**ECOLE SUPERIEURE D'INGENIEURS
EN ELECTRONIQUE
ET ELECTROTECHNIQUE**

CITÉ DESCARTES - BP 99
93162 NOISY-LE-GRAND CEDEX
TÉL. : 01 45 92 65 00 - FAX : 01 45 92 66 99
INTERNET : www.esiee.fr

Instruction Summary

Année scolaire : 2004-2005

CHAMBRE DE COMMERCE ET D'INDUSTRIE DE PARIS

A

Instruction Summary

This appendix contains PPC403GC instructions summarized alphabetically and by opcode.

- On page A-1, Section A.1 lists all PPC403GC mnemonics, including extended mnemonics, alphabetically. A short functional description is included for each mnemonic.
- On page A-42, Section A.2 lists all PPC403GC instructions, sorted by primary and secondary opcodes. Extended mnemonics are not included in the opcode list.
- On page A-50, Section A.3 illustrates the "Forms" (allowed arrangement of fields within instructions) for PPC403GC instructions.

A.1 Instruction Set and Extended Mnemonics – Alphabetical

Table A-1 summarizes the PPC403GC instruction set, including required extended mnemonics. All mnemonics are listed alphabetically, without regard to whether the mnemonic is realized in hardware or software. When an instruction supports multiple hardware mnemonics (for example, **b**, **ba**, **bl**, **bla** are all forms of **b**), the instruction is alphabetized under the root form. The hardware instructions are described in detail in Chapter 11 (Instruction Set) which is also alphabetized under the root form. Chapter 11 also describes the instruction operands and notation.

Note the following for every Branch Conditional mnemonic:

Bit 4 of the BO field provides a hint about the most likely outcome of a conditional branch (see Section 2.7.5 for a full discussion of Branch Prediction). Assemblers should set $BO_4 = 0$ unless a specific reason exists otherwise. In the BO field values specified in the table below, $BO_4 = 0$ has always been assumed. The assembler must allow the programmer to specify Branch Prediction. To do this, the assembler will support a suffix to every conditional branch mnemonic, as follows:

+ Predict branch to be taken.
– Predict branch not to be taken.

As specific examples, **bc** also could be coded as **bc+** or **bc–**, and **bne** also could be coded **bne+** or **bne–**. These alternate codings set $BO_4 = 1$ only if the requested prediction differs from the Standard Prediction (see Section 2.7.5).

Table A-1. PPC403GC Instruction Syntax Summary

Mnemonic	Operands	Function	Other Registers Changed	Page
add	RT, RA, RB	Add (RA) to (RB). Place result in RT.		11-6
add.			CR[CR0]	
addo			XER[SO, OV]	
addo.			CR[CR0] XER[SO, OV]	
addc	RT, RA, RB	Add (RA) to (RB). Place result in RT. Place carry-out in XER[CA].		11-7
addc.			CR[CR0]	
addco			XER[SO, OV]	
addco.			CR[CR0] XER[SO, OV]	
adde	RT, RA, RB	Add XER[CA], (RA), (RB). Place result in RT. Place carry-out in XER[CA].		11-8
adde.			CR[CR0]	
addeo			XER[SO, OV]	
addeo.			CR[CR0] XER[SO, OV]	
addi	RT, RA, IM	Add EXTS(IM) to (RAI0). Place result in RT.		11-9
addic	RT, RA, IM	Add EXTS(IM) to (RAI0). Place result in RT. Place carry-out in XER[CA].		11-10
addic.	RT, RA, IM	Add EXTS(IM) to (RAI0). Place result in RT. Place carry-out in XER[CA].	CR[CR0]	11-11
addis	RT, RA, IM	Add (IM $\|\|$ $^{16}0$) to (RAI0). Place result in RT.		11-12
addme	RT, RA	Add XER[CA], (RA), (-1). Place result in RT. Place carry-out in XER[CA].		11-13
addme.			CR[CR0]	
addmeo			XER[SO, OV]	
addmeo.			CR[CR0] XER[SO, OV]	

A

Table A-1. PPC403GC Instruction Syntax Summary (cont.)

Mnemonic	Operands	Function	Other Registers Changed	Page
addze	RT, RA	Add XER[CA] to (RA). Place result in RT. Place carry-out in XER[CA].		11-14
addze.			CR[CR0]	
addzeo			XER[SO, OV]	
addzeo.			CR[CR0] XER[SO, OV]	
and	RA, RS, RB	AND (RS) with (RB). Place result in RA.		11-15
and.			CR[CR0]	
andc	RA, RS, RB	AND (RS) with ¬(RB). Place result in RA.		11-16
andc.			CR[CR0]	
andi.	RA, RS, IM	AND (RS) with ($^{16}0$ || IM). Place result in RA.	CR[CR0]	11-17
andis.	RA, RS, IM	AND (RS) with (IM || $^{16}0$). Place result in RA.	CR[CR0]	11-18
b	target	Branch unconditional relative. LI ← $(target - CIA)_{6:29}$ NIA ← CIA + EXTS(LI || $^{2}0$)		11-19
ba		Branch unconditional absolute. LI ← $target_{6:29}$ NIA ← EXTS(LI || $^{2}0$)		
bl		Branch unconditional relative. LI ← $(target - CIA)_{6:29}$ NIA ← CIA + EXTS(LI || $^{2}0$)	(LR) ← CIA + 4.	
bla		Branch unconditional absolute. LI ← $target_{6:29}$ NIA ← EXTS(LI || $^{2}0$)	(LR) ← CIA + 4.	

Table A-1. PPC403GC Instruction Syntax Summary (cont.)

Mnemonic	Operands	Function	Other Registers Changed	Page
bc	BO, BI, target	Branch conditional relative. BD ← (target – CIA)$_{16:29}$ NIA ← CIA + EXTS(BD || $^{2}0$)	CTR if BO_2 = 0.	11-20
bca		Branch conditional absolute. BD ← target$_{16:29}$ NIA ← EXTS(BD || $^{2}0$)	CTR if BO_2 = 0.	
bcl		Branch conditional relative. BD ← (target – CIA)$_{16:29}$ NIA ← CIA + EXTS(BD || $^{2}0$)	CTR if BO_2 = 0. (LR) ← CIA + 4.	
bcla		Branch conditional absolute. BD ← target$_{16:29}$ NIA ← EXTS(BD || $^{2}0$)	CTR if BO_2 = 0. (LR) ← CIA + 4.	
bcctr	BO, BI	Branch conditional to address in CTR. Using (CTR) at exit from instruction, NIA ← CTR$_{0:29}$ || $^{2}0$.	CTR if BO_2 = 0.	11-27
bcctrl			CTR if BO_2 = 0. (LR) ← CIA + 4.	
bclr	BO, BI	Branch conditional to address in LR. Using (LR) at entry to instruction, NIA ← LR$_{0:29}$ || $^{2}0$.	CTR if BO_2 = 0.	11-31
bclrl			CTR if BO_2 = 0. (LR) ← CIA + 4.	
bctr		Branch unconditionally, to address in CTR. *Extended mnemonic* for **bcctr 20,0**		11-27
bctrl		*Extended mnemonic* for **bcctrl 20,0**	(LR) ← CIA + 4.	
bdnz	target	Decrement CTR. Branch if CTR ≠ 0. *Extended mnemonic* for **bc 16,0,target**		11-20
bdnza		*Extended mnemonic* for **bca 16,0,target**		
bdnzl		*Extended mnemonic* for **bcl 16,0,target**	(LR) ← CIA + 4.	
bdnzla		*Extended mnemonic* for **bcla 16,0,target**	(LR) ← CIA + 4.	

A

Table A-1. PPC403GC Instruction Syntax Summary (cont.)

Mnemonic	Operands	Function	Other Registers Changed	Page
bdnzlr		Decrement CTR. Branch if CTR ≠ 0, to address in LR. *Extended mnemonic* for **bclr 16,0**		11-31
bdnzlrl		*Extended mnemonic* for **bclrl 16,0**	(LR) ← CIA + 4.	
bdnzf	cr_bit, target	Decrement CTR. Branch if CTR ≠ 0 AND CR_{cr_bit} = 0. *Extended mnemonic* for **bc 0,cr_bit,target**		11-20
bdnzfa		*Extended mnemonic* for **bca 0,cr_bit,target**		
bdnzfl		*Extended mnemonic* for **bcl 0,cr_bit,target**	(LR) ← CIA + 4.	
bdnzfla		*Extended mnemonic* for **bcla 0,cr_bit,target**	(LR) ← CIA + 4.	
bdnzflr	cr_bit	Decrement CTR. Branch if CTR ≠ 0 AND CR_{cr_bit} = 0, to address in LR. *Extended mnemonic* for **bclr 0,cr_bit**		11-31
bdnzflrl		*Extended mnemonic* for **bclrl 0,cr_bit**	(LR) ← CIA + 4.	
bdnzt	cr_bit, target	Decrement CTR. Branch if CTR ≠ 0 AND CR_{cr_bit} = 1. *Extended mnemonic* for **bc 8,cr_bit,target**		11-20
bdnzta		*Extended mnemonic* for **bca 8,cr_bit,target**		
bdnztl		*Extended mnemonic* for **bcl 8,cr_bit,target**	(LR) ← CIA + 4.	
bdnztla		*Extended mnemonic* for **bcla 8,cr_bit,target**	(LR) ← CIA + 4.	

A

Table A-1. PPC403GC Instruction Syntax Summary (cont.)

Mnemonic	Operands	Function	Other Registers Changed	Page
bdnztlr	cr_bit	Decrement CTR. Branch if CTR ≠ 0 AND CR_{cr_bit} = 1, to address in LR. *Extended mnemonic* for **bclr 8,cr_bit**		11-31
bdnztlrl		*Extended mnemonic* for **bclrl 8,cr_bit**	(LR) ← CIA + 4.	
bdz	target	Decrement CTR. Branch if CTR = 0. *Extended mnemonic* for **bc 18,0,target**		11-20
bdza		*Extended mnemonic* for **bca 18,0,target**		
bdzl		*Extended mnemonic* for **bcl 18,0,target**	(LR) ← CIA + 4.	
bdzla		*Extended mnemonic* for **bcla 18,0,target**	(LR) ← CIA + 4.	
bdzlr		Decrement CTR. Branch if CTR = 0, to address in LR. *Extended mnemonic* for **bclr 18,0**		11-31
bdzlrl		*Extended mnemonic* for **bclrl 18,0**	(LR) ← CIA + 4.	
bdzf	cr_bit, target	Decrement CTR. Branch if CTR = 0 AND CR_{cr_bit} = 0. *Extended mnemonic* for **bc 2,cr_bit,target**		11-20
bdzfa		*Extended mnemonic* for **bca 2,cr_bit,target**		
bdzfl		*Extended mnemonic* for **bcl 2,cr_bit,target**	(LR) ← CIA + 4.	
bdzfla		*Extended mnemonic* for **bcla 2,cr_bit,target**	(LR) ← CIA + 4.	

A

Table A-1. PPC403GC Instruction Syntax Summary (cont.)

Mnemonic	Operands	Function	Other Registers Changed	Page
bdzflr	cr_bit	Decrement CTR. Branch if CTR = 0 AND CR_{cr_bit} = 0 to address in LR. *Extended mnemonic* for **bclr 2,cr_bit**		11-31
bdzflrl		*Extended mnemonic* for **bclrl 2,cr_bit**	(LR) ← CIA + 4.	
bdzt	cr_bit, target	Decrement CTR. Branch if CTR = 0 AND CR_{cr_bit} = 1. *Extended mnemonic* for **bc 10,cr_bit,target**		11-20
bdzta		*Extended mnemonic* for **bca 10,cr_bit,target**		
bdztl		*Extended mnemonic* for **bcl 10,cr_bit,target**	(LR) ← CIA + 4.	
bdztla		*Extended mnemonic* for **bcla 10,cr_bit,target**	(LR) ← CIA + 4.	
bdztlr	cr_bit	Decrement CTR. Branch if CTR = 0 AND CR_{cr_bit} = 1, to address in LR. *Extended mnemonic* for **bclr 10,cr_bit**		11-31
bdztlrl		*Extended mnemonic* for **bclrl 10,cr_bit**	(LR) ← CIA + 4.	
beq	[cr_field,] target	Branch if equal. Use CR0 if cr_field is omitted. *Extended mnemonic* for **bc 12,4*cr_field+2,target**		11-20
beqa		*Extended mnemonic* for **bca 12,4*cr_field+2,target**		
beql		*Extended mnemonic* for **bcl 12,4*cr_field+2,target**	(LR) ← CIA + 4.	
beqla		*Extended mnemonic* for **bcla 12,4*cr_field+2,target**	(LR) ← CIA + 4.	

A

Table A-1. PPC403GC Instruction Syntax Summary (cont.)

Mnemonic	Operands	Function	Other Registers Changed	Page
beqctr	[cr_field]	Branch if equal, to address in CTR. Use CR0 if cr_field is omitted. *Extended mnemonic* for **bcctr 12,4*cr_field+2**		11-27
beqctrl		*Extended mnemonic* for **bcctrl 12,4*cr_field+2**	(LR) ← CIA + 4.	
beqlr	[cr_field]	Branch if equal, to address in LR. Use CR0 if cr_field is omitted. *Extended mnemonic* for **bclr 12,4*cr_field+2**		11-31
beqlrl		*Extended mnemonic* for **bclrl 12,4*cr_field+2**	(LR) ← CIA + 4.	
bf	cr_bit, target	Branch if $CR_{cr\_bit} = 0$. *Extended mnemonic* for **bc 4,cr_bit,target**		11-20
bfa		*Extended mnemonic* for **bca 4,cr_bit,target**		
bfl		*Extended mnemonic* for **bcl 4,cr_bit,target**	(LR) ← CIA + 4.	
bfla		*Extended mnemonic* for **bcla 4,cr_bit,target**	(LR) ← CIA + 4.	
bfctr	cr_bit	Branch if $CR_{cr\_bit} = 0$, to address in CTR. *Extended mnemonic* for **bcctr 4,cr_bit**		11-27
bfctrl		*Extended mnemonic* for **bcctrl 4,cr_bit**	(LR) ← CIA + 4.	
bflr	cr_bit	Branch if $CR_{cr\_bit} = 0$, to address in LR. *Extended mnemonic* for **bclr 4,cr_bit**		11-31
bflrl		*Extended mnemonic* for **bclrl 4,cr_bit**	(LR) ← CIA + 4.	

A

Table A-1. PPC403GC Instruction Syntax Summary (cont.)

Mnemonic	Operands	Function	Other Registers Changed	Page
bge	[cr_field,] target	Branch if greater than or equal. Use CR0 if cr_field is omitted. *Extended mnemonic* for **bc 4,4*cr_field+0,target**		11-20
bgea		*Extended mnemonic* for **bca 4,4*cr_field+0,target**		
bgel		*Extended mnemonic* for **bcl 4,4*cr_field+0,target**	(LR) ← CIA + 4.	
bgela		*Extended mnemonic* for **bcla 4,4*cr_field+0,target**	(LR) ← CIA + 4.	
bgectr	[cr_field]	Branch if greater than or equal, to address in CTR. Use CR0 if cr_field is omitted. *Extended mnemonic* for **bcctr 4,4*cr_field+0**		11-27
bgectrl		*Extended mnemonic* for **bcctrl 4,4*cr_field+0**	(LR) ← CIA + 4.	
bgelr	[cr_field]	Branch if greater than or equal, to address in LR. Use CR0 if cr_field is omitted. *Extended mnemonic* for **bclr 4,4*cr_field+0**		11-31
bgelrl		*Extended mnemonic* for **bclrl 4,4*cr_field+0**	(LR) ← CIA + 4.	
bgt	[cr_field,] target	Branch if greater than. Use CR0 if cr_field is omitted. *Extended mnemonic* for **bc 12,4*cr_field+1,target**		11-20
bgta		*Extended mnemonic* for **bca 12,4*cr_field+1,target**		
bgtl		*Extended mnemonic* for **bcl 12,4*cr_field+1,target**	(LR) ← CIA + 4.	
bgtla		*Extended mnemonic* for **bcla 12,4*cr_field+1,target**	(LR) ← CIA + 4.	

Table A-1. PPC403GC Instruction Syntax Summary (cont.)

Mnemonic	Operands	Function	Other Registers Changed	Page
bgtctr	[cr_field]	Branch if greater than, to address in CTR. Use CR0 if cr_field is omitted. *Extended mnemonic* for **bcctr 12,4*cr_field+1**		11-27
bgtctrl		*Extended mnemonic* for **bcctrl 12,4*cr_field+1**	(LR) ← CIA + 4.	
bgtlr	[cr_field]	Branch if greater than, to address in LR. Use CR0 if cr_field is omitted. *Extended mnemonic* for **bclr 12,4*cr_field+1**		11-31
bgtlrl		*Extended mnemonic* for **bclrl 12,4*cr_field+1**	(LR) ← CIA + 4.	
ble	[cr_field,] target	Branch if less than or equal. Use CR0 if cr_field is omitted. *Extended mnemonic* for **bc 4,4*cr_field+1,target**		11-20
blea		*Extended mnemonic* for **bca 4,4*cr_field+1,target**		
blel		*Extended mnemonic* for **bcl 4,4*cr_field+1,target**	(LR) ← CIA + 4.	
blela		*Extended mnemonic* for **bcla 4,4*cr_field+1,target**	(LR) ← CIA + 4.	
blectr	[cr_field]	Branch if less than or equal, to address in CTR. Use CR0 if cr_field is omitted. *Extended mnemonic* for **bcctr 4,4*cr_field+1**		11-27
blectrl		*Extended mnemonic* for **bcctrl 4,4*cr_field+1**	(LR) ← CIA + 4.	
blelr	[cr_field]	Branch if less than or equal, to address in LR. Use CR0 if cr_field is omitted. *Extended mnemonic* for **bclr 4,4*cr_field+1**		11-31
blelrl		*Extended mnemonic* for **bclrl 4,4*cr_field+1**	(LR) ← CIA + 4.	

A

Table A-1. PPC403GC Instruction Syntax Summary (cont.)

Mnemonic	Operands	Function	Other Registers Changed	Page
blr		Branch unconditionally, to address in LR. *Extended mnemonic* for **bclr 20,0**		11-31
blrl		*Extended mnemonic* for **bclrl 20,0**	(LR) ← CIA + 4.	
blt	[cr_field,] target	Branch if less than. Use CR0 if cr_field is omitted. *Extended mnemonic* for **bc 12,4*cr_field+0,target**		11-20
blta		*Extended mnemonic* for **bca 12,4*cr_field+0,target**		
bltl		*Extended mnemonic* for **bcl 12,4*cr_field+0,target**	(LR) ← CIA + 4.	
bltla		*Extended mnemonic* for **bcla 12,4*cr_field+0,target**	(LR) ← CIA + 4.	
bltctr	[cr_field]	Branch if less than, to address in CTR. Use CR0 if cr_field is omitted. *Extended mnemonic* for **bcctr 12,4*cr_field+0**		11-27
bltctrl		*Extended mnemonic* for **bcctrl 12,4*cr_field+0**	(LR) ← CIA + 4.	
bltlr	[cr_field]	Branch if less than, to address in LR. Use CR0 if cr_field is omitted. *Extended mnemonic* for **bclr 12,4*cr_field+0**		11-31
bltlrl		*Extended mnemonic* for **bclrl 12,4*cr_field+0**	(LR) ← CIA + 4.	

A

Table A-1. PPC403GC Instruction Syntax Summary (cont.)

Mnemonic	Operands	Function	Other Registers Changed	Page
bne	[cr_field,] target	Branch if not equal. Use CR0 if cr_field is omitted. *Extended mnemonic* for **bc 4,4*cr_field+2,target**		11-20
bnea		*Extended mnemonic* for **bca 4,4*cr_field+2,target**		
bnel		*Extended mnemonic* for **bcl 4,4*cr_field+2,target**	(LR) ← CIA + 4.	
bnela		*Extended mnemonic* for **bcla 4,4*cr_field+2,target**	(LR) ← CIA + 4.	
bnectr	[cr_field]	Branch if not equal, to address in CTR. Use CR0 if cr_field is omitted. *Extended mnemonic* for **bcctr 4,4*cr_field+2**		11-27
bnectrl		*Extended mnemonic* for **bcctrl 4,4*cr_field+2**	(LR) ← CIA + 4.	
bnelr	[cr_field]	Branch if not equal, to address in LR. Use CR0 if cr_field is omitted. *Extended mnemonic* for **bclr 4,4*cr_field+2**		11-31
bnelrl		*Extended mnemonic* for **bclrl 4,4*cr_field+2**	(LR) ← CIA + 4.	
bng	[cr_field,] target	Branch if not greater than. Use CR0 if cr_field is omitted. *Extended mnemonic* for **bc 4,4*cr_field+1,target**		11-20
bnga		*Extended mnemonic* for **bca 4,4*cr_field+1,target**		
bngl		*Extended mnemonic* for **bcl 4,4*cr_field+1,target**	(LR) ← CIA + 4.	
bngla		*Extended mnemonic* for **bcla 4,4*cr_field+1,target**	(LR) ← CIA + 4.	

A

Table A-1. PPC403GC Instruction Syntax Summary (cont.)

Mnemonic	Operands	Function	Other Registers Changed	Page
bngctr	[cr_field]	Branch if not greater than, to address in CTR. Use CR0 if cr_field is omitted. *Extended mnemonic* for **bcctr 4,4*cr_field+1**		11-27
bngctrl		*Extended mnemonic* for **bcctrl 4,4*cr_field+1**	(LR) ← CIA + 4.	
bnglr	[cr_field]	Branch if not greater than, to address in LR. Use CR0 if cr_field is omitted. *Extended mnemonic* for **bclr 4,4*cr_field+1**		11-31
bnglrl		*Extended mnemonic* for **bclrl 4,4*cr_field+1**	(LR) ← CIA + 4.	
bnl	[cr_field,] target	Branch if not less than. Use CR0 if cr_field is omitted. *Extended mnemonic* for **bc 4,4*cr_field+0,target**		11-20
bnla		*Extended mnemonic* for **bca 4,4*cr_field+0,target**		
bnll		*Extended mnemonic* for **bcl 4,4*cr_field+0,target**	(LR) ← CIA + 4.	
bnlla		*Extended mnemonic* for **bcla 4,4*cr_field+0,target**	(LR) ← CIA + 4.	
bnlctr	[cr_field]	Branch if not less than, to address in CTR. Use CR0 if cr_field is omitted. *Extended mnemonic* for **bcctr 4,4*cr_field+0**		11-27
bnlctrl		*Extended mnemonic* for **bcctrl 4,4*cr_field+0**	(LR) ← CIA + 4.	
bnllr	[cr_field]	Branch if not less than, to address in LR. Use CR0 if cr_field is omitted. *Extended mnemonic* for **bclr 4,4*cr_field+0**		11-31
bnllrl		*Extended mnemonic* for **bclrl 4,4*cr_field+0**	(LR) ← CIA + 4.	

A

Table A-1. PPC403GC Instruction Syntax Summary (cont.)

Mnemonic	Operands	Function	Other Registers Changed	Page
bns	[cr_field,] target	Branch if not summary overflow. Use CR0 if cr_field is omitted. *Extended mnemonic* for **bc 4,4*cr_field+3,target**		11-20
bnsa		*Extended mnemonic* for **bca 4,4*cr_field+3,target**		
bnsl		*Extended mnemonic* for **bcl 4,4*cr_field+3,target**	(LR) ← CIA + 4.	
bnsla		*Extended mnemonic* for **bcla 4,4*cr_field+3,target**	(LR) ← CIA + 4.	
bnsctr	[cr_field]	Branch if not summary overflow, to address in CTR. Use CR0 if cr_field is omitted. *Extended mnemonic* for **bcctr 4,4*cr_field+3**		11-27
bnsctrl		*Extended mnemonic* for **bcctrl 4,4*cr_field+3**	(LR) ← CIA + 4.	
bnslr	[cr_field]	Branch if not summary overflow, to address in LR. Use CR0 if cr_field is omitted. *Extended mnemonic* for **bclr 4,4*cr_field+3**		11-31
bnslrl		*Extended mnemonic* for **bclrl 4,4*cr_field+3**	(LR) ← CIA + 4.	
bnu	[cr_field,] target	Branch if not unordered. Use CR0 if cr_field is omitted. *Extended mnemonic* for **bc 4,4*cr_field+3,target**		11-20
bnua		*Extended mnemonic* for **bca 4,4*cr_field+3,target**		
bnul		*Extended mnemonic* for **bcl 4,4*cr_field+3,target**	(LR) ← CIA + 4.	
bnula		*Extended mnemonic* for **bcla 4,4*cr_field+3,target**	(LR) ← CIA + 4.	

A

Table A-1. PPC403GC Instruction Syntax Summary (cont.)

Mnemonic	Operands	Function	Other Registers Changed	Page
bnuctr	[cr_field]	Branch if not unordered, to address in CTR. Use CR0 if cr_field is omitted. *Extended mnemonic* for **bcctr 4,4*cr_field+3**		11-27
bnuctrl		*Extended mnemonic* for **bcctrl 4,4*cr_field+3**	(LR) ← CIA + 4.	
bnulr	[cr_field]	Branch if not unordered, to address in LR. Use CR0 if cr_field is omitted. *Extended mnemonic* for **bclr 4,4*cr_field+3**		11-31
bnulrl		*Extended mnemonic* for **bclrl 4,4*cr_field+3**	(LR) ← CIA + 4.	
bso	[cr_field,] target	Branch if summary overflow. Use CR0 if cr_field is omitted. *Extended mnemonic* for **bc 12,4*cr_field+3,target**		11-20
bsoa		*Extended mnemonic* for **bca 12,4*cr_field+3,target**		
bsol		*Extended mnemonic* for **bcl 12,4*cr_field+3,target**	(LR) ← CIA + 4.	
bsola		*Extended mnemonic* for **bcla 12,4*cr_field+3,target**	(LR) ← CIA + 4.	
bsoctr	[cr_field]	Branch if summary overflow, to address in CTR. Use CR0 if cr_field is omitted. *Extended mnemonic* for **bcctr 12,4*cr_field+3**		11-27
bsoctrl		*Extended mnemonic* for **bcctrl 12,4*cr_field+3**	(LR) ← CIA + 4.	
bsolr	[cr_field]	Branch if summary overflow, to address in LR. Use CR0 if cr_field is omitted. *Extended mnemonic* for **bclr 12,4*cr_field+3**		11-31
bsolrl		*Extended mnemonic* for **bclrl 12,4*cr_field+3**	(LR) ← CIA + 4.	

A

Table A-1. PPC403GC Instruction Syntax Summary (cont.)

Mnemonic	Operands	Function	Other Registers Changed	Page
bt	cr_bit, target	Branch if CR_{cr_bit} = 1. *Extended mnemonic* for **bc 12,cr_bit,target**		11-20
bta		*Extended mnemonic* for **bca 12,cr_bit,target**		
btl		*Extended mnemonic* for **bcl 12,cr_bit,target**	(LR) ← CIA + 4.	
btla		*Extended mnemonic* for **bcla 12,cr_bit,target**	(LR) ← CIA + 4.	
btctr	cr_bit	Branch if CR_{cr_bit} = 1, to address in CTR. *Extended mnemonic* for **bcctr 12,cr_bit**		11-27
btctrl		*Extended mnemonic* for **bcctrl 12,cr_bit**	(LR) ← CIA + 4.	
btlr	cr_bit	Branch if CR_{cr_bit} = 1, to address in LR. *Extended mnemonic* for **bclr 12,cr_bit**		11-31
btlrl		*Extended mnemonic* for **bclrl 12,cr_bit**	(LR) ← CIA + 4.	
bun	[cr_field,] target	Branch if unordered. Use CR0 if cr_field is omitted. *Extended mnemonic* for **bc 12,4*cr_field+3,target**		11-20
buna		*Extended mnemonic* for **bca 12,4*cr_field+3,target**		
bunl		*Extended mnemonic* for **bcl 12,4*cr_field+3,target**	(LR) ← CIA + 4.	
bunla		*Extended mnemonic* for **bcla 12,4*cr_field+3,target**	(LR) ← CIA + 4.	
bunctr	[cr_field]	Branch if unordered, to address in CTR. Use CR0 if cr_field is omitted. *Extended mnemonic* for **bcctr 12,4*cr_field+3**		11-27
bunctrl		*Extended mnemonic* for **bcctrl 12,4*cr_field+3**	(LR) ← CIA + 4.	

A

Table A-1. PPC403GC Instruction Syntax Summary (cont.)

Mnemonic	Operands	Function	Other Registers Changed	Page
bunlr	[cr_field]	Branch if unordered, to address in LR. Use CR0 if cr_field is omitted. *Extended mnemonic* for **bclr 12,4*cr_field+3**		11-31
bunlrl		*Extended mnemonic* for **bclrl 12,4*cr_field+3**	(LR) ← CIA + 4.	
clrlwi	RA, RS, n	Clear left immediate. (n < 32) $(RA)_{0:n-1} \leftarrow {}^{n}0$ *Extended mnemonic* for **rlwinm RA,RS,0,n,31**		11-130
clrlwi.		*Extended mnemonic* for **rlwinm. RA,RS,0,n,31**	CR[CR0]	
clrlslwi	RA, RS, b, n	Clear left and shift left immediate. (n ≤ b < 32) $(RA)_{b-n:31-n} \leftarrow (RS)_{b:31}$ $(RA)_{32-n:31} \leftarrow {}^{n}0$ $(RA)_{0:b-n-1} \leftarrow {}^{b-n}0$ *Extended mnemonic* for **rlwinm RA,RS,n,b–n,31–n**		11-130
clrlslwi.		*Extended mnemonic* for **rlwinm. RA,RS,n,b–n,31–n**	CR[CR0]	
clrrwi	RA, RS, n	Clear right immediate. (n < 32) $(RA)_{32-n:31} \leftarrow {}^{n}0$ *Extended mnemonic* for **rlwinm RA,RS,0,0,31–n**		11-130
clrrwi.		*Extended mnemonic* for **rlwinm. RA,RS,0,0,31–n**	CR[CR0]	
cmp	BF, 0, RA, RB	Compare (RA) to (RB), signed. Results in CR[CRn], where n = BF.		11-36
cmpi	BF, 0, RA, IM	Compare (RA) to EXTS(IM), signed. Results in CR[CRn], where n = BF.		11-37
cmpl	BF, 0, RA, RB	Compare (RA) to (RB), unsigned. Results in CR[CRn], where n = BF.		11-38
cmpli	BF, 0, RA, IM	Compare (RA) to ($^{16}0$ || IM), unsigned. Results in CR[CRn], where n = BF.		11-39

A

Table A-1. PPC403GC Instruction Syntax Summary (cont.)

Mnemonic	Operands	Function	Other Registers Changed	Page
cmplw	[BF,] RA, RB	Compare Logical Word. Use CR0 if BF is omitted. *Extended mnemonic* for **cmpl BF,0,RA,RB**		11-38
cmplwi	[BF,] RA, IM	Compare Logical Word Immediate. Use CR0 if BF is omitted. *Extended mnemonic* for **cmpli BF,0,RA,IM**		11-39
cmpw	[BF,] RA, RB	Compare Word. Use CR0 if BF is omitted. *Extended mnemonic* for **cmp BF,0,RA,RB**		11-36
cmpwi	[BF,] RA, IM	Compare Word Immediate. Use CR0 if BF is omitted. *Extended mnemonic* for **cmpi BF,0,RA,IM**		11-37
cntlzw	RA, RS	Count leading zeros in RS. Place result in RA.		11-40
cntlzw.			CR[CR0]	
crand	BT, BA, BB	AND bit (CR_{BA}) with (CR_{BB}). Place result in CR_{BT}.		11-41
crandc	BT, BA, BB	AND bit (CR_{BA}) with $\neg(CR_{BB})$. Place result in CR_{BT}.		11-42
crclr	bx	Condition register clear. *Extended mnemonic* for **crxor bx,bx,bx**		11-48
creqv	BT, BA, BB	Equivalence of bit CR_{BA} with CR_{BB}. $CR_{BT} \leftarrow \neg(CR_{BA} \oplus CR_{BB})$		11-43
crmove	bx, by	Condition register move. *Extended mnemonic* for **cror bx,by,by**		11-46
crnand	BT, BA, BB	NAND bit (CR_{BA}) with (CR_{BB}). Place result in CR_{BT}.		11-44
crnor	BT, BA, BB	NOR bit (CR_{BA}) with (CR_{BB}). Place result in CR_{BT}.		11-45
crnot	bx, by	Condition register not. *Extended mnemonic* for **crnor bx,by,by**		11-45

A

Table A-1. PPC403GC Instruction Syntax Summary (cont.)

Mnemonic	Operands	Function	Other Registers Changed	Page
cror	BT, BA, BB	OR bit (CR_{BA}) with (CR_{BB}). Place result in CR_{BT}.		11-46
crorc	BT, BA, BB	OR bit (CR_{BA}) with ¬(CR_{BB}). Place result in CR_{BT}.		11-47
crset	bx	Condition register set. *Extended mnemonic* for **creqv bx,bx,bx**		11-43
crxor	BT, BA, BB	XOR bit (CR_{BA}) with (CR_{BB}). Place result in CR_{BT}.		11-48
dcbf	RA, RB	Flush (store, then invalidate) the data cache block which contains the effective address (RA|0) + (RB).		11-49
dcbi	RA, RB	Invalidate the data cache block which contains the effective address (RA|0) + (RB).		11-50
dcbst	RA, RB	Store the data cache block which contains the effective address (RA|0) + (RB).		11-51
dcbt	RA, RB	Load the data cache block which contains the effective address (RA|0) + (RB).		11-52
dcbtst	RA,RB	Load the data cache block which contains the effective address (RA|0) + (RB).		11-54
dcbz	RA, RB	Zero the data cache block which contains the effective address (RA|0) + (RB).		11-56
dccci	RA, RB	Invalidate the data cache congruence class associated with the effective address (RA|0) + (RB).		11-58
dcread	RT, RA, RB	Read either tag or data information from the data cache congruence class associated with the effective address (RA|0) + (RB). Place the results in RT.		11-60
divw	RT, RA, RB	Divide (RA) by (RB), signed. Place result in RT.		11-62
divw.			CR[CR0]	
divwo			XER[SO, OV]	
divwo.			CR[CR0] XER[SO, OV]	

A

Table A-1. PPC403GC Instruction Syntax Summary (cont.)

Mnemonic	Operands	Function	Other Registers Changed	Page
divwu	RT, RA, RB	Divide (RA) by (RB), unsigned. Place result in RT.		11-63
divwu.			CR[CR0]	
divwuo			XER[SO, OV]	
divwuo.			CR[CR0] XER[SO, OV]	
eieio		Storage synchronization. All loads and stores that precede the **eieio** instruction complete before any loads and stores that follow the instruction access main storage. Implemented as **sync**, which is more restrictive.		11-64
eqv	RA, RS, RB	Equivalence of (RS) with (RB). (RA) ← ¬((RS) ⊕ (RB))		11-65
eqv.			CR[CR0]	
extlwi	RA, RS, n, b	Extract and left justify immediate. (n > 0) $(RA)_{0:n-1} \leftarrow (RS)_{b:b+n-1}$ $(RA)_{n:31} \leftarrow {}^{32-n}0$ *Extended mnemonic* for **rlwinm RA,RS,b,0,n–1**		11-130
extlwi.		*Extended mnemonic* for **rlwinm. RA,RS,b,0,n–1**	CR[CR0]	
extrwi	RA, RS, n, b	Extract and right justify immediate. (n > 0) $(RA)_{32-n:31} \leftarrow (RS)_{b:b+n-1}$ $(RA)_{0:31-n} \leftarrow {}^{32-n}0$ *Extended mnemonic* for **rlwinm RA,RS,b+n,32–n,31**		11-130
extrwi.		*Extended mnemonic* for **rlwinm. RA,RS,b+n,32–n,31**	CR[CR0]	
extsb	RA, RS	Extend the sign of byte $(RS)_{24:31}$. Place the result in RA.		11-66
extsb.			CR[CR0]	
extsh	RA, RS	Extend the sign of halfword $(RS)_{16:31}$. Place the result in RA.		11-67
extsh.			CR[CR0]	
icbi	RA, RB	Invalidate the instruction cache block which contains the effective address (RAI0) + (RB).		11-68
icbt	RA, RB	Load the instruction cache block which contains the effective address (RAI0) + (RB).		11-70

A

Table A-1. PPC403GC Instruction Syntax Summary (cont.)

Mnemonic	Operands	Function	Other Registers Changed	Page
iccci	RA, RB	Invalidate instruction cache congruence class associated with the effective address (RAI0) + (RB).		11-72
icread	RA, RB	Read either tag or data information from the instruction cache congruence class associated with the effective address (RAI0) + (RB). Place the results in ICDBDR.		11-74
inslwi	RA, RS, n, b	Insert from left immediate. (n > 0) $(RA)_{b:b+n-1} \leftarrow (RS)_{0:n-1}$ *Extended mnemonic* for **rlwimi RA,RS,32–b,b,b+n–1**		11-129
inslwi.		*Extended mnemonic* for **rlwimi. RA,RS,32–b,b,b+n–1**	CR[CR0]	
insrwi	RA, RS, n, b	Insert from right immediate. (n > 0) $(RA)_{b:b+n-1} \leftarrow (RS)_{32-n:31}$ *Extended mnemonic* for **rlwimi RA,RS,32–b–n,b,b+n–1**		11-129
insrwi.		*Extended mnemonic* for **rlwimi. RA,RS,32–b–n,b,b+n–1**	CR[CR0]	
isync		Synchronize execution context by flushing the prefetch queue.		11-76
la	RT, D(RA)	Load address. (RA ≠ 0) D is an offset from a base address that is assumed to be (RA). (RT) ← (RA) + EXTS(D) *Extended mnemonic* for **addi RT,RA,D**		11-9
lbz	RT, D(RA)	Load byte from EA = (RAI0) + EXTS(D) and pad left with zeroes, (RT) ← $^{24}0$ || MS(EA,1).		11-77
lbzu	RT, D(RA)	Load byte from EA = (RAI0) + EXTS(D) and pad left with zeroes, (RT) ← $^{24}0$ || MS(EA,1). Update the base address, (RA) ← EA.		11-78

Table A-1. PPC403GC Instruction Syntax Summary (cont.)

