AN2491
Rev. 0, 9/2003

Simplified Mnemonics for PowerPC™ Instructions

Jerry Young,
NCSD Applications

Simplified Mnemonics for PowerPC™ Instructions

This document describes simplified mnemonics, which are provided for easier coding of assembly language programs. Simplified mnemonics are defined for the most frequently used forms of branch conditional, compare, trap, rotate and shift, and certain other instructions defined by the PowerPC™ architecture and by implementations of and extensions to the PowerPC architecture.

Most of this information is also provided in the appendixes of reference manuals and the *Programming Environments Manual for 32-Bit Implementations of the PowerPC Architecture* (referred to as the *Programming Environment Manual*). However, Section 12, "Comprehensive List of Simplified Mnemonics," provides an alphabetical listing of simplified mnemonics that are used by a variety of processors. Some assemblers may define additional simplified mnemonics not included here. The simplified mnemonics listed here should be supported by all compilers.

This document describes only simplified mnemonics for 32-bit instructions.

1 Overview

Simplified (or extended) mnemonics allow an assembly-language programmer to program using more intuitive mnemonics and symbols than the instructions and syntax defined by the instruction set architecture. For example, to code the conditional call “branch to an absolute target if CR4 specifies a greater than condition, setting the LR” without simplified mnemonics, the programmer would write the branch conditional instruction **bc 12,17,***target*. The simplified mnemonic, branch if greater than, **bgt cr4,***target*, incorporates the conditions. Not only is it easier to remember the symbols than the numbers when programming, it is also easier to interpret simplified mnemonics when reading existing code.

Although the original PowerPC architecture documents include a set of simplified mnemonics, these are not a formal part of the architecture, but rather a recommendation for assemblers that support the instruction set.

Many simplified mnemonics have been added to those originally included in the architecture documentation. Some assemblers created their own, and others have been added to support extensions to the instruction set (for example, AltiVec instructions and Book E auxiliary processing units (APUs)). Simplified mnemonics for new architecturally defined and new implementation-specific special-purpose registers (SPRs) are described here only in a very general way.

2 Subtract Simplified Mnemonics

This section describes simplified mnemonics for subtract instructions.

2.1 Subtract Immediate

There is no subtract immediate instruction; however, its effect is achieved by negating the immediate operand of an Add Immediate instruction, **addi**. Simplified mnemonics include this negation, making the intent of the computation clearer. These are listed in Table 1.

Table 1. Subtract Immediate Simplified Mnemonics

Simplified Mnemonic	Standard Mnemonic
subi r**D**,r**A**,value	**addi** r**D**,r**A**,–value
subis r**D**,r**A**,value	**addis** r**D**,r**A**,–value
subic r**D**,r**A**,value	**addic** r**D**,r**A**,–value
subic. r**D**,r**A**,value	**addic.** r**D**,r**A**,–value

2.2 Subtract

Subtract from instructions subtract the second operand (**r**A) from the third (**r**B). The simplified mnemonics in Table 2 use the more common order in which the third operand is subtracted from the second.

Table 2. Subtract Simplified Mnemonics

Simplified Mnemonic	Standard Mnemonic [1]
sub[**o**][.] r**D**,r**A**,r**B**	**subf**[**o**][.] r**D**,r**B**,r**A**
subc[**o**][.] r**D**,r**A**,r**B**	**subfc**[**o**][.] r**D**,r**B**,r**A**

[1] r**D**,r**B**,r**A** is not the standard order for the operands. The order of r**B** and r**A** is reversed to show the equivalent behavior of the simplified mnemonic.

3 Rotate and Shift Simplified Mnemonics

Rotate and shift instructions provide powerful, general ways to manipulate register contents, but can be difficult to understand. Simplified mnemonics are provided for the following 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- or right-justified field of *n* bits in the source register; insert this field starting at bit position *b* of the target register; leave other bits of the target register unchanged.
- 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.

3.1 Operations on Words

The simplified mnemonics in Table 3 can be coded with a dot (**.**) suffix to cause the Rc bit to be set in the underlying instruction.

Table 3. Word Rotate and Shift Simplified Mnemonics

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

Examples using word mnemonics follow:

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

4 Branch Instruction Simplified Mnemonics

Branch conditional instructions can be coded with the operations, a condition to be tested, and a prediction, as part of the instruction mnemonic rather than as numeric operands (the BO and BI operands). Table 4 shows the four general types of branch instructions. Simplified mnemonics are defined only for branch instructions that include BO and BI operands; there is no need to simplify unconditional branch mnemonics.

Table 4. Branch Instructions

Instruction Name	Mnemonic	Syntax
Branch	**b** (**ba bl bla**)	target_addr
Branch Conditional	**bc** (**bca bcl bcla**)	BO,BI,target_addr
Branch Conditional to Link Register	**bclr** (**bclrl**)	BO,BI
Branch Conditional to Count Register	**bcctr** (**bcctrl**)	BO,BI

The BO and BI operands correspond to two fields in the instruction opcode, as Figure 1 shows for Branch Conditional (**bc**, **bca**, **bcl**, and **bcla**) instructions.

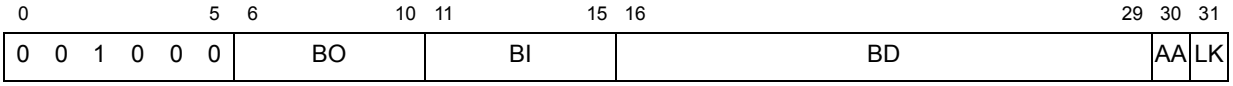


Figure 1. Branch Conditional (bc) Instruction Format

The BO operand specifies branch operations that involve decrementing CTR. It is also used to determine whether testing a CR bit causes a branch to occur if the condition is true or false.

The BI operand identifies a CR bit to test (whether a comparison is less than or greater than, for example). The simplified mnemonics avoid the need to memorize the numerical values for BO and BI.

For example, **bc 16,0,***target* is a conditional branch that, as a BO value of 16 (0b1_0000) indicates, decrements the CTR, then branches if the decremented CTR is not zero. The operation specified by BO is abbreviated as **d** (for decrement) and **nz** (for not zero), which replace the **c** in the original mnemonic; so the simplified mnemonic for **bc** becomes **bdnz**. The branch does not depend on a condition in the CR, so BI can be eliminated, reducing the expression to **bdnz** *target*.

In addition to CTR operations, the BO operand provides an optional prediction bit, and a true or false indicator can be added. For example, if the previous instruction should branch only on an equal condition in CR0, the instruction becomes **bc 8,2,***target*. To incorporate a true condition, the BO value becomes 8 (as shown in Table 6); the CR0 equal field is indicated by a BI value of 2 (as shown in Table 7). Incorporating the branch-if-true condition adds a '**t**' to the simplified mnemonic, **bdnzt.** The BI value of 2 is replaced by the **eq** symbol. Using the simplified mnemonic and the **eq** operand, the expression becomes **bdnzt eq,***target*.

This example tests CR0[EQ]; however, to test the equal condition in CR5 (CR bit 22), the expression becomes **bc 8,22,***target*. The BI operand of 22 indicates CR[22] (CR5[2], or BI field 0b10110), as shown in Table 7. This can be expressed as the simplified mnemonic. **bdnzt 4 * cr5 + eq,***target*.

The notation, **4 * cr5 + eq** may at first seem awkward, but it eliminates computing the value of the CR bit. It can be seen that (4 * 5) + 2 = 22. Note that although 32-bit registers in Book E processors are numbered 32–63, only values 0–31 are valid (or possible) for BI operands. As shown in Table 8, a Book E–compliant processor automatically translates the bit values; specifying a BI value of 22 selects bit 54 on a Book E processor, or CR5[2] = CR5[EQ].

4.1 Key Facts about Simplified Branch Mnemonics

The following key points are helpful in understanding how to use simplified branch mnemonics:

- All simplified branch mnemonics eliminate the BO operand, so if any operand is present in a branch simplified mnemonic, it is the BI operand (or a reduced form of it).
- If the CR is not involved in the branch, the BI operand can be deleted
- If the CR is involved in the branch, the BI operand can be treated in the following ways:
  - It can be specified as a numeric value, just as it is in the architecturally defined instruction, or it can be indicated with an easier to remember formula, **4 * cr***n* **+** [test bit symbol], where *n* indicates the CR field number.
  - The condition of the test bit (eq, lt, gt, and so) can be incorporated into the mnemonic, leaving the need for an operand that defines only the CR field.
    - If the test bit is in CR0, no operand is needed.
    - If the test bit is in CR1–CR7, the BI operand can be replaced with a **cr**S operand (that is, **cr1**, **cr2**, **cr3**, and so forth.

4.2 Eliminating the BO Operand

The 5-bit BO field, shown in Figure 2, encodes the following operations in conditional branch instructions:

- Decrement count register (CTR)
  - And test if result is equal to zero
  - And test if result is not equal to zero
- Test condition register (CR)
  - Test condition true
  - Test condition false
- Branch prediction (taken, fall through). If the prediction bit, *y*, is needed, it is signified by appending a plus or minus sign as described in Section 4.3, “Incorporating the BO Branch Prediction.”

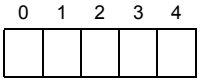


Figure 2. BO Field (Bits 6–10 of the Instruction Encoding)

BO bits can be interpreted individually as described in Table 5.

Table 5. BO Bit Encodings

BO Bit	Description
0	If set, ignore the CR bit comparison.
1	If set, the CR bit comparison is against true; if not set the CR bit comparison is against false.
2	If set, the CTR is not decremented.
3	If BO[2] is set, this bit determines whether the CTR comparison is for equal to zero or not equal to zero.
4	The *y* bit. If set, reverse the static prediction. Use of the this bit is optional and independent from the interpretation of the rest of the BO operand. Because simplified branch mnemonics eliminate the BO operand, this bit is programmed by adding a plus or minus sign to the simplified mnemonic, as described in Section 4.3, “Incorporating the BO Branch Prediction.”

Thus, a BO encoding of 10100 (decimal 20) means ignore the CR bit comparison and do not decrement the CTR—in other words, branch unconditionally. Encodings for the BO operand are shown in Table 6. A *z* bit indicates that the bit is ignored. However, these bits should be cleared, as they may be assigned a meaning in a future version of the architecture.

