5304804 0010039 751 LLC

LR33000 Family Instruction Set Details

This document provides detailed descriptions of the instructions in the MIPS instruction set that are common to all of the LR33000 Self-Embedding processors. Any instructions unique to a processor are defined in that processor's technical manual.

This document contains the following sections:

- Classes
- Formats
- Instruction Notation Conventions
- Instruction Set Details
- Unimplemented Instructions
- Opcode Bit Encoding

Classes

Instructions are divided into the following classes:

- Load/Store Instructions
- Computational Instructions
- Jump and Branch Instructions
- Coprocessor Instructions
- Coprocessor 0 Instructions
- Special Instructions

Load/Store Instructions

The load/store instructions move data between memory and general registers. These instructions are all I-type, because the only mode supported is base register + 16-bit immediate offset.

Computational Instructions

The computational instructions perform arithmetic, logical, and shift operations on values in registers. They occur in both R-type (both operands are registers) and I-type (one operand is a 16-bit immediate) formats.

Jump and Branch Instructions

The jump and branch instructions change the control flow of a program. Jumps are always to absolute 26-bit word addresses (J-type format, for subroutine calls) or 32-bit register byte addresses (R-type, for returns and dispatches). Branches have 16-bit offsets relative to the program counter (I-type). Jump and Link instructions save a return address in register *r31*.

All jump and branch instructions are implemented with a delay of exactly one instruction. That is, the instruction immediately following a jump or branch (occupying the delay slot) is always executed while the target instruction is being fetched from storage. Neither a jump nor a branch instruction may occupy a delay slot. The result of such an operation is not detected and is undefined.

If an exception or interrupt prevents the completion of a legal instruction during a delay slot, the LR33000 sets the EPC Register to point at the jump or branch instruction that precedes it. When the code is restarted, both the jump or branch instruction and the instruction in the delay slot are re-executed.

Because jump and branch instructions may be restarted after exceptions or interrupts, they must be restartable. Therefore when a jump or branch instruction stores a return link value, the link register (*r31*) may not be used as a target register in the exception handler.

Coprocessor Instructions

The LR33000 processors do not support all external coprocessors and do not implement the entire coprocessor instruction set. All of the LR33000 processors support the Branch on Coprocessor z True (BCzT) and Branch on Coprocessor z False (BCzF) instructions, which allow software to test the condition of the CPC[3:1] signals on the LR33000's external interface.

Check the specific LR33000 Technical Manual to see which coprocessors are supported. All unsupported coprocessor instructions produce undefined results if executed.

Because most of the coprocessor instructions produce undefined results, software should disable the coprocessors by setting the Cu[3:1] bits in the

Status Register to zero. When the coprocessors are disabled in this fashion, coprocessor instructions in the instruction stream cause the LR33000 processors to take a coprocessor unusable exception. To use the BCzT and BCzF instructions, software must enable the coprocessor associated with the signal to be tested.

Coprocessor 0 Instructions

Coprocessor 0 (CP0) provides the exception-handling, cache invalidate control, and debug facilities of the processor, which are monitored and controlled via the CP0 registers. Access to these registers is provided by the Move From System Control Processor (MFC0) and Move To System Control Processor (MTC0) instructions.

Because the LR33000 processors have no Translation Lookaside Buffer (TLB), they do not implement the TLB load, store, and probe entry instructions. The LR33000 processors take a Reserved Instruction Exception if they decode a TLB access instruction.

Special Instructions

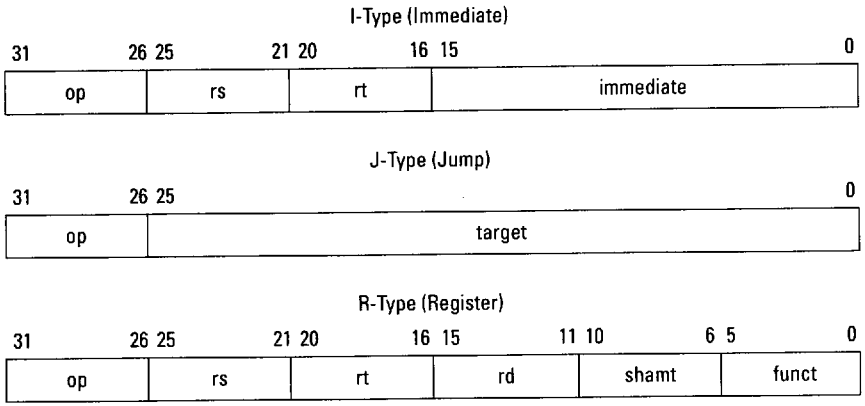
The special instructions perform a variety of tasks, including movement of data between special and general registers, trap, and breakpoint. These instructions are always R-type.

Formats

Every LR33000 instruction consists of a single word (32 bits) aligned on a word boundary. As shown in Figure 1, an instruction is in one of three formats: I-type, J-type, or R-type. In this manual, all variable subfields in an instruction format (such as *rt, rd, immediate*, etc.) are shown in lower case. All subfields that are filled with constant values have uppercase names.

The two instruction subfields *op* and *funct* have constant six-bit values for specific instructions. These values are given uppercase mnemonic names. For example, *op* = LB in the Load Byte instruction and *op* = SPECIAL and *funct* = ADD in the Add instruction.

Figure 1
LR33000 Instruction Formats



Notes:

op	6-bit operation code
rs	5-bit source register specifier
rt	5-bit target (source/destination register)
immediate	16-bit immediate, branch displacement, or address displacement
target	26-bit jump target address
rd	5-bit destination register specifier
shamt	5-bit shift amount
funct	6-bit function field

A single field may have both fixed and variable subfields, such that the name contains both uppercase and lowercase characters. For example, MFCz (Move From Coprocessor) represents four different six-bit opcodes, which designate one of three coprocessor classes (1 through 3), concatenated with the fixed five-bit subfield MF.

For the sake of clarity, an alias is sometimes used for a variable subfield in the formats of specific instructions. For example, *base* is used in place of *rs* in the format for load and store instructions. Such an alias is always lower case, since it refers to a variable subfield.

Instruction Notation Conventions

This section provides information on the instruction's internal operation section. In the detailed instruction descriptions, the operation section describes the operation performed by each instruction using a high-level language notation. This section defines the special symbols used in the high-level language notation and defines the functions used for load and store operations.

Table 1 describes the special symbols used in the high-level language notation.

Table 1 Instruction Operation Notations

Symbol	*Meaning*
←	Assignment
||	Bit string concatenation
x^y	Replication of bit value x into a y-bit string. Note that x is always a single-bit value.
$x_{y..z}$	Selection of bits y through z of bit string x. Little-endian bit notation is always used. If y is less than z, this expression is an empty (zero length) bit string.
+	Two's complement addition
-	Two's complement subtraction
*	Two's complement multiplication
div	Two's complement integer division
mod	Two's complement modulo
<	Two's complement less-than comparison
AND	Bitwise logic AND
OR	Bitwise logic OR
XOR	Bitwise logic XOR
NOR	Bitwise logic NOR
GPR[*x*]	General Register x. Valid values of x are 0 to 31 for each of the 32 CPU registers.
CPR[*z, x*]	Coprocessor unit z, general register x
CCR[*z, x*]	Coprocessor unit z, control register x
BigEndian	Endian mode as configured at reset
HI	EntryHI register
LO	EntryLO register
PC	Program Counter
T + *i*	Indicates the time steps (CPU cycles) between operations. Thus operations identified as occurring at T+i are performed during the cycle following the one where the instruction was initiated. This type of operation occurs with loads, stores, jumps, branches, and coprocessor instructions.
Addr	Effective Address generated by load and store instructions
pAddr	Physical Address: generally the same as Addr, but mapped for accesses in *kseg0 and kseg1*.

The variables GPR[0:31], PC, HI, LO, CPR[0:3, 0:31], CPC[0:3], and BigEndian represent the state of the machine and are therefore static. All other variables used in describing instructions are dynamic and local to the function or instruction involved.

Instruction Notation Examples

The following examples illustrate the application of some of the instruction notation conventions:

Example 1:

$GPR[r2] \leftarrow immediate \,||\, 0^{16}$

In example 1, 16 zero bits are concatenated with an immediate value (typically 16 bits). The resulting 32-bit string (with the lower 16 bits set to zero) is assigned to General Purpose Register *r2*.

Example 2:

$(immediate_{15})^{16} \,||\, immediate_{15:0}$

In example 2, bit 15 (the sign bit) of an immediate value is extended for 16 bit positions. The result is concatenated with bits 15 through 0 of the immediate value to form a 32-bit, sign-extended value.

