MPRPPCPRG-01

MPCPRG/D
10/95

PowerPC™ Microprocessor Family: The Programmer's Reference Guide

PowerPC

This document contains information on a new product under development by Motorola and IBM. Motorola and IBM reserve the right to change or discontinue this product without notice. Information in this document is provided solely to enable system and software implementers to use PowerPC microprocessors. There are no express or implied copyright or patent licenses granted hereunder by Motorola or IBM to design, modify the design of, or fabricate circuits based on the information in this document.

This document was created with FrameMaker 4.0.4

Introduction

The primary objective of this document is to provide a concise method by which system software and hardware developers and application programmers may more readily provide software that is compatible across the family of PowerPC processors and other devices. A more detailed account of the following topics or the PowerPC architecture in general, may be obtained from the *PowerPC Microprocessor Family: The Programming Environments*, referred to as *The Programming Environments Manual*. (*The PowerPC Architecture: A Specification for a New Family of RISC Processors* defines the architecture from the perspective of the three programming environments and remains the defining document for the PowerPC architecture.)

This document is divided into four parts:

- Part 1, “Register Summary,” on page 4 provides a brief overview of the PowerPC register set, including a programming model and quick reference guides for both 32- and 64-bit registers.
- Part 2, “Memory Control Model,” on page 28 provides a brief outline of the page table entry and segment table entry for both 32- and 64-bit implementations.
- Part 3, “Exception Vectors,” on page 40 provides a quick reference for exception types and the conditions that cause them.
- Part 4, “PowerPC Instruction Set,” on page 41 provides detailed information on the instruction field summary—including syntax and notation conventions. Also included, is the entire PowerPC instruction set, sorted by mnemonic and opcode.

In this document, the term “60x” is used to denote a 32-bit microprocessor from the PowerPC architecture family. 60x processors implement the PowerPC architecture as it is specified for 32-bit addressing, which provides 32-bit effective (logical) addresses, integer data types of 8, 16, and 32 bits, and floating-point data types of 32 and 64 bits (single-precision and double-precision).

Table 1 contains acronyms and abbreviations that are used in this document. Note that the meanings for some acronyms (such as SDR1 and XER) are historical, and the words for which an acronym stands may not be intuitively obvious.

Table 1. Acronyms and Abbreviated Terms

Term	Meaning
ASR	Address space register
BAT	Block address translation
BUID	Bus unit ID
CR	Condition register
CTR	Count register
DAR	Data address register
DBAT	Data BAT
DEC	Decrementer register
DSISR	Register used for determining the source of a DSI exception
DTLB	Data translation lookaside buffer
EA	Effective address
EAR	External access register
FPR	Floating-point register
FPSCR	Floating-point status and control register
GPR	General-purpose register
IBAT	Instruction BAT
IEEE	Institute of Electrical and Electronics Engineers
IU	Integer unit
LR	Link register
MMU	Memory management unit
msb	Most significant bit
MSR	Machine state register
NaN	Not a number
No-Op	No operation
OEA	Operating environment architecture
PTE	Page table entry
PTEG	Page table entry group
PVR	Processor version register
RISC	Reduced instruction set computing
SDR1	Register that specifies the page table base address for virtual-to-physical address translation
SIMM	Signed immediate value
SLB	Segment lookaside buffer

Table 1. Acronyms and Abbreviated Terms (Continued)

Term	Meaning
SPR	Special-purpose register
SPRG*n*	Registers available for general purposes
SR	Segment register
SRR0	Machine status save/restore register 0
SRR1	Machine status save/restore register 1
TB	Time base register
TLB	Translation lookaside buffer
UIMM	Unsigned immediate value
UISA	User instruction set architecture
VEA	Virtual environment architecture
XER	Register used for indicating conditions such as carries and overflows for integer operations

Table 2 describes instruction field notation conventions used in this document.

Table 2. Instruction Field Conventions

The Architecture Specification	Equivalent to:
BA, BB, BT	**crb**A, **crb**B, **crb**D (respectively)
BF, BFA	**crf**D, **crf**S (respectively)
D	d
DS	ds
FLM	FM
FRA, FRB, FRC, FRT, FRS	**fr**A, **fr**B, **fr**C, **fr**D, **fr**S (respectively)
FXM	CRM
RA, RB, RT, RS	**r**A, **r**B, **r**D, **r**S (respectively)
SI	SIMM
U	IMM
UI	UIMM
/, //, ///	0...0 (shaded)

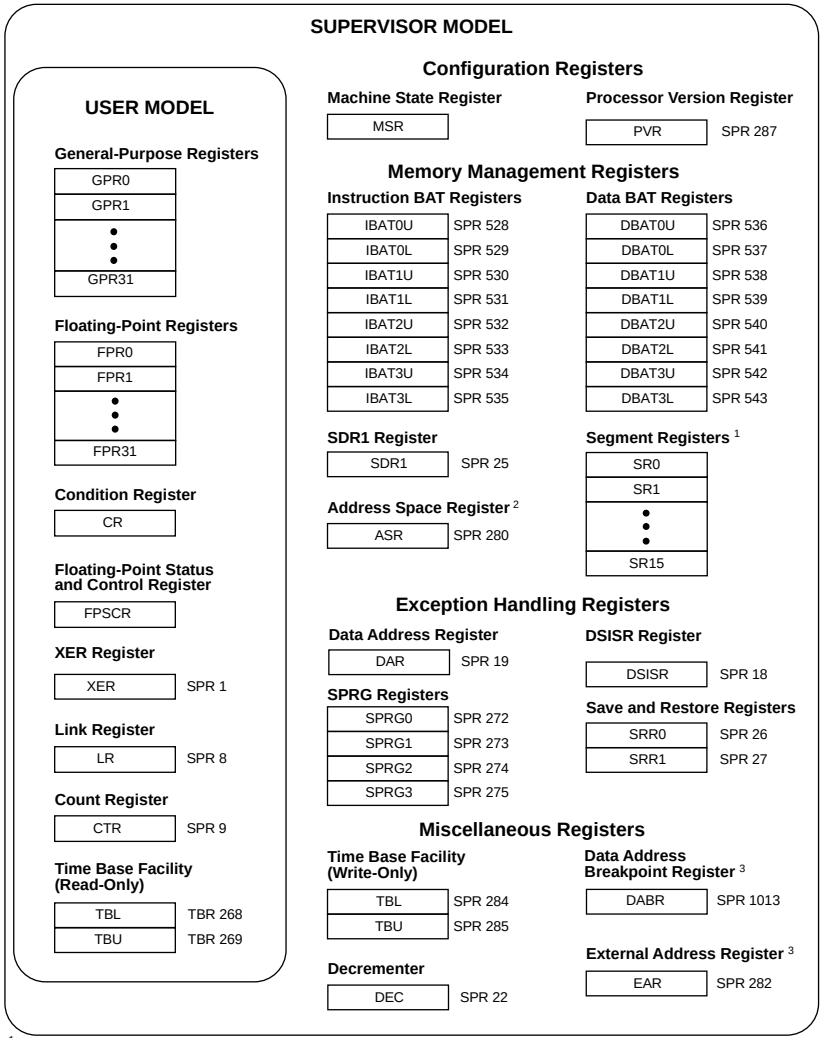
Part 1 Register Summary

This section describes the register organization defined by the three levels of the PowerPC architecture—user instruction set architecture (UISA), virtual environment architecture (VEA), and operating environment architecture (OEA). The PowerPC architecture defines register-to-register operations for all computational instructions. Source data for these instructions are accessed from the on-chip registers or are provided as immediate values embedded in the opcode. The three-register instruction format allows specification of a target register distinct from the two source registers, thus preserving the original data for use by other instructions and reducing the number of instructions required for certain operations. Data is transferred between memory and registers with explicit load and store instructions only.

Figure 1 shows a graphic representation of the entire PowerPC register set. The number to the right of the register name indicates the number that is used in the syntax of the instruction operands to access the register (for example, the number used to access the XER is SPR1).

Many of the SPRs can be accessed only by supervisor-level instructions; any attempt to access these SPRs with user-level instructions results in a supervisor-level exception. Some SPRs are implementation-specific. In some cases, not all of a register's bits are implemented in hardware. When a PowerPC microprocessor detects SPR encodings other than those defined in this document, it either takes a program exception (if bit 0 of the SPR encoding is set) or it treats the instruction as a no-op (if bit 0 of the SPR encoding is clear).

Note that the general purpose registers (GPRs), link register (LR), count register (CTR), machine state register (MSR), data address register (DAR), SDR1, save and restore registers 0 and 1 (SRR0 and SRR1), SPRG0–SPRG3, and data address breakpoint register (DABR) are 64 bits in length in 64-bit implementations and 32 bits in length in 32-bit implementations.

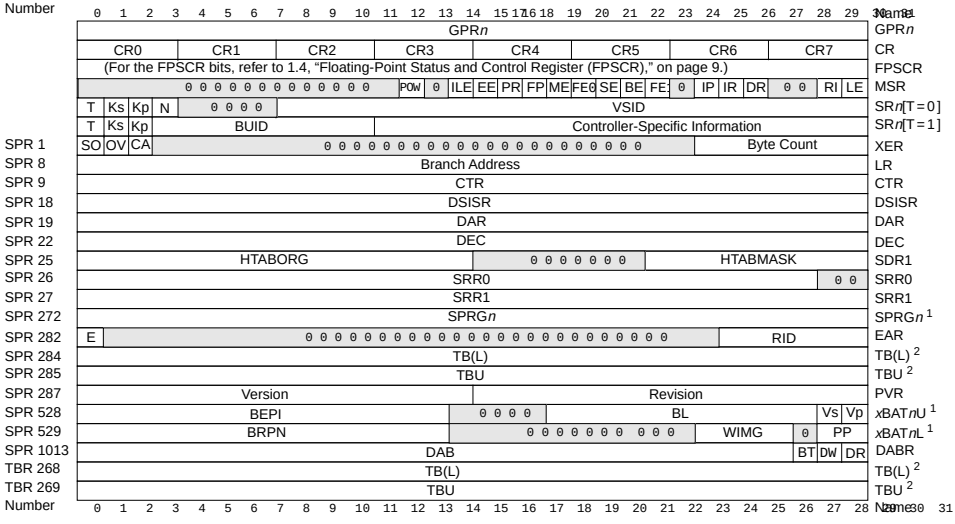


[1] These registers are in 32-bit implementations only.
[2] These registers are in 64-bit implementations only.
[3] These registers are optional in the PowerPC architecture.

Figure 1. PowerPC Programming Model—Registers

Table 3 provides a quick method by which to reference the SPR and TBR numbers and bit fields for all 32-bit PowerPC registers. Note that reserved bits are shaded.

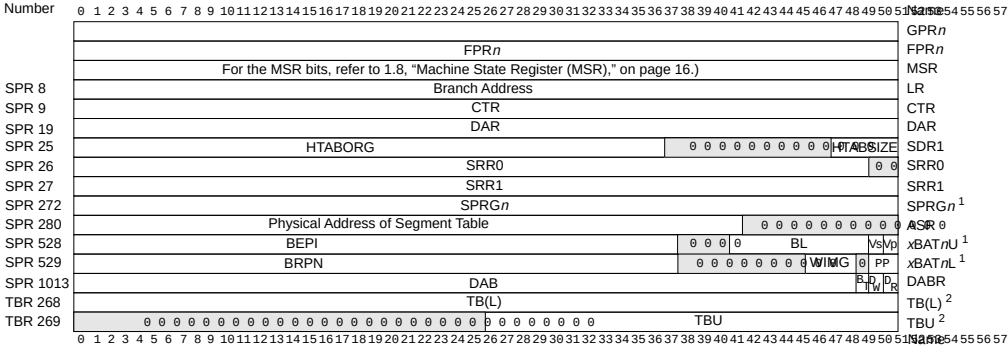
Table 3. Quick Reference Guide—32-Bit Registers



Notes:
1. For all SPR numbers refer to Figure 1
2. Write-Only
3. Read-Only

Table 4 provides a quick method by which to reference the SPR and TBR numbers and bit fields for all 64-bit PowerPC registers. Note that reserved bits are shaded.

Table 4. Quick Reference Guide—64-Bit Registers



Notes:
1. For all SPR numbers refer to Figure 1
2. Read-only

1.1 General-Purpose Registers (GPRs)

Integer data is manipulated in the processor’s 32 GPRs shown in Figure 2. These registers are 64-bit registers in 64-bit implementations and 32-bit registers in 32-bit implementations. The GPRs are accessed as source and destination registers in the instruction syntax.

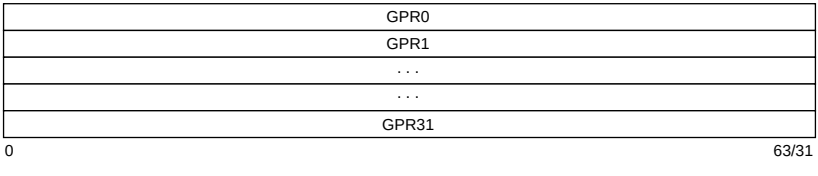


Figure 2. General-Purpose Registers (GPRs)

1.2 Floating-Point Registers (FPRs)

The PowerPC architecture provides thirty-two 64-bit FPRs as shown in Figure 3. These registers are accessed as source and destination registers for floating-point instructions. Each FPR supports the double-precision floating-point format. Every instruction that interprets the contents of an FPR as a floating-point value uses the double-precision floating-point format for this interpretation.

All floating-point arithmetic instructions operate on data located in FPRs and, with the exception of compare instructions, place the result into an FPR. Information about the status of floating-point operations is placed into the FPSCR and in some cases, into the CR after the completion of instruction execution.

The floating-point arithmetic instructions produce intermediate results that may be regarded as infinitely precise. After normalization or denormalization, if the precision of the intermediate result cannot be represented in the destination format (single or double precision), it is rounded to the specified precision before being placed in the target FPR. The final result is then placed into the FPR in the double-precision format.

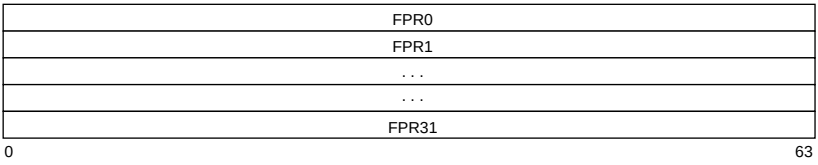


Figure 3. Floating-Point Registers (FPRs)

1.3 XER Register (XER)

The XER register (XER) is shown in Figure 4.