As shown in Table 6, the '**c**' in the standard mnemonic is replaced with the operations otherwise specified in the BO field, (**d** for decrement, **z** for zero, **nz** for non-zero, **t** for true, and **f** for false).

Table 6. BO Operand Encodings

BO Field	Value [1] (Decimal)	Description	Symbol
0000*y*	0	Decrement the CTR, then branch if the decremented CTR ≠ 0 and condition is FALSE.	**dnzf**
0001*y*	2	Decrement the CTR, then branch if the decremented CTR = 0 and condition is FALSE.	**dzf**
001*z* [2] *y*	4	Branch if the condition is FALSE. [3] Note that 'false' and 'four' both start with 'f'.	**f**
0100*y*	8	Decrement the CTR, then branch if the decremented CTR ≠ 0 and condition is TRUE.	**dnzt**
0101*y*	10	Decrement the CTR, then branch if the decremented CTR = 0 and condition is TRUE.	**dzt**
011*z* [2] *y*	12	Branch if the condition is TRUE. [3] Note that 'true' and 'twelve' both start with 't'.	**t**
1*z* [2] 00*y* [4]	16	Decrement the CTR, then branch if the decremented CTR ≠ 0.	**dnz** [5]
1*z* [2] 01*y* [4]	18	Decrement the CTR, then branch if the decremented CTR = 0.	**dz** [5]
1*z* [2] 1*zz* [4]	20	Branch always.	—

[1] Assumes y = z = 0. Section 4.3, "Incorporating the BO Branch Prediction," describes how to use simplified mnemonics to program the *y* bit for static prediction.

[2] A *z* bit indicates a bit that is ignored. However, these bits should be cleared, as they may be assigned a meaning in a future version of the architecture.

[3] Instructions for which B0 is 12 (branch if condition true) or 4 (branch if condition false) do not depend on the CTR value and can be alternately coded by incorporating the condition specified by the BI field, as described in Section 4.6, "Simplified Mnemonics that Incorporate CR Conditions (Eliminates BO and Replaces BI with **cr**S)."

[4] Simplified mnemonics for branch instructions that do not test CR bits (BO = 16, 18, and 20) should specify only a target. Otherwise a programming error may occur.

[5] Notice that these instructions do not use the branch if condition true or false operations. For that reason, simplified mnemonics for these should not specify a BI operand.

4.3 Incorporating the BO Branch Prediction

As shown in Table 6, the low-order bit (*y* bit) of the BO field provides a hint about whether the branch is likely to be taken (static branch prediction). 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. This suffix can be added to any branch conditional mnemonic, either standard or simplified.

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

For branches to an address in the LR or CTR (**bclr**[**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 less than condition, specifying that the branch should be predicted as taken.
   blt+ *target*
2. Same as (1), but target address is in the LR and the branch should be predicted as not taken.
   bltlr–

4.4 The BI Operand—CR Bit and Field Representations

With standard branch mnemonics, the BI operand is used when it is necessary to test a CR bit, as shown in the example in Section 4, "Branch Instruction Simplified Mnemonics,"

With simplified mnemonics, the BI operand is handled differently depending on whether the simplified mnemonic incorporates a CR condition to test, as follows:

- Some branch simplified mnemonics incorporate only the BO operand. These simplified mnemonics can use the architecturally defined BI operand to specify the CR bit, as follows:
  - — The BI operand can be presented exactly as it is with standard mnemonics—as a decimal number, 0–31.
  - — Symbols can be used to replace the decimal operand, as shown in the example in Section 4, "Branch Instruction Simplified Mnemonics," where **bdnzt 4 * cr5 + eq,***target* could be used instead of **bdnzt 22,***target*. This is described in Section 4.4.1.1, "Specifying a CR Bit."

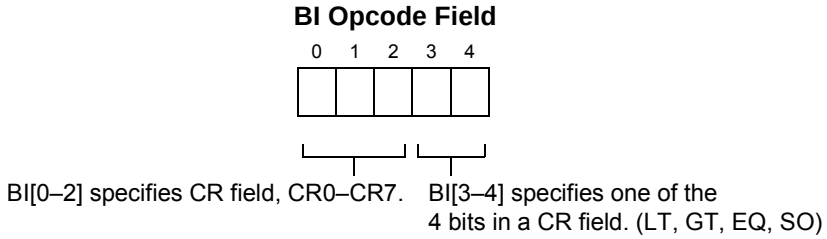
  The simplified mnemonics in Section 4.5, "Simplified Mnemonics that Incorporate the BO Operand," use one of these two methods to specify a CR bit.
- Additional simplified mnemonics are specified that incorporate CR conditions that would otherwise be specified by the BI operand, so the BI operand is replaced by the **cr**S operand to specify the CR field, CR0–CR7. See Section 4.4.1, "BI Operand Instruction Encoding."

  These mnemonics are described in Section 4.6, "Simplified Mnemonics that Incorporate CR Conditions (Eliminates BO and Replaces BI with crS)."

4.4.1 BI Operand Instruction Encoding

The entire 5-bit BI field, shown in Figure 3, represents the bit number for the CR bit to be tested. For standard branch mnemonics and for branch simplified mnemonics that do not incorporate a CR condition, the BI operand provides all 5 bits.

For simplified branch mnemonics described in Section 4.6, "Simplified Mnemonics that Incorporate CR Conditions (Eliminates BO and Replaces BI with crS)," the BI operand is replaced by a **cr**S operand. To understand this, it is useful to view the BI operand as comprised of two parts. As Figure 3 shows, BI[0–2] indicates the CR field and BI[3–4] represents the condition to test.



Simplified mnemonics based on CR conditions but not CTR values—BO = 12 (branch if true) and BO = 4 branch if false)	Specified by a separate, reduced BI operand (**cr**S)	Incorporated into the simplified mnemonic.
Standard branch mnemonics and simplified mnemonics based on CTR values	The BI operand specifies the entire 5-bit field. If CR0 is used, the bit can be identified by LT, GT, EQ, or SO. If CR1–CR7 are used, the form 4 * **cr**S + LT|GT|EQ|SO can be used.	

Figure 3. BI Field (Bits 11–14 of the Instruction Encoding)

Integer record-form instructions update CR0 and floating-point record-form instructions update CR1, as described in Table 7.

4.4.1.1 Specifying a CR Bit

Note that the AIM version of the PowerPC architecture numbers CR bits 0–31 and Book E numbers them 32–63. However, no adjustment is necessary to the code; in Book E devices, 32 is automatically added to the BI value, as shown in Table 7 and Table 8.

Table 7. CR0 and CR1 Fields as Updated by Integer and Floating-Point Instructions

CR*n* Bit	CR Bits		BI		Description
	AIM	Book E	0–2	3–4	
CR0[0]	0	32	000	00	Negative (LT)—Set when the result is negative.
CR0[1]	1	33	000	01	Positive (GT)—Set when the result is positive (and not zero).
CR0[2]	2	34	000	10	Zero (EQ)—Set when the result is zero.
CR0[3]	3	35	000	11	Summary overflow (SO). Copy of XER[SO] at the instruction's completion.
CR1[0]	4	36	001	00	Copy of FPSCR[FX] at the instruction's completion.
CR1[1]	5	37	001	01	Copy of FPSCR[FEX] at the instruction's completion.
CR1[2]	6	38	001	10	Copy of FPSCR[VX] at the instruction's completion.
CR1[3]	7	39	001	11	Copy of FPSCR[OX] at the instruction's completion.

Some simplified mnemonics incorporate only the BO field (as described Section 4.2, "Eliminating the BO Operand"). If one of these simplified mnemonics is used and the CR must be accessed, the BI operand can be specified either as a numeric value or by using the symbols in Table 8.

Compare word instructions (described in Section 5, "Compare Word Simplified Mnemonics"), floating-point compare instructions, move to CR instructions, and others can also modify CR fields, so CR0 and CR1 may hold values that do not adhere to the meanings described in Table 7. CR logical instructions, described in Section 6, "Condition Register Logical Simplified Mnemonics," can update individual CR bits.

Table 8. BI Operand Settings for CR Fields for Branch Comparisons

CR*n* Bit	Bit Expression	CR Bits		BI		Description
		AIM (BI Operand)	Book E	0–2	3–4	
CR*n*[0]	**4 * cr0 + lt** (or **lt**) **4 * cr1 + lt** **4 * cr2 + lt** **4 * cr3+ lt** **4 * cr4 + lt** **4 * cr5 + lt** **4 * cr6 + lt** **4 * cr7 + lt**	0 4 8 12 16 20 24 28	32 36 40 44 48 52 56 60	000 001 010 011 100 101 110 111	00	Less than or floating-point less than (LT, FL). For integer compare instructions: **r**A < SIMM or **r**B (signed comparison) or **r**A < UIMM or **r**B (unsigned comparison). For floating-point compare instructions: **fr**A < **fr**B.
CR*n*[1]	**4 * cr0 + gt** (or **gt**) **4 * cr1 + gt** **4 * cr2 + gt** **4 * cr3+ gt** **4 * cr4 + gt** **4 * cr5 + gt** **4 * cr6 + gt** **4 * cr7 + gt**	1 5 9 13 17 21 25 29	33 37 41 45 49 53 57 61	000 001 010 011 100 101 110 111	01	Greater than or floating-point greater than (GT, FG). For integer compare instructions: rA > SIMM or rB (signed comparison) or **r**A > UIMM or **r**B (unsigned comparison). For floating-point compare instructions: **fr**A > **fr**B.
CR*n*[2]	**4 * cr0 + eq** (or **eq**) **4 * cr1 + eq** **4 * cr2 + eq** **4 * cr3+ eq** **4 * cr4 + eq** **4 * cr5 + eq** **4 * cr6 + eq** **4 * cr7 + eq**	2 6 10 14 18 22 26 30	34 38 42 46 50 54 58 62	000 001 010 011 100 101 110 111	10	Equal or floating-point equal (EQ, FE). For integer compare instructions: **r**A = SIMM, UIMM, or **r**B. For floating-point compare instructions: **fr**A = **fr**B.
CR*n*[3]	**4 * cr0 + so/un** (or **so/un**) **4 * cr1 + so/un** **4 * cr2 + so/un** **4 * cr3 + so/un** **4 * cr4 + so/un** **4 * cr5 + so/un** **4 * cr6 + so/un** **4 * cr7 + so/un**	3 7 11 15 19 23 27 31	35 39 43 47 51 55 59 63	000 001 010 011 100 101 110 111	11	Summary overflow or floating-point unordered (SO, FU). For integer compare instructions, this is a copy of XER[SO] at instruction completion. For floating-point compare instructions, one or both of **fr**A and **fr**B is a NaN.