Memory Access Specification

The functions listed in Table 2 summarize the handling of addresses, cache memory, and physical memory. These functions are found in the load and store instruction operation sections.

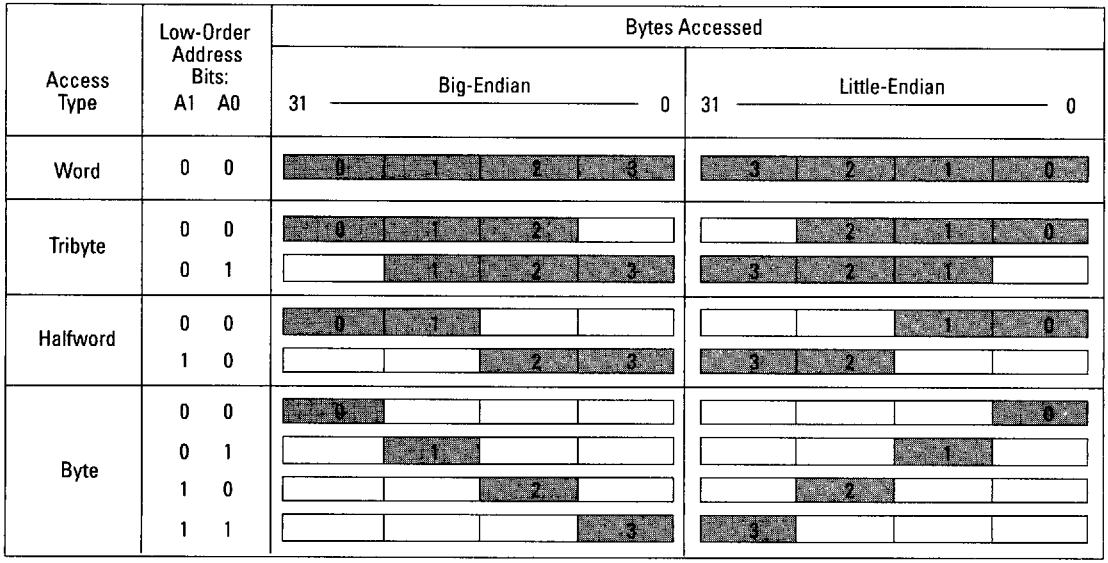
Table 2
Load/Store Common Functions

Symbol	*Meaning*
BIU (uncached, DATA)	The Bus Interface Unit (BIU) maps accesses in *kseg0* and *kseg1* to 0x0000.0000 – 0x1FFF.FFFF and tests for cacheability.
Load Memory (uncached, accType, pAddr, I or D)	The LR33000 processors load the addressed word from cache or main memory. If the cache is enabled for this access, the word is returned and loaded into the cache.
Store Memory (uncached, accType, memoryWord, pAddr)	The LR33000 processors store the data into the cache and/or main memory at the specified address.

The load/store instruction operation code (opcode) determines the access type, which in turn indicates the size of the data item to be loaded or stored. Regardless of access type or byte-numbering order (endianness), the address specifies the byte that has the smallest byte address of all the bytes in the addressed field. For a big-endian machine, this is the leftmost byte; for a little-endian machine, this is the rightmost byte.

The bytes that are used within the addressed word can be determined directly from the access type and the two low-order bits of the address, as shown in Figure 2. Note that certain combinations of access type and low-order address bits can never occur; only the combinations shown in Figure 2 are permissible.

Figure 2
Byte Specifications for Loads/Stores



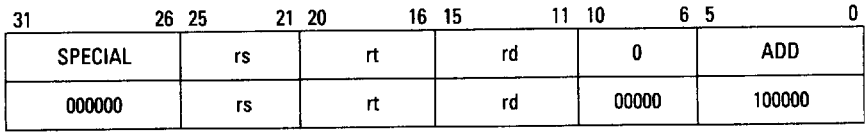
Instruction Set Details

The instruction set descriptions are listed alphabetically. Each instruction definition consists of the instruction format, instruction syntax, instruction definition, internal operation, and possible exceptions. The description of the immediate causes and manner of handling exceptions is omitted from the instruction descriptions in this document. Refer to the chapter on exceptions in the specific LR33000 Technical Manual for detailed descriptions of exceptions and handling.

The instruction format divides the 32-bit instruction into fields. The first line lists the format with the mnemonic field names. The following line(s) lists the format with the bit encodings substituted for the mnemonics. For the coprocessor instructions, the coprocessor numbers are listed to the left of the respective formats.

ADD Add

Format



31 26	25 21	20 16	15 11	10 6	5 0
SPECIAL	rs	rt	rd	0	ADD
000000	rs	rt	rd	00000	100000

Syntax

ADD *rd, rs, rt*

Description

The contents of general register *rs* and the contents of general register *rt* are added to form a 32-bit result. The result is placed into general register *rd*.

An Overflow Exception occurs if the two highest-order, carry-out bits differ (two's complement overflow).

Operation

T: GPR[rd] ← GPR[rs] + GPR[rt]

Exceptions

Overflow Exception

ADDI — Add Immediate

Format

31 — 26	25 — 21	20 — 16	15 — 0
ADDI	rs	rt	immediate
001000	rs	rt	immediate

Syntax

ADDI *rt, rs, immediate*

Description

The 16-bit *immediate* is sign-extended and added to the contents of general register *rs* to form a 32-bit result. The result is placed into general register *rt*.

An Overflow Exception occurs if the two highest-order, carry-out bits differ (two's complement overflow).

Operation

T: $GPR[rt] \leftarrow GPR[rs] + (immediate_{15})^{16} \,||\, immediate_{15:0}$

Exceptions

Overflow Exception

ADDIU — Add Immediate Unsigned

Format

31 26	25 21	20 16	15 0
ADDIU	rs	rt	immediate
001001	rs	rt	immediate

Syntax

ADDIU *rt, rs, immediate*

Description

The 16-bit *immediate* is sign-extended and added to the contents of general register *rs* to form a 32-bit result. The result is placed into general register *rt*. No Overflow Exception occurs under any circumstances.

Note that the only difference between this instruction and the ADDI instruction is that ADDIU never causes an Overflow Exception.

Operation

T: $GPR[rt] \leftarrow GPR[rs] + (immediate_{15})^{16} \,||\, immediate_{15:0}$

Exceptions

None

ADDU — Add Unsigned

Format

31 26	25 21	20 16	15 11	10 6	5 0
SPECIAL	rs	rt	rd	0	ADDU
000000	rs	rt	rd	00000	100001

Syntax

ADDU *rd, rs, rt*

Description

The contents of general register *rs* and the contents of general register *rt* are added to form a 32-bit result. The result is placed into general register *rd*.

No Overflow Exception occurs under any circumstances.

Note that the only difference between this instruction and the ADD instruction is that ADDU never causes an Overflow Exception.

Operation

T: GPR[rd] ← GPR[rs] + GPR[rt]

Exceptions

None

AND — AND

Format

31 26	25 21	20 16	15 11	10 6	5 0
SPECIAL	rs	rt	rd	0	AND
000000	rs	rt	rd	00000	100100

Syntax

AND *rd, rs, rt*

Description

The contents of general register *rs* are combined with the contents of general register *rt* in a bitwise, logical-AND operation. The result is placed into general register *rd.*

Operation

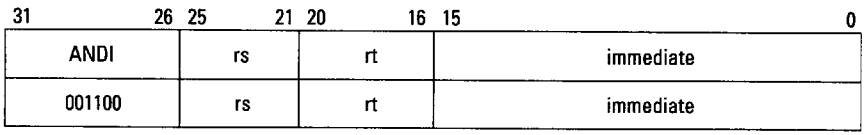
T: GPR[rd] ← GPR[rs] AND GPR[rt]

Exceptions

None

ANDI — AND Immediate

Format



Syntax

ANDI *rt, rs, immediate*

Description

The 16-bit *immediate* is zero-extended and combined with the contents of general register *rs* in a bitwise, logical-AND operation. The result is placed into general register *rt*.

Operation

T: $GPR[rt] \leftarrow 0^{16} \,||\, (immediate \text{ AND } GPR[rs]_{15:0})$

Exceptions

None

BCzF Branch on Coprocessor z False

Format

	31 26	25 21	20 16	15 0
	COPz	BC	BCF	offset
CP0	010000	01000	00000	offset
CP1	010001	01000	00000	offset
CP2	010010	01000	00000	offset
CP3	010011	01000	00000	offset

Syntax

BCzF *offset*

Description

A branch target address is computed from the sum of the address of the instruction in the delay slot and the 16-bit *offset*, shifted left two bits, and sign-extended to 32 bits. If the contents of Coprocessor *z*'s condition line (CPC[z]) is false, then the program branches to the target address with a delay of one instruction.

