SECTION 2
REGISTERS

This section describes the RCPU register organization as defined by the three levels of the PowerPC architecture: the user instruction set architecture (UISA), the virtual environment architecture (VEA), and the operating environment architecture (OEA), as well as the RCPU's implementation-specific registers.

2.1 Programming Models

The processor operates at one of two privilege levels: supervisor level (typically used by the operating environment) or user level (used by the application software). This division allows the operating system to control the application environment, protecting operating-system and critical machine resources. Instructions that control the state of the processor and supervisor registers can be executed only when the processor is operating at the supervisor level.

Supervisor-level access is provided through the processor's exception mechanism. That is, when an exception is taken, either due to an error or problem that needs to be serviced or deliberately through the use of a trap instruction, the processor begins operating in supervisor mode. The level of access is indicated by the privilege-level (PR) bit in the machine state register (MSR).

Figure 2-1 shows the user-level and supervisor-level RCPU programming models and also illustrates the three levels of the PowerPC architecture. The numbers to the left of the SPRs indicate the decimal number that is used in the syntax of the instruction operands to access the register.

NOTE

Registers such as the general-purpose registers (GPRs) and floating-point registers (FPRs) are accessed through operands that are part of the instructions. Access to registers can be explicit (that is, through the use of specific instructions for that purpose such as move to special-purpose register (**mtspr**) and move from special-purpose register (**mfspr**) instructions) or implicitly as the part of the execution of an instruction. Some registers are accessed both explicitly and implicitly.

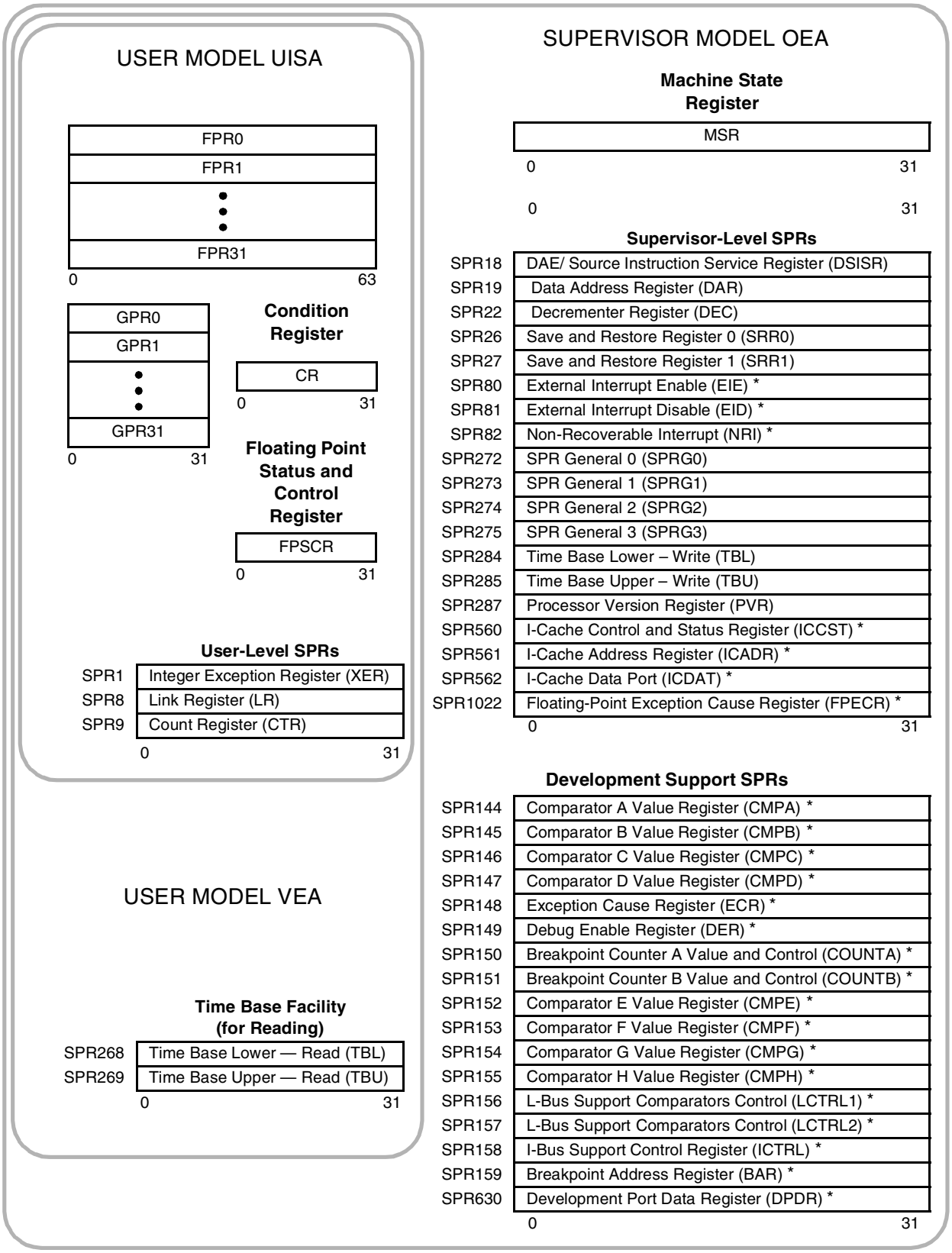


Figure 2-1 RCPU Programming Model

Where not otherwise noted, reserved fields in registers are ignored when written and return zero when read. An exception to this rule is XER[16:23]. These bits are set to the value written to them and return that value when read.

