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

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

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

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

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

EA is placed into **r**A.

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

Other registers altered:

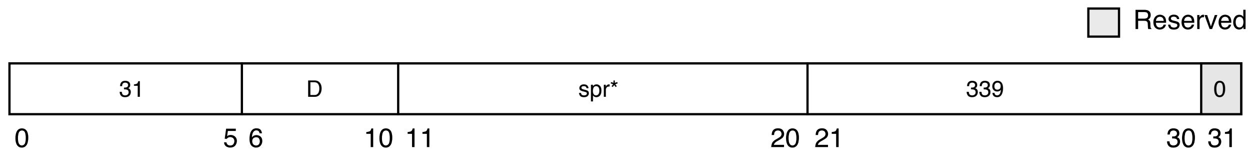
- None

PowerPC Architecture Level	Supervisor Level	PowerPC Optional	Form
UISA			D

mfspr

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

mfspr **r**D,SPR



NOTE: *This is a split field.

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

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

.

Table 8-9. PowerPC UISA SPR Encodings for mfspr

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

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

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

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

Other registers altered:

- None

Simplified mnemonics:

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

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

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

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

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

Other registers altered:

None

Table 8-10. PowerPC OEA SPR Encodings for mfspr

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

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

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

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

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

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

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

mtspr 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]
  SPR(n) ← (rS)
```

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

Table 8-12. PowerPC UISA SPR Encodings for mtspr

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

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

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

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

Other registers altered:

- See Table 8-12.

Simplified mnemonics:

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

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

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

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

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

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

Other registers altered:

- See Table 8-13.

Table 8-13. PowerPC OEA SPR Encodings for mtspr

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

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

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

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

.

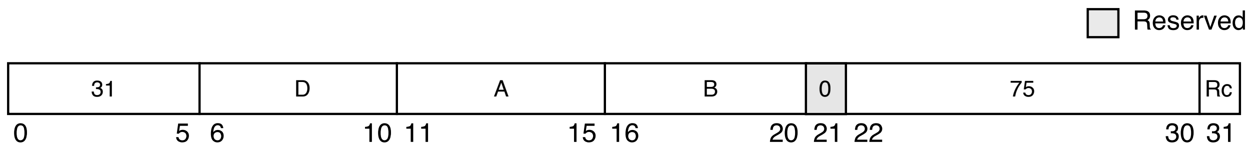
NOTE: **mtspr** is supervisor-level only if SPR[0] = 1.

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

mulhw*x*

Multiply High Word (x’7C00 0096’)

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



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

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

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

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

Other registers altered:

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

PowerPC Architecture Level	Supervisor Level	PowerPC Optional	Form
UISA			XO

mulhwu*x*

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) * (rB)
rD ← prod[0–31]
```

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

Both the operands and the product are interpreted as unsigned integers, except that if Rc = 1 the first three bits of CR0 field are set by signed comparison of the result to zero.

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

Other registers altered:

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

PowerPC Architecture Level	Supervisor Level	PowerPC Optional	Form
UISA			XO

mulli mulli

Multiply Low Immediate (x’1C00 0000’)

mulli rD,rA,SIMM

07	D	A	SIMM
0 5	6 10	11 15	16 31

```
prod[0–63]← (rA) * EXTS(SIMM)
rD ← prod[32-63]
```

The first operand is (**r**A). The second operand is the sign-extended value of the SIMM field. The low-order 32-bits of the 64-bit product of the operands are placed into **r**D.

Both the operands and the product are interpreted as signed integers. The low-order 32-bits of the product are calculated independently of whether the operands are treated as signed or unsigned 32-bit integers.

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

Other registers altered:

- None

PowerPC Architecture Level	Supervisor Level	PowerPC Optional	Form
UISA			D

mullw*x*

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)

31	D	A	B	OE	235	Rc
0 5	6 10	11 15	16 20	21	22 30	31

```
prod[0–63]← (rA) * (rB)
rD ← prod[32-63]
```

The 32-bit operands are the contents of **r**A and **r**B. The low-order 32-bits of the 64-bit product (**r**A) * (**r**B) are placed into **r**D.

The low-order 32-bits of the product are independent of whether the operands are regarded as signed or unsigned 32-bit integers.

If OE = 1, then OV is set if the product cannot be represented in 32 bits. Both the operands and the product are interpreted as signed integers.

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

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

Other registers altered:

- Condition Register (CR0 field):

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

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

- XER:

  Affected: SO, OV (If OE = 1)

PowerPC Architecture Level	Supervisor Level	PowerPC Optional	Form
UISA			XO

nand*x*

NAND (x’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	PowerPC Optional	Form
UISA			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 one’s complement of the value in **r**A, and the resulting two’s complement is placed into **r**D.

If **r**A contains the most negative 32-bit number (0x8000_0000), the result is the most negative number and, if OE = 1, OV is set.

Other registers altered:

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

PowerPC Architecture Level	Supervisor Level	PowerPC Optional	Form
UISA			XO

nor*x*

NOR (x'7C00 00F8')

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

31	S	A	B	124	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 complemented result is placed into **r**A.

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

Other registers altered:

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

Simplified mnemonics:

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

PowerPC Architecture Level	Supervisor Level	PowerPC Optional	Form
UISA			X

or*x*

OR (x’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	PowerPC Optional	Form
UISA			X

ori ori

OR Immediate (x’6000 0000’)

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

24	S	A	UIMM
0 5	6 10	11 15	16 31

```
rA ← (rS) | ((16)0 || UIMM)
```

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

The preferred no-op (an instruction that does nothing) is **ori 0,0,0**.

Other registers altered:

- None

Simplified mnemonics:

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

PowerPC Architecture Level	Supervisor Level	PowerPC Optional	Form
UISA			D

oris oris

OR Immediate Shifted (x’6400 0000’)

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

25	S	A	UIMM
0 5	6 10	11 15	16 31

```
rA ← (rS) | (UIMM || (16)0)
```

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

Other registers altered:

- None

PowerPC Architecture Level	Supervisor Level	PowerPC Optional	Form
UISA			D

rlwimi*x* 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)

20	S	A	SH	MB	ME	Rc
0 5	6 10	11 15	16 20	21 25	26 30	31

```
n ← SH
r ← ROTL(rS, n)
m ← MASK(MB, ME)
rA ← (r & m) | (rA & ¬ m)
```

The contents of **r**S are rotated left the number of bits specified by operand SH. A mask is generated having 1 bits from bit MB through bit ME and 0 bits elsewhere. The rotated data is inserted into **r**A under control of the generated mask.

NOTE: **rlwimi** can be used to copy a bit field of any length from register **r**S into the contents of **r**A. This field can start from any bit position in **r**S and be placed into any position in **r**A. The length of the field can range from 0 to 32 bits. The remaining bits in register **r**A remain unchanged:

- To copy byte_0 (bits 0-7) from rS into byte_3 (bits 24-31) of rA, set SH = 8 , MB = *24*, and ME = 31.
- In general, to copy an *n*-bit field that starts in bit position b in register rS into register rA starting a bit position c: set SH = 32 - c + b Mod(32), set MB = *c*, and set ME = (*c* + *n*) – 1 Mod(32).

