4.2.3.1 Integer Load and Store Address Generation

Integer load and store operations generate effective addresses using register indirect with immediate index mode, register indirect with index mode, or register indirect mode. See Section 4.1.4.2, “Effective Address Calculation,” for information about calculating effective addresses. Note that in some implementations, operations that are not naturally aligned may suffer performance degradation. Refer to Section 6.4.6.1, “Integer Alignment Exceptions,” for additional information about load and store address alignment exceptions.

4.2.3.1.1 Register Indirect with Immediate Index Addressing for Integer Loads and Stores

Instructions using this addressing mode contain a signed 16-bit immediate index (d operand) which is sign extended, and added to the contents of a general-purpose register specified in the instruction (**r**A operand) to generate the effective address. If the **r**A field of the instruction specifies **r0**, a value of zero is added to the immediate index (d operand) in place of the contents of **r0**. The option to specify **r**A or 0 is shown in the instruction descriptions as (**r**A|0).

Figure 4-1 shows how an effective address is generated when using register indirect with immediate index addressing.

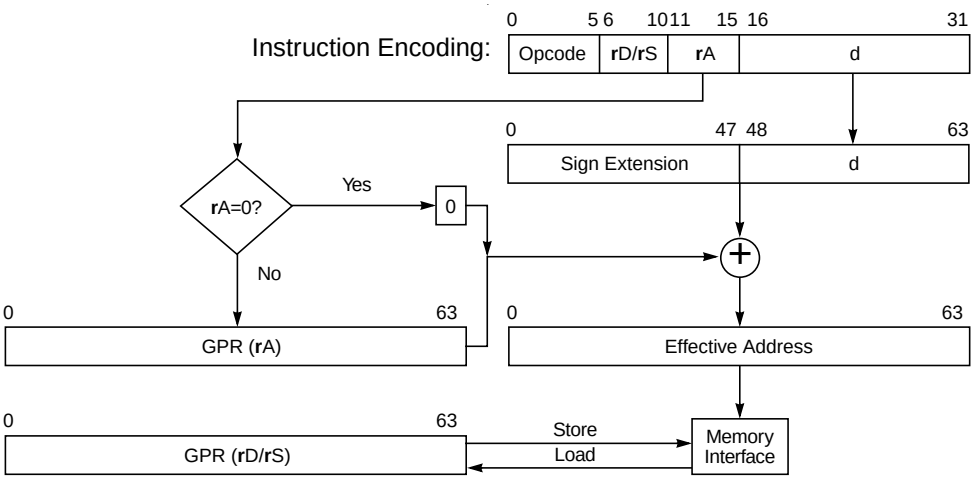

Figure 4-1. Register Indirect with Immediate Index Addressing for Integer Loads/Stores

4.2.3.1.2 Register Indirect with Index Addressing for Integer Loads and Stores

Instructions using this addressing mode cause the contents of two general-purpose registers (specified as operands **r**A and **r**B) to be added in the generation of the effective address. A zero in place of the **r**A operand causes a zero to be added to the contents of the general-purpose register specified in operand **r**B (or the value zero for **lswi** and **stswi** instructions). The option to specify **r**A or 0 is shown in the instruction descriptions as (**r**A|0).

Figure 4-2 shows how an effective address is generated when using register indirect with index addressing.

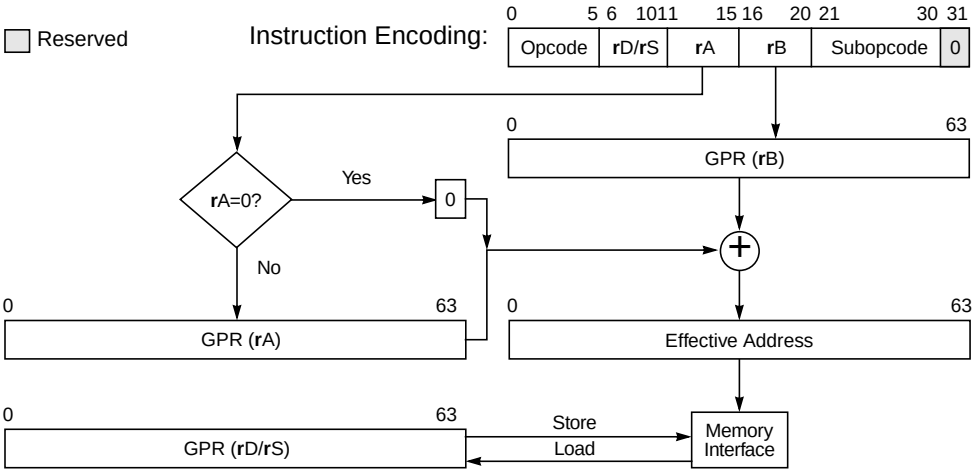

Figure 4-2. Register Indirect with Index Addressing for Integer Loads/Stores

4.2.3.1.3 Register Indirect Addressing for Integer Loads and Stores

Instructions using this addressing mode use the contents of the general-purpose register specified by the **r**A operand as the effective address. A zero in the **r**A operand causes an effective address of zero to be generated. The option to specify **r**A or 0 is shown in the instruction descriptions as (**r**A|0).