2.2 PowerPC UISA Register Set

The PowerPC UISA registers can be accessed by either user- or supervisor-level instructions. The general-purpose registers and floating-point registers are accessed through instruction operands. Access to registers can be explicit (that is, through the use of specific instructions for that purpose such as the **mtspr** and **mfspr** instructions) or implicit as part of the execution (or side effect) of an instruction. Some registers are accessed both explicitly and implicitly.

2.2.1 General Purpose Registers (GPRs)

Integer data is manipulated in the integer unit's thirty-two 32-bit GPRs, shown below. These registers are accessed as source and destination registers through operands in the instruction syntax.

GPRs — General Purpose Registers

0	31
GPR0	
GPR1	
...	
...	
GPR31	

RESET:UNCHANGED

2.2.2 Floating-Point Registers (FPRs)

The PowerPC architecture provides thirty-two 64-bit FPRs. These registers are accessed as source and destination registers through operands in 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 the compare instructions (which update the CR), place the result into an FPR. Information about the status of floating-point operations is placed into the floating-point status and control register (FPSCR) and in some cases, into the CR, after the completion of the operation's writeback stage. For information on how the CR is affected by floating-point operations, see **2.2.4 Condition Register (CR)**.

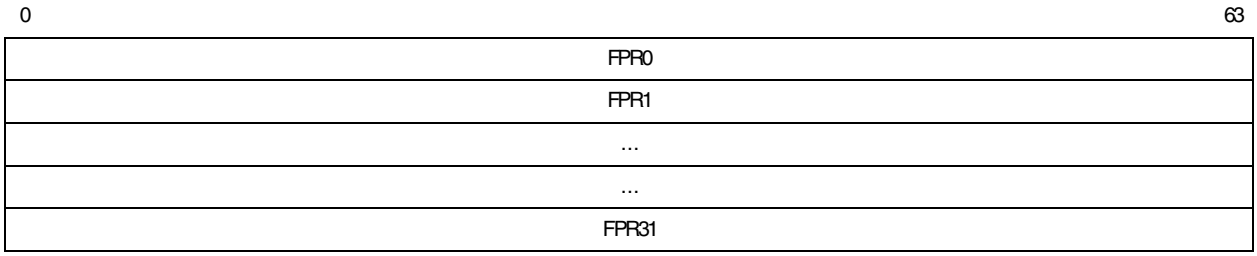
Load and store double instructions transfer 64 bits of data between memory and the FPRs in the floating-point processor with no conversion. Load single instruc-

tions transfer and convert floating-point values in single-precision floating-point format from memory to the same value in double-precision floating-point format in the FPRs. Store single instructions are provided to read a double-precision floating-point value from a floating-point register, convert it to single-precision floating-point format, and place it in the target memory location.

Single- and double-precision arithmetic instructions accept values from the FPRs in double-precision format. For single-precision arithmetic instructions, all input values must be representable in single-precision format; otherwise, the result placed into the target FPR and the setting of status bits in the FPSCR and in the condition register are undefined.

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 (either 32-bit or 64-bit) then it must be rounded. The final result is then placed into the FPR in the double-precision format.

FPRs — Floating-Point Registers



RESET:UNCHANGED

2.2.3 Floating-Point Status and Control Register (FPSCR)

The FPSCR controls the handling of floating-point exceptions and records status resulting from the floating-point operations. FPSCR[0:23] are status bits. FPSCR[24:31] are control bits.

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

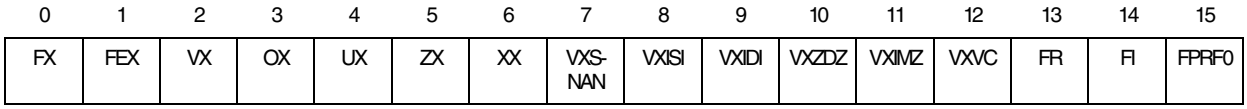
Table 2-1 summarizes which bits in the FPSCR are sticky bits, which are normal status bits, and which are control bits.