Mnemonic	Operands	Function	Other Registers Changed	Page
lbzux	RT, RA, RB	Load byte from EA = (RA|0) + (RB) and pad left with zeroes, (RT) ← $^{24}0$ || MS(EA,1). Update the base address, (RA) ← EA.		11-79
lbzx	RT, RA, RB	Load byte from EA = (RA|0) + (RB) and pad left with zeroes, (RT) ← $^{24}0$ || MS(EA,1).		11-80
lha	RT, D(RA)	Load halfword from EA = (RA|0) + EXTS(D) and sign extend, (RT) ← EXTS(MS(EA,2)).		11-81
lhau	RT, D(RA)	Load halfword from EA = (RA|0) + EXTS(D) and sign extend, (RT) ← EXTS(MS(EA,2)). Update the base address, (RA) ← EA.		11-82
lhaux	RT, RA, RB	Load halfword from EA = (RA|0) + (RB) and sign extend, (RT) ← EXTS(MS(EA,2)). Update the base address, (RA) ← EA.		11-83
lhax	RT, RA, RB	Load halfword from EA = (RA|0) + (RB) and sign extend, (RT) ← EXTS(MS(EA,2)).		11-84
lhbrx	RT, RA, RB	Load halfword from EA = (RA|0) + (RB) then reverse byte order and pad left with zeroes, (RT) ← $^{16}0$ || MS(EA+1,1) || MS(EA,1).		11-85
lhz	RT, D(RA)	Load halfword from EA = (RA|0) + EXTS(D) and pad left with zeroes, (RT) ← $^{16}0$ || MS(EA,2).		11-86
lhzu	RT, D(RA)	Load halfword from EA = (RA|0) + EXTS(D) and pad left with zeroes, (RT) ← $^{16}0$ || MS(EA,2). Update the base address, (RA) ← EA.		11-87

A

Table A-1. PPC403GC Instruction Syntax Summary (cont.)

Mnemonic	Operands	Function	Other Registers Changed	Page
lhzux	RT, RA, RB	Load halfword from EA = (RA|0) + (RB) and pad left with zeroes, (RT) ← $^{16}0$ || MS(EA,2). Update the base address, (RA) ← EA.		11-88
lhzx	RT, RA, RB	Load halfword from EA = (RA|0) + (RB) and pad left with zeroes, (RT) ← $^{16}0$ || MS(EA,2).		11-89
li	RT, IM	Load immediate. (RT) ← EXTS(IM) *Extended mnemonic* for **addi RT,0,value**		11-9
lis	RT, IM	Load immediate shifted. (RT) ← (IM || $^{16}0$) *Extended mnemonic* for **addis RT,0,value**		11-12
lmw	RT, D(RA)	Load multiple words starting from EA = (RA|0) + EXTS(D). Place into consecutive registers, RT through GPR(31). RA is not altered unless RA = GPR(31).		11-90
lswi	RT, RA, NB	Load consecutive bytes from EA=(RA|0). Number of bytes n=32 if NB=0, else n=NB. Stack bytes into words in CEIL(n/4) consecutive registers starting with RT, to R_{FINAL} ← ((RT + CEIL(n/4) – 1) % 32). GPR(0) is consecutive to GPR(31). RA is not altered unless RA = R_{FINAL}.		11-91
lswx	RT, RA, RB	Load consecutive bytes from EA=(RA|0)+(RB). Number of bytes n=XER[TBC]. Stack bytes into words in CEIL(n/4) consecutive registers starting with RT, to R_{FINAL} ← ((RT + CEIL(n/4) – 1) % 32). GPR(0) is consecutive to GPR(31). RA is not altered unless RA = R_{FINAL}. RB is not altered unless RB = R_{FINAL}. If n=0, content of RT is undefined.		11-93
lwarx	RT, RA, RB	Load word from EA = (RA|0) + (RB) and place in RT, (RT) ← MS(EA,4). Set the Reservation bit.		11-95

Table A-1. PPC403GC Instruction Syntax Summary (cont.)

Mnemonic	Operands	Function	Other Registers Changed	Page
lwbrx	RT, RA, RB	Load word from EA = (RAI0) + (RB) then reverse byte order, (RT) ← MS(EA+3,1) II MS(EA+2,1) II MS(EA+1,1) II MS(EA,1).		11-96
lwz	RT, D(RA)	Load word from EA = (RAI0) + EXTS(D) and place in RT, (RT) ← MS(EA,4).		11-97
lwzu	RT, D(RA)	Load word from EA = (RAI0) + EXTS(D) and place in RT, (RT) ← MS(EA,4). Update the base address, (RA) ← EA.		11-98
lwzux	RT, RA, RB	Load word from EA = (RAI0) + (RB) and place in RT, (RT) ← MS(EA,4). Update the base address, (RA) ← EA.		11-99
lwzx	RT, RA, RB	Load word from EA = (RAI0) + (RB) and place in RT, (RT) ← MS(EA,4).		11-100
mcrf	BF, BFA	Move CR field, (CR[CRn]) ← (CR[CRm]) where m ← BFA and n ← BF.		11-101
mcrxr	BF	Move XER[0:3] into field CRn, where n←BF. CR[CRn] ← (XER[SO, OV, CA]). (XER[SO, OV, CA]) ← $^{3}0$.		11-102
mfcr	RT	Move from CR to RT, (RT) ← (CR).		11-103

A

Table A-1. PPC403GC Instruction Syntax Summary (cont.)

Mnemonic	Operands	Function	Other Registers Changed	Page
mfbear **mfbesr** **mfbr0** **mfbr1** **mfbr2** **mfbr3** **mfbr4** **mfbr5** **mfbr6** **mfbr7** **mfdmacc0** **mfdmacc1** **mfdmacc2** **mfdmacc3** **mfdmacr0** **mfdmacr1** **mfdmacr2** **mfdmacr3** **mfdmact0** **mfdmact1** **mfdmact2** **mfdmact3** **mfdmada0** **mfdmada1** **mfdmada2** **mfdmada3** **mfdmasa0** **mfdmasa1** **mfdmasa2** **mfdmasa3** **mfdmasr** **mfexisr** **mfexier** **mfiocr**	RT	Move from device control register DCRN. *Extended mnemonic* for **mfdcr RT,DCRN** See Table 12-3 on page 12-4 for listing of valid DCRN values.		11-104
mfdcr	RT, DCRN	Move from DCR to RT, (RT) ← (DCR(DCRN)).		11-104
mfmsr	RT	Move from MSR to RT, (RT) ← (MSR).		11-106

A

Table A-1. PPC403GC Instruction Syntax Summary (cont.)

Mnemonic	Operands	Function	Other Registers Changed	Page
mfcdbcr **mfctr** **mfdac1** **mfdac2** **mfdbsr** **mfdccr** **mfdcwr** **mfdear** **mfesr** **mfevpr** **mfiac1** **mfiac2** **mficcr** **mficdbdr** **mflr** **mfpbl1** **mfpbl2** **mfpbu1** **mfpbu2** **mfpid** **mfpit** **mfpvr** **mfsgr** **mfsprg0** **mfsprg1** **mfsprg2** **mfsprg3** **mfsrr0** **mfsrr1** **mfsrr2** **mfsrr3** **mftbhi** **mftbhu** **mftblo** **mftblu** **mftcr** **mftsr** **mfxer** **mfzpr**	RT	Move from special purpose register SPRN. *Extended mnemonic* for **mfspr RT,SPRN** See Table 12-2 on page 12-2 for listing of valid SPRN values.		11-107
mfspr	RT, SPRN	Move from SPR to RT, (RT) ← (SPR(SPRN)).		11-107

A

Table A-1. PPC403GC Instruction Syntax Summary (cont.)

Mnemonic	Operands	Function	Other Registers Changed	Page
mr	RT, RS	Move register. (RT) ← (RS) *Extended mnemonic* for **or RT,RS,RS**		11-123
mr.		*Extended mnemonic* for **or. RT,RS,RS**	CR[CR0]	
mtcr	RS	Move to Condition Register. *Extended mnemonic* for **mtcrf 0xFF,RS**		11-109
mtcrf	FXM, RS	Move some or all of the contents of RS into CR as specified by FXM field, mask ← $^4(FXM_0) \parallel {}^4(FXM_1) \parallel \ldots \parallel {}^4(FXM_6) \parallel {}^4(FXM_7)$. (CR)←((RS) ∧ mask) ∨ (CR) ∧ ¬mask).		11-109

Table A-1. PPC403GC Instruction Syntax Summary (cont.)

Mnemonic	Operands	Function	Other Registers Changed	Page
mtbear **mtbesr** **mtbr0** **mtbr1** **mtbr2** **mtbr3** **mtbr4** **mtbr5** **mtbr6** **mtbr7** **mtdmacc0** **mtdmacc1** **mtdmacc2** **mtdmacc3** **mtdmacr0** **mtdmacr1** **mtdmacr2** **mtdmacr3** **mtdmact0** **mtdmact1** **mtdmact2** **mtdmact3** **mtdmada0** **mtdmada1** **mtdmada2** **mtdmada3** **mtdmasa0** **mtdmasa1** **mtdmasa2** **mtdmasa3** **mtdmasr** **mtexisr** **mtexier** **mtiocr**	RS	Move to device control register DCRN. *Extended mnemonic* for **mtdcr DCRN,RS** See Table 12-3 on page 12-4 for listing of valid DCRN values.		11-111
mtdcr	DCRN, RS	Move to DCR from RS, (DCR(DCRN)) ← (RS).		11-111
mtmsr	RS	Move to MSR from RS, (MSR) ← (RS).		11-113

A

Table A-1. PPC403GC Instruction Syntax Summary (cont.)

Mnemonic	Operands	Function	Other Registers Changed	Page
mtcdbcr **mtctr** **mtdac1** **mtdac2** **mtdbsr** **mtdccr** **mtdcwr** **mtesr** **mtevpr** **mtiac1** **mtiac2** **mticcr** **mticdbdr** **mtlr** **mtpbl1** **mtpbl2** **mtpbu1** **mtpbu2** **mtpid** **mtpit** **mtsgr** **mtsprg0** **mtsprg1** **mtsprg2** **mtsprg3** **mtsrr0** **mtsrr1** **mtsrr2** **mtsrr3** **mttbhi** **mttblo** **mttcr** **mttsr** **mtxer** **mtzpr**	RS	Move to special purpose register SPRN. *Extended mnemonic* for **mtspr SPRN,RS** See Table 12-2 on page 12-2 for listing of valid SPRN values.		11-114
mtspr	SPRN, RS	Move to SPR from RS, (SPR(SPRN)) ← (RS).		11-114
mulhw	RT, RA, RB	Multiply (RA) and (RB), signed. Place hi-order result in RT. $prod_{0:63}$ ← (RA) × (RB) (signed). (RT) ← $prod_{0:31}$.		11-116
mulhw.			CR[CR0]	

Table A-1. PPC403GC Instruction Syntax Summary (cont.)

Mnemonic	Operands	Function	Other Registers Changed	Page
mulhwu	RT, RA, RB	Multiply (RA) and (RB), unsigned. Place hi-order result in RT. $prod_{0:63} \leftarrow$ (RA) $\times$ (RB) (unsigned). (RT) $\leftarrow prod_{0:31}$.		11-117
mulhwu.			CR[CR0]	
mulli	RT, RA, IM	Multiply (RA) and IM, signed. Place lo-order result in RT. $prod_{0:47} \leftarrow$ (RA) $\times$ IM (signed) (RT) $\leftarrow prod_{16:47}$		11-118
mullw	RT, RA, RB	Multiply (RA) and (RB), signed. Place lo-order result in RT. $prod_{0:63} \leftarrow$ (RA) $\times$ (RB) (signed). (RT) $\leftarrow prod_{32:63}$.		11-119
mullw.			CR[CR0]	
mullwo			XER[SO, OV]	
mullwo.			CR[CR0] XER[SO, OV]	
nand	RA, RS, RB	NAND (RS) with (RB). Place result in RA.		11-120
nand.			CR[CR0]	
neg	RT, RA	Negative (two's complement) of RA. (RT) $\leftarrow \neg$(RA) + 1		11-121
neg.			CR[CR0]	
nego			XER[SO, OV]	
nego.			CR[CR0] XER[SO, OV]	
nop		Preferred no-op, triggers optimizations based on no-ops. *Extended mnemonic* for **ori 0,0,0**		11-125
nor	RA, RS, RB	NOR (RS) with (RB). Place result in RA.		11-122
nor.			CR[CR0]	
not	RA, RS	Complement register. (RA) $\leftarrow \neg$(RS) *Extended mnemonic* for **nor RA,RS,RS**		11-122
not.		*Extended mnemonic* for **nor. RA,RS,RS**	CR[CR0]	
or	RA, RS, RB	OR (RS) with (RB). Place result in RA.		11-123
or.			CR[CR0]	

A

Table A-1. PPC403GC Instruction Syntax Summary (cont.)

Mnemonic	Operands	Function	Other Registers Changed	Page
orc	RA, RS, RB	OR (RS) with ¬(RB). Place result in RA.		11-124
orc.			CR[CR0]	
ori	RA, RS, IM	OR (RS) with ($^{16}0$ || IM). Place result in RA.		11-125
oris	RA, RS, IM	OR (RS) with (IM || $^{16}0$). Place result in RA.		11-126
rfci		Return from critical interrupt (PC) ← (SRR2). (MSR) ← (SRR3).		11-127
rfi		Return from interrupt. (PC) ← (SRR0). (MSR) ← (SRR1).		11-128
rlwimi	RA, RS, SH, MB, ME	Rotate left word immediate, then insert according to mask. r ← ROTL((RS), SH) m ← MASK(MB, ME) (RA) ← (r ∧ m) ∨ ((RA) ∧ ¬m)		11-129
rlwimi.			CR[CR0]	
rlwinm	RA, RS, SH, MB, ME	Rotate left word immediate, then AND with mask. r ← ROTL((RS), SH) m ← MASK(MB, ME) (RA) ← (r ∧ m)		11-130
rlwinm.			CR[CR0]	
rlwnm	RA, RS, RB, MB, ME	Rotate left word, then AND with mask. r ← ROTL((RS), $(RB)_{27:31}$) m ← MASK(MB, ME) (RA) ← (r ∧ m)		11-133
rlwnm.			CR[CR0]	
rotlw	RA, RS, RB	Rotate left. (RA) ← ROTL((RS), $(RB)_{27:31}$) *Extended mnemonic* for **rlwnm RA,RS,RB,0,31**		11-133
rotlw.		*Extended mnemonic* for **rlwnm. RA,RS,RB,0,31**	CR[CR0]	
rotlwi	RA, RS, n	Rotate left immediate. (RA) ← ROTL((RS), n) *Extended mnemonic* for **rlwinm RA,RS,n,0,31**		11-130
rotlwi.		*Extended mnemonic* for **rlwinm. RA,RS,n,0,31**	CR[CR0]	

A

Table A-1. PPC403GC Instruction Syntax Summary (cont.)

Mnemonic	Operands	Function	Other Registers Changed	Page
rotrwi	RA, RS, n	Rotate right immediate. (RA) ← ROTL((RS), 32–n) *Extended mnemonic* for **rlwinm RA,RS,32–n,0,31**		11-130
rotrwi.		*Extended mnemonic* for **rlwinm. RA,RS,32–n,0,31**	CR[CR0]	
sc		System call exception is generated. (SRR1) ← (MSR) (SRR0) ← (PC) PC ← $EVPR_{0:15}$ || x'0C00' (MSR[WE, PR, EE, PE, DR, IR]) ← 0 (MSR[LE]) ← (MSR[ILE])		11-134
slw	RA, RS, RB	Shift left (RS) by $(RB)_{27:31}$. 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.		11-135
slw.			CR[CR0]	
slwi	RA, RS, n	Shift left immediate. (n < 32) $(RA)_{0:31-n}$ ← $(RS)_{n:31}$ $(RA)_{32-n:31}$ ← $^{n}0$ *Extended mnemonic* for **rlwinm RA,RS,n,0,31–n**		11-130
slwi.		*Extended mnemonic* for **rlwinm. RA,RS,n,0,31–n**	CR[CR0]	
sraw	RA, RS, RB	Shift right algebraic (RS) by $(RB)_{27:31}$. 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).		11-136
sraw.			CR[CR0]	
srawi	RA, RS, SH	Shift right algebraic (RS) by SH. 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).		11-137
srawi.			CR[CR0]	

A

Table A-1. PPC403GC Instruction Syntax Summary (cont.)

Mnemonic	Operands	Function	Other Registers Changed	Page
srw	RA, RS, RB	Shift right (RS) by $(RB)_{27:31}$. $n \leftarrow (RB)_{27:31}$. $r \leftarrow ROTL((RS), 32 - n)$. if $(RB)_{26} = 0$ then $m \leftarrow MASK(n, 31)$ else $m \leftarrow {}^{32}0$. $(RA) \leftarrow r \wedge m$.		11-138
srw.			CR[CR0]	
srwi	RA, RS, n	Shift right immediate. ($n < 32$) $(RA)_{n:31} \leftarrow (RS)_{0:31-n}$ $(RA)_{0:n-1} \leftarrow {}^{n}0$ *Extended mnemonic* for **rlwinm RA,RS,32–n,n,31**		11-130
srwi.		*Extended mnemonic* for **rlwinm. RA,RS,32–n,n,31**	CR[CR0]	
stb	RS, D(RA)	Store byte $(RS)_{24:31}$ in memory at EA = (RA|0) + EXTS(D).		11-139
stbu	RS, D(RA)	Store byte $(RS)_{24:31}$ in memory at EA = (RA|0) + EXTS(D). Update the base address, (RA) ← EA.		11-140
stbux	RS, RA, RB	Store byte $(RS)_{24:31}$ in memory at EA = (RA|0) + (RB). Update the base address, (RA) ← EA.		11-141
stbx	RS, RA, RB	Store byte $(RS)_{24:31}$ in memory at EA = (RA|0) + (RB).		11-142
sth	RS, D(RA)	Store halfword $(RS)_{16:31}$ in memory at EA = (RA|0) + EXTS(D).		11-143
sthbrx	RS, RA, RB	Store halfword $(RS)_{16:31}$ byte-reversed in memory at EA = (RA|0) + (RB). $MS(EA, 2) \leftarrow (RS)_{24:31} \parallel (RS)_{16:23}$		11-144
sthu	RS, D(RA)	Store halfword $(RS)_{16:31}$ in memory at EA = (RA|0) + EXTS(D). Update the base address, (RA) ← EA.		11-145
sthux	RS, RA, RB	Store halfword $(RS)_{16:31}$ in memory at EA = (RA|0) + (RB). Update the base address, (RA) ← EA.		11-146

A

Table A-1. PPC403GC Instruction Syntax Summary (cont.)

Mnemonic	Operands	Function	Other Registers Changed	Page
sthx	RS, RA, RB	Store halfword $(RS)_{16:31}$ in memory at EA = (RA|0) + (RB).		11-147
stmw	RS, D(RA)	Store consecutive words from RS through GPR(31) in memory starting at EA = (RA|0) + EXTS(D).		11-148
stswi	RS, RA, NB	Store consecutive bytes in memory starting at EA=(RA|0). Number of bytes n=32 if NB=0, else n=NB. Bytes are unstacked from CEIL(n/4) consecutive registers starting with RS. GPR(0) is consecutive to GPR(31).		11-149
stswx	RS, RA, RB	Store consecutive bytes in memory starting at EA=(RA|0)+(RB). Number of bytes n=XER[TBC]. Bytes are unstacked from CEIL(n/4) consecutive registers starting with RS. GPR(0) is consecutive to GPR(31).		11-150
stw	RS, D(RA)	Store word (RS) in memory at EA = (RA|0) + EXTS(D).		11-152
stwbrx	RS, RA, RB	Store word (RS) byte-reversed in memory at EA = (RA|0) + (RB). $MS(EA, 4) \leftarrow (RS)_{24:31} \| \| (RS)_{16:23} \| \| (RS)_{8:15} \| \| (RS)_{0:7}$		11-153

stwcx.	RS, RA, RB	Store word (RS) in memory at EA = (RA|0) + (RB) only if reservation bit is set.
		if RESERVE = 1 then
		MS(EA, 4) ← (RS)
		RESERVE ← 0
		$(CR[CR0]) \leftarrow {}^{2}0 \| \| 1 \| \| XER_{so}$
		else
		$(CR[CR0]) \leftarrow {}^{2}0 \| \| 0 \| \| XER_{so}$.
				11-154

stwu	RS, D(RA)	Store word (RS) in memory at EA = (RA|0) + EXTS(D). Update the base address, (RA) ← EA.		11-156
stwux	RS, RA, RB	Store word (RS) in memory at EA = (RA|0) + (RB). Update the base address, (RA) ← EA.		11-157

A

Table A-1. PPC403GC Instruction Syntax Summary (cont.)

Mnemonic	Operands	Function	Other Registers Changed	Page
stwx	RS, RA, RB	Store word (RS) in memory at EA = (RA|0) + (RB).		11-158
sub	RT, RA, RB	Subtract (RB) from (RA). (RT) ← ¬(RB) + (RA) + 1. *Extended mnemonic* for **subf RT,RB,RA**		11-159
sub.		*Extended mnemonic* for **subf. RT,RB,RA**	CR[CR0]	
subo		*Extended mnemonic* for **subfo RT,RB,RA**	XER[SO, OV]	
subo.		*Extended mnemonic* for **subfo. RT,RB,RA**	CR[CR0] XER[SO, OV]	
subc	RT, RA, RB	Subtract (RB) from (RA). (RT) ← ¬(RB) + (RA) + 1. Place carry-out in XER[CA]. *Extended mnemonic* for **subfc RT,RB,RA**		11-160
subc.		*Extended mnemonic* for **subfc. RT,RB,RA**	CR[CR0]	
subco		*Extended mnemonic* for **subfco RT,RB,RA**	XER[SO, OV]	
subco.		*Extended mnemonic* for **subfco. RT,RB,RA**	CR[CR0] XER[SO, OV]	
subf	RT, RA, RB	Subtract (RA) from (RB). (RT) ← ¬(RA) + (RB) + 1.		11-159
subf.			CR[CR0]	
subfo			XER[SO, OV]	
subfo.			CR[CR0] XER[SO, OV]	
subfc	RT, RA, RB	Subtract (RA) from (RB). (RT) ← ¬(RA) + (RB) + 1. Place carry-out in XER[CA].		11-160
subfc.			CR[CR0]	
subfco			XER[SO, OV]	
subfco.			CR[CR0] XER[SO, OV]	

A

Table A-1. PPC403GC Instruction Syntax Summary (cont.)

Mnemonic	Operands	Function	Other Registers Changed	Page
subfe	RT, RA, RB	Subtract (RA) from (RB) with carry-in. (RT) ← ¬(RA) + (RB) + XER[CA]. Place carry-out in XER[CA].		11-162
subfe.			CR[CR0]	
subfeo			XER[SO, OV]	
subfeo.			CR[CR0] XER[SO, OV]	
subfic	RT, RA, IM	Subtract (RA) from EXTS(IM). (RT) ← ¬(RA) + EXTS(IM) + 1. Place carry-out in XER[CA].		11-163
subfme	RT, RA, RB	Subtract (RA) from (–1) with carry-in. (RT) ← ¬(RA) + (–1) + XER[CA]. Place carry-out in XER[CA].		11-164
subfme.			CR[CR0]	
subfmeo			XER[SO, OV]	
subfmeo.			CR[CR0] XER[SO, OV]	
subfze	RT, RA, RB	Subtract (RA) from zero with carry-in. (RT) ← ¬(RA) + XER[CA]. Place carry-out in XER[CA].		11-165
subfze.			CR[CR0]	
subfzeo			XER[SO, OV]	
subfzeo.			CR[CR0] XER[SO, OV]	
subi	RT, RA, IM	Subtract EXTS(IM) from (RA|0). Place result in RT. *Extended mnemonic* for **addi RT,RA,–IM**		11-9
subic	RT, RA, IM	Subtract EXTS(IM) from (RA). Place result in RT. Place carry-out in XER[CA]. *Extended mnemonic* for **addic RT,RA,–IM**		11-10
subic.	RT, RA, IM	Subtract EXTS(IM) from (RA). Place result in RT. Place carry-out in XER[CA]. *Extended mnemonic* for **addic. RT,RA,–IM**	CR[CR0]	11-11
subis	RT, RA, IM	Subtract (IM || $^{16}0$) from (RA|0). Place result in RT. *Extended mnemonic* for **addis RT,RA,–IM**		11-12

A

Table A-1. PPC403GC Instruction Syntax Summary (cont.)

Mnemonic	Operands	Function	Other Registers Changed	Page
sync		Synchronization. All instructions that precede **sync** complete before any instructions that follow **sync** begin. When **sync** completes, all storage accesses initiated prior to **sync** will have completed.		11-166
tlbia		All of the entries in the TLB are invalidated and become unavailable for translation by clearing the valid (V) bit in the TLBHI portion of each TLB entry. The rest of the fields in the TLB entries are unmodified.		11-167
tlbre	RT, RA,WS	If WS = 0: Load TLBHI portion of the selected TLB entry into RT. Load the PID register with the contents of the TID field of the selected TLB entry. (RT) ← TLBHI[(RA)] (PID) ← $TLB[(RA)]_{TID}$ If WS = 1: Load TLBLO portion of the selected TLB entry into RT. (RT) ← TLBLO[(RA)]		11-168
tlbrehi	RT, RA	Load TLBHI portion of the selected TLB entry into RT. Load the PID register with the contents of the TID field of the selected TLB entry. (RT) ← TLBHI[(RA)] (PID) ← $TLB[(RA)]_{TID}$ *Extended mnemonic* for **tlbre RT,RA,0**		11-168
tlbrelo	RT, RA	Load TLBLO portion of the selected TLB entry into RT. (RT) ← TLBLO[(RA)] *Extended mnemonic* for **tlbre RT,RA,1**		11-168

A

Table A-1. PPC403GC Instruction Syntax Summary (cont.)

Mnemonic	Operands	Function	Other Registers Changed	Page
tlbsx	RT,RA,RB	Search the TLB array for a valid entry which translates the effective address EA = (RA|0) + (RB). If found, (RT) ← Index of TLB entry. If not found, (RT) Undefined.		11-170
tlbsx.		If found, (RT) ← Index of TLB entry. $CR[CR0]_{EQ}$ ← 1. If not found, (RT) Undefined. $CR[CR0]_{EQ}$ ← 1.	$CR[CR0]_{LT,GT,SO}$	
tlbsync		**tlbsync** does not complete until all previous TLB-update instructions executed by this processor have been received and completed by all other processors. For PPC403GC, **tlbsync** is a no-op.		11-171
tlbwe	RS, RA,WS	If WS = 0: Write TLBHI portion of the selected TLB entry from RS. Write the TID field of the selected TLB entry from the PID register. $TLBHI[(RA)] \leftarrow (RS)$ $TLB[(RA)]_{TID} \leftarrow (PID)_{24:31}$ If WS = 1: Write TLBLO portion of the selected TLB entry from RS. $TLBLO[(RA)] \leftarrow (RS)$		11-172
tlbwehi	RS, RA	Write TLBHI portion of the selected TLB entry from RS. Write the TID field of the selected TLB entry from the PID register. $TLBHI[(RA)] \leftarrow (RS)$ $TLB[(RA)]_{TID} \leftarrow (PID)_{24:31}$ *Extended mnemonic* for **tlbwe RS,RA,0**		11-172
tlbwelo	RS, RA	Write TLBLO portion of the selected TLB entry from RS. $TLBLO[(RA)] \leftarrow (RS)$ *Extended mnemonic* for **tlbwe RS,RA,1**		11-172

A

Table A-1. PPC403GC Instruction Syntax Summary (cont.)

Mnemonic	Operands	Function	Other Registers Changed	Page
trap		Trap unconditionally. *Extended mnemonic* for **tw 31,0,0**		11-174
tweq	RA, RB	Trap if (RA) equal to (RB). *Extended mnemonic* for **tw 4,RA,RB**		
twge		Trap if (RA) greater than or equal to (RB). *Extended mnemonic* for **tw 12,RA,RB**		
twgt		Trap if (RA) greater than (RB). *Extended mnemonic* for **tw 8,RA,RB**		
twle		Trap if (RA) less than or equal to (RB). *Extended mnemonic* for **tw 20,RA,RB**		
twlge		Trap if (RA) logically greater than or equal to (RB). *Extended mnemonic* for **tw 5,RA,RB**		
twlgt		Trap if (RA) logically greater than (RB). *Extended mnemonic* for **tw 1,RA,RB**		
twlle		Trap if (RA) logically less than or equal to (RB). *Extended mnemonic* for **tw 6,RA,RB**		
twllt		Trap if (RA) logically less than (RB). *Extended mnemonic* for **tw 2,RA,RB**		
twlng		Trap if (RA) logically not greater than (RB). *Extended mnemonic* for **tw 6,RA,RB**		
twlnl		Trap if (RA) logically not less than (RB). *Extended mnemonic* for **tw 5,RA,RB**		
twlt		Trap if (RA) less than (RB). *Extended mnemonic* for **tw 16,RA,RB**		
twne		Trap if (RA) not equal to (RB). *Extended mnemonic* for **tw 24,RA,RB**		
twng		Trap if (RA) not greater than (RB). *Extended mnemonic* for **tw 20,RA,RB**		
twnl		Trap if (RA) not less than (RB). *Extended mnemonic* for **tw 12,RA,RB**		
tw	TO, RA, RB	Trap exception is generated if, comparing (RA) with (RB), any condition specified by TO is true.		11-174

A

Table A-1. PPC403GC Instruction Syntax Summary (cont.)

Mnemonic	Operands	Function	Other Registers Changed	Page
tweqi	RA, IM	Trap if (RA) equal to EXTS(IM). *Extended mnemonic* for **twi 4,RA,IM**		11-177
twgei		Trap if (RA) greater than or equal to EXTS(IM). *Extended mnemonic* for **twi 12,RA,IM**		
twgti		Trap if (RA) greater than EXTS(IM). *Extended mnemonic* for **twi 8,RA,IM**		
twlei		Trap if (RA) less than or equal to EXTS(IM). *Extended mnemonic* for **twi 20,RA,IM**		
twlgei		Trap if (RA) logically greater than or equal to EXTS(IM). *Extended mnemonic* for **twi 5,RA,IM**		
twlgti		Trap if (RA) logically greater than EXTS(IM). *Extended mnemonic* for **twi 1,RA,IM**		
twllei		Trap if (RA) logically less than or equal to EXTS(IM). *Extended mnemonic* for **twi 6,RA,IM**		
twllti		Trap if (RA) logically less than EXTS(IM). *Extended mnemonic* for **twi 2,RA,IM**		
twlngi		Trap if (RA) logically not greater than EXTS(IM). *Extended mnemonic* for **twi 6,RA,IM**		
twlnli		Trap if (RA) logically not less than EXTS(IM). *Extended mnemonic* for **twi 5,RA,IM**		
twlti		Trap if (RA) less than EXTS(IM). *Extended mnemonic* for **twi 16,RA,IM**		
twnei		Trap if (RA) not equal to EXTS(IM). *Extended mnemonic* for **twi 24,RA,IM**		
twngi		Trap if (RA) not greater than EXTS(IM). *Extended mnemonic* for **twi 20,RA,IM**		
twnli		Trap if (RA) not less than EXTS(IM). *Extended mnemonic* for **twi 12,RA,IM**		
twi	TO, RA, IM	Trap exception is generated if, comparing (RA) with EXTS(IM), any condition specified by TO is true.		11-177

Table A-1. PPC403GC Instruction Syntax Summary (cont.)

Mnemonic	Operands	Function	Other Registers Changed	Page
wrtee	RS	Write value of RS_{16} to the External Enable bit (MSR[EE]).		11-180
wrteei	E	Write value of E to the External Enable bit (MSR[EE]).		11-181
xor	RA, RS, RB	XOR (RS) with (RB). Place result in RA.		11-182
xor.			CR[CR0]	
xori	RA, RS, IM	XOR (RS) with ($^{16}0$ || IM). Place result in RA.		11-183
xoris	RA, RS, IM	XOR (RS) with (IM || $^{16}0$). Place result in RA.		11-184

A.2 Instructions Sorted by Opcode

All instructions are four bytes long and word aligned. All instructions have a primary opcode field (shown as field OPCD in Figure A-1 through Figure A-9 beginning on page A-52) in bits 0:5. Some instructions also have a seconday opcode field (shown as field XO in Figure A-1 through Figure A-9). PPC403GC instructions sorted by primary and secondary opcode may be found in Table A-2 below.

The "Form" indicated in the table refers to the arrangement of valid field combinations within the four-byte instruction. See Section A.3 (Instruction Formats) on page A-50 for illustration of the field layouts associated with each form.

Form X has a 10-bit secondary opcode field, while form XO uses only the low-order 9-bits of that field. Form XO uses the high-order secondary opcode bit (the tenth bit) as a variable; therefore, every form XO instruction really consumes two secondary opcodes from the 10-bit secondary-opcode space. The implicitly consumed secondary opcode is listed in parentheses for form XO instructions in the table below.

Table A-2. PPC403GC Instructions by Opcode

Primary Opcode	Secondary Opcode	Form	Mnemonic	Operands	Page
3		D	**twi**	TO, RA, IM	11-177
7		D	**mulli**	RT, RA, IM	11-118
8		D	**subfic**	RT, RA, IM	11-163
10		D	**cmpli**	BF, 0, RA, IM	11-39
11		D	**cmpi**	BF, 0, RA, IM	11-37
12		D	**addic**	RT, RA, IM	11-10
13		D	**addic.**	RT, RA, IM	11-11
14		D	**addi**	RT, RA, IM	11-9
15		D	**addis**	RT, RA, IM	11-12
16		B	**bc**	BO, BI, target	11-20
			bca		
			bcl		
			bcla		
17		SC	**sc**		11-134

A

Table A-2. PPC403GC Instructions by Opcode (cont.)

Primary Opcode	Secondary Opcode	Form	Mnemonic	Operands	Page
18		I	**b**	target	11-19
			ba		
			bl		
			bla		
19	0	XL	**mcrf**	BF, BFA	11-101
19	16	XL	**bclr**	BO, BI	11-31
			bclrl		
19	33	XL	**crnor**	BT, BA, BB	11-45
19	50	XL	**rfi**		11-128
19	51	XL	**rfci**		11-127
19	129	XL	**crandc**	BT, BA, BB	11-42
19	150	XL	**isync**		11-76
19	193	XL	**crxor**	BT, BA, BB	11-48
19	225	XL	**crnand**	BT, BA, BB	11-44
19	257	XL	**crand**	BT, BA, BB	11-41
19	289	XL	**creqv**	BT, BA, BB	11-43
19	417	XL	**crorc**	BT, BA, BB	11-47
19	449	XL	**cror**	BT, BA, BB	11-46
19	528	XL	**bcctr**	BO, BI	11-27
			bcctrl		
20		M	**rlwimi**	RA, RS, SH, MB, ME	11-129
			rlwimi.		
21		M	**rlwinm**	RA, RS, SH, MB, ME	11-130
			rlwinm.		
23		M	**rlwnm**	RA, RS, RB, MB, ME	11-133
			rlwnm.		
24		D	**ori**	RA, RS, IM	11-125
25		D	**oris**	RA, RS, IM	11-126
26		D	**xori**	RA, RS, IM	11-183

Table A-2. PPC403GC Instructions by Opcode (cont.)

Primary Opcode	Secondary Opcode	Form	Mnemonic	Operands	Page
27		D	**xoris**	RA, RS, IM	11-184
28		D	**andi.**	RA, RS, IM	11-17
29		D	**andis.**	RA, RS, IM	11-18
31	0	X	**cmp**	BF, 0, RA, RB	11-36
31	4	X	**tw**	TO, RA, RB	11-174
31	8 (520)	XO	**subfc**	RT, RA, RB	11-160
			subfc.		
			subfco		
			subfco.		
31	10 (522)	XO	**addc**	RT, RA, RB	11-7
			addc.		
			addco		
			addco.		
31	11 (523)	XO	**mulhwu**	RT, RA, RB	11-117
			mulhwu.		
31	19	X	**mfcr**	RT	11-103
31	20	X	**lwarx**	RT, RA, RB	11-95
31	23	X	**lwzx**	RT, RA, RB	11-100
31	24	X	**slw**	RA, RS, RB	11-135
			slw.		
31	26	X	**cntlzw**	RA, RS	11-40
			cntlzw.		
31	28	X	**and**	RA, RS, RB	11-15
			and.		
31	32	X	**cmpl**	BF, 0, RA, RB	11-38
31	40 (552)	XO	**subf**	RT, RA, RB	11-159
			subf.		
			subfo		
			subfo.		

Table A-2. PPC403GC Instructions by Opcode (cont.)

Primary Opcode	Secondary Opcode	Form	Mnemonic	Operands	Page
31	54	X	**dcbst**	RA, RB	11-51
31	55	X	**lwzux**	RT, RA, RB	11-99
31	60	X	**andc**	RA, RS, RB	11-16
			andc.		
31	75 (587)	XO	**mulhw**	RT, RA, RB	11-116
			mulhw.		
31	83	X	**mfmsr**	RT	11-106
31	86	X	**dcbf**	RA, RB	11-49
31	87	X	**lbzx**	RT, RA, RB	11-80
31	104 (616)	XO	**neg**	RT, RA	11-121
			neg.		
			nego		
			nego.		
31	119	X	**lbzux**	RT, RA, RB	11-79
31	124	X	**nor**	RA, RS, RB	11-122
			nor.		
31	131	X	**wrtee**	RS	11-180
31	136 (648)	XO	**subfe**	RT, RA, RB	11-162
			subfe.		
			subfeo		
			subfeo.		
31	138 (650)	XO	**adde**	RT, RA, RB	11-8
			adde.		
			addeo		
			addeo.		
31	144	XFX	**mtcrf**	FXM, RS	11-109
31	146	X	**mtmsr**	RS	11-113
31	150	X	**stwcx.**	RS, RA, RB	11-154
31	151	X	**stwx**	RS, RA, RB	11-158

Table A-2. PPC403GC Instructions by Opcode (cont.)

Primary Opcode	Secondary Opcode	Form	Mnemonic	Operands	Page
31	163	X	**wrteei**	E	11-181
31	183	X	**stwux**	RS, RA, RB	11-157
31	200 (712)	XO	**subfze**	RT, RA, RB	11-165
			subfze.		
			subfzeo		
			subfzeo.		
31	202 (714)	XO	**addze**	RT, RA	11-14
			addze.		
			addzeo		
			addzeo.		
31	215	X	**stbx**	RS, RA, RB	11-142
31	232 (744)	XO	**subfme**	RT, RA, RB	11-164
			subfme.		
			subfmeo		
			subfmeo.		
31	234 (746)	XO	**addme**	RT, RA	11-13
			addme.		
			addmeo		
			addmeo.		
31	235 (747)	XO	**mullw**	RT, RA, RB	11-119
			mullw.		
			mullwo		
			mullwo.		
31	246	X	**dcbtst**	RA,RB	11-54
31	247	X	**stbux**	RS, RA, RB	11-141
31	262	X	**icbt**	RA, RB	11-70

A

Table A-2. PPC403GC Instructions by Opcode (cont.)