Because the condition line is sampled during the previous instruction, there must be at least one instruction between a coprocessor instruction that changes the condition line and this instruction.

Note that for all LR33000 processors this instruction is valid only for $z = 0$ (Coprocessor 0). Refer to the specific LR33000 Technical Manual for other coprocessors that may be valid.

Operation

```
T-1:  condition ← not CPC[z]
T:    target ← (offset15)^14 || offset || 0^2
T+1:  if condition then
          PC ← PC + target
      endif
```

Exceptions

Coprocessor Unusable Exception

Compatibility Note

When in kernel mode with Cu0 set to zero, the LR33000 processors take a CpU Exception when they decode a BC0T or BC0F instruction. In like conditions, however, the R3000 *does not* take an exception.

BCzT Branch on Coprocessor z True

Format

	31 26	25 21	20 16	15 0
	COPz	BC	BCT	offset
CP0	010000	01000	00001	offset
CP1	010001	01000	00001	offset
CP2	010010	01000	00001	offset
CP3	010011	01000	00001	offset

Syntax

BC*z*T *offset*

Description

A branch target address is computed from the sum of the address of the instruction in the delay slot and the 16-bit *offset*, shifted left two bits, and sign-extended to 32 bits. If the contents of Coprocessor *z*'s condition line (CPC[z]) are true, then the program branches to the target address with a delay of one instruction.

Because the condition line is sampled during the previous instruction, there must be *at least* one instruction between a coprocessor instruction that changes the condition line and this instruction. Two instructions in between is recommended.

Note that for all LR33000 processors this instruction is valid only for $z = 0$ (Coprocessor 0). Refer to the specific LR33000 Technical Manual for other coprocessors that may be valid.

Operation

```
T-1: condition ←CPC[z]
T:   target ←(offset 15)^14 || offset || 0^2
T+1: if condition then
        PC ← PC + target
     endif
```

Exceptions

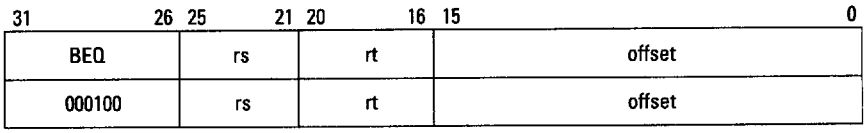
Coprocessor Unusable Exception

Compatibility Note

When in kernel mode with Cu0 set to zero, the LR33000 processors take a CpU Exception when they decode a BC0T or BC0F instruction. In like conditions, however, the R3000 *does not* take an exception.

BEQ Branch on Equal

Format



31 26	25 21	20 16	15 0
BEQ	rs	rt	offset
000100	rs	rt	offset

Syntax

BEQ *rs, rt, offset*

Description

A branch target address is computed from the sum of the address of the instruction in the delay slot and the 16-bit *offset*, shifted left two bits, and sign-extended to 32 bits. The contents of general register *rs* and the contents of general register *rt* are compared. If the contents of the two registers are equal, then the program branches to the target address with a delay of one instruction.

Operation

```
T:   target ←(offset15)^14 || offset || 0^2
     condition ←(GPR[rs] = GPR[rt])
T+1: if condition then
        PC ←PC + target
     endif
```

Exceptions

None

BGEZ — Branch on Greater than or Equal to Zero

Format

31 — 26	25 — 21	20 — 16	15 — 0
REGIMM	rs	BGEZ	offset
000001	rs	00001	offset

Syntax

BGEZ *rs, offset*

Description

A branch target address is computed from the sum of the address of the instruction in the delay slot and the 16-bit *offset*, shifted left two bits, and sign-extended to 32 bits. If the sign bit of the contents of general register *rs* is cleared, then the program branches to the target address with a delay of one instruction.

Operation

```
T:   target ← (offset15)^14 || offset || 0^2
     condition ← (GPR[rs]31 = 0)
T+1: if condition then
         PC ← PC + target
     endif
```

Exceptions

None

BGEZAL — Branch on Greater than or Equal to Zero And Link

Format

31 26	25 21	20 16	15 0
REGIMM	rs	BGEZAL	offset
000001	rs	10001	offset

Syntax

BGEZAL *rs, offset*

Description

A branch target address is computed from the sum of the address of the instruction in the delay slot and the 16-bit *offset,* shifted left two bits, and sign-extended to 32 bits. Unconditionally, the address of the instruction after the delay slot is placed in the link register, *r31*. If the sign bit of the contents of general register *rs* is cleared, then the program branches to the target address with a delay of one instruction.

General register *rs* may not be general register *r31*, because such an instruction is not restartable. However, an attempt to execute this instruction is *not* trapped.

Operation

```
T:   target ← (offset15)^14 || offset || 0^2
     condition ← (GPR[rs]31 = 0)
     GPR[31] ← PC + 8
T+1: if condition then
         PC ← PC + target
     endif
```

Exceptions

None

BGTZ — Branch on Greater Than Zero

Format

31 26	25 21	20 16	15 0
BGTZ	rs	0	offset
000111	rs	00000	offset

Syntax

BGTZ *rs, offset*

Description

A branch target address is computed from the sum of the address of the instruction in the delay slot and the 16-bit *offset*, shifted left two bits, and sign-extended to 32 bits. The contents of general register *rs* are compared to zero. If the contents of general register *rs* have the sign bit cleared and are not equal to zero, then the program branches to the target address with a delay of one instruction.

Operation

```
T:   target ← (offset15)^14 || offset || 0^2
     condition ← (GPR[rs]31 = 0) OR (GPR[rs] ≠ 0^32)
T+1: if condition then
        PC ← PC + target
     endif
```

Exceptions

None

BLEZ — Branch on Less than or Equal to Zero

Format

31 26	25 21	20 16	15 0
BLEZ	rs	0	offset
000110	rs	00000	offset

Syntax

BLEZ *rs, offset*

Description

A branch target address is computed from the sum of the address of the instruction in the delay slot and the 16-bit *offset*, shifted left two bits, and sign-extended to 32 bits. The contents of general register *rs* are compared to zero. If the contents of general register *rs* have the sign bit set or are equal to zero, then the program branches to the target address with a delay of one instruction.

Operation

```
T:   target ← (offset15)^14 || offset || 0^2
     condition ← (GPR[rs]31 = 1) OR (GPR[rs] = 0^32)
T+1: if condition then
        PC ← PC + target
     endif
```

Exceptions

None

BLTZ — Branch on Less Than Zero

Format

31 — 26	25 — 21	20 — 16	15 — 0
REGIMM	rs	BLTZ	offset
000001	rs	00000	offset

Syntax

BLTZ *rs, offset*

Description

A branch target address is computed from the sum of the address of the instruction in the delay slot and the 16-bit *offset*, shifted left two bits, and sign-extended to 32 bits. If the sign bit of the contents of general register *rs* is set, then the program branches to the target address with a delay of one instruction.

Operation

```
T:   target ← (offset15)^14 || offset || 0^2
     condition ← (GPR[rs]31 = 1)
T+1: if condition then
        PC ← PC + target
     endif
```

Exceptions

None

BLTZAL — Branch on Less Than Zero And Link

Format

31 26	25 21	20 16	15 0
REGIMM	rs	BLTZAL	offset
000001	rs	10000	offset

Syntax

BLTZAL *rs, offset*

Description

A branch target address is computed from the sum of the address of the instruction in the delay slot and the 16-bit *offset,* shifted left two bits, and sign-extended to 32 bits. Unconditionally, the address of the instruction after the delay slot is placed in the link register, *r31*. If the sign bit of the contents of general register *rs* is set, then the program branches to the target address with a delay of one instruction.

General register *rs* may not be general register *r31*, because such an instruction is not restartable. However, an attempt to execute this instruction is *not* trapped.

Operation

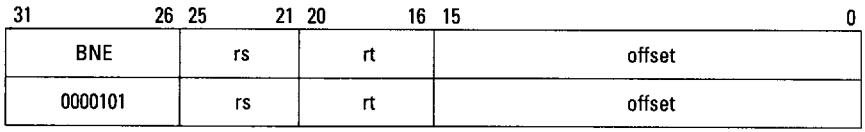
```
T:   target ← (offset15)^14 || offset || 0^2
     condition ← (GPR[rs]31 = 1)
     GPR[31] ← PC + 8
T+1: if condition then
        PC ← PC + target
     endif
```

Exceptions

None

BNE — Branch on Not Equal

Format



31 26	25 21	20 16	15 0
BNE	rs	rt	offset
0000101	rs	rt	offset

Syntax

BNE *rs, rt, offset*

Description

A branch target address is computed from the sum of the address of the instruction in the delay slot and the 16-bit *offset,* shifted left two bits, and sign-extended to 32 bits. The contents of general register *rs* and the contents of general register *rt* are compared. If the contents of the two registers are not equal, then the program branches to the target address with a delay of one instruction.