Table 2-1 FPSCR Bit Categories

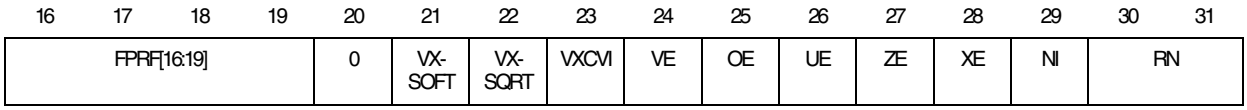
Bits	Type
[0], [3:12], [21:23]	Status, sticky
[1:2], [13:20]	Status, not sticky
[24:31]	Control

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 — Floating-Point Status and Control Register



RESET:UNCHANGED



RESET:UNCHANGED

A listing of FPSCR bit settings is shown in **Table 2-2**.

Table 2-2 FPSCR Bit Settings

Bit(s)	Name	Description
0	FX	Floating-point exception summary. Every floating-point instruction implicitly sets FPSCR[FX] if that instruction causes any of the floating-point exception bits in the FPSCR to change from zero to one. The **mcrfs** instruction implicitly clears FPSCR[FX] if the FPSCR field containing FPSCR[FX] is copied. The **mtfsf**, **mtfsfi**, **mtfsb0**, and **mtfsb1** instructions can set or clear 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 with their respective enable bits. The **mcrfs** instruction implicitly clears FPSCR[FEX] if the result of the logical OR described above becomes zero. The **mtfsf**, **mtfsfi**, **mtfsb0**, and **mtfsb1** instructions cannot set or clear 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** instruction implicitly clears FPSCR[VX] if the result of the logical OR described above becomes zero. The **mtfsf**, **mtfsfi**, **mtfsb0**, and **mtfsb1** instructions cannot set or clear FPSCR[VX] explicitly. This is not a sticky bit.

Table 2-2 FPSCR Bit Settings (Continued)

Bit(s)	Name	Description
3	OX	Floating-point overflow exception. This is a sticky bit. See **6.11.10.8 Overflow Exception Condition**.
4	UX	Floating-point underflow exception. This is a sticky bit. See **6.11.10.9 Underflow Exception Condition**.
5	ZX	Floating-point zero divide exception. This is a sticky bit. See **6.11.10.7 Zero Divide Exception Condition**.
6	XX	Floating-point inexact exception. This is a sticky bit. See **6.11.10.10 Inexact Exception Condition**.
7	VXSNAN	Floating-point invalid operation exception for SNaN. This is a sticky bit. See **6.11.10.6 Invalid Operation Exception Conditions**.
8	VXISI	Floating-point invalid operation exception for ∞-∞. This is a sticky bit. See **6.11.10.6 Invalid Operation Exception Conditions**.
9	VXIDI	Floating-point invalid operation exception for ∞/∞. This is a sticky bit. See **6.11.10.6 Invalid Operation Exception Conditions**.
10	VXZDZ	Floating-point invalid operation exception for 0/0. This is a sticky bit. See **6.11.10.6 Invalid Operation Exception Conditions**.
11	VXIMZ	Floating-point invalid operation exception for ×*0. This is a sticky bit. See **6.11.10.6 Invalid Operation Exception Conditions**.
12	VXVC	Floating-point invalid operation exception for invalid compare. This is a sticky bit. See **6.11.10.6 Invalid Operation Exception Conditions**.
13	FR	Floating-point fraction rounded. The last floating-point instruction that potentially rounded the intermediate result incremented the fraction. (See **3.3.11 Rounding**.) This bit is not sticky.
14	FI	Floating-point fraction inexact. The last floating-point instruction that potentially rounded the intermediate result produced an inexact fraction or a disabled exponent overflow. (See **3.3.11 Rounding**.) This bit is not sticky.
[15:19]	FPRF	Floating-point result flags. This field is based on the value placed into the target register even if that value is undefined. Refer to **Table 2-3** for specific bit settings. 15 Floating-point result class descriptor (C). Floating-point instructions other than the compare instructions may set this bit with the FPCC bits, to indicate the class of the result. [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. Other floating-point 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 ?)
20	—	Reserved

Table 2-2 FPSCR Bit Settings (Continued)

Bit(s)	Name	Description
21	VXSOFT	Floating-point invalid operation exception for software request. This bit can be altered only by the **mcrfs**, **mtfsfi**, **mtfsf**, **mtfsb0**, or **mtfsb1** instructions. The purpose of VXSOFT is to allow software to cause an invalid operation condition for a condition that is not necessarily associated with the execution of a floating-point instruction. For example, it might be set by a program that computes a square root if the source operand is negative. This is a sticky bit. See **6.11.10.6 Invalid Operation Exception Conditions**.
22	VXSQRT	Floating-point invalid operation exception for invalid square root. This is a sticky bit. This guarantees that software can simulate **fsqrt** and **frsqrte**, and to provide a consistent interface to handle exceptions caused by square-root operations. See **6.11.10.6 Invalid Operation Exception Conditions**.
23	VXCVI	Floating-point invalid operation exception for invalid integer convert. This is a sticky bit. See **6.11.10.6 Invalid Operation Exception Conditions**.
24	VE	Floating-point invalid operation exception enable. See **6.11.10.6 Invalid Operation Exception Conditions**.
25	OE	Floating-point overflow exception enable. See **6.11.10.8 Overflow Exception Condition**.
26	UE	Floating-point underflow exception enable. This bit should not be used to determine whether denormalization should be performed on floating-point stores. See **6.11.10.9 Underflow Exception Condition**.
27	ZE	Floating-point zero divide exception enable. See **6.11.10.7 Zero Divide Exception Condition**.
28	XE	Floating-point inexact exception enable. See **6.11.10.10 Inexact Exception Condition**.
29	NI	Non-IEEE mode bit. See **3.4.3 Non-IEEE Operation**.
[30:31]	RN	Floating-point rounding control. See **3.3.11 Rounding**. 00 = Round to nearest 01 = Round toward zero 10 = Round toward +infinity 11 = Round toward –infinity