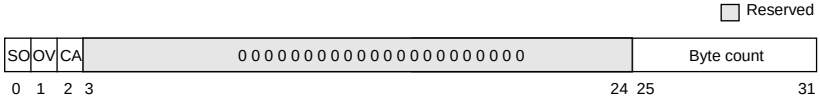


Figure 4. XER Register

Table 5 provides bit setting information for XER.

Table 5. XER Bit Definitions

Bit(s)	Name	Description
0	SO	Summary overflow. The summary overflow bit (SO) is set whenever an instruction (except **mtspr**) sets the overflow bit (OV). Once set, the SO bit remains set until it is cleared by an **mtspr** instruction (specifying the XER) or an **mcrxr** instruction. It is not altered by compare instructions, nor by other instructions (except **mtspr** to the XER, and **mcrxr**) that cannot overflow. Executing an **mtspr** instruction to the XER, supplying the values zero for SO and one for OV, causes SO to be cleared and OV to be set.
1	OV	Overflow. The overflow bit (OV) is set to indicate that an overflow has occurred during execution of an instruction. Add, subtract from, and negate instructions having OE = 1 set the OV bit if the carry out of the msb is not equal to the carry out of the msb + 1, and clear it otherwise. Multiply low and divide instructions having OE = 1 set the OV bit if the result cannot be represented in 64 bits (**mulld**, **divd**, **divdu**) or in 32 bits (**mullw**, **divw**, **divwu**), and clear it otherwise. The OV bit is not altered by compare instructions that cannot overflow (except **mtspr** to the XER, and **mcrxr**).

Table 5. XER Bit Definitions (Continued)

Bit(s)	Name	Description
2	CA	Carry. The carry bit (CA) is set during execution of the following instructions: • Add carrying, subtract from carrying, add extended, and subtract from extended instructions set CA if there is a carry out of the msb, and clear it otherwise. • Shift right algebraic instructions set CA if any 1 bits have been shifted out of a negative operand, and clear it otherwise. The CA bit is not altered by compare instructions, nor by other instructions that cannot carry (except shift right algebraic, **mtspr** to the XER, and **mcrxr**).
3–24	—	Reserved
25–31	Byte Count	This field specifies the number of bytes to be transferred by a Load String Word Indexed (**lswx**) or Store String Word Indexed (**stswx**) instruction.

1.4 Floating-Point Status and Control Register (FPSCR)

Figure 5 shows the format of the floating-point status and control register (FPSCR).

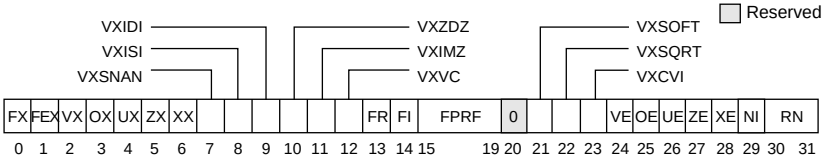


Figure 5. Floating-Point Status and Control Register (FPSCR)

The FPSCR contains bits to do the following:

- Record exceptions generated by floating-point operations
- Record the type of the result produced by a floating-point operation
- Control the rounding mode used by floating-point operations
- Enable or disable the reporting of exceptions (invoking the exception handler)

Bits 0–23 are status bits. Bits 24–31 are control bits. Status bits in the FPSCR are updated at the completion of the instruction execution.

Except for the floating-point enabled exception summary (FEX) and floating-point invalid operation exception summary (VX), the exception condition bits in the FPSCR (bits 0–12 and 21–23) are sticky. Once set, sticky bits remain set until they are cleared by an **mcrfs**, **mtfsfi**, **mtfsf**, or **mtfsb0** instruction.

FEX and VX are the logical ORs of other FPSCR bits. Therefore, these two bits are not listed among the FPSCR bits directly affected by the various instructions.

FPSCR bit settings are shown in Table 6.

Table 6. FPSCR Bit Settings

Bit(s)	Name	Description
0	FX	Floating-point exception summary. Every floating-point instruction, except **mtfsfi** and **mtfsf**, implicitly sets FPSCR[FX] if that instruction causes any of the floating-point exception bits in the FPSCR to transition from 0 to 1. The **mcrfs**, **mtfsfi**, **mtfsf**, **mtfsb0**, and **mtfsb1** instructions can alter FPSCR[FX] explicitly. This is a sticky bit.
1	FEX	Floating-point enabled exception summary. This bit signals the occurrence of any of the enabled exception conditions. It is the logical OR of all the floating-point exception bits masked by their respective enable bits. The **mcrfs**, **mtfsf**, **mtfsfi**, **mtfsb0**, and **mtfsb1** instructions cannot alter FPSCR[FEX] explicitly. This is not a sticky bit.
2	VX	Floating-point invalid operation exception summary. This bit signals the occurrence of any invalid operation exception. It is the logical OR of all of the invalid operation exceptions. The **mcrfs**, **mtfsf**, **mtfsfi**, **mtfsb0**, and **mtfsb1** instructions cannot alter FPSCR[VX] explicitly. This is not a sticky bit.
3	OX	Floating-point overflow exception. This is a sticky bit.
4	UX	Floating-point underflow exception. This is a sticky bit.
5	ZX	Floating-point zero divide exception. This is a sticky bit.
6	XX	Floating-point inexact exception. This is a sticky bit. FPSCR[XX] is the sticky version of FPSCR[FI]. The following rules describe how FPSCR[XX] is set by a given instruction: • If the instruction affects FPSCR[FI], the new value of FPSCR[XX] is obtained by logically ORing the old value of FPSCR[XX] with the new value of FPSCR[FI]. • If the instruction does not affect FPSCR[FI], the value of FPSCR[XX] is unchanged.
7	VXSNAN	Floating-point invalid operation exception for SNaN. This is a sticky bit.
8	VXISI	Floating-point invalid operation exception for ∞ – ∞. This is a sticky bit.
9	VXIDI	Floating-point invalid operation exception for ∞ ÷ ∞. This is a sticky bit.
10	VXZDZ	Floating-point invalid operation exception for 0 ÷ 0. This is a sticky bit.
11	VXIMZ	Floating-point invalid operation exception for ∞ * 0. This is a sticky bit.
12	VXVC	Floating-point invalid operation exception for invalid compare. This is a sticky bit.
13	FR	Floating-point fraction rounded. The last arithmetic or rounding and conversion instruction that rounded the intermediate result incremented the fraction. This bit is not sticky.
14	FI	Floating-point fraction inexact. The last arithmetic or rounding and conversion instruction either rounded the intermediate result (producing an inexact fraction) or caused a disabled overflow exception. This is not a sticky bit. For more information regarding the relationship between FPSCR[FI] and FPSCR[XX], see the description of the FPSCR[XX] bit.

Table 6. FPSCR Bit Settings (Continued)

Bit(s)	Name	Description
15–19	FPRF	Floating-point result flags. For arithmetic, rounding, and conversion instructions the field is based on the result placed into the target register, except that if any portion of the result is undefined, the value placed here is undefined. 15 Floating-point result class descriptor (C). Arithmetic, rounding and conversion instructions may set this bit with the FPCC bits to indicate the class of the result; see Table 7. 16–19 Floating-point condition code (FPCC). Floating-point compare instructions always set one of the FPCC bits to one and the other three FPCC bits to zero. Arithmetic, rounding and conversion instructions may set the FPCC bits with the C bit to indicate the class of the result. Note that in this case the high-order three bits of the FPCC retain their relational significance indicating that the value is less than, greater than, or equal to zero. 16 Floating-point less than or negative (FL or <) 17 Floating-point greater than or positive (FG or >) 18 Floating-point equal or zero (FE or =) 19 Floating-point unordered or NaN (FU or ?) These are not sticky bits.
20	—	Reserved
21	VXSOFT	Floating-point invalid operation exception for software request. This is a sticky bit. This bit can be altered only by the **mcrfs**, **mtfsfi**, **mtfsf**, **mtfsb0**, or **mtfsb1** instructions.
22	VXSQRT	Floating-point invalid operation exception for invalid square root. This is a sticky bit.
23	VXCVI	Floating-point invalid operation exception for invalid integer convert. This is a sticky bit.
24	VE	Floating-point invalid operation exception enable. This is not a sticky bit.
25	OE	IEEE floating-point overflow exception enable. This is not a sticky bit.
26	UE	IEEE floating-point underflow exception enable. This is not a sticky bit.
27	ZE	IEEE floating-point zero divide exception enable. This is not a sticky bit.
28	XE	Floating-point inexact exception enable. This is not a sticky bit.
29	NI	Floating-point non-IEEE mode. If this bit is set, results need not conform with IEEE standards and the other FPSCR bits may have meanings other than those described here. If the bit is set and if all implementation-specific requirements are met and if an IEEE-conforming result of a floating-point operation would be a denormalized number, the result produced is zero (retaining the sign of the denormalized number). Any other effects associated with setting this bit are described in the user's manual for the implementation. Effects of the setting of this bit is implementation-dependent. This is not a sticky bit.
30–31	RN	Floating-point rounding control. 00 Round to nearest 01 Round toward zero 10 Round toward +infinity 11 Round toward –infinity These are not sticky bits.

Table 7 illustrates the floating-point result flags used by PowerPC processors. The result flags correspond to FPSCR bits 15–19.

Table 7. Floating-Point Result Flags in FPSCR

Result Flags (Bits 15–19)					Result Value Class
C	<	>	=	?	
1	0	0	0	1	Quiet NaN
0	1	0	0	1	–Infinity
0	1	0	0	0	–Normalized number
1	1	0	0	0	–Denormalized number
1	0	0	1	0	–Zero
0	0	0	1	0	+Zero
1	0	1	0	0	+Denormalized number
0	0	1	0	0	+Normalized number
0	0	1	0	1	+Infinity

1.5 Condition Register (CR)

The format of the condition register (CR) is shown in Figure 6.

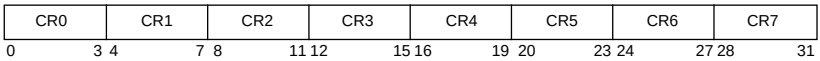


Figure 6. Condition Register (CR)

The CR fields can be set in one of the following ways:

- Specified fields of the CR can be set by a move instruction (**mtcrf**) to the CR from a GPR.
- A specified field of the CR can be moved to another CR field with the **mcrf** instruction.
- A specified field of the XER can be copied to the CR by the **mcrxr** instruction.
- A specified field of the FPSCR can be copied to the CR by the **mcrfs** instruction.
- Condition register logical instructions can be used to perform logical operations on specified bits in the condition register.
- CR0 can be the implicit result of an integer instruction.
- CR1 can be the implicit result of a floating-point instruction.
- A specified CR field can indicate the result of either an integer or floating-point compare instruction.

Note that branch instructions are provided to test individual CR bits.

The following tables, Table 8–Table 10, provide bit setting information for CR0, CR1, and the CR*n* fields, respectively.

Table 8. Bit Settings for CR0 Field of CR

CR0 Bit	Description
0	Negative (LT)—This bit is set when the result is negative.
1	Positive (GT)—This bit is set when the result is positive (and not zero).
2	Zero (EQ)—This bit is set when the result is zero.
3	Summary overflow (SO)—This is a copy of the final state of XER[SO] at the completion of the instruction.

Table 9. Bit Settings for CR1 Field of CR

CR1 Bit	Description
4	Floating-point exception (FX)—This is a copy of the final state of FPSCR[FX] at the completion of the instruction.
5	Floating-point enabled exception (FEX)—This is a copy of the final state of FPSCR[FEX] at the completion of the instruction.
6	Floating-point invalid exception (VX)—This is a copy of the final state of FPSCR[VX] at the completion of the instruction.
7	Floating-point overflow exception (OX)—This is a copy of the final state of FPSCR[OX] at the completion of the instruction.

Note: For more information on the FPSCR refer to Section 1.4, “Floating-Point Status and Control Register (FPSCR).”

Table 10. CR*n* Field Bit Settings for Compare Instructions

CR*n* Bit[1]	Description[2]
0	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.
1	Greater than or floating-point greater than (GT, FG). 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.
2	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.
3	Summary overflow or floating-point unordered (SO, FU). For integer compare instructions: This is a copy of the final state of XER[SO] at the completion of the instruction. For floating-point compare instructions: One or both of **fr**A and **fr**B is a Not a Number (NaN).

Notes:
1. Here, the bit indicates the bit number in any one of the four-bit subfields, CR0–CR7.
2. For a complete description of instruction syntax conventions, refer to Table 31.

1.6 Link Register (LR)

The link register (LR) is a 64-bit register in 64-bit implementations and a 32-bit register in 32-bit implementations. The LR supplies the branch target address for the Branch Conditional to Link Register (**bclr***x*) instruction, and can be used to hold the logical address of the instruction that follows a branch and link instruction. The format of LR is shown in Figure 7.

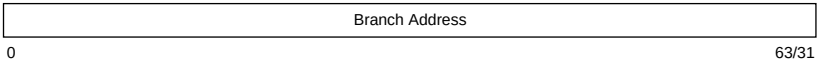


Figure 7. Link Register (LR)

Note that although the two least-significant bits can accept any values written to them, they are ignored when the LR is used as an address. The link register can be accessed by the **mtspr** and **mfspr** instructions using SPR8. Fetching instructions along the target path (loaded by an **mtspr** instruction) is possible provided the link register is loaded sufficiently ahead of the branch instruction. It is possible for a PowerPC microprocessor to fetch along a target path loaded by a branch and link instruction.

Both conditional and unconditional branch instructions include the option of placing the effective address of the instruction following the branch instruction in the LR.

1.7 Count Register (CTR)

The count register (CTR) is a 64-bit register in 64-bit implementations and a 32-bit register in 32-bit implementations. The CTR can hold a loop count that can be decremented during execution of branch instructions that contain an appropriately coded BO field. If the value in CTR is 0 before being decremented, it is –1 afterward. The CTR can also provide the branch target address for the Branch Conditional to Count Register (**bcctr***x*) instruction. The CTR is shown in Figure 8.

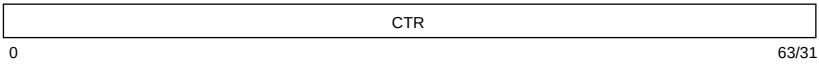


Figure 8. Count Register (CTR)

Fetching instructions along the target path is also possible provided the count register is loaded sufficiently ahead of the branch instruction.

The count register can be accessed by the **mtspr** and **mfspr** instructions by specifying SPR9. In branch conditional instructions, the BO field specifies the conditions under which the branch is taken. The first four bits of the BO field specify how the branch is affected by or affects the CR and the CTR. The encoding for the BO field is shown in Table 11.

Table 11. 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 is ignored. The *z* bits should be cleared to zero, as they may be assigned a meaning in some future version of the PowerPC architecture.

