- An attempt to access memory with an effective address alignment that is invalid for the instruction causes the alignment exception handler to be invoked.
- The execution of an **sc** instruction permits a program to call on the system to perform a service, by causing a system call exception handler to be invoked.
- The execution of a trap instruction invokes the program exception trap handler.
- The execution of a floating-point instruction when floating-point instructions are disabled invokes the floating-point unavailable exception handler.
- The execution of an instruction that causes a floating-point exception that is enabled invokes the floating-point enabled exception handler.
- The execution of a floating-point instruction that requires system software assistance causes the floating-point assist exception handler to be invoked. The conditions under which such software assistance is required are implementation-dependent.

Exceptions caused by asynchronous events are described in Chapter 6, “Exceptions.”

4.2 PowerPC UISA Instructions

The PowerPC user instruction set architecture (UISA) includes the base user-level instruction set (excluding a few user-level cache-control, synchronization, and time base instructions), user-level registers, programming model, data types, and addressing modes. This section discusses the instructions defined in the UISA.

4.2.1 Integer Instructions

The integer instructions consist of the following:

- Integer arithmetic instructions
- Integer compare instructions
- Integer logical instructions
- Integer rotate and shift instructions

Integer instructions use the content of the GPRs as source operands and place results into GPRs. Integer arithmetic, shift, rotate, and string move instructions may update or read values from the XER, and the condition register (CR) fields may be updated if the Rc bit of the instruction is set.

These instructions treat the source operands as signed integers unless the instruction is explicitly identified as performing an unsigned operation. For example, Multiply High-Word Unsigned (**mulhwu**) and Divide Word Unsigned (**divwu**) instructions interpret both operands as unsigned integers.

The integer instructions that are coded to update the condition register, and the integer arithmetic instruction, **addic.**, set CR bits 0–3 (CR0) to characterize the result of the operation. CR0 is set to reflect a signed comparison of the result to zero.

The integer arithmetic instructions, **addic**, **addic.**, **subfic**, **addc**, **subfc**, **adde**, **subfe**, **addme**, **subfme**, **addze**, and **subfze**, always set the XER bit, CA, to reflect the carry out of bit 0. Integer arithmetic instructions with the overflow enable (OE) bit set in the instruction encoding (instructions with o suffix) cause the XER[SO] and XER[OV] to reflect an overflow of the result. Except for the multiply low and divide instructions, these integer arithmetic instructions reflect the overflow of the result.

Instructions that select the overflow option (enable XER[OV]) or that set the XER carry bit (CA) may delay the execution of subsequent instructions.

Unless otherwise noted, when CR0 and the XER are set, they reflect the value placed in the target register.

4.2.1.1 Integer Arithmetic Instructions

Table 4-1 lists the integer arithmetic instructions for the PowerPC processors.

Table 4-1. Integer Arithmetic Instructions

Name	Mnemonic	Operand Syntax	Operation
Add Immediate	**addi**	**r**D,**r**A,SIMM	The sum (**r**A|0) + SIMM is placed into **r**D.
Add Immediate Shifted	**addis**	**r**D,**r**A,SIMM	The sum (**r**A|0) + (SIMM || 0x0000) is placed into **r**D.
Add	**add** **add.** **addo** **addo.**	**r**D,**r**A,**r**B	The sum (**r**A) + (**r**B) is placed into **r**D. **add** Add **add.** Add with CR Update. The dot suffix enables the update of the CR. **addo** Add with Overflow Enabled. The o suffix enables the overflow bit (OV) in the XER. **addo.** Add with Overflow and CR Update. The o. suffix enables the update of the CR and enables the overflow bit (OV) in the XER.
Subtract From	**subf** **subf.** **subfo** **subfo.**	**r**D,**r**A,**r**B	The sum ¬ (**r**A) + (**r**B) +1 is placed into **r**D. **subf** Subtract From **subf.** Subtract from with CR Update. The dot suffix enables the update of the CR. **subfo** Subtract from with Overflow Enabled. The o suffix enables the overflow bit (OV) in the XER. **subfo.** Subtract from with Overflow and CR Update. The o. suffix enables the update of the CR and enables the overflow bit (OV) in the XER.
Add Immediate Carrying	**addic**	**r**D,**r**A,SIMM	The sum (**r**A) + SIMM is placed into **r**D.
Add Immediate Carrying and Record	**addic.**	**r**D,**r**A,SIMM	The sum (**r**A) + SIMM is placed into **r**D. The CR is updated.
Subtract from Immediate Carrying	**subfic**	**r**D,**r**A,SIMM	The sum ¬ (**r**A) + SIMM + 1 is placed into **r**D.