Primary Opcode	Secondary Opcode	Form	Mnemonic	Operands	Page
31	266 (778)	XO	**add**	RT, RA, RB	11-6
			add.		
			addo		
			addo.		
31	278	X	**dcbt**	RA, RB	11-52
31	279	X	**lhzx**	RT, RA, RB	11-89
31	284	X	**eqv**	RA, RS, RB	11-65
			eqv.		
31	311	X	**lhzux**	RT, RA, RB	11-88
31	316	X	**xor**	RA, RS, RB	11-182
			xor.		
31	323	XFX	**mfdcr**	RT, DCRN	11-104
31	339	XFX	**mfspr**	RT, SPRN	11-107
31	343	X	**lhax**	RT, RA, RB	11-84
31	370	X	**tlbia**		11-167
31	375	X	**lhaux**	RT, RA, RB	11-83
31	407	X	**sthx**	RS, RA, RB	11-147
31	412	X	**orc**	RA, RS, RB	11-124
			orc.		
31	439	X	**sthux**	RS, RA, RB	11-146
31	444	X	**or**	RA, RS, RB	11-123
			or.		
31	451	XFX	**mtdcr**	DCRN, RS	11-111
31	454	X	**dccci**	RA, RB	11-58
31	459 (971)	XO	**divwu**	RT, RA, RB	11-63
			divwu.		
			divwuo		
			divwuo.		
31	467	XFX	**mtspr**	SPRN, RS	11-114

Table A-2. PPC403GC Instructions by Opcode (cont.)

Primary Opcode	Secondary Opcode	Form	Mnemonic	Operands	Page
31	470	X	**dcbi**	RA, RB	11-50
31	476	X	**nand**	RA, RS, RB	11-120
			nand.		
31	486	X	**dcread**	RT, RA, RB	11-60
31	491 (1003)	XO	**divw**	RT, RA, RB	11-62
			divw.		
			divwo		
			divwo.		
31	512	X	**mcrxr**	BF	11-102
31	533	X	**lswx**	RT, RA, RB	11-93
31	534	X	**lwbrx**	RT, RA, RB	11-96
31	536	X	**srw**	RA, RS, RB	11-138
			srw.		
31	566	X	**tlbsync**		11-171
31	597	X	**lswi**	RT, RA, NB	11-91
31	598	X	**sync**		11-166
31	661	X	**stswx**	RS, RA, RB	11-150
31	662	X	**stwbrx**	RS, RA, RB	11-153
31	725	X	**stswi**	RS, RA, NB	11-149
31	790	X	**lhbrx**	RT, RA, RB	11-85
31	792	X	**sraw**	RA, RS, RB	11-136
			sraw.		
31	824	X	**srawi**	RA, RS, SH	11-137
			srawi.		
31	854	X	**eieio**		11-64
31	914	X	**tlbsx**	RT,RA,RB	11-170
			tlbsx.		
31	918	X	**sthbrx**	RS, RA, RB	11-144

Table A-2. PPC403GC Instructions by Opcode (cont.)

Primary Opcode	Secondary Opcode	Form	Mnemonic	Operands	Page
31	922	X	**extsh**	RA, RS	11-67
			extsh.		
31	946	X	**tlbre**	RT, RA,WS	11-168
31	954	X	**extsb**	RA, RS	11-66
			extsb.		
31	966	X	**iccci**	RA, RB	11-72
31	978	X	**tlbwe**	RS, RA,WS	11-172
31	982	X	**icbi**	RA, RB	11-68
31	998	X	**icread**	RA, RB	11-74
31	1014	X	**dcbz**	RA, RB	11-56
32		D	**lwz**	RT, D(RA)	11-97
33		D	**lwzu**	RT, D(RA)	11-98
34		D	**lbz**	RT, D(RA)	11-77
35		D	**lbzu**	RT, D(RA)	11-78
36		D	**stw**	RS, D(RA)	11-152
37		D	**stwu**	RS, D(RA)	11-156
38		D	**stb**	RS, D(RA)	11-139
39		D	**stbu**	RS, D(RA)	11-140
40		D	**lhz**	RT, D(RA)	11-86
41		D	**lhzu**	RT, D(RA)	11-87
42		D	**lha**	RT, D(RA)	11-81
43		D	**lhau**	RT, D(RA)	11-82
44		D	**sth**	RS, D(RA)	11-143
45		D	**sthu**	RS, D(RA)	11-145
46		D	**lmw**	RT, D(RA)	11-90
47		D	**stmw**	RS, D(RA)	11-148

A.3 Instruction Formats

Instructions are four bytes long. Instruction addresses are always word-aligned.

Instruction bits 0 through 5 always contain the primary opcode. Many instructions have an extended opcode in another field. The remaining instruction bits contain additional fields. All instruction fields belong to one of the following categories:

- Defined

  These instructions contain values, such as opcodes, that cannot be altered. The instruction format diagrams specify the values of defined fields.

- Variable

  These fields contain operands, such as general purpose register selectors and immediate values, that may vary from execution to execution. The instruction format diagrams specify the operands in variable fields.

- Reserved

  Bits in a reserved field should be set to 0. In the instruction format diagrams, reserved fields are shaded.

If any bit in a defined field does not contain the expected value, the instruction is illegal and an illegal instruction exception occurs. If any bit in a reserved field does not contain 0, the instruction form is invalid and its result is architecturally undefined. The PPC403GC executes all invalid instruction forms without causing an illegal instruction exception.

A.3.1 Instruction Fields

PPC403GC instructions contain various combinations of the following fields, as indicated in the instruction format diagrams. The numbers, enclosed in parentheses, that follow the field names indicate the bit positions; bit fields are indicated by starting and stopping bit positions separated by colons.

AA (30) Absolute address bit.

- 0 The immediate field represents an address relative to the current instruction address (CIA). The effective address (EA) of the branch is either the sum of the LI field sign-extended to 32 bits and the branch instruction address, or the sum of the BD field sign-extended to 32 bits and the branch instruction address.
- 1 The immediate field represents an absolute address. The EA of the branch is either the LI field or the BD field, sign-extended to 32 bits.

BA (11:15) Specifies a bit in the condition register (CR) used as a source of a CR-Logical instruction.

BB (16:20) Specifies a bit in the CR used as a source of a CR-Logical instruction.

BD (16:29) An immediate field specifying a 14-bit signed twos complement branch displacement. This field is concatenated on the right with 0b00 and sign-extended to 32 bits.

BF (6:8) Specifies a field in the CR used as a target in a compare or **mcrf** instruction.

BFA (11:13) Specifies a field in the CR used as a source in a **mcrf** instruction.

BI (11:15) Specifies a bit in the CR used as a source for the condition of a conditional branch instruction.

BO (6:10) Specifies options for conditional branch instructions. See Section 2.7.4.

BT (6:10) Specifies a bit in the CR used as a target as the result of a CR-Logical instruction.

D (16:31) Specifies a 16-bit signed two's-complement integer displacement for load/ store instructions.

DCRN (11:20) Specifies a device control register (DCR).

FXM (12:19) Field mask used to identify CR fields to be updated by the **mtcrf** instruction.

IM (16:31) An immediate field used to specify a 16-bit value (either signed integer or unsigned).

LI (6:29) An immediate field specifying a 24-bit signed twos complement branch displacement; this field is concatenated on the right with b'00' and sign-extended to 32 bits.

LK (31) Link bit.

0 Do not update the link register (LR).

1 Update the LR with the address of the next instruction.

MB (21:25) Mask begin.

Used in rotate-and-mask instructions to specify the beginning bit of a mask.

ME (26:30) Mask end.

Used in rotate-and-mask instructions to specify the ending bit of a mask.

NB (16:20) Specifies the number of bytes to move in an immediate string load or store.

OPCD (0:5) Primary opcode. Primary opcodes, in decimal, appear in the instruction format diagrams presented with individual instructions. The OPCD field name does not appear in instruction descriptions.

OE (21) Enables setting the OV and SO fields in the fixed-point exception register (XER) for extended arithmetic.

RA (11:15) A GPR used as a source or target.

RB (16:20) A GPR used as a source.

Rc (31) Record bit.

0 Do not set the CR.

1 Set the CR to reflect the result of an operation.

See Section 2.3.3 on page 2-13 for a further discussion of how the CR bits are set.

RS (6:10) A GPR used as a source.

RT (6:10) A GPR used as a target.

SH (16:20) Specifies a shift amount.

SPRF (11:20) Specifies a special purpose register (SPR).

TO (6:10) Specifies the conditions on which to trap, as described under **tw** and **twi** instructions.

XO (21:30) Extended opcode for instructions without an OE field. Extended opcodes, in decimal, appear in the instruction format diagrams presented with individual instructions. The XO field name does not appear in instruction descriptions.

XO (22:30) Extended opcode for instructions with an OE field. Extended opcodes, in decimal, appear in the instruction format diagrams presented with individual instructions. The XO field name does not appear in instruction descriptions.

A.3.2 Instruction Format Diagrams

The "Forms" shown in Figure A-1 through Figure A-9 are valid combinations of instruction fields for the PPC403GC. Table A-2 on page A-42 indicates which "Form" is utilized by each PPC403GC opcode. Fields indicated by slashes (/, //, or ///) are reserved. These figures have been adapted from the PowerPC User Instruction Set Architecture.

I-Form

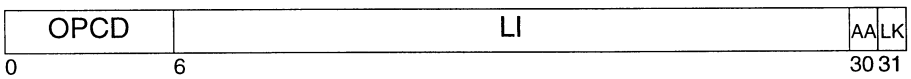


Figure A-1. I Instruction Format

B-Form

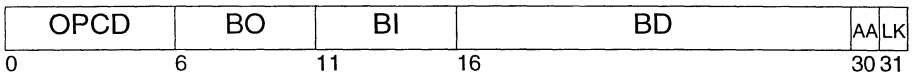


Figure A-2. B Instruction Format

SC-Form

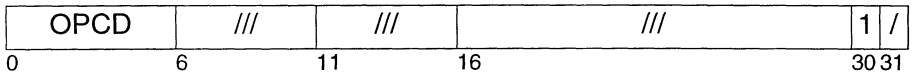


Figure A-3. SC Instruction Format

D-Form

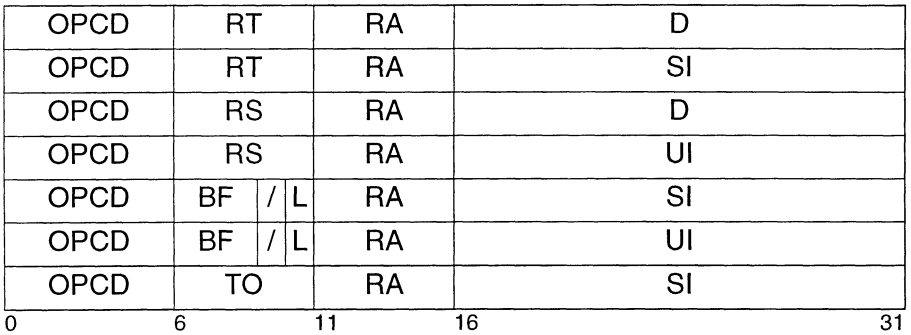


Figure A-4. D Instruction Format

A

X-Form

OPCD	RT	RA	RB	XO	Rc
OPCD	RT	RA	RB	XO	/
OPCD	RT	RA	NB	XO	/
OPCD	RT	RA	WS	XO	/
OPCD	RT	///	RB	XO	/
OPCD	RT	///	///	XO	/
OPCD	RS	RA	RB	XO	Rc
OPCD	RS	RA	RB	XO	1
OPCD	RS	RA	RB	XO	/
OPCD	RS	RA	NB	XO	/
OPCD	RS	RA	WS	XO	/
OPCD	RS	RA	SH	XO	Rc
OPCD	RS	RA	///	XO	Rc
OPCD	RS	///	RB	XO	/
OPCD	RS	///	///	XO	/
OPCD	BF / L	RA	RB	XO	/
OPCD	BF //	BFA //	///	XO	/
OPCD	BF //	///	U /	XO	Rc
OPCD	BF //	///	///	XO	/
OPCD	TO	RA	RB	XO	/
OPCD	BT	///	///	XO	Rc
OPCD	///	RA	RB	XO	/
OPCD	///	///	RB	XO	/
OPCD	///	///	///	XO	/
OPCD	///	///	E //	XO	/
0	6	11	16	21	31

Figure A-5. X Instruction Format

A

XL-Form

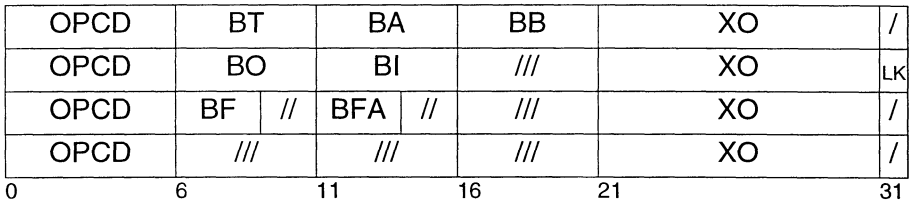


Figure A-6. XL Instruction Format

XFX-Form

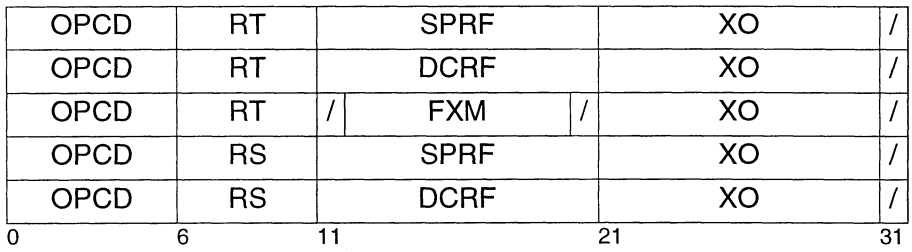


Figure A-7. XFX Instruction Format

XO-Form

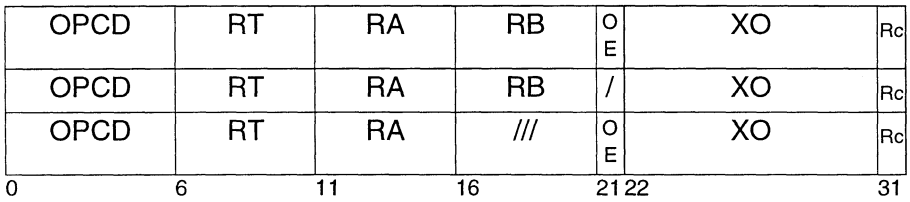


Figure A-8. XO Instruction Format

M-Form

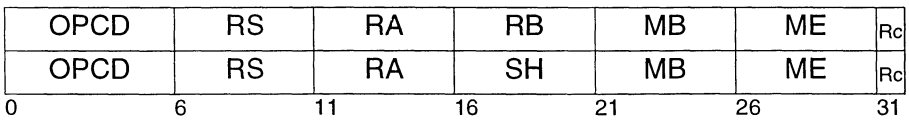


Figure A-9. M Instruction Format

A

A

11

Instruction Set

Descriptions of the PPC403GC instructions follow. Each description contains these elements:

- Instruction names (mnemonic and full)
- Instruction syntax
- Instruction format diagram specific to the individual instruction
- Pseudocode description of the instruction operation
- Prose description of the instruction operation
- Registers altered
- Architecture notes identifying the associated PowerPC Architecture component

Where appropriate, instruction descriptions list invalid instruction forms and provide programming notes.

11.1 Instruction Formats

For a more complete discussion of instruction formats, including a summary of instruction field usage and a compilation of general instruction format diagrams appropriate to the PPC403GC, see Section A.3 on page A-50.

Instructions are four bytes long. Instruction addresses are always word-aligned.

Instruction bits 0 through 5 always contain the primary opcode. Many instructions have an extended opcode in another field. The remaining instruction bits contain additional fields. All instruction fields belong to one of the following categories:

- Defined

  These instructions contain values, such as opcodes, that cannot be altered. The instruction format diagrams specify the values of defined fields.

- Variable

  These fields contain operands, such as general purpose register selectors and immediate values, that may vary from execution to execution. The instruction format diagrams specify the operands in variable fields.

- Reserved

  Bits in a reserved field should be set to 0. In the instruction format diagrams, reserved fields are shaded.

If any bit in a defined field does not contain the expected value, the instruction is illegal and an illegal instruction exception occurs. If any bit in a reserved field does not contain 0, the instruction form is invalid and its result is architecturally undefined. The PPC403GC executes all invalid instruction forms without causing an illegal instruction exception.

11.2 Pseudocode

The pseudocode that appears in the instruction descriptions provides a semi-formal language for describing instruction operations.

The pseudocode uses the following notation:

$\leftarrow$	Assignment
$\wedge$	AND logical operator
$\neg$	NOT logical operator
$\vee$	OR logical operator
$\oplus$	Exclusive-OR (XOR) logical operator
+	Twos complement addition
–	Twos complement subtraction, unary minus
$\times$	Multiplication
$\div$	Division yielding a quotient
%	Remainder of an integer division; (33 % 32) = 1.
||	Concatenation
$=, \neq$	Equal, not equal relations
<, >	Signed comparison relations
$\overset{u}{<}, \overset{u}{>}$	Unsigned comparison relations
if...then...else...	Conditional execution; if *condition* then *a* else *b*, where *a* and *b* represent one or more pseudocode statements. Indenting indicates the ranges of *a* and *b*. If *b* is null, the else does not appear.
do	Do loop. "to" and "by" clauses specify incrementing an iteration variable; "while" and "until" clauses specify terminating conditions. Indenting indicates the range of the loop.

leave	Leave innermost do loop or do loop specified in a leave statement.
n	A decimal number
x'n'	A hexadecimal number
b'n'	A binary number
FLD	An instruction field
FLD_b	A bit in an instruction field
$FLD_{b:b}$	A range of bits in an instruction field
$FLD_{b,b,\ldots}$	A list of bits, by number or name, in a named field
REG_b	A bit in a named register
$REG_{b:b}$	A range of bits in a named register
$REG_{b,b,\ldots}$	A list of bits, by number or name, in a named register
REG[FLD]	A field in a named register
REG[FLD, FLD . . .]	A list of fields in a named register
GPR(r)	General Purpose Register r, where $0 \leq r \leq 31$.
(GPR(r))	The contents of General Purpose Register r, where $0 \leq r \leq 31$.
DCR(DCRN)	A DCR specified by the DCRF field in a **mfdcr** or **mtdcr** instruction
SPR(SPRN)	An SPR specified by the SPRF field in a **mfspr** or **mtspr** instruction
RA, RB, . . .	GPRs
(Rx)	The contents of a GPR, where *x* is A, B, S, or T
(RA|0)	The contents of the register RA or 0, if the RA field is 0.
$c_{0:3}$	A four-bit object used to store condition results in compare instructions.
^{n}b	The bit or bit value *b* is replicated *n* times.
xx	Bit positions which are don't-cares.
CEIL(x)	Least integer $\geq x$.
EXTS(x)	The result of extending *x* on the left with sign bits.
PC	Program counter.
RESERVE	Reserve bit; indicates whether a process has reserved a block of storage.

CIA	Current instruction address; the 32-bit address of the instruction being described by a sequence of pseudocode. This address is used to set the next instruction address (NIA). Does not correspond to any architected register.
NIA	Next instruction address; the 32-bit address of the next instruction to be executed. In pseudocode, a successful branch is indicated by assigning a value to NIA. For instructions that do not branch, the NIA is CIA +4.
MS(addr, n)	The number of bytes represented by *n* at the location in main storage represented by *addr*.
EA	Effective address; the 32-bit address, derived by applying indexing or indirect addressing rules to the specified operand, that specifies an location in main storage.
ROTL((RS),n)	Rotate left; the contents of RS are shifted left the number of bits specified by *n*.
MASK(MB,ME)	Mask having 1's in positions MB through ME (wrapping if MB > ME) and 0's elsewhere.
instruction(EA)	An instruction operating on a data or instruction cache block associated with an effective address.

The following table lists the pseudocode operators and their associativity in descending order of precedence:

Table 11-1. Operator Precedence

Operators	Associativity
REG_b, REG[FLD], function evaluation	Left to right
^nb	Right to left
$\neg$, – (unary minus)	Right to left
$\times$, $\div$	Left to right
+, –	Left to right
$\|\|$	Left to right
=, $\neq$, <, >, $\overline{<}^u$, $\overline{>}^u$	Left to right
$\wedge$, $\oplus$	Left to right
$\vee$	Left to right
$\leftarrow$	None

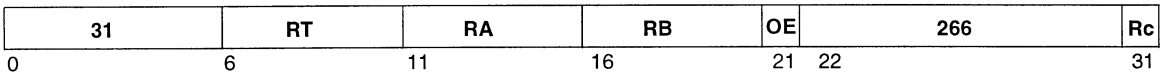
11.3 Register Usage

Each instruction description lists the registers altered by the instruction. Some register changes are explicitly detailed in the instruction description (for example, the target register of a load instruction). Other registers are changed, with the details of the change not included in the instruction description. This category frequently includes the Condition Register (CR) and the Fixed-point Exception Register (XER). For discussion of CR, see Section 2.3.3 on page 2-13. For discussion of XER, see Section 2.3.2.5 on page 2-10.

add

Add

add	RT,RA,RB	(OE=0, Rc=0)
add.	RT,RA,RB	(OE=0, Rc=1)
addo	RT,RA,RB	(OE=1, Rc=0)
addo.	RT,RA,RB	(OE=1, Rc=1)



(RT) ← (RA) + (RB)

The sum of the contents of register RA and the contents of register RB is placed into register RT.

Registers Altered

- RT
- $CR[CR0]_{LT, GT, EQ, SO}$ if Rc contains 1
- XER[SO, OV] if OE contains 1

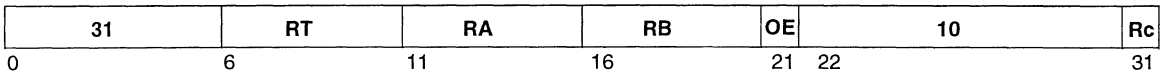
Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

addc

Add Carrying

addc	RT,RA,RB	(OE=0, Rc=0)
addc.	RT,RA,RB	(OE=0, Rc=1)
addco	RT,RA,RB	(OE=1, Rc=0)
addco.	RT,RA,RB	(OE=1, Rc=1)



(RT) ← (RA) + (RB)
if (RA) + (RB) $\overset{u}{>}$ $2^{32} - 1$ then
  XER[CA] ← 1
else
  XER[CA] ← 0

The sum of the contents of register RA and register RB is placed into register RT.

XER[CA] is set to a value determined by the unsigned magnitude of the result of the add operation.

Registers Altered

- RT
- XER[CA]
- $CR[CR0]_{LT, GT, EQ, SO}$ if Rc contains 1
- XER[SO, OV] if OE contains 1

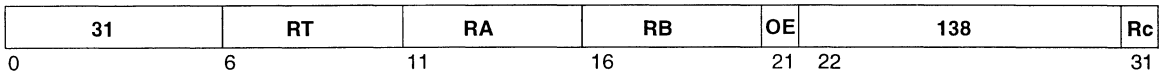
Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

adde

Add Extended

adde	RT,RA,RB	(OE=0, Rc=0)
adde.	RT,RA,RB	(OE=0, Rc=1)
addeo	RT,RA,RB	(OE=1, Rc=0)
addeo.	RT,RA,RB	(OE=1, Rc=1)



(RT) ← (RA) + (RB) + XER[CA]
if (RA) + (RB) + XER[CA] $\overset{u}{>}$ $2^{32} - 1$ then
  XER[CA] ← 1
else
  XER[CA] ← 0

The sum of the contents of register RA, register RB, and XER[CA] is placed into register RT.

XER[CA] is set to a value determined by the unsigned magnitude of the result of the add operation.

Registers Altered

- RT
- XER[CA]
- $CR[CR0]_{LT, GT, EQ, SO}$ if Rc contains 1
- XER[SO, OV] if OE contains 1

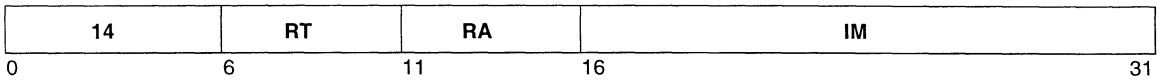
Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

addi

Add Immediate

addi RT,RA,IM



(RT) ← (RAI0) + EXTS(IM)

If the RA field is 0, the IM field, sign-extended to 32 bits, is placed into register RT.

If the RA field is nonzero, the sum of the contents of register RA and the contents of the IM field, sign-extended to 32 bits, is placed into register RT.

Registers Altered

- RT

Programming Note

To place an immediate, sign-extended value into the GPR specified by the RT field, set the RA field to 0.

Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

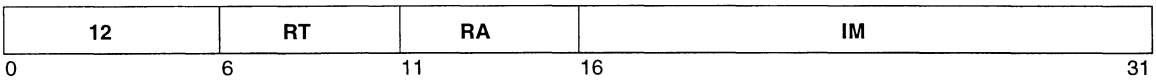
Table 11-2. Extended Mnemonics for addi

Mnemonic	Operands	Function	Other Registers Changed
la	RT, D(RA)	Load address. (RA ≠ 0) D is an offset from a base address that is assumed to be (RA). (RT) ← (RA) + EXTS(D) *Extended mnemonic* for **addi RT,RA,D**	
li	RT, IM	Load immediate. (RT) ← EXTS(IM) *Extended mnemonic* for **addi RT,0,IM**	
subi	RT, RA, IM	Subtract EXTS(IM) from (RAI0). Place result in RT. *Extended mnemonic* for **addi RT,RA,–IM**	

addic

Add Immediate Carrying

addic RT,RA,IM



(RT) ← (RA) + EXTS(IM)
if (RA) + EXTS(IM) $\overset{u}{>} 2^{32} - 1$ then
  XER[CA] ← 1
else
  XER[CA] ← 0

The sum of the contents of register RA and the contents of the IM field, sign-extended to 32 bits, is placed into register RT.

XER[CA] is set to a value determined by the unsigned magnitude of the result of the add operation.

Registers Altered

- RT
- XER[CA]

Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

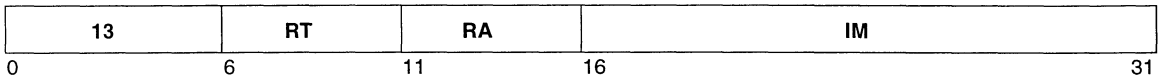
Table 11-3. Extended Mnemonics for addic

Mnemonic	Operands	Function	Other Registers Changed
subic	RT, RA, IM	Subtract EXTS(IM) from (RA). Place result in RT. Place carry-out in XER[CA]. *Extended mnemonic* for **addic RT,RA,–IM**	

addic.

Add Immediate Carrying and Record

addic. RT,RA,IM



(RT) ← (RA) + EXTS(IM)
if (RA) + EXTS(IM) $\overset{u}{\geq} 2^{32} - 1$ then
  XER[CA] ← 1
else
  XER[CA] ← 0

The sum of the contents of register RA and the contents of the IM field, sign-extended to 32 bits, is placed into register RT.

XER[CA] is set to a value determined by the unsigned magnitude of the result of the add operation.

Registers Altered

- RT
- XER[CA]
- $CR[CR0]_{LT, GT, EQ, SO}$

Programming Note

addic. is one of three instructions that implicitly update CR[CR0] without having an RC field. The other instructions are **andi.** and **andis.**.

Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

11

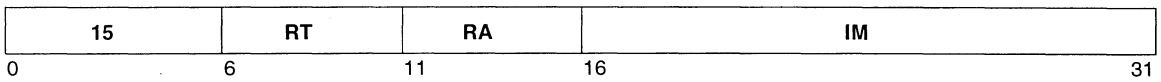
Table 11-4. Extended Mnemonics for addic.

Mnemonic	Operands	Function	Other Registers Changed
subic.	RT, RA, IM	Subtract EXTS(IM) from (RA). Place result in RT. Place carry-out in XER[CA]. *Extended mnemonic* for **addic. RT,RA,–IM**	CR[CR0]

addis

Add Immediate Shifted

addis RT,RA,IM



(RT) ← (RA|0) + (IM || $^{16}0$)

If the RA field is 0, the IM field is concatenated on its right with sixteen 0-bits and placed into register RT.

If the RA field is nonzero, the contents of register RA are added to the contents of the extended IM field. The sum is stored into register RT.

Registers Altered

- RT

Programming Note

An **addi** instruction stores a sign-extended 16-bit value in a GPR. An **addis** instruction followed by an **ori** instruction stores an arbitrary 32-bit value in a GPR, as shown in the following example:

```
addis    RT, 0, high 16 bits of value
ori      RT, RT, low 16 bits of value
```

Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

11

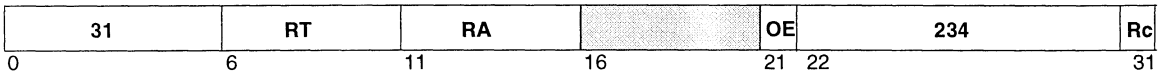
Table 11-5. Extended Mnemonics for addis

Mnemonic	Operands	Function	Other Registers Changed
lis	RT, IM	Load immediate shifted. (RT) ← (IM || $^{16}0$) *Extended mnemonic* for **addis RT,0,IM**	
subis	RT, RA, IM	Subtract (IM || $^{16}0$) from (RA|0). Place result in RT. *Extended mnemonic* for **addis RT,RA,–IM**	

addme

Add to Minus One Extended

addme	RT,RA	(OE=0, Rc=0)
addme.	RT,RA	(OE=0, Rc=1)
addmeo	RT,RA	(OE=1, Rc=0)
addmeo.	RT,RA	(OE=1, Rc=1)



(RT) ← (RA) + XER[CA] + (–1)
if (RA) + XER[CA] + 0xFFFF FFFF $\overset{u}{>}$ $2^{32} - 1$ then
  XER[CA] ← 1
else
  XER[CA] ← 0

The sum of the contents of register RA, XER[CA], and –1 is placed into register RT.

XER[CA] is set to a value determined by the unsigned magnitude of the result of the add operation.

Registers Altered

- RT
- XER[CA]
- $CR[CR0]_{LT, GT, EQ, SO}$ if Rc contains 1
- XER[SO, OV] if OE contains 1

Invalid Instruction Forms

- Reserved fields

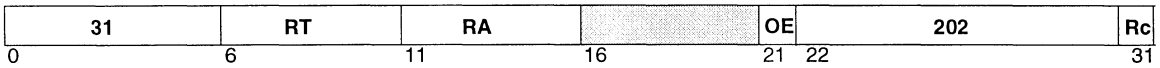
Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

addze

Add to Zero Extended

addze	RT,RA	(OE=0, Rc=0)
addze.	RT,RA	(OE=0, Rc=1)
addzeo	RT,RA	(OE=1, Rc=0)
addzeo.	RT,RA	(OE=1, Rc=1)



(RT) ← (RA) + XER[CA]
if (RA) + XER[CA] $\overset{u}{>}$ $2^{32} - 1$ then
  XER[CA] ← 1
else
  XER[CA] ← 0

The sum of the contents of register RA and XER[CA] is placed into register RT.

XER[CA] is set to a value determined by the unsigned magnitude of the result of the add operation.

Registers Altered

- RT
- XER[CA]
- $CR[CR0]_{LT, GT, EQ, SO}$ if Rc contains 1
- XER[SO, OV] if OE contains 1

Invalid Instruction Forms

- Reserved fields

11

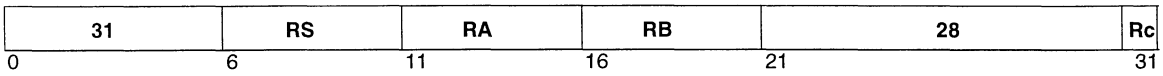
Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

and

AND

and	RA,RS,RB	(Rc=0)
and.	RA,RS,RB	(Rc=1)



(RA) ← (RS) ∧ (RB)

The contents of register RS is ANDed with the contents of register RB and the result is placed into register RA.

Registers Altered

- RA
- $CR[CR0]_{LT, GT, EQ, SO}$ if Rc contains 1

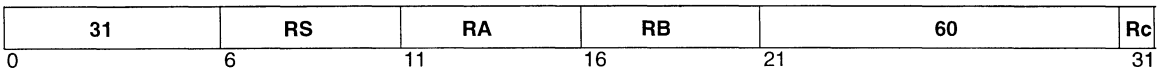
Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

andc

AND with Complement

andc	RA,RS,RB	(Rc=0)
andc.	RA,RS,RB	(Rc=1)



(RA) ← (RS) ∧ ¬(RB)

The contents of register RS is ANDed with the ones complement of the contents of register RB; the result is placed into register RA.

Registers Altered

- RA
- $CR[CR0]_{LT, GT, EQ, SO}$ if Rc contains 1

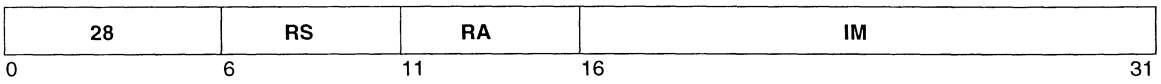
Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

andi.

AND Immediate

andi. RA,RS,IM



(RA) ← (RS) ∧ (^{16}0 || IM)

The IM field is extended to 32 bits by concatenating 16 0-bits on its left. The contents of register RS is ANDed with the extended IM field; the result is placed into register RA.

Registers Altered

- RA
- $CR[CR0]_{LT, GT, EQ, SO}$

Programming Note

The **andi.** instruction can test whether any of the 16 least-significant bits in a GPR are 1-bits.

andi. is one of three instructions that implicitly update CR[CR0] without having an Rc field. The other instructions are **addic.** and **andis.**.

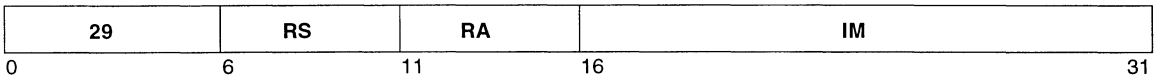
Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

andis.

AND Immediate Shifted

andis. RA,RS,IM



(RA) ← (RS) ∧ (IM || $^{16}0$)

The IM field is extended to 32 bits by concatenating 16 0-bits on its right. The contents of register RS are ANDed with the extended IM field; the result is placed into register RA.

Registers Altered

- RA
- $CR[CR0]_{LT, GT, EQ, SO}$

Programming Note

The **andis.** instruction can test whether any of the 16 most-significant bits in a GPR are 1-bits.

andis. is one of three instructions that implicitly update CR[CR0] without having an Rc field. The other instructions are **addic.** and **andi.**.

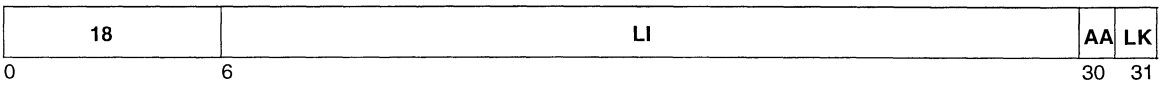
Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

b

Branch

b	target	(AA=0, LK=0)
ba	target	(AA=1, LK=0)
bl	target	(AA=0, LK=1)
bla	target	(AA=1, LK=1)




```
If AA = 1 then
   LI ← target6:29
   NIA ← EXTS(LI || 20)
else
   LI ← (target – CIA)6:29
   NIA ← CIA + EXTS(LI || 20)
if LK = 1 then
   (LR) ← CIA + 4
PC ← NIA
```

The next instruction address (NIA) is the effective address of the branch. The NIA is formed by adding a displacement to a base address. The displacement is obtained by concatenating two 0-bits to the right of the LI field and sign-extending the result to 32 bits.

If the AA field contains 0, the base address is the address of the branch instruction, which is also the current instruction address (CIA). If the AA field contains 1, the base address is 0.

Program flow is transferred to the NIA.

If the LK field contains 1, then (CIA + 4) is placed into the LR.

Registers Altered

- LR if LK contains 1

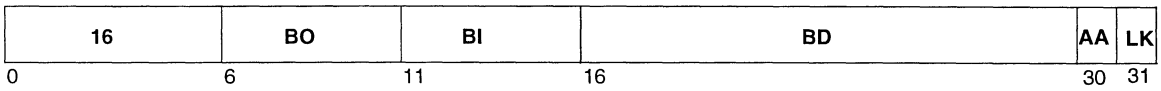
Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

bc

Branch Conditional

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




```
if BO2 = 0 then
   CTR ← CTR – 1
if (BO2 = 1 ∨ ((CTR = 0) = BO3)) ∧ (BO0 = 1 ∨ (CRBI = BO1))  then
   if AA = 1 then
      BD ← target16:29
      NIA ← EXTS(BD || 20)
   else
      BD ← (target – CIA)16:29
      NIA ← CIA + EXTS(BD || 20)
else
   NIA ← CIA + 4
if LK = 1 then
   (LR) ← CIA + 4
PC ← NIA
```

If bit 2 of the BO field contains 0, the CTR is decremented.

The BI field specifies a bit in the CR to be used as the condition of the branch.

The next instruction address (NIA) is the effective address of the branch. The NIA is formed by adding a displacement to a base address. The displacement is obtained by concatenating two 0-bits to the right of the BD field and sign-extending the result to 32 bits.

If the AA field contains 0, the base address is the address of the branch instruction, which is also the current instruction address (CIA). If the AA field contains 1, the base address is 0.

The BO field controls options that determine when program flow is transferred to the NIA. The BO field also controls Branch Prediction, a performance-improvement feature. See Section 2.7.4 and Section 2.7.5 for a complete discussion.

If the LK field contains 1, then (CIA + 4) is placed into the LR.

Registers Altered

- CTR if BO_2 contains 0
- LR if LK contains 1

Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

Table 11-6. Extended Mnemonics for bc, bca, bcl, bcla

Mnemonic	Operands	Function	Other Registers Changed
bdnz	target	Decrement CTR. Branch if CTR ≠ 0. *Extended mnemonic* for **bc 16,0,target**	
bdnza		*Extended mnemonic* for **bca 16,0,target**	
bdnzl		*Extended mnemonic* for **bcl 16,0,target**	(LR) ← CIA + 4.
bdnzla		*Extended mnemonic* for **bcla 16,0,target**	(LR) ← CIA + 4.
bdnzf	cr_bit, target	Decrement CTR. Branch if CTR ≠ 0 AND CR_{cr_bit} = 0. *Extended mnemonic* for **bc 0,cr_bit,target**	
bdnzfa		*Extended mnemonic* for **bca 0,cr_bit,target**	
bdnzfl		*Extended mnemonic* for **bcl 0,cr_bit,target**	(LR) ← CIA + 4.
bdnzfla		*Extended mnemonic* for **bcla 0,cr_bit,target**	(LR) ← CIA + 4.
bdnzt	cr_bit, target	Decrement CTR. Branch if CTR ≠ 0 AND CR_{cr_bit} = 1. *Extended mnemonic* for **bc 8,cr_bit,target**	
bdnzta		*Extended mnemonic* for **bca 8,cr_bit,target**	
bdnztl		*Extended mnemonic* for **bcl 8,cr_bit,target**	(LR) ← CIA + 4.
bdnztla		*Extended mnemonic* for **bcla 8,cr_bit,target**	(LR) ← CIA + 4.