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

1.8 Machine State Register (MSR)

The machine state register (MSR), is a 64-bit register on 64-bit implementations (see Figure 9) and a 32-bit register in 32-bit implementations (see Figure 10).

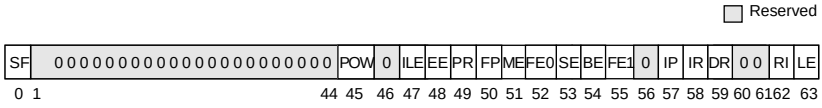


Figure 9. Machine State Register (MSR)—64-bit Implementations

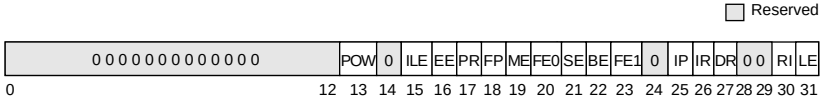


Figure 10. Machine State Register (MSR)—32-bit Implementations

Table 12 shows the bit definitions for the MSR. Full function reserved bits are saved in SRR1 when an exception occurs; partial function reserved bits are not saved.

Table 12. MSR Bit Settings

Bit(s)		Name	Description
64 Bit	32 Bit		
0	—	SF	Sixty-four bit mode 0 The 64-bit processor runs in 32-bit mode. 1 The 64-bit processor runs in 64-bit mode. Note that this is the default setting.
1–32	0	—	Reserved. Full function.
33–36	1–4	—	Reserved. Partial function.
37–41	5–9	—	Reserved. Full function.
42–44	10–12	—	Reserved. Partial function.
45	13	POW	Power management enable 0 Power management disabled (normal operation mode). 1 Power management enabled (reduced power mode). Note: Power management functions are implementation-dependent. If the function is not implemented, this bit is treated as reserved.
46	14	—	Reserved—Implementation-specific
47	15	ILE	Exception little-endian mode. When an exception occurs, this bit is copied into MSR[LE] to select the endian mode for the context established by the exception.
48	16	EE	External interrupt enable 0 While the bit is cleared the processor delays recognition of external interrupts and decrementer exception conditions. 1 The processor is enabled to take an external interrupt or the decrementer exception.

Table 12. MSR Bit Settings (Continued)

Bit(s)		Name	Description
64 Bit	32 Bit		
49	17	PR	Privilege level 0 The processor can execute both user- and supervisor-level instructions. 1 The processor can only execute user-level instructions.
50	18	FP	Floating-point available 0 The processor prevents dispatch of floating-point instructions, including floating-point loads, stores, and moves. 1 The processor can execute floating-point instructions.
51	19	ME	Machine check enable 0 Machine check exceptions are disabled. 1 Machine check exceptions are enabled.
52	20	FE0	Floating-point exception mode 0 (see Table 13).
53	21	SE	Single-step trace enable (Optional) 0 The processor executes instructions normally. 1 The processor generates a single-step trace exception upon the successful execution of the next instruction. Note: If the function is not implemented, this bit is treated as reserved.
54	22	BE	Branch trace enable (Optional) 0 The processor executes branch instructions normally. 1 The processor generates a branch trace exception after completing the execution of a branch instruction, regardless of whether or not the branch was taken. Note: If the function is not implemented, this bit is treated as reserved.
55	23	FE1	Floating-point exception mode 1 (see Table 13).
56	24	—	Reserved. Full function.
57	25	IP	Exception prefix. The setting of this bit specifies whether an exception vector offset is prepended with Fs or 0s. In the following description, *nnnnn* is the offset of the exception. See Table 30. 0 Exceptions are vectored to the physical address 0x000*n_nnnn* in 32-bit implementations and 0x0000_0000_000*n_nnnn* in 64-bit implementations. 1 Exceptions are vectored to the physical address 0xFFF*n_nnnn* in 32-bit implementations and 0xFFFF_FFFF_FFF*n_nnnn* in 64-bit implementations.
58	26	IR	Instruction address translation 0 Instruction address translation is disabled. 1 Instruction address translation is enabled.
59	27	DR	Data address translation 0 Data address translation is disabled. 1 Data address translation is enabled.
60–61	28–29	—	Reserved. Full function.
62	30	RI	Recoverable exception (for system reset and machine check exceptions). 0 Exception is not recoverable. 1 Exception is recoverable.
63	31	LE	Little-endian mode enable 0 The processor runs in big-endian mode. 1 The processor runs in little-endian mode.

The floating-point exception mode bits (FE0–FE1) are interpreted as shown in Table 13. Note that these bits can be logically ORed, so that if either is set the processor operates in precise mode.

Table 13. Floating-Point Exception Mode Bits

FE0	FE1	Mode
0	0	Floating-point exceptions disabled
0	1	Floating-point imprecise nonrecoverable
1	0	Floating-point imprecise recoverable
1	1	Floating-point precise mode

Table 14 indicates the initial state of the MSR.

Table 14. State of MSR at Power Up

Bit(s)		Name	64-Bit Description	32-Bit Description
64 Bit	32 Bit			
0	—	SF	1	—
1–44	0–12	—	Unspecified[1]	Unspecified[1]
45	13	POW	0	0
46	14	—	Unspecified[1]	Unspecified[1]
47	15	ILE	0	0
48	16	EE	0	0
49	17	PR	0	0
50	18	FP	0	0
51	19	ME	0	0
52	20	FE0	0	0
53	21	SE	0	0
54	22	BE	0	0
55	23	FE1	0	0
56	24	—	Unspecified[1]	Unspecified[1]
57	25	IP	1[2]	1[2]
58	26	IR	0	0
59	27	DR	0	0

Table 14. State of MSR at Power Up (Continued)

Bit(s)		Name	64-Bit Description	32-Bit Description
64 Bit	32 Bit			
60–61	28–29	—	Unspecified[1]	Unspecified[1]
62	30	RI	0	0
63	31	LE	0	0

Notes:
1. Unspecified can be either 0 or 1
2. 1 is typical, but might be 0

1.9 Processor Version Register (PVR)

The processor version register (PVR) is a 32-bit, read-only register that contains a value identifying the specific version (model) and revision level of the PowerPC processor (see Figure 11). The contents of the PVR can be copied to a GPR by the **mfspr** instruction. Read access to the PVR is supervisor-level only; write access is not provided.

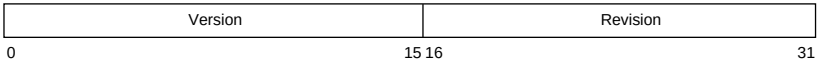


Figure 11. Processor Version Register (PVR)

The PVR consists of two 16-bit fields:

- Version (bits 0–15)—A 16-bit number that uniquely determines a particular processor version and version of the PowerPC architecture. This number can be used to determine the version of a processor; it may not distinguish between different product models if more than one model uses the same processor.
- Revision (bits 16–31)—A 16-bit number that distinguishes between various releases of a particular version (that is, an engineering change level). The value of the revision portion of the PVR is implementation-specific. The processor revision level is changed for each revision of the device.

1.10 BAT Registers

Figure 12 and Figure 13 show the format of the upper and lower BAT registers for 64-bit PowerPC processors.

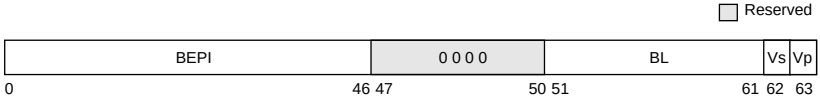


Figure 12. Upper BAT Register—64-Bit Implementations

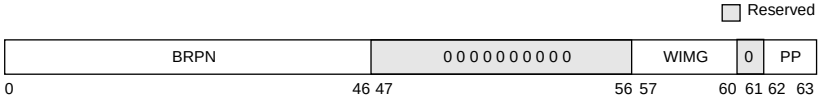


Figure 13. Lower BAT Register—64-Bit Implementations

Figure 14 and Figure 15 show the format of the upper and lower BAT registers for 32-bit PowerPC processors.

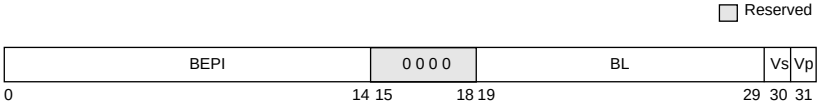


Figure 14. Format of Upper BAT Registers—32-Bit Implementations

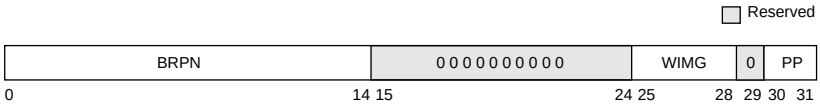


Figure 15. Format of Lower BAT Registers—32-Bit Implementations

Table 15 describes the bits in the BAT registers.

Table 15. BAT Registers—Field and Bit Descriptions

Upper/ Lower BAT	Bits		Name	Description
	64 Bit	32 Bit		
Upper BAT Register	0–46	0–14	BEPI	Block effective page index. This field is compared with high-order bits of the logical address to determine if there is a hit in that BAT array entry. (The architecture specification refers to logical address as effective address.)
	46–50	15–18	—	Reserved
	51–61	19–29	BL	Block length. BL is a mask that encodes the size of the block. Values for this field are listed in Table 16.
	62	30	Vs	Supervisor mode valid bit. This bit interacts with MSR[PR] to determine if there is a match with the logical address.
	63	31	Vp	User mode valid bit. This bit also interacts with MSR[PR] to determine if there is a match with the logical address.
Lower BAT Register	0–46	0–14	BRPN	This field is used in conjunction with the BL field to generate high-order bits of the physical address of the block.
	47–56	15–24	—	Reserved
	57–60	25–28	WIMG	Memory/cache access mode bits W Write-through I Caching-inhibited M Memory coherence G Guarded
	61	29	—	Reserved
	62–63	30–31	PP	Protection bits for block

Table 16 lists the BAT area lengths encoded in the BL field of the upper BAT registers.

Table 16. BAT Area Lengths

BAT Area Length	BL Encoding
128 Kbytes	000 0000 0000
256 Kbytes	000 0000 0001
512 Kbytes	000 0000 0011
1 Mbyte	000 0000 0111
2 Mbytes	000 0000 1111
4 Mbytes	000 0001 1111
8 Mbytes	000 0011 1111
16 Mbytes	000 0111 1111
32 Mbytes	000 1111 1111

Table 16. BAT Area Lengths (Continued)

BAT Area Length	BL Encoding
64 Mbytes	001 1111 1111
128 Mbytes	011 1111 1111
256 Mbytes	111 1111 1111

1.11 SDR1

The SDR1 is a 64-bit register in 64-bit implementations and a 32-bit register in 32-bit implementations. Refer to Section 2.3.3, “SDR1 Register Definitions,” for a complete description of SDR1.

1.12 Address Space Register (ASR)

The address space register (ASR) is a 64-bit SPR that holds 0–51 of the segment table’s physical address. The segment table is the segment descriptor mechanism for 64-bit implementations. For more detailed information about the ASR, refer to Section 2.2.1.1, “Address Space Register (ASR).”

1.13 Segment Registers (SRs)

Segment registers are used in page and direct-store segment address translations. Refer to Section 2.2, “Segment Descriptor Definitions,” for information on segment registers.

1.14 Data Address Register (DAR)

The DAR is a 64-bit register in 64-bit implementations and a 32-bit register in 32-bit implementations. The DAR is shown in Figure 16.

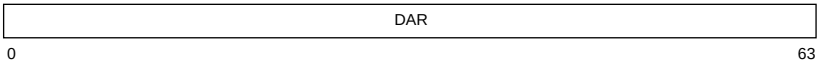


Figure 16. Data Address Register (DAR)

The effective address generated by a memory access instruction is placed in the DAR if the access causes an exception (for example, an alignment exception). If the exception occurs in a 64-bit implementation operating in 32-bit mode, the high-order 32 bits of the DAR are cleared.

1.15 SPRG0–SPRG3

SPRG0–SPRG3 are 64-bit or 32-bit registers, depending on the type of PowerPC microprocessor. They are provided for general operating system use, such as performing a fast state save or for supporting multiprocessor implementations. The formats of SPRG0 through SPRG3 are shown in Figure 17.

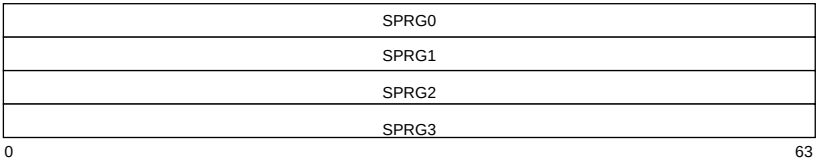


Figure 17. SPRG0–SPRG3

Table 17 provides a description of conventional uses of SPRG0–SPRG3.

Table 17. Conventional Uses of SPRG0–SPRG3

Register	Description
SPRG0	Software may load a unique physical address in this register to identify an area of memory reserved for use by the first-level exception handler. This area must be unique for each processor in the system.
SPRG1	This register may be used as a scratch register by the first-level exception handler to save the content of a GPR. That GPR then can be loaded from SPRG0 and used as a base register to save other GPR's to memory.
SPRG2	This register may be used by the operating system as needed.
SPRG3	This register may be used by the operating system as needed.

1.16 DSISR

The 32-bit DSISR, shown in Figure 18, identifies the cause of DSI and alignment exceptions.

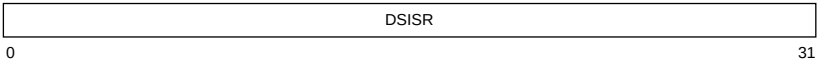


Figure 18. DSISR

1.17 Machine Status Save/Restore Register 0 (SRR0)

The SRR0 is a 64-bit register in 64-bit implementations and a 32-bit register in 32-bit implementations. SRR0 is used to save machine status on exceptions and restore machine

status when an **rfi** instruction is executed. It also holds the EA for the instruction that follows the System Call (**sc**) instruction. The format of SRR0 is shown in Figure 19. For 32-bit implementations, the format of SRR0 follows the low-order bits (32–63) of Figure 19.

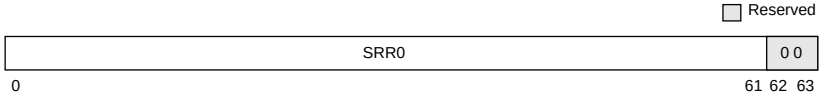


Figure 19. Machine Status Save/Restore Register 0 (SRR0)

When an exception occurs, SRR0 is set to point to an instruction such that all prior instructions have completed execution and no subsequent instruction has begun execution. When **rfi** is executed, the contents of SRR0 are copied to the next instruction address (NIA)—the 64- or 32-bit address of the next instruction to be executed. The instruction addressed by SRR0 may not have completed execution, depending on the exception type. SRR0 addresses either the instruction causing the exception or the instruction that immediately follows. The instruction addressed can be determined from the exception type and status bits.