Table 4-1. Integer Arithmetic Instructions (Continued)

Name	Mnemonic	Operand Syntax	Operation
Add Carrying	**addc** **addc.** **addco** **addco.**	**r**D,**r**A,**r**B	The sum (**r**A) + (**r**B) is placed into **r**D. **addc** Add Carrying **addc.** Add Carrying with CR Update. The dot suffix enables the update of the CR. **addco** Add Carrying with Overflow Enabled. The o suffix enables the overflow bit (OV) in the XER. **addco.** Add Carrying with Overflow and CR Update. The o. suffix enables the update of the CR and enables the overflow bit (OV) in the XER.
Subtract from Carrying	**subfc** **subfc.** **subfco** **subfco.**	**r**D,**r**A,**r**B	The sum ¬ (**r**A) + (**r**B) + 1 is placed into **r**D. **subfc** Subtract from Carrying **subfc.** Subtract from Carrying with CR Update. The dot suffix enables the update of the CR. **subfco** Subtract from Carrying with Overflow. The o suffix enables the overflow bit (OV) in the XER. **subfco.** Subtract from Carrying with Overflow and CR Update. The o. suffix enables the update of the CR and enables the overflow bit (OV) in the XER.
Add Extended	**adde** **adde.** **addeo** **addeo.**	**r**D,**r**A,**r**B	The sum (**r**A) + (**r**B) + XER[CA] is placed into **r**D. **adde** Add Extended **adde.** Add Extended with CR Update. The dot suffix enables the update of the CR. **addeo** Add Extended with Overflow. The o suffix enables the overflow bit (OV) in the XER. **addeo.** Add Extended with Overflow and CR Update. The o. suffix enables the update of the CR and enables the overflow bit (OV) in the XER.
Subtract from Extended	**subfe** **subfe.** **subfeo** **subfeo.**	**r**D,**r**A,**r**B	The sum ¬ (**r**A) + (**r**B) + XER[CA] is placed into **r**D. **subfe** Subtract from Extended **subfe.** Subtract from Extended with CR Update. The dot suffix enables the update of the CR. **subfeo** Subtract from Extended with Overflow. The o suffix enables the overflow bit (OV) in the XER. **subfeo.** Subtract from Extended with Overflow and CR Update. The o. suffix enables the update of the CR and enables the overflow (OV) bit in the XER.
Add to Minus One Extended	**addme** **addme.** **addmeo** **addmeo.**	**r**D,**r**A	The sum (**r**A) + XER[CA] added to 0xFFFF_FFFF is placed into **r**D. **addme** Add to Minus One Extended **addme.** Add to Minus One Extended with CR Update. The dot suffix enables the update of the CR. **addmeo** Add to Minus One Extended with Overflow. The o suffix enables the overflow bit (OV) in the XER. **addmeo.** Add to Minus One Extended with Overflow and CR Update. The o. suffix enables the update of the CR and enables the overflow (OV) bit in the XER.

