APPENDIX E
SIMPLIFIED MNEMONICS

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

E.1 Symbols

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

Table E-1 Condition Register CR Field Bit Symbols

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

The simplified mnemonics in **E.5 Simplified Mnemonics for Branch Instructions** and **E.6 Simplified Mnemonics for Condition Register Logical Instructions** require identification of a CR bit. If one of the CR field symbols is used, it must be multiplied by four and added to a symbol or value (zero to three) representing the bit number within the CR field.

The simplified mnemonics in **E.5.3 Branch Mnemonics Incorporating Condi-**

tions and **E.3 Simplified Mnemonics for Compare Instructions** require identification of a CR field. If one of the CR field symbols is used, it must *not* be multiplied by four. Refer to each of these sections for examples that use the symbols in **Table E-1**.

E.2 Simplified Mnemonics for Subtract Instructions

This section discusses simplified mnemonics for the subtract instructions.

E.2.1 Subtract Immediate

Although there is not a "subtract immediate" instruction, its effect can be achieved by using an **addi** instruction with the immediate operand negated. Simplified mnemonics are provided that include this negation, making the intent of the computation clearer. In these examples, the immediate operand "value" is subtracted from the value in **r**A and the result placed in **r**D.

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

E.2.2 Subtract

The "subtract-from" instructions subtract the second operand (**r**A) from the third (**r**B). Simplified mnemonics are provided in which the third operand is subtracted from the second. Both these mnemonics can be coded with a final 'o' or '.' (or both) to cause the OE or Rc bit, respectively, to be set in the underlying instruction. In these examples, the value in **r**B is subtracted from the value in **r**A and the result placed in **r**D.

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

E.3 Simplified Mnemonics for Compare Instructions

The instructions listed in **Table 4-3** are simplified mnemonics that provide compare word capability for 32-bit operands. These instructions correctly clear the L value in the instruction (specifying a 32-bit operand; refer to **4.3.2 Integer Compare Instructions**) rather than requiring it to be coded as a numeric operand.

The **crf**D field can be omitted if the result of the comparison is to be placed into the CR0 field. Otherwise, the target CR field must be specified as the first operand. The CR field symbols defined in **E.1 Symbols** can be used to identify the condition register field.

CAUTION

If the **crf**D field is omitted from a compare mnemonic, the L field must also be omitted. That is, when the assembler encounters a compare instruction with three operands, it interprets the first operand to be the **crf**D field.

Table E-2 Word Compare Simplified Mnemonics

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

The following examples demonstrate the use of the word compare mnemonics:

11. Compare 32 bits in register **r**A with immediate value 100 and place result in condition register field CR0.

    cmpwi **r**A,**100** (equivalent to **cmpi 0,0,r**A,**100**)

12. Same as (1), but place results in condition register field CR4.

    cmpwi **cr4,r**A,**100** (equivalent to **cmpi 4,0,r**A,**100**)

13. Compare registers **r**A and **r**B as logical 32-bit quantities and place result in condition register field CR0.

    cmplw **r**A,**r**B (equivalent to **cmpl 0,0,r**A,**r**B)

14. Same as (3), but place result in condition register field CR4.

    cmplw **cr4**,r**A,r**B (equivalent to **cmpl 4,0,r**A,**r**B)

E.4 Simplified Mnemonics for Rotate and Shift Instructions

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

- Extract — Select a field of *n* bits starting at bit position *b* in the source register; left or right justify this field in the target register; clear all other bits of the target register.
- Insert — Select a left-justified or right-justified field of *n* bits in the source reg-

ister; insert this field starting at bit position *b* of the target register; leave other bits of the target register unchanged. (No simplified mnemonic is provided for insertion of a left-justified field when operating on double words, because such an insertion requires more than one instruction.)

- Rotate — Rotate the contents of a register right or left *n* bits without masking.
- Shift — Shift the contents of a register right or left *n* bits, clearing vacated bits (logical shift).
- Clear — Clear the leftmost or rightmost *n* bits of a register.
- Clear left and shift left — Clear the leftmost *b* bits of a register, then shift the register left by *n* bits. This operation can be used to scale a (known non-negative) array index by the width of an element.