To provide simplified mnemonics for every possible combination of BO and BI (that is, including bits that identified the CR field) would require $2^{10} = 1024$ mnemonics, most of which would be only marginally useful. The abbreviated set in Section 4.5, “Simplified Mnemonics that Incorporate the BO Operand,” covers useful cases. Unusual cases can be coded using a standard branch conditional syntax.

4.4.1.2 The crS Operand

The **cr**S symbols are shown in Table 9. Note that either the symbol or the operand value can be used in the syntax used with the simplified mnemonic.

Table 9. CR Field Identification Symbols

Symbol	BI[0–2]	CR Bits
cr0 (default, can be eliminated from syntax)	000	32–35
cr1	001	36–39

Table 9. CR Field Identification Symbols (continued)

Symbol	BI[0–2]	CR Bits
cr2	010	40–43
cr3	011	44–47
cr4	100	48–51
cr5	101	52–55
cr6	110	56–59
cr7	111	60–63

To identify a CR bit, an expression in which a CR field symbol is multiplied by 4 and then added to a bit-number-within-CR-field symbol can be used, (for example, **cr0 * 4 + eq**).

4.5 Simplified Mnemonics that Incorporate the BO Operand

The mnemonics in Table 10 allow common BO operand encodings to be specified as part of the mnemonic, along with the absolute address (AA) and set link register bits (LK). There are no simplified mnemonics for relative and absolute unconditional branches. For these, the basic mnemonics **b**, **ba**, **bl**, and **bla** are used.

Table 10. Branch Simplified Mnemonics

Branch Semantics	LR Update Not Enabled				LR Update Enabled			
	bc	**bca**	**bclr**	**bcctr**	**bcl**	**bcla**	**bclrl**	**bcctrl**
Branch unconditionally [1]	—	—	**blr**	**bctr**	—	—	**blrl**	**bctrl**
Branch if condition true	**bt**	**bta**	**btlr**	**btctr**	**btl**	**btla**	**btlrl**	**btctrl**
Branch if condition false	**bf**	**bfa**	**bflr**	**bfctr**	**bfl**	**bfla**	**bflrl**	**bfctrl**
Decrement CTR, branch if CTR ≠ 0 [1]	**bdnz**	**bdnza**	**bdnzlr**	—	**bdnzl**	**bdnzla**	**bdnzlrl**	—
Decrement CTR, branch if CTR ≠ 0 and condition true	**bdnzt**	**bdnzta**	**bdnztlr**	—	**bdnztl**	**bdnztla**	**bdnztlrl**	—
Decrement CTR, branch if CTR ≠ 0 and condition false	**bdnzf**	**bdnzfa**	**bdnzflr**	—	**bdnzfl**	**bdnzfla**	**bdnzflrl**	—
Decrement CTR, branch if CTR = 0 [1]	**bdz**	**bdza**	**bdzlr**	—	**bdzl**	**bdzla**	**bdzlrl**	—
Decrement CTR, branch if CTR = 0 and condition true	**bdzt**	**bdzta**	**bdztlr**	—	**bdztl**	**bdztla**	**bdztlrl**	—
Decrement CTR, branch if CTR = 0 and condition false	**bdzf**	**bdzfa**	**bdzflr**	—	**bdzfl**	**bdzfla**	**bdzflrl**	—

[1] Simplified mnemonics for branch instructions that do not test CR bits should specify only a target. Otherwise a programming error may occur.

Table 11 shows the syntax for basic simplified branch mnemonics

Table 11. Branch Instructions

Instruction	Standard Mnemonic	Syntax	Simplified Mnemonic	Syntax
Branch	**b** (**ba bl bla**)	target_addr	N/A, syntax does not include BO	
Branch Conditional	**bc** (**bca bcl bcla**)	BO,BI,target_addr	**b***x* [1] (**b***x***a b***x***l b***x***la**)	BI [2],target_addr
Branch Conditional to Link Register	**bclr** (**bclrl**)	BO,BI	**b***x***lr** (**b***x***lrl**)	BI
Branch Conditional to Count Register	**bcctr** (**bcctrl**)	BO,BI	**b***x***ctr** (**b***x***ctrl**)	BI

[1] *x* stands for one of the symbols in Table 6, where applicable.
[2] BI can be a numeric value or an expression as shown in Table 9.

The simplified mnemonics in Table 10 that test a condition require a corresponding CR bit as the first operand (as the examples 2–5 in Section 4.5.1, “Examples that Eliminate the BO Operand,” below illustrate). The symbols in Table 9 can be used in place of a numeric value.

4.5.1 Examples that Eliminate the BO Operand

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

1. Decrement CTR and branch if it is still nonzero (closure of a loop controlled by a count loaded into CTR) (note that no CR bits are tested).
   bdnz *target* equivalent to **bc 16,0,***target*
   Because this instruction does not test a CR bit, the simplified mnemonic should specify only a target operand. Specifying a CR (for example, **bdnz** 0,*target* or **bdnz cr0,***target*) may be considered a programming error. Subsequent examples test conditions).
2. Same as (1) but branch only if CTR is nonzero and equal condition in CR0.
   bdnzt eq,*target* equivalent to **bc 8,2,***target*
   Other equivalents include **bdnzt 2,***target* or the unlikely **bdnzt 4 * cr0 + eq,***target*
3. Same as (2), but equal condition is in CR5.
   bdnzt 4 * cr5 + eq,*target* equivalent to **bc 8,22,***target*
   bdnzt 22,*target* would also work
4. Branch if bit 59 of CR is false.
   bf 27,*target* equivalent to **bc 4,27,***target*
   bf 4 * cr6 + so,*target* would also work
5. Same as (4), but set the link register. This is a form of conditional call.
   bfl 27,*target* equivalent to **bcl 4,27,***target*

Table 12 lists simplified mnemonics and syntax for **bc** and **bca** without LR updating.

Table 12. Simplified Mnemonics for bc and bca without LR Update

Branch Semantics	bc	Simplified Mnemonic	bca	Simplified Mnemonic
Branch unconditionally	—	—	—	—
Branch if condition true [1]	**bc 12,BI,**target	**bt BI,**target	**bca 12,BI,**target	**bta BI,**target

Table 12. Simplified Mnemonics for bc and bca without LR Update (continued)

Branch Semantics	bc	Simplified Mnemonic	bca	Simplified Mnemonic
Branch if condition false [1]	**bc 4,BI,**target	**bf BI,**target	**bca 4,BI,**target	**bfa BI,**target
Decrement CTR, branch if CTR ≠ 0	**bc 16,0,**target	**bdnz** target [2]	**bca 16,0,**target	**bdnza** target [2]
Decrement CTR, branch if CTR ≠ 0 and condition true	**bc 8,BI,**target	**bdnzt BI,**target	**bca 8,BI,**target	**bdnzta BI,**target
Decrement CTR, branch if CTR ≠ 0 and condition false	**bc 0,BI,**target	**bdnzf BI,**target	**bca 0,BI,**target	**bdnzfa BI,**target
Decrement CTR, branch if CTR = 0	**bc 18,0,**target	**bdz** target[2]	**bca 18,0,**target	**bdza** target[2]
Decrement CTR, branch if CTR = 0 and condition true	**bc 10,BI,**target	**bdzt BI,**target	**bca 10,BI,**target	**bdzta BI,**target
Decrement CTR, branch if CTR = 0 and condition false	**bc 2,BI,**target	**bdzf BI,**target	**bca 2,BI,**target	**bdzfa BI,**target

1 Instructions for which B0 is either 12 (branch if condition true) or 4 (branch if condition false) do not depend on the CTR value and can be alternately coded by incorporating the condition specified by the BI field, as described in Section 4.6, "Simplified Mnemonics that Incorporate CR Conditions (Eliminates BO and Replaces BI with **cr**S)."

2 Simplified mnemonics for branch instructions that do not test CR bits should specify only a target. Otherwise a programming error may occur.

Table 13 lists simplified mnemonics and syntax for **bclr** and **bcctr** without LR updating.

Table 13. Simplified Mnemonics for bclr and bcctr without LR Update

Branch Semantics	bclr	Simplified Mnemonic	bcctr	Simplified Mnemonic
Branch unconditionally	**bclr 20,0**	**blr** [1]	**bcctr 20,0**	**bctr** [1]
Branch if condition true [2]	**bclr 12,BI**	**btlr BI**	**bcctr 12,BI**	**btctr BI**
Branch if condition false [2]	**bclr 4,BI**	**bflr BI**	**bcctr 4,BI**	**bfctr BI**
Decrement CTR, branch if CTR ≠ 0	**bclr 16,BI**	**bdnzlr BI**	—	—
Decrement CTR, branch if CTR ≠ 0 and condition true	**bclr 8,BI**	**bdnztlr BI**	—	—
Decrement CTR, branch if CTR ≠ 0 and condition false	**bclr 0,BI**	**bdnzflr BI**	—	—
Decrement CTR, branch if CTR = 0	**bclr 18,0**	**bdzlr** [1]	—	—
Decrement CTR, branch if CTR = 0 and condition true	**bclr 8,BI**	**bdnztlr BI**	—	—
Decrement CTR, branch if CTR = 0 and condition false	**bclr 2,BI**	**bdzflr BI**	—	—

1 Simplified mnemonics for branch instructions that do not test a CR bit should not specify one; a programming error may occur.

2 Instructions for which B0 is 12 (branch if condition true) or 4 (branch if condition false) do not depend on a CTR value and can be alternately coded by incorporating the condition specified by the BI field. See Section 4.6, "Simplified Mnemonics that Incorporate CR Conditions (Eliminates BO and Replaces BI with **cr**S)."

Table 14 provides simplified mnemonics and syntax for **bcl** and **bcla**.