Table 4-1. Integer Arithmetic Instructions (Continued)

Name	Mnemonic	Operand Syntax	Operation
Subtract from Minus One Extended	**subfme** **subfme.** **subfmeo** **subfmeo.**	**r**D,**r**A	The sum ¬ (**r**A) + XER[CA] added to 0xFFFF_FFFF is placed into **r**D. **subfme** Subtract from Minus One Extended **subfme.** Subtract from Minus One Extended with CR Update. The dot suffix enables the update of the CR. **subfmeo** Subtract from Minus One Extended with Overflow. The o suffix enables the overflow bit (OV) in the XER. **subfmeo.** Subtract from Minus One Extended with Overflow and CR Update. The o. suffix enables the update of the CR and enables the overflow bit (OV) in the XER.
Add to Zero Extended	**addze** **addze.** **addzeo** **addzeo.**	**r**D,**r**A	The sum (**r**A) + XER[CA] is placed into **r**D. **addze** Add to Zero Extended **addze.** Add to Zero Extended with CR Update. The dot suffix enables the update of the CR. **addzeo** Add to Zero Extended with Overflow. The o suffix enables the overflow bit (OV) in the XER. **addzeo.** Add to Zero Extended with Overflow and CR Update. The o. suffix enables the update of the CR and enables the overflow bit (OV) in the XER.
Subtract from Zero Extended	**subfze** **subfze.** **subfzeo** **subfzeo.**	**r**D,**r**A	The sum ¬ (**r**A) + XER[CA] is placed into **r**D. **subfze** Subtract from Zero Extended **subfze.** Subtract from Zero Extended with CR Update. The dot suffix enables the update of the CR. **subfzeo** Subtract from Zero Extended with Overflow. The o suffix enables the overflow bit (OV) in the XER. **subfzeo.** Subtract from Zero Extended with Overflow and CR Update. The o. suffix enables the update of the CR and enables the overflow bit (OV) in the XER.
Negate	**neg** **neg.** **nego** **nego.**	**r**D,**r**A	The sum ¬ (**r**A) + 1 is placed into **r**D. **neg** Negate **neg.** Negate with CR Update. The dot suffix enables the update of the CR. **nego** Negate with Overflow. The o suffix enables the overflow bit (OV) in the XER. **nego.** Negate with Overflow and CR Update. The o. suffix enables the update of the CR and enables the overflow bit (OV) in the XER.
Multiply Low Immediate	**mulli**	**r**D,**r**A,SIMM	The low-order 32 bits of the product (**r**A) * SIMM are placed into **r**D. This instruction can be used with **mulhd***x* or **mulhw***x* to calculate a full 64-bit product.

Table 4-1. Integer Arithmetic Instructions (Continued)

Name	Mnemonic	Operand Syntax	Operation
Multiply Low	**mullw** **mullw.** **mullwo** **mullwo.**	**r**D,**r**A,**r**B	The 32-bit product (**r**A) * (**r**B) is placed into register **r**D. This instruction can be used with **mulhw***x* to calculate a full 64-bit product. **mullw** Multiply Low **mullw.** Multiply Low with CR Update. The dot suffix enables the update of the CR. **mullwo** Multiply Low with Overflow. The o suffix enables the overflow bit (OV) in the XER. **mullwo.** Multiply Low with Overflow and CR Update. The o. suffix enables the update of the condition register and enables the overflow bit (OV) in the XER.
Multiply High Word	**mulhw** **mulhw.**	**r**D,**r**A,**r**B	The contents of **r**A and **r**B are interpreted as 32-bit signed integers. The 64-bit product is formed. The high-order 32 bits of the 64-bit product are placed into **r**D. **mulhw** Multiply High Word **mulhw.** Multiply High Word with CR Update. The dot suffix enables the update of the CR.
Multiply High Word Unsigned	**mulhwu** **mulhwu.**	**r**D,**r**A,**r**B	The contents of **r**A and of **r**B are interpreted as 32-bit unsigned integers. The 64-bit product is formed. The high-order 32 bits of the 64-bit product are placed into **r**D. **mulhwu** Multiply High Word Unsigned **mulhwu.** Multiply High Word Unsigned with CR Update. The dot suffix enables the update of the CR.
Divide Word	**divw** **divw.** **divwo** **divwo.**	**r**D,**r**A,**r**B	The dividend is the signed value of **r**A. The divisor is the signed value of **r**B. The quotient is placed into **r**D. The remainder is not supplied as a result. **divw** Divide Word **divw.** Divide Word with CR Update. The dot suffix enables the update of the CR. **divwo** Divide Word with Overflow. The o suffix enables the overflow bit (OV) in the XER. **divwo.** Divide Word with Overflow and CR Update. The o. suffix enables the update of the CR and enables the overflow bit (OV) in the XER.
Divide Word Unsigned	**divwu** **divwu.** **divwuo** **divwuo.**	**r**D,**r**A,**r**B	The dividend is the zero-extended value in **r**A. The divisor is the zero-extended value in **r**B. The quotient is placed into **r**D. The remainder is not supplied as a result. **divwu** Divide Word Unsigned **divwu.** Divide Word Unsigned with CR Update. The dot suffix enables the update of the CR. **divwuo** Divide Word Unsigned with Overflow. The o suffix enables the overflow bit (OV) in the XER. **divwuo.** Divide Word Unsigned with Overflow and CR Update. The o. suffix enables the update of the CR and enables the overflow bit (OV) in the XER.