The word rotate and shift operations shown in **Table E-3** are available in all implementations. All these mnemonics can be coded with a final '.' to cause the Rc bit to be set in the underlying instruction.

Table E-3 Word Rotate and Shift Instructions

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

The following examples illustrate the use of these mnemonics.

1. Extract the sign bit (bit 32) of **r**S and place the result right-justified into **r**A.

   extrwi **r**A,**r**S,**1**,**0** (equivalent to: **rlwinm** **r**A,**r**S,**1**,**31**,**31**)

2. Insert the bit extracted in (1) into the sign bit (bit 32) of **r**B.

   insrwi **r**B,**r**A,**1**,**0** (equivalent to: **rlwimi** **r**B,**r**A,**31**,**0**,**0**

3. Shift the contents of **r**A left 8 bits, clearing the high-order 32 bits.

   slwi **r**A,**r**A,**8** (equivalent to: **rlwinm** **r**A,**r**A,**8**,**0**,**23**

E.5 Simplified Mnemonics for Branch Instructions

Mnemonics are provided so that branch conditional instructions can be coded with the condition as part of the instruction mnemonic rather than as a numeric operand. The mnemonics discussed in this section are variations of the branch conditional instructions.

E.5.1 BO and BI Fields

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

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

Table E-4 BO Operand Encodings

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

The *z* indicates a bit that must be zero; otherwise, the instruction form is invalid.

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

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

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

E.5.2 Basic Branch Mnemonics

Table E-5 provides the simplified mnemonics for the most commonly performed conditional branches. These mnemonics allow all the BO operand encodings shown in **Table E-4** to be specified as part of the mnemonic, along with the absolute address (AA) and set link register (LK) bits. (The *y* bit in the BO operand is always cleared in these simplified mnemonics.)

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

Table E-5 Simplified Branch Mnemonics

	LK Bit Not Set (LR Update Not Enabled)				LK Bit Set (LR Update Enabled)			
Branch Semantics	**bc** Relative	**bca** Absolute	**bclr** to LR	**bcctr** to CTR	**bcl** Relative	**bcla** Absolute	**bclrl** to LR	**bcctrl** to CTR
Branch unconditionally	b[1]	ba[1]	**blr**	**bctr**	bl[1]	bla[1]	**blrl**	**bctrl**
Branch if condition true[2]	**bt**	**bta**	**btlr**	**btctr**	**btl**	**btla**	**btlrl**	**btctrl**
Branch if condition false[2]	**bf**	**bfa**	**bflr**	**bfctr**	**bfl**	**bfla**	**bflrl**	**bfctrl**
Decrement CTR, branch if CTR non-zero	**bdnz**	**bdnza**	**bdnzlr**	—	**bdnzl**	**bdnzla**	**bdnzlrl**	—
Decrement CTR, branch if CTR non-zero AND condition true	**bdnzt**	**bdnzta**	**bdnztlr**	—	**bdnztl**	**bdnztla**	**bdnztlrl**	—
Decrement CTR, branch if CTR non-zero AND condition false	**bdnzf**	**bdnzfa**	**bdnzflr**	—	**bdnzfl**	**bdnzfla**	**bdnzflrl**	—
Decrement CTR, branch if CTR zero	**bdz**	**bdza**	**bdzlr**	—	**bdzl**	**bdzla**	**bdzlrl**	—
Decrement CTR, branch if CTR zero AND condition true	**bdzt**	**bdzta**	**bdztlr**	—	**bdztl**	**bdztla**	**bdztlrl**	—
Decrement CTR, branch if CTR zero AND condition false	**bdzf**	**bdzfa**	**bdzflr**	—	**bdzfl**	**bdzfla**	**bdzflrl**	—

NOTES:
1. These are basic mnemonics, not simplified mnemonics.
2. Refer to **Table E-7** for an expanded set of simplified mnemonics for “branch if condition true” and “branch if condition false.” This expanded set of simplified mnemonics incorporates the condition being tested as part of the mnemonic.

Table E-6 provides the operands for the simplified mnemonics in **Table E-5**, as well as the operands of the corresponding basic branch instruction.