If the exception occurs in 32-bit mode of the 64-bit implementation, the high-order 32 bits of SRR0 are cleared and the high-order 32 bits of the NIA are cleared when returning to 32-bit mode.

Note that in some implementations, every instruction fetch, when MSR[IR] = 1, and every instruction execution requiring address translation when MSR[DR] = 1, may modify SRR0.

1.18 Machine Status Save/Restore Register 1 (SRR1)

The SRR0 is a 64-bit register in 64-bit implementations and a 32-bit register in 32-bit implementations. SRR1 is used to save machine status on exceptions and to restore machine status when an **rfi** instruction is executed. The format of SRR1 is shown in Figure 20.

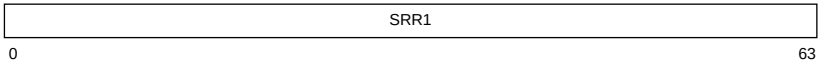


Figure 20. Machine Status Save/Restore Register 1 (SRR1)

On 64-bit implementations, when an exception occurs, bits 33–36 and 42–47 of SRR1 are loaded with exception-specific information and bits 0–32, 37–41, and 48–63 of MSR are placed into the corresponding bit positions of SRR1.

For 32-bit implementations, when an exception occurs, bits 1–4 and 10–15 of SRR1 are loaded with exception-specific information and bits 0, 5–9, and 16–31 of MSR are placed into the corresponding bit positions of SRR1.

Note that, in some implementations, every instruction fetch when MSR[IR] = 1, and every instruction execution requiring address translation when MSR[DR] = 1, may modify SRR1.

1.19 Time Base Facility (TB)

The time base (TB), shown in Figure 21, is a 64-bit structure that contains a 64-bit unsigned integer that is incremented periodically. Each increment adds 1 to the low-order bit (bit 63). The frequency at which the counter is incremented is implementation-dependent.

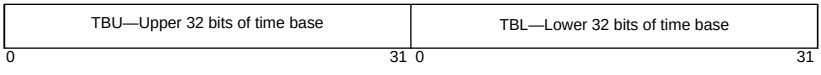


Figure 21. Time Base (TB)

The TB increments until its value becomes 0xFFFF_FFFF_FFFF_FFFF ($2^{64} - 1$). At the next increment its value becomes 0x0000_0000_0000_0000. Note that there is no explicit indication that this has occurred (that is, no exception is generated).

The period of the time base depends on the driving frequency. The TB is implemented such that the following requirements are satisfied:

1. Loading a GPR from the time base has no effect on the accuracy of the time base.
2. Storing a GPR to the time base replaces the value in the time base with the value in the GPR.

The PowerPC VEA does not specify a relationship between the frequency at which the time base is updated and other frequencies, such as the processor clock. The TB update frequency is not required to be constant; however, for the system software to maintain time of day and operate interval timers, one of two things is required:

- The system provides an implementation-dependent exception to software whenever the update frequency of the time base changes and a means to determine the current update frequency; or
- The system software controls the update frequency of the time base.

Note that if the operating system initializes the TB to some ‘reasonable’ value and the update frequency of the TB is constant, the TB can be used as a source of values that increase at a constant rate, such as for time stamps in trace entries.

Even if the update frequency is not constant, values read from the TB are monotonically increasing (except when the TB wraps from $2^{64} - 1$ to 0). If a trace entry is recorded each time the update frequency changes, the sequence of TB values can be post-processed to become actual time values.

For information on reading, writing, and computing time of day on the time base, refer to Chapter 2, "PowerPC Register Set," *The Programming Environments Manual*.

1.20 Decrementer Register (DEC)

The DEC, shown in Figure 22, is a 32-bit decrementing counter that provides a mechanism for causing a decrementer exception after a programmable delay. The DEC frequency is based on the same implementation-dependent frequency that drives the time base.

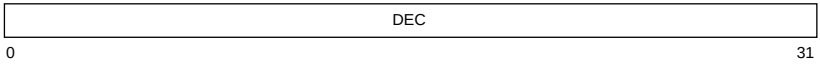


Figure 22. Decrementer Register (DEC)

For information on writing and reading the DEC, refer to Chapter 2, "PowerPC Register Set," *The Programming Environments Manual*.

1.21 Data Address Breakpoint Register (DABR)

The data address breakpoint facility is controlled by the DABR, a 64-bit register in 64-bit implementations and a 32-bit register in 32-bit implementations. The data address breakpoint facility is optional to the PowerPC architecture, as is the DABR. However, if the data address breakpoint facility is implemented, it is recommended, but not required, that it be implemented as described in this section.

The data address breakpoint facility provides a means to detect accesses to a designated double word. The address comparison is done on an effective address, and it applies to data accesses only. It does not apply to instruction fetches.

The DABR is shown in Figure 23.

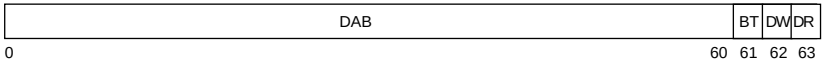


Figure 23. Data Address Breakpoint Register—64-Bit Implementations

Table 18 describes the fields in the DABR.

Table 18. DABR—Field Descriptions

Bits		Name	Description
64 Bit	32 Bit		
0–60	0–28	DAB	Data address breakpoint
61	29	BT	Breakpoint translation enable
62	30	DW	Data write enable
63	31	DR	Data read enable

A data address breakpoint match is detected for a load or store instruction if the three following conditions are met for any byte accessed:

- EA[0–60] = DABR[DAB]
- MSR[DR] = DABR[BT]
- The instruction is a store and DABR[DW] = 1, or the instruction is a load and DABR[DR] = 1.

In 32-bit mode of a 64-bit implementation, the high-order 32 bits of the EA are treated as zero for the purpose of detecting a match.

1.22 External Access Register (EAR)

The EAR is an optional 32-bit SPR that controls access to the external control facility and identifies the target device for external control operations. The external control facility provides a means for user-level instructions to communicate with special external devices. The EAR is shown in Figure 24. Note that the EAR is an optional register.

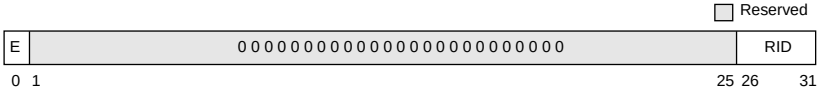


Figure 24. External Access Register (EAR)

The high-order bits of the resource ID (RID) field that correspond to bits of the RID beyond the width of the RID supported by a particular implementation are treated as reserved bits.

The EAR register is provided to support the External Control In Word Indexed (**eciwx**) and External Control Out Word Indexed (**ecowx**) instructions. Access to the EAR is supervisor-level, thus the operating system can determine which tasks are allowed to issue external access instructions and when they are allowed to do so. The bit settings for the EAR are described in Table 19.

The data access of **eciwx** and **ecowx** is performed as though the memory access mode bits (WIMG) were 0101. For example, if the external control facility is used to support a graphics adapter, the **ecowx** instruction could be used to send the translated physical address of a buffer containing graphics data to the graphics device. The **eciwx** instruction could be used to load status information from the graphics adapter.

Table 19. External Access Register (EAR) Bit Settings

Bit	Name	Description
0	E	Enable bit 1 Enabled 0 Disabled If this bit is set, the **eciwx** and **ecowx** instructions can perform the specified external operation. If the bit is cleared, an **eciwx** or **ecowx** instruction causes a DSI exception.
1–25	—	Reserved
26–31	RID	Resource ID

This register can also be accessed by using the **mtspr** and **mfspr** instructions.

Part 2 Memory Control Model

Memory in the PowerPC OEA is divided into 256-Mbyte segments. This segmented memory model provides a way to map 4-Kbyte pages of effective addresses to 4-Kbyte pages in physical memory (page address translation), while providing the programming flexibility afforded by a large virtual address space (80 or 52 bits).

The page address translation uses segment descriptors, which provide virtual address and protection information, and page table entries (PTEs), which provide the physical address and page protection information. The segment descriptors are programmed by the operating system to provide the virtual ID for a segment. In addition, the operating system also creates the page tables in memory that provide the virtual to physical address mappings (in the form of PTEs) for the pages in memory.

Segments in the OEA are defined as one of the following two types:

- Memory segment—An effective address in these segments represents a virtual address that is used to define the physical address of the page.
- Direct-store segment—References made to direct-store segments do not use the virtual paging mechanism of the processor.

The T bit in the segment descriptor selects between memory segments and direct-store segments, as shown in Table 20.

Table 20. Segment Descriptor Types

Segment Descriptor T Bit	Segment Type
0	Memory segment
1	Direct-store segment

All accesses generated by the processor map to a segment descriptor. If MSR[IR] = 0 or MSR[DR] = 0 for an instruction or data access, respectively, then real addressing mode translation is performed. Otherwise, if T = 0 in the corresponding segment descriptor (and the address is not translated by the BAT mechanism), the access maps to memory space and page address translation is performed.

After a memory segment is selected, the processor creates the virtual address for the segment and searches for the PTE that dictates the physical page number to be used for the access. Note that I/O devices can be easily mapped into memory space and used as memory-mapped I/O.

2.1 Address Translation Overview

The following sections provide a brief overview of the page and direct-store segment address translation. For more information, refer to Chapter 7, “Memory Management,” in *The Programming Environments Manual*.

2.1.1 Page Address Translation

The first step in page address translation for 64-bit implementations is the conversion of the 64-bit effective address of an access into the 80-bit virtual address. The virtual address is then used to locate the PTE in the page tables in memory. The physical page number is then extracted from the PTE and used in the formation of the physical address of the access.

The translation of an effective address to a physical address for 64-bit implementations is described briefly:

- Bits 0–35 of the effective address comprise the effective segment ID used to select a segment descriptor, from which the virtual segment ID (VSID) is extracted.
- Bits 36–51 of the effective address correspond to the page number within the segment; these are concatenated with the VSID from the segment descriptor to form the virtual page number (VPN). The VPN is used to search for the PTE in either an on-chip TLB or the page table. The PTE then provides the physical page number (RPN).
- Bits 52–63 of the effective address are the byte offset within the page; these are concatenated with the RPN field of a PTE to form the physical address used to access memory.

The translation of effective addresses to physical addresses for 32-bit implementations is similar to that for 64-bit implementations, except that 32-bit implementations index into an array of 16 segment registers instead of segment tables in memory to locate the segment descriptor, and the address ranges are obviously different. Thus, the address translation is as follows:

- Bits 0–3 of the effective address comprise the segment register number used to select a segment descriptor, from which the virtual segment ID (VSID) is extracted.
- Bits 4–19 of the effective address correspond to the page number within the segment; these are concatenated with the VSID from the segment descriptor to form the virtual page number (VPN). The VPN is used to search for the PTE in either an on-chip TLB or the page table. The PTE then provides the physical page number (RPN).
- Bits 20–31 of the effective address are the byte offset within the page; these are concatenated with the RPN field of a PTE to form the physical address used to access memory.

2.1.2 Direct-Store Segment Address Translation

As described for memory segments, all accesses generated by the processor (with translation enabled) that do not map to a BAT area, map to a segment descriptor. If T = 1 for the selected segment descriptor, the access maps to the direct-store interface, invoking a specific bus protocol for accessing some special-purpose I/O devices. Direct-store segments are provided for POWER compatibility. As the direct-store interface is present only for compatibility with existing I/O devices that used this interface and the direct-store interface protocol is not optimized for performance, its use is discouraged. Applications that require low-latency load/store access to external address space should use memory-mapped I/O, rather than the direct-store interface.

2.2 Segment Descriptor Definitions

The format of the segment descriptors is different for 64-bit and 32-bit implementations. Additionally, the fields in the segment descriptors are interpreted differently depending on the value of the T bit within the descriptor. When T = 1, the segment descriptor defines a direct-store segment.

2.2.1 STE Format—64-Bit Implementations

In 64-bit implementations, the segment descriptors reside as segment table entries (STEs) in hashed segment tables in memory. These STEs are generated and placed in segment tables in memory by the operating system. Each STE is a 128-bit entity (two double words) that maps one effective segment ID to one virtual segment ID. Information in the STE controls the segment table search process and provides input to the memory protection

mechanism. Figure 25 shows the format of both double words that comprise a T = 0 segment descriptor (or STE) in a 64-bit implementation.

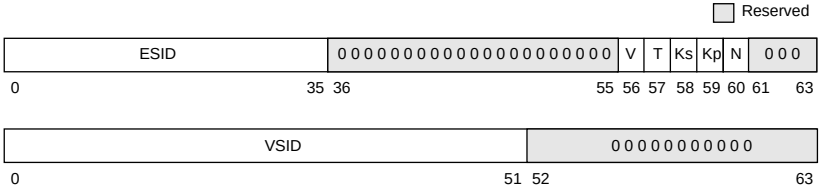


Figure 25. STE Format—64-Bit Implementations

Table 21 lists the bit definitions for each double word in an STE.

Table 21. STE Bit Definitions for Page Address Translation—64-Bit Implementations

Double Word	Bit	Name	Description
0	0–35	ESID	Effective segment ID
	36–55	—	Reserved
	56	V	Entry valid (V = 1) or invalid (V = 0)
	57	T	T = 0 selects this format
	58	Ks	Supervisor-state protection key
	59	Kp	User-state protection key
	60	N	No-execute protection bit
	61–63	—	Reserved
1	0–51	VSID	Virtual segment ID
	52–63	—	Reserved

The Ks and Kp bits partially define the access protection for the pages within the segment. The virtual segment ID field is used as the high-order bits of the virtual page number (VPN).

The segment descriptors are programmed by the operating system and placed into segment tables in memory, although some processors may additionally have on-chip segment lookaside buffers (SLBs). These SLBs store copies of recently-used STEs that can be accessed quickly, providing increased overall performance.

2.2.1.1 Address Space Register (ASR)

The ASR contains the control information for the segment table structure in that it defines the highest order bits for the physical base address of the segment table. The format of the

ASR is shown in Figure 26. The ASR contains bits 0–51 of the 64-bit physical base address of the segment table. Bits 52–56 of the STEG address are derived from the hashing function, (and bits 57–63 are zero at the beginning of a segment table search operation to point to the beginning of an STEG). Therefore, the beginning of the segment table lies on a 2^{12} byte (4 Kbyte) boundary.