Table 14. Simplified Mnemonics for bcl and bcla with LR Update

Branch Semantics	bcl	Simplified Mnemonic	bcla	Simplified Mnemonic
Branch unconditionally	—	—	—	—
Branch if condition true [1]	**bcl 12,BI,**target	**btl BI,**target	**bcla 12,BI,**target	**btla BI,**target
Branch if condition false [1]	**bcl 4,BI,**target	**bfl BI,**target	**bcla 4,BI,**target	**bfla BI,**target
Decrement CTR, branch if CTR ≠ 0	**bcl 16,0,**target	**bdnzl** target [2]	**bcla 16,0,**target	**bdnzla** target [2]
Decrement CTR, branch if CTR ≠ 0 and condition true	**bcl 8,0,**target	**bdnztl BI,**target	**bcla 8,BI,**target	**bdnztla BI,**target
Decrement CTR, branch if CTR ≠ 0 and condition false	**bcl 0,BI,**target	**bdnzfl BI,**target	**bcla 0,BI,**target	**bdnzfla BI,**target
Decrement CTR, branch if CTR = 0	**bcl 18,BI,**target	**bdzl** target [2]	**bcla 18,BI,**target	**bdzla** target [2]
Decrement CTR, branch if CTR = 0 and condition true	**bcl 10,BI,**target	**bdztl BI,**target	**bcla 10,BI,**target	**bdztla BI,**target
Decrement CTR, branch if CTR = 0 and condition false	**bcl 2,BI,**target	**bdzfl BI,**target	**bcla 2,BI,**target	**bdzfla BI,**target

[1] Instructions for which B0 is either 12 (branch if condition true) or 4 (branch if condition false) do not depend on the CTR value and can be alternately coded by incorporating the condition specified by the BI field. See Section 4.6, "Simplified Mnemonics that Incorporate CR Conditions (Eliminates BO and Replaces BI with **cr**S)."

[2] Simplified mnemonics for branch instructions that do not test CR bits should specify only a target. A programming error may occur.

Table 15 provides simplified mnemonics and syntax for **bclrl** and **bcctrl** with LR updating.

Table 15. Simplified Mnemonics for bclrl and bcctrl with LR Update

Branch Semantics	bclrl	Simplified Mnemonic	bcctrl	Simplified Mnemonic
Branch unconditionally	**bclrl 20,0**	**blrl** [1]	**bcctrl 20,0**	**bctrl** [1]
Branch if condition true	**bclrl 12,BI**	**btlrl BI**	**bcctrl 12,BI**	**btctrl BI**
Branch if condition false	**bclrl 4,BI**	**bflrl BI**	**bcctrl 4,BI**	**bfctrl BI**
Decrement CTR, branch if CTR ≠ 0	**bclrl 16,0**	**bdnzlrl** [1]	—	—
Decrement CTR, branch if CTR ≠ 0 and condition true	**bclrl 8,BI**	**bdnztlrl BI**	—	—
Decrement CTR, branch if CTR ≠ 0 and condition false	**bclrl 0,BI**	**bdnzflrl BI**	—	—
Decrement CTR, branch if CTR = 0	**bclrl 18,0**	**bdzlrl** [1]	—	—
Decrement CTR, branch if CTR = 0 and condition true	**bclrl 10, BI**	**bdztlrl BI**	—	—
Decrement CTR, branch if CTR = 0 and condition false	**bclrl 2,BI**	**bdzflrl BI**	—	—

[1] Simplified mnemonics for branch instructions that do not test a CR bit should not specify one. A programming error may occur.

4.6 Simplified Mnemonics that Incorporate CR Conditions (Eliminates BO and Replaces BI with crS)

The mnemonics in Table 18 are variations of the branch-if-condition-true (BO = 12) and branch-if-condition-false (BO = 4) encodings. Because these instructions do not depend on the CTR, the true/false conditions specified by BO can be combined with the CR test bit specified by BI to create a different set of simplified mnemonics that eliminates the BO operand and the portion of the BI operand (BI[3–4]) that specifies one of the four possible test bits. However, the simplified mnemonic cannot specify in which of the eight CR fields the test bit falls, so the BI operand is replaced by a **cr**S operand.

The standard codes shown in Table 16 are used for the most common combinations of branch conditions. Note that for ease of programming, these codes include synonyms; for example, less than or equal (**le**) and not greater than (**ng**) achieve the same result.

NOTE

A CR field symbol, **cr0**–**cr7**, is used as the first operand after the simplified mnemonic. If the default, CR0, is used, no **cr**S is necessary,

Table 16. Standard Coding for Branch Conditions

Code	Description	Equivalent	Bit Tested
lt	Less than	—	LT
le	Less than or equal (equivalent to **ng**)	**ng**	GT
eq	Equal	—	EQ
ge	Greater than or equal (equivalent to **nl**)	**nl**	LT
gt	Greater than	—	GT
nl	Not less than (equivalent to **ge**)	**ge**	LT
ne	Not equal	—	EQ
ng	Not greater than (equivalent to **le**)	**le**	GT
so	Summary overflow	—	SO
ns	Not summary overflow	—	SO
un	Unordered (after floating-point comparison)	—	SO
nu	Not unordered (after floating-point comparison)	—	SO

Table 17 shows the syntax for simplified branch mnemonics that incorporate CR conditions. Here, **cr**S replaces a BI operand to specify only a CR field (because the specific CR bit within the field is now part of the simplified mnemonic. Note that the default is CR0; if no **cr**S is specified, CR0 is used.

Table 17. Branch Instructions and Simplified Mnemonics that Incorporate CR Conditions

Instruction	Standard Mnemonic	Syntax	Simplified Mnemonic	Syntax
Branch	**b** (**ba bl bla**)	target_addr	—	
Branch Conditional	**bc** (**bca bcl bcla**)	BO,BI,target_addr	**b***x* [1] (**b***x***a b***x***l b***x***la**)	**cr**S [2],target_addr
Branch Conditional to Link Register	**bclr** (**bclrl**)	BO,BI	**b***x***lr** (**b***x***lrl**)	**cr**S
Branch Conditional to Count Register	**bcctr** (**bcctrl**)	BO,BI	**b***x***ctr** (**b***x***ctrl**)	**cr**S

[1] *x* stands for one of the symbols in Table 16, where applicable.

[2] BI can be a numeric value or an expression as shown in Table 9.

Table 18 shows the simplified branch mnemonics incorporating conditions.

Table 18. Simplified Mnemonics with Comparison Conditions

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

Instructions using the mnemonics in Table 18 indicate the condition bit, but not the CR field. If no field is specified, CR0 is used. The CR field symbols defined in Table 9 (**cr0**–**cr7**) are used for this operand, as shown in examples 2–4 of Section 4.6.1, “Branch Simplified Mnemonics that Incorporate CR Conditions: Examples,” below.

4.6.1 Branch Simplified Mnemonics that Incorporate CR Conditions: Examples

The following examples use the simplified mnemonics shown in Table 18:

1. Branch if CR0 reflects not-equal condition.
   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 condition, setting the LR. This is a form of conditional call.

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

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

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

4.6.2 Branch Simplified Mnemonics that Incorporate CR Conditions: Listings

Table 19 shows simplified branch mnemonics and syntax for **bc** and **bca** without LR updating.

Table 19. Simplified Mnemonics for bc and bca without Comparison Conditions or LR Updating

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

[1] The value in the BI operand selects CR*n*[0], the LT bit.
[2] The value in the BI operand selects CR*n*[1], the GT bit.
[3] The value in the BI operand selects CR*n*[2], the EQ bit.
[4] The value in the BI operand selects CR*n*[3], the SO bit.

Table 20 shows simplified branch mnemonics and syntax for **bclr** and **bcctr** without LR updating.

Table 20. Simplified Mnemonics for bclr and bcctr without Comparison Conditions and LR Updating

Branch Semantics	bclr	Simplified Mnemonic	bcctr	Simplified Mnemonic
Branch if less than	**bclr 12,**BI [1],target	**bltlr cr**S,target	**bcctr 12,**BI[1],target	**bltctr cr**S,target
Branch if less than or equal	**bclr 4,**BI [2],target	**blelr cr**S,target	**bcctr 4,**BI[2],target	**blectr cr**S,target
Branch if not greater than		**bnglr cr**S,target		**bngctr cr**S,target
Branch if equal	**bclr 12,**BI [3],target	**beqlr cr**S,target	**bcctr 12,**BI[3],target	**beqctr cr**S,target
Branch if greater than or equal	**bclr 4,**BI[1],target	**bgelr cr**S,target	**bcctr 4,**BI[1],target	**bgectr cr**S,target
Branch if not less than		**bnllr cr**S,target		**bnlctr cr**S,target

Table 20. Simplified Mnemonics for bclr and bcctr without Comparison Conditions and LR Updating (continued)

Branch Semantics	bclr	Simplified Mnemonic	bcctr	Simplified Mnemonic
Branch if greater than	**bclr 12,**BI[2],target	**bgtlr crS,**target	**bcctr 12,**BI[2],target	**bgtctr crS,**target
Branch if not equal	**bclr 4,**BI[3],target	**bnelr crS,**target	**bcctr 4,**BI[3],target	**bnectr crS,**target
Branch if summary overflow	**bclr 12,**BI [4],target	**bsolr crS,**target	**bcctr 12,**BI[4],target	**bsoctr crS,**target
Branch if unordered		**bunlr crS,**target		**bunctr crS,**target
Branch if not summary overflow	**bclr 4,**BI[4],target	**bnslr crS,**target	**bcctr 4,**BI[4],target	**bnsctr crS,**target
Branch if not unordered		**bnulr crS,**target		**bnuctr crS,**target

1 The value in the BI operand selects CR*n*[0], the LT bit.
2 The value in the BI operand selects CR*n*[1], the GT bit.
3 The value in the BI operand selects CR*n*[2], the EQ bit.
4 The value in the BI operand selects CR*n*[3], the SO bit.

Table 21 shows simplified branch mnemonics and syntax for **bcl** and **bcla**.

Table 21. Simplified Mnemonics for bcl and bcla with Comparison Conditions and LR Updating

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

1 The value in the BI operand selects CR*n*[0], the LT bit.
2 The value in the BI operand selects CR*n*[1], the GT bit.
3 The value in the BI operand selects CR*n*[2], the EQ bit.
4 The value in the BI operand selects CR*n*[3], the SO bit.