Table 2-3 illustrates the floating-point result flags that correspond to FPSCR bits [15:19].

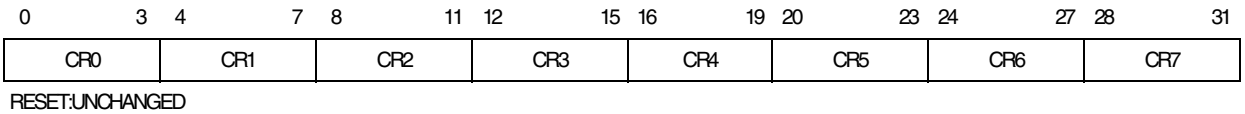
Table 2-3 Floating-Point Result Flags in FPSCR

Result Flags (Bits [15:19]) C<>=?	Result value class
10001	Quiet NaN
01001	– Infinity
01000	– Normalized number
11000	– Denormalized number
10010	– Zero
00010	+ Zero
10100	+ Denormalized number
00100	+Normalized number
00101	+Infinity

2.2.4 Condition Register (CR)

The condition register (CR) is a 32-bit register that reflects the result of certain operations and provides a mechanism for testing and branching. The bits in the CR are grouped into eight 4-bit fields, CR0 to CR7.

CR — Condition Register



The CR fields can be set in the following ways:

- Specified fields of the CR can be set by a move instruction (**mtcrf**) to the CR from a GPR.
- Specified fields of the CR can be moved from one CR*x* field to another with the **mcrf** instruction**.**
- A specified field of the CR can be set by a move instruction (**mcrfs**) to the CR from the FPSCR.
- A specified field of the CR can be set by a move instruction (**mcrxr**) to the CR from the XER.
- 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 operation.
- CR1 can be the implicit result of a floating-point operation.
- A specified CR field can be the explicit result of either an integer or floating-point compare instruction.

Instructions are provided to test individual CR bits.

2.2.4.1 Condition Register CR0 Field Definition

In most integer instructions, when the CR is set to reflect the result of the operation (that is, when Rc = 1), and for **addic.**, **andi.**, and **andis.**, the first three bits of CR0 are set by an algebraic comparison of the result to zero; the fourth bit of CR0 is copied from XER[SO]. For integer instructions, CR[0:3] are set to reflect the result as a signed quantity. The result as an unsigned quantity or a bit string can be deduced from the EQ bit.

The CR0 bits are interpreted as shown in **Table 2-4**. If any portion of the result (the 32-bit value placed into the destination register) is undefined, the value placed in the first three bits of CR0 is undefined.

Table 2-4 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.

2.2.4.2 Condition Register CR1 Field Definition

In all floating-point instructions when the CR is set to reflect the result of the operation (that is, when Rc = 1), the CR1 field (bits 4 to 7 of the CR) is copied from FPSCR[0:3] to indicate the floating-point exception status. For more information about the FPSCR, see **2.2.3 Floating-Point Status and Control Register (FPSCR)**. The bit settings for the CR1 field are shown in **Table 2-5**.

Table 2-5 Bit Settings for CR1 Field of CR

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

2.2.4.3 Condition Register CR*n* Field — Compare Instruction

When a specified CR field is set by a compare instruction, the bits of the specified field are interpreted as shown in **Table 2-6**. A condition register field can also be accessed by the **mfcr**, **mcrf**, and **mtcrf** instructions.

Table 2-6 CR*n* Field Bit Settings for Compare Instructions

CR*n* Bit[1]	Description
0	Less than, floating-point less than (LT, FL). For integer compare instructions, (**r**A) < SIMM, UIMM, or (**r**B) (algebraic comparison) or (**r**A) SIMM, UIMM, or (**r**B) (logical comparison). For floating-point compare instructions, (**fr**A) < (**fr**B).
1	Greater than, floating-point greater than (GT, FG). For integer compare instructions, (**r**A) > SIMM, UIMM, or (**r**B) (algebraic comparison) or (**r**A) SIMM, UIMM, or (**r**B) (logical comparison). For floating-point compare instructions, (**fr**A) > (**fr**B).
2	Equal, 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, 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 not a number (NaN).