Figure 4-3 shows how an effective address is generated when using register indirect addressing.

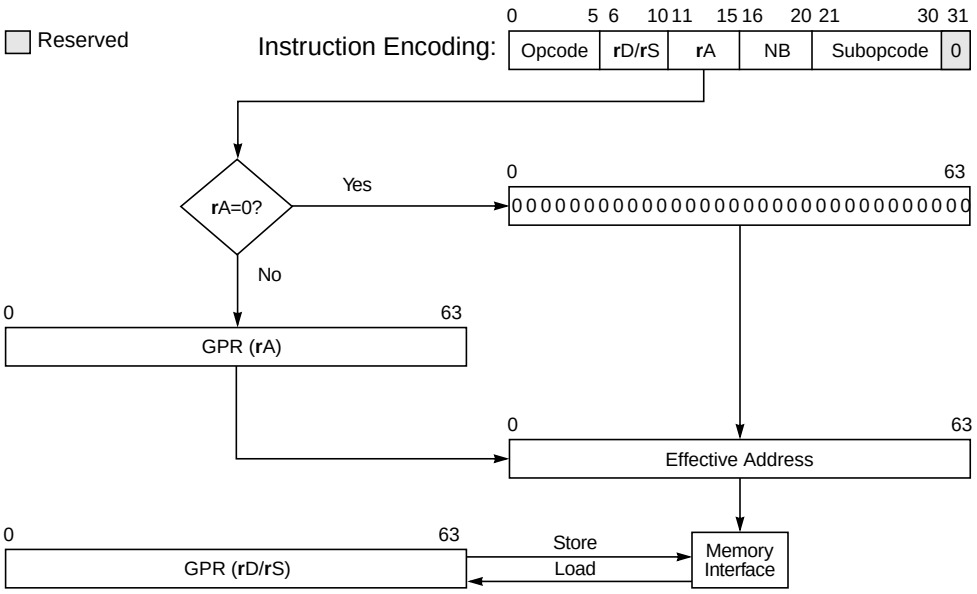

Figure 4-3. Register Indirect Addressing for Integer Loads/Stores

4.2.3.2 Integer Load Instructions

For integer load instructions, the byte, half word, word, or double word addressed by the EA (effective address) is loaded into **r**D. Many integer load instructions have an update form, in which **r**A is updated with the generated effective address. For these forms, if **r**A ≠ 0 and **r**A ≠ **r**D (otherwise invalid), the EA is placed into **r**A and the memory element (byte, half word, word, or double word) addressed by the EA is loaded into **r**D. Note that the PowerPC architecture defines load with update instructions with operand **r**A = 0 or **r**A = **r**D as invalid forms.

The default byte and bit ordering is big-endian in the PowerPC architecture; see Section 3.1.2, “Byte Ordering,” for information about little-endian byte ordering.

Note that in some implementations of the architecture, the load word algebraic instructions (**lha**, **lhax**, **lwa**, **lwax**) and the load with update (**lbzu**, **lbzux**, **lhzu**, **lhzux**, **lhau**, **lhaux**, **lwaux**, **ldu**, **ldux**) instructions may execute with greater latency than other types of load instructions. Moreover, the load with update instructions may take longer to execute in some implementations than the corresponding pair of a nonupdate load followed by an add instruction.

Table 4-13 summarizes the integer load instructions.