Table 11-6. Extended Mnemonics for bc, bca, bcl, bcla (cont.)

Mnemonic	Operands	Function	Other Registers Changed
bdz	target	Decrement CTR. Branch if CTR = 0. *Extended mnemonic* for **bc 18,0,target**	
bdza		*Extended mnemonic* for **bca 18,0,target**	
bdzl		*Extended mnemonic* for **bcl 18,0,target**	(LR) ← CIA + 4.
bdzla		*Extended mnemonic* for **bcla 18,0,target**	(LR) ← CIA + 4.
bdzf	cr_bit, target	Decrement CTR. Branch if CTR = 0 AND CR_{cr_bit} = 0. *Extended mnemonic* for **bc 2,cr_bit,target**	
bdzfa		*Extended mnemonic* for **bca 2,cr_bit,target**	
bdzfl		*Extended mnemonic* for **bcl 2,cr_bit,target**	(LR) ← CIA + 4.
bdzfla		*Extended mnemonic* for **bcla 2,cr_bit,target**	(LR) ← CIA + 4.
bdzt	cr_bit, target	Decrement CTR. Branch if CTR = 0 AND CR_{cr_bit} = 1. *Extended mnemonic* for **bc 10,cr_bit,target**	
bdzta		*Extended mnemonic* for **bca 10,cr_bit,target**	
bdztl		*Extended mnemonic* for **bcl 10,cr_bit,target**	(LR) ← CIA + 4.
bdztla		*Extended mnemonic* for **bcla 10,cr_bit,target**	(LR) ← CIA + 4.
beq	[cr_field,] target	Branch if equal. Use CR0 if cr_field is omitted. *Extended mnemonic* for **bc 12,4*cr_field+2,target**	
beqa		*Extended mnemonic* for **bca 12,4*cr_field+2,target**	
beql		*Extended mnemonic* for **bcl 12,4*cr_field+2,target**	(LR) ← CIA + 4.
beqla		*Extended mnemonic* for **bcla 12,4*cr_field+2,target**	(LR) ← CIA + 4.

Table 11-6. Extended Mnemonics for bc, bca, bcl, bcla (cont.)

Mnemonic	Operands	Function	Other Registers Changed
bf	cr_bit, target	Branch if $CR_{cr\_bit} = 0$. *Extended mnemonic* for **bc 4,cr_bit,target**	
bfa		*Extended mnemonic* for **bca 4,cr_bit,target**	
bfl		*Extended mnemonic* for **bcl 4,cr_bit,target**	LR
bfla		*Extended mnemonic* for **bcla 4,cr_bit,target**	LR
bge	[cr_field,] target	Branch if greater than or equal. Use CR0 if cr_field is omitted. *Extended mnemonic* for **bc 4,4*cr_field+0,target**	
bgea		*Extended mnemonic* for **bca 4,4*cr_field+0,target**	
bgel		*Extended mnemonic* for **bcl 4,4*cr_field+0,target**	LR
bgela		*Extended mnemonic* for **bcla 4,4*cr_field+0,target**	LR
bgt	[cr_field,] target	Branch if greater than. Use CR0 if cr_field is omitted. *Extended mnemonic* for **bc 12,4*cr_field+1,target**	
bgta		*Extended mnemonic* for **bca 12,4*cr_field+1,target**	
bgtl		*Extended mnemonic* for **bcl 12,4*cr_field+1,target**	LR
bgtla		*Extended mnemonic* for **bcla 12,4*cr_field+1,target**	LR
ble	[cr_field,] target	Branch if less than or equal. Use CR0 if cr_field is omitted. *Extended mnemonic* for **bc 4,4*cr_field+1,target**	
blea		*Extended mnemonic* for **bca 4,4*cr_field+1,target**	
blel		*Extended mnemonic* for **bcl 4,4*cr_field+1,target**	LR
blela		*Extended mnemonic* for **bcla 4,4*cr_field+1,target**	LR

Table 11-6. Extended Mnemonics for bc, bca, bcl, bcla (cont.)

Mnemonic	Operands	Function	Other Registers Changed
blt	[cr_field,] target	Branch if less than. Use CR0 if cr_field is omitted. *Extended mnemonic* for **bc 12,4*cr_field+0,target**	
blta		*Extended mnemonic* for **bca 12,4*cr_field+0,target**	
bltl		*Extended mnemonic* for **bcl 12,4*cr_field+0,target**	(LR) ← CIA + 4.
bltla		*Extended mnemonic* for **bcla 12,4*cr_field+0,target**	(LR) ← CIA + 4.
bne	[cr_field,] target	Branch if not equal. Use CR0 if cr_field is omitted. *Extended mnemonic* for **bc 4,4*cr_field+2,target**	
bnea		*Extended mnemonic* for **bca 4,4*cr_field+2,target**	
bnel		*Extended mnemonic* for **bcl 4,4*cr_field+2,target**	(LR) ← CIA + 4.
bnela		*Extended mnemonic* for **bcla 4,4*cr_field+2,target**	(LR) ← CIA + 4.
bng	[cr_field,] target	Branch if not greater than. Use CR0 if cr_field is omitted. *Extended mnemonic* for **bc 4,4*cr_field+1,target**	
bnga		*Extended mnemonic* for **bca 4,4*cr_field+1,target**	
bngl		*Extended mnemonic* for **bcl 4,4*cr_field+1,target**	(LR) ← CIA + 4.
bngla		*Extended mnemonic* for **bcla 4,4*cr_field+1,target**	(LR) ← CIA + 4.
bnl	[cr_field,] target	Branch if not less than. Use CR0 if cr_field is omitted. *Extended mnemonic* for **bc 4,4*cr_field+0,target**	
bnla		*Extended mnemonic* for **bca 4,4*cr_field+0,target**	
bnll		*Extended mnemonic* for **bcl 4,4*cr_field+0,target**	(LR) ← CIA + 4.
bnlla		*Extended mnemonic* for **bcla 4,4*cr_field+0,target**	(LR) ← CIA + 4.

Table 11-6. Extended Mnemonics for bc, bca, bcl, bcla (cont.)

Mnemonic	Operands	Function	Other Registers Changed
bns	[cr_field,] target	Branch if not summary overflow. Use CR0 if cr_field is omitted. *Extended mnemonic* for **bc 4,4*cr_field+3,target**	
bnsa		*Extended mnemonic* for **bca 4,4*cr_field+3,target**	
bnsl		*Extended mnemonic* for **bcl 4,4*cr_field+3,target**	(LR) ← CIA + 4.
bnsla		*Extended mnemonic* for **bcla 4,4*cr_field+3,target**	(LR) ← CIA + 4.
bnu	[cr_field,] target	Branch if not unordered. Use CR0 if cr_field is omitted. *Extended mnemonic* for **bc 4,4*cr_field+3,target**	
bnua		*Extended mnemonic* for **bca 4,4*cr_field+3,target**	
bnul		*Extended mnemonic* for **bcl 4,4*cr_field+3,target**	(LR) ← CIA + 4.
bnula		*Extended mnemonic* for **bcla 4,4*cr_field+3,target**	(LR) ← CIA + 4.
bso	[cr_field,] target	Branch if summary overflow. Use CR0 if cr_field is omitted. *Extended mnemonic* for **bc 12,4*cr_field+3,target**	
bsoa		*Extended mnemonic* for **bca 12,4*cr_field+3,target**	
bsol		*Extended mnemonic* for **bcl 12,4*cr_field+3,target**	(LR) ← CIA + 4.
bsola		*Extended mnemonic* for **bcla 12,4*cr_field+3,target**	(LR) ← CIA + 4.
bt	cr_bit, target	Branch if $CR_{cr\_bit} = 1$. *Extended mnemonic* for **bc 12,cr_bit,target**	
bta		*Extended mnemonic* for **bca 12,cr_bit,target**	
btl		*Extended mnemonic* for **bcl 12,cr_bit,target**	(LR) ← CIA + 4.
btla		*Extended mnemonic* for **bcla 12,cr_bit,target**	(LR) ← CIA + 4.

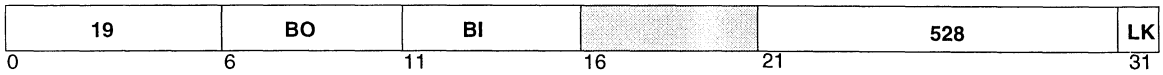
Table 11-6. Extended Mnemonics for bc, bca, bcl, bcla (cont.)

Mnemonic	Operands	Function	Other Registers Changed
bun	[cr_field,] target	Branch if unordered. Use CR0 if cr_field is omitted. *Extended mnemonic* for **bc 12,4*cr_field+3,target**	
buna		*Extended mnemonic* for **bca 12,4*cr_field+3,target**	
bunl		*Extended mnemonic* for **bcl 12,4*cr_field+3,target**	(LR) ← CIA + 4.
bunla		*Extended mnemonic* for **bcla 12,4*cr_field+3,target**	(LR) ← CIA + 4.

bcctr

Branch Conditional to Count Register

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



if BO_2 = 0 then
  CTR ← CTR – 1
if (BO_2 = 1 ∨ ((CTR = 0) = BO_3)) ∧ (BO_0 = 1 ∨ (CR_{BI} = BO_1)) then
  NIA ← $CTR_{0:29}$ || $^{2}0$
else
  NIA ← CIA + 4
if LK = 1 then
  (LR) ← CIA + 4
PC ← NIA

The BI field specifies a bit in the CR to be used as the condition of the branch.

The next instruction address (NIA) is the target address of the branch. The NIA is formed by concatenating the 30 most significant bits of the CTR with two 0-bits on the right.

The BO field controls options that determine when program flow is transferred to the NIA. The BO field also controls Branch Prediction, a performance-improvement feature. See Section 2.7.4 and Section 2.7.5 for a complete discussion.

If the LK field contains 1, then (CIA + 4) is placed into the LR.

Registers Altered

- CTR if BO_2 contains 0
- LR if LK contains 1

Invalid Instruction Forms

- Reserved fields
- If bit 2 of the BO field contains 0, the instruction form is invalid, but the pseudocode applies. If the branch condition is true, the branch is taken; the NIA is the contents of the CTR after it is decremented.

Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

bcctr

Branch Conditional to Count Register

Table 11-7. Extended Mnemonics for bcctr, bcctrl

Mnemonic	Operands	Function	Other Registers Changed
bctr		Branch unconditionally, to address in CTR. *Extended mnemonic* for **bcctr 20,0**	
bctrl		*Extended mnemonic* for **bcctrl 20,0**	(LR) ← CIA + 4.
beqctr	[cr_field]	Branch if equal, to address in CTR. Use CR0 if cr_field is omitted. *Extended mnemonic* for **bcctr 12,4*cr_field+2**	
beqctrl		*Extended mnemonic* for **bcctrl 12,4*cr_field+2**	(LR) ← CIA + 4.
bfctr	cr_bit	Branch if $CR_{cr\_bit} = 0$, to address in CTR. *Extended mnemonic* for **bcctr 4,cr_bit**	
bfctrl		*Extended mnemonic* for **bcctrl 4,cr_bit**	(LR) ← CIA + 4.
bgectr	[cr_field]	Branch if greater than or equal, to address in CTR. Use CR0 if cr_field is omitted. *Extended mnemonic* for **bcctr 4,4*cr_field+0**	
bgectrl		*Extended mnemonic* for **bcctrl 4,4*cr_field+0**	(LR) ← CIA + 4.
bgtctr	[cr_field]	Branch if greater than, to address in CTR. Use CR0 if cr_field is omitted. *Extended mnemonic* for **bcctr 12,4*cr_field+1**	
bgtctrl		*Extended mnemonic* for **bcctrl 12,4*cr_field+1**	(LR) ← CIA + 4.
blectr	[cr_field]	Branch if less than or equal, to address in CTR. Use CR0 if cr_field is omitted. *Extended mnemonic* for **bcctr 4,4*cr_field+1**	
blectrl		*Extended mnemonic* for **bcctrl 4,4*cr_field+1**	(LR) ← CIA + 4.

Table 11-7. Extended Mnemonics for bcctr, bcctrl (cont.)

Mnemonic	Operands	Function	Other Registers Changed
bltctr	[cr_field]	Branch if less than, to address in CTR. Use CR0 if cr_field is omitted. *Extended mnemonic* for **bcctr 12,4*cr_field+0**	
bltctrl		*Extended mnemonic* for **bcctrl 12,4*cr_field+0**	(LR) ← CIA + 4.
bnectr	[cr_field]	Branch if not equal, to address in CTR. Use CR0 if cr_field is omitted. *Extended mnemonic* for **bcctr 4,4*cr_field+2**	
bnectrl		*Extended mnemonic* for **bcctrl 4,4*cr_field+2**	(LR) ← CIA + 4.
bngctr	[cr_field]	Branch if not greater than, to address in CTR. Use CR0 if cr_field is omitted. *Extended mnemonic* for **bcctr 4,4*cr_field+1**	
bngctrl		*Extended mnemonic* for **bcctrl 4,4*cr_field+1**	(LR) ← CIA + 4.
bnlctr	[cr_field]	Branch if not less than, to address in CTR. Use CR0 if cr_field is omitted. *Extended mnemonic* for **bcctr 4,4*cr_field+0**	
bnlctrl		*Extended mnemonic* for **bcctrl 4,4*cr_field+0**	(LR) ← CIA + 4.
bnsctr	[cr_field]	Branch if not summary overflow, to address in CTR. Use CR0 if cr_field is omitted. *Extended mnemonic* for **bcctr 4,4*cr_field+3**	
bnsctrl		*Extended mnemonic* for **bcctrl 4,4*cr_field+3**	(LR) ← CIA + 4.
bnuctr	[cr_field]	Branch if not unordered, to address in CTR. Use CR0 if cr_field is omitted. *Extended mnemonic* for **bcctr 4,4*cr_field+3**	
bnuctrl		*Extended mnemonic* for **bcctrl 4,4*cr_field+3**	(LR) ← CIA + 4.

bcctr

Branch Conditional to Count Register

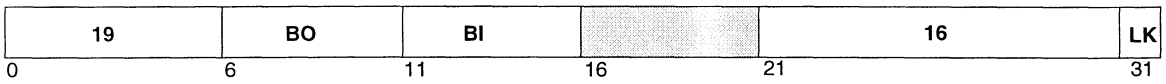
Table 11-7. Extended Mnemonics for bcctr, bcctrl (cont.)

Mnemonic	Operands	Function	Other Registers Changed
bsoctr	[cr_field]	Branch if summary overflow, to address in CTR. Use CR0 if cr_field is omitted. *Extended mnemonic* for **bcctr 12,4*cr_field+3**	
bsoctrl		*Extended mnemonic* for **bcctrl 12,4*cr_field+3**	(LR) ← CIA + 4.
btctr	cr_bit	Branch if CR_{cr_bit} = 1, to address in CTR. *Extended mnemonic* for **bcctr 12,cr_bit**	
btctrl		*Extended mnemonic* for **bcctrl 12,cr_bit**	(LR) ← CIA + 4.
bunctr	[cr_field]	Branch if unordered, to address in CTR. Use CR0 if cr_field is omitted. *Extended mnemonic* for **bcctr 12,4*cr_field+3**	
bunctrl		*Extended mnemonic* for **bcctrl 12,4*cr_field+3**	(LR) ← CIA + 4.

bclr

Branch Conditional to Link Register

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



if BO_2 = 0 then
  CTR ← CTR − 1
if (BO_2 = 1 ∨ ((CTR = 0) = BO_3)) ∧ (BO_0 = 1 ∨ (CR_{BI} = BO_1)) then
  NIA ← $LR_{0:29}$ || $^{2}0$
else
  NIA ← CIA + 4
if LK = 1 then
  (LR) ← CIA + 4
PC ← NIA

If bit 2 of the BO field contains 0, the CTR is decremented.

The BI field specifies a bit in the CR to be used as the condition of the branch.

The next instruction address (NIA) is the target address of the branch. The NIA is formed by concatenating the 30 most significant bits of the LR with two 0-bits on the right.

The BO field controls options that determine when program flow is transferred to the NIA. The BO field also controls Branch Prediction, a performance-improvement feature. See Section 2.7.4 and Section 2.7.5 for a complete discussion.

If the LK field contains 1, then (CIA + 4) is placed into the LR.

Registers Altered

- CTR if BO_2 contains 0
- LR if LK contains 1

Invalid Instruction Forms

- Reserved fields

Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

bclr

Branch Conditional to Link Register

Table 11-8. Extended Mnemonics for bclr, bclrl

Mnemonic	Operands	Function	Other Registers Changed
blr		Branch unconditionally, to address in LR. *Extended mnemonic* for **bclr 20,0**	
blrl		*Extended mnemonic* for **bclrl 20,0**	(LR) ← CIA + 4.
bdnzlr		Decrement CTR. Branch if CTR ≠ 0, to address in LR. *Extended mnemonic* for **bclr 16,0**	
bdnzlrl		*Extended mnemonic* for **bclrl 16,0**	(LR) ← CIA + 4.
bdnzflr	cr_bit	Decrement CTR. Branch if CTR ≠ 0 AND CR_{cr_bit} = 0, to address in LR. *Extended mnemonic* for **bclr 0,cr_bit**	
bdnzflrl		*Extended mnemonic* for **bclrl 0,cr_bit**	(LR) ← CIA + 4.
bdnztlr	cr_bit	Decrement CTR. Branch if CTR ≠ 0 AND CR_{cr_bit} = 1, to address in LR. *Extended mnemonic* for **bclr 8,cr_bit**	
bdnztlrl		*Extended mnemonic* for **bclrl 8,cr_bit**	(LR) ← CIA + 4.
bdzlr		Decrement CTR. Branch if CTR = 0, to address in LR. *Extended mnemonic* for **bclr 18,0**	
bdzlrl		*Extended mnemonic* for **bclrl 18,0**	(LR) ← CIA + 4.

Table 11-8. Extended Mnemonics for bclr, bclrl (cont.)

Mnemonic	Operands	Function	Other Registers Changed
bdzflr	cr_bit	Decrement CTR. Branch if CTR = 0 AND CR_{cr_bit} = 0 to address in LR. *Extended mnemonic* for **bclr 2,cr_bit**	
bdzflrl		*Extended mnemonic* for **bclrl 2,cr_bit**	(LR) ← CIA + 4.
bdztlr	cr_bit	Decrement CTR. Branch if CTR = 0 AND CR_{cr_bit} = 1, to address in LR. *Extended mnemonic* for **bclr 10,cr_bit**	
bdztlrl		*Extended mnemonic* for **bclrl 10,cr_bit**	(LR) ← CIA + 4.
beqlr	[cr_field]	Branch if equal, to address in LR. Use CR0 if cr_field is omitted. *Extended mnemonic* for **bclr 12,4*cr_field+2**	
beqlrl		*Extended mnemonic* for **bclrl 12,4*cr_field+2**	(LR) ← CIA + 4.
bflr	cr_bit	Branch if CR_{cr_bit} = 0, to address in LR. *Extended mnemonic* for **bclr 4,cr_bit**	
bflrl		*Extended mnemonic* for **bclrl 4,cr_bit**	(LR) ← CIA + 4.
bgelr	[cr_field]	Branch if greater than or equal, to address in LR. Use CR0 if cr_field is omitted. *Extended mnemonic* for **bclr 4,4*cr_field+0**	
bgelrl		*Extended mnemonic* for **bclrl 4,4*cr_field+0**	(LR) ← CIA + 4.
bgtlr	[cr_field]	Branch if greater than, to address in LR. Use CR0 if cr_field is omitted. *Extended mnemonic* for **bclr 12,4*cr_field+1**	
bgtlrl		*Extended mnemonic* for **bclrl 12,4*cr_field+1**	(LR) ← CIA + 4.

bclr

Branch Conditional to Link Register

Table 11-8. Extended Mnemonics for bclr, bclrl (cont.)

Mnemonic	Operands	Function	Other Registers Changed
blelr	[cr_field]	Branch if less than or equal, to address in LR. Use CR0 if cr_field is omitted. *Extended mnemonic* for **bclr 4,4*cr_field+1**	
blelrl		*Extended mnemonic* for **bclrl 4,4*cr_field+1**	(LR) ← CIA + 4.
bltlr	[cr_field]	Branch if less than, to address in LR. Use CR0 if cr_field is omitted. *Extended mnemonic* for **bclr 12,4*cr_field+0**	
bltlrl		*Extended mnemonic* for **bclrl 12,4*cr_field+0**	(LR) ← CIA + 4.
bnelr	[cr_field]	Branch if not equal, to address in LR. Use CR0 if cr_field is omitted. *Extended mnemonic* for **bclr 4,4*cr_field+2**	
bnelrl		*Extended mnemonic* for **bclrl 4,4*cr_field+2**	(LR) ← CIA + 4.
bnglr	[cr_field]	Branch if not greater than, to address in LR. Use CR0 if cr_field is omitted. *Extended mnemonic* for **bclr 4,4*cr_field+1**	
bnglrl		*Extended mnemonic* for **bclrl 4,4*cr_field+1**	(LR) ← CIA + 4.
bnllr	[cr_field]	Branch if not less than, to address in LR. Use CR0 if cr_field is omitted. *Extended mnemonic* for **bclr 4,4*cr_field+0**	
bnllrl		*Extended mnemonic* for **bclrl 4,4*cr_field+0**	(LR) ← CIA + 4.
bnslr	[cr_field]	Branch if not summary overflow, to address in LR. Use CR0 if cr_field is omitted. *Extended mnemonic* for **bclr 4,4*cr_field+3**	
bnslrl		*Extended mnemonic* for **bclrl 4,4*cr_field+3**	(LR) ← CIA + 4.

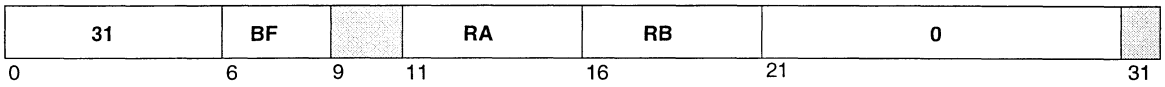
Table 11-8. Extended Mnemonics for bclr, bclrl (cont.)

Mnemonic	Operands	Function	Other Registers Changed
bnulr	[cr_field]	Branch if not unordered, to address in LR. Use CR0 if cr_field is omitted. *Extended mnemonic* for **bclr 4,4*cr_field+3**	
bnulrl		*Extended mnemonic* for **bclrl 4,4*cr_field+3**	(LR) ← CIA + 4.
bsolr	[cr_field]	Branch if summary overflow, to address in LR. Use CR0 if cr_field is omitted. *Extended mnemonic* for **bclr 12,4*cr_field+3**	
bsolrl		*Extended mnemonic* for **bclrl 12,4*cr_field+3**	(LR) ← CIA + 4.
btlr	cr_bit	Branch if $CR_{cr\_bit} = 1$, to address in LR. *Extended mnemonic* for **bclr 12,cr_bit**	
btlrl		*Extended mnemonic* for **bclrl 12,cr_bit**	(LR) ← CIA + 4.
bunlr	[cr_field]	Branch if unordered, to address in LR. Use CR0 if cr_field is omitted. *Extended mnemonic* for **bclr 12,4*cr_field+3**	
bunlrl		*Extended mnemonic* for **bclrl 12,4*cr_field+3**	(LR) ← CIA + 4.

cmp

Compare

cmp BF,0,RA,RB



$c_{0:3} \leftarrow {}^{4}0$
if (RA) < (RB) then $c_0 \leftarrow 1$
if (RA) > (RB) then $c_1 \leftarrow 1$
if (RA) = (RB) then $c_2 \leftarrow 1$
$c_3 \leftarrow$ XER[SO]
n ← BF
CR[CRn] ← $c_{0:3}$

The contents of register RA are compared with the contents of register RB using a 32-bit signed compare.

The CR field specified by the BF field is updated to reflect the results of the compare and the value of XER[SO] is placed into the same CR field.

If instruction bit 31 contains 1, the contents of CR[CR0] are undefined.

Registers Altered

- CR[CR*n*] where *n* is specified by the BF field

Invalid Instruction Forms

- Reserved fields

Programming Note

11

The PowerPC Architecture defines this instruction as **cmp BF,L,RA,RB**, where L selects operand size for 64-bit PowerPC implementations. For all 32-bit PowerPC implementations, L = 0 is required (L = 1 is an invalid form); hence for PPC403GC, use of the extended mnemonic **cmpw BF,RA,RB** is recommended.

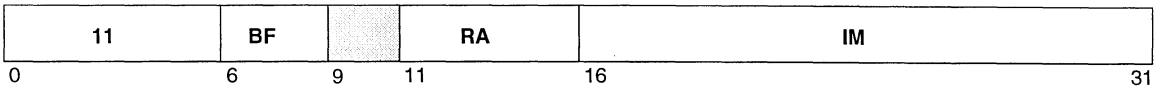
Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

Table 11-9. Extended Mnemonics for cmp

Mnemonic	Operands	Function	Other Registers Changed
cmpw	[BF,] RA, RB	Compare Word. Use CR0 if BF is omitted. *Extended mnemonic* for **cmp BF,0,RA,RB**	

cmpi BF,0,RA,IM



$c_{0:3} \leftarrow {}^{4}0$
if (RA) < EXTS(IM) then $c_0 \leftarrow 1$
if (RA) > EXTS(IM) then $c_1 \leftarrow 1$
if (RA) = EXTS(IM) then $c_2 \leftarrow 1$
c3 ← XER[SO]
n ← BF
CR[CRn] ← $c_{0:3}$

The IM field is sign-extended to 32 bits. The contents of register RA are compared with the extended IM field, using a 32-bit signed compare.

The CR field specified by the BF field is updated to reflect the results of the compare and the value of XER[SO] is placed into the same CR field.

Registers Altered

- CR[CR*n*] where *n* is specified by the BF field

Invalid Instruction Forms

- Reserved fields

Programming Note

The PowerPC Architecture defines this instruction as **cmpi BF,L,RA,IM**, where L selects operand size for 64-bit PowerPC implementations. For all 32-bit PowerPC implementations, L = 0 is required (L = 1 is an invalid form); hence for PPC403GC, use of the extended mnemonic **cmpwi BF,RA,IM** is recommended.

Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

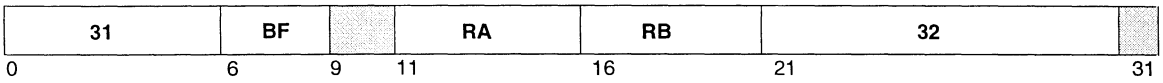
Table 11-10. Extended Mnemonics for cmpi

Mnemonic	Operands	Function	Other Registers Changed
cmpwi	[BF,] RA, IM	Compare Word Immediate. Use CR0 if BF is omitted. *Extended mnemonic* for **cmpi BF,0,RA,IM**	

cmpl

Compare Logical

cmpl BF,0,RA,RB



$c_{0:3} \leftarrow {}^{4}0$
if (RA) $<^{u}$ (RB) then $c_0 \leftarrow 1$
if (RA) $>^{u}$ (RB) then $c_1 \leftarrow 1$
if (RA) = (RB) then $c_2 \leftarrow 1$
$c_3 \leftarrow$ XER[SO]
n $\leftarrow$ BF
CR[CRn] $\leftarrow c_{0:3}$

The contents of register RA are compared with the contents of register RB, using a 32-bit unsigned compare.

The CR field specified by the BF field is updated to reflect the results of the compare and the value of XER[SO] is placed into the same CR field.

If instruction bit 31 contains 1, the contents of CR[CR0] are undefined.

Registers Altered

- CR[CR*n*] where *n* is specified by the BF field

Invalid Instruction Forms

- Reserved fields

Programming Notes

11

The PowerPC Architecture defines this instruction as **cmpl BF,L,RA,RB**, where L selects operand size for 64-bit PowerPC implementations. For all 32-bit PowerPC implementations, L = 0 is required (L = 1 is an invalid form); hence for PPC403GC, use of the extended mnemonic **cmplw BF,RA,RB** is recommended.

Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

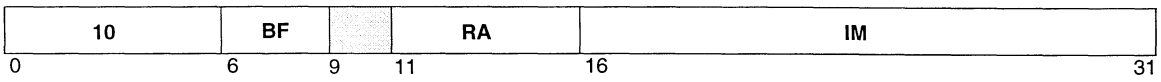
Table 11-11. Extended Mnemonics for cmpl

Mnemonic	Operands	Function	Other Registers Changed
cmplw	[BF,] RA, RB	Compare Logical Word. Use CR0 if BF is omitted. *Extended mnemonic* for **cmpl BF,0,RA,RB**	

cmpli

Compare Logical Immediate

cmpli BF,0,RA,IM



$c_{0:3} \leftarrow {}^{4}0$
if (RA) $\begin{matrix} < \\ > \\ = \end{matrix}^{u}$ ($^{16}0$ || IM) then $c_0 \leftarrow 1$
if (RA) $\begin{matrix} < \\ > \\ = \end{matrix}^{u}$ ($^{16}0$ || IM) then $c_1 \leftarrow 1$
if (RA) $\begin{matrix} < \\ > \\ = \end{matrix}^{u}$ ($^{16}0$ || IM) then $c_2 \leftarrow 1$
$c_3 \leftarrow$ XER[SO]
n $\leftarrow$ BF
CR[CRn] $\leftarrow c_{0:3}$

The IM field is extended to 32 bits by concatenating 16 0-bits to its left. The contents of register RA are compared with IM using a 32-bit unsigned compare.

The CR field specified by the BF field is updated to reflect the results of the compare and the value of XER[SO] is placed into the same CR field.

Registers Altered

- CR[CR*n*] where *n* is specified by the BF field

Invalid Instruction Forms

- Reserved fields

Programming Note

The PowerPC Architecture defines this instruction as **cmpli BF,L,RA,IM**, where L selects operand size for 64-bit PowerPC implementations. For all 32-bit PowerPC implementations, L = 0 is required (L = 1 is an invalid form); hence for PPC403GC, use of the extended mnemonic **cmplwi BF,RA,IM** is recommended.

Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

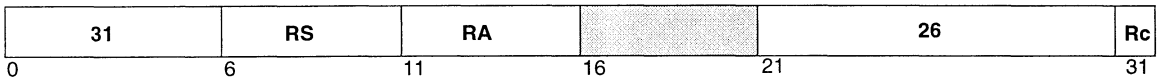
Table 11-12. Extended Mnemonics for cmpli

Mnemonic	Operands	Function	Other Registers Changed
cmplwi	[BF,] RA, IM	Compare Logical Word Immediate. Use CR0 if BF is omitted. *Extended mnemonic* for **cmpli BF,0,RA,IM**	

cntlzw

Count Leading Zeros Word

cntlzw	RA,RS	(Rc=0)
cntlzw.	RA,RS	(Rc=1)




```
n ← 0
do while n < 32
   if (RS)n = 1 then leave
   n ← n + 1
(RA) ← n
```

The consecutive leading 0 bits in register RS are counted; the count is placed into register RA.

The count ranges from 0 through 32, inclusive.

Registers Altered

- RA
- $CR[CR0]_{LT, GT, EQ, SO}$ if Rc contains 1

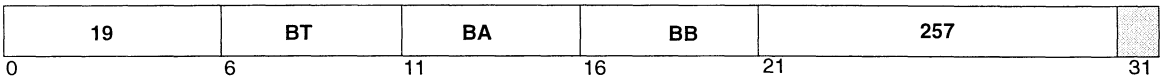
Invalid Instruction Forms

- Reserved fields

crand

Condition Register AND

crand BT,BA,BB



$CR_{BT} \leftarrow CR_{BA} \wedge CR_{BB}$

The CR bit specified by the BA field is ANDed with the CR bit specified by the BB field; the result is placed into the CR bit specified by the BT field.

Registers Altered

- CR

Invalid Instruction Forms

- Reserved fields

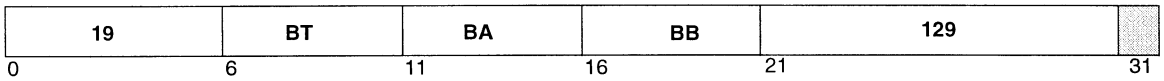
Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

crandc

Condition Register AND with Complement

crandc BT,BA,BB



$CR_{BT} \leftarrow CR_{BA} \wedge \neg CR_{BB}$

The CR bit specified by the BA field is ANDed with the ones complement of the CR bit specified by the BB field; the result is placed into the CR bit specified by the BT field.

Registers Altered

- CR

Invalid Instruction Forms

- Reserved fields

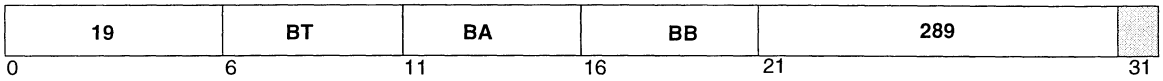
Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

creqv

Condition Register Equivalent

creqv BT,BA,BB



$CR_{BT} \leftarrow \neg(CR_{BA} \oplus CR_{BB})$

The CR bit specified by the BA field is XORed with the CR bit specified by the BB field; the ones complement of the result is placed into the CR bit specified by the BT field.

Registers Altered

- CR

Invalid Instruction Forms

- Reserved fields

Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

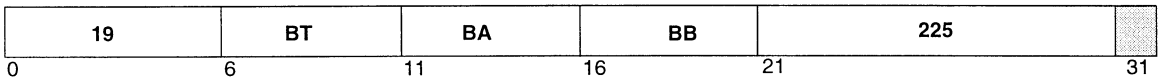
Table 11-13. Extended Mnemonics for creqv

Mnemonic	Operands	Function	Other Registers Changed
crset	bx	Condition register set. *Extended mnemonic* for **creqv bx,bx,bx**	

crnand

Condition Register NAND

crnand BT,BA,BB



$CR_{BT} \leftarrow \neg(CR_{BA} \wedge CR_{BB})$

The CR bit specified by the BA field is ANDed with the CR bit specified by the BB field; the ones complement of the result is placed into the CR bit specified by the BT field.

Registers Altered

- CR

Invalid Instruction Forms

- Reserved fields

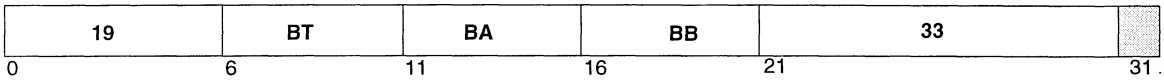
Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

crnor

Condition Register NOR

crnor BT,BA,BB



$CR_{BT} \leftarrow \neg(CR_{BA} \vee CR_{BB})$

The CR bit specified by the BA field is ORed with the CR bit specified by the BB field; the ones complement of the result is placed into the CR bit specified by the BT field.

Registers Altered

- CR

Invalid Instruction Forms

- Reserved fields

Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

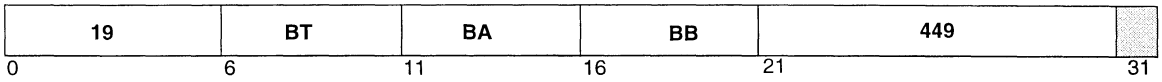
Table 11-14. Extended Mnemonics for crnor

Mnemonic	Operands	Function	Other Registers Changed
crnot	bx, by	Condition register not. *Extended mnemonic* for **crnor bx,by,by**	

cror

Condition Register OR

cror BT,BA,BB



$CR_{BT} \leftarrow CR_{BA} \vee CR_{BB}$

The CR bit specified by the BA field is ORed with the CR bit specified by the BB field; the result is placed into the CR bit specified by the BT field.

Registers Altered

- CR

Invalid Instruction Forms

- Reserved fields

Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

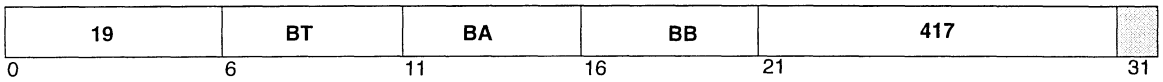
Table 11-15. Extended Mnemonics for cror

Mnemonic	Operands	Function	Other Registers Changed
crmove	bx, by	Condition register move. *Extended mnemonic* for **cror bx,by,by**	

crorc

Condition Register OR with Complement

crorc BT,BA,BB



$CR_{BT} \leftarrow CR_{BA} \vee \neg CR_{BB}$

The condition register (CR) bit specified by the BA field is ORed with the ones complement of the CR bit specified by the BB field; the result is placed into the CR bit specified by the BT field.

Registers Altered

- CR

Invalid Instruction Forms

- Reserved fields

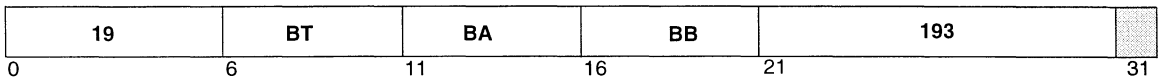
Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

crxor

Condition Register XOR

crxor BT,BA,BB



$CR_{BT} \leftarrow CR_{BA} \oplus CR_{BB}$

The CR bit specified by the BA field is XORed with the CR bit specified by the BB field; the result is placed into the CR bit specified by the BT field.

Registers Altered

- CR

Invalid Instruction Forms

- Reserved fields

Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

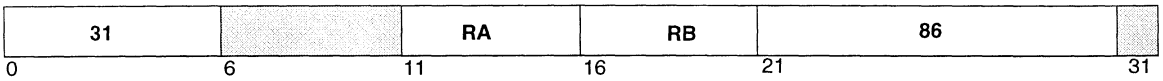
Table 11-16. Extended Mnemonics for crxor

Mnemonic	Operands	Function	Other Registers Changed
crclr	bx	Condition register clear. *Extended mnemonic* for **crxor bx,bx,bx**	

dcbf

Data Cache Block Flush

dcbf RA,RB



EA ← (RAI0) + (RB)
DCBF(EA)

An effective address is formed by adding an index to a base address. The index is the contents of register RB. The base address is 0 if the RA field is 0 and is the contents of register RA otherwise.

If the data block corresponding to the effective address is in the data cache and marked as modified (stored into), the data block is copied back to main storage and then marked invalid in the data cache. If the data block is not marked as modified, it is simply marked invalid in the data cache. The operation is performed whether or not the effective address is marked as cacheable.

If the data block at the effective address is not in the data cache, no operation is performed.