Table E-6 Operands for Simplified Branch Mnemonics

Branch Type	Simplified Mnemonic		Equivalent to:	
	Mnemonic	Operands	Mnemonic	Operands
Branch unconditionally	**blr**	None	**bclr**	**20,0**
	bctr	None	**bcctr**	**20,0**
	blrl	None	**bclrl**	**20,0**
	bctrl	None	**bcctrl**	**20,0**
Branch if true	**bt**	BI,target	**bc**	**12**,BI,target
	bta	BI,target	**bca**	**12**,BI,target
	btlr	BI	**bclr**	**12**,BI
	btctr	BI	**bcctr**	**12**,BI
	btl	BI,target	**bcl**	**12**,BI,target
	btla	BI,target	**bcla**	**12**,BI,target
	btlrl	BI	**bclrl**	**12**,BI
	btctrl	BI	**bcctrl**	**12**,BI
Branch if false	**bf**	BI,target	**bc**	**4**,BI,target
	bfa	BI,target	**bca**	**4**,BI,target
	bflr	BI	**bclr**	**4**,BI
	bfctr	BI	**bcctr**	**4**,BI
	bfl	BI,target	**bcl**	**4**,BI,target
	bfla	BI,target	**bcla**	**4**,BI,target
	bflrl	BI	**bclrl**	**4**,BI
	bfctrl	BI	**bcctrl**	**4**,BI
Decrement CTR, branch if CTR non-zero	**bdnz**	target	**bc**	**16,0**,target
	bdnza	target	**bca**	**16,0**,target
	bdnzlr	None	**bclr**	**16,0**
	bdnzl	target	**bcl**	**16,0**,target
	bdnzla	target	**bcla**	**16,0**,target
	bdnzlrl	None	**bclrl**	**16,0**
Decrement CTR, branch if CTR non-zero AND condition true	**bdnzt**	BI,target	**bc**	**8**,BI,target
	bdnzta	BI,target	**bca**	**8**,BI,target
	bdnztlr	BI	**bclr**	**8**,BI
	bdnztl	BI,target	**bcl**	**8**,BI,target
	bdnztla	BI,target	**bcla**	**8**,BI,target
	bdnztlrl	BI	**bclrl**	**8**,BI
Decrement CTR, branch if CTR non-zero AND condition false	**bdnzf**	BI,target	**bc**	**0**,BI,target
	bdnzfa	BI,target	**bca**	**0**,BI,target
	bdnzflr	BI	**bclr**	**0**,BI
	bdnzfl	BI,target	**bcl**	**0**,BI,target
	bdnzfla	BI,target	**bcla**	**0**,BI,target
	bdnzflrl	BI	**bclrl**	**0**,BI

Table E-6 Operands for Simplified Branch Mnemonics (Continued)

Branch Type	Simplified Mnemonic		Equivalent to:	
	Mnemonic	Operands	Mnemonic	Operands
Decrement CTR, branch if CTR zero	**bdz**	target	**bc**	**18,0**,target
	bdza	target	**bca**	**18,0**,target
	bdzlr	None	**bclr**	**18,0**
	bdzl	target	**bcl**	**18,0**,target
	bdzla	target	**bcla**	**18,0**,target
	bdzlrl	None	**bclrl**	**18,0**
Decrement CTR, branch if CTR zero AND condition true	**bdzt**	BI,target	**bc**	**10**,BI,target
	bdzta	BI,target	**bca**	**10**,BI,target
	bdztlr	BI	**bclr**	**10**,BI
	bdztl	BI,target	**bcl**	**10**,BI,target
	bdztla	BI,target	**bcla**	**10**,BI,target
	bdztlrl	BI	**bclrl**	**10**,BI
Decrement CTR, branch if CTR zero AND condition false	**bdzf**	BI,target	**bc**	**2**,BI,target
	bdzfa	BI,target	**bca**	**2**,BI,target
	bdzflr	BI	**bclr**	**2**,BI
	bdzfl	BI,target	**bcl**	**2**,BI,target
	bdzfla	BI,target	**bcla**	**2**,BI,target
	bdzflrl	BI	**bclrl**	**2**,BI

Instructions using a mnemonic from **Table E-5** that test a condition specify the condition (bit in the condition register) as the first (BI) operand of the instruction. The symbols defined in **E.1 Symbols** can be used in this operand. If one of the CR field symbols is used, it must be multiplied by four and added to a symbol or value (zero to three) representing the bit number within the CR field.