Table 22 shows the simplified branch mnemonics and syntax for **bclrl** and **bcctrl** with LR updating.

Table 22. Simplified Mnemonics for bclrl and bcctrl with Comparison Conditions and LR Update

Branch Semantics	bclrl	Simplified Mnemonic	bcctrl	Simplified Mnemonic
Branch if less than	**bclrl 12,**BI [1],target	**bltlrl crS,**target	**bcctrl 12,**BI[1],target	**bltctrl crS,**target
Branch if less than or equal	**bclrl 4,**BI [2],target	**blelrl crS,**target	**bcctrl 4,**BI[2],target	**blectrl crS,**target
Branch if not greater than		**bnglrl crS,**target		**bngctrl crS,**target
Branch if equal	**bclrl 12,**BI [3],target	**beqlrl crS,**target	**bcctrl 12,**BI[3],target	**beqctrl crS,**target
Branch if greater than or equal	**bclrl 4,**BI[1],target	**bgelrl crS,**target	**bcctrl 4,**BI[1],target	**bgectrl crS,**target
Branch if not less than		**bnllrl crS,**target		**bnlctrl crS,**target
Branch if greater than	**bclrl 12,**BI2,target	**bgtlrl crS,**target	**bcctrl 12,**BI[2],target	**bgtctrl crS,**target
Branch if not equal	**bclrl 4,**BI[3],target	**bnelrl crS,**target	**bcctrl 4,**BI[3],target	**bnectrl crS,**target
Branch if summary overflow	**bclrl 12,**BI [4],target	**bsolrl crS,**target	**bcctrl 12,**BI[4],target	**bsoctrl crS,**target
Branch if unordered		**bunlrl crS,**target		**bunctrl crS,**target
Branch if not summary overflow	**bclrl 4,**BI[4],target	**bnslrl crS,**target	**bcctrl 4,**BI[4],target	**bnsctrl crS,**target
Branch if not unordered		**bnulrl crS,**target		**bnuctrl crS,**target

[1] The value in the BI operand selects CR*n*[0], the LT bit.
[2] The value in the BI operand selects CR*n*[1], the GT bit.
[3] The value in the BI operand selects CR*n*[2], the EQ bit.
[4] The value in the BI operand selects CR*n*[3], the SO bit.

5 Compare Word Simplified Mnemonics

In compare word instructions, the L operand indicates a word (L = 0) or double-word (L = 1). Simplified mnemonics in Table 23 eliminate the L operand for word comparisons.

Table 23. Word Compare Simplified Mnemonics

Operation	Simplified Mnemonic	Equivalent to:
Compare Word Immediate	**cmpwi crD,rA,**SIMM	**cmpi crD,0,rA,**SIMM
Compare Word	**cmpw crD,rA,rB**	**cmp crD,0,rA,rB**
Compare Logical Word Immediate	**cmplwi crD,rA,**UIMM	**cmpli crD,0,rA,**UIMM
Compare Logical Word	**cmplw crD,rA,rB**	**cmpl crD,0,rA,rB**

As with branch mnemonics, the **cr**D field of a compare instruction can be omitted if CR0 is used, as shown in examples 1 and 3 below. Otherwise, the target CR field must be specified as the first operand. The following examples use word compare mnemonics:

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

6 Condition Register Logical Simplified Mnemonics

The CR logical instructions, shown in Table 24, can be used to set, clear, copy, or invert a given CR bit. Simplified mnemonics allow these operations to be coded easily. Note that the symbols defined in Table 8 can be used to identify the CR bit.

Table 24. Condition Register Logical Simplified Mnemonics

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

Examples using the CR logical mnemonics follow:

1. Set CR[57].
   crset 25 equivalent to **creqv 25,25,25**
2. Clear CR0[SO].
   crclr so equivalent to **crxor 3,3,3**
3. Same as (2), but clear CR3[SO].
   crclr 4 * cr3 + so equivalent to **crxor 15,15,15**
4. Invert the CR0[EQ].
   crnot eq,eq equivalent to **crnor 2,2,2**
5. Same as (4), but CR4[EQ] is inverted and the result is placed into CR5[EQ].
   crnot 4 * cr5 + eq,4 * cr4 + eq equivalent to **crnor 22,18,18**

7 Trap Instructions Simplified Mnemonics

The codes in Table 25 have been adopted for the most common combinations of trap conditions.

Table 25. Standard Codes for Trap Instructions

Code	Description	TO 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

Table 25. Standard Codes for Trap Instructions (continued)

Code	Description	TO Encoding	<	>	=	<U [1]	>U [2]
lge	Logically greater than or equal	5	0	0	1	0	1
lgt	Logically greater than	1	0	0	0	0	1
lnl	Logically not less than	5	0	0	1	0	1
lng	Logically not greater than	6	0	0	1	1	0
—	Unconditional	31	1	1	1	1	1

[1] The symbol '<U' indicates an unsigned less-than evaluation is performed.
[2] The symbol '>U' indicates an unsigned greater-than evaluation is performed.

The mnemonics in Table 26 are variations of trap instructions, with the most useful TO values represented in the mnemonic rather than specified as a numeric operand.

Table 26. Trap Simplified Mnemonics

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

The following examples use the trap mnemonics shown in Table 26:

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

4. Trap unconditionally.
   trap equivalent to **tw 31,0,0**

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

The comparison results in five conditions that are ANDed with operand TO. If the result is not 0, the trap exception handler is invoked. See Table 27 for these conditions.

Table 27. 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

8 Simplified Mnemonics for Accessing SPRs

The **mtspr** and **mfspr** instructions specify a special-purpose register (SPR) as a numeric operand. Simplified mnemonics are provided that represent the SPR in the mnemonic rather than requiring it to be coded as a numeric operand. The pattern for **mtspr** and **mfspr** simplified mnemonics is straightforward: replace the **-spr** portion of the mnemonic with the abbreviation for the spr (for example XER, SRR0, or LR), eliminate the SPRN operand, leaving the source or destination GPR operand, **r**S or **r**D.

Following are examples using the SPR simplified mnemonics:

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

The examples above show simplified mnemonics for accessing SPRs defined by the AIM version of the PowerPC architecture; however, the same formula is used for Book E, EIS, and implementation-specific SPRs, as shown in the following examples:

1. Copy the contents of **r**S to CSRR0.
   mtcsrr0 rS equivalent to **mtspr 58,r**S
2. Copy the contents of IVOR0 to **r**S.
   mfivor0 rD equivalent to **mfspr r**D**,400**
3. Copy the contents of **r**S to the MAS1.
   mtmas1 rS equivalent to **mtspr 625,r**S

There is an additional simplified mnemonic formula for accessing IBATs, DBATs, and SPRGs, although not all of these more complicated simplified mnemonics are supported by all assemblers. These are shown in Table 28 along with the equivalent simplified mnemonic using the formula described above.

Table 28. Additional Simplified Mnemonics for Accessing IBATs, DBATs, and SPRGs

SPR	Move to SPR		Move from SPR	
	Simplified Mnemonic	Equivalent to	Simplified Mnemonic	Equivalent to
DBAT register, lower	**mtdbatl** *n*,**r**S	**mtspr** 537 + (2 * *n*),**r**S	**mfdbatl r**D,*n*	**mfspr r**D,537 + (2 * *n*)
	mtdbatl*n*,**r**S		**mfdbatl***n* **r**D	
DBAT register, lower	**mtdbatu** *n*,**r**S	**mtspr** 536 + (2 **n*),**r**S	**mfdbatu r**D,*n*	**mfspr r**D,536 + (2 **n*)
	mtdbatu*n*,**r**S		**mfdbatu***n* **r**D	
IBAT register, lower	**mtibatl** *n*,**r**S	**mtspr** 529 + (2 * *n*),**r**S	**mfibatl r**D,*n*	**mfspr r**D,529 + (2 * *n*)
	mtibatl*n*,**r**S		**mfibatl***n* **r**D	
IBAT register, upper	**mtibatu** *n*,**r**S	**mtspr** 528 + (2 * *n*),**r**S	**mfibatu r**D,*n*	**mfspr r**D,528 + (2 * *n*)
	mtibatu*n*,**r**S		**mfibatu***n* **r**D	
SPRGs	**mtsprg** *n*,**r**S	**mtspr** 272 + *n*,**r**S	**mfsprg r**D,*n*	**mfspr r**D,272 + *n*
	mtsprg*n*,**r**S		**mfsprg***n* **r**D	

9 AltiVec Simplified Mnemonics

Simplified mnemonics are provided for the Data Stream Stop (**dss)** instruction so that it can be coded with the all streams indicator as part of the mnemonic. These are shown as examples with the instructions in Table 29.

Table 29. Altivec Data Stream Stop (dss) Simplified Mnemonics

Operation	Simplified Mnemonic	Equivalent to
Data Stream Stop (one stream)	**dss** STRM	**dss** STRM,**0**
Data Stream Stop All	**dssall**	**dss 0**,**1**

Simplified mnemonics for two vector instructions are also supported, as shown in Table 30.

Table 30. Altivec Vector Simplified Mnemonics

Operation	Simplified Mnemonic	Equivalent to
Vector Move Register	**vmr v**D,**v**S	**vor v**D,**v**S,**v**S
Vector Logical Not	**vnot v**D,**v**S	**vnor v**D,**v**S,**v**S

10 Recommended Simplified Mnemonics

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

10.1 No-Op (nop)

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

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

10.2 Load Immediate (li)

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

1. Load a 16-bit signed immediate value into **r**D.

li rD**,**value	equivalent to	**addi r**D**,0,**value

2. Load a 16-bit signed immediate value, shifted left by 16 bits, into **r**D.

lis rD**,**value	equivalent to	**addis r**D**,0,**value

10.3 Load Address (la)

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

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

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

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

10.4 Move Register (mr)

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

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

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

10.5 Complement Register (not)

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

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

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

10.6 Move to Condition Register (mtcr)

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

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

11 EIS-Specific Simplified Mnemonics

This section describes simplified mnemonics for instructions defines by auxiliary processing units (APUs) defined as part of the Freescale Book E implementation standards (EIS).