Although there is no “Subtract Immediate” instruction, its effect can be achieved by using an **addi** instruction with the immediate operand negated. Simplified mnemonics are provided that include this negation. The **subf** instructions subtract the second operand (**r**A) from the third operand (**r**B). Simplified mnemonics are provided in which the third operand is subtracted from the second operand. See Appendix F, “Simplified Mnemonics,” for examples.

4.2.1.2 Integer Compare Instructions

The integer compare instructions algebraically or logically compare the contents of register **r**A with either the zero-extended value of the UIMM operand, the sign-extended value of the SIMM operand, or the contents of register **r**B. The comparison is signed for the **cmpi** and **cmp** instructions, and unsigned for the **cmpli** and **cmpl** instructions. Table 4-2 summarizes the integer compare instructions.

Appendix F, “Simplified MnemonicsFor 32-bit implementations, the L field must be cleared, otherwise the instruction form is invalid.

The integer compare instructions (shown in Table 4-2) set one of the leftmost three bits of the designated CR field, and clear the other two. XER[SO] is copied into bit 3 of the CR field.

Table 4-2. Integer Compare Instructions

Name	Mnemonic	Operand Syntax	Operation
Compare Immediate	**cmpi**	**crf**D,L,**r**A,SIMM	The value in register **r**A is compared with the sign-extended value of the SIMM operand, treating the operands as signed integers. The result of the comparison is placed into the CR field specified by operand **crf**D.
Compare	**cmp**	**crf**D,L,**r**A,**r**B	The value in register **r**A is compared with the value in register **r**B, treating the operands as signed integers. The result of the comparison is placed into the CR field specified by operand **crf**D.
Compare Logical Immediate	**cmpli**	**crf**D,L,**r**A,UIMM	The value in register **r**A is compared with 0x0000 || UIMM, treating the operands as unsigned integers. The result of the comparison is placed into the CR field specified by operand **crf**D.
Compare Logical	**cmpl**	**crf**D,L,**r**A,**r**B	The value in register **r**A is compared with the value in register **r**B, treating the operands as unsigned integers. The result of the comparison is placed into the CR field specified by operand **crf**D.

The **crf**D operand can be omitted if the result of the comparison is to be placed in CR0. Otherwise the target CR field must be specified in the instruction **crf**D field, using an explicit field number.

For information on simplified mnemonics for the integer compare instructions see Appendix F, “Simplified Mnemonics.”

4.2.1.3 Integer Logical Instructions

The logical instructions shown in Table 4-3 perform bit-parallel operations on 32-bit operands. Logical instructions with the CR updating enabled (uses dot suffix) and instructions **andi.** and **andis.** set CR field CR0 (bits 0 to 2) to characterize the result of the logical operation. Logical instructions without CR update and the remaining logical instructions do not modify the CR. Logical instructions do not affect the XER[SO], XER[OV], and XER[CA] bits.

See Appendix F, “Simplified Mnemonics,” for simplified mnemonic examples for integer logical operations.