The simplified mnemonics found in **Table E-5** are illustrated in the following examples:

1. Decrement CTR and branch if it is still non-zero (closure of a loop controlled by a count loaded into CTR).

   bdnz target (equivalent to **bc 16,0**, target)

2. Same as (1) but branch only if CTR is non-zero and condition in CR0 is “equal.”

   bdnzt **eq**, target (equivalent to **bc 8,2**,target)

3. Same as (2), but “equal” condition is in CR5.

   bdnzt **4 * cr5+eq**,target (equivalent to **bc 8,22**,target)

4. Branch if bit 27 of CR is false.

   bf **27**,target (equivalent to **bc 4,27**,target)

5. Same as (4), but set the link register. This is a form of conditional “call.”

   bfl **27**,target (equivalent to **bcl 4,27**,target)

E.5.3 Branch Mnemonics Incorporating Conditions

The mnemonics defined in **Table E-7** are variations of the “branch if condition true” and “branch if condition false” BO encodings, with the most common values of the BI operand represented in the mnemonic rather than specified as a numeric operand.

Table E-7 Simplified Branch Mnemonics with Comparison Conditions

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

Table E-8 shows the operands used with the simplified branch mnemonics in **Table E-7**. The examples provided are for the first column of **Table E-7** (simplified forms of the **bc** instruction), but all entries within a row in **Table E-7** use the same operands (except that branches to the LR or CTR do not require a “target” operand). **Table E-8** also indicates the operands used with the corresponding basic branch mnemonic.

Table E-8 Operands for Simplified Branch Mnemonics with Comparison Conditions

Branch	Simplified Mnemonics Example	Equivalent to
Branch if less than	**blt crf**D,target	**bc 12,4*crf**D,target
Branch if less than or equal	**ble crf**D,target	**bc 4,4*crf**D**+1**,target
Branch if equal	**beq crf**D,target	**bc 12**, **4*crf**D**+2**,target
Branch if greater than	**bgt crf**D,target	**bc 12,4*crf**D**+1**,target

Table E-8 Operands for Simplified Branch Mnemonics with Comparison Conditions (Continued)

Branch	Simplified Mnemonics Example	Equivalent to
Branch if greater than or equal	**bge crf**D,target	**bc 4,4*crf**D,target
Branch if not less than	**bnl crf**D,target	**bc 4,4*crf**D,target[1]
Branch if not equal	**bne crf**D,target	**bc 4,4*crf**D**+2**,target
Branch if not greater than	**bng crf**D,target	**bc 4,4*crf**D**+1**,target[2]
Branch if summary overflow	**bso crf**D,target	**bc 12,4*crf**D**+3**,target
Branch if not summary overflow	**bns crf**D,target	**bc 4,4*crf**D**+3**,target
Branch if unordered	**bun crf**D,target	**bc 12,4*crf**D**+3**,target
Branch if not unordered	**bnu crf**D,target	**bc 4,4*crf**D**+3**,target

NOTES:
1. Same as “branch if greater than or equal.”
2. Same as “branch if less than or equal.”

Instructions using the mnemonics in **Table E-7** specify the condition register field in an optional first operand. If the CR field being tested is CR0, this operand need not be specified. Otherwise, one of the CR field symbols defined in **E.1 Symbols** can be used for this operand.

If one of the CR field symbols is used, it must *not* be multiplied by four. The bit number within the CR field is part of the simplified mnemonic. The CR field is identified, and the assembler does the multiplication and addition required to produce a CR bit number for the BI field of the underlying basic mnemonic.)