NOTES:
1. Here, the bit indicates the bit number in any one of the four-bit subfields, CR[0:7]

2.2.5 Integer Exception Register (XER)

The integer exception register (XER) is a user-level, 32-bit register.

XER — Integer Exception Register **SPR 1**

0	1	2	3 … 24	25 … 31
SO	OV	CA	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	BYTES

RESET:UNCHANGED

The SPR number for the XER is one. The bit definitions for XER, shown in **Table 2-7**, are based on the operation of an instruction considered as a whole, not on intermediate results. For example, the result of the subtract from carrying (**subfc***x*) instruction is specified as the sum of three values. This instruction sets bits in the XER based on the entire operation, not on an intermediate sum.

In most cases, reserved fields in registers are ignored when written and return zero when read. However, XER[16:23] are set to the value written to them and return that value when read.

Table 2-7 Integer Exception Register Bit Definitions

Bit(s)	Name	Description
0	SO	Summary Overflow (SO) — The summary overflow bit is set whenever an instruction sets the overflow bit (OV) to indicate overflow and remains set until software clears it. It is not altered by compare instructions or other instructions that cannot overflow.
1	OV	Overflow (OV) — The overflow bit is set to indicate that an overflow has occurred during execution of an instruction. Integer and subtract instructions having OE = 1 set OV if the carry out of bit 0 is not equal to the carry out of bit 1, and clear it otherwise. The OV bit is not altered by compare instructions or other instructions that cannot overflow.
2	CA	Carry (CA) — In general, the carry bit is set to indicate that a carry out of bit 0 occurred during execution of an instruction. Add carrying, subtract from carrying, add extended, and subtract from extended instructions set CA to one if there is a carry out of bit 0, and clear it otherwise. The CA bit is not altered by compare instructions or other instructions that cannot carry, except that shift right algebraic instructions set the CA bit to indicate whether any '1' bits have been shifted out of a negative quantity.
[3:24]	—	Reserved
[25:31]	BYTES	This field specifies the number of bytes to be transferred by a load string word indexed (**lswx**) or store string word indexed (**stswx**) instruction.

2.2.6 Link Register (LR)

The 32-bit link register 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.

LR — Link Register **SPR 8**

NOTE

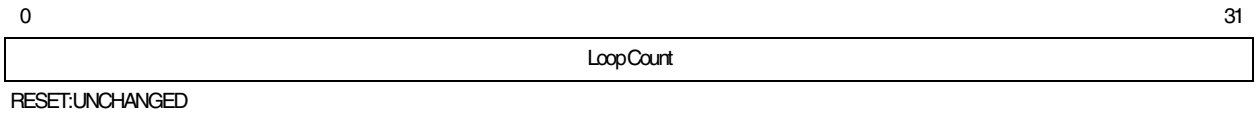
Although the two least-significant bits can accept any values written to them, they are ignored when the LR is used as an address. The link register can be accessed by the **mtspr** and **mfspr** instructions using the SPR number eight. Prefetching instructions along the target path (loaded by an **mtspr** instruction) is possible provided the link register is loaded sufficiently ahead of the branch instruction. It is usually possible to prefetch 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. This is done regardless of whether the branch is taken.

2.2.7 Count Register (CTR)

The count register (CTR) is a 32-bit register for holding 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 zero before being decremented, it is negative one afterward. The count register provides the branch target address for the branch conditional to count register (**bcctr***x*) instruction.

CTR — Count Register **SPR 9**



Prefetching 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 SPR 9. 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 condition register and the count register. The encoding for the BO field is shown in **Table 4-21** in **SECTION 4 ADDRESSING MODES AND INSTRUCTION SET SUMMARY**.

2.3 PowerPC VEA Register Set — Time Base

The PowerPC virtual environment architecture (VEA) defines registers in addition to those in the UISA register set. The PowerPC VEA register set can be accessed by all software with either user- or supervisor-level privileges.

The PowerPC VEA includes the time base facility (TB), a 64-bit structure that contains a 64-bit unsigned integer that is incremented periodically. The frequency at which the counter is updated is implementation-dependent and need not be constant over long periods of time.

The TB consists of two 32-bit registers: time base upper (TBU) and time base lower (TBL). In the context of the VEA, user-level applications are permitted read-only access to the TB. The OEA defines supervisor-level access to the TB for writing values to the TB. Different SPR encodings are provided for reading and writing the time base.

Refer to **2.4 PowerPC OEA Register Set** for more information on writing to the TB. Refer to **4.7.2 Move to/from Special Purpose Register Instructions** for simplified mnemonics for reading and writing to the time base. For information on the time base clock source, refer to the ***System Interface Unit Reference Manual (SIURM/AD)***.