Table 4-3. Integer Logical Instructions

Name	Mnemonic	Operand Syntax	Operation
AND Immediate	**andi.**	**r**A,**r**S,UIMM	The contents of **r**S are ANDed with 0x0000 || UIMM and the result is placed into **r**A. The CR is updated.
AND Immediate Shifted	**andis.**	**r**A,**r**S,UIMM	The content of **r**S are ANDed with UIMM || 0x0000 and the result is placed into **r**A. The CR is updated.
OR Immediate	**ori**	**r**A,**r**S,UIMM	The contents of **r**S are ORed with 0x0000 || UIMM and the result is placed into **r**A. The preferred no-op is **ori 0,0,0**
OR Immediate Shifted	**oris**	**r**A,**r**S,UIMM	The contents of **r**S are ORed with UIMM || 0x0000 and the result is placed into **r**A.
XOR Immediate	**xori**	**r**A,**r**S,UIMM	The contents of **r**S are XORed with 0x0000 || UIMM and the result is placed into **r**A.
XOR Immediate Shifted	**xoris**	**r**A,**r**S,UIMM	The contents of **r**S are XORed with UIMM || 0x0000 and the result is placed into **r**A.
AND	**and** **and.**	**r**A,**r**S,**r**B	The contents of **r**S are ANDed with the contents of register **r**B and the result is placed into **r**A. **and** AND **and.** AND with CR Update**.** The dot suffix enables the update of the CR.
OR	**or** **or.**	**r**A,**r**S,**r**B	The contents of **r**S are ORed with the contents of **r**B and the result is placed into **r**A. **or** OR **or.** OR with CR Update**.** The dot suffix enables the update of the CR.
XOR	**xor** **xor.**	**r**A,**r**S,**r**B	The contents of **r**S are XORed with the contents of **r**B and the result is placed into **r**A. **xor** XOR **xor.** XOR with CR Update**.** The dot suffix enables the update of the CR.

Table 4-3. Integer Logical Instructions (Continued)

Name	Mnemonic	Operand Syntax	Operation
NAND	**nand** **nand.**	**r**A,**r**S,**r**B	The contents of **r**S are ANDed with the contents of **r**B and the one's complement of the result is placed into **r**A. **nand** NAND **nand.** NAND with CR Update**.** The dot suffix enables the update of CR. Note that **nand***x*, with **r**S = **r**B, can be used to obtain the one's complement.
NOR	**nor** **nor.**	**r**A,**r**S,**r**B	The contents of **r**S are ORed with the contents of **r**B and the one's complement of the result is placed into **r**A. **nor** NOR **nor.** NOR with CR Update**.** The dot suffix enables the update of the CR. Note that **nor***x*, with **r**S = **r**B, can be used to obtain the one's complement.
Equivalent	**eqv** **eqv.**	**r**A,**r**S,**r**B	The contents of **r**S are XORed with the contents of **r**B and the complemented result is placed into **r**A. **eqv** Equivalent **eqv.** Equivalent with CR Update**.** The dot suffix enables the update of the CR.
AND with Complement	**andc** **andc.**	**r**A,**r**S,**r**B	The contents of **r**S are ANDed with the one's complement of the contents of **r**B and the result is placed into **r**A. **andc** AND with Complement **andc.** AND with Complement with CR Update**.** The dot suffix enables the update of the CR.
OR with Complement	**orc** **orc.**	**r**A,**r**S,**r**B	The contents of **r**S are ORed with the complement of the contents of **r**B and the result is placed into **r**A. **orc** OR with Complement **orc.** OR with Complement with CR Update**.** The dot suffix enables the update of the CR.
Extend Sign Byte	**extsb** **extsb.**	**r**A,**r**S	The contents of the low-order eight bits of **r**S are placed into the low-order eight bits of **r**A. Bit 24 of **r**S is placed into the remaining high-order bits of **r**A. **extsb** Extend Sign Byte **extsb.** Extend Sign Byte with CR Update**.** The dot suffix enables the update of the CR.
Extend Sign Half Word	**extsh** **extsh.**	**r**A,**r**S	The contents of the low-order 16 bits of **r**S are placed into the low-order 16 bits of **r**A. Bit 16 of **r**S is placed into the remaining high-order bits of **r**A. **extsh** Extend Sign Half Word **extsh.** Extend Sign Half Word with CR Update**.** The dot suffix enables the update of the CR.
Count Leading Zeros Word	**cntlzw** **cntlzw.**	**r**A,**r**S	A count of the number of consecutive zero bits starting at bit 0 of **r**S is placed into **r**A. This number ranges from 0 to 32, inclusive. If Rc = 1 (dot suffix), LT is cleared in CR0. **cntlzw** Count Leading Zeros Word **cntlzw.** Count Leading Zeros Word with CR Update**.** The dot suffix enables the update of the CR.