Operation

```
T:   target ← (offset15)^14 || offset || 0^2
     condition ← (GPR[rs] ≠ GPR[rt])
T+1: if condition then
         PC ← PC + target
     endif
```

Exceptions

None

BREAK Breakpoint

Format

31 26	25 6	5 0
SPECIAL	code	BREAK
000000	code	001101

Syntax BREAK

Description A Breakpoint Exception occurs, immediately and unconditionally transferring control to the exception handler.

The *code* field is available for use for software parameters, but is retrieved only by the exception handler by loading the contents of the memory word containing the instruction.

Operation T: PC ← ExceptionHandler

Exceptions Breakpoint Exception

DIV Divide

Format

31 26	25 21	20 16	15 6	5 0
SPECIAL	rs	rt	0	DIV
000000	rs	rt	0000000000	011010

Syntax

DIV *rs, rt*

Description

The contents of general register *rs* are divided by the contents of general register *rt* with both operands treated as 32-bit two's complement values. When the operation is completed, the quotient word of the double result is loaded into special register LO, and the remainder word of the double result is loaded into special register HI.

No Overflow Exception occurs under any circumstances.

If either of the two preceding instructions is MFHI or MFLO, the results of those instructions are undefined. Correct operation requires separating reads of HI or LO from writes by two or more instructions.

Thirty-five (35) cycles are required between a DIV and a subsequent MFHI or MFLO operation in order that no interlock or stall occurs.

Operation

```
T-2: LO ← undefined
     HI ← undefined
T-1: LO ← undefined
     HI ← undefined
T:   LO ← GPR[rs] div GPR[rt]
     HI ← GPR[rs] mod [rt]
```

Exceptions

None

DIVU — Divide Unsigned

Format

31 26	25 21	20 16	15 11	10 6	5 0
SPECIAL	rs	rt	0	0	DIVU
000000	rs	rt	00000	0000000000	011011

Syntax

DIVU *rs, rt*

Description

The contents of general register *rs* are divided by the contents of general register *rt* with both operands treated as 32-bit unsigned values. When the operation is completed, the quotient word of the double result is loaded into special register LO, and the remainder word of the double result is loaded into special register HI.

No Overflow Exception occurs under any circumstances.

The results of this operation are undefined when the divisor is zero. Typically this instruction will be followed by additional instructions to check for a zero divisor.

If either of the two preceding instructions is MFHI or MFLO, the results of those instructions are undefined. Correct operation requires separating reads of HI or LO from writes by two or more instructions.

Thirty-five (35) cycles are required between a DIVU and a subsequent MFHI or MFLO operation in order that no interlock or stall occurs.

Operation

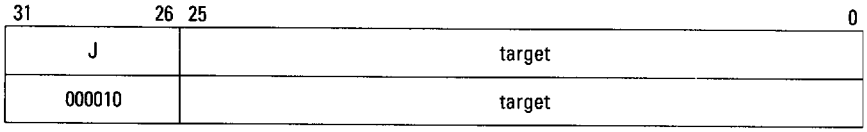
```
T-2: LO ← undefined
     HI ← undefined
T-1: LO ← undefined
     HI ← undefined
T:   LO ← (0 || GPR[rs]) div (0 || GPR[rt])
     HI ← (0 || GPR[rs]) mod (0 || GPR[rt])
```

Exceptions

None

J — Jump

Format



Syntax

J *target*

Description

The 26-bit *target* address is shifted left two bits and combined with the high-order four bits of the delay slot address. The program unconditionally jumps to the calculated address with a delay of one instruction.

Operation

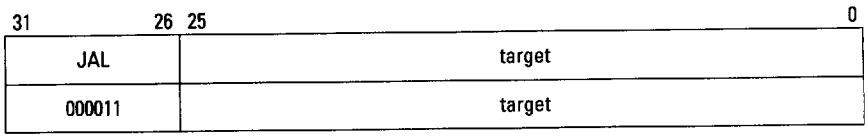
T: temp ← target
T+1: PC ← PC[31:28] || temp || 0^2

Exceptions

None

JAL — Jump And Link

Format



Syntax

JAL *target*

Description

The 26-bit *target* address is shifted left two bits and combined with the high-order four bits of the delay slot address. The program unconditionally jumps to the calculated address with a delay of one instruction. The address of the instruction after the delay slot is placed in the *Link* register, *r31*.

Operation

```
T:   temp ← target
     GPR[31] ← PC + 8
T+1: PC ← PC[31:28] || temp || 0^2
```

Exceptions

None

JALR — Jump And Link Register

Format

31 26	25 21	20 16	15 11	10 6	5 0
SPECIAL	rs	0	rd	0	JALR
000000	rs	00000	rd	00000	001001

Syntax

JALR *rs*
JALR *rs, rd*

Description

The program unconditionally jumps to the address contained in general register *rs* with a delay of one instruction. The address of the instruction after the delay slot is placed in general register, *rd*. The default value of *rd*, if omitted in the assembly language instruction, is 31.

Register specifiers *rs* and *rd* may not be equal, because such an instruction does not have the same effect when re-executed. However, an attempt to execute this instruction is *not* trapped; the result of executing such an instruction is undefined.

Since instructions must be word-aligned, a *Jump and Link* register instruction must specify a target register (*rs*) whose two low-order bits are zero. If these low-order bits are not zero, an address exception will occur when the jump target instruction is subsequently fetched.

Operation

```
T:   temp ← GPR[rs]
     GPR[rd] ← PC + 8
T+1: PC ← temp
```

Exceptions

None

JR Jump Register

Format

31 26	25 21	20 6	5 0
SPECIAL	rs	0	JR
000000	rs	00000000000000	001000

Syntax

JR *rs*

Description

The program unconditionally jumps to the address contained in general register *rs* with a delay of one instruction.

Since instructions must be word-aligned, a *Jump* register instruction must specify a target register (*rs)* whose two low-order bits are zero. If these low-order bits are not zero, an address exception will occur when the jump target instruction is subsequently fetched.

Operation

T: temp ← GPR[rs]
T+1: PC ← temp

Exceptions

None

LB — Load Byte

Format

31 — 26	25 — 21	20 — 16	15 — 0
LB	base	rt	offset
1000000	base	rt	offset

Syntax

LB *rt, offset(base)*

Description

The 16-bit *offset* is sign-extended and added to the contents of general register *base* to form a 32-bit unsigned effective address. The contents of the byte at the memory location specified by the effective address are sign-extended and loaded into general register *rt*.

The contents of general register *rt* are undefined for time T of the instruction immediately following this load instruction.

Operation

```
T:   Addr ← (offset15)16 || offset[15:0] + GPR[base]
     (pAddr, uncached) ← BIU (Addr, DATA)
     mem ← LoadMemory (uncached, BYTE, pAddr, DATA)
     byte ← Addr1:0 XOR BigEndian2
     GPR[rt] ← undefined
T+1: GPR[rt] ← (mem7+8*byte)24 || mem7+8*byte:8 *byte
```

Exceptions

Bus Error Exception
Address Error Exception

LBU Load Byte Unsigned

Format

31 26	25 21	20 16	15 0
LBU	base	rt	offset
100100	base	rt	offset

Syntax

LBU *rt, offset(base)*

Description

The 16-bit *offset* is sign-extended and added to the contents of general register *base* to form a 32-bit unsigned effective address. The contents of the byte at the memory location specified by the effective address are zero-extended and loaded into general register *rt*.

The contents of general register *rt* are undefined for time T of the instruction immediately following this load instruction.

Operation

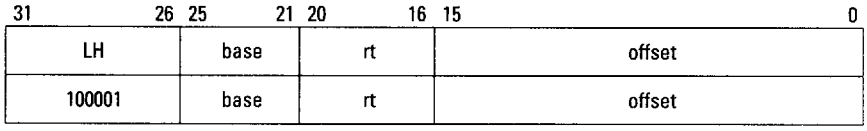
```
T:   Addr ← (offset15)^16 || offset[15:0] + GPR[base]
     (pAddr, uncached) ← BIU (Addr, DATA)
     mem ← LoadMemory (uncached, BYTE, pAddr, DATA)
     byte ← Addr1:0 XOR BigEndian^2
     GPR[rt] ← undefined
T+1: GPR[rt] ←0^24 || mem7+8*byte:8*byte
```

Exceptions

Bus Error Exception
Address Error Exception

LH Load Halfword

Format



31 26	25 21	20 16	15 0
LH	base	rt	offset
100001	base	rt	offset

Syntax

LH *rt, offset(base)*

Description

The 16-bit *offset* is sign-extended and added to the contents of general register *base* to form a 32-bit unsigned effective address. The contents of the halfword at the memory location specified by the effective address are sign-extended and loaded into general register *rt*.