Other registers altered:

- Condition Register (CR0 field):

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

Simplified mnemonics:
inslwi rA,rS,n,b equivalent to rlwimi rA,rS,32 – b,b,b + n – 1
insrwi rA,rS,n,b (n > 0) equivalent to rlwimi rA,rS,32 – (b + n),b, (b + n) – 1

PowerPC Architecture Level	Supervisor Level	PowerPC Optional	Form
UISA			M

rlwinm*x*

Rotate Left Word Immediate then AND with Mask (x’5400 0000’)

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

21	S	A	SH	MB	ME	Rc
0 5	6 10	11 15	16 20	21 25	26 30	31

```
n ← SH
r ← ROTL(rS, n)
m ← MASK(MB , ME)
rA ← r & m
```

The contents of **r**S are rotated left the number of bits specified by operand SH. A mask is generated having 1 bits from bit MB through bit ME and 0 bits elsewhere. The rotated data is ANDed with the generated mask and the result is placed into **r**A.

NOTE: **rlwinm** can be used to extract, rotate, shift, and clear bit fields using the methods shown below:

- To extract an *n*-bit field, that starts at bit position *b* in **r**S, right-justified into **r**A (clearing the remaining 32 – *n* bits of **r**A), set SH = *b* + *n*, MB = 32 – *n*, and ME = 31.
- To extract an *n*-bit field, that starts at bit position *b* in **r**S, left-justified into **r**A (clearing the remaining 32 – *n* bits of **r**A), set SH = *b*, MB = 0, and ME = *n* – 1.
- To rotate the contents of a register left (or right) by *n* bits, set SH = *n* (32 – *n*), MB = 0, and ME = 31.
- To shift the contents of a register right by *n* bits, by setting SH = 32 – *n*, MB = *n*, and ME = 31. It can be used to clear the high-order *b* bits of a register and then shift the result left by *n* bits by setting SH = *n*, MB = *b* – *n* and ME = 31 – *n*.
- To clear the low-order *n* bits of a register, by setting SH = 0, MB = 0, and ME = 31 – *n*..

Other registers altered:

- Condition Register (CR0 field):

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

Simplified mnemonics:

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

PowerPC Architecture Level	Supervisor Level	PowerPC Optional	Form
UISA			M

rlwnm*x* **rlwnm***x*

Rotate Left Word then AND with Mask (x’5C00 0000’)

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

23	S	A	B	MB	ME	Rc
0 5	6 10	11 15	16 20	21 25	26 30	31

```
n ← rB[27-31]
r ← ROTL(rS, n)
m ← MASK(MB, ME)
rA ← r & m
```

The contents of **r**S are rotated left the number of bits specified by the low-order five bits of **r**B. A mask is generated having 1 bits from bit MB through bit ME and 0 bits elsewhere. The rotated data is ANDed with the generated mask and the result is placed into **r**A.

NOTE: **rlwnm** can be used to extract and rotate bit fields using the methods shown as follows:

- To extract an *n*-bit field, that starts at variable bit position *b* in **r**S, right-justified into **r**A (clearing the remaining 32 – *n* bits of **r**A), by setting the low-order five bits of **r**B to *b* + *n*, MB = 32 – *n*, and ME = 31.
- To extract an *n*-bit field, that starts at variable bit position *b* in **r**S, left-justified into **r**A (clearing the remaining 32 – *n* bits of **r**A), by setting the low-order five bits of **r**B to *b*, MB = 0, and ME = *n* – 1.
- To rotate the contents of a register left (or right) by *n* bits, by setting the low-order five bits of **r**B to *n* (32 – *n*), MB = 0, and ME = 31.

Other registers altered:

- Condition Register (CR0 field):

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

Simplified mnemonics:

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

PowerPC Architecture Level	Supervisor Level	PowerPC Optional	Form
UISA			M

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)

31	S	A	B	24	Rc
0 5	6 10	11 15	16 20	21 30	31

```
n← rB[27-31]
r ← ROTL(rS, n)
if rB[26] = 0
     then m ← MASK(0, 31 – n)
     else m ← (32)0
rA ← r & m
```

The contents of **r**S are shifted left the number of bits specified by the low-order five bits of **r**B. Bits shifted out of position 0 are lost. Zeros are supplied to the vacated positions on the right. The 32-bit result is placed into **r**A. However, shift amounts from 32 to 63 give a zero result.

Other registers altered:

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

PowerPC Architecture Level	Supervisor Level	PowerPC Optional	Form
UISA			X

sraw*x* **sraw***x*

Shift Right Algebraic Word (x'7C00 0630')

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

31	S	A	B	792	Rc
0 5	6 10	11 15	16 20	21 30	31

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

The contents of **r**S are shifted right the number of bits specified by the low-order five bits of **r**B (shift amounts between 0-31). Bits shifted out of position 31 are lost. Bit 0 of **r**S is replicated to fill the vacated positions on the left. The 32-bit result is placed into **r**A. XER[CA] is set if **r**S contains a negative number and any 1 bits are shifted out of position 31; otherwise XER[CA] is cleared. A shift amount of zero causes **r**A to receive the 32 bits of **r**S, and XER[CA] to be cleared. However, shift amounts from 32 to 63 give a result of 32 sign bits, and cause XER[CA] to receive the sign bit of **r**S.

NOTE: The **sraw** instruction, followed by **addze**, can be used to divide quickly by 2^n.

Other registers altered:

- Condition Register (CR0 field):

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

  Affected: CA

PowerPC Architecture Level	Supervisor Level	PowerPC Optional	Form
UISA			X

Shift Right Algebraic Word Immediate (x’7C00 0670’)

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

31	S	A	SH	824	Rc
0 5	6 10	11 15	16 20	21 30	31

```
n ← SH
r ← ROTL(rS, 32− n)
m← MASK(n, 31)
S ← rS(0)
rA ← r & m | (32)S & ¬ m
XER[CA] ← S & ((r & ¬ m) ≠ 0)
```

The contents of **r**S are shifted right SH bits. Bits shifted out of position 31 are lost. Bit 0 of **r**S is replicated to fill the vacated positions on the left. The result is placed into **r**A. XER[CA] is set if the 32 bits of **r**S contain a negative number and any 1 bits are shifted out of position 31; otherwise XER[CA] is cleared. A shift amount of zero causes **r**A to receive the value of **r**S, and XER[CA] to be cleared.

NOTE: The **srawi** instruction, followed by **addze**, can be used to divide quickly by 2^n.

Other registers altered:

- Condition Register (CR0 field):

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

  Affected: CA

PowerPC Architecture Level	Supervisor Level	PowerPC Optional	Form
UISA			X

srw*x* srw*x*

Shift Right Word (x’7C00 0430’)

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

31	S	A	B	536	Rc
0 5	6 10	11 15	16 20	21 30	31

```
n ← rB[27-31]
r ← ROTL(rS, 32—n)
if rB[26] = 0
     then m ← MASK(n, 31)
     else m ← (32)0
rA ← r & m
```

The contents of **r**S are shifted right the number of bits specified by the low-order five bits of **r**B (shift amounts between 0-31). Bits shifted out of position 31 are lost. Zeros are supplied to the vacated positions on the left. The 32-bit result is placed into **r**A. However, shift amounts from 32 to 63 give a zero result.