4.2.1.4 Integer Rotate and Shift Instructions

Rotation operations are performed on data from a GPR, and the result, or a portion of the result, is returned to a GPR. The rotation operations rotate a 32-bit quantity left by a specified number of bit positions. Bits that exit from position 0 enter at position 31.

The rotate and shift instructions employ a mask generator. The mask is 32 bits long and consists of '1' bits from a start bit, Mstart, through and including a stop bit, Mstop, and '0' bits elsewhere. The values of Mstart and Mstop range from 0 to 31. If Mstart > Mstop, the '1' bits wrap around from position 31 to position 0. Thus the mask is formed as follows:

if Mstart ≤ Mstop then

    mask[mstart–mstop] = ones
    mask[all other bits] = zeros
else
    mask[mstart–31] = ones
    mask[0–mstop] = ones
    mask[all other bits] = zeros

It is not possible to specify an all-zero mask. The use of the mask is described in the following sections.

If CR updating is enabled, rotate and shift instructions set CR0[0–2] according to the contents of **r**A at the completion of the instruction. Rotate and shift instructions do not change the values of XER[OV] and XER[SO] bits. Rotate and shift instructions, except algebraic right shifts, do not change the XER[CA] bit.

See Appendix F, "Simplified Mnemonics," for a complete list of simplified mnemonics that allows simpler coding of often-used functions such as clearing the leftmost or rightmost bits of a register, left justifying or right justifying an arbitrary field, and simple rotates and shifts.

4.2.1.4.1 Integer Rotate Instructions

Integer rotate instructions rotate the contents of a register. The result of the rotation is either inserted into the target register under control of a mask (if a mask bit is 1 the associated bit of the rotated data is placed into the target register, and if the mask bit is 0 the associated bit in the target register is unchanged), or ANDed with a mask before being placed into the target register.

Rotate left instructions allow right-rotation of the contents of a register to be performed by a left-rotation of 64 – *n*, where *n* is the number of bits by which to rotate right. It also allows right-rotation of the contents of the low-order 32 bits of a register to be performed by a left-rotation of 32 – *n*, where *n* is the number of bits by which to rotate right.

The integer rotate instructions are summarized in Table 4-4.