Note that unless all accesses to be performed by the processor can be translated by the BAT mechanism when address translation is enabled (MSR[DR] or MSR[IR] = 1), the ASR must point to a valid segment table. If the processor does not support 64 bits of physical address, software should write zeros to those unsupported bits in the ASR. Otherwise, a machine check exception can occur.

Additionally, values x0, 0x1000, and 0x2000 should not be used as segment table addresses as they correspond to areas of the exception vector table reserved for implementation-specific purposes.

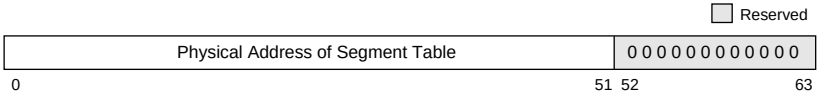


Figure 26. ASR Register Format—64-Bit Implementations Only

2.2.2 Segment Descriptor Format—32-Bit Implementations

In 32-bit implementations, the segment descriptors are 32-bits long and reside in one of 16 segment registers. Figure 27 shows the format of a segment register used in page address translation (T = 0) in a 32-bit implementation.

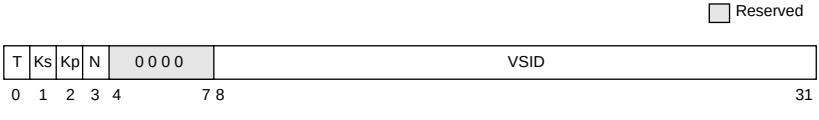


Figure 27. Segment Register Format for Page Address Translation—32-Bit Implementations

Table 22 provides the corresponding bit definitions of the segment register in 32-bit implementations.

Table 22. Segment Register Bit Definition for Page Address Translation—32-Bit Implementations

Bit	Name	Description
0	T	T = 0 selects this format
1	Ks	Supervisor-state protection key
2	Kp	User-state protection key
3	N	No-execute protection bit
4–7	—	Reserved
8–31	VSID	Virtual segment ID

The Ks and Kp bits partially define the access protection for the pages within the segment. The virtual segment ID field is used as the high-order bits of the virtual page number (VPN).

The segment register instructions are summarized in Table 23. These instructions are privileged in that they are executable only while operating in supervisor mode.

Table 23. Segment Register Instructions—32-Bit Implementations Only

Instruction	Description
mtsr SR,**r**S	Move to Segment Register SR[SR]← **r**S
mtsrin rS,**r**B	Move to Segment Register Indirect SR[**r**B[0–3]]←**r**S
mfsr rD,SR	Move from Segment Register **r**D←SR[SR]
mfsrin rD,**r**B	Move from Segment Register Indirect **r**D←SR[**r**B[0–3]]

2.2.3 Segment Descriptors for Direct-Store Segments

The format of many of the fields in the segment descriptors depends on the value of the T bit. Figure 28 shows the format of segment descriptors (residing as STEs in segment tables) that define direct-store segments for 64-bit implementations (T bit is set).

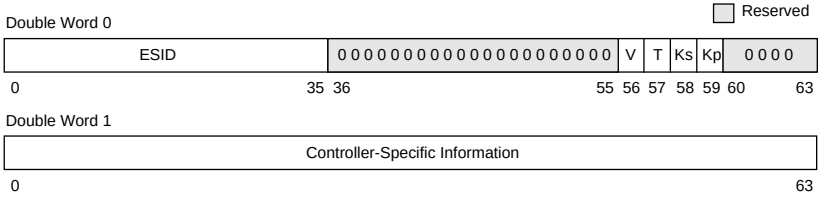


Figure 28. Segment Descriptor Format for Direct-Store Segments—64-Bit Implementations

Table 24 shows the bit definitions for the segment descriptors when the T bit is set for 64-bit implementations.

Table 24. Segment Descriptor Bit Definitions for Direct-Store Segments—64-Bit Implementations

Double Word	Bit	Name	Description
0	0–35	ESID	Effective segment ID
	36–55	—	Reserved
	56	V	Entry valid (V = 1) or invalid (V = 0)
	57	T	T = 0 selects this format
	58	Ks	Supervisor-state protection key
	59	Kp	User-state protection key
	61–63	—	Reserved
1	0–63	—	Device specific data for I/O controller

In 32-bit implementations, the segment descriptors reside in one of 16 segment registers. Figure 29 shows the register format for the segment registers when the T bit is set for 32-bit implementations.

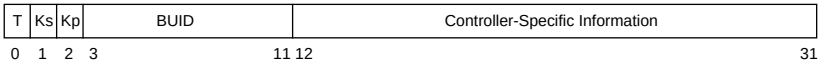


Figure 29. Segment Register Format for Direct-Store Segments—32-Bit Implementations

Table 25 shows the bit definitions for the segment registers when the T bit is set for 32-bit implementations.

Table 25. Segment Register Bit Definitions for Direct-Store Segments

Bit	Name	Description
0	T	T = 1 selects this format.
1	Ks	Supervisor-state protection key
2	Kp	User-state protection key
3–11	BUID	Bus unit ID
12–31	—	Device specific data for I/O controller

2.3 Page Table Entry (PTE) Definitions

Page table entries (PTEs) are generated and placed in page tables in memory by the operating system. The PowerPC OEA defines similar PTE formats for both 64- and 32-bit implementations in that the same fields are defined. However, 64-bit implementations define PTEs that are 128 bits in length while 32-bit implementations define PTEs that are 64 bits in length. Additionally, care must be taken when programming for both 64 and 32-bit implementations, as the bit placements of some fields are different. Some of the fields are defined as follows:

- The virtual segment ID field corresponds to the high-order bits of the virtual page number (VPN), and, along with the H, V, and API fields, it is used to locate the PTE (used as match criteria in comparing the PTE with the segment information).
- The R and C bits maintain history information for the page.
- The WIMG bits define the memory/cache control mode for accesses to the page.
- The PP bits define the remaining access protection constraints for the page.

Conceptually, the page table in memory must be searched to translate the address of every reference.

2.3.1 PTE Format for 64-Bit Implementations

In 64-bit implementations, each PTE is a 128-bit entity (two double words) that maps a virtual page number (VPN) to a physical page number (RPN). Information in the PTE is used in the page table search process (to determine a page table hit) and provides input to

the memory protection mechanism. Figure 30 shows the format of the two double words that comprise a PTE for 64-bit implementations.

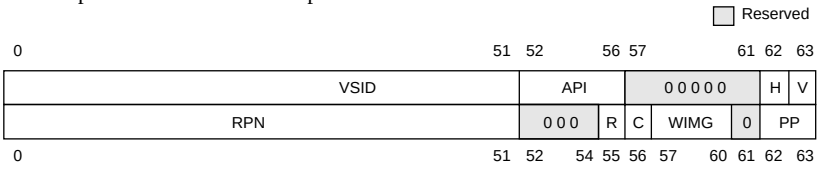


Figure 30. Page Table Entry Format—64-Bit Implementations

Table 26 lists the corresponding bit definitions for each double word in a PTE as defined above.

Table 26. PTE Bit Definitions—64-Bit Implementations

Double Word	Bit	Name	Description
0	0–51	VSID	Virtual segment ID—corresponds to the high-order bits of the virtual page number (VPN)
	52–56	API	Abbreviated page index
	57–61	—	Reserved
	62	H	Hash function identifier
	63	V	Entry valid (V = 1) or invalid (V = 0)
1	0–51	RPN	Physical page number
	52–54	—	Reserved
	55	R	Referenced bit
	56	C	Changed bit
	57–60	WIMG	Memory/cache access control bits
	61	—	Reserved
	62–63	PP	Page protection bits

The PTE contains an abbreviated page index rather than the complete page index field because at least 11 of the low-order bits of the page index are used in the hash function to select a PTE group (PTEG) address (PTEG addresses define the location of a PTE). Therefore, these 11 lower-order bits are not repeated in the PTEs of that PTEG.

2.3.2 PTE Format for 32-Bit Implementations

Figure 31 shows the format of the two words that comprise a PTE for 32-bit implementations.

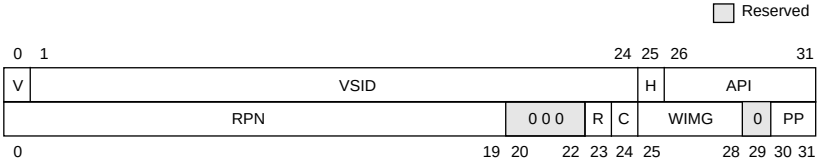


Figure 31. Page Table Entry Format—32-Bit Implementations

Table 27 lists the corresponding bit definitions for each word in a PTE as defined above.

Table 27. PTE Bit Definitions—32-Bit Implementations

Word	Bit	Name	Description
0	0	V	Entry valid (V = 1) or invalid (V = 0)
	1–24	VSID	Virtual segment ID
	25	H	Hash function identifier
	26–31	API	Abbreviated page index
1	0–19	RPN	Physical page number
	20–22	—	Reserved
	23	R	Referenced bit
	24	C	Changed bit
	25–28	WIMG	Memory/cache control bits
	29	—	Reserved
	30–31	PP	Page protection bits

In this case, the PTE contains an abbreviated page index rather than the complete page index field because at least ten of the low-order bits of the page index are used in the hash function to select a PTEG address (PTEG addresses define the location of a PTE). Therefore, these ten lower-order bits are not repeated in the PTEs of that PTEG.

2.3.3 SDR1 Register Definitions

The SDR1 register contains the control information for the page table structure in that it defines the highest order bits for the physical base address of the page table and it defines the size of the table. The format of the SDR1 register differs for 64-bit and 32-bit implementations, as shown below.

2.3.3.1 SDR1 Register Definition for 64-Bit Implementations

The format of the SDR1 register for a 64-bit implementation is shown in Figure 32 and the bit settings are described in Table 28.

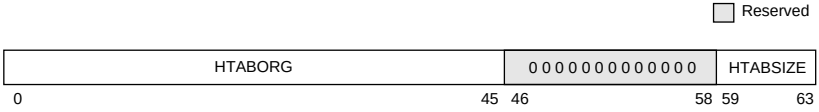


Figure 32. SDR1 Register Format—64-Bit Implementations

Table 28. SDR1 Register Bit Settings—64-Bit Implementations

Bits	Name	Description
0–45	HTABORG	Physical base address of page table
46–58	—	Reserved
59-63	HTABSIZE	Encoded size of page table (used to generate mask)

The HTABORG field in SDR1 contains the high-order 46 bits of the 64-bit physical address of the page table. Therefore, the beginning of the page table lies on a 2^{18} byte (256 Kbyte) boundary at a minimum. If the processor does not support 64 bits of physical address, software should write zeroes to those unsupported bits in the HTABORG field (as the implementation treats them as reserved). Otherwise, a machine check exception can occur.

A page table can be any size 2^n bytes where $18 \le n \le 46$. The HTABSIZE field in SDR1 contains an integer value that specifies how many bits from the output of the hashing function are used as the page table index. HTABSIZE is used to generate a mask of the form 0b00...011...1 (a string of (HTABSIZE – 28) 0 bits followed by a string of 1 bits). As the table size increases, more bits are used from the output of the hashing function to index into the table. The 1 bits in the mask determine how many additional bits (beyond the minimum of 11) from the hash are used in the index; the HTABORG field must have this same number of lower-order bits equal to 0.

2.3.3.2 SDR1 Register Definition for 32-Bit Implementations

The format of SDR1 for 32-bit implementations is similar to that of 64-bit implementations except that the register size is 32 bits and the HTABMASK field is programmed explicitly into SDR1. Additionally, the address ranges correspond to a 32-bit physical address and the range of page table sizes is smaller. Figure 33 shows the format of the SDR1 register for 32-bit implementations; the bit settings are described in Table 29.

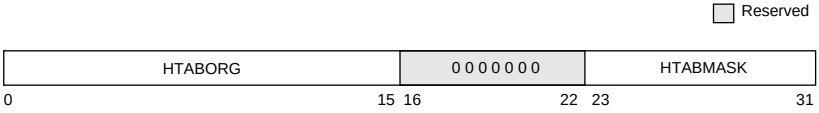


Figure 33. SDR1 Register Format—32-Bit Implementations

Table 29. SDR1 Register Bit Settings—32-Bit Implementations

Bits	Name	Description
0–15	HTABORG	Physical base address of page table
16–22	—	Reserved
23–31	HTABMASK	Mask for page table address

The HTABORG field in SDR1 contains the high-order 16 bits of the 32-bit physical address of the page table. Therefore, the beginning of the page table lies on a 2^{16} byte (64 Kbyte) boundary at a minimum. As with 64-bit implementations, if the processor does not support 32 bits of physical address, software should write zeroes to those unsupported bits in the HTABORG field (as the implementation treats them as reserved). Otherwise, a machine check exception can occur.

A page table can be any size 2^n bytes where $16 \le n \le 25$. The HTABMASK field in SDR1 contains a mask value that determines how many bits from the output of the hashing function are used as the page table index. This mask must be of the form 0b00...011...1 (a string of 0 bits followed by a string of 1 bits). As the table size increases, more bits are used from the output of the hashing function to index into the table. The 1 bits in HTABMASK determine how many additional bits (beyond the minimum of 10) from the hash are used in the index; the HTABORG field must have the same number of lower-order bits equal to 0 as the HTABMASK field has lower-order bits equal to 1.

Part 3 Exception Vectors

Exceptions, and conditions that cause them, are listed in Table 30.

Table 30. Exceptions and Conditions

Exception Type	Vector Offset (hex)	Causing Conditions
Reserved	00000	—
System reset	00100	The causes of system reset exceptions are implementation-dependent. If the conditions that cause the exception also cause the processor state to be corrupted such that the contents of SRR0 and SRR1 are no longer valid or such that other processor resources are so corrupted that the processor cannot reliably resume execution, the copy of the RI bit copied from the MSR to SRR1 is cleared.
Machine check	00200	The causes for machine check exceptions are implementation-dependent, but typically these causes are related to conditions such as bus parity errors or attempting to access an invalid physical address. Typically, these exceptions are triggered by an input signal to the processor. Note that not all processors provide the same level of error checking. The machine check exception is disabled when MSR[ME] = 0. If a machine check exception condition exists and the ME bit is cleared, the processor goes into the checkstop state. If the conditions that cause the exception also cause the processor state to be corrupted such that the contents of SRR0 and SRR1 are no longer valid or such that other processor resources are so corrupted that the processor cannot reliably resume execution, the copy of the RI bit copied from the MSR to SRR1 is cleared.
DSI	00300	A DSI exception occurs when a data memory access cannot be performed. Such accesses can be generated by load/store instructions, certain memory control instructions, and certain cache control instructions. For more detailed information, refer to Chapter 6, “Exceptions,” in *The Programming Environments Manual*.
ISI	00400	An ISI exception occurs when an instruction fetch cannot be performed. For more detailed information, refer to Chapter 6, “Exceptions,” in *The Programming Environments Manual*.
External interrupt	00500	An external interrupt is generated only when an external exception is pending (typically signaled by a signal defined by the implementation) and the interrupt is enabled (MSR[EE] = 1).
Alignment	00600	An alignment exception may occur when the processor cannot perform a memory access because of alignment or endian reasons. Note that an implementation is allowed to perform the operation correctly and not cause an alignment exception. For more detailed information, refer to Chapter 6, “Exceptions,” in *The Programming Environments Manual*.
Program	00700	A program exception is caused conditions which correspond to bit settings in SRR1 and arise during execution of an instruction. For more detailed information, refer to Chapter 6, “Exceptions,” in *The Programming Environments Manual*.
Floating-point unavailable	00800	A floating-point unavailable exception is caused by an attempt to execute a floating-point instruction (including floating-point load, store, and move instructions) when the floating-point available bit is cleared, MSR[FP] = 0.