Other registers altered:

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

PowerPC Architecture Level	Supervisor Level	PowerPC Optional	Form
UISA			X

stw

Store Word (x’9000 0000’)

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

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
```

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

Other registers altered:

- None

PowerPC Architecture Level	Supervisor Level	PowerPC Optional	Form
UISA			D

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. (equivlent to (rB)--(**r**A))

The **subf** instruction is preferred for subtraction because it sets few status bits.

Other registers altered:

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

Simplified mnemonics:

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

PowerPC Architecture Level	Supervisor Level	PowerPC Optional	Form
UISA			XO

Appendix F. Simplified Mnemonics

This appendix is provided in order to simplify the writing and comprehension of assembler language programs. Included are a set of simplified mnemonics and symbols that define the simple shorthand used for the most frequently-used forms of branch conditional, compare, trap, rotate and shift, and certain other instructions.

NOTE: The architecture specification refers to simplified mnemonics as extended mnemonics.

F.1 Symbols

The symbols in Table F-1 are defined for use in instructions (basic or simplified mnemonics) that specify a condition register (CR) field or a bit in the CR.

Table F-1. Condition Register Bit and Identification Symbol Descriptions

Symbol	Value	Bit Field Range	Description
lt	0	—	Less than. Identifies a bit number within a CR field.
gt	1	—	Greater than. Identifies a bit number within a CR field.
eq	2	—	Equal. Identifies a bit number within a CR field.
so	3	—	Summary overflow. Identifies a bit number within a CR field.
un	3	—	Unordered (after floating-point comparison). Identifies a bit number in a CR field.
cr0	0	0–3	CR0 field
cr1	1	4–7	CR1 field
cr2	2	8–11	CR2 field
cr3	3	12–15	CR3 field
cr4	4	16–19	CR4 field
cr5	5	20–23	CR5 field
cr6	6	24–27	CR6 field
cr7	7	28–31	CR7 field

Note: To identify a CR bit, an expression in which a CR field symbol is multiplied by 4 and then added to a bit-number-within-CR-field symbol can be used.

The simplified mnemonics in Section F.5.2, “Basic Branch Mnemonics,” and Section F.6, “Simplified Mnemonics for Condition Register Logical Instructions,” require identification of a CR bit—if one of the CR field symbols is used, it must be multiplied by 4 and added to a bit-number-within-CR-field (value in the range of 0–3, explicit or symbolic).

The simplified mnemonics in Section F.5.3, “Branch Mnemonics Incorporating Conditions,” and Section F.3, “Simplified Mnemonics for Compare Instructions,” require identification of a CR field—if one of the CR field symbols is used, it must not be multiplied by 4.

Also, for the simplified mnemonics in Section F.5.3, “Branch Mnemonics Incorporating Conditions,” the bit number within the CR field is part of the simplified mnemonic. The CR field is identified, and the assembler does the multiplication and addition required to produce a CR bit number for the BI field of the underlying basic mnemonic.

F.2 Simplified Mnemonics for Subtract Instructions

This section discusses simplified mnemonics for the subtract instructions.

F.2.1 Subtract Immediate

Although there is no subtract immediate instruction, its effect can be achieved by using an add immediate instruction with the immediate operand negated. Simplified mnemonics are provided that include this negation, making the intent of the computation more clear.

subi rD**,r**A**,**value	(equivalent to	**addi r**D**,r**A**,**–value)
subis rD**,r**A**,**value	(equivalent to	**addis r**D**,r**A**,**–value)
subic rD**,r**A**,**value	(equivalent to	**addic r**D**,r**A**,**–value)
subic. rD**,r**A**,**value	(equivalent to	**addic. r**D**,r**A**,**–value)

F.2.2 Subtract

The subtract from instructions subtract the second operand (**r**A) from the third (**r**B). Simplified mnemonics are provided that use the more normal order in which the third operand is subtracted from the second. Both these mnemonics can be coded with an **o** suffix and/or dot (**.**) suffix to cause the OE and/or Rc bit to be set in the underlying instruction.

sub rD**,r**A**,r**B	(equivalent to	**subf r**D**,r**B**,r**A)
subc rD**,r**A**,r**B	(equivalent to	**subfc r**D**,r**B**,r**A)

F.3 Simplified Mnemonics for Compare Instructions

The **crf**D field can be omitted if the result of the comparison is to be placed into the CR0 field. Otherwise, the target CR field must be specified as the first operand. One of the CR field symbols defined in Section F.1, "Symbols," can be used for this operand.

NOTE: The basic compare mnemonics of PowerPC are the same as those of POWER, but the POWER instructions have three operands whereas the PowerPC instructions have four.
The assembler recognizes a basic compare mnemonic with the three operands as the POWER form, and generates the instruction with L = 0. The **crf**D field can normally be omitted when the CR0 field is the target.

F.3.1 Word Comparisons

The instructions listed in Table F-2 are simplified mnemonics that should be supported by assemblers for all PowerPC implementations.

Table F-2. Simplified Mnemonics for Word Compare Instructions

Operation	Simplified Mnemonic	Equivalent to:
Compare Word Immediate	**cmpwi crf**D,**r**A,SIMM	**cmpi crf**D,**0**,**r**A,SIMM
Compare Word	**cmpw crf**D,**r**A,**r**B	**cmp crf**D,**0**,**r**A,**r**B
Compare Logical Word Immediate	**cmplwi crf**D,**r**A,UIMM	**cmpli crf**D,**0**,**r**A,UIMM
Compare Logical Word	**cmplw crf**D,**r**A,**r**B	**cmpl crf**D,**0**,**r**A,**r**B

Following are examples using the word compare mnemonics.

1. Compare **r**A with immediate value 100 as signed 32-bit integers and place result in CR0.
   cmpwi rA**,100** (equivalent to **cmpi 0,0,r**A**,100**)
2. Same as (1), but place results in CR4.
   cmpwi cr4,rA**,100** (equivalent to **cmpi 4,0,r**A**,100**)
3. Compare **r**A and **r**B as unsigned 32-bit integers and place result in CR0.
   cmplw rA**,r**B (equivalent to **cmpl 0,0,r**A**,r**B)

F

F.4 Simplified Mnemonics for Rotate and Shift Instructions

The rotate and shift instructions provide powerful and general ways to manipulate register contents, but can be difficult to understand. Simplified mnemonics that allow some of the simpler operations to be coded easily are provided for the following types of operations:

- Extract—Select a field of *n* bits starting at bit position *b* in the source register; left or right justify this field in the target register; clear all other bits of the target register.