11.1 Integer Select (isel)

The following mnemonics simplify the most common variants of the **isel** instruction that access CR0:

Integer Select Less Than		
isellt rD,**r**A,**r**B	equivalent to	**isel r**D,**r**A,**r**B,**0**
Integer Select Greater Than		
iselgt rD,**r**A,**r**B	equivalent to	**isel r**D,**r**A,**r**B,**1**
Integer Select Equal		
iseleq rD,**r**A,**r**B	equivalent to	**isel r**D,**r**A,**r**B,**2**

11.2 SPE Mnemonics

The following mnemonic handles moving of the full 64-bit SPE GPR:

Vector Move		
evmr rD,**r**A	equivalent to	**evor r**D,**r**A,**r**A

The following mnemonic performs a complement register:

Vector Not		
evnot rD,**r**A	equivalent to	**evnor r**D,**r**A,**r**A

12 Comprehensive List of Simplified Mnemonics

Table 31 lists simplified mnemonics. Note that compiler designers may implement additional simplified mnemonics not listed here.

Table 31. Simplified Mnemonics

Simplified Mnemonic	Mnemonic	Instruction
bctr [1]	**bcctr 20,0**	Branch unconditionally (**bcctr** without LR update)
bctrl [1]	**bcctrl 20,0**	Branch unconditionally (**bcctrl** with LR Update)
bdnz target [1]	**bc 16,0,**target	Decrement CTR, branch if CTR ≠ 0 (**bc** without LR update)
bdnza target [1]	**bca 16,0,**target	Decrement CTR, branch if CTR ≠ 0 (**bca** without LR update)
bdnzf BI,target	**bc 0,BI,**target	Decrement CTR, branch if CTR ≠ 0 and condition false (**bc** without LR update)
bdnzfa BI,target	**bca 0,BI,**target	Decrement CTR, branch if CTR ≠ 0 and condition false (**bca** without LR update)
bdnzfl BI,target	**bcl 0,BI,**target	Decrement CTR, branch if CTR ≠ 0 and condition false (**bcl** with LR update)
bdnzfla BI,target	**bcla 0,BI,**target	Decrement CTR, branch if CTR ≠ 0 and condition false (**bcla** with LR update)
bdnzflr BI	**bclr 0,BI**	Decrement CTR, branch if CTR ≠ 0 and condition false (**bclr** without LR update)
bdnzflrl BI	**bclrl 0,BI**	Decrement CTR, branch if CTR ≠ 0 and condition false (**bclrl** with LR Update)
bdnzl target [1]	**bcl 16,0,**target	Decrement CTR, branch if CTR ≠ 0 (**bcl** with LR update)
bdnzla target [1]	**bcla 16,0,**target	Decrement CTR, branch if CTR ≠ 0 (**bcla** with LR update)
bdnzlr BI	**bclr 16,BI**	Decrement CTR, branch if CTR ≠ 0 (**bclr** without LR update)
bdnzlrl [1]	**bclrl 16,0**	Decrement CTR, branch if CTR ≠ 0 (**bclrl** with LR Update)
bdnzt BI,target	**bc 8,BI,**target	Decrement CTR, branch if CTR ≠ 0 and condition true (**bc** without LR update)
bdnzta BI,target	**bca 8,BI,**target	Decrement CTR, branch if CTR ≠ 0 and condition true (**bca** without LR update)
bdnztl BI,target	**bcl 8,0,**target	Decrement CTR, branch if CTR ≠ 0 and condition true (**bcl** with LR update)
bdnztla BI,target	**bcla 8,BI,**target	Decrement CTR, branch if CTR ≠ 0 and condition true (**bcla** with LR update)
bdnztlr BI	**bclr 8,BI**	Decrement CTR, branch if CTR ≠ 0 and condition true (**bclr** without LR update)

Table 31. Simplified Mnemonics (continued)

Simplified Mnemonic	Mnemonic	Instruction
bdnztlr BI	**bclr 8,BI**	Decrement CTR, branch if CTR = 0 and condition true (**bclr** without LR update)
bdnztlrl BI	**bclrl 8,BI**	Decrement CTR, branch if CTR ≠ 0 and condition true (**bclrl** with LR Update)
bdz target [1]	**bc 18,0,**target	Decrement CTR, branch if CTR = 0 (**bc** without LR update)
bdza target [1]	**bca 18,0,**target	Decrement CTR, branch if CTR = 0 (**bca** without LR update)
bdzf BI,target	**bc 2,BI,**target	Decrement CTR, branch if CTR = 0 and condition false (**bc** without LR update)
bdzfa BI,target	**bca 2,BI,**target	Decrement CTR, branch if CTR = 0 and condition false (**bca** without LR update)
bdzfl BI,target	**bcl 2,BI,**target	Decrement CTR, branch if CTR = 0 and condition false (**bcl** with LR update)
bdzfla BI,target	**bcla 2,BI,**target	Decrement CTR, branch if CTR = 0 and condition false (**bcla** with LR update)
bdzflr BI	**bclr 2,BI**	Decrement CTR, branch if CTR = 0 and condition false (**bclr** without LR update)
bdzflrl BI	**bclrl 2,BI**	Decrement CTR, branch if CTR = 0 and condition false (**bclrl** with LR Update)
bdzl target [1]	**bcl 18,BI,**target	Decrement CTR, branch if CTR = 0 (**bcl** with LR update)
bdzla target [1]	**bcla 18,BI,**target	Decrement CTR, branch if CTR = 0 (**bcla** with LR update)
bdzlr [1]	**bclr 18,0**	Decrement CTR, branch if CTR = 0 (**bclr** without LR update)
bdzlrl [1]	**bclrl 18,0**	Decrement CTR, branch if CTR = 0 (**bclrl** with LR Update)
bdzt BI,target	**bc 10,BI,**target	Decrement CTR, branch if CTR = 0 and condition true (**bc** without LR update)
bdzta BI,target	**bca 10,BI,**target	Decrement CTR, branch if CTR = 0 and condition true (**bca** without LR update)
bdztl BI,target	**bcl 10,BI,**target	Decrement CTR, branch if CTR = 0 and condition true (**bcl** with LR update)
bdztla BI,target	**bcla 10,BI,**target	Decrement CTR, branch if CTR = 0 and condition true (**bcla** with LR update)
bdztlrl BI	**bclrl 10,BI**	Decrement CTR, branch if CTR = 0 and condition true (**bclrl** with LR Update)
beq crS,target	**bc 12,**BI [2],target	Branch if equal (**bc** without comparison conditions or LR updating)
beqa crS,target	**bca 12,**BI[2],target	Branch if equal (**bca** without comparison conditions or LR updating)

Table 31. Simplified Mnemonics (continued)

Simplified Mnemonic	Mnemonic	Instruction
beqctr crS,target	**bcctr 12,**BI[2],target	Branch if equal (**bcctr** without comparison conditions and LR updating)
beqctrl crS,target	**bcctrl 12,**BI[2],target	Branch if equal (**bcctrl** with comparison conditions and LR update)
beql crS,target	**bcl 12,**BI[2],target	Branch if equal (**bcl** with comparison conditions and LR updating)
beqla crS,target	**bcla 12,**BI[2],target	Branch if equal (**bcla** with comparison conditions and LR updating)
beqlr crS,target	**bclr 12,**BI[2],target	Branch if equal (**bclr** without comparison conditions and LR updating)
beqlrl crS,target	**bclrl 12,**BI[2],target	Branch if equal (**bclrl** with comparison conditions and LR update)
bf BI,target	**bc 4,BI,**target	Branch if condition false [3] (**bc** without LR update)
bfa BI,target	**bca 4,BI,**target	Branch if condition false [3] (**bca** without LR update)
bfctr BI	**bcctr 4,BI**	Branch if condition false [3] (**bcctr** without LR update)
bfctrl BI	**bcctrl 4,BI**	Branch if condition false [3](**bcctrl** with LR Update)
bfl BI,target	**bcl 4,BI,**target	Branch if condition false [3] (**bcl** with LR update)
bfla BI,target	**bcla 4,BI,**target	Branch if condition false [3] (**bcla** with LR update)
bflr BI	**bclr 4,BI**	Branch if condition false [3] (**bclr** without LR update)
bflrl BI	**bclrl 4,BI**	Branch if condition false [3](**bclrl** with LR Update)
bge crS,target	**bc 4,**BI [4],target	Branch if greater than or equal (**bc** without comparison conditions or LR updating)
bgea crS,target	**bca 4,**BI[4],target	Branch if greater than or equal (**bca** without comparison conditions or LR updating)
bgectr crS,target	**bcctr 4,**BI[4],target	Branch if greater than or equal (**bcctr** without comparison conditions and LR updating)
bgectrl crS,target	**bcctrl 4,**BI[4],target	Branch if greater than or equal (**bcctrl** with comparison conditions and LR update)
bgel crS,target	**bcl 4,**BI[4],target	Branch if greater than or equal (**bcl** with comparison conditions and LR updating)
bgela crS,target	**bcla 4,**BI[4],target	Branch if greater than or equal (**bcla** with comparison conditions and LR updating)
bgelr crS,target	**bclr 4,**BI[4],target	Branch if greater than or equal (**bclr** without comparison conditions and LR updating)
bgelrl crS,target	**bclrl 4,**BI[4],target	Branch if greater than or equal (**bclrl** with comparison conditions and LR update)
bgt crS,target	**bc 12,**BI [5],target	Branch if greater than (**bc** without comparison conditions or LR updating)

Table 31. Simplified Mnemonics (continued)