The simplified mnemonics found in **Table E-7** are used in the following examples:

1. Branch if CR0 reflects condition “not equal.”

   bne target equivalent to **bc 4,2**,target)

2. Same as (1), but condition is in CR3.

   bne **cr3**,target equivalent to **bc 4,14**,target)

3. Branch to an absolute target if CR4 specifies “greater than,” setting the link register. This is a form of conditional “call”, as the return address is saved in the link register.

   bgtla **cr4**,target (equivalent to **bcla 12,17**,target)

4. Same as (3), but target address is in the count register.

   bgtctrl **cr4** (equivalent to **bcctrl 12,17**)

E.5.4 Branch Prediction

In branch conditional instructions that are not always taken, the low-order bit (*y* bit)

of the BO field provides a hint about whether the branch is likely to be taken. See **4.6.2 Conditional Branch Control** for more information on the *y* bit.

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

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

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

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

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

Examples of branch prediction follow:

1. Branch if CR0 reflects condition "less than," specifying that the branch should be predicted to be taken.

   blt+ target

2. Same as (1), but target address is in the LR and the branch should be predicted not to be taken.

   bltlr–

E.6 Simplified Mnemonics for Condition Register Logical Instructions

The condition register logical instructions are used to set, clear, copy, or invert a given condition register bit. The simplified mnemonics shown in **Table E-9** allow these operations to be coded easily.

Table E-9 Condition Register Logical Mnemonics

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

The symbols defined in **E.1 Symbols** can be used to identify the condition register bit. If one of the CR field symbols is used, it must be multiplied by four and added to a symbol or value (zero to three) representing the bit number within the CR field.

The following examples illustrate the condition register logical mnemonics:

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

E.7 Simplified Mnemonics for Trap Instructions

A standard set of codes, shown in **Table E-10**, has been adopted for the most common combinations of trap conditions.

Table E-10 Trap Mnemonics Encoding

Code	Meaning	TO Operand Encoding	<	>	=	<U[1]	>U[2]
lt	Less than	16	1	0	0	0	0
le	Less than or equal	20	1	0	1	0	0
eq	Equal	4	0	0	1	0	0
ge	Greater than or equal	12	0	1	1	0	0
gt	Greater than	8	0	1	0	0	0
nl	Not less than	12	0	1	1	0	0
ne	Not equal	24	1	1	0	0	0
ng	Not greater than	20	1	0	1	0	0
llt	Logically less than	2	0	0	0	1	0
lle	Logically less than or equal	6	0	0	1	1	0
lge	Logically greater than or equal	5	0	0	1	0	1
lgt	Logically greater than	1	0	0	0	0	1
lnl	Logically not less than	5	0	0	1	0	1
lng	Logically not greater than	6	0	0	1	1	0
(none)	Unconditional	31	1	1	1	1	1

NOTES:
1. The symbol ‘<U’ indicates an unsigned “less than” evaluation will be performed.
2. The symbol ‘>U’ indicates an unsigned “greater than” evaluation will be performed.

The mnemonics defined in **Table E-11** are variations of the trap instructions, with the most useful values of the trap instruction TO operand represented as a mnemonic rather than specified as a numeric operand.

Table E-11 Trap Mnemonics

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

The following examples illustrate the use of simplified mnemonics for trap instructions:

1. Trap if Rx, considered as a 32-bit quantity, is logically greater than 0x7FF.

   twlg **r**A**,** 0x7FF (equivalent to **twi 1,r**A**,** 0x7FF)

2. Trap unconditionally.

   trap (equivalent to **tw 31,0,0**)

Trap instructions evaluate a trap condition as follows: the contents of register **r**A are compared with either the sign-extended SIMM field or the contents of register **r**B, depending on the trap instruction.

The comparison results in five conditions which are ANDed with operand TO. If the result is not zero, the trap exception handler is invoked. See **Table E-12** for these conditions.

Table E-12 TO Operand Bit Encoding

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

E.8 Simplified Mnemonics for Special-Purpose Registers

The **mtspr** and **mfspr** instructions specify an SPR as a numeric operand. Simplified mnemonics are provided that represent the SPR in the mnemonic rather than requiring it to be coded as an operand. **Table E-13** below specifies the simplified mnemonics provided for SPR operations.