If the least-significant bit of the effective address is non-zero, an Address Error Exception occurs.

The contents of general register *rt* are undefined during time T of the instruction immediately following this load instruction.

Operation

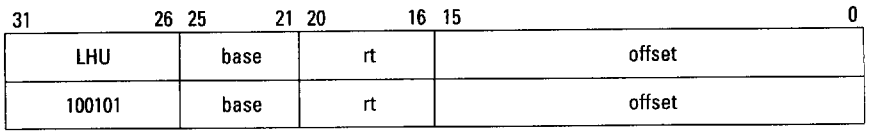
```
T:   Addr ← (offset15)^16 || offset[15:0] + GPR[base]
     (pAddr, uncached) ← BIU (Addr, DATA)
     mem ← LoadMemory (uncached, HALFWORD, pAddr, DATA)
     byte ← Addr1:0 XOR (BigEndian || 0)
     GPR[rt] ← undefined
T+1: GPR[rt] ←(mem15+8*byte)^16 || mem15+8*byte:8*byte
```

Exceptions

Bus Error Exception
Address Error Exception

LHU Load Halfword Unsigned

Format



31 26	25 21	20 16	15 0
LHU	base	rt	offset
100101	base	rt	offset

Syntax

LHU *rt, offset(base)*

Description

The 16-bit *offset* is sign-extended and added to the contents of general register *base* to form a 32-bit unsigned effective address. The contents of the halfword at the memory location specified by the effective address are zero-extended and loaded into general register *rt*.

If the least-significant bit of the effective address is non-zero, an Address Error Exception occurs.

The contents of general register *rt* are undefined for time T of the instruction immediately following this load instruction.

Operation

```
T:   Addr ← (offset15)^16 || offset[15:0] + GPR[base]
     (pAddr, uncached) ← BIU (Addr, DATA)
     mem ← LoadMemory (uncached, HALFWORD, pAddr, DATA)
     byte ← Addr1:0 XOR (BigEndian || 0)
     GPR[rt] ← undefined
T+1: GPR[rt] ←0^16 || mem15+8*byte:8*byte
```

Exceptions

Bus Error Exception
Address Error Exception

LUI — Load Upper Immediate

Format

31 — 26	25 — 21	20 — 16	15 — 0
LUI	0	rt	immediate
001111	00000	rt	immediate

Syntax

LUI *rt, immediate*

Description

The 16-bit *immediate* is shifted left 16 bits and concatenated to 16 bits of zeros. The result is placed into general register *rt*.

Operation

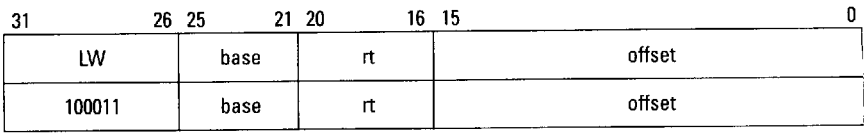
T: GPR[rt] ← immediate || 0^{16}

Exceptions

None

LW Load Word

Format



31 26	25 21	20 16	15 0
LW	base	rt	offset
100011	base	rt	offset

Syntax

LW *rt, offset(base)*

Description

The 16-bit *offset* is sign-extended and added to the contents of general register *base* to form a 32-bit unsigned effective address. The contents of the word at the memory location specified by the effective address are loaded into general register *rt*.

If either of the two least-significant bits of the effective address is non-zero, an Address Error Exception occurs.

The contents of general register *rt* are undefined for time T of the instruction immediately following this load instruction.

Operation

```
T:   Addr ← (offset15)16 || offset[15:0] + GPR[base]
     (pAddr, uncached) ← BIU (Addr, DATA)
     mem ← LoadMemory (uncached, WORD, pAddr, DATA)
     GPR[rt] ← undefined
T+1: GPR[rt] ←mem
```

Exceptions

Bus Error Exception
Address Error Exception

LWL Load Word Left

Format

31 26	25 21	20 16	15 0
LWL	base	rt	offset
100010	base	rt	offset

Syntax

LWL *rt, offset(base)*

Description

The 16-bit *offset* is sign-extended and added to the contents of general register *base* to form a 32-bit unsigned effective address. The contents of the word at the memory location specified by the effective address are shifted left so that the addressed byte is in the leftmost byte of a word. The bytes loaded from memory are merged with the contents of general register *rt*, and the result is loaded into general register *rt*.

The contents of general register *rt* are undefined for time T of the instruction immediately following this load instruction. However, the contents of general register *rt* are internally bypassed within the processor so that it is permissible to specify register *rt* as the target register of a load instruction on the previous instruction.

Address Error Exceptions due to byte alignment are suppressed by this instruction.

Operation

```
T:   Addr ← (offset15)16 || offset[15:0] + GPR[base]
     (pAddr, uncached) ← BIU (Addr, DATA)
     byte ← Addr1:0 XOR BigEndian2
     if Big Endian then
        pAddr ← pAddr31:2 || 02
     endif
     mem ← LoadMemory (uncached, BYTE, pAddr, DATA)
T+1: GPR[rt] ← mem7+8* byte:0 || GPR[rt]23-8* byte:0
```

Exceptions

Bus Error Exception
Address Error Exception

LWR Load Word Right

Format

31 26	25 21	20 16	15 0
LWR	base	rt	offset
100110	base	rt	offset

Syntax

LWR *rt, offset(base)*

Description

The 16-bit *offset* is sign-extended and added to the contents of general register *base* to form a 32-bit unsigned effective address. The contents of the word at the memory location specified by the effective address are shifted right so that the addressed byte is in the rightmost byte of a word. The bytes loaded from memory are merged with the contents of general register *rt*, and the result is loaded into general register *rt*.

The contents of general register *rt* are undefined for time T of the instruction immediately following this load instruction. However, the contents of general register *rt* are internally bypassed within the processor so that it is permissible to specify register *rt* as the target register of a load instruction on the previous instruction.

Address Error Exceptions due to byte alignment are suppressed by this instruction.

Operation

```
T:   Addr ← (offset15)^16 || offset15:0 + GPR[base]
     (pAddr, uncached) ← BIU (Addr, DATA)
     byte ← Addr1:0 XOR BigEndian^2
     if Big Endian then
        pAddr ← pAddr31:2 || 0^2
     endif
     mem ← LoadMemory (uncached, WORD-byte, pAddr, DATA)
T+1: GPR[rt] ← GPR[rt]31:32-8*byte || mem31:8*byte
```

Exceptions

Bus Error Exception
Address Error Exception

MFCz — Move From Coprocessor

Format

	31 26	25 21	20 16	15 11	10 0
	COPz	MF	rt	rd	0
CP0	010000	00000	rt	rd	00000000000
CP1	010001	00000	rt	rd	00000000000
CP2	010010	00000	rt	rd	00000000000
CP3	010011	00000	rt	rd	00000000000

Syntax

MFC*z* *rt, rd*

Description

The contents of coprocessor register *rd* of coprocessor unit *z* are loaded into general register *rt.*

For the LR33000 processors, MFC*z* is valid only with *z* equal to zero (CP0). Results for the execution of MFC*z* with *z* equal to one, two, or three are undefined.

If MFC0 is executed with CP0 disabled (user mode), the LR33000 processors take a reserved instruction exception.

Operation

T: data ← CPR[z,rd]
T+1: GPR[rt] ← data

Exceptions

Coprocessor unusable exception
Reserved instruction exception

Compatibility Note

When in user mode with Cu0 set to zero, the LR33000 processors take a Reserved Instruction (RI) Exception if they decode a RFE, MTC0, or MFC0 instruction. In like conditions, the R3000 takes a Coprocessor Unusable Exception (CpU).

MFHI Move From HI

Format

31 26	25 16	15 11	10 6	5 0
SPECIAL	0	rd	0	MFHI
000000	0000000000	rd	00000	010000

Syntax

MFHI *rd*

Description

The contents of special register HI are loaded into general register *rd*.

To ensure proper operation in the event of interruptions, the two instructions that follow an MFHI instruction may not be an instruction that modifies the HI register (MULT, MULTU, DIV, DIVU, MTHI).