If instruction bit 31 contains 1, the contents of CR[CR0] are undefined.

Registers Altered

- None

Invalid Instruction Forms

- Reserved fields

Exceptions

This instruction is considered a "load" with respect to Data Storage exceptions. See Section 9.4.3 (Access Protection for Cache Instructions) on page 9-20 for further discussion.

This instruction is considered a "store" with respect to data address compare (DAC) debug exceptions. See Section 10.6.3.1 (DAC Applied to Cache Instructions) on page 10-12 for further discussion.

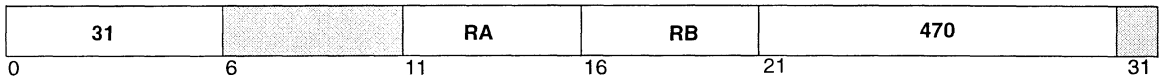
Architecture Note

This instruction is part of the PowerPC Virtual Environment Architecture.

dcbi

Data Cache Block Invalidate

dcbi RA,RB



EA ← (RAI0) + (RB)
DCBI(EA)

An effective address is formed by adding an index to a base address. The index is the contents of register RB. The base address is 0 if the RA field is 0 and is the contents of register RA otherwise.

If the data block at the effective address is in the data cache, the data block is marked invalid, regardless of whether or not the effective address is marked as cacheable. If modified data existed in the data block prior to the operation of this instruction, that data is lost.

If the data block at the effective address is not in the data cache, no operation is performed.

If instruction bit 31 contains 1, the contents of CR[CR0] are undefined.

Registers Altered

- None

Invalid Instruction Forms

- Reserved fields

Programming Notes

11

Execution of this instruction is privileged.

Exceptions

This instruction is considered a "store" with respect to Data Storage exceptions. See Section 9.4.3 (Access Protection for Cache Instructions) on page 9-20 for further discussion.

This instruction is considered a "store" with respect to data address compare (DAC) debug exceptions. See Section 10.6.3.1 (DAC Applied to Cache Instructions) on page 10-12 for further discussion.

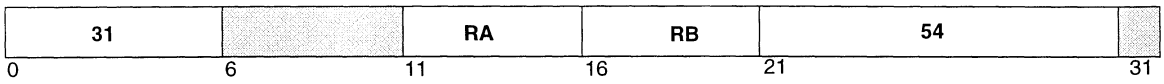
Architecture Note

This instruction is part of the PowerPC Operating Environment Architecture.

dcbst

Data Cache Block Store

dcbst RA,RB



EA ← (RAI0) + (RB)
DCBST(EA)

An effective address is formed by adding an index to a base address. The index is the contents of register RB. The base address is 0 if the RA field is 0, and is the contents of register RA otherwise.

If the data block at the effective address is in the data cache and marked as modified, the data block is copied back to main storage and marked as unmodified in the data cache.

If the data block at the effective address is in the data cache, and is not marked as modified, or if the data block at the effective address is not in the data cache, no operation is performed.

The operation specified by this instruction is performed whether or not the effective address is marked as cacheable.

If instruction bit 31 contains 1, the contents of CR[CR0] are undefined.

Registers Altered

- None

Invalid Instruction Forms

- Reserved fields

Exceptions

This instruction is considered a "load" with respect to Data Storage exceptions. See Section 9.4.3 (Access Protection for Cache Instructions) on page 9-20 for further discussion.

This instruction is considered a "store" with respect to data address compare (DAC) debug exceptions. See Section 10.6.3.1 (DAC Applied to Cache Instructions) on page 10-12 for further discussion.

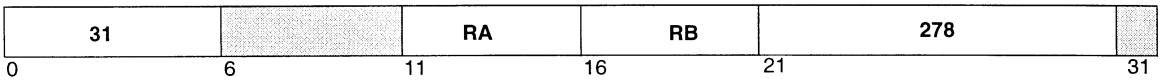
Architecture Note

This instruction is part of the PowerPC Virtual Environment Architecture.

dcbt

Data Cache Block Touch

dcbt RA,RB



EA ← (RA|0) + (RB)
DCBT(EA)

An effective address is formed by adding an index to a base address. The index is the contents of register RB. The base address is 0 when the RA field is 0, and is the contents of register RA otherwise.

If the data block at the effective address is not in the data cache and the effective address is marked as cacheable, the block is read from main storage into the data cache.

If the data block at the effective address is in the data cache, or if the effective address is marked as non-cacheable, no operation is performed.

This instruction is not allowed to cause Data Storage Exceptions or Data TLB Miss Exceptions. If execution of the instruction would otherwise cause such an exception, then no operation is performed, and no exception occurs.

If instruction bit 31 contains 1, the contents of CR[CR0] are undefined.

Registers Altered

- None

Invalid Instruction Forms

- Reserved fields

Programming Notes

The **dcbt** instruction allows a program to begin a cache block fetch from main storage before the program needs the data. The program can later load data from the cache into registers without incurring the latency of a cache miss.

Exceptions

This instruction is considered a "load" with respect to Data Storage exceptions. See Section 9.4.3 (Access Protection for Cache Instructions) on page 9-20 for further discussion.

This instruction is considered a "load" with respect to data address compare (DAC) debug exceptions. See Section 10.6.3.1 (DAC Applied to Cache Instructions) on page 10-12 for further discussion.

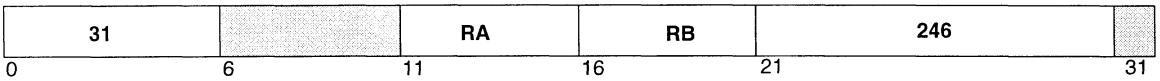
Architecture Note

This instruction is part of the PowerPC Virtual Environment Architecture.

dcbtst

Data Cache Block Touch for Store

dcbtst RA,RB



EA ← (RAI0) + (RB)
DCBTST(EA)

An effective address is formed by adding an index to a base address. The index is the contents of register RB. The base address is 0 if the RA field is 0 and is the contents of register RA otherwise.

If the data block at the effective address is not in the data cache and the effective address is marked as cacheable, the data block is loaded into the data cache.

If the effective address is marked as non-cacheable, or if the data block at the effective address is in the data cache, no operation is performed.

This instruction is not allowed to cause Data Storage Exceptions or Data TLB Miss Exceptions. If execution of the instruction would otherwise cause such an exception, then no operation is performed, and no exception occurs.

If instruction bit 31 contains 1, the contents of CR[CR0] are undefined.

Registers Altered

- None

Invalid Instruction Forms

- Reserved fields

Programming Notes

The **dcbtst** instruction allows a program to begin a cache block fetch from main storage before the program needs the data. The program can later store data from GPRs into the cache block, without incurring the latency of a cache miss.

Architecturally, **dcbtst** brings data into the cache in "Exclusive" mode, which allows the program to alter the cached data. "Exclusive" mode is part of the MESI protocol for multi-processor systems, and is not implemented on the PPC403GC. The implementation of the **dcbtst** instruction on the PPC403GC is identical to the implementation of the **dcbt** instruction.

Exceptions

This instruction is considered a "load" with respect to Data Storage exceptions. See Section 9.4.3 (Access Protection for Cache Instructions) on page 9-20 for further discussion.

This instruction is considered a "load" with respect to data address compare (DAC) debug exceptions. See Section 10.6.3.1 (DAC Applied to Cache Instructions) on page 10-12 for further discussion.

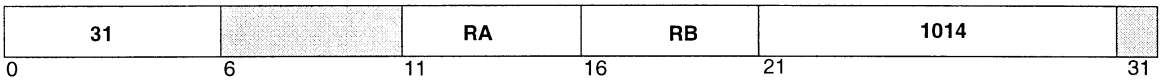
Architecture Note

This instruction is part of the PowerPC Virtual Environment Architecture.

dcbz

Data Cache Block Set to Zero

dcbz RA,RB



EA ← (RAI0) + (RB)
DCBZ(EA)

An effective address is formed by adding an index to a base address. The index is the contents of register RB. The base address is 0 if the RA field is 0 and is the contents of register RA otherwise.

If the data block at the effective address is in the data cache and the effective address is marked as cacheable and non-write-through, the data in the cache block is set to 0.

If the data block at the effective address is not in the data cache and the effective address is marked as cacheable and non-write-through, a cache block is established and set to 0. Note that nothing is read from main storage, as described in the programming note below.

If the effective address is marked as non-cacheable or as write-through, an alignment exception occurs.

If instruction bit 31 contains 1, the contents of CR[CR0] are undefined.

Registers Altered

- None

Invalid Instruction Forms

- Reserved fields

Programming Notes

Because the **dcbz** instruction can establish an address in the data cache without copying the contents of that address from main storage, the address established may be invalid with respect to the storage subsystem. A subsequent operation may cause the address to be copied back to main storage to make room for a new data block. A machine check exception could result.

If **dcbz** is attempted to an effective address which is marked as non-cacheable, the software alignment exception handler should emulate the instruction by storing zeros to the block in main storage. Note: if a data block corresponding to the effective address exists in the cache, but the effective address is non-cacheable, **dcbz** to that address is considered a programming error (see Section 8.2.3 on page 8-8).

If **dcbz** is attempted to an effective address which is marked as write-through, the software alignment exception handler should emulate the instruction by storing zeros to the block in main storage. An effective address which is marked as write-through should also be marked as cacheable; when **dcbz** is attempted to such an address, the alignment exception handler should maintain coherency of cache and memory.

Exceptions

If **dcbz** is attempted to an effective address which is marked as non-cacheable or as write-through, an alignment exception occurs.

This instruction is considered a "store" with respect to Data Storage exceptions. See Section 9.4.3 (Access Protection for Cache Instructions) on page 9-20 for further discussion.

This instruction is considered a "store" with respect to data address compare (DAC) debug exceptions. See Section 10.6.3.1 (DAC Applied to Cache Instructions) on page 10-12 for further discussion.

Architecture Note

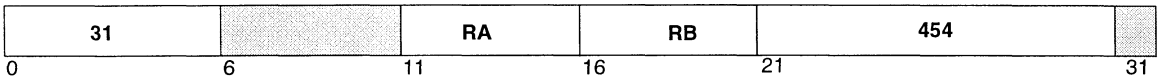
This instruction is part of the PowerPC Virtual Environment Architecture.

This instruction is specific to the PowerPC Embedded Controller family

dccci

Data Cache Congruence Class Invalidate

dccci RA,RB



EA ← (RAI0) + (RB)
DCCCI(EA)

An effective address is formed by adding an index to a base address. The index is the contents of register RB. The base address is 0 if the RA field is 0 and is the contents of register RA otherwise.

Both cache lines in the congruence class specified by $EA_{23:27}$ are invalidated, whether or not they match the effective address. If modified data existed in the cache congruence class prior to the operation of this instruction, that data is lost.

The operation specified by this instruction is performed whether or not the effective address is marked as cacheable.

If instruction bit 31 contains 1, the contents of CR[CR0] are undefined.

Registers Altered

- None

Invalid Instruction Forms

- Reserved fields

Programming Note

11

Execution of this instruction is privileged.

This instruction is intended for use in the power-on reset routine to invalidate the entire data cache tag array before enabling the data cache. A series of **dccci** instruction should be executed, one for each congruence class. Cacheability can then be enabled.

This instruction is specific to the PowerPC Embedded Controller family

dccci

Data Cache Congruence Class Invalidate

Exceptions

The execution of a **dccci** instruction can cause a Data TLB Miss Exception, at the specified effective address, in spite of the non-specific intent of that effective address.

This instruction is considered a "store" with respect to Data Storage exceptions. See Section 9.4.3 (Access Protection for Cache Instructions) on page 9-20 for further discussion.

This instruction will not cause data address compare (DAC) debug exceptions. See Section 10.6.3.1 (DAC Applied to Cache Instructions) on page 10-12 for further discussion.

Architecture Note

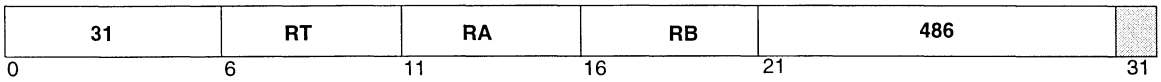
This instruction is specific to the PowerPC Embedded Controller family; it is not described in *PowerPC Architecture*. Programs using this instruction may not be portable to other PowerPC implementations.

This instruction is specific to the PowerPC Embedded Controller family

dcread

Data Cache Read

dcread RT,RA,RB



EA ← (RAI0) + (RB)
if ((CDBCR[CIS] = 0) ∧ (CDBCR[CSS] = 0)) then (RT) ← (d-cache data, side A)
if ((CDBCR[CIS] = 0) ∧ (CDBCR[CSS] = 1)) then (RT) ← (d-cache data, side B)
if ((CDBCR[CIS] = 1) ∧ (CDBCR[CSS] = 0)) then (RT) ← (d-cache tag, side A)
if ((CDBCR[CIS] = 1) ∧ (CDBCR[CSS] = 1)) then (RT) ← (d-cache tag, side B)

An effective address is formed by adding an index to a base address. The index is the contents of register RB. The base address is 0 if the RA field is 0 and is the contents of register RA otherwise.

This instruction is a debugging tool for reading the data cache entries for the congruence class specified by $EA_{23:27}$. The cache information will be read into the General Purpose Register RT.

If (CDBCR[CIS] = 0), the information will be one word of data-cache data from the addressed congruence class. The word is specified by $EA_{28:29}$ ($EA_{0:22}$ are ignored; an alignment exception will result if $EA_{30:31} \neq 00$). If (CDBCR[CSS] = 0), the data will be from the A-side, otherwise from the B-side.

If (CDBCR[CIS] = 1), the information will be a cache tag from the addressed congruence class ($EA_{0:22}$ and $EA_{28:29}$ are ignored; an alignment exception will result if $EA_{30:31} \neq 00$). If (CDBCR[CSS] = 0), the tag will be from the A-side, otherwise from the B-side. Data cache tag information is placed into register RT as follows:

0:22	TAG	Cache Tag
23:25		reserved
26	D	Cache Line Dirty 0 - Not dirty 1 - Dirty
27	V	Cache Line Valid 0 - Not valid 1 - Valid
28:30		reserved
31	LRU	Least Recently Used 0 - A side least-recently-used 1 - B side least-recently-used

If instruction bit 31 contains 1, the contents of CR[CR0] are undefined.

This instruction is specific to the PowerPC Embedded Controller family

dcread

Data Cache Read

Registers Altered

- RT

Invalid Instruction Forms

- Reserved fields

Programming Note

Execution of this instruction is privileged.

Exceptions

An alignment exception will result if $EA_{30:31} \neq 00$.

The execution of a **dcread** instruction can cause a Data TLB Miss Exception, at the specified effective address, in spite of the non-specific intent of that effective address.

This instruction is considered a "load" with respect to Data Storage exceptions. See Section 9.4.3 (Access Protection for Cache Instructions) on page 9-20 for further discussion.

This instruction is considered a "load" with respect to data address compare (DAC) debug exceptions. See Section 10.6.3.1 (DAC Applied to Cache Instructions) on page 10-12 for further discussion.

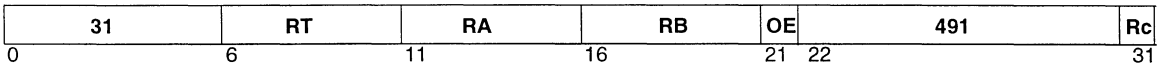
Architecture Note

This instruction is specific to the PowerPC Embedded Controller family; it is not described in *PowerPC Architecture*. Programs using this instruction may not be portable to other PowerPC implementations.

divw

Divide Word

divw	RT,RA,RB	(OE=0, Rc=0)
divw.	RT,RA,RB	(OE=0, Rc=1)
divwo	RT,RA,RB	(OE=1, Rc=0)
divwo.	RT,RA,RB	(OE=1, Rc=1)



(RT) ← (RA) ÷ (RB)

The contents of register RA are divided by the contents of register RB. The quotient is placed into register RT.

Both the dividend and the divisor are interpreted as signed integers. The quotient is the unique signed integer that satisfies:

dividend = (quotient × divisor) + remainder

where the remainder has the same sign as the dividend and its magnitude is less than that of the divisor.

If an attempt is made to perform (x'8000 0000' ÷ –1) or (n ÷ 0), the contents of register RT are undefined; if the Rc also contains 1, the contents of CR[CR0] are undefined. Either invalid division operation sets XER[OV, SO] to 1 if the OE field contains 1.

Registers Altered

- RT
- $CR[CR0]_{LT, GT, EQ, SO}$ if Rc contains 1
- XER[OV, SO] if OE contains 1

Programming Note

The 32-bit remainder can be calculated using the following sequence of instructions:

```
divw    RT,RA,RB    # RT = quotient
mullw   RT,RT,RB    # RT = quotient × divisor
subf    RT,RT,RA    # RT = remainder
```

The sequence does not calculate correct results for the invalid divide operations.

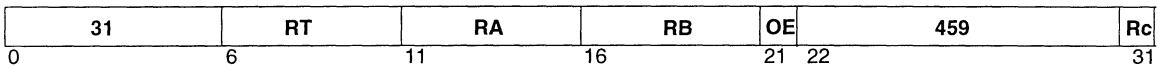
Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

divwu

Divide Word Unsigned

divwu	RT,RA,RB	(OE=0, Rc=0)
divwu.	RT,RA,RB	(OE=0, Rc=1)
divwuo	RT,RA,RB	(OE=1, Rc=0)
divwuo.	RT,RA,RB	(OE=1, Rc=1)



(RT) ← (RA) ÷ (RB)

The contents of register RA are divided by the contents of register RB. The quotient is placed into register RT.

The dividend and the divisor are interpreted as unsigned integers. The quotient is the unique unsigned integer that satisfies

dividend = (quotient × divisor) + remainder

If an attempt is made to perform ($n \div 0$), the contents of register RT are undefined; if the Rc also contains 1, the contents of CR[CR0] are also undefined. The invalid division operation also sets XER[OV, SO] to 1 if the OE field contains 1.

Registers Altered

- RT
- $CR[CR0]_{LT, GT, EQ, SO}$ if Rc contains 1
- XER[OV, SO] if OE contains 1

Programming Note

The 32-bit remainder can be calculated using the following sequence of instructions

```
divwu     RT,RA,RB      # RT = quotient
mullwu    RT,RT,RB      # RT = quotient × divisor
subf      RT,RT,RA      # RT = remainder
```

This sequence does not calculate the correct result if the divisor is zero.

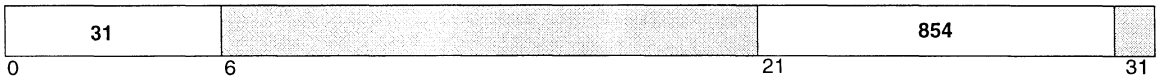
Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

eieio

Enforce In Order Execution of I/O

eieio



The **eieio** instruction ensures that all loads and stores preceding an **eieio** instruction complete with respect to main storage before any loads and stores following the **eieio** instruction access main storage.

If instruction bit 31 contains 1, the contents of CR[CR0] are undefined.

Registers Altered

- None

Invalid Instruction Forms

- Reserved fields

Programming Note

Architecturally, **eieio** orders storage access, not instruction completion. Therefore, non-storage operations after **eieio** could complete before storage operations that were before **eieio**. The **sync** instruction guarantees ordering of both instruction completion and storage access. For the PPC403GC, the **eieio** instruction is implemented to behave as a **sync** instruction. To write code which is portable between various PowerPC implementations, programmers should use the mnemonic which corresponds to the desired behavior.

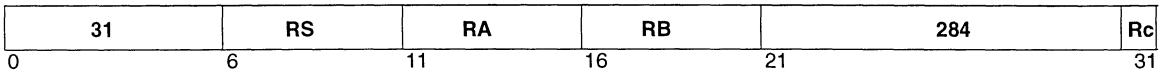
Architecture Note

This instruction is part of the PowerPC Virtual Environment Architecture.

eqv

Equivalent

eqv	RA,RS,RB	(Rc=0)
eqv.	RA,RS,RB	(Rc=1)



(RA) $\leftarrow$ $\neg$((RS) $\oplus$ (RB))

The contents of register RS are XORed with the contents of register RB; the ones complement of the result is placed into register RA.

Registers Altered

- RA
- $CR[CR0]_{LT, GT, EQ, SO}$ if Rc contains 1

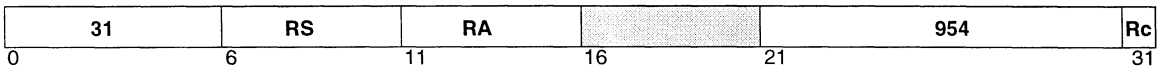
Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

extsb

Extend Sign Byte

extsb	RA,RS	(Rc=0)
extsb.	RA,RS	(Rc=1)



(RA) $\leftarrow$ EXTS(RS)$_{24:31}$

The least significant byte of register RS is sign-extended to 32 bits by replicating bit 24 of the register into bits 0 through 23 of the result. The result is placed into register RA.

Registers Altered

- RA
- CR[CR0]$_{LT, GT, EQ, SO}$ if Rc contains 1

Invalid Instruction Forms

- Reserved fields

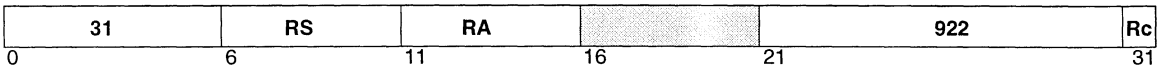
Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

extsh

Extend Sign Halfword

extsh RA,RS (Rc=0)
extsh. RA,RS (Rc=1)



(RA) ← $EXTS(RS)_{16:31}$

The least significant halfword of register RS is sign-extended to 32 bits by replicating bit 16 of the register into bits 0 through 15 of the result. The result is placed into register RA.

Registers Altered

- RA
- $CR[CR0]_{LT, GT, EQ, SO}$ if Rc contains 1

Invalid Instruction Forms

- Reserved fields

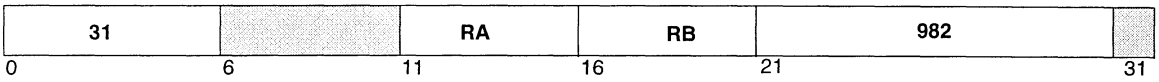
Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

icbi

Instruction Cache Block Invalidate

icbi RA,RB



EA ← (RAI0) + (RB)
ICBI(EA)

An effective address is formed by adding an index to a base address. The index is the contents of register RB. The base address is 0 if the RA field is 0 and is the contents of register RA otherwise.

If the instruction block at the effective address is in the instruction cache, the cache block is marked invalid.

If the instruction block at the effective address is not in the instruction cache, no operation is performed.

The operation specified by this instruction is performed whether or not the effective address is marked as cacheable.

If instruction bit 31 contains 1, the contents of CR[CR0] are undefined.

Registers Altered

- None

Invalid Instruction Forms

- Reserved fields

11

Programming Note

Instruction cache ops use MSR[DR], not MSR[IR], to determine translation of their operands.

When data translation is disabled, cacheability for the effective address of the operand of instruction cache ops is determined by the ICCR, not the DCCR.

Exceptions

Instruction Storage Exceptions and Instruction-side TLB Miss Exceptions are associated with the fetching of instructions, not with the execution of instructions. Exceptions that occur during the execution of instruction cache ops cause data-side exceptions (Data Storage Exceptions and Data TLB Miss Exceptions).

This instruction is considered a "load" with respect to Data Storage exceptions. See Section 9.4.3 (Access Protection for Cache Instructions) on page 9-20 for further discussion.

This instruction is considered a "load" with respect to data address compare (DAC) debug exceptions. See Section 10.6.3.1 (DAC Applied to Cache Instructions) on page 10-12 for further discussion.

Architecture Note

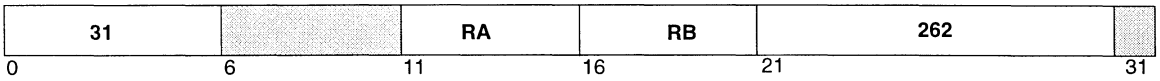
This instruction is part of the PowerPC Virtual Environment Architecture.

This instruction is specific to the PowerPC Embedded Controller family

icbt

Instruction Cache Block Touch

icbt RA,RB



EA ← (RAI0) + (RB)
ICBT(EA)

An effective address is formed by adding an index to a base address. The index is the contents of register RB. The base address is 0 if the RA field is 0 and is the contents of register RA otherwise.

If the instruction block at the effective address is not in the instruction cache, and is marked as cacheable, the instruction block is loaded into the instruction cache.

If the instruction block at the effective address is already in the instruction cache, or if the effective address is marked as non-cacheable, no operation is performed.

This instruction is not allowed to cause Data Storage Exceptions or Data TLB Miss Exceptions. If execution of the instruction would otherwise cause such an exception, then no operation is performed, and no exception occurs.

If instruction bit 31 contains 1, the contents of CR[CR0] are undefined.

Registers Altered

- None

Invalid Instruction Forms

- Reserved fields

Programming Notes

Execution of this instruction is privileged.

This instruction allows a program to begin a cache block fetch from main storage before the program needs the instruction. The program can later branch to the instruction address and fetch the instruction from the cache without incurring the latency of a cache miss.

Instruction cache ops use MSR[DR], not MSR[IR], to determine translation of their operands.

When data translation is disabled, cacheability for the effective address of the operand of instruction cache ops is determined by the ICCR, not the DCCR.

This instruction is specific to the PowerPC Embedded Controller family

icbt

Instruction Cache Block Touch

Exceptions

Instruction Storage Exceptions and Instruction-side TLB Miss Exceptions are associated with the fetching of instructions, not with the execution of instructions. Exceptions that occur during the execution of instruction cache ops cause data-side exceptions (Data Storage Exceptions and Data TLB Miss Exceptions).

This instruction is not allowed to cause Data TLB Miss Exceptions. If execution of the instruction would otherwise cause such an exception, then no operation is performed, and no exception occurs.

This instruction is considered a "load" with respect to Data Storage exceptions. See Section 9.4.3 (Access Protection for Cache Instructions) on page 9-20 for further discussion.

This instruction is considered a "load" with respect to data address compare (DAC) debug exceptions. See Section 10.6.3.1 (DAC Applied to Cache Instructions) on page 10-12 for further discussion.

Architecture Note

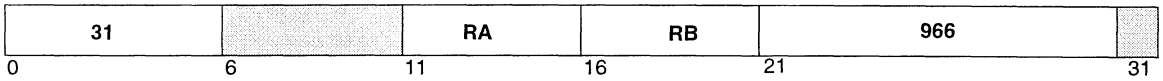
This instruction is specific to the PowerPC Embedded Controller family; it is not described in *PowerPC Architecture*. Programs using this instruction may not be portable to other PowerPC implementations.

This instruction is specific to the PowerPC Embedded Controller family

iccci

Instruction Cache Congruence Class Invalidate

iccci RA,RB



EA ← (RAI0) + (RB)
ICCCI(EA)

An effective address is formed by adding an index to a base address. The index is the contents of register RB. The base address is 0 if the RA field is 0 and is the contents of register RA otherwise.

Both cache lines in the congruence class specified by $EA_{22:27}$ are invalidated, whether or not they match the effective address.

The operation specified by this instruction is performed whether or not the effective address is marked cacheable.

If instruction bit 31 contains 1, the contents of CR[CR0] are undefined.

Registers Altered

- None

Invalid Instruction Forms

- Reserved fields

Programming Notes

11

Execution of this instruction is privileged.

This instruction is intended for use in the power-on reset routine to invalidate the entire instruction cache tag array before enabling the instruction cache. A series of **iccci** instructions should be executed, one for each congruence class. Cacheability can then be enabled.

Instruction cache ops use MSR[DR], not MSR[IR], to determine translation of their operands.

When data translation is disabled, cacheability for the effective address of the operand of instruction cache ops is determined by the ICCR, not the DCCR.

This instruction is specific to the PowerPC Embedded Controller family

iccci

Instruction Cache Congruence Class Invalidate

Exceptions

Instruction Storage Exceptions and Instruction-side TLB Miss Exceptions are associated with the fetching of instructions, not with the execution of instructions. Exceptions that occur during the execution of instruction cache ops cause data-side exceptions (Data Storage Exceptions and Data TLB Miss Exceptions).

The execution of an **iccci** instruction can cause a Data TLB Miss Exception, at the specified effective address, in spite of the non-specific intent of that effective address.

This instruction is considered a "load" with respect to Data Storage exceptions. See Section 9.4.3 (Access Protection for Cache Instructions) on page 9-20 for further discussion.

This instruction is considered a "load" with respect to data address compare (DAC) debug exceptions. See Section 10.6.3.1 (DAC Applied to Cache Instructions) on page 10-12 for further discussion.

Architecture Note

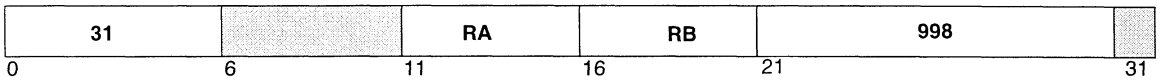
This instruction is specific to the PowerPC Embedded Controller family; it is not described in *PowerPC Architecture*. Programs using this instruction may not be portable to other PowerPC implementations.

This instruction is specific to the PowerPC Embedded Controller family

icread

Instruction Cache Read

icread RA,RB



EA ← (RAI0) + (RB)
if ((CDBCR{CIS] = 0) ∧ (CDBCR[CSS] = 0)) then (ICDBDR) ← (i-cache data, side A)
if ((CDBCR{CIS] = 0) ∧ (CDBCR[CSS] = 1)) then (ICDBDR) ← (i-cache data, side B)
if ((CDBCR{CIS] = 1) ∧ (CDBCR[CSS] = 0)) then (ICDBDR) ← (i-cache tag, side A)
if ((CDBCR{CIS] = 1) ∧ (CDBCR[CSS] = 1)) then (ICDBDR) ← (i-cache tag, side B)

An effective address is formed by adding an index to a base address. The index is the contents of register RB. The base address is 0 if the RA field is 0 and is the contents of register RA otherwise.

This instruction is a debugging tool for reading the instruction cache entries for the congruence class specified by $EA_{22:27}$. The cache information will be read into the Instruction Cache Debug Data Register (ICDBDR), from where it can be read into a GPR via **mficdbdr**.

If (CDBCR[CIS] = 0), the information will be one word of instruction cache data from the addressed congruence class. The word is specified by $EA_{28:29}$ ($EA_{0:21}$ and $EA_{30:31}$ are ignored). If (CDBCR[CSS] = 0), the data will be from the A-side, otherwise from the B-side.

If (CDBCR[CIS] = 1), the information will be a cache tag from the addressed congruence class ($EA_{0:21}$ and $EA_{28:31}$ are ignored). If (CDBCR[CSS] = 0), the tag will be from the A-side, otherwise from the B-side. Instruction cache tag information is placed in the ICDBDR as follows:

0:21	TAG	Cache Tag
22:26		reserved
27	V	Cache Line Valid 0 - Not valid 1 - Valid
28:30		reserved
31	LRU	Least Recently Used 0 - A side least-recently-used 1 - B side least-recently-used

If instruction bit 31 contains 1, the contents of CR[CR0] are undefined.

Registers Altered

- ICDBDR

This instruction is specific to the PowerPC Embedded Controller family

icread

Instruction Cache Read

Invalid Instruction Forms

- Reserved fields

Programming Note

Execution of this instruction is privileged.

The instruction pipeline will not automatically wait for data from **icread** to arrive at ICDBDR before attempting to use the contents of that register. Therefore, insert at least one instruction between **icread** and **mficdbdr**:

```
icread r5,r6          # read cache information
nop                   # minimum separation
mficdbdr r7           # move information to GPR
```

Instruction cache ops use MSR[DR], not MSR[IR], to determine translation of their operands.

When data translation is disabled, cacheability for the effective address of the operand of instruction cache ops is determined by the ICCR, not the DCCR.

Exceptions

Instruction Storage Exceptions and Instruction-side TLB Miss Exceptions are associated with the fetching of instructions, not with the execution of instructions. Exceptions that occur during the execution of instruction cache ops cause data-side exceptions (Data Storage Exceptions and Data TLB Miss Exceptions).

The execution of an **iccci** instruction can cause a Data TLB Miss Exception, at the specified effective address, in spite of the non-specific intent of that effective address.

This instruction is considered a "load" with respect to Data Storage exceptions. See Section 9.4.3 (Access Protection for Cache Instructions) on page 9-20 for further discussion.

This instruction is considered a "load" with respect to data address compare (DAC) debug exceptions. See Section 10.6.3.1 (DAC Applied to Cache Instructions) on page 10-12 for further discussion.

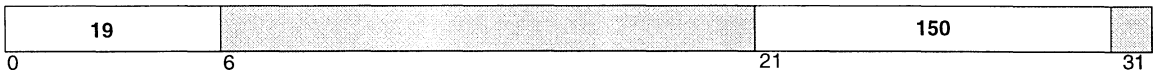
Architecture Note

This instruction is specific to the PowerPC Embedded Controller family; it is not described in *PowerPC Architecture*. Programs using this instruction may not be portable to other PowerPC implementations.

isync

Instruction Synchronize

isync



The **isync** instruction is a context synchronizing instruction.

The **isync** instruction provides an ordering function for the effects of all instructions executed by the processor. Executing **isync** insures that all instructions preceding the **isync** instruction have completed before the **isync** instruction completes, except that storage accesses caused by those instructions need not have completed. No subsequent instructions are initiated by the processor until after the **isync** instruction completes. Finally, execution of **isync** 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 caches.

If instruction bit 31 contains 1, the contents of CR[CR0] are undefined.

Registers Altered

- None

Invalid Instruction Forms

- Reserved fields

Programming Note

See the discussion of context synchronizing instructions in Section 2.10.1 on page 2-37.

The following code example illustrates the necessary steps for self-modifying code. This example assumes that addr1 is both data and instruction cacheable.

```
stw     regN, addr1     # the data in regN is to become an instruction at addr1
dcbst   addr1           # forces data from the data cache to memory
sync                    # wait until the data actually reaches the memory
icbi    addr1           # the previous value at addr1 might already be in
                          the instruction cache; invalidate in the cache
isync                   # the previous value at addr1 might already have been
                          pre-fetched into the queue; invalidate the queue
                          so that the instruction will be re-fetched
```

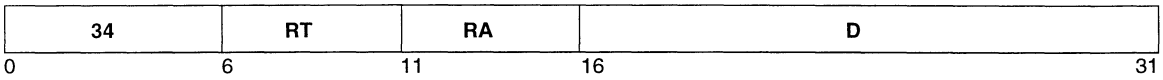
Architecture Note

This instruction is part of the PowerPC Virtual Environment Architecture.

lbz

Load Byte and Zero

lbz RT,D(RA)



EA ← (RAI0) + EXTS(D)
(RT) ← $^{24}0$ II MS(EA,1)

An effective address is formed by adding a displacement to a base address. The displacement is obtained by sign-extending the 16-bit D field to 32 bits. The base address is 0 if the RA field is 0 and is the contents of register RA otherwise.

The byte at the effective address is extended to 32 bits by concatenating 24 0-bits to its left. The result is placed into register RT.

Registers Altered

- RT

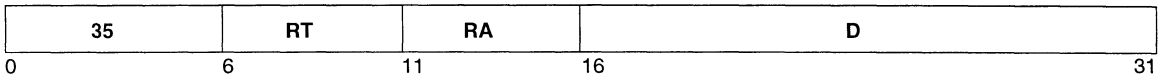
Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

lbzu

Load Byte and Zero with Update

lbzu RT,D(RA)



EA ← (RA|0) + EXTS(D)
(RA) ← EA
(RT) ← $^{24}0$ || MS(EA,1)

An effective address is formed by adding a displacement to a base address. The displacement is obtained by sign-extending the 16-bit D field to 32 bits. The base address is 0 if the RA field is 0 and is the contents of register RA otherwise. The effective address is placed into register RA.

The byte at the effective address is extended to 32 bits by concatenating 24 0-bits to its left. The result is placed into register RT.

Registers Altered

- RA
- RT

Invalid Instruction Forms

- RA=RT
- RA=0

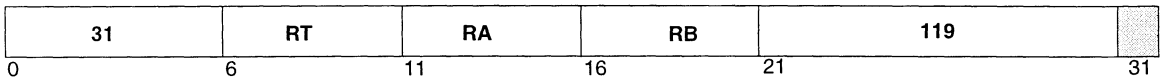
Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

lbzux

Load Byte and Zero with Update Indexed

lbzux RT,RA,RB



EA ← (RAI0) + (RB)
(RA) ← EA
(RT) ← $^{24}0$ II MS(EA,1)

An effective address is formed by adding an index to a base address. The index is the contents of register RB. The base address is 0 if the RA field is 0 and is the contents of register RA otherwise. The effective address is placed into register RA.

The byte at the effective address is extended to 32 bits by concatenating 24 0-bits to its left. The result is placed into register RT.

If instruction bit 31 contains 1, the contents of CR[CR0] are undefined.

Registers Altered

- RA
- RT

Invalid Instruction Forms

- Reserved fields
- RA=RT
- RA=0

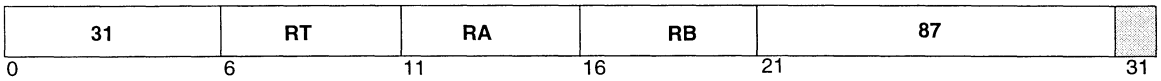
Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

lbzx

Load Byte and Zero Indexed

lbzx RT,RA,RB



EA ← (RAI0) + (RB)
(RT) ← $^{24}0$ II MS(EA,1)

An effective address is formed by adding an index to a base address. The index is the contents of register RB. The base address is 0 if the RA field is 0 and is the contents of register RA otherwise.

The byte at the effective address is extended to 32 bits by concatenating 24 0-bits to its left. The result is placed into register RT.

If instruction bit 31 contains 1, the contents of CR[CR0] are undefined.

Registers Altered

- RT

Invalid Instruction Forms

- Reserved fields

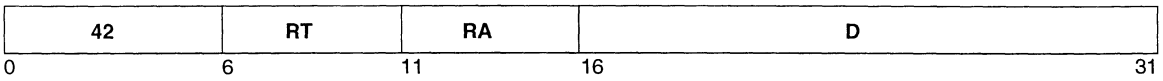
Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

lha

Load Halfword Algebraic

lha RT,D(RA)



EA ← (RAI0) + EXTS(D)
(RT) ← EXTS(MS(EA,2))

An effective address is formed by adding a displacement to a base address. The displacement is obtained by sign-extending the 16-bit D field to 32 bits. The base address is 0 if the RA field is 0 and is the contents of register RA otherwise. The EA must be halfword-aligned (a multiple of 2). If it is not, it will cause an alignment exception.

The halfword at the effective address is sign-extended to 32 bits and placed into register RT.