Table 4-13. Integer Load Instructions

Name	Mnemonic	Operand Syntax	Operation
Load Byte and Zero	**lbz**	**r**D,d(**r**A)	The EA is the sum (**r**A|0) + d. The byte in memory addressed by the EA is loaded into the low-order eight bits of **r**D. The remaining bits in **r**D are cleared.
Load Byte and Zero Indexed	**lbzx**	**r**D,**r**A,**r**B	The EA is the sum (**r**A|0) + (**r**B). The byte in memory addressed by the EA is loaded into the low-order eight bits of **r**D. The remaining bits in **r**D are cleared.
Load Byte and Zero with Update	**lbzu**	**r**D,d(**r**A)	The EA is the sum (**r**A) + d. The byte in memory addressed by the EA is loaded into the low-order eight bits of **r**D. The remaining bits in **r**D are cleared. The EA is placed into **r**A.
Load Byte and Zero with Update Indexed	**lbzux**	**r**D,**r**A,**r**B	The EA is the sum (**r**A) + (**r**B). The byte in memory addressed by the EA is loaded into the low-order eight bits of **r**D. The remaining bits in **r**D are cleared. The EA is placed into **r**A.
Load Half Word and Zero	**lhz**	**r**D,d(**r**A)	The EA is the sum (**r**A|0) + d. The half word in memory addressed by the EA is loaded into the low-order 16 bits of **r**D. The remaining bits in **r**D are cleared.
Load Half Word and Zero Indexed	**lhzx**	**r**D,**r**A,**r**B	The EA is the sum (**r**A|0) + (**r**B). The half word in memory addressed by the EA is loaded into the low-order 16 bits of **r**D. The remaining bits in **r**D are cleared.
Load Half Word and Zero with Update	**lhzu**	**r**D,d(**r**A)	The EA is the sum (**r**A) + d. The half word in memory addressed by the EA is loaded into the low-order 16 bits of **r**D. The remaining bits in **r**D are cleared. The EA is placed into **r**A.
Load Half Word and Zero with Update Indexed	**lhzux**	**r**D,**r**A,**r**B	The EA is the sum (**r**A) + (**r**B). The half word in memory addressed by the EA is loaded into the low-order 16 bits of **r**D. The remaining bits in **r**D are cleared. The EA is placed into **r**A.
Load Half Word Algebraic	**lha**	**r**D,d(**r**A)	The EA is the sum (**r**A|0) + d. The half word in memory addressed by the EA is loaded into the low-order 16 bits of **r**D. The remaining bits in **r**D are filled with a copy of the most significant bit of the loaded half word.
Load Half Word Algebraic Indexed	**lhax**	**r**D,**r**A,**r**B	The EA is the sum (**r**A|0) + (**r**B). The half word in memory addressed by the EA is loaded into the low-order 16 bits of **r**D. The remaining bits in **r**D are filled with a copy of the most significant bit of the loaded half word.
Load Half Word Algebraic with Update	**lhau**	**r**D,d(**r**A)	The EA is the sum (**r**A) + d. The half word in memory addressed by the EA is loaded into the low-order 16 bits of **r**D. The remaining bits in **r**D are filled with a copy of the most significant bit of the loaded half word. The EA is placed into **r**A.
Load Half Word Algebraic with Update Indexed	**lhaux**	**r**D,**r**A,**r**B	The EA is the sum (**r**A) + (**r**B). The half word in memory addressed by the EA is loaded into the low-order 16 bits of **r**D. The remaining bits in **r**D are filled with a copy of the most significant bit of the loaded half word. The EA is placed into **r**A.
Load Word and Zero	**lwz**	**r**D,d(**r**A)	The EA is the sum (**r**A|0) + d. The word in memory addressed by the EA is loaded into **r**D.
Load Word and Zero Indexed	**lwzx**	**r**D,**r**A,**r**B	The EA is the sum (**r**A|0) + (**r**B). The word in memory addressed by the EA is loaded into **r**D.

Table 4-13. Integer Load Instructions (Continued)

Name	Mnemonic	Operand Syntax	Operation
Load Word and Zero with Update	**lwzu**	**r**D,d(**r**A)	The EA is the sum (**r**A) + d. The word in memory addressed by the EA is loaded into **r**D. The EA is placed into **r**A.
Load Word and Zero with Update Indexed	**lwzux**	**r**D,**r**A,**r**B	The EA is the sum (**r**A) + (**r**B). The word in memory addressed by the EA is loaded into **r**D. The EA is placed into **r**A.

4.2.3.3 Integer Store Instructions

For integer store instructions, the contents of **r**S are stored into the byte, half word, word or double word in memory addressed by the EA (effective address). Many store instructions have an update form, in which **r**A is updated with the EA. For these forms, the following rules apply:

- If **r**A ≠ 0, the effective address is placed into **r**A.
- If **r**S = **r**A, the contents of register **r**S are copied to the target memory element, then the generated EA is placed into **r**A (**r**S).

In general, the PowerPC architecture defines a sequential execution model. However, when a store instruction modifies a memory location that contains an instruction, software synchronization is required to ensure that subsequent instruction fetches from that location obtain the modified version of the instruction.