Table 4-4. Integer Rotate Instructions

Name	Mnemonic	Operand Syntax	Operation
Rotate Left Word Immediate then AND with Mask	**rlwinm** **rlwinm.**	**r**A,**r**S,SH,MB,ME	The contents of register **r**S are rotated left by the number of bits specified by operand SH. A mask is generated having 1 bits from the bit specified by operand MB through the bit specified by operand ME and 0 bits elsewhere. The rotated data is ANDed with the generated mask and the result is placed into register **r**A. **rlwinm** Rotate Left Word Immediate then AND with Mask **rlwinm.** Rotate Left Word Immediate then AND with Mask with CR Update. The dot suffix enables the update of the CR.
Rotate Left Word then AND with Mask	**rlwnm** **rlwnm.**	**r**A,**r**S,**r**B,MB,ME	The contents of **r**S are rotated left by the number of bits specified by operand in the low-order five bits of **r**B. A mask is generated having 1 bits from the bit specified by operand MB through the bit specified by operand ME and 0 bits elsewhere. The rotated word is ANDed with the generated mask and the result is placed into **r**A. **rlwnm** Rotate Left Word then AND with Mask **rlwnm.** Rotate Left Word then AND with Mask with CR Update. The dot suffix enables the update of the CR.
Rotate Left Word Immediate then Mask Insert	**rlwimi** **rlwimi.**	**r**A,**r**S,SH,MB,ME	The contents of **r**S are rotated left by the number of bits specified by operand SH. A mask is generated having 1 bits from the bit specified by operand MB through the bit specified by operand ME and 0 bits elsewhere. The rotated word is inserted into **r**A under control of the generated mask. **rlwimi** Rotate Left Word Immediate then Mask **rlwimi.** Rotate Left Word Immediate then Mask Insert with CR Update. The dot suffix enables the update of the CR.

4.2.1.4.2 Integer Shift Instructions

The integer shift instructions perform left and right shifts. Immediate-form logical (unsigned) shift operations are obtained by specifying masks and shift values for certain rotate instructions. Simplified mnemonics (shown in Appendix F, “Simplified Mnemonics”) are provided to make coding of such shifts simpler and easier to understand.

Any shift right algebraic instruction, followed by **addze**, can be used to divide quickly by 2^n. The setting of XER[CA] by the shift right algebraic instruction is independent of mode.

Multiple-precision shifts can be programmed as shown in Appendix C, “Multiple-Precision Shifts.”

The integer shift instructions are summarized in Table 4-5.

Table 4-5. Integer Shift Instructions

Name	Mnemonic	Operand Syntax	Operation
Shift Left Word	**slw** **slw.**	**r**A,**r**S,**r**B	The contents of **r**S are shifted left the number of bits specified by operand in the low-order six bits of **r**B. Bits shifted out of position 0 are lost. Zeros are supplied to the vacated positions on the right. The 32-bit result is placed into **r**A. **slw** Shift Left Word **slw.** Shift Left Word with CR Update. The dot suffix enables the update of the CR.
Shift Right Word	**srw** **srw.**	**r**A,**r**S,**r**B	The contents of **r**S are shifted right the number of bits specified by the low-order six bits of **r**B. Bits shifted out of position 31 are lost. Zeros are supplied to the vacated positions on the left. The 32-bit result is placed into **r**A. **srw** Shift Right Word **srw.** Shift Right Word with CR Update. The dot suffix enables the update of the CR.
Shift Right Algebraic Word Immediate	**srawi** **srawi.**	**r**A,**r**S,SH	The contents of **r**S are shifted right the number of bits specified by operand SH. Bits shifted out of position 31are lost. The result is sign extended and placed into **r**A. **srawi** Shift Right Algebraic Word Immediate **srawi.** Shift Right Algebraic Word Immediate with CR Update. The dot suffix enables the update of the CR.
Shift Right Algebraic Word	**sraw** **sraw.**	**r**A,**r**S,**r**B	The contents of **r**S are shifted right the number of bits specified by the low-order six bits of **r**B. Bits shifted out of position 31 are lost. The result is placed into **r**A. **sraw** Shift Right Algebraic Word **sraw.** Shift Right Algebraic Word with CR Update. The dot suffix enables the update of the CR.

4.2.2 Floating-Point Instructions

This section describes the floating-point instructions, which include the following:

- Floating-point arithmetic instructions
- Floating-point multiply-add instructions
- Floating-point rounding and conversion instructions
- Floating-point compare instructions
- Floating-point status and control register instructions
- Floating-point move instructions

Note that MSR[FP] must be set in order for any of these instructions (including the floating-point loads and stores) to be executed. If MSR[FP] = 0 when any floating-point instruction is attempted, the floating-point unavailable exception is taken (see Section 6.4.8, “Floating-Point Unavailable Exception (0x00800)”). See Section 4.2.3, “Load and Store Instructions,” for information about floating-point loads and stores.