Table 30. Exceptions and Conditions (Continued)

Exception Type	Vector Offset (hex)	Causing Conditions
Decrementer	00900	The decrementer interrupt exception is taken if the interrupt is enabled and the exception is pending. The exception is created when the most significant bit changes from 0 to 1. If it is not enabled, the exception remains pending until it is taken **.**
Reserved	00A00	Reserved for implementation-specific exceptions. For example, the PowerPC 601 microprocessor uses this vector offset for direct-store exceptions.
Reserved	00B00	—
System call	00C00	A system call exception occurs when a System Call (**sc**) instruction is executed.
Trace	00D00	The trace exception is optional. It occurs if either the MSR[SE] = 1 and any instruction (except **rfi**) successfully completed or MSR[BE] = 1 and a branch instruction is completed.
Floating-Point Assist	00E00	The floating-point assist exception is optional. This exception can be used to provide software assistance for infrequent and complex floating-point operations such as denormalization.
Reserved	00E10–00FFF	—
Reserved	01000–02FFF	Reserved for implementation-specific exceptions.

Part 4 PowerPC Instruction Set

The following sections include an instruction field summary, a list of split-field notation and conventions, and the entire PowerPC instruction set, sorted by mnemonic and opcode.

4.1 Instruction Field Summary

Table 31 describes the instruction fields used in the various instruction formats.

Table 31. Instruction Syntax Conventions

Field	Description
AA (30)	Absolute address bit. 0 The immediate field represents an address relative to the current instruction address (CIA). The effective (logical) address of the branch is either the sum of the LI field sign-extended to 64 bits and the address of the branch instruction or the sum of the BD field sign-extended to 64 bits and the address of the branch instruction. 1 The immediate field represents an absolute address. The effective address (EA) of the branch is the LI field sign-extended to 64 bits or the BD field sign-extended to 64 bits. Note: The LI and BD fields are sign-extended to 32 bits in 32-bit implementations.
BD (16–29)	Immediate field specifying a 14-bit signed two's complement branch displacement that is concatenated on the right with 0b00 and sign-extended to 64 bits (32 bits in 32-bit implementations).
BI (11–15)	Field used to specify a bit in the CR to be used as the condition of a branch conditional instruction.
BO (6–10)	Field used to specify options for the branch conditional instructions.

Table 31. Instruction Syntax Conventions (Continued)

Field	Description
crbA (11–15)	Field used to specify a bit in the CR to be used as a source.
crbB (16–20)	Field used to specify a bit in the CR to be used as a source.
crbD (6–10)	Field used to specify a bit in the CR, or in the FPSCR, as the destination of the result of an instruction.
crfD (6–8)	Field used to specify one of the CR fields, or one of the FPSCR fields, as a destination.
crfS (11–13)	Field used to specify one of the CR fields, or one of the FPSCR fields, as a source.
CRM (12–19)	Field mask used to identify the CR fields that are to be updated by the **mtcrf** instruction.
d (16–31)	Immediate field specifying a 16-bit signed two's complement integer that is sign-extended to 64 bits (32 bits in 32-bit implementations).
ds (16–29)	Immediate field specifying a 14-bit signed two's complement integer which is concatenated on the right with 0b00 and sign-extended to 64 bits. This field is defined in 64-bit implementations only.
FM (7–14)	Field mask used to identify the FPSCR fields that are to be updated by the **mtfsf** instruction.
frA (11–15)	Field used to specify an FPR as a source.
frB (16–20)	Field used to specify an FPR as a source.
frC (21–25)	Field used to specify an FPR as a source.
frD (6–10)	Field used to specify an FPR as the destination.
frS (6–10)	Field used to specify an FPR as a source.
IMM (16–19)	Immediate field used as the data to be placed into a field in the FPSCR.
L (10)	Field used to specify whether an integer compare instruction is to compare 64-bit numbers or 32-bit numbers. This field is defined in 64-bit implementations only.
LI (6–29)	Immediate field specifying a 24-bit signed two's complement integer that is concatenated on the right with 0b00 and sign-extended to 64 bits (32 bits in 32-bit implementations).
LK (31)	Link bit. 0 Does not update the link register (LR). 1 Updates the LR. If the instruction is a branch instruction, the address of the instruction following the branch instruction is placed into the LR.
MB (21–25) and ME (26–30)	Fields used in rotate instructions to specify a 64-bit mask (32 bits in 32-bit implementations) consisting of 1 bits from bit MB + 32 through bit ME + 32 inclusive, and 0 bits elsewhere.
NB (16–20)	Field used to specify the number of bytes to move in an immediate string load or store.
OE (21)	Used for extended arithmetic to enable setting OV and SO in the XER.
OPCD (0–5)	Primary opcode field.
rA (11–15)	Field used to specify a GPR to be used as a source or destination.
rB (16–20)	Field used to specify a GPR to be used as a source.

Table 31. Instruction Syntax Conventions (Continued)

Field	Description
Rc (31)	Record bit. 0 Does not update the condition register (CR). 1 Updates the CR to reflect the result of the operation. For integer instructions, CR bits 0–2 are set to reflect the result as a signed quantity and CR bit 3 receives a copy of the summary overflow bit, XER[SO]. The result as an unsigned quantity or a bit string can be deduced from the EQ bit. For floating-point instructions, CR bits 4–7 are set to reflect floating-point exception, floating-point enabled exception, floating-point invalid operation exception, and floating-point overflow exception. (Note that the architecture specification refers to exceptions also as interrupts.)
rD (6–10)	Field used to specify a GPR to be used as a destination.
rS (6–10)	Field used to specify a GPR to be used as a source.
SH (16–20)	Field used to specify a shift amount.
SIMM (16–31)	Immediate field used to specify a 16-bit signed integer.
SR (12–15)	Field used to specify one of the 16 segment registers (32-bit implementations only).
TO (6–10)	Field used to specify the conditions on which to trap.
UIMM (16–31)	Immediate field used to specify a 16-bit unsigned integer.
XO (21–29, 21–30, 22–30, 26–30, 27–29, 27–30, or 30–31)	Extended opcode field. Bits 21–29, 27–29, 27–30, 30–31 pertain to 64-bit implementations only.

Split fields—mb, me, sh, spr, and tbr—are described in Table 32.

Table 32. Split-Field Notation and Conventions

Field	Description
mb (21–26)	Field used in rotate instructions to specify the first 1 bit of a 64-bit mask (32 bits in 32-bit implementations). This field is defined in 64-bit implementations only.
me (21–26)	Field used in rotate instructions to specify the last 1 bit of a 64-bit mask (32 bits in 32-bit implementations). This field is defined in 64-bit implementations only.
sh (16–20) and sh (30)	Fields used to specify a shift amount (64-bit implementations only).
spr (11–20)	Field used to specify a special purpose register for the **mtspr** and **mfspr** instructions.
tbr (11–20)	Field used to specify either the time base lower (TBL) or time base upper (TBU).

4.2 PowerPC Instruction Set Listings

This section lists the PowerPC architecture's instruction set. Instructions are sorted by mnemonic and opcode. Note that split fields, that represent the concatenation of sequences from left to right, are shown in lowercase.

Table 33 lists the instructions implemented in the PowerPC architecture in alphabetical order by mnemonic.

Key:
Reserved bits

Table 33. Complete Instruction List Sorted by Mnemonic

Name	0	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
add*x*	31	D					A					B					OE	266									Rc
addc*x*	31	D					A					B					OE	10									Rc
adde*x*	31	D					A					B					OE	138									Rc
addi	14	D					A					SIMM															
addic	12	D					A					SIMM															
addic.	13	D					A					SIMM															
addis	15	D					A					SIMM															
addme*x*	31	D					A					0	0	0	0	0	OE	234									Rc
addze*x*	31	D					A					0	0	0	0	0	OE	202									Rc
and*x*	31	S					A					B					28										Rc
andc*x*	31	S					A					B					60										Rc
andi.	28	S					A					UIMM															
andis.	29	S					A					UIMM															
b*x*	18	LI																								AA	LK
bc*x*	16	BO					BI					BD														AA	LK
bcctr*x*	19	BO					BI					0	0	0	0	0	528										LK
bclr*x*	19	BO					BI					0	0	0	0	0	16										LK
cmp	31	crfD			0	L	A					B					0										0
cmpi	11	crfD			0	L	A					SIMM															
cmpl	31	crfD			0	L	A					B					32										0
cmpli	10	crfD			0	L	A					UIMM															
cntlzd*x* [4]	31	S					A					0	0	0	0	0	58										Rc
cntlzw*x*	31	S					A					0	0	0	0	0	26										Rc
crand	19	crbD					crbA					crbB					257										0

Name	0	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
crandc	19	crbD					crbA					crbB					129										0
creqv	19	crbD					crbA					crbB					289										0
crnand	19	crbD					crbA					crbB					225										0
crnor	19	crbD					crbA					crbB					33										0
cror	19	crbD					crbA					crbB					449										0
crorc	19	crbD					crbA					crbB					417										0
crxor	19	crbD					crbA					crbB					193										0
dcbf	31	0 0 0 0 0					A					B					86										0
dcbi [1]	31	0 0 0 0 0					A					B					470										0
dcbst	31	0 0 0 0 0					A					B					54										0
dcbt	31	0 0 0 0 0					A					B					278										0
dcbtst	31	0 0 0 0 0					A					B					246										0
dcbz	31	0 0 0 0 0					A					B					1014										0
divd*x* [4]	31	D					A					B				OE	489										Rc
divdu*x* [4]	31	D					A					B				OE	457										Rc
divw*x*	31	D					A					B				OE	491										Rc
divwu*x*	31	D					A					B				OE	459										Rc
eciwx	31	D					A					B					310										0
ecowx	31	S					A					B					438										0
eieio	31	0 0 0 0 0					0 0 0 0 0					0 0 0 0 0					854										0
eqv*x*	31	S					A					B					284										Rc
extsb*x*	31	S					A					0 0 0 0 0					954										Rc
extsh*x*	31	S					A					0 0 0 0 0					922										Rc
extsw*x* [4]	31	S					A					0 0 0 0 0					986										Rc
fabs*x*	63	D					0 0 0 0 0					B					264										Rc
fadd*x*	63	D					A					B				0 0 0 0 0					21						Rc
fadds*x*	59	D					A					B				0 0 0 0 0					21						Rc
fcfid*x* [4]	63	D					0 0 0 0 0					B					846										Rc
fcmpo	63	crfD			0 0		A					B					32										0
fcmpu	63	crfD			0 0		A					B					0										0
fctid*x* [4]	63	D					0 0 0 0 0					B					814										Rc
fctidz*x* [4]	63	D					0 0 0 0 0					B					815										Rc
fctiw*x*	63	D					0 0 0 0 0					B					14										Rc

Name	0	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
fctiwz*x*	63	D					0	0	0	0	0	B					15										Rc
fdiv*x*	63	D					A					B					0	0	0	0	0	18					Rc
fdivs*x*	59	D					A					B					0	0	0	0	0	18					Rc
fmadd*x*	63	D					A					B					C					29					Rc
fmadds*x*	59	D					A					B					C					29					Rc
fmr*x*	63	D					0	0	0	0	0	B					72										Rc
fmsub*x*	63	D					A					B					C					28					Rc
fmsubs*x*	59	D					A					B					C					28					Rc
fmul*x*	63	D					A					0	0	0	0	0	C					25					Rc
fmuls*x*	59	D					A					0	0	0	0	0	C					25					Rc
fnabs*x*	63	D					0	0	0	0	0	B					136										Rc
fneg*x*	63	D					0	0	0	0	0	B					40										Rc
fnmadd*x*	63	D					A					B					C					31					Rc
fnmadds*x*	59	D					A					B					C					31					Rc
fnmsub*x*	63	D					A					B					C					30					Rc
fnmsubs*x*	59	D					A					B					C					30					Rc
fres*x* [5]	59	D					0	0	0	0	0	B					0	0	0	0	0	24					Rc
frsp*x*	63	D					0	0	0	0	0	B					12										Rc
frsqrte*x* [5]	63	D					0	0	0	0	0	B					0	0	0	0	0	26					Rc
fsel*x* [5]	63	D					A					B					C					23					Rc
fsqrt*x* [5]	63	D					0	0	0	0	0	B					0	0	0	0	0	22					Rc
fsqrts*x* [5]	59	D					0	0	0	0	0	B					0	0	0	0	0	22					Rc
fsub*x*	63	D					A					B					0	0	0	0	0	20					Rc
fsubs*x*	59	D					A					B					0	0	0	0	0	20					Rc
icbi	31	0	0	0	0	0	A					B					982										0
isync	19	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	150										0
lbz	34	D					A					d															
lbzu	35	D					A					d															
lbzux	31	D					A					B					119										0
lbzx	31	D					A					B					87										0
ld [4]	58	D					A					ds														0	
ldarx [4]	31	D					A					B					84										0
ldu [4]	58	D					A					ds														1	

Name	0	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
ldux [4]	31	D					A					B					53										0
ldx [4]	31	D					A					B					21										0
lfd	50	D					A					d															
lfdu	51	D					A					d															
lfdux	31	D					A					B					631										0
lfdx	31	D					A					B					599										0
lfs	48	D					A					d															
lfsu	49	D					A					d															
lfsux	31	D					A					B					567										0
lfsx	31	D					A					B					535										0
lha	42	D					A					d															
lhau	43	D					A					d															
lhaux	31	D					A					B					375										0
lhax	31	D					A					B					343										0
lhbrx	31	D					A					B					790										0
lhz	40	D					A					d															
lhzu	41	D					A					d															
lhzux	31	D					A					B					311										0
lhzx	31	D					A					B					279										0
lmw [3]	46	D					A					d															
lswi [3]	31	D					A					NB					597										0
lswx [3]	31	D					A					B					533										0
lwa [4]	58	D					A					ds															2
lwarx	31	D					A					B					20										0
lwaux [4]	31	D					A					B					373										0
lwax [4]	31	D					A					B					341										0
lwbrx	31	D					A					B					534										0
lwz	32	D					A					d															
lwzu	33	D					A					d															
lwzux	31	D					A					B					55										0
lwzx	31	D					A					B					23										0
mcrf	19	crfD			0	0	crfS			0	0	0	0	0	0	0	0										0
mcrfs	63	crfD			0	0	crfS			0	0	0	0	0	0	0	0										0