If a program modifies the instructions it intends to execute, it should call the appropriate system library program before attempting to execute the modified instructions to ensure that the modifications have taken effect with respect to instruction fetching.

The PowerPC architecture defines store with update instructions with **r**A = 0 as an invalid form. In addition, it defines integer store instructions with the CR update option enabled (Rc field, bit 31, in the instruction encoding = 1) to be an invalid form. Table 4-14 provides a summary of the integer store instructions.

Table 4-14. Integer Store Instructions

Name	Mnemonic	Operand Syntax	Operation
Store Byte	**stb**	**r**S,d(**r**A)	The EA is the sum (**r**A|0) + d. The contents of the low-order eight bits of **r**S are stored into the byte in memory addressed by the EA.
Store Byte Indexed	**stbx**	**r**S,**r**A,**r**B	The EA is the sum (**r**A|0) + (**r**B). The contents of the low-order eight bits of **r**S are stored into the byte in memory addressed by the EA.
Store Byte with Update	**stbu**	**r**S,d(**r**A)	The EA is the sum (**r**A) + d. The contents of the low-order eight bits of **r**S are stored into the byte in memory addressed by the EA. The EA is placed into **r**A.

Table 4-14. Integer Store Instructions (Continued)

Name	Mnemonic	Operand Syntax	Operation
Store Byte with Update Indexed	**stbux**	**r**S,**r**A,**r**B	The EA is the sum (**r**A) + (**r**B). The contents of the low-order eight bits of **r**S are stored into the byte in memory addressed by the EA. The EA is placed into **r**A.
Store Half Word	**sth**	**r**S,d**(r**A**)**	The EA is the sum (**r**A|0) + d. The contents of the low-order 16 bits of **r**S are stored into the half word in memory addressed by the EA.
Store Half Word Indexed	**sthx**	**r**S,**r**A,**r**B	The EA is the sum (**r**A|0) + (**r**B). The contents of the low-order 16 bits of **r**S are stored into the half word in memory addressed by the EA.
Store Half Word with Update	**sthu**	**r**S,d**(r**A**)**	The EA is the sum (**r**A) + d. The contents of the low-order 16 bits of **r**S are stored into the half word in memory addressed by the EA. The EA is placed into **r**A.
Store Half Word with Update Indexed	**sthux**	**r**S,**r**A,**r**B	The EA is the sum (**r**A) + (**r**B). The contents of the low-order 16 bits of **r**S are stored into the half word in memory addressed by the EA. The EA is placed into **r**A.
Store Word	**stw**	**r**S,d**(r**A**)**	The EA is the sum (**r**A|0) + d. The contents of **r**S are stored into the word in memory addressed by the EA.
Store Word Indexed	**stwx**	**r**S,**r**A,**r**B	The EA is the sum (**r**A|0) + (**r**B). The contents of **r**S are stored into the word in memory addressed by the EA.
Store Word with Update	**stwu**	**r**S,d**(r**A**)**	The EA is the sum (**r**A) + d. The contents of **r**S are stored into the word in memory addressed by the EA. The EA is placed into **r**A.
Store Word with Update Indexed	**stwux**	**r**S,**r**A,**r**B	The EA is the sum (**r**A) + (**r**B). The contents of **r**S are stored into the word in memory addressed by the EA. The EA is placed into **r**A.

4.2.3.4 Integer Load and Store with Byte-Reverse Instructions

Table 4-15 describes integer load and store with byte-reverse instructions. Note that in some PowerPC implementations, load byte-reverse instructions may have greater latency than other load instructions.

When used in a PowerPC system operating with the default big-endian byte order, these instructions have the effect of loading and storing data in little-endian order. Likewise, when used in a PowerPC system operating with little-endian byte order, these instructions have the effect of loading and storing data in big-endian order. For more information about big-endian and little-endian byte ordering, see Section 3.1.2, “Byte Ordering.”