- Insert—Select a left-justified or right-justified field of *n* bits in the source register; insert this field starting at bit position *b* of the target register; leave other bits of the target register unchanged. (No simplified mnemonic is provided for insertion of a left-justified field, when operating on double words, because such an insertion requires more than one instruction.)
- Rotate—Rotate the contents of a register right or left *n* bits without masking.
- Shift—Shift the contents of a register right or left *n* bits, clearing vacated bits (logical shift).
- Clear—Clear the leftmost or rightmost *n* bits of a register.
- Clear left and shift left—Clear the leftmost *b* bits of a register, then shift the register left by *n* bits. This operation can be used to scale a (known non-negative) array index by the width of an element.

F.4.1 Operations on Words

The operations shown in Table F-3 are available in all implementations. All these mnemonics can be coded with a dot (**.**) suffix to cause the Rc bit to be set in the underlying instruction.

Table F-3. Word Rotate and Shift Instructions

Operation	Simplified Mnemonic	Equivalent to:
Extract and left justify immediate	**extlwi r**A,**r**S,*n*,*b* (*n* > 0)	**rlwinm r**A,**r**S,*b*,**0**,*n* – 1
Extract and right justify immediate	**extrwi r**A,**r**S,*n*,*b* (*n* > 0)	**rlwinm r**A,**r**S,*b* + *n*, 32 – *n*,**31**
Insert from left immediate	**inslwi r**A,**r**S,*n*,*b* (*n* > 0)	**rlwimi r**A,**r**S,32 – *b*,*b*,(*b* + *n*) – 1
Insert from right immediate	**insrwi r**A,**r**S,*n*,*b* (*n* > 0)	**rlwimi r**A,**r**S,32 – (*b* + *n*),*b*,(*b* + *n*) – 1
Rotate left immediate	**rotlwi r**A,**r**S,*n*	**rlwinm r**A,**r**S,*n*,**0,31**
Rotate right immediate	**rotrwi r**A,**r**S,*n*	**rlwinm r**A,**r**S,32 – *n*,**0,31**
Rotate left	**rotlw r**A,**r**S,**r**B	**rlwnm r**A,**r**S,**r**B,**0,31**
Shift left immediate	**slwi r**A,**r**S,*n* (*n* < 32)	**rlwinm r**A,**r**S,*n*,**0**,31 – *n*
Shift right immediate	**srwi r**A,**r**S,*n* (*n* < 32)	**rlwinm r**A,**r**S,32 – *n*,*n*,**31**
Clear left immediate	**clrlwi r**A,**r**S,*n* (*n* < 32)	**rlwinm r**A,**r**S,**0**,*n*,**31**
Clear right immediate	**clrrwi r**A,**r**S,*n* (*n* < 32)	**rlwinm r**A,**r**S,**0**,**0**,31 – *n*
Clear left and shift left immediate	**clrlslwi r**A,**r**S,*b*,*n* (*n* ≤ *b* ≤ 31)	**rlwinm r**A,**r**S,*n*,*b* – *n*,31 – *n*

Examples using word mnemonics follow:

1. Extract the sign bit (bit 0) of **r**S and place the result right-justified into **r**A.
   extrwi rA,**r**S,**1,0** (equivalent to **rlwinm r**A,**r**S,**1,31,31**)
2. Insert the bit extracted in (1) into the sign bit (bit 0) of **r**B.
   insrwi rB,**r**A,**1,0** (equivalent to **rlwimi r**B,**r**A,**31,0,0**)
3. Shift the contents of **r**A left 8 bits.
   slwi rA,**r**A,**8** (equivalent to **rlwinm r**A,**r**A,**8,0,23**)
4. Clear the high-order 16 bits of **r**S and place the result into **r**A.
   clrlwi rA,**r**S,**16** (equivalent to **rlwinm r**A,**r**S,**0,16,31**)

F.5 Simplified Mnemonics for Branch Instructions

Mnemonics are provided so that branch conditional instructions can be coded with the condition as part of the instruction mnemonic rather than as a numeric operand. Some of these are shown as examples with the branch instructions.

The mnemonics discussed in this section are variations of the branch conditional instructions.

F.5.1 BO and BI Fields

The 5-bit BO field in branch conditional instructions encodes the following operations.

- Decrement count register (CTR)
- Test CTR equal to zero
- Test CTR not equal to zero
- Test condition true
- Test condition false
- Branch prediction (taken, fall through)

The 5-bit BI field in branch conditional instructions specifies which of the 32 bits in the CR represents the condition to test.

To provide a simplified mnemonic for every possible combination of BO and BI fields would require $2^{10} = 1024$ mnemonics and most of these would be only marginally useful. The abbreviated set found in Section F.5.2, "Basic Branch Mnemonics," is intended to cover the most useful cases. Unusual cases can be coded using a basic branch conditional mnemonic (**bc**, **bclr**, **bcctr**) with the condition to be tested specified as a numeric operand.

F.5.2 Basic Branch Mnemonics

The mnemonics in Table F-4 allow all the common BO operand encodings to be specified as part of the mnemonic, along with the absolute address (AA), and set link register (LR) bits.

Notice that there are no simplified mnemonics for relative and absolute unconditional branches. For these, the basic mnemonics **b**, **ba**, **bl**, and **bla** are used.

Table F-4 provides the abbreviated set of simplified mnemonics for the most commonly performed conditional branches.

Table F-4. Simplified Branch Mnemonics

Branch Semantics	LR Update Not Enabled				LR Update Enabled			
	bc Relative	**bca** Absolute	**bclr** to LR	**bcctr** to CTR	**bcl** Relative	**bcla** Absolute	**bclrl** to LR	**bcctrl** to CTR
Branch unconditionally	—	—	**blr**	**bctr**	—	—	**blrl**	**bctrl**
Branch if condition true	**bt**	**bta**	**btlr**	**btctr**	**btl**	**btla**	**btlrl**	**btctrl**
Branch if condition false	**bf**	**bfa**	**bflr**	**bfctr**	**bfl**	**bfla**	**bflrl**	**bfctrl**
Decrement CTR, branch if CTR non-zero	**bdnz**	**bdnza**	**bdnzlr**	—	**bdnzl**	**bdnzla**	**bdnzlrl**	—
Decrement CTR, branch if CTR non-zero AND condition true	**bdnzt**	**bdnzta**	**bdnztlr**	—	**bdnztl**	**bdnztla**	**bdnztlrl**	—
Decrement CTR, branch if CTR non-zero AND condition false	**bdnzf**	**bdnzfa**	**bdnzflr**	—	**bdnzfl**	**bdnzfla**	**bdnzflrl**	—
Decrement CTR, branch if CTR zero	**bdz**	**bdza**	**bdzlr**	—	**bdzl**	**bdzla**	**bdzlrl**	—
Decrement CTR, branch if CTR zero AND condition true	**bdzt**	**bdzta**	**bdztlr**	—	**bdztl**	**bdztla**	**bdztlrl**	—
Decrement CTR, branch if CTR zero AND condition false	**bdzf**	**bdzfa**	**bdzflr**	—	**bdzfl**	**bdzfla**	**bdzflrl**	—

The simplified mnemonics shown in Table F-4 that test a condition require a corresponding CR bit as the first operand of the instruction. The symbols defined in Section F.1, "Symbols," can be used in the operand in place of a numeric value.

The simplified mnemonics found in Table F-4 are used in the following examples:

1. Decrement CTR and branch if it is still nonzero (closure of a loop controlled by a count loaded into CTR).
   bdnz target (equivalent to **bc 16,0,**target)
2. Same as (1) but branch only if CTR is non-zero and condition in CR0 is "equal."
   bdnzt eq,target (equivalent to **bc 8,2,**target)
3. Same as (2), but "equal" condition is in CR5.
   bdnzt 4 * cr5 + eq,target (equivalent to **bc 8,22,**target)
4. Branch if bit 27 of CR is false.