Name	0	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
mcrxr	31	crfD			0	0	0	0	0	0	0	0	0	0	0	0	512										0
mfcr	31	D					0	0	0	0	0	0	0	0	0	0	19										0
mffs*x*	63	D					0	0	0	0	0	0	0	0	0	0	583										Rc
mfmsr [1]	31	D					0	0	0	0	0	0	0	0	0	0	83										0
mfspr [2]	31	D					spr										339										0
mfsr [1,6]	31	D					0	SR					0	0	0	0	595										0
mfsrin [1,6]	31	D					0	0	0	0	0	B					659										0
mftb	31	D					tbr										371										0
mtcrf	31	S					0	CRM								0	144										0
mtfsb0*x*	63	crbD					0	0	0	0	0	0	0	0	0	0	70										Rc
mtfsb1*x*	63	crbD					0	0	0	0	0	0	0	0	0	0	38										Rc
mtfsf*x*	63	0	FM								0	B					711										Rc
mtfsfi*x*	63	crfD			0	0	0	0	0	0	0	IMM				0	134										Rc
mtmsr [1]	31	S					0	0	0	0	0	0	0	0	0	0	146										0
mtspr [2]	31	S					spr										467										0
mtsr [1,6]	31	S					0	SR					0	0	0	0	210										0
mtsrin [1,6]	31	S					0	0	0	0	0	B					242										0
mulhd*x* [4]	31	D					A					B					0	73									Rc
mulhdu*x* [4]	31	D					A					B					0	9									Rc
mulhw*x*	31	D					A					B					0	75									Rc
mulhwu*x*	31	D					A					B					0	11									Rc
mulld*x* [4]	31	D					A					B					OE	233									Rc
mulli	7	D					A					SIMM															
mullw*x*	31	D					A					B					OE	235									Rc
nand*x*	31	S					A					B					476										Rc
neg*x*	31	D					A					0	0	0	0	0	OE	104									Rc
nor*x*	31	S					A					B					124										Rc
or*x*	31	S					A					B					444										Rc
orc*x*	31	S					A					B					412										Rc
ori	24	S					A					UIMM															
oris	25	S					A					UIMM															
rfi [1]	19	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	50										0
rldcl*x* [4]	30	S					A					B					mb						8				Rc

Name	0	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
rldcr*x* [4]	30	S					A					B					me						9			Rc
rldic*x* [4]	30	S					A					sh					mb						2		sh	Rc
rldicl*x* [4]	30	S					A					sh					mb						0		sh	Rc
rldicr*x* [4]	30	S					A					sh					me						1		sh	Rc
rldimi*x* [4]	30	S					A					sh					mb						3		sh	Rc
rlwimi*x*	20	S					A					SH					MB						ME			Rc
rlwinm*x*	21	S					A					SH					MB						ME			Rc
rlwnm*x*	23	S					A					B					MB						ME			Rc
sc	17	0 0 0 0 0					0 0 0 0 0					0 0 0 0 0 0 0 0 0 0 0 0 0 0													1	0
slbia [1,4,5]	31	0 0 0 0 0					0 0 0 0 0					0 0 0 0 0					498									0
slbie [1,4,5]	31	0 0 0 0 0					0 0 0 0 0					B					434									0
sld*x* [4]	31	S					A					B					27									Rc
slw*x*	31	S					A					B					24									Rc
srad*x* [4]	31	S					A					B					794									Rc
sradi*x* [4]	31	S					A					sh					413								sh	Rc
sraw*x*	31	S					A					B					792									Rc
srawi*x*	31	S					A					SH					824									Rc
srd*x* [4]	31	S					A					B					539									Rc
srw*x*	31	S					A					B					536									Rc
stb	38	S					A					d														
stbu	39	S					A					d														
stbux	31	S					A					B					247									0
stbx	31	S					A					B					215									0
std [4]	62	S					A					ds													0	
stdcx. [4]	31	S					A					B					214									1
stdu [4]	62	S					A					ds													1	
stdux [4]	31	S					A					B					181									0
stdx [4]	31	S					A					B					149									0
stfd	54	S					A					d														
stfdu	55	S					A					d														
stfdux	31	S					A					B					759									0
stfdx	31	S					A					B					727									0
stfiwx [5]	31	S					A					B					983									0

Name	0	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
stfs	52	S					A					d															
stfsu	53	S					A					d															
stfsux	31	S					A					B				695											0
stfsx	31	S					A					B				663											0
sth	44	S					A					d															
sthbrx	31	S					A					B				918											0
sthu	45	S					A					d															
sthux	31	S					A					B				439											0
sthx	31	S					A					B				407											0
stmw [3]	47	S					A					d															
stswi [3]	31	S					A					NB				725											0
stswx [3]	31	S					A					B				661											0
stw	36	S					A					d															
stwbrx	31	S					A					B				662											0
stwcx.	31	S					A					B				150											1
stwu	37	S					A					d															
stwux	31	S					A					B				183											0
stwx	31	S					A					B				151											0
subf*x*	31	D					A					B				OE		40									Rc
subfc*x*	31	D					A					B				OE		8									Rc
subfe*x*	31	D					A					B				OE		136									Rc
subfic	08	D					A					SIMM															
subfme*x*	31	D					A					0 0 0 0 0				OE		232									Rc
subfze*x*	31	D					A					0 0 0 0 0				OE		200									Rc
sync	31	0 0 0 0 0					0 0 0 0 0					0 0 0 0 0				598											0
td [4]	31	TO					A					B				68											0
tdi [4]	02	TO					A					SIMM															
tlbia [1,5]	31	0 0 0 0 0					0 0 0 0 0					0 0 0 0 0				370											0
tlbie [1,5]	31	0 0 0 0 0					0 0 0 0 0					B				306											0
tlbsync [1,5]	31	0 0 0 0 0					0 0 0 0 0					0 0 0 0 0				566											0
tw	31	TO					A					B				4											0
twi	03	TO					A					SIMM															
xor*x*	31	S					A					B				316											Rc

Name	0	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
xori	26	S						A						UIMM													
xoris	27	S						A						UIMM													

[1] Supervisor-level instruction
[2] Supervisor- and user-level instruction
[3] Load and store string or multiple instruction
[4] 64-bit instruction
[5] Optional instruction
[6] 32-bit instruction only

Table 34 lists the instructions defined in the PowerPC architecture in numeric order by opcode.

Key:
Reserved bits

Table 34. Complete Instruction List Sorted by Opcode

Name	0	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
tdi [4]	000010	TO					A						SIMM														
twi	000011	TO					A						SIMM														
mulli	000111	D					A						SIMM														
subfic	001000	D					A						SIMM														
cmpli	001010	crfD			0	L	A						UIMM														
cmpi	001011	crfD			0	L	A						SIMM														
addic	001100	D					A						SIMM														
addic.	001101	D					A						SIMM														
addi	001110	D					A						SIMM														
addis	001111	D					A						SIMM														
bc*x*	010000	BO					BI						BD													AA	LK
sc	010001	00000					00000						00000000000000													1	0
b*x*	010010	LI																								AA	LK
mcrf	010011	crfD			00			crfS			00		00000					0000000000									0
bclr*x*	010011	BO					BI						00000					0000010000									LK
crnor	010011	crbD					crbA						crbB					0000100001									0
rfi	010011	00000					00000						00000					0000110010									0
crandc	010011	crbD					crbA						crbB					0010000001									0
isync	010011	00000					00000						00000					0010010110									0
crxor	010011	crbD					crbA						crbB					0011000001									0

Name	0	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
crnand	0 1 0 0 1 1	crbD						crbA					crbB					0 0 1 1 1 0 0 0 0 1									0
crand	0 1 0 0 1 1	crbD						crbA					crbB					0 1 0 0 0 0 0 0 0 1									0
creqv	0 1 0 0 1 1	crbD						crbA					crbB					0 1 0 0 1 0 0 0 0 1									0
crorc	0 1 0 0 1 1	crbD						crbA					crbB					0 1 1 0 1 0 0 0 0 1									0
cror	0 1 0 0 1 1	crbD						crbA					crbB					0 1 1 1 0 0 0 0 0 1									0
bcctr*x*	0 1 0 0 1 1	BO						BI					0 0 0 0 0					1 0 0 0 0 0 1 0 0 0 0									LK
rlwimi*x*	0 1 0 1 0 0	S						A					SH					MB					ME				Rc
rlwinm*x*	0 1 0 1 0 1	S						A					SH					MB					ME				Rc
rlwnm*x*	0 1 0 1 1 1	S						A					B					MB					ME				Rc
ori	0 1 1 0 0 0	S						A					UIMM															
oris	0 1 1 0 0 1	S						A					UIMM															
xori	0 1 1 0 1 0	S						A					UIMM															
xoris	0 1 1 0 1 1	S						A					UIMM															
andi.	0 1 1 1 0 0	S						A					UIMM															
andis.	0 1 1 1 0 1	S						A					UIMM															
rldicl*x* [4]	0 1 1 1 1 0	S						A					sh					mb					0 0 0			sh	Rc
rldicr*x* [4]	0 1 1 1 1 0	S						A					sh					me					0 0 1			sh	Rc
rldic*x* [4]	0 1 1 1 1 0	S						A					sh					mb					0 1 0			sh	Rc
rldimi*x* [4]	0 1 1 1 1 0	S						A					sh					mb					0 1 1			sh	Rc
rldcl*x* [4]	0 1 1 1 1 0	S						A					B					mb					0 1 0 0 0				Rc
rldcr*x* [4]	0 1 1 1 1 0	S						A					B					me					0 1 0 0 1				Rc
cmp	0 1 1 1 1 1	crfD			0	L		A					B					0 0 0 0 0 0 0 0 0 0									0
tw	0 1 1 1 1 1	TO						A					B					0 0 0 0 0 0 0 1 0 0									0
subfc*x*	0 1 1 1 1 1	D						A					B				OE	0 0 0 0 0 0 1 0 0 0									Rc
mulhdu*x* [4]	0 1 1 1 1 1	D						A					B				0	0 0 0 0 0 0 1 0 0 1									Rc
addc*x*	0 1 1 1 1 1	D						A					B				OE	0 0 0 0 0 0 1 0 1 0									Rc
mulhwu*x*	0 1 1 1 1 1	D						A					B				0	0 0 0 0 0 0 1 0 1 1									Rc
mfcr	0 1 1 1 1 1	D						0 0 0 0 0					0 0 0 0 0					0 0 0 0 0 1 0 0 1 1									0
lwarx	0 1 1 1 1 1	D						A					B					0 0 0 0 0 1 0 1 0 0									0
ldx [4]	0 1 1 1 1 1	D						A					B					0 0 0 0 0 1 0 1 0 1									0
lwzx	0 1 1 1 1 1	D						A					B					0 0 0 0 0 1 0 1 1 1									0
slw*x*	0 1 1 1 1 1	S						A					B					0 0 0 0 0 1 1 0 0 0									Rc
cntlzw*x*	0 1 1 1 1 1	S						A					0 0 0 0 0					0 0 0 0 0 1 1 0 1 0									Rc

Name	0	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
sld*x* [4]	0 1 1 1 1 1	S						A					B					0	0	0	0	0	1	1	0	1	1	Rc
and*x*	0 1 1 1 1 1	S						A					B					0	0	0	0	0	1	1	1	0	0	Rc
cmpl	0 1 1 1 1 1	crfD			0	L		A					B					0	0	0	0	1	0	0	0	0	0	0
subf*x*	0 1 1 1 1 1	D						A					B					OE	0	0	0	1	0	1	0	0	0	Rc
ldux [4]	0 1 1 1 1 1	D						A					B					0	0	0	0	1	1	0	1	0	1	0
dcbst	0 1 1 1 1 1	0 0 0 0 0						A					B					0	0	0	0	1	1	0	1	1	0	0
lwzux	0 1 1 1 1 1	D						A					B					0	0	0	0	1	1	0	1	1	1	0
cntlzd*x* [4]	0 1 1 1 1 1	S						A					0 0 0 0 0					0	0	0	0	1	1	1	0	1	0	Rc
andc*x*	0 1 1 1 1 1	S						A					B					0	0	0	0	1	1	1	1	0	0	Rc
td [4]	0 1 1 1 1 1	TO						A					B					0	0	0	0	1	0	0	0	1	0	0
mulhd*x* [4]	0 1 1 1 1 1	D						A					B					0	0	0	0	1	0	0	1	0	0	1	Rc
mulhw*x*	0 1 1 1 1 1	D						A					B					0	0	0	0	1	0	0	1	0	1	1	Rc
mfmsr	0 1 1 1 1 1	D						0 0 0 0 0					0 0 0 0 0					0	0	0	1	0	0	1	0	0	1	1	0
ldarx [4]	0 1 1 1 1 1	D						A					B					0	0	0	1	0	1	0	1	0	0	0	0
dcbf	0 1 1 1 1 1	0 0 0 0 0						A					B					0	0	0	1	0	1	0	1	1	0	0	0
lbzx	0 1 1 1 1 1	D						A					B					0	0	0	1	0	1	0	1	1	1	0	0
neg*x*	0 1 1 1 1 1	D						A					0 0 0 0 0					OE	0	0	0	1	1	0	1	0	0	0	Rc
lbzux	0 1 1 1 1 1	D						A					B					0	0	0	1	1	1	0	1	1	1	0	0
nor*x*	0 1 1 1 1 1	S						A					B					0	0	0	1	1	1	1	0	0	0	Rc
subfe*x*	0 1 1 1 1 1	D						A					B					OE	0	0	1	0	0	0	1	0	0	0	Rc
adde*x*	0 1 1 1 1 1	D						A					B					OE	0	0	1	0	0	0	1	0	1	0	Rc
mtcrf	0 1 1 1 1 1	S						0	CRM								0	0	0	1	0	0	1	0	0	0	0	0
mtmsr	0 1 1 1 1 1	S						0 0 0 0 0					0 0 0 0 0					0	0	1	0	0	1	0	0	1	0	0
stdx [4]	0 1 1 1 1 1	S						A					B					0	0	1	0	0	1	0	1	0	1	0
stwcx.	0 1 1 1 1 1	S						A					B					0	0	1	0	0	1	0	1	1	0	1
stwx	0 1 1 1 1 1	S						A					B					0	0	1	0	0	1	0	1	1	1	0
stdux [4]	0 1 1 1 1 1	S						A					B					0	0	1	0	1	1	0	1	0	1	0
stwux	0 1 1 1 1 1	S						A					B					0	0	1	0	1	1	0	1	1	1	0
subfze*x*	0 1 1 1 1 1	D						A					0 0 0 0 0					OE	0	1	0	0	1	1	0	0	0	Rc
addze*x*	0 1 1 1 1 1	D						A					0 0 0 0 0					OE	0	1	0	0	1	1	0	0	1	0	Rc
mtsr [1,6]	0 1 1 1 1 1	S						0	SR				0 0 0 0 0					0	0	1	0	0	1	1	0	0	1	0	0
stdcx. [4]	0 1 1 1 1 1	S						A					B					0	0	1	0	1	1	0	1	0	1	1	0
stbx	0 1 1 1 1 1	S						A					B					0	0	1	0	1	1	0	1	1	1	1	0