Registers Altered

- RT

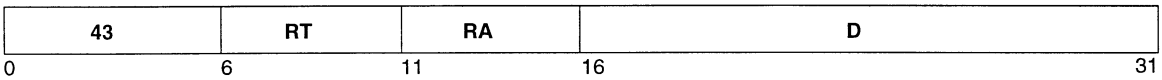
Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

lhau

Load Halfword Algebraic with Update

lhau RT,D(RA)



EA ← (RA|0) + EXTS(D)
(RA) ← EA
(RT) ← EXTS(MS(EA,2))

An effective address is formed by adding a displacement to a base address. The displacement is obtained by sign-extending the 16-bit D field to 32 bits. The base address is 0 when the RA field is 0 and is the contents of register RA otherwise. The EA must be halfword-aligned (a multiple of 2). If it is not, it will cause an alignment exception. The effective address is placed into register RA.

The halfword at the effective address is sign-extended to 32 bits and placed into register RT.

Registers Altered

- RA
- RT

Invalid Instruction Forms

- RA=RT
- RA=0

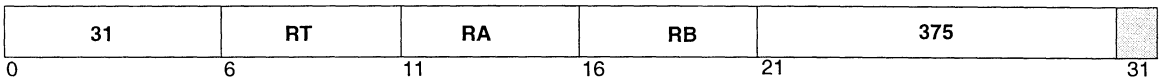
Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

lhaux

Load Halfword Algebraic with Update Indexed

lhaux RT,RA,RB



EA ← (RAI0) + (RB)
(RA) ← EA
(RT) ← EXTS(MS(EA,2))

An effective address is formed by adding an index to a base address. The index is the contents of register RB. The base address is 0 if the RA field is 0 and is the contents of register RA otherwise. The EA must be halfword-aligned (a multiple of 2). If it is not, it will cause an alignment exception. The effective address is placed into register RA.

The halfword at the effective address is sign-extended to 32 bits and placed into register RT.

If instruction bit 31 contains 1, the contents of CR[CR0] are undefined.

Registers Altered

- RA
- RT

Invalid Instruction Forms

- Reserved fields
- RA=RT
- RA=0

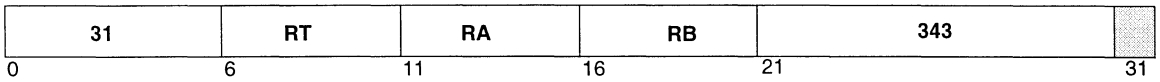
Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

lhax

Load Halfword Algebraic Indexed

lhax RT,RA,RB



EA ← (RAI0) + (RB)
(RT) ← EXTS(MS(EA,2))

An effective address is formed by adding an index to a base address. The index is the contents of register RB. The base address is 0 if the RA field is 0 and is the contents of register RA otherwise. The EA must be halfword-aligned (a multiple of 2). If it is not, it will cause an alignment exception.

The halfword at the effective address is sign-extended to 32 bits and placed into register RT.

If instruction bit 31 contains 1, the contents of CR[CR0] are undefined.

Registers Altered

- RT

Invalid Instruction Forms

- Reserved fields

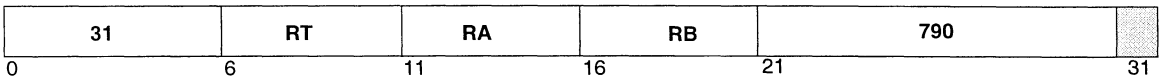
Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

lhbrx

Load Halfword Byte-Reverse Indexed

lhbrx RT,RA,RB



EA ← (RAI0) + (RB)
(RT) ← $^{16}0$ II MS(EA +1,1) II MS(EA,1)

An effective address is formed by adding an index to a base address. The index is the contents of register RB. The base address is 0 if the RA field is 0 and is the contents of register RA otherwise. The EA must be halfword-aligned (a multiple of 2). If it is not, it will cause an alignment exception.

The halfword at the effective address is byte-reversed. The resulting halfword is extended to 32 bits by concatenating 16 0-bits to its left. The result is placed into register RT.

If instruction bit 31 contains 1, the contents of CR[CR0] are undefined.

Registers Altered

- RT

Invalid Instruction Forms

- Reserved fields

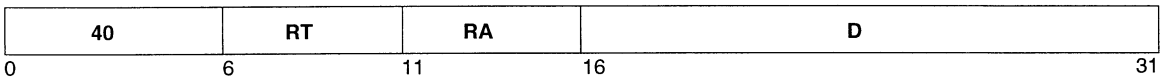
Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

lhz

Load Halfword and Zero

lhz RT,D(RA)



EA ← (RA|0) + EXTS(D)
(RT) ← $^{16}0$ || MS(EA,2)

An effective address is formed by adding a displacement to a base address. The displacement is obtained by sign-extending the 16-bit D field to 32 bits. The base address is 0 if the RA field is 0 and is the contents of register RA otherwise. The EA must be halfword-aligned (a multiple of 2). If it is not, it will cause an alignment exception.

The halfword at the effective address is extended to 32 bits by concatenating 16 0-bits to its left. The result is placed into register RT.

Registers Altered

- RT

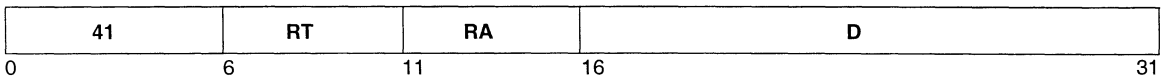
Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

lhzu

Load Halfword and Zero with Update

lhzu RT,D(RA)



EA ← (RA|0) + EXTS(D)
(RA) ← EA
(RT) ← $^{16}0$ || MS(EA,2)

An effective address is formed by adding a displacement to a base address. The displacement is obtained by sign-extending the 16-bit D field to 32 bits. The base address is 0 if the RA field is 0 and is the contents of register RA otherwise. The EA must be halfword-aligned (a multiple of 2). If it is not, it will cause an alignment exception. The effective address is placed into register RA.

The halfword at the effective address is extended to 32 bits by concatenating 16 0-bits to its left. The result is placed into register RT.

Registers Altered

- RA
- RT

Invalid Instruction Forms

- RA=RT
- RA=0

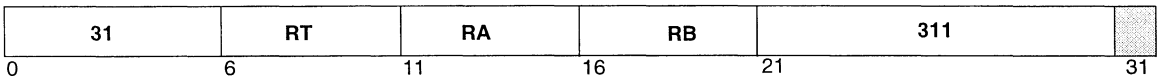
Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

lhzux

Load Halfword and Zero with Update Indexed

lhzux RT,RA,RB



EA ← (RAI0) + (RB)
(RA) ← EA
(RT) ← $^{16}0$ || MS(EA,2)

An effective address is formed by adding an index to a base address. The index is the contents of register RB. The base address is 0 if the RA field is 0 and is the contents of register RA otherwise. The EA must be halfword-aligned (a multiple of 2). If it is not, it will cause an alignment exception. The effective address is placed into register RA.

The halfword at the effective address is extended to 32 bits by concatenating 16 0-bits to its left. The result is placed into register RT.

If instruction bit 31 contains 1, the contents of CR[CR0] are undefined.

Registers Altered

- RA
- RT

Invalid Instruction Forms

- Reserved fields
- RA=RT
- RA=0

11

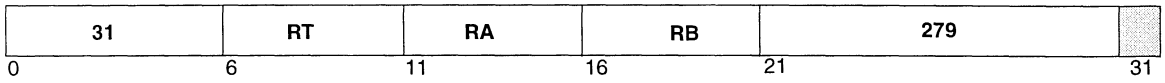
Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

lhzx

Load Halfword and Zero Indexed

lhzx RT,RA,RB



EA ← (RAI0) + (RB)
(RT) ← $^{16}0$ II MS(EA,2)

An effective address is formed by adding an index to a base address. The index is the contents of register RB. The base address is 0 if the RA field is 0 and is the contents of register RA otherwise. The EA must be halfword-aligned (a multiple of 2). If it is not, it will cause an alignment exception.

The halfword at the effective address is extended to 32 bits by concatenating 16 0-bits to its left. The result is placed into register RT.

If instruction bit 31 contains 1, the contents of CR[CR0] are undefined.

Registers Altered

- RT

Invalid Instruction Forms

- Reserved fields

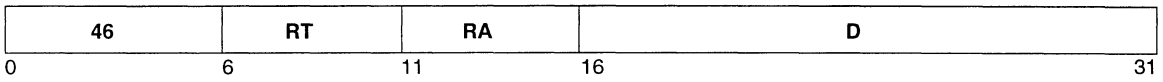
Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

lmw

Load Multiple Word

lmw RT,D(RA)




```
EA ← (RAI0) + EXTS(D)
r ← RT
do while r ≤ 31
   if ((r ≠ RA) ∨ (r = 31)) then
      (GPR(r)) ← MS(EA,4)
   r ← r + 1
   EA ← EA + 4
```

An effective address is formed by adding a displacement to a base address. The displacement is obtained by sign-extending the 16-bit D field in the instruction to 32 bits. The base address is 0 if the RA field is 0 and is the contents of register RA otherwise. The EA must be word-aligned (a multiple of 4). If it is not, it will cause an alignment exception.

A series of consecutive words starting at the effective address are loaded into a set of consecutive GPRs, starting with register RT and continuing to and including GPR(31). Register RA is not altered by this instruction (unless RA is GPR(31), which is an invalid form of this instruction). The word which would have been placed into register RA is discarded.

Registers Altered

- RT through GPR(31).

Invalid Instruction Forms

- RA is in the range of registers to be loaded, including the case where RA = RT = 0.

11

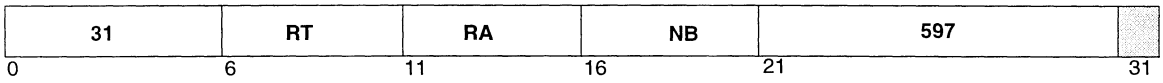
Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

lswi

Load String Word Immediate

lswi RT,RA,NB




```
EA ← (RAI0)
if NB = 0 then
   CNT ← 32
else
   CNT ← NB
n ← CNT
R_FINAL ← ((RT + CEIL(CNT/4) – 1) % 32)
r ← RT – 1
i ← 0
do while n > 0
   if i = 0 then
      r ← r + 1
      if r = 32 then
         r ← 0
      if ((r ≠ RA) ∨ (r = R_FINAL)) then
         (GPR(r)) ← 0
   if ((r ≠ RA) ∨ (r = R_FINAL)) then
      (GPR(r)_i:i+7) ← MS(EA,1)
   i ← i + 8
   if i = 32 then
      i ← 0
   EA ← EA + 1
   n ← n – 1
```

An effective address is determined by the RA field. If the RA field contains 0, the effective address is 0. Otherwise, the effective address is the contents of register RA.

A byte count CNT is determined by examining the NB field. If the NB field is 0, the byte count is CNT = 32. Otherwise, the byte count is CNT = NB.

A series of CNT consecutive bytes in main storage, starting with the byte at the effective address, are loaded into CEIL(CNT/4) consecutive GPRs, four bytes per GPR, until the byte count is exhausted. Bytes are placed into GPRs with the byte having the lowest address loaded into the most significant byte. Bit positions to the right of the last byte loaded in the last GPR used are set to 0.

The set of consecutive registers loaded starts at register RT, continues through GPR(31) and wraps to register 0, loading until the byte count is exhausted, which occurs in register R_{FINAL}. Register RA is not altered (unless RA = R_{FINAL}, which is an invalid form of this instruction). Bytes which would have been loaded into register RA are discarded.

If instruction bit 31 contains 1, the contents of CR[CR0] are undefined.

lswi

Load String Word Immediate

Registers Altered

- RT and subsequent GPRs as described above.

Invalid Instruction Forms

- Reserved fields
- RA is in the range of registers to be loaded
- RA = RT = 0

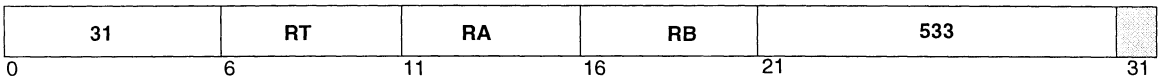
Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

lswx

Load String Word Indexed

lswx RT,RA,RB




```
EA ← (RAI0) + (RB)
CNT ←XER[TBC]
n ← CNT
R_FINAL ← ((RT + CEIL(CNT/4) – 1) % 32)
r ← RT – 1
i ← 0
do while n > 0
   if i = 0 then
      r ← r + 1
      if r = 32 then
         r ← 0
      if (((r ≠ RA) ∧ (r ≠ RB)) ∨ (r = R_FINAL)) then
         (GPR(r)) ← 0
   if (((r ≠ RA) ∧ (r ≠ RB)) ∨ (r = R_FINAL)) then
      (GPR(r)_i:i+7) ← MS(EA,1)
   i ← i + 8
   if i = 32 then
      i ← 0
   EA ← EA + 1
   n ← n – 1
```

An effective address is formed by adding an index to a base address. The index is the contents of register RB. The base address is 0 if the RA field is 0 and is the contents of register RA otherwise.

A byte count CNT is obtained from XER[TBC].

A series of CNT consecutive bytes in main storage, starting with the byte at the effective address, are loaded into CEIL(CNT/4) consecutive GPRs, four bytes per GPR, until the byte count is exhausted. Bytes are placed into GPRs with the byte having the lowest address loaded into the most significant byte. Bit positions to the right of the last byte loaded in the last register used are set to 0.

The set of consecutive GPRs loaded starts at register RT, continues through GPR(31), and wraps to register 0, loading until the byte count is exhausted, which occurs in register R_{FINAL}. Register RA is not altered (unless RA = R_{FINAL}, which is an invalid form of this instruction). Register RB is not altered (unless RB = R_{FINAL}, which is an invalid form of this instruction). Bytes which would have been loaded into registers RA or RB are discarded.

If XER[TBC] is 0, the byte count is 0 and the contents of register RT are undefined.

If instruction bit 31 contains 1, the contents of CR[CR0] are undefined.

lswx

Load String Word Indexed

Registers Altered

- RT and subsequent GPRs as described above.

Invalid Instruction Forms

- Reserved fields
- RA or RB is in the range of registers to be loaded.
- RA = RT = 0

Programming Note

If XER[TBC] is 0 the contents of register RT are undefined.

The PowerPC Architecture states that, if XER[TBC] = 0 and if accessing the EA would otherwise cause a precise data exception (if not for the zero length), then **lswx** will be treated as a no-op and the exception will not occur. Data Storage Exceptions and Data TLB Miss Exceptions are examples of precise data exceptions.

However, the architecture makes no statement regarding imprecise exceptions related to **lswx** with XER[TBC] = 0. The PPC403GC will generate an imprecise exception (Machine Check) on this instruction under these circumstances:

The instruction passes all protection checking; and
the address is cacheable; and
the address misses in the D-cache (resulting in a line fill request to the BIU); and
the address encounters some form of memory subsystem error (non-configured, etc).

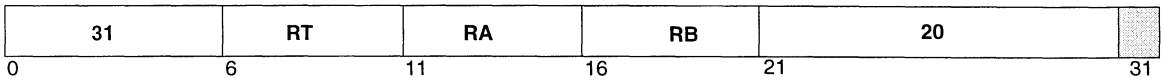
Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

lwarx

Load Word and Reserve Indexed

lwarx RT,RA,RB



EA ← (RAI0) + (RB)
RESERVE ← 1
(RT) ← MS(EA,4)

An effective address is formed by adding an index to a base address. The index is the contents of register RB. The base address is 0 if the RA field is 0 and is the contents of register RA otherwise. The EA must be word-aligned (a multiple of 4). If it is not, it will cause an alignment exception.

The word at the effective address is placed into register RT.

If instruction bit 31 contains 1, the contents of CR[CR0] are undefined.

Execution of the **lwarx** instruction sets the reservation bit.

Registers Altered

- RT

Invalid Instruction Forms

- Reserved fields

Programming Note

The reservation bit can be set to 1 only by the execution of the **lwarx** instruction. When execution of the **stwcx.** instruction completes, the reservation bit will be 0, independent of whether or not the **stwcx.** instruction sent (RS) to memory. $CR[CR0]_{EQ}$ must be examined to determine if (RS) was sent to memory. It is intended that **lwarx** and **stwcx.** be used in pairs in a loop, to create the effect of an atomic operation to a memory area which is a semaphore between asynchronous processes.

```
loop:   lwarx          # read the semaphore from memory; set reservation
        "alter"        # change the semaphore bits in register as required
        stwcx.         # attempt to store semaphore; reset reservation
        bne loop       # an asynchronous process has intervened; try again
```

All usage of **lwarx** and **stwcx.** (including usage within asynchronous processes) should be paired as shown in this example. If the asynchronous process in this example had paired **lwarx** with any store other than **stwcx.** then the reservation bit would not have been cleared in the asynchronous process, and the code above would have overwritten the semaphore.

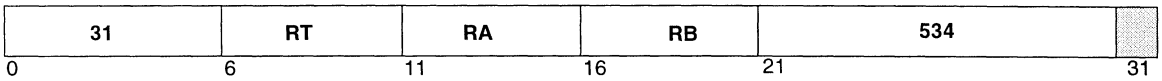
Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

lwbrx

Load Word Byte-Reverse Indexed

lwbrx RT,RA,RB



EA ← (RAI0) + (RB)
(RT) ← MS(EA+3,1) || MS(EA+2,1) || MS(EA+1,1) || MS(EA,1)

An effective address is formed by adding an index to a base address. The index is the contents of register RB. The base address is 0 if the RA field is 0 and is the contents of register RA otherwise. The EA must be word-aligned (a multiple of 4). If it is not, it will cause an alignment exception.

The word at the effective address is byte-reversed: the least significant byte becomes the most significant byte, the next least significant byte becomes the next most significant byte, and so on. The resulting word is placed into register RT.

If instruction bit 31 contains 1, the contents of CR[CR0] are undefined.

Registers Altered

- RT

Invalid Instruction Forms

- Reserved fields

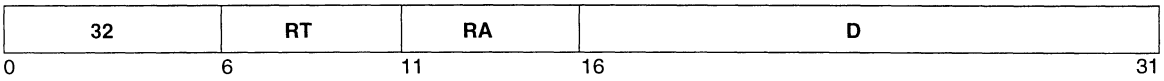
Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

lwz

Load Word and Zero

lwz RT,D(RA)



EA ← (RA|0) + EXTS(D)
(RT) ← MS(EA,4)

An effective address is formed by adding a displacement to a base address. The displacement is obtained by sign-extending the 16-bit D field to 32 bits. The base address is 0 if the RA field is 0 and is the contents of register RA otherwise. The EA must be word-aligned (a multiple of 4). If it is not, it will cause an alignment exception.

The word at the effective address is placed into register RT.

Registers Altered

- RT

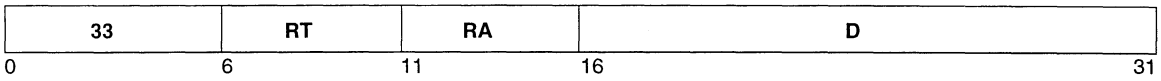
Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

lwzu

Load Word and Zero with Update

lwzu RT,D(RA)



EA ← (RAI0) + EXTS(D)
(RA) ← EA
(RT) ← MS(EA,4)

An effective address is formed by adding a displacement to a base address. The displacement is obtained by sign-extending the 16-bit D field to 32 bits. The base address is 0 if the RA field is 0 and is the contents of register RA otherwise. The EA must be word-aligned (a multiple of 4). If it is not, it will cause an alignment exception. The effective address is placed into register RA.

The word at the effective address is placed into register RT.

Registers Altered

- RA
- RT

Invalid Instruction Forms

- RA=RT
- RA=0

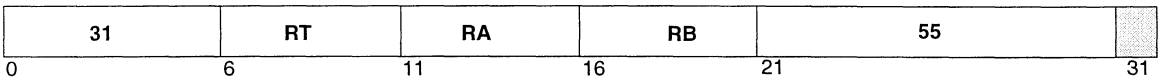
Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

lwzux

Load Word and Zero with Update Indexed

lwzux RT,RA,RB



EA ← (RAI0) + (RB)
(RA) ← EA
(RT) ← MS(EA,4)

An effective address is formed by adding an index to a base address. The index is the contents of register RB. The base address is 0 if the RA field is 0 and is the contents of register RA otherwise. The EA must be word-aligned (a multiple of 4). If it is not, it will cause an alignment exception. The effective address is placed into register RA.

The word at the effective address is placed into register RT.

If instruction bit 31 contains 1, the contents of CR[CR0] are undefined.

Registers Altered

- RA
- RT

Invalid Instruction Forms

- Reserved fields
- RA=RT
- RA=0

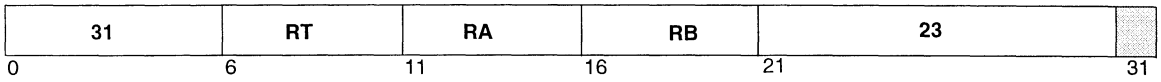
Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

lwzx

Load Word and Zero Indexed

lwzx RT,RA,RB



EA ← (RAI0) + (RB)
(RT) ← MS(EA,4)

An effective address is formed by adding an index to a base address. The index is the contents of register RB. The base address is 0 if the RA field is 0 and is the contents of register RA otherwise. The EA must be word-aligned (a multiple of 4). If it is not, it will cause an alignment exception.

The word at the effective address is placed into register RT.

If instruction bit 31 contains 1, the contents of CR[CR0] are undefined.

Registers Altered

- RT

Invalid Instruction Forms

- Reserved fields

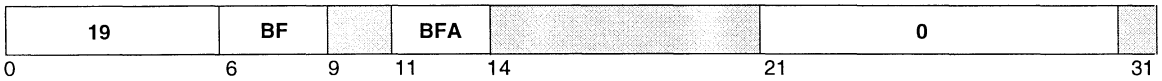
Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

mcrf

Move Condition Register Field

mcrf BF,BFA



m ← BFA
n ← BF
(CR[CRn]) ← (CR[CRm])

The contents of the CR field specified by the BFA field are placed into the CR field specified by the BF field.

Registers Altered

- CR[CR*n*] where *n* is specified by the BF field.

Invalid Instruction Forms

- Reserved fields

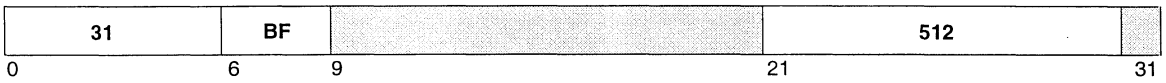
Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

mcrxr

Move to Condition Register from XER

mcrxr BF



n ← BF
(CR[CRn]) ← $XER_{0:3}$
$XER_{0:3}$ ← $^{4}0$

The contents of $XER_{0:3}$ are placed into the CR field specified by the BF field. $XER_{0:3}$ are then set to 0.

This transfer is positional, by bit number, so the mnemonics associated with each bit are changed. See the following table for clarification.

Bit	XER Usage	CR Usage
0	SO	LT
1	OV	GT
2	CA	EQ
3	reserved	SO

If instruction bit 31 contains 1, the contents of CR[CR0] are undefined.

Registers Altered

- CR[CR*n*] where *n* is specified by the BF field.
- XER[SO, OV, CA]

11

Invalid Instruction Forms

- Reserved fields

Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

mfcr

Move From Condition Register

mfcr RT

(RT) ← (CR)

The contents of the CR are placed into register RT.

If instruction bit 31 contains 1, the contents of CR[CR0] are undefined.

Registers Altered

- RT

Invalid Instruction Forms

- Reserved fields

Architecture Note

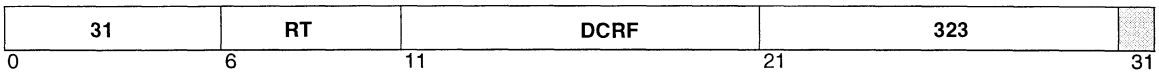
This instruction is part of the PowerPC User Instruction Set Architecture.

This instruction is specific to the PowerPC Embedded Controller family

mfdcr

Move from Device Control Register

mfdcr RT,DCRN



DCRN ← $DCRF_{5:9}$ || $DCRF_{0:4}$
(RT) ← (DCR(DCRN))

The contents of the DCR specified by the DCRF field are placed into register RT. See Table 12-3 (PPC403GC Device Control Registers) on page 12-4 for a listing of DCR mnemonics and corresponding DCRN and DCRF values.

If instruction bit 31 contains 1, the contents of CR[CR0] are undefined.

Registers Altered

- RT

Invalid Instruction Forms

- Reserved fields
- Invalid DCRF values

Programming Note

Execution of this instruction is privileged.

The DCR number (DCRN) specified in the assembler language coding of the **mfdcr** instruction refers to an actual DCR number (see Table 12-3 on page 12-4). The assembler handles the unusual DCR number encoding to generate the DCRF field.

11

Architecture Note

This instruction is specific to the PowerPC Embedded Controller family; it is not described in *PowerPC Architecture*. Programs using this instruction may not be portable to other PowerPC implementations.

mfdcr

Move from Device Control Register

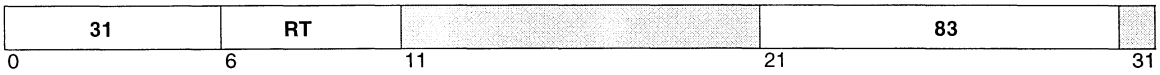
Table 11-17. Extended Mnemonics for mfdcr

Mnemonic	Operands	Function	Other Registers Changed
mfbear **mfbesr** **mfbr0** **mfbr1** **mfbr2** **mfbr3** **mfbr4** **mfbr5** **mfbr6** **mfbr7** **mfdmacc0** **mfdmacc1** **mfdmacc2** **mfdmacc3** **mfdmacr0** **mfdmacr1** **mfdmacr2** **mfdmacr3** **mfdmact0** **mfdmact1** **mfdmact2** **mfdmact3** **mfdmada0** **mfdmada1** **mfdmada2** **mfdmada3** **mfdmasa0** **mfdmasa1** **mfdmasa2** **mfdmasa3** **mfdmasr** **mfexisr** **mfexier** **mfiocr**	RT	Move from device control register DCRN. *Extended mnemonic* for **mfdcr RT,DCRN** See Table 12-3 on page 12-4 for listing of valid DCRN values.	

mfmsr

Move From Machine State Register

mfmsr RT



(RT) ← (MSR)

The contents of the MSR are placed into register RT.

If instruction bit 31 contains 1, the contents of CR[CR0] are undefined.

Registers Altered

- RT

Invalid Instruction Forms

- Reserved fields

Programming Note

Execution of this instruction is privileged.

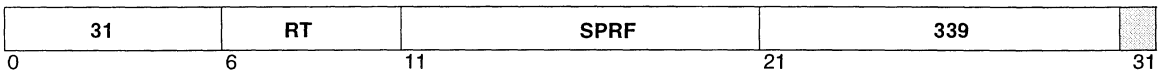
Architecture Note

This instruction is part of the PowerPC Operating Environment Architecture.

mfspr

Move From Special Purpose Register

mfspr RT,SPRN



SPRN ← $SPRF_{5:9}$ || $SPRF_{0:4}$
(RT) ← (SPR(SPRN))

The contents of the SPR specified by the SPRF field are placed into register RT. See Table 12-2 (PPC403GC Special Purpose Registers) on page 12-2 for a listing of SPR mnemonics and corresponding SPRN and SPRF values.

If instruction bit 31 contains 1, the contents of CR[CR0] are undefined.

Registers Altered

- RT

Invalid Instruction Forms

- Reserved fields
- Invalid SPRF values

Programming Note

Execution of this instruction is privileged if instruction bit 11 is '1'. See Section 2.9.4 (Privileged SPRs) on page 2-36 for further discussion.

The SPR number (SPRN) specified in the assembler language coding of the **mfspr** instruction refers to an actual SPR number (see Table 12-2 on page 12-2). The assembler handles the unusual SPR number encoding to generate the SPRF field. Also, see Section 2.9.4 for a discussion of the encoding of Privileged SPRs.

11

Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

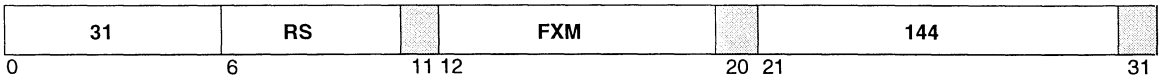
mfspr

Move From Special Purpose Register

Table 11-18. Extended Mnemonics for mfspr

Mnemonic	Operands	Function	Other Registers Changed
mfcdbcr **mfctr** **mfdac1** **mfdac2** **mfdbsr** **mfdccr** **mfdcwr** **mfdear** **mfesr** **mfevpr** **mfiac1** **mfiac2** **mficcr** **mficdbdr** **mflr** **mfpbl1** **mfpbl2** **mfpbu1** **mfpbu2** **mfpid** **mfpit** **mfpvr** **mfsgr** **mfsprg0** **mfsprg1** **mfsprg2** **mfsprg3** **mfsrr0** **mfsrr1** **mfsrr2** **mfsrr3** **mftbhi** **mftbhu** **mftblo** **mftblu** **mftcr** **mftsr** **mfxer** **mfzpr**	RT	Move from special purpose register SPRN. *Extended mnemonic* for **mfspr RT,SPRN** See Table 12-2 on page 12-2 for listing of valid SPRN values.	

mtcrf FXM,RS



mask ← $^4(FXM_0)$ || $^4(FXM_1)$ || ... || $^4(FXM_6)$ || $^4(FXM_7)$
(CR) ← ((RS) ∧ mask) ∨ ((CR) ∧ ¬mask)

Some or all of the contents of register RS are placed into the CR as specified by the FXM field.

Each bit in the FXM field controls the copying of 4 bits in register RS into the corresponding bits in the CR. The correspondence between the bits in the FXM field and the bit copying operation is shown in the following table:

FXM Bit Number	Bits Controlled
0	0:3
1	4:7
2	8:11
3	12:15
4	16:19
5	20:23
6	24:27
7	28:31

If instruction bit 31 contains 1, the contents of CR[CR0] are undefined.

11

Registers Altered

- CR

Invalid Instruction Forms

- Reserved fields

Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

mtcrf

Move to Condition Register Fields

Table 11-19. Extended Mnemonics for mtcrf

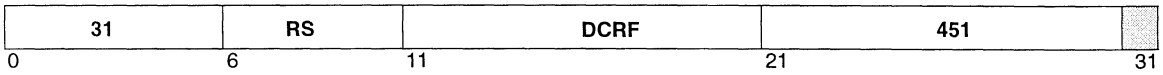
Mnemonic	Operands	Function	Other Registers Changed
mtcr	RS	Move to Condition Register. *Extended mnemonic* for **mtcrf 0xFF,RS**	

This instruction is specific to the PowerPC Embedded Controller family

mtdcr

Move To Device Control Register

mtdcr DCRN,RS



DCRN ← $DCRF_{5:9}$ || $DCRF_{0:4}$
(DCR(DCRN)) ← (RS)

The contents of register RS are placed into the DCR specified by the DCRF field. See Table 12-3 (PPC403GC Device Control Registers) on page 12-4 for a listing of DCR mnemonics and corresponding DCRN and DCRF values.

If instruction bit 31 contains 1, the contents of CR[CR0] are undefined.

Registers Altered

- DCR(DCRN)

Invalid Instruction Forms

- Reserved fields
- Invalid DCRF values

Programming Note

Execution of this instruction is privileged.

The DCR number (DCRN) specified in the assembler language coding of the **mtdcr** instruction refers to an actual DCR number (see Table 12-3 on page 12-4). The assembler handles the unusual DCR number encoding to generate the DCRF field.

Architecture Note

This instruction is specific to the PowerPC Embedded Controller family; it is not described in *PowerPC Architecture*. Programs using this instruction may not be portable to other PowerPC implementations.

This instruction is specific to the PowerPC Embedded Controller family

mtdcr

Move To Device Control Register

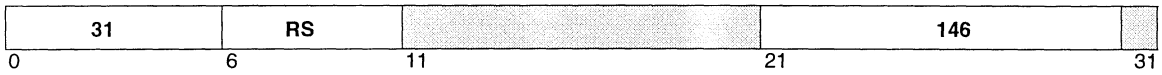
Table 11-20. Extended Mnemonics for mtdcr

Mnemonic	Operands	Function	Other Registers Changed
mtbear **mtbesr** **mtbr0** **mtbr1** **mtbr2** **mtbr3** **mtbr4** **mtbr5** **mtbr6** **mtbr7** **mtdmacc0** **mtdmacc1** **mtdmacc2** **mtdmacc3** **mtdmacr0** **mtdmacr1** **mtdmacr2** **mtdmacr3** **mtdmact0** **mtdmact1** **mtdmact2** **mtdmact3** **mtdmada0** **mtdmada1** **mtdmada2** **mtdmada3** **mtdmasa0** **mtdmasa1** **mtdmasa2** **mtdmasa3** **mtdmasr** **mtexisr** **mtexier** **mtiocr**	RS	Move to device control register DCRN. *Extended mnemonic* for **mtdcr DCRN,RS** See Table 12-3 on page 12-4 for listing of valid DCRN values.	

mtmsr

Move To Machine State Register

mtmsr RS



(MSR) ← (RS)

The contents of register RS are placed into the MSR.

If instruction bit 31 contains 1, the contents of CR[CR0] are undefined.

Registers Altered

- MSR

Invalid Instruction Forms

- Reserved fields

Programming Note

The **mtmsr** instruction is privileged and execution synchronizing.

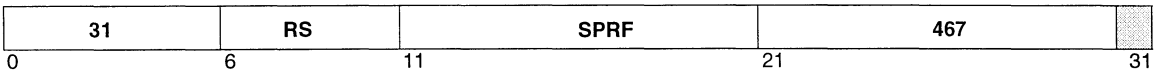
Architecture Note

This instruction is part of the PowerPC Operating Environment Architecture.

mtspr

Move To Special Purpose Register

mtspr SPRN,RS



SPRN ← $SPRF_{5:9}$ || $SPRF_{0:4}$
(SPR(SPRN)) ← (RS)

The contents of register RS are placed into the SPR specified by the SPRF field. See Table 12-2 (PPC403GC Special Purpose Registers) on page 12-2 for a listing of SPR mnemonics and corresponding SPRN and SPRF values.

If instruction bit 31 contains 1, the contents of CR[CR0] are undefined.

Registers Altered

- SPR(SPRN)

Invalid Instruction Forms

- Reserved fields
- Invalid SPRF values

Programming Note

Execution of this instruction is privileged if instruction bit 11 is '1'. See Section 2.9.4 (Privileged SPRs) on page 2-36 for further discussion.

The SPR number (SPRN) specified in the assembler language coding of the **mtspr** instruction refers to an actual SPR number (see Table 12-2 on page 12-2). The assembler handles the unusual SPR number encoding to generate the SPRF field. Also, see Section 2.9.4 for a discussion of the encoding of Privileged SPRs.

11

Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

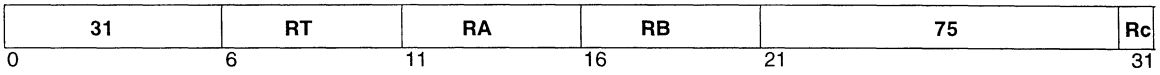
Table 11-21. Extended Mnemonics for mtspr

Mnemonic	Operands	Function	Other Registers Changed
mtcdbcr **mtctr** **mtdac1** **mtdac2** **mtdbsr** **mtdccr** **mtdcwr** **mtesr** **mtevpr** **mtiac1** **mtiac2** **mticcr** **mticdbdr** **mtlr** **mtpbl1** **mtpbl2** **mtpbu1** **mtpbu2** **mtpid** **mtpit** **mtsgr** **mtsprg0** **mtsprg1** **mtsprg2** **mtsprg3** **mtsrr0** **mtsrr1** **mtsrr2** **mtsrr3** **mttbhi** **mttblo** **mttcr** **mttsr** **mtxer** **mtzpr**	RS	Move to special purpose register SPRN. *Extended mnemonic* for **mtspr SPRN,RS** See Table 12-2 on page 12-2 for listing of valid SPRN values.	

mulhw

Multiply High Word

mulhw	RT,RA,RB	(Rc=0)
mulhw.	RT,RA,RB	(Rc=1)



$prod_{0:63} \leftarrow (RA) \times (RB)$ (signed)
$(RT) \leftarrow prod_{0:31}$

The 64-bit signed product of registers RA and RB is formed. The most significant 32 bits of the result is placed into register RT.

Registers Altered

- RT
- $CR[CR0]_{LT, GT, EQ, SO}$ if Rc contains 1

Programming Note

The most significant 32 bits of the product, unlike the least significant 32 bits, may differ depending on whether the registers RA and RB are interpreted as signed or unsigned quantities. The **mulhw** instruction generates the correct result when these operands are interpreted as signed quantities. The **mulhwu** instruction generates the correct result when these operands are interpreted as unsigned quantities.

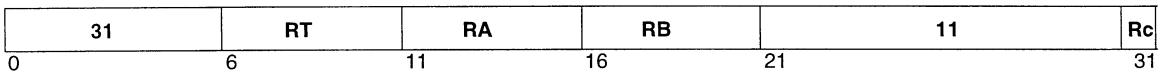
Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

mulhwu

Multiply High Word Unsigned

mulhwu	RT,RA,RB	(Rc=0)
mulhwu.	RT,RA,RB	(Rc=1)



$prod_{0:63} \leftarrow (RA) \times (RB)$ (unsigned)
$(RT) \leftarrow prod_{0:31}$

The 64-bit unsigned product of registers RA and RB is formed. The most significant 32 bits of the result are placed into register RT.

Registers Altered

- RT
- $CR[CR0]_{LT, GT, EQ, SO}$ if Rc contains 1

Programming Note

The most significant 32 bits of the product, unlike the least significant 32 bits, may differ depending on whether the registers RA and RB are interpreted as signed or unsigned quantities. The **mulhw** instruction generates the correct result when these operands are interpreted as signed quantities. The **mulhwu** instruction generates the correct result when these operands are interpreted as unsigned quantities.

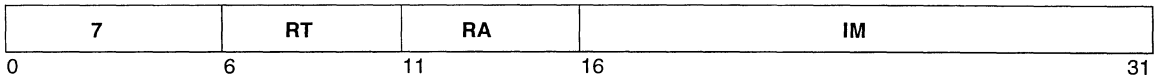
Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

mulli

Multiply Low Immediate

mulli RT,RA,IM



$prod_{0:47} \leftarrow (RA) \times IM$
$(RT) \leftarrow prod_{16:47}$

The 48-bit product of register RA and the IM field is formed. Both register RA and the IM field are interpreted as signed quantities. The least significant 32 bits of the product are placed into register RT.

Registers Altered

- RT

Programming Note

The least significant 16 bits of the product are correct, regardless of whether register RA and field IM are interpreted as signed or unsigned numbers.