   bf 27,target (equivalent to **bc 4,27,**target)
5. Same as (4), but set the link register. This is a form of conditional call.
   bfl 27,target (equivalent to **bcl 4,27,**target)

Table F-5 provides the simplified mnemonics for the **bc** and **bca** instructions without link register updating, and the syntax associated with these instructions.

NOTE: The default condition register specified by the simplified mnemonics in the table is CR0.

Table F-5. Simplified Branch Mnemonics for bc and bca Instructions without Link Register Update

Branch Semantics	LR Update Not Enabled			
	bc Relative	Simplified Mnemonic	**bca** Absolute	Simplified Mnemonic
Branch unconditionally	—	—	—	—
Branch if condition true	**bc** 12,0,target	**bt** 0,target	**bca** 12,0,target	**bta** 0,target
Branch if condition false	**bc** 4,0,target	**bf** 0,target	**bca** 4,0,target	**bfa** 0,target
Decrement CTR, branch if CTR nonzero	**bc**16,0,target	**bdnz** target	**bca** 16,0,target	**bdnza** target
Decrement CTR, branch if CTR nonzero AND condition true	**bc** 8,0,target	**bdnzt** 0,target	**bca** 8,0,target	**bdnzta** 0,target
Decrement CTR, branch if CTR nonzero AND condition false	**bc** 0,0,target	**bdnzf** 0,target	**bca** 0,0,target	**bdnzfa** 0,target
Decrement CTR, branch if CTR zero	**bc**18,0,target	**bdz** target	**bca** 18,0,target	**bdza** target
Decrement CTR, branch if CTR zero AND condition true	**bc**10,0,target	**bdzt** 0,target	**bca** 10,0,target	**bdzta** 0,target
Decrement CTR, branch if CTR zero AND condition false	**bc** 2,0,target	**bdzf** 0,target	**bca** 2,0,target	**bdzfa** 0,target

Table F-6 provides the simplified mnemonics for the **bclr** and **bcclr** instructions without link register updating, and the syntax associated with these instructions.

NOTE: The default condition register specified by the simplified mnemonics in the table is CR0.

Table F-6. Simplified Branch Mnemonics for bclr and bcclr Instructions without Link Register Update

Branch Semantics	LR Update Not Enabled			
	bclr to LR	Simplified Mnemonic	**bcctr** to CTR	Simplified Mnemonic
Branch unconditionally	**bclr** 20,0	**blr**	**bcctr** 20,0	**bctr**
Branch if condition true	**bclr** 12,0	**btlr** 0	**bcctr** 12,0	**btctr** 0
Branch if condition false	**bclr** 4,0	**bflr** 0	**bcctr** 4,0	**bfctr** 0
Decrement CTR, branch if CTR nonzero	**bclr** 16,0	**bdnzlr**	—	—
Decrement CTR, branch if CTR nonzero AND condition true	**bclr** 10,0	**bdztlr** 0	—	—
Decrement CTR, branch if CTR nonzero AND condition false	**bclr** 0,0	**bdnzflr** 0	—	—
Decrement CTR, branch if CTR zero	**bclr** 18,0	**bdzlr**	—	—
Decrement CTR, branch if CTR zero AND condition true	**bclr** 10,0	**bdztlr** 0	—	—
Decrement CTR, branch if CTR zero AND condition false	**bcctr** 0,0	**bdzflr** 0	—	—

Table F-7 provides the simplified mnemonics for the **bcl** and **bcla** instructions with link register updating, and the syntax associated with these instructions.

NOTE: The default condition register specified by the simplified mnemonics in the table is CR0.

Table F-7. Simplified Branch Mnemonics for bcl and bcla Instructions with Link Register Update

Branch Semantics	LR Update Enabled			
	bcl Relative	Simplified Mnemonic	**bcla** Absolute	Simplified Mnemonic
Branch unconditionally	—	—	—	—
Branch if condition true	**bcl1** 2,0,target	**btl** 0,target	**bcla** 12,0,target	**btla** 0,target
Branch if condition false	**bcl** 4,0,target	**bfl** 0,target	**bcla** 4,0,target	**bfla** 0,target
Decrement CTR, branch if CTR nonzero	**bcl** 16,0,target	**bdnzl** target	**bcla** 16,0,target	**bdnzla** target
Decrement CTR, branch if CTR nonzero AND condition true	**bcl** 8,0,target	**bdnztl** 0,target	**bcla** 8,0,target	**bdnztla** 0,target
Decrement CTR, branch if CTR nonzero AND condition false	**bcl** 0,0,target	**bdnzfl** 0,target	**bcla** 0,0,target	**bdnzfla** 0,target
Decrement CTR, branch if CTR zero	**bcl** 18,0,target	**bdzl** target	**bcla** 18,0,target	**bdzla** target
Decrement CTR, branch if CTR zero AND condition true	**bcl** 10,0,target	**bdztl** 0,target	**bcla** 10,0,target	**bdztla** 0,target
Decrement CTR, branch if CTR zero AND condition false	**bcl** 2,0,target	**bdzfl** 0,target	**bcla** 2,0,target	**bdzfla** 0,target

Table F-8 provides the simplified mnemonics for the **bclrl** and **bcctrl** instructions with link register updating, and the syntax associated with these instructions.

NOTE: The default condition register specified by the simplified mnemonics in the table is CR0.

Table F-8. Simplified Branch Mnemonics for bclrl and bcctrl Instructions with Link Register Update

Branch Semantics	LR Update Enabled			
	bclrl to LR	Simplified Mnemonic	**bcctrl** to CTR	Simplified Mnemonic
Branch unconditionally	**bclrl** 20,0	**blrl**	**bcctrl** 20,0	**bctrl**
Branch if condition true	**bclrl**12,0	**btlrl** 0	**bcctrl** 12,0	**btctrl** 0
Branch if condition false	**bclrl** 4,0	**bflrl** 0	**bcctrl** 4,0	**bfctrl** 0
Decrement CTR, branch if CTR nonzero	**bclrl** 16,0	**bdnzlrl**	—	—
Decrement CTR, branch if CTR nonzero AND condition true	**bclrl** 8,0	**bdnztlrl** 0	—	—
Decrement CTR, branch if CTR nonzero AND condition false	**bclrl** 0,0	**bdnzflrl** 0	—	—
Decrement CTR, branch if CTR zero	**bclrl** 18,0	**bdzlrl**	—	—
Decrement CTR, branch if CTR zero AND condition true	**bdztlrl** 0	**bdztlrl** 0	—	—
Decrement CTR, branch if CTR zero AND condition false	**bclrl** 4,0	**bflrl** 0	—	—

F.5.3 Branch Mnemonics Incorporating Conditions

The mnemonics defined in Table F-4 are variations of the branch if condition true and branch if condition false BO encodings, with the most useful values of BI represented in the mnemonic rather than specified as a numeric operand.

A standard set of codes (shown in Table F-9) has been adopted for the most common combinations of branch conditions.

Table F-9. Standard Coding for Branch Conditions

Code	Description
lt	Less than
le	Less than or equal
eq	Equal
ge	Greater than or equal
gt	Greater than
nl	Not less than
ne	Not equal
ng	Not greater than
so	Summary overflow
ns	Not summary overflow
un	Unordered (after floating-point comparison)
nu	Not unordered (after floating-point comparison)

Table F-10 shows the simplified branch mnemonics incorporating conditions.

Table F-10. Simplified Branch Mnemonics with Comparison Conditions

Branch Semantics	LR Update Not Enabled				LR Update Enabled			
	bc Relative	**bca** Absolute	**bclr** to LR	**bcctr** to CTR	**bcl** Relative	**bcla** Absolute	**bclrl** to LR	**bcctrl** to CTR
Branch if less than	**blt**	**blta**	**bltlr**	**bltctr**	**bltl**	**bltla**	**bltlrl**	**bltctrl**
Branch if less than or equal	**ble**	**blea**	**blelr**	**blectr**	**blel**	**blela**	**blelrl**	**blectrl**