Table E-13 SPR Simplified Mnemonics

Special Purpose Register	Move to SPR Simplified Mnemonic	Move to SPR Instruction	Move from SPR Simplified Mnemonic	Move from SPR Instruction[1]
Integer unit exception register	**mtxer** rS	**mtspr** 1,rS	**mfxer** rD	**mfspr** rD,**1**
Link register	**mtlr** rS	**mtspr** 8,rS	**mflr** rD	**mfspr** rD,**8**
Count register	**mtctr** rS	**mtspr** 9,rS	**mfctr** rD	**mfspr** rD,**9**
DAE/source instruction service register	**mtdsisr** rS	**mtspr** 18,rS	**mfdsisr** rD	**mfspr** rD,**18**
Data address register	**mtdar** rS	**mtspr** 19,rS	**mfdar** rD	**mfspr** rD,**19**
Decrementer	**mtdec** rS	**mtspr** 22,rS	**mfdec** rD	**mfspr** rD,**22**
Status save/restore register 0	**mtsrr0** rS	**mtspr** 26,rS	**mfsrr0** rD	**mfspr** rD,**26**
Status save/restore register 1	**mtsrr1** rS	**mtspr** 27,rS	**mfsrr1** rD	**mfspr** rD,**27**
General special purpose registers G0 through G3	**mtsprg** *n*,rS	**mtspr** 272+*n*,rS	**mfsprg** rD,*n*	**mfspr** rD,**272**+*n*
Time base (lower)	**mttbl** rS	**mtspr 284,**rS	**mftb** rD	**mftb** rD,**268**
Time base (upper)	**mttbu** rS	**mtspr 285,**rS	**mftbu** rD	**mftb** rD,**269**
Processor version register	—	—	**mfpvr** rD	**mfspr** rD,**287**

NOTES:
1. Except for **mftb** and **mftbu**

The following examples illustrate the use of SPR simplified mnemonics.

1. Copy the contents of the low-order 32 bits of **r**S to the XER.

   mtxer **rS** (equivalent to **mtspr 1,r**S)

2. Copy the contents of the LR to **r**S.

   mflr **r**S (equivalent to **mfspr rS**,8)

3. Copy the contents of **r**S to the CTR.

   mtctr **r**S (equivalent to **mtspr 9**,**r**S)

E.9 Recommended Simplified Mnemonics

This section describes some of the most commonly-used operations: no-op, load immediate, load address, move register, complement register, and move to condition register.

E.9.1 No-Op

Many PowerPC instructions can be coded in a way such that, effectively, no operation is performed. An additional mnemonic is provided for the preferred form of no-op.

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

E.9.2 Load Immediate

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

The following instruction loads a 16-bit signed immediate value into **r**A:

li **r**A**,**value (equivalent to **addi r**A**,0,**value)

The following instruction loads a 16-bit signed immediate value, shifted left by 16 bits, into **r**A:

lis **r**A**,**value (equivalent to **addi r**A**,0,**value)

E.9.3 Load Address

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

la **r**D,SIMM(**r**A) (equivalent to **addi r**D,**r**A,SIMM)

The **la** mnemonic is useful for obtaining the address of a variable specified by name, allowing the assembler to supply the base register number and compute the

displacement. If the variable *v* is located at offset SIMM*v* bytes from the address in register **r***v,* and the assembler has been told to use register **r***v* as a base for references to the data structure containing *v*, then the following line causes the address of *v* to be loaded into register **r**D.

la **r**D,*v* (equivalent to **addi r**D**,r**A**,**SIMM*v)*

E.9.4 Move Register

Several PowerPC instructions can be coded to simply copy the contents of one register to another. An extended mnemonic is provided to move data from one register to another with no computational activity.

The following instruction copies the contents of register **r**S into register **r**A. This mnemonic can be coded with a '.' to cause the condition register update option to be specified in the underlying instruction.

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

E.9.5 Complement Register

Several PowerPC instructions can be coded to complement the contents of one register and place the result in another register. A simplified mnemonic is provided that complements the contents of **r**S and places the results into register **r**A. This mnemonic can be coded with a '.' to cause the condition register update option to be specified in the underlying instruction.

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

E.9.6 Move to Condition Register

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

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