DDIVU Doubleword Divide Unsigned DDIVU

31 26	25 21	20 16	15 6	5 0
SPECIAL 0 0 0 0 0 0	rs	rt	0 0 0 0 0 0 0 0 0 0 0	DDIVU 0 1 1 1 1 1
6	5	5	10	6

Format:

DDIVU rs, rt

Description:

The contents of general register *rs* are divided by the contents of general register *rt*, treating both operands as unsigned values. No integer overflow exception occurs under any circumstances, and the result of this operation is undefined when the divisor is zero.

This instruction is typically followed by additional instructions to check for a zero divisor.

When the operation completes, 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*.

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.

This operation is only defined for the R4000 operating in 64-bit mode. Execution of this instruction in 32-bit mode causes a reserved instruction exception.

Operation:

```
64    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:

Reserved instruction exception (R4000 in 32-bit mode)

DIV — Divide — DIV

31–26	25–21	20–16	15–6	5–0
SPECIAL 000000	rs	rt	0 00 0000 0000	DIV 011010
6	5	5	10	6

Format:

DIV rs, rt

Description:

The contents of general register *rs* are divided by the contents of general register *rt*, treating both operands as 2's complement values. No overflow exception occurs under any circumstances, and the result of this operation is undefined when the divisor is zero.

In 64-bit mode, the operands must be valid sign-extended, 32-bit values.

This instruction is typically followed by additional instructions to check for a zero divisor and for overflow.

When the operation completes, 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*.

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.

DIV Divide (continued) DIV

Operation:

32	T–2:	LO	← undefined
		HI	← undefined
	T–1:	LO	← undefined
		HI	← undefined
	T:	LO	← GPR[rs] div GPR[rt]
		HI	← GPR[rs] mod GPR[rt]
64	T–2:	LO	← undefined
		HI	← undefined
	T–1:	LO	← undefined
		HI	← undefined
	T:	q	← $GPR[rs]_{31...0}$ div $GPR[rt]_{31...0}$
		r	← $GPR[rs]_{31...0}$ mod $GPR[rt]_{31...0}$
		LO	← $(q_{31})^{32} \| q_{31...0}$
		HI	← $(r_{31})^{32} \| r_{31...0}$

Exceptions:

None

DIVU Divide Unsigned DIVU

31–26	25–21	20–16	15–6	5–0
SPECIAL 0 0 0 0 0 0	rs	rt	0 0 0 0 0 0 0 0 0 0 0	DIVU 0 1 1 0 1 1
6	5	5	10	6

Format:

DIVU rs, rt

Description:

The contents of general register *rs* are divided by the contents of general register *rt,* treating both operands as unsigned values. No integer overflow exception occurs under any circumstances, and the result of this operation is undefined when the divisor is zero.

In 64-bit mode, the operands must be valid sign-extended, 32-bit values.

This instruction is typically followed by additional instructions to check for a zero divisor.

When the operation completes, 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*.

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.

DIVU — Divide Unsigned (continued) — DIVU

Operation:

32	T–2:	LO	$\leftarrow$ undefined
		HI	$\leftarrow$ undefined
	T–1:	LO	$\leftarrow$ undefined
		HI	$\leftarrow$ undefined
	T:	LO	$\leftarrow$ (0 || GPR[rs]) div (0 || GPR[rt])
		HI	$\leftarrow$ (0 || GPR[rs]) mod (0 || GPR[rt])
64	T–2:	LO	$\leftarrow$ undefined
		HI	$\leftarrow$ undefined
	T–1:	LO	$\leftarrow$ undefined
		HI	$\leftarrow$ undefined
	T:	q	$\leftarrow (0 \,\|\|\, GPR[rs]_{31...0})$ div $(0 \,\|\|\, GPR[rt]_{31...0})$
		r	$\leftarrow (0 \,\|\|\, GPR[rs]_{31...0})$ mod $(0 \,\|\|\, GPR[rt]_{31...0})$
		LO	$\leftarrow (q_{31})^{32} \,\|\|\, q_{31...0}$
		HI	$\leftarrow (r_{31})^{32} \,\|\|\, r_{31...0}$

Exceptions:

None

DMFC0 Doubleword Move From System Control Coprocessor DMFC0

31 26	25 21	20 16	15 11	10 0
COP0 0 1 0 0 0 0	DMF 0 0 0 0 1	rt	rd	0 0 0 0 0 0 0 0 0 0 00
6	5	5	5	11

Format:

DMFC0 rt, rd

Description:

The contents of coprocessor register *rd* of the CP0 are loaded into general register *rt.*

This operation is defined for the R4000 operating in 64-bit mode and in 32-bit kernel mode. Execution of this instruction in 32-bit user or supervisor mode causes a reserved instruction exception. All 64-bits of the general register destination are written from the coprocessor register source. The operation of DMFC0 on a 32-bit coprocessor 0 register is undefined.

Operation:

```
64      T:      data ←CPR[0,rd]
        T+1:    GPR[rt] ← data
```

Exceptions:

Coprocessor unusable exception
Reserved instruction exception (R4000 in 32-bit user mode
R4000 in 32-bit supervisor mode)

DMTC0 Doubleword Move To System Control Coprocessor DMTC0

31 26	25 21	20 16	15 11	10 0
COP0 0 1 0 0 0 0	DMT 0 0 1 0 1	rt	rd	0 0 0 0 0 0 0 0 0 0 00
6	5	5	5	11

Format:

DMTC0 rt, rd

Description:

The contents of general register *rt* are loaded into coprocessor register *rd* of the CP0.

This operation is defined for the R4000 operating in 64-bit mode or in 32-bit kernel mode. Execution of this instruction in 32-bit user or supervisor mode causes a reserved instruction exception.

All 64-bits of the coprocessor 0 register are written from the general register source. The operation of DMTC0 on a 32-bit coprocessor 0 register is undefined.

Because the state of the virtual address translation system may be altered by this instruction, the operation of load instructions, store instructions, and TLB operations immediately prior to and after this instruction are undefined.

Operation:

64	T:	data ← GPR[rt]
	T+1:	CPR[0,rd] ← data

Exceptions:

Coprocessor unusable exception (R4000 in 32-bit user mode
R4000 in 32-bit supervisor mode)

DMULT Doubleword Multiply DMULT

31–26	25–21	20–16	15–6	5–0
SPECIAL 0 0 0 0 0 0	rs	rt	0 0 0 0 0 0 0 0 0 0 0	DMULT 0 1 1 1 0 0
6	5	5	10	6

Format:

DMULT rs, rt

Description:

The contents of general registers *rs* and *rt* are multiplied, treating both operands as 2's complement values. No integer overflow exception occurs under any circumstances.

When the operation completes, 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*.

If either of the two preceding instructions is MFHI or MFLO, the results of these instructions are undefined. Correct operation requires separating reads of *HI* or *LO* from writes by a minimum of two other instructions.

This operation is only defined for the R4000 operating in 64-bit mode. Execution of this instruction in 32-bit mode causes a reserved instruction exception.

Operation:

64	T–2:	LO	← undefined
		HI	← undefined
	T–1:	LO	← undefined
		HI	← undefined
	T:	t	← GPR[rs] * GPR[rt]
		LO	← $t_{63...0}$
		H I	← $t_{127...64}$

Exceptions:

Reserved instruction exception (R4000 in 32-bit mode)

DMULTU Doubleword Multiply Unsigned DMULTU

31–26	25–21	20–16	15–6	5–0
SPECIAL 0 0 0 0 0 0	rs	rt	0 0 0 0 0 0 0 0 0 0 0	DMULTU 0 1 1 1 0 1
6	5	5	10	6

Format:

DMULTU rs, rt

Description:

The contents of general register *rs* and the contents of general register *rt* are multiplied, treating both operands as unsigned values. No overflow exception occurs under any circumstances.

When the operation completes, 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*.

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

This operation is only defined for the R4000 operating in 64-bit mode. Execution of this instruction in 32-bit mode causes a reserved instruction exception.

Operation:

```
64   T–2:   LO ← undefined
            HI ← undefined
     T–1:   LO ← undefined
            HI ← undefined
     T:     t ← (0 || GPR[rs]) * (0 || GPR[rt])
            LO ← t63...0
            HI ←t127...64
```

Exceptions:

Reserved instruction exception (R4000 in 32-bit mode)

DSLL Doubleword Shift Left Logical DSLL

31 26	25 21	20 16	15 11	10 6	5 0
SPECIAL 0 0 0 0 0 0	0 0 0 0 0 0	rt	rd	sa	DSLL 1 1 1 0 0 0
6	5	5	5	5	6

Format:

DSLL rd, rt, sa

Description:

The contents of general register *rt* are shifted left by *sa* bits, inserting zeros into the low-order bits. The result is placed in register *rd.*

Operation:

64	T:	$s \leftarrow 0 \mid\mid sa$
		$GPR[rd] \leftarrow GPR[rt]_{(63-s)...0} \mid\mid 0^s$

Exceptions:

Reserved instruction exception (R4000 in 32-bit mode)

DSLLV — Doubleword Shift Left Logical Variable — DSLLV

31–26	25–21	20–16	15–11	10–6	5–0
SPECIAL 000000	rs	rt	rd	0 00000	DSLLV 010100
6	5	5	5	5	6

Format:

DSLLV rd, rt, rs

Description:

The contents of general register *rt* are shifted left by the number of bits specified by the low-order six bits contained in general register *rs*, inserting zeros into the low-order bits. The result is placed in register *rd*.

This operation is only defined for the R4000 operating in 64-bit mode. Execution of this instruction in 32-bit mode causes a reserved instruction exception.

Operation:

64	T:	$s \leftarrow GPR[rs]_{5...0}$
		$GPR[rd] \leftarrow GPR[rt]_{(63-s)...0} \parallel 0^s$

Exceptions:

Reserved instruction exception (R4000 in 32-bit mode)

DSLL32 — Doubleword Shift Left Logical + 32 — DSLL32

31–26	25–21	20–16	15–11	10–6	5–0
SPECIAL 0 0 0 0 0 0	0 0 0 0 0 0	rt	rd	sa	DSLL32 1 1 1 1 0 0
6	5	5	5	5	6

Format:

DSLL32 rd, rt, sa

Description:

The contents of general register *rt* are shifted left by *32+sa* bits, inserting zeros into the low-order bits. The result is placed in register *rd*.

This operation is only defined for the R4000 operating in 64-bit mode. Execution of this instruction in 32-bit mode causes a reserved instruction exception.

Operation:

64	T:	$s \leftarrow 1 \parallel sa$
		$GPR[rd] \leftarrow GPR[rt]_{(63-s)...0} \parallel 0^s$

Exceptions:

Reserved instruction exception (R4000 in 32-bit mode)

DSRA — Doubleword Shift Right Arithmetic — DSRA

31–26	25–21	20–16	15–11	10–6	5–0
SPECIAL 0 0 0 0 0 0	0 0 0 0 0 0	rt	rd	sa	DSRA 1 1 1 0 1 1
6	5	5	5	5	6

Format:

DSRA rd, rt, sa

Description:

The contents of general register *rt* are shifted right by *sa* bits, sign-extending the high-order bits. The result is placed in register *rd*.

This operation is only defined for the R4000 operating in 64-bit mode. Execution of this instruction in 32-bit mode causes a reserved instruction exception.

Operation:

64	T:	$s \leftarrow 0 \mid\mid sa$
		$GPR[rd] \leftarrow (GPR[rt]_{63})^{s} \mid\mid GPR[rt]_{63...s}$

Exceptions:

Reserved instruction exception (R4000 in 32-bit mode)

DSRAV — Doubleword Shift Right Arithmetic Variable — DSRAV

31 … 26	25 … 21	20 … 16	15 … 11	10 … 6	5 … 0
SPECIAL 0 0 0 0 0 0	rs	rt	rd	0 0 0 0 0 0	DSRAV 0 1 0 1 1 1
6	5	5	5	5	6

Format:

DSRAV rd, rt, rs

Description:

The contents of general register *rt* are shifted right by the number of bits specified by the low-order six bits of general register *rs*, sign-extending the high-order bits. The result is placed in register *rd*.

This operation is only defined for the R4000 operating in 64-bit mode. Execution of this instruction in 32-bit mode causes a reserved instruction exception.

Operation:

64	T:	$s \leftarrow GPR[rs]_{5...0}$
		$GPR[rd] \leftarrow (GPR[rt]_{63})^{s} \parallel GPR[rt]_{63...s}$