Simplified Mnemonic	Mnemonic	Instruction
bgta crS,target	**bca 12,**BI[5],target	Branch if greater than (**bca** without comparison conditions or LR updating)
bgtctr crS,target	**bcctr 12,**BI[5],target	Branch if greater than (**bcctr** without comparison conditions and LR updating)
bgtctrl crS,target	**bcctrl 12,**BI[5],target	Branch if greater than (**bcctrl** with comparison conditions and LR update)
bgtl crS,target	**bcl 12,**BI[5],target	Branch if greater than (**bcl** with comparison conditions and LR updating)
bgtla crS,target	**bcla 12,**BI[5],target	Branch if greater than (**bcla** with comparison conditions and LR updating)
bgtlr crS,target	**bclr 12,**BI[5],target	Branch if greater than (**bclr** without comparison conditions and LR updating)
bgtlrl crS,target	**bclrl 12,**BI[5],target	Branch if greater than (**bclrl** with comparison conditions and LR update)
ble crS,target	**bc 4,**BI[5],target	Branch if less than or equal (**bc** without comparison conditions or LR updating)
blea crS,target	**bca 4,**BI[5],target	Branch if less than or equal (**bca** without comparison conditions or LR updating)
blectr crS,target	**bcctr 4,**BI[5],target	Branch if less than or equal (**bcctr** without comparison conditions and LR updating)
blectrl crS,target	**bcctrl 4,**BI[5],target	Branch if less than or equal (**bcctrl** with comparison conditions and LR update)
blel crS,target	**bcl 4,**BI[5],target	Branch if less than or equal (**bcl** with comparison conditions and LR updating)
blela crS,target	**bcla 4,**BI[5],target	Branch if less than or equal (**bcla** with comparison conditions and LR updating)
blelr crS,target	**bclr 4,**BI[5],target	Branch if less than or equal (**bclr** without comparison conditions and LR updating)
blelrl crS,target	**bclrl 4,**BI[5],target	Branch if less than or equal (**bclrl** with comparison conditions and LR update)
blr [1]	**bclr 20,0**	Branch unconditionally (**bclr** without LR update)
blrl [1]	**bclrl 20,0**	Branch unconditionally (**bclrl** with LR Update)
blt crS,target	**bc 12,**BI,target	Branch if less than (**bc** without comparison conditions or LR updating)
blta crS,target	**bca 12,**BI[4],target	Branch if less than (**bca** without comparison conditions or LR updating)
bltctr crS,target	**bcctr 12,**BI[4],target	Branch if less than (**bcctr** without comparison conditions and LR updating)
bltctrl crS,target	**bcctrl 12,**BI[4],target	Branch if less than (**bcctrl** with comparison conditions and LR update)

Table 31. Simplified Mnemonics (continued)

Simplified Mnemonic	Mnemonic	Instruction
bltl crS,target	**bcl 12,**BI[4],target	Branch if less than (**bcl** with comparison conditions and LR updating)
bltla crS,target	**bcla 12,**BI[4],target	Branch if less than (**bcla** with comparison conditions and LR updating)
bltlr crS,target	**bclr 12,**BI[4],target	Branch if less than (**bclr** without comparison conditions and LR updating)
bltlrl crS,target	**bclrl 12,**BI[4],target	Branch if less than (**bclrl** with comparison conditions and LR update)
bne crS,target	**bc 4,**BI[3],target	Branch if not equal (**bc** without comparison conditions or LR updating)
bnea crS,target	**bca 4,**BI[3],target	Branch if not equal (**bca** without comparison conditions or LR updating)
bnectr crS,target	**bcctr 4,**BI[3],target	Branch if not equal (**bcctr** without comparison conditions and LR updating)
bnectrl crS,target	**bcctrl 4,**BI[3],target	Branch if not equal (**bcctrl** with comparison conditions and LR update)
bnel crS,target	**bcl 4,**BI[3],target	Branch if not equal (**bcl** with comparison conditions and LR updating)
bnela crS,target	**bcla 4,**BI[3],target	Branch if not equal (**bcla** with comparison conditions and LR updating)
bnelr crS,target	**bclr 4,**BI[3],target	Branch if not equal (**bclr** without comparison conditions and LR updating)
bnelrl crS,target	**bclrl 4,**BI[3],target	Branch if not equal (**bclrl** with comparison conditions and LR update)
bng crS,target	**bc 4,**BI[5],target	Branch if not greater than (**bc** without comparison conditions or LR updating)
bnga crS,target	**bca 4,**BI[5],target	Branch if not greater than (**bca** without comparison conditions or LR updating)
bngctr crS,target	**bcctr 4,**BI[5],target	Branch if not greater than (**bcctr** without comparison conditions and LR updating)
bngctrl crS,target	**bcctrl 4,**BI[5],target	Branch if not greater than (**bcctrl** with comparison conditions and LR update)
bngl crS,target	**bcl 4,**BI[5],target	Branch if not greater than (**bcl** with comparison conditions and LR updating)
bngla crS,target	**bcla 4,**BI[5],target	Branch if not greater than (**bcla** with comparison conditions and LR updating)
bnglr crS,target	**bclr 4,**BI[5],target	Branch if not greater than (**bclr** without comparison conditions and LR updating)
bnglrl crS,target	**bclrl 4,**BI[5],target	Branch if not greater than (**bclrl** with comparison conditions and LR update)
bnl crS,target	**bc 4,**BI[4],target	Branch if not less than (**bc** without comparison conditions or LR updating)

Table 31. Simplified Mnemonics (continued)

Simplified Mnemonic	Mnemonic	Instruction
bnla crS,target	**bca 4,**BI[4],target	Branch if not less than (**bca** without comparison conditions or LR updating)
bnlctr crS,target	**bcctr 4,**BI[4],target	Branch if not less than (**bcctr** without comparison conditions and LR updating)
bnlctrl crS,target	**bcctrl 4,**BI[4],target	Branch if not less than (**bcctrl** with comparison conditions and LR update)
bnll crS,target	**bcl 4,**BI[4],target	Branch if not less than (**bcl** with comparison conditions and LR updating)
bnlla crS,target	**bcla 4,**BI[4],target	Branch if not less than (**bcla** with comparison conditions and LR updating)
bnllr crS,target	**bclr 4,**BI[4],target	Branch if not less than (**bclr** without comparison conditions and LR updating)
bnllrl crS,target	**bclrl 4,**BI[4],target	Branch if not less than (**bclrl** with comparison conditions and LR update)
bns crS,target	**bc 4,**BI [6],target	Branch if not summary overflow (**bc** without comparison conditions or LR updating)
bnsa crS,target	**bca 4,**BI[6],target	Branch if not summary overflow (**bca** without comparison conditions or LR updating)
bnsctr crS,target	**bcctr 4,**BI[6],target	Branch if not summary overflow (**bcctr** without comparison conditions and LR updating)
bnsctrl crS,target	**bcctrl 4,**BI[6],target	Branch if not summary overflow (**bcctrl** with comparison conditions and LR update)
bnsl crS,target	**bcl 4,**BI[6],target	Branch if not summary overflow (**bcl** with comparison conditions and LR updating)
bnsla crS,target	**bcla 4,**BI[6],target	Branch if not summary overflow (**bcla** with comparison conditions and LR updating)
bnslr crS,target	**bclr 4,**BI[6],target	Branch if not summary overflow (**bclr** without comparison conditions and LR updating)
bnslrl crS,target	**bclrl 4,**BI[6],target	Branch if not summary overflow (**bclrl** with comparison conditions and LR update)
bnu crS,target	**bc 4,**BI[6],target	Branch if not unordered (**bc** without comparison conditions or LR updating)
bnua crS,target	**bca 4,**BI[6],target	Branch if not unordered (**bca** without comparison conditions or LR updating)
bnuctr crS,target	**bcctr 4,**BI[6],target	Branch if not unordered (**bcctr** without comparison conditions and LR updating)
bnuctrl crS,target	**bcctrl 4,**BI[6],target	Branch if not unordered (**bcctrl** with comparison conditions and LR update)
bnul crS,target	**bcl 4,**BI[6],target	Branch if not unordered (**bcl** with comparison conditions and LR updating)
bnula crS,target	**bcla 4,**BI[6],target	Branch if not unordered (**bcla** with comparison conditions and LR updating)

Table 31. Simplified Mnemonics (continued)

Simplified Mnemonic	Mnemonic	Instruction
bnulr crS,target	**bclr 4,**BI[6],target	Branch if not unordered (**bclr** without comparison conditions and LR updating)
bnulrl crS,target	**bclrl 4,**BI[6],target	Branch if not unordered (**bclrl** with comparison conditions and LR update)
bso crS,target	**bc 12,**BI[6],target	Branch if summary overflow (**bc** without comparison conditions or LR updating)
bsoa crS,target	**bca 12,**BI[6],target	Branch if summary overflow (**bca** without comparison conditions or LR updating)
bsoctr crS,target	**bcctr 12,**BI[6],target	Branch if summary overflow (**bcctr** without comparison conditions and LR updating)
bsoctrl crS,target	**bcctrl 12,**BI[6],target	Branch if summary overflow (**bcctrl** with comparison conditions and LR update)
bsol crS,target	**bcl 12,**BI[6],target	Branch if summary overflow (**bcl** with comparison conditions and LR updating)
bsola crS,target	**bcla 12,**BI[6],target	Branch if summary overflow (**bcla** with comparison conditions and LR updating)
bsolr crS,target	**bclr 12,**BI[6],target	Branch if summary overflow (**bclr** without comparison conditions and LR updating)
bsolrl crS,target	**bclrl 12,**BI[6],target	Branch if summary overflow (**bclrl** with comparison conditions and LR update)
bt BI,target	**bc 12,BI,**target	Branch if condition true[3] (**bc** without LR update)
bta BI,target	**bca 12,BI,**target	Branch if condition true[3] (**bca** without LR update)
btctr BI	**bcctr 12,BI**	Branch if condition true [3] (**bcctr** without LR update)
btctrl BI	**bcctrl 12,BI**	Branch if condition true [3] (**bcctrl** with LR Update)
btl BI,target	**bcl 12,BI,**target	Branch if condition true [3] (**bcl** with LR update)
btla BI,target	**bcla 12,BI,**target	Branch if condition true [3] (**bcla** with LR update)
btlr BI	**bclr 12,BI**	Branch if condition true [3] (**bclr** without LR update)
btlrl BI	**bclrl 12,BI**	Branch if condition true [3] (**bclrl** with LR Update)
bun crS,target	**bc 12,**BI[6],target	Branch if unordered (**bc** without comparison conditions or LR updating)
buna crS,target	**bca 12,**BI[6],target	Branch if unordered (**bca** without comparison conditions or LR updating)
bunctr crS,target	**bcctr 12,**BI[6],target	Branch if unordered (**bcctr** without comparison conditions and LR updating)
bunctrl crS,target	**bcctrl 12,**BI[6],target	Branch if unordered (**bcctrl** with comparison conditions and LR update)
bunl crS,target	**bcl 12,**BI[6],target	Branch if unordered (**bcl** with comparison conditions and LR updating)
bunla crS,target	**bcla 12,**BI[6],target	Branch if unordered (**bcla** with comparison conditions and LR updating)