Branch if equal	**beq**	**beqa**	**beqlr**	**beqctr**	**beql**	**beqla**	**beqlrl**	**beqctrl**
Branch if greater than or equal	**bge**	**bgea**	**bgelr**	**bgectr**	**bgel**	**bgela**	**bgelrl**	**bgectrl**
Branch if greater than	**bgt**	**bgta**	**bgtlr**	**bgtctr**	**bgtl**	**bgtla**	**bgtlrl**	**bgtctrl**
Branch if not less than	**bnl**	**bnla**	**bnllr**	**bnlctr**	**bnll**	**bnlla**	**bnllrl**	**bnlctrl**
Branch if not equal	**bne**	**bnea**	**bnelr**	**bnectr**	**bnel**	**bnela**	**bnelrl**	**bnectrl**
Branch if not greater than	**bng**	**bnga**	**bnglr**	**bngctr**	**bngl**	**bngla**	**bnglrl**	**bngctrl**
Branch if summary overflow	**bso**	**bsoa**	**bsolr**	**bsoctr**	**bsol**	**bsola**	**bsolrl**	**bsoctrl**
Branch if not summary overflow	**bns**	**bnsa**	**bnslr**	**bnsctr**	**bnsl**	**bnsla**	**bnslrl**	**bnsctrl**
Branch if unordered	**bun**	**buna**	**bunlr**	**bunctr**	**bunl**	**bunla**	**bunlrl**	**bunctrl**
Branch if not unordered	**bnu**	**bnua**	**bnulr**	**bnuctr**	**bnul**	**bnula**	**bnulrl**	**bnuctrl**

Instructions using the mnemonics in Table F-10 specify the condition register field in an optional first operand. If the CR field being tested is CR0, this operand need not be specified. One of the CR field symbols defined in Section F.1, “Symbols,” can be used for this operand.

The simplified mnemonics found in Table F-10 are used in the following examples:

1. Branch if CR0 reflects condition “not equal.”
   bne target (equivalent to **bc 4,2,**target)
2. Same as (1) but condition is in CR3.
   bne cr3,target (equivalent to **bc 4,14,**target)
3. Branch to an absolute target if CR4 specifies “greater than,” setting the link register. This is a form of conditional “call.”
   bgtla cr4,target (equivalent to **bcla 12,17,**target)
4. Same as (3), but target address is in the CTR.
   bgtctrl cr4 (equivalent to **bcctrl 12,17**)

Table F-11 shows the simplified branch mnemonics for the **bc** and **bca** instructions without link register updating, and the syntax associated with these instructions.

NOTE: The default condition register specified by the simplified mnemonics in the table is CR0.

Table F-11. Simplified Branch Mnemonics for bc and bca Instructions without Comparison Conditions and Link Register Updating

Branch Semantics	LR Update Not Enabled			
	bc Relative	Simplified Mnemonic	**bca** Absolute	Simplified Mnemonic
Branch if less than	**bc** 12,0,target	**blt** target	**bca** 12,0,target	**blta** target
Branch if less than or equal	**bc** 4,1,target	**ble** target	**bca** 4,1,target	**blea** target
Branch if equal	**bc** 12,2,target	**beq** target	**bca** 12,2,target	**beqa** target
Branch if greater than or equal	**bc** 4,0,target	**bge** target	**bca** 4,0,target	**bgea** target
Branch if greater than	**bc** 12,1,target	**bgt** target	**bca** 12,1,target	**bgta** target
Branch if not less than	**bc** 4,0,target	**bnl** target	**bca** 4,0,target	**bnla** target
Branch if not equal	**bc** 4,2,target	**bne** target	**bca** 4,2,target	**bnea** target
Branch if not greater than	**bc** 4,1,target	**bng** target	**bca** 4,1,target	**bnga** target
Branch if summary overflow	**bc** 12,3,target	**bso** target	**bca** 12,3,target	**bsoa** target
Branch if not summary overflow	**bc** 4,3,target	**bns** target	**bca** 4,3,target	**bnsa** target
Branch if unordered	**bc** 12,3,target	**bun** target	**bca** 12,3,target	**buna** target
Branch if not unordered	**bc** 4,3,target	**bnu** target	**bca** 4,3,target	**bnua** target

Table F-12 shows the simplified branch mnemonics for the **bclr** and **bcctr** instructions without link register updating, and the syntax associated with these instructions.

NOTE: The default condition register specified by the simplified mnemonics in the table is CR0.

Table F-12. Simplified Branch Mnemonics for bclr and bcctr Instructions without Comparison Conditions and Link Register Updating

Branch Semantics	**LR Update Not Enabled**			
	bclr to LR	Simplified Mnemonic	**bcctr** to CTR	Simplified Mnemonic
Branch if less than	**bclr** 12,0	**bltlr**	**bcctr** 12,0	**bltctr**
Branch if less than or equal	**bclr** 4,1	**blelr**	**bcctr** 4,1	**blectr**
Branch if equal	**bclr** 12,2	**beqlr**	**bcctr** 12,2	**beqctr**
Branch if greater than or equal	**bclr** 4,0	**bgelr**	**bcctr** 4,0	**bgectr**
Branch if greater than	**bclr** 12,1	**bgtlr**	**bcctr** 12,1	**bgtctr**
Branch if not less than	**bclr** 4,0	**bnllr**	**bcctr** 4,0	**bnlctr**
Branch if not equal	**bclr** 4,2	**bnelr**	**bcctr** 4,2	**bnectr**
Branch if not greater than	**bclr** 4,1	**bnglr**	**bcctr** 4,1	**bngctr**
Branch if summary overflow	**bclr** 12,3	**bsolr**	**bcctr** 12,3	**bsoctr**
Branch if not summary overflow	**bclr** 4,3	**bnslr**	**bcctr** 4,3	**bnsctr**
Branch if unordered	**bclr** 12,3	**bunlr**	**bcctr** 12,3	**bunctr**
Branch if not unordered	**bclr** 4,3	**bnulr**	**bcctr** 4,3	**bnuctr**

Table F-13 shows the simplified branch mnemonics for the **bcl** and **bcla** instructions with link register updating, and the syntax associated with these instructions.

NOTE: The default condition register specified by the simplified mnemonics in the table is CR0.

Table F-13. Simplified Branch Mnemonics for bcl and bcla Instructions with Comparison Conditions and Link Register Update

Branch Semantics	LR Update Enabled			
	bcl Relative	Simplified Mnemonic	**bcla** Absolute	Simplified Mnemonic
Branch if less than	**bcl** 12,0,target	**bltl** target	**bcla** 12,0,target	**bltla** target
Branch if less than or equal	**bcl** 4,1,target	**blel** target	**bcla** 4,1,target	**blela** target
Branch if equal	**beql** target	**beql** target	**bcla** 12,2,target	**beqla** target
Branch if greater than or equal	**bcl** 4,0,target	**bgel** target	**bcla** 4,0,target	**bgela** target
Branch if greater than	**bcl** 12,1,target	**bgtl** target	**bcla** 12,1,target	**bgtla** target
Branch if not less than	**bcl** 4,0,target	**bnll** target	**bcla** 4,0,target	**bnlla** target
Branch if not equal	**bcl** 4,2,target	**bnel** target	**bcla** 4,2,target	**bnela** target
Branch if not greater than	**bcl** 4,1,target	**bngl** target	**bcla** 4,1,target	**bngla** target
Branch if summary overflow	**bcl** 12,3,target	**bsol** target	**bcla** 12,3,target	**bsola** target
Branch if not summary overflow	**bcl** 4,3,target	**bnsl** target	**bcla** 4,3,target	**bnsla** target
Branch if unordered	**bcl** 12,3,target	**bunl** target	**bcla** 12,3,target	**bunla** target
Branch if not unordered	**bcl** 4,3,target	**bnul** target	**bcla** 4,3,target	**bnula** target

Table F-14 shows the simplified branch mnemonics for the **bclrl** and **bcctl** instructions with link register updating, and the syntax associated with these instructions.