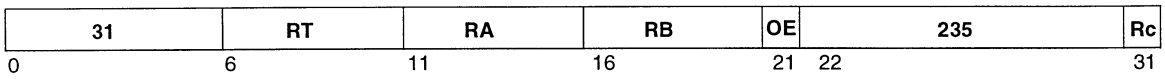
Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

mullw

Multiply Low Word

mullw	RT,RA,RB	(OE=0, Rc=0)
mullw.	RT,RA,RB	(OE=0, Rc=1)
mullwo	RT,RA,RB	(OE=1, Rc=0)
mullwo.	RT,RA,RB	(OE=1, Rc=1)



$prod_{0:63} \leftarrow$ (RA) $\times$ (RB) (signed)
(RT) $\leftarrow prod_{32:63}$

The 64-bit signed product of register RA and register RB is formed. The least significant 32 bits of the result is placed into register RT.

If all bits in positions 0 through 31 of the 64 bit product do not equal bit 0 of the result in register RT and OE=1, XER[SO, OV] are set to 1.

Registers Altered

- RT
- $CR[CR0]_{LT, GT, EQ, SO}$ if Rc contains 1
- XER[SO, OV] if OE=1

Programming Note

The least significant 32 bits of the product are correct, regardless of whether register RA and register RB are interpreted as signed or unsigned numbers.

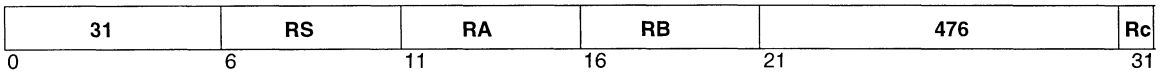
Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

nand

NAND

nand	RA,RS,RB	(Rc=0)
nand.	RA,RS,RB	(Rc=1)



(RA) $\leftarrow$ $\neg$((RS) $\wedge$ (RB))

The contents of register RS is ANDed with the contents of register RB; the ones complement of the result is placed into register RA.

Registers Altered

- RA
- $CR[CR0]_{LT, GT, EQ, SO}$ if Rc contains 1

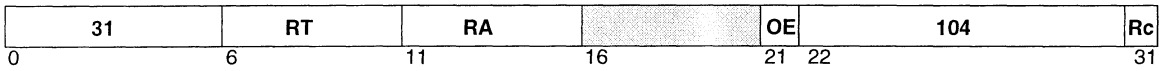
Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

neg

Negate

neg	RT,RA	(OE=0, Rc=0)
neg.	RT,RA	(OE=0, Rc=1)
nego	RT,RA	(OE=1, Rc=0)
nego.	RT,RA	(OE=1, Rc=1)



(RT) ← ¬(RA) + 1

The twos complement of the contents of register RA are placed into register RT.

Registers Altered

- RT
- $CR[CR0]_{LT, GT, EQ, SO}$ if Rc contains 1
- XER[CA, SO, OV] if OE=1

Invalid Instruction Forms

- Reserved fields

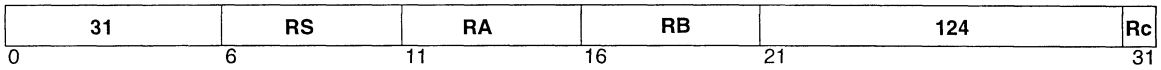
Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

nor

NOR

nor	RA,RS,RB	(Rc=0)
nor.	RA,RS,RB	(Rc=1)



(RA) ← ¬((RS) ∨ (RB))

The contents of register RS is ORed with the contents of register RB; the ones complement of the result is placed into register RA.

Registers Altered

- RA
- $CR[CR0]_{LT, GT, EQ, SO}$ if Rc contains 1

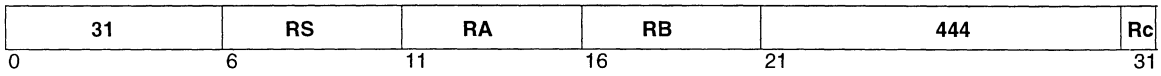
Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

Table 11-22. Extended Mnemonics for nor, nor.

Mnemonic	Operands	Function	Other Registers Changed
not	RA, RS	Complement register. (RA) ← ¬(RS) *Extended mnemonic* for **nor RA,RS,RS**	
not.		*Extended mnemonic* for **nor. RA,RS,RS**	CR[CR0]

or	RA,RS,RB	(Rc=0)
or.	RA,RS,RB	(Rc=1)



(RA) ← (RS) ∨ (RB)

The contents of register RS is ORed with the contents of register RB; the result is placed into register RA.

Registers Altered

- RA
- $CR[CR0]_{LT, GT, EQ, SO}$ if Rc contains 1

Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

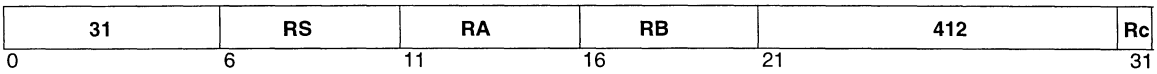
Table 11-23. Extended Mnemonics for or, or.

Mnemonic	Operands	Function	Other Registers Changed
mr	RT, RS	Move register. (RT) ← (RS) *Extended mnemonic* for **or RT,RS,RS**	
mr.		*Extended mnemonic* for **or. RT,RS,RS**	CR[CR0]

orc

OR with Complement

orc	RA,RS,RB	(Rc=0)
orc.	RA,RS,RB	(Rc=1)



(RA) ← (RS) ∨ ¬(RB)

The contents of register RS is ORed with the ones complement of the contents of register RB; the result is placed into register RA.

Registers Altered

- RA
- $CR[CR0]_{LT, GT, EQ, SO}$ if Rc contains 1

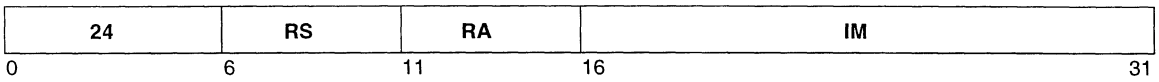
Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

ori

OR Immediate

ori RA,RS,IM



(RA) ← (RS) ∨ (^{16}0 || IM)

The IM field is extended to 32 bits by concatenating 16 0-bits on the left. Register RS is ORed with the extended IM field; the result is placed into register RA.

Registers Altered

- RA

Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

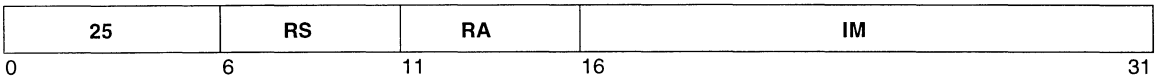
Table 11-24. Extended Mnemonics for ori

Mnemonic	Operands	Function	Other Registers Changed
nop		Preferred no-op, triggers optimizations based on no-ops. *Extended mnemonic* for **ori 0,0,0**	

oris

OR Immediate Shifted

oris RA,RS,IM



(RA) ← (RS) ∨ (IM || $^{16}0$)

The IM Field is extended to 32 bits by concatenating 16 0-bits on the right. Register RS is ORed with the extended IM field and the result is placed into register RA.

Registers Altered

- RA

Architecture Note

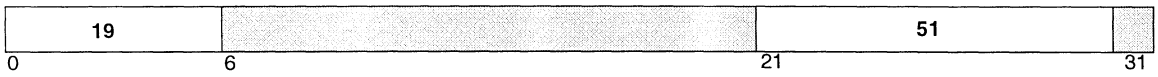
This instruction is part of the PowerPC User Instruction Set Architecture.

This instruction is specific to the PowerPC Embedded Controller family

rfci

Return From Critical Interrupt

rfci



(PC) ← (SRR2)
(MSR) ← (SRR3)

The program counter (PC) is restored with the contents of SRR2 and the MSR is restored with the contents of SRR3.

Instruction execution returns to the address contained in the PC.

Registers Altered

- MSR

Programming Note

Execution of this instruction is privileged.

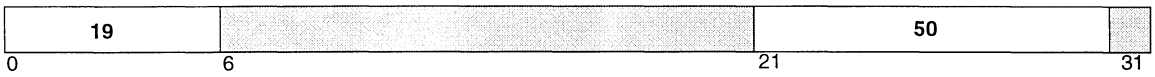
Architecture Note

This instruction is specific to the PowerPC Embedded Controller family; it is not described in *PowerPC Architecture.* Programs using this instruction may not be portable to other PowerPC implementations.

rfi

Return From Interrupt

rfi



(PC) ← (SRR0)
(MSR) ← (SRR1)

The program counter (PC) is restored with the contents of SRR0 and the MSR is restored with the contents of SRR1.

Instruction execution returns to the address contained in the PC.

Registers Altered

- MSR

Invalid Instruction Forms

- Reserved fields

Programming Note

Execution of this instruction is privileged.

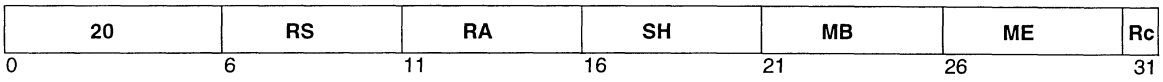
Architecture Note

This instruction is part of the PowerPC Operating Environment Architecture.

rlwimi

Rotate Left Word Immediate then Mask Insert

rlwimi RA,RS,SH,MB,ME (Rc=0)
rlwimi. RA,RS,SH,MB,ME (Rc=1)



r ← ROTL((RS), SH)
m ← MASK(MB, ME)
(RA) ← (r ∧ m) ∨ ((RA) ∧ ¬m)

The contents of register RS are rotated left by the number of bit positions specified in the SH field. A mask is generated, having 1-bits starting at the bit position specified in the MB field and ending in the bit position specified by the ME field, with 0-bits elsewhere.

If the starting point of the mask is at a higher bit position than the ending point, the 1-bits portion of the mask wraps from the highest bit position back around to the lowest. The rotated data is inserted into register RA, in positions corresponding to the bit positions in the mask that contain a 1-bit.

Registers Altered

- RA
- $CR[CR0]_{LT, GT, EQ, SO}$ if Rc contains 1

Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

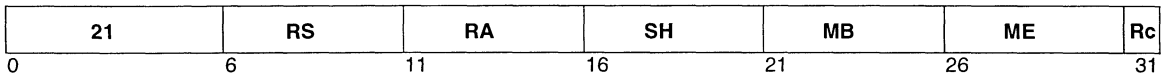
Table 11-25. Extended Mnemonics for rlwimi, rlwimi.

Mnemonic	Operands	Function	Other Registers Changed
inslwi	RA, RS, n, b	Insert from left immediate. (n > 0) $(RA)_{b:b+n-1} \leftarrow (RS)_{0:n-1}$ *Extended mnemonic* for **rlwimi RA,RS,32–b,b,b+n–1**	
inslwi.		*Extended mnemonic* for **rlwimi. RA,RS,32–b,b,b+n–1**	CR[CR0]
insrwi	RA, RS, n, b	Insert from right immediate. (n > 0) $(RA)_{b:b+n-1} \leftarrow (RS)_{32-n:31}$ *Extended mnemonic* for **rlwimi RA,RS,32–b–n,b,b+n–1**	
insrwi.		*Extended mnemonic* for **rlwimi. RA,RS,32–b–n,b,b+n–1**	CR[CR0]

rlwinm

Rotate Left Word Immediate then AND with Mask

rlwinm	RA,RS,SH,MB,ME	(Rc=0)
rlwinm.	RA,RS,SH,MB,ME	(Rc=1)



r ← ROTL((RS), SH)
m ← MASK(MB, ME)
(RA) ← r ∧ m

The contents of register RS is rotated left by the number of bit positions specified in the SH field. A mask is generated, having 1-bits starting at the bit position specified in the MB field and ending in the bit position specified by the ME field with 0-bits elsewhere.

If the starting point of the mask is at a higher bit position than the ending point, the 1-bits portion of the mask wraps from the highest bit position back around to the lowest. The rotated data is ANDed with the generated mask; the result is placed into register RA.

Registers Altered

- RA
- $CR[CR0]_{LT, GT, EQ, SO}$ if Rc contains 1

Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

Table 11-26. Extended Mnemonics for rlwinm, rlwinm.

Mnemonic	Operands	Function	Other Registers Changed
clrlwi	RA, RS, n	Clear left immediate. (n < 32) $(RA)_{0:n-1} \leftarrow {}^{n}0$ *Extended mnemonic* for **rlwinm RA,RS,0,n,31**	
clrlwi.		*Extended mnemonic* for **rlwinm. RA,RS,0,n,31**	CR[CR0]

Table 11-26. Extended Mnemonics for rlwinm, rlwinm. (cont.)

Mnemonic	Operands	Function	Other Registers Changed
clrlslwi	RA, RS, b, n	Clear left and shift left immediate. ($n \le b < 32$) $(RA)_{b-n:31-n} \leftarrow (RS)_{b:31}$ $(RA)_{32-n:31} \leftarrow {}^{n}0$ $(RA)_{0:b-n-1} \leftarrow {}^{b-n}0$ *Extended mnemonic* for **rlwinm RA,RS,n,b–n,31–n**	
clrlslwi.		*Extended mnemonic* for **rlwinm. RA,RS,n,b–n,31–n**	CR[CR0]
clrrwi	RA, RS, n	Clear right immediate. ($n < 32$) $(RA)_{32-n:31} \leftarrow {}^{n}0$ *Extended mnemonic* for **rlwinm RA,RS,0,0,31–n**	
clrrwi.		*Extended mnemonic* for **rlwinm. RA,RS,0,0,31–n**	CR[CR0]
extlwi	RA, RS, n, b	Extract and left justify immediate. ($n > 0$) $(RA)_{0:n-1} \leftarrow (RS)_{b:b+n-1}$ $(RA)_{n:31} \leftarrow {}^{32-n}0$ *Extended mnemonic* for **rlwinm RA,RS,b,0,n–1**	
extlwi.		*Extended mnemonic* for **rlwinm. RA,RS,b,0,n–1**	CR[CR0]
extrwi	RA, RS, n, b	Extract and right justify immediate. ($n > 0$) $(RA)_{32-n:31} \leftarrow (RS)_{b:b+n-1}$ $(RA)_{0:31-n} \leftarrow {}^{32-n}0$ *Extended mnemonic* for **rlwinm RA,RS,b+n,32–n,31**	
extrwi.		*Extended mnemonic* for **rlwinm. RA,RS,b+n,32–n,31**	CR[CR0]
rotlwi	RA, RS, n	Rotate left immediate. (RA) ← ROTL((RS), n) *Extended mnemonic* for **rlwinm RA,RS,n,0,31**	
rotlwi.		*Extended mnemonic* for **rlwinm. RA,RS,n,0,31**	CR[CR0]
rotrwi	RA, RS, n	Rotate right immediate. (RA) ← ROTL((RS), 32–n) *Extended mnemonic* for **rlwinm RA,RS,32–n,0,31**	
rotrwi.		*Extended mnemonic* for **rlwinm. RA,RS,32–n,0,31**	CR[CR0]

rlwinm

Rotate Left Word Immediate then AND with Mask

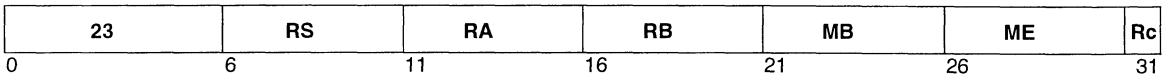
Table 11-26. Extended Mnemonics for rlwinm, rlwinm. (cont.)

Mnemonic	Operands	Function	Other Registers Changed
slwi	RA, RS, n	Shift left immediate. (n < 32) $(RA)_{0:31-n} \leftarrow (RS)_{n:31}$ $(RA)_{32-n:31} \leftarrow {}^{n}0$ *Extended mnemonic* for **rlwinm RA,RS,n,0,31–n**	
slwi.		*Extended mnemonic* for **rlwinm. RA,RS,n,0,31–n**	CR[CR0]
srwi	RA, RS, n	Shift right immediate. (n < 32) $(RA)_{n:31} \leftarrow (RS)_{0:31-n}$ $(RA)_{0:n-1} \leftarrow {}^{n}0$ *Extended mnemonic* for **rlwinm RA,RS,32–n,n,31**	
srwi.		*Extended mnemonic* for **rlwinm. RA,RS,32–n,n,31**	CR[CR0]

rlwnm

Rotate Left Word then AND with Mask

rlwnm RA,RS,RB,MB,ME (Rc=0)
rlwnm. RA,RS,RB,MB,ME (Rc=1)



r ← ROTL((RS), $(RB)_{27:31}$)
m ← MASK(MB, ME)
(RA) ← r ∧ m

The contents of register RS is rotated left by the number of bit positions specified by the contents of register RB bits 27 through 31. A mask is generated having 1-bits starting at the bit position specified in the MB field and ending in the bit position specified by the ME field with 0-bits elsewhere.

If the starting point of the mask is at a higher bit position than the ending point, the ones portion of the mask wraps from the highest bit position back around to the lowest. The rotated data is ANDed with the generated mask and the result is placed into register RA.

Registers Altered

- RA
- $CR[CR0]_{LT, GT, EQ, SO}$ if Rc contains 1

Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

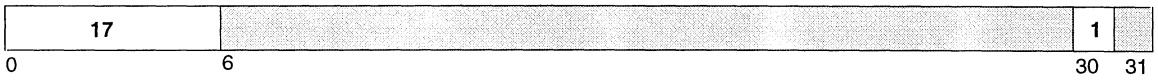
Table 11-27. Extended Mnemonics for rlwnm, rlwnm.

Mnemonic	Operands	Function	Other Registers Changed
rotlw	RA, RS, RB	Rotate left. (RA) ← ROTL((RS), $(RB)_{27:31}$) *Extended mnemonic* for **rlwnm RA,RS,RB,0,31**	
rotlw.		*Extended mnemonic* for **rlwnm. RA,RS,RB,0,31**	CR[CR0]

sc

System Call

sc



(SRR1) ← (MSR)
(SRR0) ← (PC)
PC ← $EVPR_{0:15}$ || x'0C00'
(MSR[WE, EE, PR, PE, DR, IR]) ← 0
(MSR[LE]) ← (MSR[ILE])

A system call exception is generated. The contents of the MSR are copied into SRR1 and (4 + address of **sc** instruction) is placed into SRR0.

The PC is then loaded with the exception vector address. The exception vector address is calculated by concatenating the high halfword of the Exception Vector Prefix Register (EVPR) to the left of 0x0C00.

The MSR[WE, PR, EE, PE, DR, IR] bits are set to 0, and MSR[ILE] is copied to MSR[LE].

Program execution continues at the new address in the PC.

The **sc** instruction is context synchronizing.

Registers Altered

- SRR0
- SRR1
- MSR[WE, EE, PR, PE, DR, IR, LE]

Invalid Instruction Forms

- Reserved fields

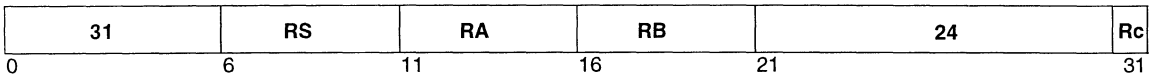
Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

slw

Shift Left Word

slw	RA,RS,RB	(Rc=0)
slw.	RA,RS,RB	(Rc=1)



$n \leftarrow (RB)_{27:31}$
$r \leftarrow ROTL((RS), n)$
if $(RB)_{26} = 0$ then
  $m \leftarrow MASK(0, 31 - n)$
else
  $m \leftarrow {}^{32}0$
$(RA) \leftarrow r \wedge m$

The contents of register RS are shifted left by the number of bits specified by bits 27 through 31 of register RB. Bits shifted left out of the most significant bit are lost, and 0-bits are supplied to fill vacated bit positions on the right. The result is placed into register RA.

If bit 26 of register RB contains a one, register RA is set to zero.

Registers Altered

- RA
- $CR[CR0]_{LT, GT, EQ, SO}$ if Rc contains 1

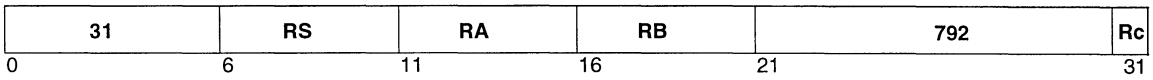
Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

sraw

Shift Right Algebraic Word

sraw	RA,RS,RB	(Rc=0)
sraw.	RA,RS,RB	(Rc=1)



$n \leftarrow (RB)_{27:31}$
$r \leftarrow ROTL((RS), 32 - n)$
if $(RB)_{26} = 0$ then
  $m \leftarrow MASK(n, 31)$
else
  $m \leftarrow {}^{32}0$
$s \leftarrow (RS)_0$
$(RA) \leftarrow (r \wedge m) \vee ({}^{32}s \wedge \neg m)$
$XER[CA] \leftarrow s \wedge ((r \wedge \neg m) \neq 0)$

The contents of register RS are shifted right by the number of bits specified by bits 27 through 31 of register RB. Bits shifted out of the least significant bit are lost. Bit 0 of register RS is replicated to fill the vacated positions on the left. The result is placed into register RA.

If register RS contains a negative number and any 1-bits were shifted out of the least significant bit position, XER[CA] is set to 1; otherwise, it is set to 0.

If bit 26 of register RB contains 1, register RA and XER[CA] are set to bit 0 of register RS.

Registers Altered

- RA
- XER[CA]
- $CR[CR0]_{LT, GT, EQ, SO}$ if Rc contains 1

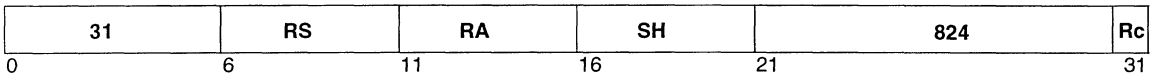
Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

srawi

Shift Right Algebraic Word Immediate

srawi	RA,RS,SH	(Rc=0)
srawi.	RA,RS,SH	(Rc=1)



n ← SH
r ← ROTL((RS), 32 – n)
m ← MASK(n, 31)
s ← $(RS)_0$
(RA) ← (r ∧ m) ∨ (^{32}s ∧ ¬m)
XER[CA] ← s ∧ ((r ∧ ¬m)≠0)

The contents of register RS are shifted right by the number of bits specified in the SH field. Bits shifted out of the least significant bit are lost. Bit 0 of register RS is replicated to fill the vacated positions on the left. The result is placed into register RA.

If register RS contains a negative number and any 1-bits were shifted out of the least significant bit position, XER[CA] is set to 1; otherwise, it is set to 0.

Registers Altered

- RA
- XER[CA]
- $CR[CR0]_{LT, GT, EQ, SO}$ if Rc contains 1

Architecture Note

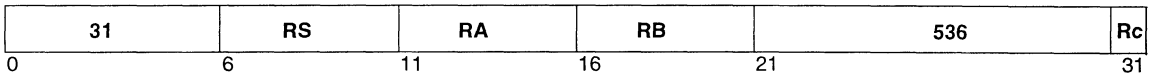
This instruction is part of the PowerPC User Instruction Set Architecture.

11

srw

Shift Right Word

srw	RA,RS,RB	(Rc=0)
srw.	RA,RS,RB	(Rc=1)



$n \leftarrow (RB)_{27:31}$
$r \leftarrow ROTL((RS), 32 - n)$
if $(RB)_{26} = 0$ then
  $m \leftarrow MASK(n, 31)$
else
  $m \leftarrow {}^{32}0$
$(RA) \leftarrow r \wedge m$

The contents of register RS are shifted right by the number of bits specified by bits 27 through 31 of register RB. Bits shifted right out of the least significant bit are lost, and 0-bits are supplied to fill the vacated bit positions on the left. The result is placed into register RA.

If bit 26 of register RB contains a one, register RA is set to 0.

Registers Altered

- RA
- $CR[CR0]_{LT, GT, EQ, SO}$ if Rc contains 1

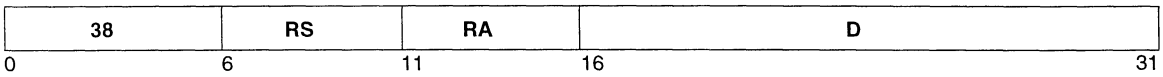
Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

stb

Store Byte

stb RS,D(RA)



EA ← (RAI0) + EXTS(D)
MS(EA, 1) ← $(RS)_{24:31}$

An effective address is formed by adding a displacement to a base address. The displacement is obtained by sign-extending the 16-bit D field to 32 bits. The base address is 0 when the RA field is 0, and is the contents of register RA otherwise.

The least significant byte of register RS is stored into the byte at the effective address in main storage.

Registers Altered

- None

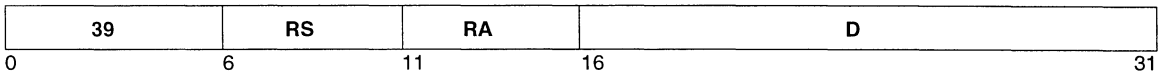
Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

stbu

Store Byte with Update

stbu RS,D(RA)



EA ← (RAI0) + EXTS(D)
MS(EA, 1) ← $(RS)_{24:31}$
(RA) ← EA

An effective address is formed by adding a displacement to a base address. The displacement is obtained by sign-extending the 16-bit D field to 32 bits. The base address is 0 when the RA field is 0, and is the contents of register RA otherwise.

The least significant byte of register RS is stored into the byte at the effective address in main storage.

The effective address is placed into register RA.

Registers Altered

- RA

Invalid Instruction Forms

RA = 0

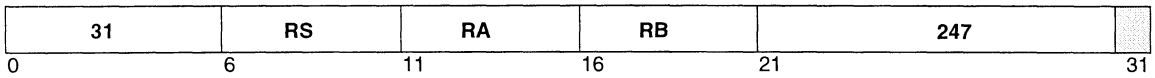
Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

stbux

Store Byte with Update Indexed

stbux RS,RA,RB



EA ← (RAI0) + (RB)
MS(EA, 1) ← $(RS)_{24:31}$
(RA) ← EA

An effective address is formed by adding an index to a base address. The index is the contents of register RB. The base address is 0 when the RA field is 0, and is the contents of register RA otherwise.

The least significant byte of register RS is stored into the byte at the effective address in main storage.

The effective address is placed into register RA.

If instruction bit 31 contains 1, the contents of CR[CR0] are undefined.

Registers Altered

- RA

Invalid Instruction Forms

- Reserved fields
- RA = 0

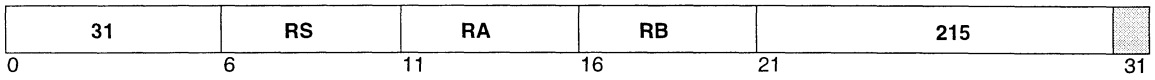
Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

stbx

Store Byte Indexed

stbx RS,RA,RB



EA ← (RAI0) + (RB)
MS(EA, 1) ← $(RS)_{24:31}$

An effective address is formed by adding an index to a base address. The index is the contents of register RB. The base address is 0 when the RA field is 0, and is the contents of register RA otherwise.

The least significant byte of register RS is stored into the byte at the effective address in main storage.

If instruction bit 31 contains 1, the contents of CR[CR0] are undefined.

Registers Altered

- None

Invalid Instruction Forms

- Reserved fields

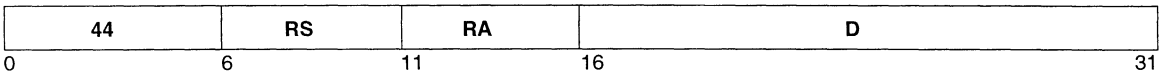
Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

sth

Store Halfword

sth RS,D(RA)



EA ← (RAI0) + EXTS(D)
MS(EA, 2) ← $(RS)_{16:31}$

An effective address is formed by adding a displacement to a base address. The displacement is obtained by sign-extending the 16-bit D field to 32 bits. The base address is 0 when the RA field is 0 and is the contents of register RA otherwise. The EA must be halfword-aligned (a multiple of 2). If it is not, it will cause an alignment exception.

The least significant halfword of register RS is stored into the halfword at the effective address in main storage.

Registers Altered

- None

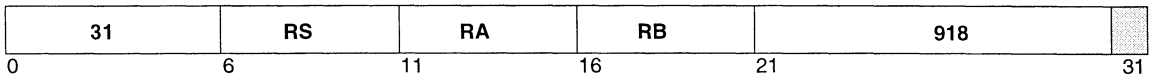
Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

sthbrx

Store Halfword Byte-Reverse Indexed

sthbrx RS,RA,RB



EA ← (RAI0) + (RB)
MS(EA, 2) ← $(RS)_{24:31}$ || $(RS)_{16:23}$

An effective address is formed by adding an index to a base address. The index is the contents of register RB. The base address is 0 when the RA field is 0, and is the contents of register RA otherwise. The EA must be halfword-aligned (a multiple of 2). If it is not, it will cause an alignment exception.

The least significant halfword of register RS is byte-reversed. The result is stored into the halfword at the effective address in main storage.

If instruction bit 31 contains 1, the contents of CR[CR0] are undefined.

Registers Altered

- None

Invalid Instruction Forms

- Reserved fields

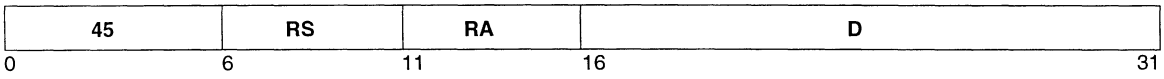
Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

sthu

Store Halfword with Update

sthu RS,D(RA)



EA ← (RAI0) + EXTS(D)
MS(EA, 2) ← $(RS)_{16:31}$
(RA) ← EA

An effective address is formed by adding a displacement to a base address. The displacement is obtained by sign-extending the 16-bit D field to 32 bits. The base address is 0 when the RA field is 0, and is the contents of register RA otherwise. The EA must be halfword-aligned (a multiple of 2). If it is not, it will cause an alignment exception.

The least significant halfword of register RS is stored into the halfword at the effective address in main storage.

The effective address is placed into register RA.

Registers Altered

- RA

Invalid Instruction Forms

- RA = 0

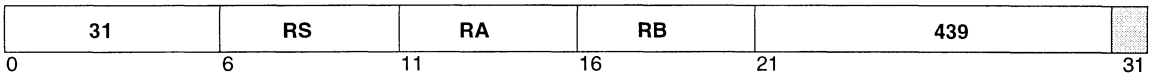
Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

sthux

Store Halfword with Update Indexed

sthux RS,RA,RB



EA ← (RAI0) + (RB)
MS(EA, 2) ← $(RS)_{16:31}$
(RA) ← EA

An effective address is formed by adding an index to a base address. The index is the contents of register RB. The base address is 0 when the RA field is 0, and is the contents of register RA otherwise. The EA must be halfword-aligned (a multiple of 2). If it is not, it will cause an alignment exception.

The least significant halfword of register RS is stored into the halfword at the effective address in main storage.

The effective address is placed into register RA.

If instruction bit 31 contains 1, the contents of CR[CR0] are undefined.

Registers Altered

- RA

Invalid Instruction Forms

- Reserved fields
- RA=0

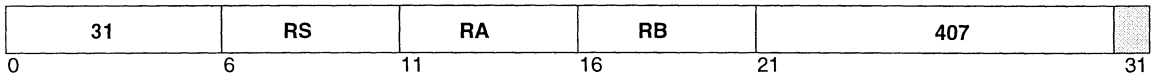
Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

sthx

Store Halfword Indexed

sthx RS,RA,RB



EA ← (RAI0) + (RB)
MS(EA, 2) ← $(RS)_{16:31}$

An effective address is formed by adding an index to a base address. The index is the contents of register RB. The base address is 0 when the RA field is 0, and is the contents of register RA otherwise. The EA must be halfword-aligned (a multiple of 2). If it is not, it will cause an alignment exception.

The least significant halfword of register RS is stored into the halfword at the effective address in main storage.

If instruction bit 31 contains 1, the contents of CR[CR0] are undefined.

Registers Altered

- None

Invalid Instruction Forms

- Reserved fields

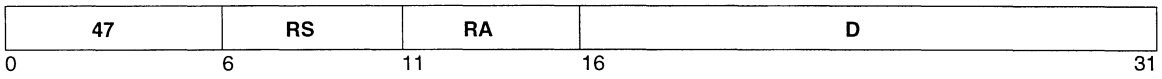
Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

stmw

Store Multiple Word

stmw RS,D(RA)




```
EA ← (RAI0) + EXTS(D)
r ← RS
do while r ≤ 31
   MS(EA, 4) ← (GPR(r))
   r ← r + 1
   EA ← EA + 4
```

An effective address is formed by adding a displacement to a base address. The displacement is obtained by sign-extending the 16-bit D field to 32 bits. The base address is 0 when the RA field is 0, and is the contents of register RA otherwise. The EA must be word-aligned (a multiple of 4). If it is not, it will cause an alignment exception.

The contents of a series of consecutive registers, starting with register RS and continuing through GPR(31), are stored into consecutive words in main storage starting at the effective address.

Registers Altered

- None

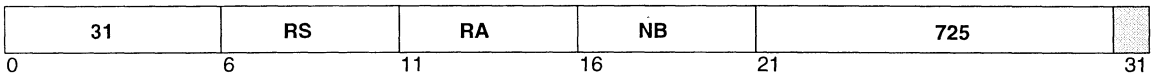
Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

stswi

Store String Word Immediate

stswi RS,RA,NB




```
EA ← (RAI0)
if NB = 0 then
   n ← 32
else
   n ← NB
r ← RS – 1
i ← 0
do while n > 0
   if i = 0 then
      r ← r + 1
   if r = 32 then
      r ← 0
   MS(EA,1) ← (GPR(r)i:i+7)
   i ← i + 8
   if i = 32 then
      i ← 0
   EA ← EA + 1
   n ← n – 1
```

An effective address is determined by the RA field. If the RA field contains 0, the effective address is 0; otherwise, the effective address is the contents of register RA.

A byte count is determined by the NB field. If the NB field contains 0, the byte count is 32; otherwise, the byte count is the NB field.

The contents of a series of consecutive GPRs (starting with register RS, continuing through GPR(31), wrapping to GPR(0), and continuing to the final byte count) are stored into main storage starting at the effective address. The bytes in each GPR are accessed starting with the most significant byte. The byte count determines the number of transferred bytes.

If instruction bit 31 contains 1, the contents of CR[CR0] are undefined.

Registers Altered

- None

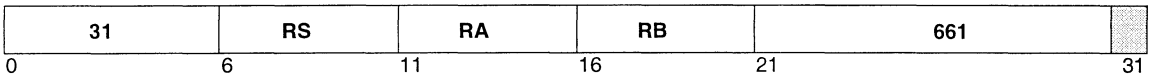
Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

stswx

Store String Word Indexed

stswx RS,RA,RB




```
EA ← (RAI0) + (RB)
n ← XER[TBC]
r ← RS – 1
i ← 0
do while n > 0
   if i = 0 then
      r ← r + 1
   if r = 32 then
      r ← 0
   MS(EA, 1) ← (GPR(r)i:i+7)
   i ← i + 8
   if i = 32 then
      i ← 0
   EA ← EA + 1
   n ← n – 1
```

An effective address is formed by adding an index to a base address. The index is the contents of register RB. The base address is 0 when the RA field is 0, and is the contents of register RA otherwise.

A byte count is contained in XER[TBC].

The contents of a series of consecutive GPRs (starting with register RS, continuing through GPR(31), wrapping to GPR(0), and continuing to the final byte count) are stored starting at the effective address. The bytes in each GPR are accessed starting with the most significant byte. The byte count determines the number of transferred bytes.

If instruction bit 31 contains 1, the contents of CR[CR0] are undefined.

Registers Altered

- None

Invalid Instruction Forms

- Reserved fields

Programming Note

If XER[TBC] contains 0, the **stswx** instruction transfers no bytes; the instruction will be treated as a no-op.

The PowerPC Architecture states that, if XER[TBC] = 0 and if accessing the EA would otherwise cause a precise data exception (if not for the zero length), then **stswx** will be treated as a no-op and the exception will not occur. Data Storage Exceptions and Data TLB Miss Exceptions are examples of precise data exceptions.

However, the architecture makes no statement regarding imprecise exceptions related to **stswx** with XER[TBC] = 0. The PPC403GC will generate an imprecise exception (Machine Check) on this instruction under these circumstances:

The instruction passes all protection checking; and
the address is cacheable; and
the address misses in the D-cache (resulting in a line fill request to the BIU); and
the address encounters some form of memory subsystem error (non-configured, etc).

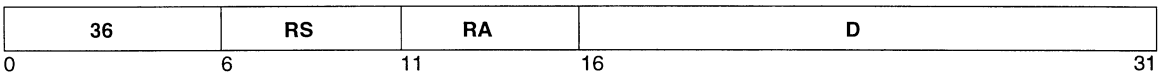
Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

stw

Store Word

stw RS,D(RA)



EA ← (RAI0) + EXTS(D)
MS(EA, 4) ← (RS)

An effective address is formed by adding a displacement to a base address. The displacement is obtained by sign-extending the 16-bit D field to 32 bits. The base address is 0 when the RA field is 0, and is the contents of register RA otherwise. The EA must be word-aligned (a multiple of 4). If it is not, it will cause an alignment exception.

The contents of register RS are stored at the effective address.

Registers Altered

- None

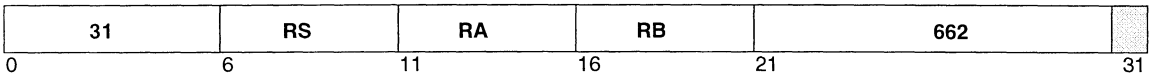
Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

stwbrx

Store Word Byte-Reverse Indexed

stwbrx RS,RA,RB



EA ← (RAI0) + (RB)
MS(EA, 4) ← $(RS)_{24:31}$ || $(RS)_{16:23}$ || $(RS)_{8:15}$ || $(RS)_{0:7}$

An effective address is formed by adding an index to a base address. The index is the contents of register RB. The base address is 0 when the RA field is 0, and is the contents of register RA otherwise. The EA must be word-aligned (a multiple of 4). If it is not, it will cause an alignment exception.

The contents of register RS are byte-reversed: the least significant byte becomes the most significant byte, the next least significant byte becomes the next most significant byte, and so on. The result is stored into word at the effective address.

If instruction bit 31 contains 1, the contents of CR[CR0] are undefined.

Registers Altered

- None

Invalid Instruction Forms

- Reserved fields

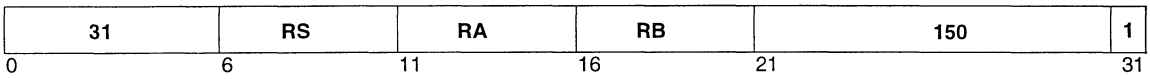
Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

stwcx.