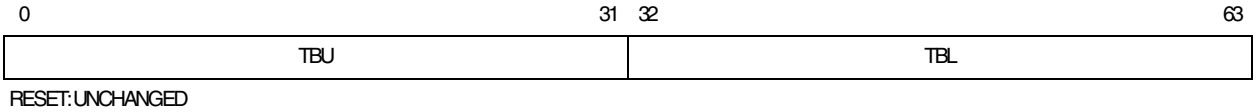


Table 2-8 Time Base Field Definitions

Bits	Name	Description
[0:31]	TBU	Time Base (Upper) — The high-order 32 bits of the time base
[32:63]	TBL	Time Base (Lower) — The low-order 32 bits of the time base

In 32-bit PowerPC implementations such as the RCPU, it is not possible to read the entire 64-bit time base in a single instruction. The **mftb** simplified mnemonic copies the lower half of the time base register (TBL) to a GPR, and the **mftbu** simplified mnemonic copies the upper half of the time base (TBU) to a GPR.

Because of the possibility of a carry from TBL to TBU occurring between reads of the TBL and TBU, a sequence such as the following example is necessary to read the time base on RCPU-based systems.

```
loop:
            mftbu       rx          #load from TBU
            mftb        ry          #load from TBL
            mftbu       rz          #load from TBU
            cmpw        rz,rx       #see if 'old'='new'
            bne         loop        #loop if carry occurred
```

The comparison and loop are necessary to ensure that a consistent pair of values has been obtained.

2.4 PowerPC OEA Register Set

The PowerPC operating environment architecture (OEA) includes a number of SPRs and other registers that are accessible only by supervisor-level instructions. Some SPRs are RCPU-specific; some RCPU SPRs may not be implemented in other PowerPC processors, or may not be implemented in the same way.

2.4.1 Machine State Register (MSR)

The machine state register is a 32-bit register that defines the state of the processor. When an exception occurs, the current contents of the MSR are loaded into SRR1, and the MSR is updated to reflect the new machine state. The MSR can also be modified by the **mtmsr**, **sc**, and **rfi** instructions. It can be read by the **mfmsr** instruction.

MSR — Machine State Register

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
RESERVED															ILE
RESET:															
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
EE	PR	FP	ME	FE0	SE	BE	FE1	0	IP	RESERVED				RI	LE
RESET:															
0	0	0	U	0	0	0	0	0	*	0	0	0	0	0	0

* Reset value of this bit depends on the value of the internal reset configuration word. Refer to the *System Interface Unit Reference Manual* (SIURM/AD) for more information.

Table 2-13 shows the bit definitions for the MSR.

Table 2-9 Machine State Register Bit Settings

Bit(s)	Name	Description
[0:14]	—	Reserved
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. 0 = Processor runs in big-endian mode during exception processing. 1 = Processor runs in little-endian mode during exception processing.
16	EE	External interrupt enable 0 = 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.
17	PR	Privilege level 0 = The processor can execute both user- and supervisor-level instructions. 1 = The processor can only execute user-level instructions.
18	FP	Floating-point available 0 = The processor prevents dispatch of floating-point instructions, including floating-point loads, stores and moves. Floating-point enabled program exceptions can still occur and the FPRs can still be accessed. 1 = The processor can execute floating-point instructions, and can take floating-point enabled exception type program exceptions.
19	ME	Machine check enable 0 = Machine check exceptions are disabled. 1 = Machine check exceptions are enabled.
20	FE0	Floating-point exception mode zero (See **Table 2-10**.)
21	SE	Single-step trace enable 0 = The processor executes instructions normally. 1 = The processor generates a single-step trace exception upon the successful execution of the next instruction. When this bit is set, the processor dispatches instructions in strict program order. Successful execution means the instruction caused no other exception. Single-step tracing may not be present on all implementations.
22	BE	Branch trace enable 0 = No trace exception occurs when a branch instruction is completed 1 = Trace exception occurs when a branch instruction is completed
23	FE1	Floating-point exception mode 1 (See**Table 2-10**.)
24	—	Reserved.
25	IP	Exception prefix. The setting of this bit specifies the location of the exception vector table. 0 = Exception vector table starts at the physical address 0x0000 0000. 1 = Exception vector table starts at the physical address 0xFFF0 0000.
[26:29]	—	Reserved
30	RI	Recoverable exception (for machine check and non-maskable breakpoint exceptions) 0 = Machine state is not recoverable. 1 = Machine state is recoverable. Refer to **SECTION 6 EXCEPTIONS** for more information.
31	LE	Little-endian mode 0 = Processor operates in big-endian mode during normal processing. 1 = Processor operates in little-endian mode during normal processing.

The floating-point exception mode bits are interpreted as shown in **Table 2-10**. For further details see **6.11.10.5 Floating-Point Enabled Exceptions**.

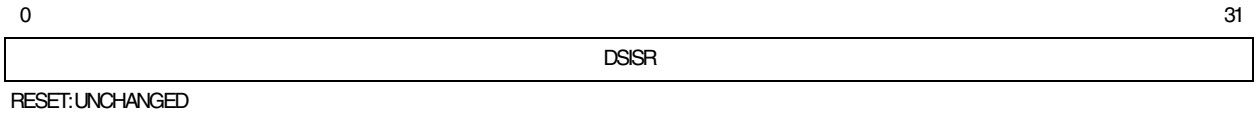
Table 2-10 Floating-Point Exception Mode Bits

FE[0:1]	Mode
00	Ignore exceptions mode — Floating-point exceptions do not cause the floating-point assist error handler to be invoked.
01, 10, 11	Floating-point precise mode — The system floating-point assist error handler is invoked precisely at the instruction that caused the enabled exception.