In order that no interlock or stall occurs, an MFHI instruction should follow a MULT or MULTU instruction by a minimum of 12 cycles; a MFHI instruction should follow a DIV or DIVU instruction by a minimum of 35 cycles.

Operation

T: GPR[rd] ← HI

Exceptions

None

MFLO — Move From LO

Format

31 26	25 16	15 11	10 6	5 0
SPECIAL	0	rd	0	MFLO
000000	0000000000	rd	00000	010010

Syntax

MFLO *rd*

Description

The contents of special register LO are loaded into general register *rd*.

To ensure proper operation in the event of interruptions, the two instructions that follow a MFLO instruction may not be an instruction that modifies the LO register (MULT, MULTU, DIV, DIVU, MTLO).

In order that no interlock or stall occurs, a MFLO instruction should follow a MULT or MULTU instruction by a minimum of 12 cycles; a MFLO instruction should follow a DIV or DIVU instruction by a minimum of 35 cycles.

Operation

T: GPR[rd] ← LO

Exceptions

None

MTCz — Move To Coprocessor

Format

	31–26	25–21	20–16	15–11	10–0
	COPz	MT	rt	rd	0
CP0	010000	00100	rt	rd	00000000000
CP1	010001	00100	rt	rd	00000000000
CP2	010010	00100	rt	rd	00000000000
CP3	010011	00100	rt	rd	00000000000

Syntax

MTC*z* *rt, rd*

Description

The contents of general register *rt* are loaded into coprocessor register *rd* of coprocessor unit *z*.

For the LR33000 processors, MTC*z* is valid only with *z* equal to zero (CP0). Results for the execution of MTC*z* with *z* equal to one, two, or three are undefined.

If MTC0 is executed with CP0 disabled (user mode), the LR33000 processors take a reserved instruction exception.

Operation

T: data ← GPR[rt]
T+1: CPR[z,rd] ← data

Exceptions

Coprocessor unusable exception
Reserved instruction exception

Compatibility Note

When in user mode with Cu0 set to zero, the LR33000 processors take a Reserved Instruction (RI) Exception if they decode a RFE, MTC0, or MFC0 instruction. In like conditions, the R3000 takes a Coprocessor Unusable Exception (CpU).

MTHI — Move To HI

Format

31 26	25 21	20 6	5 0
SPECIAL	rs	0	MTHI
000000	rs	00000000000000	010001

Syntax

MTHI *rs*

Description

The contents of general register *rs* are loaded into special register HI.

The contents of special register LO are undefined if a MTHI operation is executed following a MULT, MULTU, DIV, or DIVU instruction but before an MFLO, MFHI, MTLO, or MTHI instruction.

In order that no interlock or stall occurs, a MTHI instruction should follow a MULT or MULTU instruction by a minimum of 12 cycles; a MTHI instruction should follow a DIV or DIVU instruction by a minimum of 35 cycles.

Operation

T-2: HI ← undefined
T-1: HI ← undefined
T: HI ← GPR[rs]

Exceptions

None

MTLO Move To LO

Format

31 26	25 21	20 6	5 0
SPECIAL	rs	0	MTLO
000000	rs	00000000000000	010011

Syntax

MTLO *rs*

Description

The contents of general register *rs* are loaded into special register LO.

The contents of special register HI are undefined if an MTLO operation is executed following a MULT, MULTU, DIV, or DIVU instruction but before any MFLO, MFHI, MTLO, or MTHI instruction.

In order that no interlock or stall occurs, an MTLO instruction should follow a MULT or MULTU instruction by a minimum of 12 cycles; an MTLO instruction should follow a DIV or DIVU instruction by a minimum of 35 cycles.

Operation

```
T-2:  LO ← undefined
T-1:  LO ← undefined
T:    LO ← GPR[rs]
```

Exceptions

None

MULT Multiply

Format

31 26	25 21	20 16	15 6	5 0
SPECIAL	rs	rt	0	MULT
000000	rs	rt	0000000000	001110

Syntax

MULT *rs, rt*

Description

The contents of general register *rs* and the contents of general register *rt* are multiplied with both operands treated as 32-bit two's complement values. When the operation is completed, the low-order word of the double result is loaded into special register LO, and the high-order word of the double result is loaded into special register HI.

No Overflow Exception occurs under any circumstances.

If either of the two preceding instructions is MFHI or MFLO, the results of those instructions are undefined. Correct operation requires separating reads of HI or LO from writes by two or more instructions.

Twelve (12) cycles are required between a MULT and a subsequent MFHI or MFLO operation in order that no interlock or stall occurs.

Operation

```
T-2:  LO ← undefined
      HI ← undefined
T-1:  LO ← undefined
      HI ← undefined
T:    t ← GPR[rs] * GPR[rt]
      LO ← t31:0
      HI ← t63:32
```

Exceptions

None

MULTU — Multiply Unsigned

Format

31 26	25 21	20 16	15 6	5 0
SPECIAL	rs	rt	0	MULTU
000000	rs	rt	0000000000	011000

Syntax

MULTU *rs, rt*

Description

The contents of general register *rs* and the contents of general register *rt* are multiplied with both operands treated as 32-bit unsigned values. When the operation is completed, the low-order word of the double result is loaded into special register LO, and the high-order word of the double result is loaded into special register HI.

No Overflow Exception occurs under any circumstances.

If either of the two preceding instructions is MFHI or MFLO, the results of those instructions are undefined. Correct operation requires separating reads of HI or LO from writes by two or more instructions.

Twelve (12) cycles are required between a MULTU and a subsequent MFHI or MFLO operation in order that no interlock or stall occurs.

Operation

```
T-2: LO ← undefined
     HI ← undefined
T-1: LO ← undefined
     HI ← undefined
T:   t ← (0 || GPR[rs]) * (0 || GPR[rt])
     LO ← t31:0
     HI ← t63:32
```

Exceptions

None

NOR NOR

Format

31 26	25 21	20 16	15 11	10 6	5 0
SPECIAL	rs	rt	rd	0	NOR
000000	rs	rt	rd	00000	100111

Syntax

NOR *rd, rs, rt*

Description

The contents of general register *rs* are combined with the contents of general register *rt* in a bitwise, logical-NOR operation. The result is placed into general register *rd*.

Operation

T: GPR[rd] ← GPR[rs] NOR GPR[rt]

Exceptions

None

OR

OR

Format

31 26	25 21	20 16	15 11	10 6	5 0
SPECIAL	rs	rt	rd	0	OR
000000	rs	rt	rd	00000	100101

Syntax

OR *rd, rs, rt*

Description

The contents of general register *rs* are combined with the contents of general register *rt* in a bitwise, logical-OR operation. The result is placed into general register *rd*.

Operation

T: GPR[rd] ← GPR[rs] OR GPR[rt]

Exceptions

None

ORI — OR Immediate

Format

31 — 26	25 — 21	20 — 16	15 — 0
ORI	rs	rt	immediate
001101	rs	rt	immediate

Syntax

ORI *rt, rs, immediate*

Description

The 16-bit *immediate* is zero-extended and combined with the contents of general register *rs* in a bitwise, logical-OR operation. The result is placed into general register *rt*.

Operation

T: $GPR[rt] \leftarrow GPR[rs]_{31:16} \,||\, (immediate\ OR\ GPR[rs]_{15:0})$

Exceptions

None

RFE — Restore From Exception

Format

31 — 26	25	24 — 5	4 — 0
COP0	CO	0	RFE
010000	1	00000	10000

Syntax

RFE

Description

The RFE command restores the *previous* interrupt mask and kernel/user-mode bits (IEp and KUp) of the *Status* register (SR) into the corresponding *current* status bits (IEc and KUc) and restores the *old* status bits (IEo and KUo) into the corresponding status bits (IEp and KUp). The *old* status bits remain unchanged.

The operation of memory references associated with load/store instructions immediately prior to an RFE instruction is unspecified. Normally, the RFE instruction follows in the delay slot of a JR *(Jump* register) instruction to restore the PC.

Operation

T: $SR \leftarrow SR_{31:4} \,||\, SR_{5:2}$

Exceptions

Reserved Instruction Exception

Compatibility Note

When in user mode with Cu0 set to zero, the LR33000 processors take a Reserved Instruction (RI) Exception if they decode a RFE, MTC0, or MFC0 instruction. In like conditions, the R3000 takes a Coprocessor Unusable Exception (CpU).

SB Store Byte

Format

31 26	25 21	20 16	15 0
SB	base	rt	offset
101000	base	rt	offset

Syntax

SB *rt, offset(base)*

Description

The 16-bit *offset* is sign-extended and added to the contents of general register *base* to form a 32-bit unsigned effective address. The least-significant byte of the contents of register *rt* is stored at the effective address.