NOTE: The default condition register specified by the simplified mnemonics in the table is CR0.

Table F-14. Simplified Branch Mnemonics for bclrl and bcctl Instructions with Comparison Conditions and Link Register Update

Branch Semantics	LR Update Enabled			
	bclrl to LR	Simplified Mnemonic	**bcctrl** to CTR	Simplified Mnemonic
Branch if less than	**bclrl** 12,0	**bltlrl** 0	**bcctrl** 12,0	**bltctrl** 0
Branch if less than or equal	**bclrl** 4,1	**blelrl** 0	**bcctrl** 4,1	**blectrl** 0
Branch if equal	**bclrl** 12,2	**beqlrl** 0	**bcctrl** 12,2	**beqctrl** 0
Branch if greater than or equal	**bclrl** 4,0	**bgelrl** 0	**bcctrl** 4,0	**bgectrl** 0
Branch if greater than	**bclrl** 12,1	**bgtlrl** 0	**bcctrl** 12,1	**bgtctrl** 0
Branch if not less than	**bclrl** 4,0	**bnllrl** 0	**bcctrl** 4,0	**bnlctrl** 0
Branch if not equal	**bclrl** 4,2	**bnelrl** 0	**bcctrl** 4,2	**bnectrl** 0
Branch if not greater than	**bclrl** 4,1	**bnglrl** 0	**bcctrl** 4,1	**bngctrl** 0
Branch if summary overflow	**bclrl** 12,3	**bsolrl** 0	**bcctrl** 12,3	**bsoctrl** 0
Branch if not summary overflow	**bclrl** 4,3	**bnslrl** 0	**bcctrl** 4,3	**bnsctrl** 0
Branch if unordered	**bclrl** 12,3	**bunlrl** 0	**bcctrl** 12,3	**bunctrl** 0
Branch if not unordered	**bclrl** 4,3	**bnulrl** 0	**bcctrl** 4,3	**bnuctrl** 0

F.5.4 Branch Prediction

In branch conditional instructions that are not always taken, the low-order bit (*y* bit) of the BO field provides a hint about whether the branch is likely to be taken. See Section 4.2.4.2, “Conditional Branch Control,” for more information on the *y* bit.

Assemblers should clear this bit unless otherwise directed. This default action indicates the following:

- A branch conditional with a negative displacement field is predicted to be taken.
- A branch conditional with a non-negative displacement field is predicted not to be taken (fall through).
- A branch conditional to an address in the LR or CTR is predicted not to be taken (fall through).

If the likely outcome (branch or fall through) of a given branch conditional instruction is known, a suffix can be added to the mnemonic that tells the assembler how to set the *y* bit. That is, '+' indicates that the branch is to be taken and '–' indicates that the branch is not to be taken. Such a suffix can be added to any branch conditional mnemonic, either basic or simplified.

For relative and absolute branches (**bc**[**l**][**a**]), the setting of the *y* bit depends on whether the displacement field is negative or non-negative. For negative displacement fields, coding the suffix '+' causes the bit to be cleared, and coding the suffix '–' causes the bit to be set. For non-negative displacement fields, coding the suffix '+' causes the bit to be set, and coding the suffix '–' causes the bit to be cleared.

For branches to an address in the LR or CTR (**bcclr**[**l**] or **bcctr**[**l**]), coding the suffix '+' causes the *y* bit to be set, and coding the suffix '–' causes the bit to be cleared.

Examples of branch prediction follow:

1. Branch if CR0 reflects condition "less than," specifying that the branch should be predicted to be taken.
   blt+ target
2. Same as (1), but target address is in the LR and the branch should be predicted not to be taken.
   bltlr–

F.6 Simplified Mnemonics for Condition Register Logical Instructions

The condition register logical instructions, shown in Table F-15, can be used to set, clear, copy, or invert a given condition register bit. Simplified mnemonics are provided that allow these operations to be coded easily.

NOTE: The symbols defined in Section F.1, "Symbols," can be used to identify the condition register bit.

Table F-15. Condition Register Logical Mnemonics

Operation	Simplified Mnemonic	Equivalent to
Condition register set	**crset bx**	**creqv bx,bx,bx**
Condition register clear	**crclr bx**	**crxor bx,bx,bx**
Condition register move	**crmove bx,by**	**cror bx,by,by**
Condition register not	**crnot bx,by**	**crnor bx,by,by**

Examples using the condition register logical mnemonics follow:

1. Set CR bit 25.
   crset 25 (equivalent to **creqv 25,25,25**)
2. Clear the SO bit of CR0.
   crclr so (equivalent to **crxor 3,3,3**)
3. Same as (2), but SO bit to be cleared is in CR3.
   crclr 4 * cr3 + so (equivalent to **crxor 15,15,15**)
4. Invert the EQ bit.
   crnot eq,eq (equivalent to **crnor 2,2,2**)
5. Same as (4), but EQ bit to be inverted is in CR4, and the result is to be placed into the EQ bit of CR5.
   crnot 4 * cr5 + eq, 4 * cr4 + eq (equivalent to **crnor 22,18,18**)

F.7 Simplified Mnemonics for Trap Instructions

A standard set of codes, shown in Table F-16, has been adopted for the most common combinations of trap conditions.

Table F-16. Standard Codes for Trap Instructions

Code	Description	TO Encoding	<	>	=	<U	>U
lt	Less than	16	1	0	0	0	0
le	Less than or equal	20	1	0	1	0	0
eq	Equal	4	0	0	1	0	0
ge	Greater than or equal	12	0	1	1	0	0
gt	Greater than	8	0	1	0	0	0
nl	Not less than	12	0	1	1	0	0
ne	Not equal	24	1	1	0	0	0
ng	Not greater than	20	1	0	1	0	0
llt	Logically less than	2	0	0	0	1	0
lle	Logically less than or equal	6	0	0	1	1	0
lge	Logically greater than or equal	5	0	0	1	0	1
lgt	Logically greater than	1	0	0	0	0	1
lnl	Logically not less than	5	0	0	1	0	1
lng	Logically not greater than	6	0	0	1	1	0
—	Unconditional	31	1	1	1	1	1

Note: The symbol "<U" indicates an unsigned less than evaluation will be performed. The symbol ">U" indicates an unsigned greater than evaluation will be performed.

The mnemonics defined in Table F-17 are variations of trap instructions, with the most useful values of TO represented in the mnemonic rather than specified as a numeric operand.

Table F-17. Trap Mnemonics

Trap Semantics	32-Bit Comparison	
	twi Immediate	**tw Register**
Trap unconditionally	—	trap
Trap if less than	**twlti**	**twlt**
Trap if less than or equal	**twlei**	**twle**
Trap if equal	**tweqi**	**tweq**
Trap if greater than or equal	**twgei**	**twge**
Trap if greater than	**twgti**	**twgt**
Trap if not less than	**twnli**	**twnl**
Trap if not equal	**twnei**	**twne**
Trap if not greater than	**twngi**	**twng**
Trap if logically less than	**twllti**	**twllt**
Trap if logically less than or equal	**twllei**	**twlle**
Trap if logically greater than or equal	**twlgei**	**twlge**
Trap if logically greater than	**twlgti**	**twlgt**
Trap if logically not less than	**twlnli**	**twlnl**
Trap if logically not greater than	**twlngi**	**twlng**

Examples of the uses of trap mnemonics, shown in Table F-17, follow:

1. Trap if register **r**A is not zero.
   twnei **r**A,**0** (equivalent to **twi 24,r**A,**0**)
2. Trap if register **r**A is not equal to **r**B.
   twne **r**A, **r**B (equivalent to **tw 24,r**A,**r**B)