Table 31. Simplified Mnemonics (continued)

Simplified Mnemonic	Mnemonic	Instruction
bunlr crS,target	**bclr 12,**BI[6],target	Branch if unordered (**bclr** without comparison conditions and LR updating)
bunlrl crS,target	**bclrl 12,**BI[6],target	Branch if unordered (**bclrl** with comparison conditions and LR update)
clrlslwi rA,rS,*b*,*n* (*n* ≤ *b* ≤ 31)	**rlwinm** rA,rS,*n*,*b* – *n*,31 – *n*	Clear left and shift left word immediate
clrlwi rA,rS,*n* (*n* < 32)	**rlwinm** rA,rS,**0**,*n*,**31**	Clear left word immediate
clrrwi rA,rS,*n* (*n* < 32)	**rlwinm** rA,rS,**0**,**0**,31 – *n*	Clear right word immediate
cmplw crD,rA,rB	**cmpl** crD,**0**,rA,rB	Compare logical word
cmplwi crD,rA,UIMM	**cmpli** crD,**0**,rA,UIMM	Compare logical word immediate
cmpw crD,rA,rB	**cmp** crD,**0**,rA,rB	Compare word
cmpwi crD,rA,SIMM	**cmpi** crD,**0**,rA,SIMM	Compare word immediate
crclr bx	**crxor** bx,bx,bx	Condition register clear
crmove bx,by	**cror** bx,by,by	Condition register move
crnot bx,by	**crnor** bx,by,by	Condition register not
crset bx	**creqv** bx,bx,bx	Condition register set
dss STRM	**dss** STRM,**0**	Data Stream Stop (one stream)
dssall	**dss 0**,**1**	Data Stream Stop All
evmr rD,rA	**evor** rD,rA,rA	Vector Move Register
evnot rD,rA	**evnor** rD,rA,rA	Vector Complement Register
evsubiw rD,rB,UIMM	**evsubifw** rD,UIMM,rB	Vector subtract word immediate
evsubw rD,rB,rA	**evsubfw** rD,rA,rB	Vector subtract word
extlwi rA,rS,*n*,*b* (*n* > 0)	**rlwinm** rA,rS,*b*,**0**,*n* – 1	Extract and left justify word immediate
extrwi rA,rS,*n*,*b* (*n* > 0)	**rlwinm** rA,rS,*b* + *n*,32 – *n*,**31**	Extract and right justify word immediate
inslwi rA,rS,*n*,*b* (*n* > 0)	**rlwimi** rA,rS,32 – *b*,*b*,(*b* + *n*) – 1	Insert from left word immediate
insrwi rA,rS,*n*,*b* (*n* > 0)	**rlwimi** rA,rS,32 – (*b* + *n*),*b*,(*b* + *n*) – 1	Insert from right word immediate
iseleq rD,rA,rB	**isel** rD,rA,rB,**2**	Integer Select Equal
iselgt rD,rA,rB	**isel** rD,rA,rB,**1**	Integer Select Greater Than
isellt rD,rA,rB	**isel** rD,rA,rB,**0**	Integer Select Less Than
la rD,d(rA)	**addi** rD,rA,d	Load address
li rD,value	**addi** rD,**0**,value	Load immediate
lis rD,value	**addis** rD,**0**,value	Load immediate signed
mf*spr* rD	**mfspr** rD,SPRN	Move from SPR (see Section 8, “Simplified Mnemonics for Accessing SPRs.”)
mr rA,rS	**or** rA,rS,rS	Move register
mtcr rS	**mtcrf** 0xFF,rS	Move to Condition Register

Table 31. Simplified Mnemonics (continued)

Simplified Mnemonic	Mnemonic	Instruction
mt*spr* rS	**mfspr** SPRN,rS	Move to SPR (see Section 8, “Simplified Mnemonics for Accessing SPRs.”)
nop	**ori 0,0,0**	No-op
not rA,rS	**nor** rA,rS,rS	NOT
not rA,rS	**nor** rA,rS,rS	Complement register
rotlw rA,rS,rB	**rlwnm** rA,rS,rB,**0,31**	Rotate left word
rotlwi rA,rS,*n*	**rlwinm** rA,rS,*n*,**0,31**	Rotate left word immediate
rotrwi rA,rS,*n*	**rlwinm** rA,rS,32 – *n*,**0,31**	Rotate right word immediate
slwi rA,rS,*n* (*n* < 32)	**rlwinm** rA,rS,*n*,**0**,31 – *n*	Shift left word immediate
srwi rA,rS,*n* (*n* < 32)	**rlwinm** rA,rS,32 – *n*,*n*,**31**	Shift right word immediate
sub rD,rA,rB	**subf** rD,rB,rA	Subtract from
subc rD,rA,rB	**subfc** rD,rB,rA	Subtract from carrying
subi rD,rA,value	**addi** rD,rA,–value	Subtract immediate
subic rD,rA,value	**addic** rD,rA,–value	Subtract immediate carrying
subic. rD,rA,value	**addic.** rD,rA,–value	Subtract immediate carrying
subis rD,rA,value	**addis** rD,rA,–value	Subtract immediate signed
tweq rA,SIMM	**tw 4,**rA,SIMM	Trap if equal
tweqi rA,SIMM	**twi 4,**rA,SIMM	Trap immediate if equal
twge rA,SIMM	**tw 12,**rA,SIMM	Trap if greater than or equal
twgei rA,SIMM	**twi 12,**rA,SIMM	Trap immediate if greater than or equal
twgt rA,SIMM	**tw 8,**rA,SIMM	Trap if greater than
twgti rA,SIMM	**twi 8,**rA,SIMM	Trap immediate if greater than
twle rA,SIMM	**tw 20,**rA,SIMM	Trap if less than or equal
twlei rA,SIMM	**twi 20,**rA,SIMM	Trap immediate if less than or equal
twlge rA,SIMM	**tw 12,**rA,SIMM	Trap if logically greater than or equal
twlgei rA,SIMM	**twi 12,**rA,SIMM	Trap immediate if logically greater than or equal
twlgt rA,SIMM	**tw 1,**rA,SIMM	Trap if logically greater than
twlgti rA,SIMM	**twi 1,**rA,SIMM	Trap immediate if logically greater than
twlle rA,SIMM	**tw 6,**rA,SIMM	Trap if logically less than or equal
twllei rA,SIMM	**twi 6,**rA,SIMM	Trap immediate if logically less than or equal
twllt rA,SIMM	**tw 2,**rA,SIMM	Trap if logically less than
twllti rA,SIMM	**twi 2,**rA,SIMM	Trap immediate if logically less than
twlng rA,SIMM	**tw 6,**rA,SIMM	Trap if logically not greater than
twlngi rA,SIMM	**twi 6,**rA,SIMM	Trap immediate if logically not greater than

Table 31. Simplified Mnemonics (continued)

Simplified Mnemonic	Mnemonic	Instruction
twlnl **r**A,SIMM	**tw 5,r**A,SIMM	Trap if logically not less than
twlnli **r**A,SIMM	**twi 5,r**A,SIMM	Trap immediate if logically not less than
twlt **r**A,SIMM	**tw 16,r**A,SIMM	Trap if less than
twlti **r**A,SIMM	**twi 16,r**A,SIMM	Trap immediate if less than
twne **r**A,SIMM	**tw 24,r**A,SIMM	Trap if not equal
twnei **r**A,SIMM	**twi 24,r**A,SIMM	Trap immediate if not equal
twng **r**A,SIMM	**tw 20,r**A,SIMM	Trap if not greater than
twngi **r**A,SIMM	**twi 20,r**A,SIMM	Trap immediate if not greater than
twnl **r**A,SIMM	**tw 12,r**A,SIMM	Trap if not less than
twnli **r**A,SIMM	**twi 12,r**A,SIMM	Trap immediate if not less than
vmr **v**D,**v**S	**vor** **v**D,**v**S,**v**S	Vector Move Register
vnot **v**D,**v**S	**vnor** **v**D,**v**S,**v**S	Vector Not

1 Simplified mnemonics for branch instructions that do not test a CR bit should not specify one; a programming error may occur.

2 The value in the BI operand selects CR*n*[2], the EQ bit.

3 Instructions for which B0 is either 12 (branch if condition true) or 4 (branch if condition false) do not depend on the CTR value and can be alternately coded by incorporating the condition specified by the BI field, as described in Section 4.6, “Simplified Mnemonics that Incorporate CR Conditions (Eliminates BO and Replaces BI with crS).”

4 The value in the BI operand selects CR*n*[0], the LT bit.

5 The value in the BI operand selects CR*n*[1], the GT bit.

6 The value in the BI operand selects CR*n*[3], the SO bit.

THIS PAGE INTENTIONALLY LEFT BLANK

How to Reach Us:

Home Page:
www.freescale.com

E-mail:
support@freescale.com

USA/Europe or Locations Not Listed:
Freescale Semiconductor
Technical Information Center, CH370
1300 N. Alma School Road
Chandler, Arizona 85224
+1-800-521-6274 or +1-480-768-2130
support@freescale.com

Europe, Middle East, and Africa:
Freescale Halbleiter Deutschland GmbH
Technical Information Center
Schatzbogen 7
81829 Muenchen, Germany
+44 1296 380 456 (English)
+46 8 52200080 (English)
+49 89 92103 559 (German)
+33 1 69 35 48 48 (French)
support@freescale.com

Japan:
Freescale Semiconductor Japan Ltd.
Headquarters
ARCO Tower 15F
1-8-1, Shimo-Meguro, Meguro-ku,
Tokyo 153-0064
Japan
0120 191014 or +81 3 5437 9125
support.japan@freescale.com

Asia/Pacific:
Freescale Semiconductor Hong Kong Ltd.
Technical Information Center
2 Dai King Street
Tai Po Industrial Estate
Tai Po, N.T., Hong Kong
+800 2666 8080
support.asia@freescale.com

For Literature Requests Only:
Freescale Semiconductor Literature Distribution Center
P.O. Box 5405
Denver, Colorado 80217
1-800-441-2447 or 303-675-2140
Fax: 303-675-2150
LDCForFreescaleSemiconductor@hibbertgroup.com

freescale™
semiconductor