Operation

```
T:   Addr ← (offset15)^16 || offset15:0 + GPR[base]
     (pAddr, uncached) ← BIU (Addr, DATA)
     byte ← Addr1:0 XOR BigEndian^2
     data ← GPR[rt]31-8*byte:0 || 0^8*byte
T+1: StoreMemory ← (uncached, BYTE, data, pAddr, DATA)
```

Exceptions

Bus Error Exception
Address Error Exception

SH — Store Halfword

Format

31 26	25 21	20 16	15 0
SH	base	rt	offset
101001	base	rt	offset

Syntax

SH *rt, offset(base)*

Description

The 16-bit *offset* is sign-extended and added to the contents of general register *base* to form a 32-bit unsigned effective address. The least-significant halfword of the contents of register *rt* is stored at the effective address.

If the least-significant bit of the effective address is non-zero, an Address Error Exception occurs.

Operation

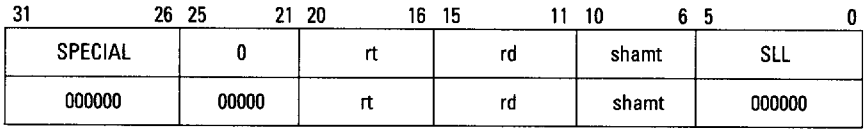
```
T:   Addr ← (offset15)^16 || offset15:0 + GPR[base]
     (pAddr, uncached) ← BIU (Addr, DATA)
     byte ← Addr1:0 XOR (BigEndian || 0)
     data ← GPR[rt]31-8* byte:0 || 0^8*byte
T+1: StoreMemory ← (uncached, HALFWORD, data, pAddr, DATA)
```

Exceptions

Bus Error Exception
Address Error Exception

SLL — Shift Left Logical

Format



31 26	25 21	20 16	15 11	10 6	5 0
SPECIAL	0	rt	rd	shamt	SLL
000000	00000	rt	rd	shamt	000000

Syntax

SLL *rd, rt, shamt*

Description

The contents of general register *rt* are shifted left by *shamt* bits. Zeros are inserted into the low-order bits. The 32-bit result is placed in register *rd.*

Operation

T: $s \leftarrow shamt$
$GPR[rd] \leftarrow GPR[rt]_{31-s:0} \,||\, 0^{s}$

Exceptions

None

SLLV — Shift Left Logical Variable

Format

31 26	25 21	20 16	15 11	10 6	5 0
SPECIAL	rs	rt	rd	0	SLLV
000000	rs	rt	rd	00000	000100

Syntax

SLLV *rd, rt, rs*

Description

The contents of general register *rt* are shifted left by the number of bits specified by the low-order five bits of the contents of general register *rs*. Zeros are inserted into the low-order bits. The 32-bit result is placed in register *rd*.

Operation

T: $s \leftarrow GPR[rs]_{4:0}$
$GPR[rd] \leftarrow GPR[rt]_{31-s:0} \,||\, 0^{s}$

Exceptions

None

SLT — Set on Less Than

Format

31 26	25 21	20 16	15 11	10 6	5 0
SPECIAL	rs	rt	rd	0	SLT
000000	rs	rt	rd	00000	101010

Syntax

SLT *rd, rs, rt*

Description

The contents of general register *rt* are compared with the contents of general register *rs*. Considering both quantities as signed 32-bit integers, if the contents of general register *rs* are less than the contents of general register *rt*, the result is set to one; otherwise, the result is set to zero. The result is placed into general register *rd*.

No Overflow Exception occurs under any circumstances. The comparison is valid even if the subtraction used during the comparison overflows.

Operation

```
T:  if GPR[rs] < GPR[rt] then
        GPR[rd] ← 0^31 || 1
    else
        GPR[rd] ← 0^32
    endif
```

Exceptions

None

SLTI — Set on Less Than Immediate

Form

31 26	25 21	20 16	15 0
SLTI	rs	rt	immediate
001010	rs	rt	immediate

Syntax

SLTI *rt, rs, immediate*

Description

The 16-bit *immediate* is sign-extended and compared from the contents of general register *rs*. Considering both quantities as signed 32-bit integers, if *rs* is less than the sign-extended immediate, the result is set to one; otherwise the result is set to zero. The result is placed into general register *rt*.

No Overflow Exception occurs under any circumstances. The comparison is valid even if the subtraction used during the comparison overflows.

Operation

```
T:  if GPR[rs] < (immediate15)^16 || immediate15:0 then
        GPR[rt] ← 0^31 || 1
    else
        GPR[rt] ← 0^32
    endif
```

Exceptions

None

SLTIU — Set on Less Than Immediate Unsigned

Format

31 26	25 21	20 16	15 0
SLTIU	rs	rt	immediate
001011	rs	rt	immediate

Syntax

SLTIU *rt, rs, immediate*

Description

The 16-bit *immediate* is sign-extended and compared from the contents of general register *rs*. Considering both quantities as unsigned 32-bit integers, if *rs* is less than the sign-extended immediate, the result is set to one; otherwise the result is set to zero. The result is placed into general register *rt*.

No Overflow Exception occurs under any circumstances. The comparison is valid even if the subtraction used during the comparison overflows.

Operation

```
T:  if (0 || GPR[rs]) < 0 || (immediate15)16 || immediate15:0 then
        GPR[rt] ← 0^31 || 1
    else
        GPR[rt] ← 0^32
    endif
```

Exceptions

None

SLTU — Set on Less Than Unsigned

Format

31 26	25 21	20 16	15 11	10 6	5 0
SPECIAL	rs	rt	rd	0	SLTU
000000	rs	rt	rd	00000	101011

Syntax

SLTU *rd, rs, rt*

Description

The contents of general register *rt* are compared with the contents of general register *rs*. Considering both quantities as unsigned 32-bit integers, if the contents of general register *rs* are less than the contents of general register *rt*, the result is set to one; otherwise, the result is set to zero. The result is placed into general register *rd*.

No Overflow Exception occurs under any circumstances. The comparison is valid even if the subtraction used during the comparison overflows.

Operation

```
T:  if (0 || GPR[rs]) < (0 || GPR[rt]) then
        GPR[rd] ← 0^31 || 1
    else
        GPR[rd] ← 0^32
    endif
```

Exceptions

None

SRA — Shift Right Arithmetic

Format

31 26	25 21	20 16	15 11	10 6	5 0
SPECIAL	0	rt	rd	shamt	SRA
000000	00000	rt	rd	shamt	000011

Syntax

SRA *rd, rt, shamt*

Description

The contents of general register *rt* are shifted right by *shamt* bits. The high-order bits are sign-extended. The 32-bit result is placed in register *rd*.

Operation

T: $s \leftarrow$ shamt
$GPR[rd] \leftarrow (GPR[rt]_{31})^{s} \,||\, GPR[rt]_{31:s}$

Exceptions

None

SRAV — Shift Right Arithmetic Variable

Format

31 26	25 21	20 16	15 11	10 6	5 0
SPECIAL	rs	rt	rd	0	SRAV
000000	rs	rt	rd	00000	000111

Syntax

SRAV *rd, rt, rs*

Description

The contents of general register *rt* are shifted right by the number of bits specified by the low-order five bits of the contents of general register *rs*. The high-order bits are sign-extended. The 32-bit result is placed in register *rd*.

Operation

T: $s \leftarrow GPR[rs]_{4:0}$
$GPR[rd] \leftarrow (GPR[rt]_{31})^{s} \,||\, GPR[rt]_{31:s}$

Exceptions

None

SRL — Shift Right Logical

Format

31 26	25 21	20 16	15 11	10 6	5 0
SPECIAL	0	rt	rd	shamt	SRL
000000	00000	rt	rd	shamt	000010

Syntax

SRL *rd, rt, shamt*

Description

The contents of general register *rt* are shifted right by *shamt* bits. Zeros are inserted into the high-order bits. The 32-bit result is placed in register *rd*.

Operation

T: s ← shamt
GPR[rd] ← 0^s || $GPR[rt]_{31:s}$

Exceptions

None

SRLV — Shift Right Logical Variable

Format

31 26	25 21	20 16	15 11	10 6	5 0
SPECIAL	rs	rt	rd	0	SRLV
000000	rs	rt	rd	00000	000110

Syntax

SRLV *rd, rt, rs*

Description

The contents of general register *rt* are shifted right by the number of bits specified by the low-order five bits of the contents of general register *rs*. Zeros are inserted into the high-order bits. The 32-bit result is placed in register *rd*.