Name	0	5	6 7 8 9	10 11 12 13 14	15 16 17 18 19	20	21 22 23 24 25 26 27 28 29	30	31
subfme*x*	011111		D	A	00000	OE	0011101000		Rc
mulld [4]	011111		D	A	B	OE	0011101001		Rc
addme*x*	011111		D	A	00000	OE	0011101010		Rc
mullw*x*	011111		D	A	B	OE	0011101011		Rc
mtsrin [1,6]	011111		S	00000	B		0011110010		0
dcbtst	011111		00000	A	B		0011110110		0
stbux	011111		S	A	B		0011110111		0
add*x*	011111		D	A	B	OE	0100001010		Rc
dcbt	011111		00000	A	B		0100010110		0
lhzx	011111		D	A	B		0100010111		0
eqv*x*	011111		S	A	B		0100011100		Rc
tlbie [1,5]	011111		00000	00000	B		0100110010		0
eciwx	011111		D	A	B		0100110110		0
lhzux	011111		D	A	B		0100110111		0
xor*x*	011111		S	A	B		0100111100		Rc
mfspr [2]	011111		D	spr			0101010011		0
lwax [4]	011111		D	A	B		0101010101		0
lhax	011111		D	A	B		0101010111		0
tlbia [1,5]	011111		00000	00000	00000		0101110010		0
mftb	011111		D	tbr			0101110011		0
lwaux [4]	011111		D	A	B		0101110101		0
lhaux	011111		D	A	B		0101110111		0
sthx	011111		S	A	B		0110010111		0
orc*x*	011111		S	A	B		0110011100		Rc
sradi*x* [4]	011111		S	A	sh		1100111011	sh	Rc
slbie [1,4,5]	011111		00000	00000	B		0110110010		0
ecowx	011111		S	A	B		0110110110		0
sthux	011111		S	A	B		0110110111		0
or*x*	011111		S	A	B		0110111100		Rc
divdu*x* [4]	011111		D	A	B	OE	0111001001		Rc
divwu*x*	011111		D	A	B	OE	0111001011		Rc
mtspr [2]	011111		S	spr			0111010011		0
dcbi	011111		00000	A	B		0111010110		0

Name	0	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
nand*x*	0 1 1 1 1 1		S					A					B					0 1 1 1 0 1 1 1 0 0										Rc
divd*x* [4]	0 1 1 1 1 1		D					A					B					OE	0 1 1 1 1 0 1 0 0 1									Rc
divw*x*	0 1 1 1 1 1		D					A					B					OE	0 1 1 1 1 0 1 0 1 1									Rc
slbia [1,4,5]	0 1 1 1 1 1		0 0 0 0 0					0 0 0 0 0					0 0 0 0 0					0 1 1 1 1 1 0 0 1 0										0
mcrxr	0 1 1 1 1 1		crfD			0 0		0 0 0 0 0					0 0 0 0 0					1 0 0 0 0 0 0 0 0 0										0
lswx [3]	0 1 1 1 1 1		D					A					B					1 0 0 0 0 1 0 1 0 1										0
lwbrx	0 1 1 1 1 1		D					A					B					1 0 0 0 0 1 0 1 1 0										0
lfsx	0 1 1 1 1 1		D					A					B					1 0 0 0 0 1 0 1 1 1										0
srw*x*	0 1 1 1 1 1		S					A					B					1 0 0 0 0 1 1 0 0 0										Rc
srd*x* [4]	0 1 1 1 1 1		S					A					B					1 0 0 0 0 1 1 0 1 1										Rc
tlbsync [1,5]	0 1 1 1 1 1		0 0 0 0 0					0 0 0 0 0					0 0 0 0 0					1 0 0 0 1 1 0 1 1 0										0
lfsux	0 1 1 1 1 1		D					A					B					1 0 0 0 1 1 0 1 1 1										0
mfsr [1,6]	0 1 1 1 1 1		D					0	SR				0 0 0 0 0					1 0 0 1 0 1 0 0 1 1										0
lswi [3]	0 1 1 1 1 1		D					A					NB					1 0 0 1 0 1 0 1 0 1										0
sync	0 1 1 1 1 1		0 0 0 0 0					0 0 0 0 0					0 0 0 0 0					1 0 0 1 0 1 0 1 1 0										0
lfdx	0 1 1 1 1 1		D					A					B					1 0 0 1 0 1 0 1 1 1										0
lfdux	0 1 1 1 1 1		D					A					B					1 0 0 1 1 1 0 1 1 1										0
mfsrin [1,6]	0 1 1 1 1 1		D					0 0 0 0 0					B					1 0 1 0 0 1 0 0 1 1										0
stswx [3]	0 1 1 1 1 1		S					A					B					1 0 1 0 0 1 0 1 0 1										0
stwbrx	0 1 1 1 1 1		S					A					B					1 0 1 0 0 1 0 1 1 0										0
stfsx	0 1 1 1 1 1		S					A					B					1 0 1 0 0 1 0 1 1 1										0
stfsux	0 1 1 1 1 1		S					A					B					1 0 1 0 1 1 0 1 1 1										0
stswi [3]	0 1 1 1 1 1		S					A					NB					1 0 1 1 0 1 0 1 0 1										0
stfdx	0 1 1 1 1 1		S					A					B					1 0 1 1 0 1 0 1 1 1										0
stfdux	0 1 1 1 1 1		S					A					B					1 0 1 1 1 1 0 1 1 1										0
lhbrx	0 1 1 1 1 1		D					A					B					1 1 0 0 0 1 0 1 1 0										0
sraw*x*	0 1 1 1 1 1		S					A					B					1 1 0 0 0 1 1 0 0 0										Rc
srad*x* [4]	0 1 1 1 1 1		S					A					B					1 1 0 0 0 1 1 0 1 0										Rc
srawi*x*	0 1 1 1 1 1		S					A					SH					1 1 0 0 1 1 1 0 0 0										Rc
eieio	0 1 1 1 1 1		0 0 0 0 0					0 0 0 0 0					0 0 0 0 0					1 1 0 1 0 1 0 1 1 0										0
sthbrx	0 1 1 1 1 1		S					A					B					1 1 1 0 0 1 0 1 1 0										0
extsh*x*	0 1 1 1 1 1		S					A					0 0 0 0 0					1 1 1 0 0 1 1 0 1 0										Rc
extsb*x*	0 1 1 1 1 1		S					A					0 0 0 0 0					1 1 1 0 1 1 1 0 1 0										Rc

Name	0 5	6 7 8 9 10	11 12 13 14 15	16 17 18 19 20	21 22 23 24 25 26 27 28 29 30	31
icbi	011111	00000	A	B	1111010110	0
stfiwx [5]	011111	S	A	B	1111010111	0
extsw [4]	011111	S	A	00000	1111011010	Rc
dcbz	011111	00000	A	B	1111110110	0
lwz	100000	D	A	d		
lwzu	100001	D	A	d		
lbz	100010	D	A	d		
lbzu	100011	D	A	d		
stw	100100	S	A	d		
stwu	100101	S	A	d		
stb	100110	S	A	d		
stbu	100111	S	A	d		
lhz	101000	D	A	d		
lhzu	101001	D	A	d		
lha	101010	D	A	d		
lhau	101011	D	A	d		
sth	101100	S	A	d		
sthu	101101	S	A	d		
lmw [3]	101110	D	A	d		
stmw [3]	101111	S	A	d		
lfs	110000	D	A	d		
lfsu	110001	D	A	d		
lfd	110010	D	A	d		
lfdu	110011	D	A	d		
stfs	110100	S	A	d		
stfsu	110101	S	A	d		
stfd	110110	S	A	d		
stfdu	110111	S	A	d		
ld [4]	111010	D	A	ds		00
ldu [4]	111010	D	A	ds		01
lwa [4]	111010	D	A	ds		10
fdivs*x*	111011	D	A	B	00000 10010	Rc
fsubs*x*	111011	D	A	B	00000 10100	Rc

Name	0	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
fadds*x*	1 1 1 0 1 1		D					A					B					0 0 0 0 0					1 0 1 0 1				Rc
fsqrts*x* [5]	1 1 1 0 1 1		D					0 0 0 0 0					B					0 0 0 0 0					1 0 1 1 0				Rc
fres*x* [5]	1 1 1 0 1 1		D					0 0 0 0 0					B					0 0 0 0 0					1 1 0 0 0				Rc
fmuls*x*	1 1 1 0 1 1		D					A					0 0 0 0 0					C					1 1 0 0 1				Rc
fmsubs*x*	1 1 1 0 1 1		D					A					B					C					1 1 1 0 0				Rc
fmadds*x*	1 1 1 0 1 1		D					A					B					C					1 1 1 0 1				Rc
fnmsubs*x*	1 1 1 0 1 1		D					A					B					C					1 1 1 1 0				Rc
fnmadds*x*	1 1 1 0 1 1		D					A					B					C					1 1 1 1 1				Rc
std [4]	1 1 1 1 1 0		S					A					ds													0	0
stdu [4]	1 1 1 1 1 0		S					A					ds													0	1
fcmpu	1 1 1 1 1 1		crfD			0	0	A					B					0 0 0 0 0 0 0 0 0 0									0
frsp*x*	1 1 1 1 1 1		D					0 0 0 0 0					B					0 0 0 0 0 0 1 1 0 0									Rc
fctiw*x*	1 1 1 1 1 1		D					0 0 0 0 0					B					0 0 0 0 0 0 1 1 1 0									
fctiwz*x*	1 1 1 1 1 1		D					0 0 0 0 0					B					0 0 0 0 0 0 1 1 1 1									Rc
fdiv*x*	1 1 1 1 1 1		D					A					B					0 0 0 0 0					1 0 0 1 0				Rc
fsub*x*	1 1 1 1 1 1		D					A					B					0 0 0 0 0					1 0 1 0 0				Rc
fadd*x*	1 1 1 1 1 1		D					A					B					0 0 0 0 0					1 0 1 0 1				Rc
fsqrt*x* [5]	1 1 1 1 1 1		D					0 0 0 0 0					B					0 0 0 0 0					1 0 1 1 0				Rc
fsel*x* [5]	1 1 1 1 1 1		D					A					B					C					1 0 1 1 1				Rc
fmul*x*	1 1 1 1 1 1		D					A					0 0 0 0 0					C					1 1 0 0 1				Rc
frsqrte*x* [5]	1 1 1 1 1 1		D					0 0 0 0 0					B					0 0 0 0 0					1 1 0 1 0				Rc
fmsub*x*	1 1 1 1 1 1		D					A					B					C					1 1 1 0 0				Rc
fmadd*x*	1 1 1 1 1 1		D					A					B					C					1 1 1 0 1				Rc
fnmsub*x*	1 1 1 1 1 1		D					A					B					C					1 1 1 1 0				Rc
fnmadd*x*	1 1 1 1 1 1		D					A					B					C					1 1 1 1 1				Rc
fcmpo	1 1 1 1 1 1		crfD			0	0	A					B					0 0 0 0 1 0 0 0 0 0									0
mtfsb1*x*	1 1 1 1 1 1		crbD					0 0 0 0 0					0 0 0 0 0					0 0 0 0 1 0 0 1 1 0									Rc
fneg*x*	1 1 1 1 1 1		D					0 0 0 0 0					B					0 0 0 0 1 0 1 0 0 0									Rc
mcrfs	1 1 1 1 1 1		crfD			0	0	crfS			0	0	0 0 0 0 0					0 0 0 1 0 0 0 0 0 0									0
mtfsb0*x*	1 1 1 1 1 1		crbD					0 0 0 0 0					0 0 0 0 0					0 0 0 1 0 0 0 1 1 0									Rc
fmr*x*	1 1 1 1 1 1		D					0 0 0 0 0					B					0 0 0 1 0 0 1 0 0 0									Rc
mtfsfi*x*	1 1 1 1 1 1		crfD			0	0	0 0 0 0 0					IMM				0	0 0 1 0 0 0 0 1 1 0									Rc
fnabs*x*	1 1 1 1 1 1		D					0 0 0 0 0					B					0 0 1 0 0 0 1 0 0 0									Rc

Name	0	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
fabs*x*	1 1 1 1 1 1		D					0 0 0 0 0					B					0 1 0 0 0 0 1 0 0 0									Rc
mffs*x*	1 1 1 1 1 1		D					0 0 0 0 0					0 0 0 0 0					1 0 0 1 0 0 0 1 1 1									Rc
mtfsf*x*	1 1 1 1 1 1		0	FM								0	B					1 0 1 1 0 0 0 1 1 1									Rc
fctid*x* [4]	1 1 1 1 1 1		D					0 0 0 0 0					B					1 1 0 0 1 0 1 1 1 0									Rc
fctidz*x* [4]	1 1 1 1 1 1		D					0 0 0 0 0					B					1 1 0 0 1 0 1 1 1 1									Rc
fcfid*x* [4]	1 1 1 1 1 1		D					0 0 0 0 0					B					1 1 0 1 0 0 1 1 1 0									Rc

[1] Supervisor-level instruction
[2] Supervisor- and user-level instruction
[3] Load and store string or multiple instruction
[4] 64-bit instruction
[5] Optional instruction
[6] 32-bit instruction only

This document contains information on a new product under development by Motorola and IBM. Motorola and IBM reserve the right to change or discontinue this product without notice. Information in this document is provided solely to enable system and software implementers to use PowerPC microprocessors. There are no express or implied copyright or patent licenses granted hereunder by Motorola or IBM to design, modify the design of, or fabricate circuits based on the information in this document.

This document was created with FrameMaker 4.0.4