Table 4-15. Integer Load and Store with Byte-Reverse Instructions

Name	Mnemonic	Operand Syntax	Operation
Load Half Word Byte-Reverse Indexed	**lhbrx**	**r**D,**r**A,**r**B	The EA is the sum (**r**A|0) + (**r**B). The high-order eight bits of the half word addressed by the EA are loaded into the low-order eight bits of **r**D. The next eight higher-order bits of the half word in memory addressed by the EA are loaded into the next eight lower-order bits of **r**D. The remaining **r**D bits are cleared.
Load Word Byte-Reverse Indexed	**lwbrx**	**r**D,**r**A,**r**B	The EA is the sum (**r**A|0) + (**r**B). Bits 0–7 of the word in memory addressed by the EA are loaded into the low-order eight bits of **r**D. Bits 8–15 of the word in memory addressed by the EA are loaded into bits 16–23 of **r**D. Bits 16–23 of the word in memory addressed by the EA are loaded into bits 8–15. Bits 24–31 of the word in memory addressed by the EA are loaded into bits 0–7. The remaining bits in **r**D are cleared.
Store Half Word Byte-Reverse Indexed	**sthbrx**	**r**S,**r**A,**r**B	The EA is the sum (**r**A|0) + (**r**B). The contents of the low-order eight bits of **r**S are stored into the high-order eight bits of the half word in memory addressed by the EA. The contents of the next lower-order eight bits of **r**S are stored into the next eight higher-order bits of the half word in memory addressed by the EA.
Store Word Byte-Reverse Indexed	**stwbrx**	**r**S,**r**A,**r**B	The effective address is the sum (**r**A|0) + (**r**B). The contents of the low-order eight bits of **r**S are stored into bits 0–7 of the word in memory addressed by EA. The contents of the next eight lower-order bits of **r**S are stored into bits 8–15 of the word in memory addressed by the EA. The contents of the next eight lower-order bits of **r**S are stored into bits 16–23 of the word in memory addressed by the EA. The contents of the next eight lower-order bits of **r**S are stored into bits 24–31 of the word addressed by the EA.

4.2.3.5 Integer Load and Store Multiple Instructions

The load/store multiple instructions are used to move blocks of data to and from the GPRs. The load multiple and store multiple instructions may have operands that require memory accesses crossing a 4-Kbyte page boundary. As a result, these instructions may be interrupted by a DSI exception associated with the address translation of the second page. Table 4-16 summarizes the integer load and store multiple instructions.

In the load/store multiple instructions, the combination of the EA and **r**D (**r**S) is such that the low-order byte of GPR31 is loaded from or stored into the last byte of an aligned quad word in memory; if the effective address is not correctly aligned, it may take significantly longer to execute.

In some PowerPC implementations operating with little-endian byte order, execution of an **lmw** or **stmw** instruction causes the system alignment error handler to be invoked; see Section 3.1.2, “Byte Ordering,” for more information.

The PowerPC architecture defines the load multiple word (**lmw**) instruction with **r**A in the range of registers to be loaded, including the case in which **r**A = 0, as an invalid form.

Table 4-16. Integer Load and Store Multiple Instructions

Name	Mnemonic	Operand Syntax	Operation
Load Multiple Word	**lmw**	**r**D,d**(r**A**)**	The EA is the sum (**r**A|0) + d. *n* = (32 – **r**D).
Store Multiple Word	**stmw**	**r**S,d**(r**A**)**	The EA is the sum (**r**A|0) + d. *n* = (32 – **r**S).

4.2.3.6 Integer Load and Store String Instructions

The integer load and store string instructions allow movement of data from memory to registers or from registers to memory without concern for alignment. These instructions can be used for a short move between arbitrary memory locations or to initiate a long move between misaligned memory fields. However, in some implementations, these instructions are likely to have greater latency and take longer to execute, perhaps much longer, than a sequence of individual load or store instructions that produce the same results. Table 4-17 summarizes the integer load and store string instructions.

Load and store string instructions execute more efficiently when **r**D or **r**S = 5, and the last register loaded or stored is less than or equal to 12.

In some PowerPC implementations operating with little-endian byte order, execution of a load or string instruction causes the system alignment error handler to be invoked; see Section 3.1.2, “Byte Ordering,” for more information.

Table 4-17. Integer Load and Store String Instructions

Name	Mnemonic	Operand Syntax	Operation
Load String Word Immediate	**lswi**	**r**D**,r**A,NB	The EA is (**r**A|0).
Load String Word Indexed	**lswx**	**r**D**,r**A**,r**B	The EA is the sum (**r**A|0) + (**r**B).
Store String Word Immediate	**stswi**	**r**S**,r**A,NB	The EA is (**r**A|0).
Store String Word Indexed	**stswx**	**r**S**,r**A**,r**B	The EA is the sum (**r**A|0) + (**r**B).

Load string and store string instructions may involve operands that are not word-aligned. As described in Section 6.4.6, “Alignment Exception (0x00600),” a misaligned string operation suffers a performance penalty compared to an aligned operation of the same type. A non–word-aligned string operation that crosses a double-word boundary is also slower than a word-aligned string operation.