Store Word Conditional Indexed

stwcx. RS,RA,RB




```
EA ← (RA|0) + (RB)
if RESERVE = 1 then
   MS(EA, 4) ← (RS)
   RESERVE ← 0
   (CR[CR0]) ← ²0 || 1 || XER_SO
else
   (CR[CR0]) ← ²0 || 0 || XER_SO
```

An effective address is formed by adding an index to a base address. The index is the contents of register RB. The base address is 0 when the RA field is 0, and is the contents of register RA otherwise. The EA must be word-aligned (a multiple of 4). If it is not, it will cause an alignment exception.

If the reservation bit contains 1 when the instruction is executed, the contents of register RS are stored into the word at the effective address and the reservation bit is cleared. If the reservation bit contains 0 when the instruction is executed, no store operation is performed.

CR[CR0] is set as follows:

- $CR[CR0]_{LT, GT}$ are cleared
- $CR[CR0]_{EQ}$ is set to the state of the reservation bit at the start of the instruction
- $CR[CR0]_{SO}$ is set to the contents of the XER[SO] bit.

Programming Note

The reservation bit can be set to 1 only by the execution of the **lwarx** instruction. When execution of the **stwcx.** instruction completes, the reservation bit will be 0, independent of whether or not the **stwcx.** instruction sent (RS) to memory. $CR[CR0]_{EQ}$ must be examined to determine if (RS) was sent to memory. It is intended that **lwarx** and **stwcx.** be used in pairs in a loop, to create the effect of an atomic operation to a memory area which is a semaphore between asynchronous processes.

```
loop:   lwarx          # read the semaphore from memory; set reservation
        "alter"        # change the semaphore bits in register as required
        stwcx.         # attempt to store semaphore; reset reservation
        bne loop       # an asynchronous process has intervened; try again
```

All usage of **lwarx** and **stwcx.** (including usage within asynchronous processes) should be paired as shown in this example. If the asynchronous process in this example had paired **lwarx** with any store other than **stwcx.** then the reservation bit would not have been cleared in the asynchronous process, and the code above would have overwritten the semaphore.

Registers Altered

- $CR[CR0]_{LT, GT, EQ, SO}$

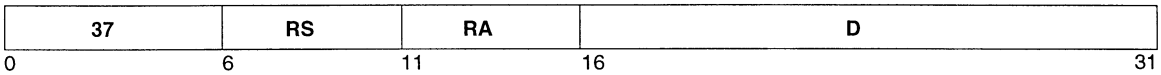
Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

stwu

Store Word with Update

stwu RS,D(RA)



EA ← (RAI0) + EXTS(D)
MS(EA, 4) ← (RS)
(RA) ← EA

An effective address is formed by adding a displacement to a base address. The displacement is obtained by sign-extending the 16-bit D field to 32 bits. The base address is 0 when the RA field is 0, and is the contents of register RA otherwise. The EA must be word-aligned (a multiple of 4). If it is not, it will cause an alignment exception.

The contents of register RS are stored into the word at the effective address.

The effective address is placed into register RA.

Registers Altered

- RA

Invalid Instruction Forms

- RA = 0

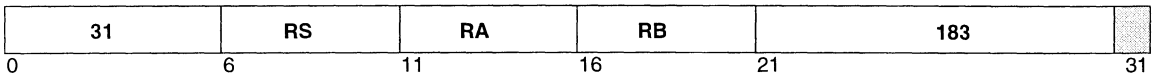
Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

stwux

Store Word with Update Indexed

stwux RS,RA,RB



EA ← (RAI0) + (RB)
MS(EA, 4) ← (RS)
(RA) ← EA

An effective address is formed by adding an index to a base address. The index is the contents of register RB. The base address is 0 when the RA field is 0, and is the contents of register RA otherwise. The EA must be word-aligned (a multiple of 4). If it is not, it will cause an alignment exception.

The contents of register RS are stored into the word at the effective address.

The effective address is placed into register RA.

If instruction bit 31 contains 1, the contents of CR[CR0] are undefined.

Registers Altered

- RA

Invalid Instruction Forms

- Reserved fields
- RA = 0

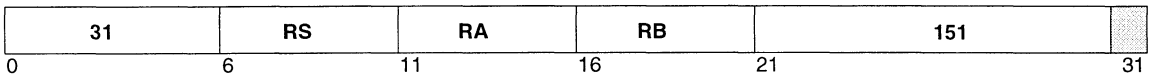
Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

stwx

Store Word Indexed

stwx RS,RA,RB



EA ← (RA|0) + (RB)
MS(EA,4) ← (RS)

An effective address is formed by adding an index to a base address. The index is the contents of register RB. The base address is 0 when the RA field is 0, and is the contents of register RA otherwise. The EA must be word-aligned (a multiple of 4). If it is not, it will cause an alignment exception.

The contents of register RS are stored into the word at the effective address.

If instruction bit 31 contains 1, the contents of CR[CR0] are undefined.

Registers Altered

- None

Invalid Instruction Forms

- Reserved fields

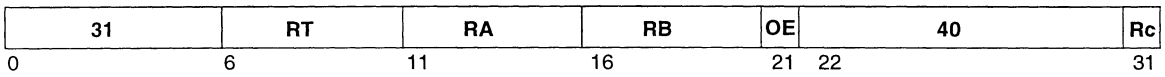
Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

subf

Subtract From

subf	RT,RA,RB	(OE=0, Rc=0)
subf.	RT,RA,RB	(OE=0, Rc=1)
subfo	RT,RA,RB	(OE=1, Rc=0)
subfo.	RT,RA,RB	(OE=1, Rc=1)



(RT) ← ¬(RA) + (RB) + 1

The sum of the ones complement of register RA, register RB, and 1 is stored into register RT.

Registers Altered

- RT
- $CR[CR0]_{LT, GT, EQ, SO}$ if Rc contains 1
- XER[SO, OV] if OE contains 1

Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

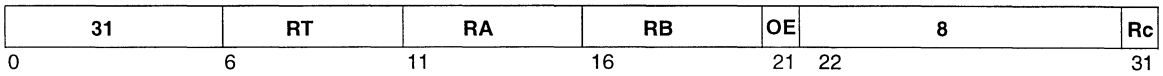
Table 11-28. Extended Mnemonics for subf, subf., subfo, subfo.

Mnemonic	Operands	Function	Other Registers Changed
sub	RT, RA, RB	Subtract (RB) from (RA). (RT) ← ¬(RB) + (RA) + 1. *Extended mnemonic* for **subf RT,RB,RA**	
sub.		*Extended mnemonic* for **subf. RT,RB,RA**	CR[CR0]
subo		*Extended mnemonic* for **subfo RT,RB,RA**	XER[SO, OV]
subo.		*Extended mnemonic* for **subfo. RT,RB,RA**	CR[CR0] XER[SO, OV]

subfc

Subtract From Carrying

subfc	RT,RA,RB	(OE=0, Rc=0)
subfc.	RT,RA,RB	(OE=0, Rc=1)
subfco	RT,RA,RB	(OE=1, Rc=0)
subfco.	RT,RA,RB	(OE=1, Rc=1)



(RT) $\leftarrow$ $\neg$(RA) + (RB) + 1
if $\neg$(RA) + (RB) + 1 $\overset{u}{\geq}$ $2^{32} - 1$ then
  XER[CA] $\leftarrow$ 1
else
  XER[CA] $\leftarrow$ 0

The sum of the ones complement of register RA, register RB, and 1 is stored into register RT.

XER[CA] is set to a value determined by the unsigned magnitude of the result of the subtract operation.

Registers Altered

- RT
- XER[CA]
- $CR[CR0]_{LT, GT, EQ, SO}$ if Rc contains 1
- XER[SO, OV] if OE contains 1

Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

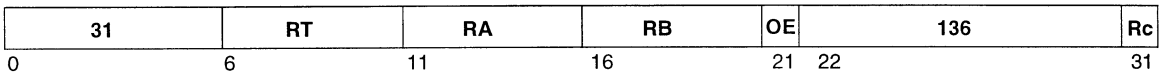
Table 11-29. Extended Mnemonics for subfc, subfc., subfco, subfco.

Mnemonic	Operands	Function	Other Registers Changed
subc	RT, RA, RB	Subtract (RB) from (RA). (RT) ← ¬(RB) + (RA) + 1. Place carry-out in XER[CA]. *Extended mnemonic* for **subfc RT,RB,RA**	
subc.		*Extended mnemonic* for **subfc. RT,RB,RA**	CR[CR0]
subco		*Extended mnemonic* for **subfco RT,RB,RA**	XER[SO, OV]
subco.		*Extended mnemonic* for **subfco. RT,RB,RA**	CR[CR0] XER[SO, OV]

subfe

Subtract From Extended

subfe	RT,RA,RB	(OE=0, Rc=0)
subfe.	RT,RA,RB	(OE=0, Rc=1)
subfeo	RT,RA,RB	(OE=1, Rc=0)
subfeo.	RT,RA,RB	(OE=1, Rc=1)



(RT) $\leftarrow$ $\neg$(RA) + (RB) + XER[CA]
if $\neg$(RA) + (RB) + XER[CA] $\overset{u}{\geq}$ $2^{32} - 1$ then
  XER[CA] $\leftarrow$ 1
else
  XER[CA] $\leftarrow$ 0

The sum of the ones complement of register RA, register RB, and XER[CA] is placed into register RT.

XER[CA] is set to a value determined by the unsigned magnitude of the result of the subtract operation.

Registers Altered

- RT
- XER[CA]
- $CR[CR0]_{LT, GT, EQ, SO}$ if Rc contains 1
- XER[SO, OV] if OE contains 1

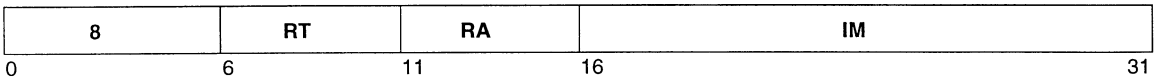
Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

subfic

Subtract From Immediate Carrying

subfic RT,RA,IM



(RT) ← ¬(RA) + EXTS(IM) + 1
if ¬(RA) + EXTS(IM) + 1 $\overset{u}{>}$ $2^{32} - 1$ then
  XER[CA] ← 1
else
  XER[CA] ← 0

The sum of the ones complement of RA, the IM field sign-extended to 32 bits, and 1 is placed into register RT.

XER[CA] is set to a value determined by the unsigned magnitude of the result of the subtract operation.

Registers Altered

- RT
- XER[CA]

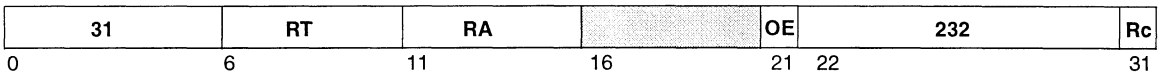
Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

subfme

Subtract from Minus One Extended

subfme	RT,RA	(OE=0, Rc=0)
subfme.	RT,RA	(OE=0, Rc=1)
subfmeo	RT,RA	(OE=1, Rc=0)
subfmeo.	RT,RA	(OE=1, Rc=1)



(RT) ← ¬(RA) – 1 + XER[CA]
if ¬(RA) + 0xFFFF FFFF + XER[CA] $\overset{u}{\geq}$ 2^{32} – 1 then
  XER[CA] ← 1
else
  XER[CA] ← 0

The sum of the ones complement of register RA, –1, and XER[CA] is placed into register RT.

XER[CA] is set to a value determined by the unsigned magnitude of the result of the subtract operation.

Registers Altered

- RT
- $CR[CR0]_{LT, GT, EQ, SO}$ if Rc contains 1
- XER[SO, OV] if OE contains 1
- XER[CA]

Invalid Instruction Forms

- Reserved fields

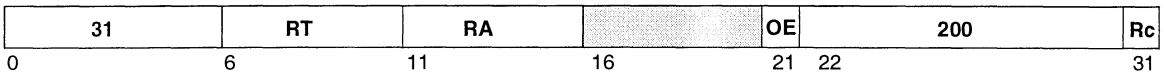
Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

subfze

Subtract from Zero Extended

subfze	RT,RA	(OE=0, Rc=0)
subfze.	RT,RA	(OE=0, Rc=1)
subfzeo	RT,RA	(OE=1, Rc=0)
subfzeo.	RT,RA	(OE=1, Rc=1)



(RT) ← ¬(RA) + XER[CA]
if ¬(RA) + XER[CA] $\overset{u}{>}$ $2^{32} - 1$ then
  XER[CA] ← 1
else
  XER[CA] ← 0

The sum of the ones complement of register RA and XER[CA] is stored into register RT.

XER[CA] is set to a value determined by the unsigned magnitude of the result of the subtract operation.

Registers Altered

- RT
- XER[CA]
- $CR[CR0]_{LT, GT, EQ, SO}$ if Rc contains 1
- XER[SO, OV] if OE contains 1

Invalid Instruction Forms

- Reserved fields

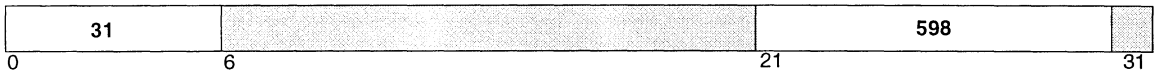
Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

sync

Synchronize

sync



Synchronize System

The **sync** instruction guarantees that all instructions initiated by the processor preceding the **sync** instruction will complete before the **sync** instruction completes, and that no subsequent instructions will be initiated by the processor until after **sync** completes. When **sync** completes, all storage accesses initiated by the processor prior to **sync** will have been completed with respect to all mechanisms that access storage.

If instruction bit 31 contains 1, the contents of CR[CR0] are undefined.

Registers Altered

- None.

Invalid Instruction Forms

- Reserved fields

Programming Note

Architecturally, **eieio** orders storage access, not instruction completion. Therefore, non-storage operations after **eieio** could complete before storage operations that were before **eieio**. The **sync** instruction guarantees ordering of both instruction completion and storage access. For the PPC403GC, the **eieio** instruction is implemented to behave as a **sync** instruction. To write code which is portable between various PowerPC implementations, programmers should use the mnemonic which corresponds to the desired behavior.

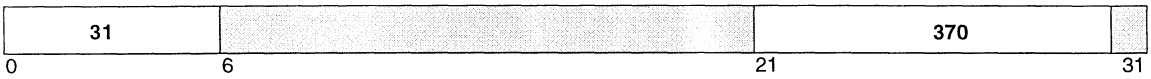
Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

tlbia

TLB Invalidate All

tlbia



All of the entries in the TLB are invalidated and become unavailable for translation by clearing the valid (V) bit in the TLBHI portion of each TLB entry. The rest of the fields in the TLB entries are unmodified.

Registers Altered

- None.

Invalid Instruction Forms

- None.

Programming Note

This instruction is privileged. Translation is not required to be active during the execution of this instruction. The effects of the invalidation are not guaranteed to be visible to the programming model until the completion of a context synchronizing operation.

Architecture Note

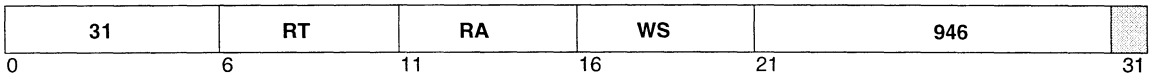
This instruction is part of the PowerPC Operating Environment Architecture.

This instruction is specific to the PowerPC Embedded Controller family

tlbre

TLB Read Entry

tlbre RT, RA, WS



if $WS_4 = 1$
  (RT) ← TLBLO[$(RA)_{26:31}$]
else
  (RT) ← TLBHI[$(RA)_{26:31}$]
  (PID) ← TID from TLB[$(RA)_{26:31}$]

The contents of the selected TLB entry are placed into register RT (and possibly into PID).

Bits 26:31 of the contents of RA are used as an index into the TLB.

The WS field specifies which portion (TLBHI or TLBLO) of the entry is loaded into RT. If TLBHI is being accessed, the PID SPR is set to the value of the TID field in the TLB entry.

If the WS field is not 0 or 1, the instruction form is invalid and the result is undefined.

If $(RA)_{0:25} \neq 0$, the results are undefined.

If instruction bit 31 contains 1, the contents of CR[CR0] are undefined.

Registers Altered

- RT
- PID (if WS = 0)

Invalid Instruction Forms

- Reserved fields
- Invalid WS value

This instruction is specific to the PowerPC Embedded Controller family

tlbre

TLB Read Entry

Programming Notes

This instruction is privileged. Translation is not required to be active during the execution of this instruction.

The contents of RT after the execution of this instruction are interpreted as follows:

If WS = 0 (TLBHI):

RT[0:21] ← EPN[0:21]
RT[22:24] ← SIZE[0:2]
RT[25] ← V
RT[26:31] ← 0
PID[24:31] ← TID[0:7]; (note that the TID is copied to the PID, not to RT)

If WS = 1 (TLBLO):

RT[0:21] ← RPN[0:21]
RT[22:23] ← EX,WR
RT[24:27] ← ZSEL[0:3]
RT[28:31] ← WIMG

Architecture Note

This instruction is specific to the PowerPC Embedded Controller family; it is not described in *PowerPC Architecture*. Programs using this instruction may not be portable to other PowerPC implementations.

Table 11-30. Extended Mnemonics for tlbre

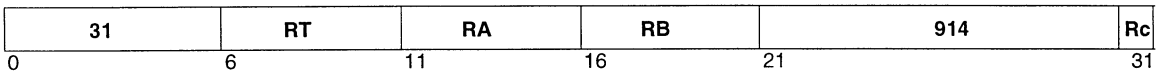
Mnemonic	Operands	Function	Other Registers Changed
tlbrehi	RT, RA	Load TLBHI portion of the selected TLB entry into RT. Load the PID register with the contents of the TID field of the selected TLB entry. (RT) ← TLBHI[(RA)] (PID) ← $TLB[(RA)]_{TID}$ *Extended mnemonic* for **tlbre RT,RA,0**	
tlbrelo	RT, RA	Load TLBLO portion of the selected TLB entry into RT. (RT) ← TLBLO[(RA)] *Extended mnemonic* for **tlbre RT,RA,1**	

This instruction is specific to the PowerPC Embedded Controller family

tlbsx

TLB Search Indexed

tlbsx	RT,RA,RB	(Rc=0)
tlbsx.	RT,RA,RB	(Rc=1)




```
EA ← (RAI0) + (RB)
if Rc = 1
   CR[CR0]LT ← 0
   CR[CR0]GT ← 0
   CR[CR0]SO ← XER[SO}
if  Valid TLB entry matching EA and PID is in the TLB then
   (RT) ← Index of matching TLB Entry
   if Rc = 1
      CR[CR0]EQ ← 1
else
   (RT) Undefined
   if Rc = 1
      CR[CR0]EQ ← 0
```

An effective address is formed by adding an index to a base address. The index is the contents of register RB. The base address is 0 if the RA field is 0 and is the contents of register RA otherwise.

The TLB is searched for a valid entry which translates EA and PID. See Section 9.2.3.1 (Page Identification Fields) on page 9-8 for details. The record bit (Rc) specifies whether the results of the search will affect CR[CR0] as shown above. The intention is that $CR[CR0]_{EQ}$ can be tested after a **tlbsx.** instruction if there is a possibility that the search may fail.

11

Registers Altered

- $CR[CR0]_{LT, GT, EQ, SO}$ if Rc contains 1

Invalid Instruction Forms

- None.

Programming Note

This instruction is privileged. Translation is not required to be active during the execution of this instruction.

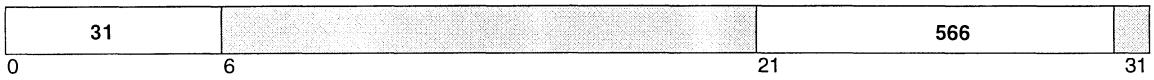
Architecture Note

This instruction is specific to the PowerPC Embedded Controller family; it is not described in *PowerPC Architecture*. Programs using this instruction may not be portable to other PowerPC implementations.

tlbsync

TLB Synchronize

tlbsync



The **tlbsync** instruction is provided in the PowerPC architecture to support synchronization of TLB operations among the processors of a multi-processor system. On PPC403GC, this instruction performs no operation, and is provided to facilitate code portability.

Registers Altered

- None.

Invalid Instruction Forms

- None.

Programming Notes

This instruction is privileged. Translation is not required to be active during the execution of this instruction.

Since PPC403GC does not support tightly-coupled multi-processor systems, **tlbsync** performs no operation.

Architecture Note

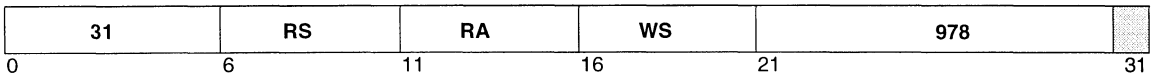
This instruction is part of the PowerPC Operating Environment Architecture.

This instruction is specific to the PowerPC Embedded Controller family

tlbwe

TLB Write Entry

tlbwe RS , RA, WS



if $WS_4 = 1$
  $TLBLO[(RA)_{26:31}] \leftarrow (RS)$
else
  $TLBHI[(RA)_{26:31}] \leftarrow (RS)$
  $TID\ of\ TLB[(RA)_{26:31}] \leftarrow (PID)_{24:31}$

The contents of the selected TLB entry are replaced with the contents of register RS (and possibly PID).

Bits 26:31 of the contents of RA are used as an index into the TLB.

The WS field specifies which portion (TLBHI or TLBLO) of the entry is replaced from RS. For instructions that specify TLBHI, the TID field in the TLB entry is supplied from $PID_{24:31}$.

If the WS field is not 0 or 1, the instruction form is invalid and the result is undefined.

If $(RA)_{0:25} \neq 0$, the results are undefined.

If instruction bit 31 contains 1, the contents of CR[CR0] are undefined.

Registers Altered

- None.

Invalid Instruction Forms

- Reserved fields
- Invalid WS value

This instruction is specific to the PowerPC Embedded Controller family

tlbwe

TLB Write Entry

Programming Notes

This instruction is privileged. Translation is not required to be active during the execution of this instruction.

The effects of the update are not guaranteed to be visible to the programming model until the completion of a context synchronizing operation. For example, updating a zone selection field within the TLB while in supervisor code should be followed by an **isync** instruction (or other context synchronizing operation) to guarantee that the desired translation and protection domains are used.

The TLB fields are written from RS by this instruction as follows:

If WS = 0 (TLBHI):

EPN[0:21] ← RS[0:21]
SIZE[0:2] ← RS[22:24]
V ← RS[25]
TID[0:7] ← PID[24:31]; (note that the TID is written from the PID, not RS)

If WS = 1 (TLBLO):

RPN[0:21] ← RS[0:21]
EX,WR ← RS[22:23]
ZSEL[0:3] ← RS[24:27]
WIMG← RS[28:31]

Architecture Note

This instruction is specific to the PowerPC Embedded Controller family; it is not described in *PowerPC Architecture*. Programs using this instruction may not be portable to other PowerPC implementations.

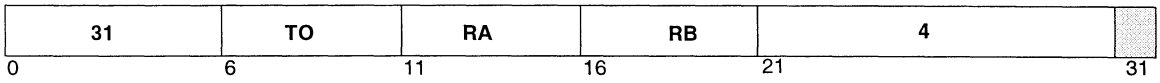
Table 11-31. Extended Mnemonics for tlbwe

Mnemonic	Operands	Function	Other Registers Changed
tlbwehi	RS, RA	Write TLBHI portion of the selected TLB entry from RS. Write the TID field of the selected TLB entry from the PID register. TLBHI[(RA)] ← (RS) $TLB[(RA)]_{TID}$ ← $(PID)_{24:31}$ *Extended mnemonic* for **tlbwe RS,RA,0**	
tlbwelo	RS, RA	Write TLBLO portion of the selected TLB entry from RS. TLBLO[(RA)] ← (RS) *Extended mnemonic* for **tlbwe RS,RA,1**	

tw

Trap Word

tw TO,RA,RB



if (((RA) $<$ (RB) $\wedge$ TO_0 = 1) $\vee$
((RA) $>$ (RB) $\wedge$ TO_1 = 1) $\vee$
((RA) = (RB) $\wedge$ TO_2 = 1) $\vee$
((RA) $\overset{u}{<}$ (RB) $\wedge$ TO_3 = 1) $\vee$
((RA) $\overset{u}{>}$ (RB) $\wedge$ TO_4 = 1)) then TRAP (see details below)

Register RA is compared with register RB. If any comparison condition selected by the TO field is true, a TRAP occurs. The behavior of a TRAP depends upon the Debug Mode of the processor, as described below:

- If TRAP is not enabled as a debug event (DBCR[TDE] = 0 or DBCR[EDM,IDM] = 0,0):

  TRAP will cause a Program interrupt. See Section 6.9 (Program Exceptions) on page 6-32 and Section 6.2.5 (Exception Syndrome Register (ESR)) on page 6-12 for further information.

  (SRR0) ← address of **tw** instruction
  (SRR1) ← (MSR)
  (ESR[PTR]) ← 1
  (MSR[WE, EE, PR, PE, DR, IR]) ← 0
  (MSR[LE]) ← (MSR[ILE])
  PC ← $EVPR_{0:15}$ || x'0700'

- If TRAP is enabled as an External debug event (DBCR[TDE] = 1 and DBCR[EDM] = 1):

  TRAP will go to the Debug Stop state, to be handled by an external debugger with hardware control over the PPC403GC.

  (DBSR[TIE]) ← 1

  In addition, if TRAP is also enabled as an Internal debug event (DBCR[IDM] = 1) and Debug Exceptions are disabled (MSR[DE] = 0), then an imprecise event will be reported by setting (DBSR[IDE]) ← 1

  PC ← address of **tw** instruction

- If TRAP is enabled as an Internal debug event and not an External debug event (DBCR[TDE] = 1 and DBCR[EDM,IDM] = 0,1) and Debug Exceptions are enabled (MSR[DE] = 1):

  TRAP will cause a Debug interrupt. See Section 6.16 (Debug Exception Handling) on page 6-38 for further information.

(SRR2) ← address of **tw** instruction
(SRR3) ← (MSR)
(DBSR[TIE]) ← 1
(MSR[WE, EE, PR, PE, CE, DE, DR, IR]) ← 0
(MSR[LE]) ← (MSR[ILE])
PC ← $EVPR_{0:15}$ || x'2000'

- If TRAP is enabled as an Internal debug event and not an External debug event (DBCR[TDE] = 1 and DBCR[EDM,IDM] = 0,1) and Debug Exceptions are disabled (MSR[DE] = 0):

  TRAP will report the debug event as an imprecise event and will cause a Program interrupt. See Section 6.9 (Program Exceptions) on page 6-32 and Section 6.2.5 (Exception Syndrome Register (ESR)) on page 6-12 for further information.

  (SRR0) ← address of **tw** instruction
  (SRR1) ← (MSR)
  (ESR[PTR]) ← 1
  (DBSR[TIE,IDE]) ← 1,1
  (MSR[WE, EE, PR, PE, DR, IR]) ← 0
  (MSR[LE]) ← (MSR[ILE])
  PC ← $EVPR_{0:15}$ || x'0700'

If instruction bit 31 contains 1, the contents of CR[CR0] are undefined.

Registers Altered

- None

Invalid Instruction Forms

- Reserved fields

Programming Note

This instruction is inserted into the execution stream by a debugger to implement breakpoints, and is not typically used by application code.

Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

tw

Trap Word

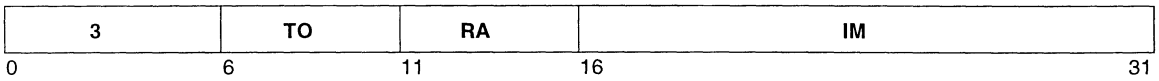
Table 11-32. Extended Mnemonics for tw

Mnemonic	Operands	Function	Other Registers Changed
trap		Trap unconditionally. *Extended mnemonic* for **tw 31,0,0**	
tweq	RA, RB	Trap if (RA) equal to (RB). *Extended mnemonic* for **tw 4,RA,RB**	
twge		Trap if (RA) greater than or equal to (RB). *Extended mnemonic* for **tw 12,RA,RB**	
twgt		Trap if (RA) greater than (RB). *Extended mnemonic* for **tw 8,RA,RB**	
twle		Trap if (RA) less than or equal to (RB). *Extended mnemonic* for **tw 20,RA,RB**	
twlge		Trap if (RA) logically greater than or equal to (RB). *Extended mnemonic* for **tw 5,RA,RB**	
twlgt		Trap if (RA) logically greater than (RB). *Extended mnemonic* for **tw 1,RA,RB**	
twlle		Trap if (RA) logically less than or equal to (RB). *Extended mnemonic* for **tw 6,RA,RB**	
twllt		Trap if (RA) logically less than (RB). *Extended mnemonic* for **tw 2,RA,RB**	
twlng		Trap if (RA) logically not greater than (RB). *Extended mnemonic* for **tw 6,RA,RB**	
twlnl		Trap if (RA) logically not less than (RB). *Extended mnemonic* for **tw 5,RA,RB**	
twlt		Trap if (RA) less than (RB). *Extended mnemonic* for **tw 16,RA,RB**	
twne		Trap if (RA) not equal to (RB). *Extended mnemonic* for **tw 24,RA,RB**	
twng		Trap if (RA) not greater than (RB). *Extended mnemonic* for **tw 20,RA,RB**	
twnl		Trap if (RA) not less than (RB). *Extended mnemonic* for **tw 12,RA,RB**	

twi

Trap Word Immediate

twi TO,RA,IM



if (((RA) < EXTS(IM) ∧ TO_0 = 1) ∨
((RA) > EXTS(IM) ∧ TO_1 = 1) ∨
((RA) = EXTS(IM) ∧ TO_2 = 1) ∨
((RA) <u EXTS(IM) ∧ TO_3 = 1) ∨
((RA) >u EXTS(IM) ∧ TO_4 = 1)) then TRAP (see details below)

Register RA is compared with the IM field, which has been sign-extended to 32 bits. If any comparison condition selected by the TO field is true, a TRAP occurs. The behavior of a TRAP depends upon the Debug Mode of the processor, as described below:

- If TRAP is not enabled as a debug event (DBCR[TDE] = 0 or DBCR[EDM,IDM] = 0,0):

  TRAP will cause a Program interrupt. See Section 6.9 (Program Exceptions) on page 6-32 and Section 6.2.5 (Exception Syndrome Register (ESR)) on page 6-12 for further information.

  (SRR0) ← address of **twi** instruction
  (SRR1) ← (MSR)
  (ESR[PTR]) ← 1
  (MSR[WE, EE, PR, PE, DR, IR]) ← 0
  (MSR[LE]) ← (MSR[ILE])
  PC ← $EVPR_{0:15}$ || x'0700'

- If TRAP is enabled as an External debug event (DBCR[TDE] = 1 and DBCR[EDM] = 1):

  TRAP will go to the Debug Stop state, to be handled by an external debugger with hardware control over the PPC403GC.

  (DBSR[TIE]) ← 1

  In addition, if TRAP is also enabled as an Internal debug event (DBCR[IDM] = 1) and Debug Exceptions are disabled (MSR[DE] = 0), then an imprecise event will be reported by setting (DBSR[IDE]) ← 1

  PC ← address of **twi** instruction

- If TRAP is enabled as an Internal debug event and not an External debug event (DBCR[TDE] = 1 and DBCR[EDM,IDM] = 0,1) and Debug Exceptions are enabled (MSR[DE] = 1):

  TRAP will cause a Debug interrupt. See Section 6.16 (Debug Exception Handling) on page 6-38 for further information.

twi

Trap Word Immediate

(SRR2) ← address of **twi** instruction
(SRR3) ← (MSR)
(DBSR[TIE]) ← 1
(MSR[WE, EE, PR, PE, CE, DE, DR, IR]) ← 0
(MSR[LE]) ← (MSR[ILE])
PC ← $EVPR_{0:15}$ || x'2000'

- If TRAP is enabled as an Internal debug event and not an External debug event (DBCR[TDE] = 1 and DBCR[EDM,IDM] = 0,1) and Debug Exceptions are disabled (MSR[DE] = 0):

  TRAP will report the debug event as an imprecise event and will cause a Program interrupt. See Section 6.9 (Program Exceptions) on page 6-32 and Section 6.2.5 (Exception Syndrome Register (ESR)) on page 6-12 for further information.

  (SRR0) ← address of **twi** instruction
  (SRR1) ← (MSR)
  (ESR[PTR]) ← 1
  (DBSR[TIE,IDE]) ← 1,1
  (MSR[WE, EE, PR, PE, DR, IR]) ← 0
  (MSR[LE]) ← (MSR[ILE])
  PC ← $EVPR_{0:15}$ || x'0700'

Registers Altered

- None

Programming Note

This instruction is inserted into the execution stream by a debugger to implement breakpoints, and is not typically used by application code.

Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

Table 11-33. Extended Mnemonics for twi

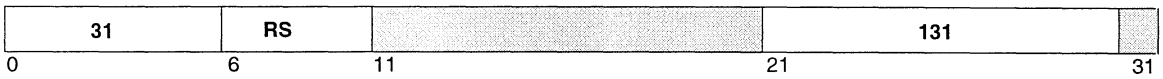
Mnemonic	Operands	Function	Other Registers Changed
tweqi	RA, IM	Trap if (RA) equal to EXTS(IM). *Extended mnemonic* for **twi 4,RA,IM**	
twgei		Trap if (RA) greater than or equal to EXTS(IM). *Extended mnemonic* for **twi 12,RA,IM**	
twgti		Trap if (RA) greater than EXTS(IM). *Extended mnemonic* for **twi 8,RA,IM**	
twlei		Trap if (RA) less than or equal to EXTS(IM). *Extended mnemonic* for **twi 20,RA,IM**	
twlgei		Trap if (RA) logically greater than or equal to EXTS(IM). *Extended mnemonic* for **twi 5,RA,IM**	
twlgti		Trap if (RA) logically greater than EXTS(IM). *Extended mnemonic* for **twi 1,RA,IM**	
twllei		Trap if (RA) logically less than or equal to EXTS(IM). *Extended mnemonic* for **twi 6,RA,IM**	
twllti		Trap if (RA) logically less than EXTS(IM). *Extended mnemonic* for **twi 2,RA,IM**	
twlngi		Trap if (RA) logically not greater than EXTS(IM). *Extended mnemonic* for **twi 6,RA,IM**	
twlnli		Trap if (RA) logically not less than EXTS(IM). *Extended mnemonic* for **twi 5,RA,IM**	
twlti		Trap if (RA) less than EXTS(IM). *Extended mnemonic* for **twi 16,RA,IM**	
twnei		Trap if (RA) not equal to EXTS(IM). *Extended mnemonic* for **twi 24,RA,IM**	
twngi		Trap if (RA) not greater than EXTS(IM). *Extended mnemonic* for **twi 20,RA,IM**	
twnli		Trap if (RA) not less than EXTS(IM). *Extended mnemonic* for **twi 12,RA,IM**	

This instruction is specific to the PowerPC Embedded Controller family

wrtee

Write External Enable

wrtee RS



MSR[EE] ← $(RS)_{16}$

The MSR[EE] is set to the value specified by bit 16 of register RS.

If instruction bit 31 contains 1, the contents of CR[CR0] are undefined.

Registers Altered

- MSR[EE]

Invalid Instruction Forms:

- Reserved fields

Programming Note

Execution of this instruction is privileged.

This instuction is used to provide atomic update of MSR[EE]. Typical usage is:

```
mfmsr    Rn          #save EE in Rn[16]
wrteei   0           #turn off EE
 • • • • • • • • • • • #code with EE disabled
wrtee    Rn          #restore EE without affecting other MSR changes
                      that may have occurred during the disabled code
```

11

Architecture Note

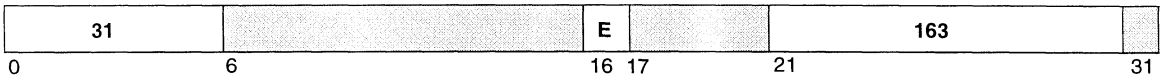
This instruction is specific to the PowerPC Embedded Controller family; it is not described in *PowerPC Architecture*. Programs using this instruction may not be portable to other PowerPC implementations.

This instruction is specific to the PowerPC Embedded Controller family

wrteei

Write External Enable Immediate

wrteei E



MSR[EE] ← E

The MSR[EE] is set to the value specified by the E field.

If instruction bit 31 contains 1, the contents of CR[CR0] are undefined.

Registers Altered

- MSR[EE]

Invalid Instruction Forms:

- Reserved fields

Programming Note

Execution of this instruction is privileged.

This instuction is used to provide atomic update of MSR[EE]. Typical usage is:

```
mfmsr    Rn          #save EE in Rn[16]
wrteei   0           #turn off EE
• • • • • • • • • • • #code with EE disabled
wrtee    Rn          #restore EE without affecting other MSR changes
                      that may have occurred during the disabled code
```

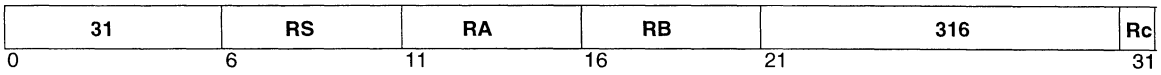
Architecture Note

This instruction is specific to the PowerPC Embedded Controller family; it is not described in *PowerPC Architecture*. Programs using this instruction may not be portable to other PowerPC implementations.

xor

XOR

xor	RA,RS,RB	(Rc=0)
xor.	RA,RS,RB	(Rc=1)



(RA) ← (RS) ⊕ (RB)

The contents of register RS are XORed with the contents of register RB; the result is placed into register RA.

Registers Altered

- $CR[CR0]_{LT, GT, EQ, SO}$ if Rc contains 1
- RA

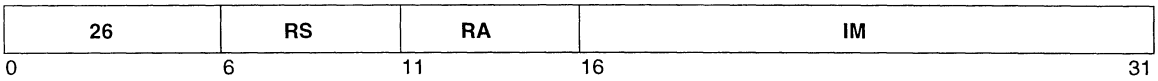
Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

xori

XOR Immediate

xori RA,RS,IM



(RA) ← (RS) ⊕ (^{16}0 || IM)

The IM field is extended to 32 bits by concatenating 16 0-bits on the left. The contents of register RS are XORed with the extended IM field; the result is placed into register RA.

Registers Altered

- RA

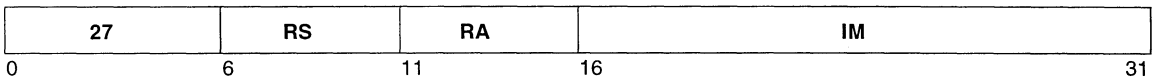
Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

xoris

XOR Immediate Shifted

xoris RA,RS,IM



(RA) ← (RS) ⊕ (IM || $^{16}0$)

The IM field is extended to 32 bits by concatenating 16 0-bits on the right. The contents of register RS are XORed with the extended IM field; the result is placed into register RA.

Registers Altered

- RA

Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.