2.4.2 DAE/Source Instruction Service Register (DSISR)

The 32-bit DSISR identifies the cause of data access and alignment exceptions.

DSISR — DAE/Source Instruction Service Register **SPR 18**



For information about bit settings, see **6.11.4 Alignment Exception (0x00600)**.

2.4.3 Data Address Register (DAR)

After an alignment exception, the DAR is set to the effective address of a load or store element. For information, see **6.11.4 Alignment Exception (0x00600)**.

DAR — Data Address Register **SPR 19**

2.4.4 Time Base Facility (TB) — OEA

As described in **2.3 PowerPC VEA Register Set — Time Base**, the time base (TB) provides a 64-bit incrementing counter. The VEA defines user-level, read-only access to the TB. Writing to the TB is reserved for supervisor-level applications such as operating systems and bootstrap routines. The OEA defines supervisor-level write access to the TB.

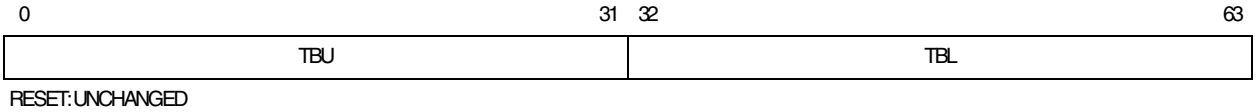


Table 2-11 Time Base Field Definitions

Bits	Name	Description
[0:31]	TBU	Time Base (Upper) — The high-order 32 bits of the time base
[32:63]	TBL	Time Base (Lower) — The low-order 32 bits of the time base

The TB can be written at the supervisor privilege level only. The **mttbl** and **mttbu** simplified mnemonics write the lower and upper halves of the TB, respectively. The **mtspr**, **mttbl**, and **mttbu** instructions treat TBL and TBU as separate 32-bit registers; setting one leaves the other unchanged. It is not possible to write the entire 64-bit time base in a single instruction.

The TB can be written by a sequence such as the following:

```
lwz         rx,upper        # load 64-bit value for
lwz         ry,lower        # TB into rx and ry
li          rz,0
mttbl       rz              # force TBL to 0
mttbu       rx              # set TBU
mttbl       ry              # set TBL
```

Loading zero into TBL prevents the possibility of a carry from TBL to TBU while the time base is being initialized.

For information about reading the time base, refer to **2.3 PowerPC VEA Register Set — Time Base**.

2.4.5 Decrementer Register (DEC)

The DEC 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 a subdivision of the processor clock. Refer to the ***System Interface Unit Reference Manual (SIURM/AD)*** for information on the clock source for the decrementer.

DEC — Decrementer Register **SPR 22**

RESET: UNCHANGED

The DEC counts down, causing an exception (unless masked) when it passes through zero. The DEC satisfies the following requirements:

- Loading a GPR from the DEC has no effect on the DEC.
- Storing a GPR to the DEC replaces the value in the DEC with the value in the GPR.
- Whenever bit 0 of the DEC changes from zero to one, a decrementer exception request is signaled. Multiple DEC exception requests may be received before the first exception occurs; however, any additional requests are canceled when the exception occurs for the first request. Refer to **6.11.7 Decrementer Exception (0x00900)** for additional information.
- If the DEC is altered by software and the content of bit 0 is changed from zero to one, an exception request is signaled.

The content of the DEC can be read or written using the **mfspr** and **mtspr** instructions. Using a simplified mnemonic for the **mtspr** instruction, the DEC can be written from GPR **r**A with the following:

mtdec rA

If the execution of this instruction causes bit 0 of the DEC to change from zero to one, an exception request is signaled. The DEC can be read into GPR **r**A with the following instruction:

mfdec rA

Copying the DEC to a GPR does not affect the DEC content or the exception mechanism.

2.4.6 Machine Status Save/Restore Register 0 (SRR0)

The machine status save/restore register 0 (SRR0) is a 32-bit register that identifies where instruction execution should resume when an **rfi** instruction is executed following an exception. It also holds the effective address of the instruction that follows the system call (**sc**) instruction.

SRR0— Machine Status Save/Restore Register 0 **SPR 26**

RESET: UNDEFINED

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. 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 immediately following instruction. The instruction addressed can be determined from the exception type and status bits.

For information on how specific exceptions affect SRR0, refer to the descriptions of individual exceptions in **SECTION 6 EXCEPTIONS**.

2.4.7 Machine Status Save/Restore Register 1 (SRR1)

SRR1 is a 32-bit register used to save machine status on exceptions and to restore machine status when an **rfi** instruction is executed.