3. Trap if **r**A is logically greater than 0x7FF.
   twlgti rA**,** 0x7FF (equivalent to **twi 1,r**A**,** 0x7FF)
4. Trap unconditionally.
   trap (equivalent to **tw 31,0,0**)

Trap instructions evaluate a trap condition as follows:

- The contents of register **r**A are compared with either the sign-extended SIMM field or the contents of register **r**B, depending on the trap instruction.

F

The comparison results in five conditions which are ANDed with operand TO. If the result is not 0, the trap exception handler is invoked.

NOTE: Exceptions are referred to as interrupts in the architecture specification.See Table F-18 for these conditions.

Table F-18. TO Operand Bit Encoding

TO Bit	ANDed with Condition
0	Less than, using signed comparison
1	Greater than, using signed comparison
2	Equal
3	Less than, using unsigned comparison
4	Greater than, using unsigned comparison

F.8 Simplified Mnemonics for Special-Purpose Registers

The **mtspr** and **mfspr** instructions specify a special-purpose register (SPR) as a numeric operand. Simplified mnemonics are provided that represent the SPR in the mnemonic rather than requiring it to be coded as a numeric operand. Table F-19 provides a list of the simplified mnemonics that should be provided by assemblers for SPR operations.

Table F-19. Simplified Mnemonics for SPRs

Special-Purpose Register	Move to SPR		Move from SPR	
	Simplified Mnemonic	Equivalent to	Simplified Mnemonic	Equivalent to
XER	**mtxer r**S	**mtspr** 1,**r**S	**mfxer r**D	**mfspr r**D,**1**
Link register	**mtlr r**S	**mtspr 8,r**S	**mflr r**D	**mfspr r**D,**8**
Count register	**mtctr r**S	**mtspr 9,r**S	**mfctr r**D	**mfspr r**D,**9**
DSISR	**mtdsisr r**S	**mtspr 18,r**S	**mfdsisr r**D	**mfspr r**D,**18**
Data address register	**mtdar r**S	**mtspr 19,r**S	**mfdar r**D	**mfspr r**D,**19**
Decrementer	**mtdec r**S	**mtspr 22,r**S	**mfdec r**D	**mfspr r**D,**22**
SDR1	**mtsdr1 r**S	**mtspr 25,r**S	**mfsdr1 r**D	**mfspr r**D,**25**
Save and restore register 0	**mtsrr0 r**S	**mtspr 26,r**S	**mfsrr0 r**D	**mfspr r**D,**26**
Save and restore register 1	**mtsrr1 r**S	**mtspr 27,r**S	**mfsrr1 r**D	**mfspr r**D,**27**
SPRG0–SPRG3	**mtspr** *n*, **r**S	**mtspr** 272 + *n*,**r**S	**mfsprg r**D, *n*	**mfspr r**D,272 + *n*
External access register	**mtear r**S	**mtspr 282,r**S	**mfear r**D	**mfspr r**D,**282**

Table F-19. Simplified Mnemonics for SPRs (Continued)

Special-Purpose Register	Move to SPR		Move from SPR	
	Simplified Mnemonic	Equivalent to	Simplified Mnemonic	Equivalent to
Time base lower	**mttbl r**S	**mtspr 284,r**S	**mftb r**D	**mftb r**D,**268**
Time base upper	**mttbu r**S	**mtspr 285,r**S	**mftbu r**D	**mftb r**D,**269**
Processor version register	—	—	**mfpvr r**D	**mfspr r**D,**287**
IBAT register, upper	**mtibatu** *n,* **r**S	**mtspr** 528 + (2 * *n*),**r**S	**mfibatu r**D*, n*	**mfspr r**D,528 + (2 * *n*)
IBAT register, lower	**mtibatl** *n,* **r**S	**mtspr** 529 + (2 * *n*),**r**S	**mfibatl r**D*, n*	**mfspr r**D,529 + (2 * *n*)
DBAT register, upper	**mtdbatu** *n,* **r**S	**mtspr** 536 + (2 **n*),**r**S	**mfdbatu r**D*, n*	**mfspr r**D,536 + (2 **n*)
DBAT register, lower	**mtdbatl** *n,* **r**S	**mtspr** 537 + (2 * *n*),**r**S	**mfdbatl r**D*, n*	**mfspr r**D,537 + (2 * *n*)

Following are examples using the SPR simplified mnemonics found in Table F-19:

1. Copy the contents of **r**S to the XER.
   mtxer rS (equivalent to **mtspr 1,r**S)
2. Copy the contents of the LR to **r**S.
   mflr rS (equivalent to **mfspr r**S**,8**)
3. Copy the contents of **r**S to the CTR.
   mtctr rS (equivalent to **mtspr 9,r**S)

F.9 Recommended Simplified Mnemonics

This section describes some of the most commonly-used operations (such as no-op, load immediate, load address, move register, and complement register).

F.9.1 No-Op (nop)

Many PowerPC instructions can be coded in a way that, effectively, no operation is performed. An additional mnemonic is provided for the preferred form of no-op. If an implementation performs any type of run-time optimization related to no-ops, the preferred form is the no-op that triggers the following:

nop (equivalent to **ori 0,0,0**)

F.9.2 Load Immediate (li)

The **addi** and **addis** instructions can be used to load an immediate value into a register. Additional mnemonics are provided to convey the idea that no addition is being performed but that data is being moved from the immediate operand of the instruction to a register.

1. Load a 16-bit signed immediate value into **r**D.
   li rD**,**value (equivalent to **addi r**D**,0,**value)

2. Load a 16-bit signed immediate value, shifted left by 16 bits, into **r**D.
   lis rD,value (equivalent to **addis r**D**,0,**value)

F.9.3 Load Address (la)

This mnemonic permits computing the value of a base-displacement operand, using the **addi** instruction which normally requires a separate register and immediate operands.

la rD,d(**r**A) (equivalent to **addi r**D**,r**A**,**d)

The **la** mnemonic is useful for obtaining the address of a variable specified by name, allowing the assembler to supply the base register number and compute the displacement. If the variable *v* is located at offset d*v* bytes from the address in register **r***v,* and the assembler has been told to use register **r***v* as a base for references to the data structure containing *v*, the following line causes the address of *v* to be loaded into register **r**D:

la rD**,***v* (equivalent to **addi r**D**,r***v***,**d*v*

F.9.4 Move Register (mr)

Several PowerPC instructions can be coded to copy the contents of one register to another. A simplified mnemonic is provided that signifies that no computation is being performed, but merely that data is being moved from one register to another.

The following instruction copies the contents of **r**S into **r**A. This mnemonic can be coded with a dot (**.**) suffix to cause the Rc bit to be set in the underlying instruction.

mr rA**,r**S (equivalent to **or r**A**,r**S**,r**S)

F.9.5 Complement Register (not)

Several PowerPC instructions can be coded in a way that they complement the contents of one register and place the result into another register. A simplified mnemonic is provided that allows this operation to be coded easily.

The following instruction complements the contents of **r**S and places the result into **r**A. This mnemonic can be coded with a dot (**.**) suffix to cause the Rc bit to be set in the underlying instruction.

not rA**,r**S (equivalent to **nor r**A**,r**S**,r**S)

F.9.6 Move to Condition Register (mtcr)

This mnemonic permits copying the contents of a GPR to the condition register, using the same syntax as the **mfcr** instruction.

mtcr rS (equivalent to **mtcrf** 0xFF**,r**S)

F