Operation

T: $s \leftarrow GPR[rs]_{4:0}$
$GPR[rd] \leftarrow 0^s \,||\, GPR[rt]$

Exceptions

None

SUB Subtract

Format

31 26	25 21	20 16	15 11	10 6	5 0
SPECIAL	rs	rt	rd	0	SUB
000000	rs	rt	rd	00000	100010

Syntax

SUB *rd, rs, rt*

Description

The contents of general register *rt* are subtracted from the contents of general register *rs* to form a 32-bit result. The result is placed into general register *rd.*

An Overflow Exception occurs if the two highest-order, carry-out bits differ (two's complement overflow).

Operation

T: GPR[rd] ← GPR[rs] – GPR[rt]

Exceptions

Overflow Exception

SUBU — Subtract Unsigned

Format

31 26	25 21	20 16	15 11	10 6	5 0
SPECIAL	rs	rt	rd	0	SUBU
000000	rs	rt	rd	00000	100011

Syntax

SUBU *rd, rs, rt*

Description

The contents of general register *rt* are subtracted from the contents of general register *rs* to form a 32-bit result. The result is placed into general register *rd*.

No Overflow Exception occurs under any circumstances.

Note that the only difference between this instruction and the SUB instruction is that SUBU never causes an Overflow Exception.

Operation

T: GPR[rd] ← GPR[rs] – GPR[rt]

Exceptions

None

SW — Store Word

Format

31 26	25 21	20 16	15 0
SW	base	rt	offset
101011	base	rt	offset

Syntax

SW *rt, offset(base)*

Description

The 16-bit *offset* is sign-extended and added to the contents of general register *base* to form a 32-bit unsigned effective address. The contents of general register *rt* are stored at the memory location specified by the effective address.

If either of the two least-significant bits of the effective address is nonzero, an Address Error Exception occurs.

Operation

```
T:   Addr ← (offset15)^16 || offset15:0 + GPR[base]
     (pAddr, uncached) ← BIU (Addr, DATA)
     data ← GPR[rt]
T+1: StoreMemory (uncached, WORD, data, pAddr DATA)
```

Exceptions

Bus Error Exception
Address Error Exception

SWL Store Word Left

Format

31 26	25 21	20 16	15 0
SWL	base	rt	offset
101010	base	rt	offset

Syntax

SWL *rt, offset(base)*

Description

The 16-bit *offset* is sign-extended and added to the contents of general register *base* to form a 32-bit unsigned effective address. The contents of general register *rt* are shifted right so that the leftmost byte of the word is in the position of the addressed byte. The bytes contained in the word after shifting are stored into the word containing the addressed byte.

Address Error Exceptions due to byte alignment are suppressed by this instruction.

Operation

```
T:   Addr ← (offset15)^16 || offset15:0 + GPR[base]
     (pAddr, uncached) ← BIU (Addr, DATA)
     byte ← Addr1:0 XOR BigEndian^2
     data ← 0^(24-8*byte) || GPR[rt]31:24-8* byte
     If BigEndian then
        pAddr ← pAddr31:2 || 0^2
     endif
T+1: StoreMemory (uncached, BYTE, data, pAddr, DATA)
```

Exceptions

Bus Error Exception
Address Error Exception

SWR Store Word Right

Format

31 26	25 21	20 16	15 0
SWR	base	rt	offset
101110	base	rt	offset

Syntax

SWR *rt, offset(base)*

Description

The 16-bit *offset* is sign-extended and added to the contents of general register *base* to form a 32-bit unsigned effective address. The contents of general register *rt* are shifted left so that the rightmost byte of the word is in the position of the addressed byte. The bytes contained in the word after shifting are stored into the word containing the addressed byte.

Address Error Exceptions due to byte alignment are suppressed by this instruction.

Operation

```
T:   Addr ← (offset15)^16 || offset15:0 + GPR[base]
     (pAddr, uncached) ← BIU (Addr, DATA)
     byte ← Addr1:0 XOR BigEndian^2
     data ← GPR[rt]31-8*byte:0 || 0^8*byte
     If BigEndian then
        pAddr ← pAddr31:2 || 0^2
     endif
T + 1:StoreMemory (uncached, WORD-byte, data, pAddr, DATA)
```

Exceptions

Bus Error Exception
Address Error Exception

SYSCALL System Call

Format

31 26	25 6	5 0
SPECIAL	code	SYSCALL
000000	code	001100

Syntax SYSCALL

Description A System Call Exception occurs, immediately and unconditionally transferring control to the exception handler.

The *code* field is available for use for software parameters, but is retrieved only by the exception handler by loading the contents of the memory word containing the instruction.

Operation

```
T:  PC ← ExceptionHandler
```

Exceptions System Call Exception

XOR — Exclusive OR

Format

31 26	25 21	20 16	15 11	10 6	5 0
SPECIAL	rs	rt	rd	0	XOR
000000	rs	rt	rd	00000	100110

Syntax

XOR *rd, rs, rt*

Description

The contents of general register *rs* are combined with the contents of general register *rt* in a bitwise, logical exclusive-OR operation. The result is placed into general register *rd.*

Operation

```
T:  GPR[rd] ← GPR[rs] XOR GPR[rt]
```

Exceptions

None

XORI — Exclusive OR Immediate

Format

31 26	25 21	20 16	15 0
XORI	rs	rt	immediate
001110	rs	rt	immediate

Syntax

XORI *rt, rs, immediate*

Description

The 16-bit *immediate* is zero-extended and combined with the contents of general register *rs* in a bitwise, logical exclusive-OR operation. The result is placed into general register *rt.*

Operation

T: $GPR[rt] \leftarrow GPR[rs]_{31:16} \,||\, (immediate \text{ XOR } GPR[rs]_{15:0})$

Exceptions

None

Unimplemented Instructions

The LR33000 Self-Embedding processors do not implement the entire MIPS instruction set. The instructions that are not implemented are:

- TLBP
- TLBR
- TLBWI
- TLBWR

The LR33000 processors do not implement these instructions. If the processors decode one of these instructions, then they take a reserved instruction exception.

Opcode Bit Encoding

This section lists the major and minor opcodes with their respective bit encodings in tabular form. Table 3 lists the bit encoding for the LR33000 family major opcodes. Tables 4 through 8 list the bit encoding for the minor opcodes. Undefined bit encodings cause Reserved Instruction Exceptions and are reserved for future versions of the architecture.

Table 3 Major Opcode Bit Encoding

Major Opcode	*Bit Encoding (Bits 31:26)*	*Major Opcode*	*Bit Encoding (Bits 31:26)*
SPECIAL	000000	COP0	010000
REGIMM	000001	COP1	010001
J	000010	COP2	010010
JAL	000011	COP3	010011
BEQ	000100	LB	100000
BNE	000101	LH	100001
BLEZ	000110	LWL	100010
BGTZ	000111	LW	100011
ADDI	001000	LBU	100100
ADDIU	001001	LHU	100101
SLTI	001010	LWR	100110
SLTIU	001011	SB	101000
ANDI	001100	SH	101001
ORI	001101	SWL	101010
XORI	001110	SW	101011
LUI	001111	SWR	101110

Table 4 SPECIAL Minor Opcode Bit Encoding

Special Minor Opcode	*Bit Encoding (Bits 5:0)*	*Special Minor Opcode*	*Bit Encoding (Bits 5:0)*
SLL	000000	MULT	001110
SRL	000010	MULTU	011000
SRA	000011	DIV	011010
SLLV	000100	DIVU	011011
SRLV	000110	ADD	100000
SRAV	000111	ADDU	100001
JR	001000	SUB	100010
JALR	001001	SUBU	100011
SYSCALL	001100	AND	100100
BREAK	001101	OR	100101
MFHI	010000	XOR	100110
MTHI	010001	NOR	100111
MFLO	010010	SLT	101010
MTLO	010011	SLTU	101011

Table 5 REGIMM Minor Opcode Bit Encoding

REGIMM Minor Opcode	*Bit Encoding (Bits 20:16)*
BLTZ	00000
BGEZ	00001
BLTZAL	10000
BGEZAL	10001

Table 6 COPz rs Minor Opcode Bit Encoding

COPz rs Minor Opcode	*Bit Encoding (Bits 25:21)*
MF	00000
MT	00100
BC	01000
CO	1xxxx

Table 7 COPz rt Minor Opcode Bit Encoding

COPz rt Minor Opcode	*Bit Encoding (Bits 20:16)*
BCF	00000
BCT	00001

Table 8 COP0 Minor Opcode Bit Encoding

COP0 Minor Opcode	*Bit Encoding (Bits 4:0)*
RFE	10000

Find price and stock options from leading distributors for lr330 on Findchips.com:

https://findchips.com/search/lr330

Find CAD models and details for this part:

https://findchips.com/detail/lr330