SRR1 — Machine Status Save/Restore Register 1 **SPR 27**

0	31
SRR1	

RESET: UNDEFINED

In general, when an exception occurs, SRR1[0:15] are loaded with exception-specific information and of MSR[16:31] are placed into SRR1[16:31].

For information on how specific exceptions affect SRR1, refer to the individual exceptions in **SECTION 6 EXCEPTIONS**.

2.4.8 General SPRs (SPRG0–SPRG3)

SPRG0 through SPRG3 are 32-bit registers provided for general operating system use, such as performing a fast state save and for supporting multiprocessor implementations. SPRG0–SPRG3 are shown below.

SPRG0–SPRG3 — General Special-Purpose Registers 0–3 **SPR 272 – SPR 275**

0	31
SPRG0	
SPRG1	
SPRG2	
SPRG3	

RESET: UNCHANGED

Uses for SPRG0–SPRG3 are shown in **Table 2-12**.

Table 2-12 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 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 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 GPRs 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.

2.4.9 Processor Version Register (PVR)

The PVR is a 32-bit, read-only register that identifies the version and revision level of the PowerPC processor. The contents of the PVR can be copied to a GPR by the **mfspr** instruction. Read access to the PVR is available in supervisor mode only; write access is not provided.

PVR — Processor Version Register **SPR 287**

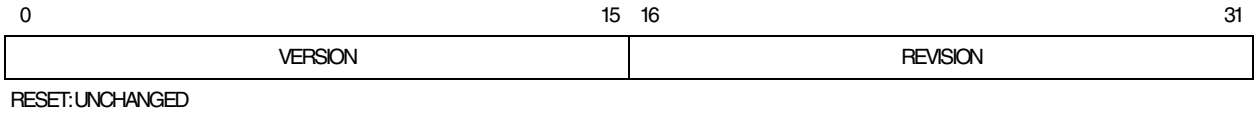


Table 2-13 Processor Version Register Bit Settings

Bit(s)	Name	Description
[0:15]	VERSION	A 16-bit number that identifies the version of the processor and of the PowerPC architecture
[16:31]	REVISION	A 16-bit number that distinguishes between various releases of a particular version

2.4.10 Implementation-Specific SPRs

The RCPU includes several implementation-specific SPRs that are not defined by the PowerPC architecture. These registers can be accessed by supervisor-level instructions only.

2.4.10.1 EIE, EID, and NRI Special-Purpose Registers

The RCPU includes three implementation-specific SPRs to facilitate the software manipulation of the MSR[RI] and MSR[EE] bits. Issuing the **mtspr** instruction with one of these registers as an operand causes the RI and EE bits to be set or cleared as shown in **Table 2-14**.

A read (**mfspr**) of any of these locations is treated as an unimplemented instruction, resulting in a software emulation exception.

Table 2-14 EIE, EID, AND NRI Registers

SPR Number (Decimal)	Mnemonic	MSR[EE]	MSR[RI]
80	EIE	1	1
81	EID	0	1
82	NRI	0	0

Refer to **SECTION 6 EXCEPTIONS** for more information on these registers.

2.4.10.2 Instruction-Cache Control Registers

The implementation-specific supervisor-level SPRs shown in **Table 2-15** control the operation of the instruction cache.

Table 2-15 Instruction Cache Control Registers

SPR Number (Decimal)	Name	Description
560	ICCST	I-cache control and status register
561	ICADR	I-cache address register
562	ICDAT	I-cache data port (read only)

Refer to **SECTION 5 INSTRUCTION CACHE** for details on these registers.

2.4.10.3 Development Support Registers

Table 2-16 lists the implementation-specific RCPU registers provided for development support.

Table 2-16 Development Support Registers

SPR Number (Decimal)	Mnemonic	Name
144	CMPA	Comparator A Value Register
145	CMPB	Comparator B Value Register
146	CMPC	Comparator C Value Register
147	CMPD	Comparator D Value Register
148	ECR	Exception Cause Register
149	DER	Debug Enable Register
150	COUNTA	Breakpoint Counter A Value and Control Register
151	COUNTB	Breakpoint Counter B Value and Control Register
152	CMPE	Comparator E Value Register
153	CMPF	Comparator F Value Register
154	CMPG	Comparator G Value Register
155	CMPH	Comparator H Value Register
156	LCTRL1	L-Bus Support Control Register 1
157	LCTRL2	L-Bus Support Control Register 2
158	ICTRL	I-Bus Support Control Register
159	BAR	Breakpoint Address Register
630	DPDR	Development Port Data Register

Refer to **SECTION 8 DEVELOPMENT SUPPORT** for details about these registers.

2.4.10.4 Floating-Point Exception Cause Register (FPECR)

The FPECR is a 32-bit supervisor-level internal status and control register used by the floating-point assist software envelope. Refer to **6.11.10 Floating-Point Assist Exception (0x00E00)** for more information on this register.