Exceptions:

Reserved instruction exception (R4000 in 32-bit mode)

DSRA32 Doubleword Shift Right Arithmetic + 32 DSRA32

31 26	25 21	20 16	15 11	10 6	5 0
SPECIAL 0 0 0 0 0 0	0 0 0 0 0 0	rt	rd	sa	DSRA32 1 1 1 1 1 1
6	5	5	5	5	6

Format:

DSRA32 rd, rt, sa

Description:

The contents of general register *rt* are shifted right by *32+sa* bits, sign-extending the high-order bits. The result is placed in register *rd*.

This operation is only defined for the R4000 operating in 64-bit mode. Execution of this instruction in 32-bit mode causes a reserved instruction exception.

Operation:

64	T:	$s \leftarrow 1 \mid\mid sa$
		$GPR[rd] \leftarrow (GPR[rt]_{63})^{s} \mid\mid GPR[rt]_{63...s}$

Exceptions:

Reserved instruction exception (R4000 in 32-bit mode)

DSRL — Doubleword Shift Right Logical — DSRL

31–26	25–21	20–16	15–11	10–6	5–0
SPECIAL 0 0 0 0 0 0	0 0 0 0 0 0	rt	rd	sa	DSRL 1 1 1 0 1 0
6	5	5	5	5	6

Format:

DSRL rd, rt, sa

Description:

The contents of general register *rt* are shifted right by *sa* bits, inserting zeros into the high-order bits. The result is placed in register *rd*.

This operation is only defined for the R4000 operating in 64-bit mode. Execution of this instruction in 32-bit mode causes a reserved instruction exception.

Operation:

64	T:	$s \leftarrow 0 \mid\mid sa$
		$GPR[rd] \leftarrow 0^{s} \mid\mid GPR[rt]_{63...s}$

Exceptions:

Reserved instruction exception (R4000 in 32-bit mode)

DSRLV — Doubleword Shift Right Logical Variable — DSRLV

31–26	25–21	20–16	15–11	10–6	5–0
SPECIAL 0 0 0 0 0 0	rs	rt	rd	0 0 0 0 0 0	DSRLV 0 1 0 1 1 0
6	5	5	5	5	6

Format:

DSRLV rd, rt, rs

Description:

The contents of general register *rt* are shifted right by the number of bits specified by the low-order six bits of general register *rs,* inserting zeros into the high-order bits. The result is placed in register *rd*.

This operation is only defined for the R4000 operating in 64-bit mode. Execution of this instruction in 32-bit mode causes a reserved instruction exception.

Operation:

64	T:	$s \leftarrow GPR[rs]_{5...0}$
		$GPR[rd] \leftarrow 0^{s} \mid\mid GPR[rt]_{63...s}$

Exceptions:

Reserved instruction exception (R4000 in 32-bit mode)

DSRL32 Doubleword Shift Right Logical + 32 DSRL32

31 26	25 21	20 16	15 11	10 6	5 0
SPECIAL 0 0 0 0 0 0	0 0 0 0 0 0	rt	rd	sa	DSRL32 1 1 1 1 1 0
6	5	5	5	5	6

Format:

DSRL32 rd, rt, sa

Description:

The contents of general register *rt* are shifted right by *32+sa* bits, inserting zeros into the high-order bits. The result is placed in register *rd*.

This operation is only defined for the R4000 operating in 64-bit mode. Execution of this instruction in 32-bit mode causes a reserved instruction exception.

Operation:

64	T:	$s \leftarrow 1 \,\|\|\, sa$
		$GPR[rd] \leftarrow 0^{s} \,\|\|\, GPR[rt]_{63...s}$

Exceptions:

Reserved instruction exception (R4000 in 32-bit mode)

DSUB Doubleword Subtract DSUB

31–26	25–21	20–16	15–11	10–6	5–0
SPECIAL 0 0 0 0 0 0	rs	rt	rd	0 0 0 0 0 0	DSUB 1 0 1 1 1 0
6	5	5	5	5	6

Format:

DSUB rd, rs, rt

Description:

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

The only difference between this instruction and the DSUBU instruction is that DSUBU never traps on overflow.

An integer overflow exception takes place if the carries out of bits 62 and 63 differ (2's complement overflow). The destination register *rd* is not modified when an integer overflow exception occurs.

This operation is only defined for the R4000 operating in 64-bit mode. Execution of this instruction in 32-bit mode causes a reserved instruction exception.

Operation:

64	T:	GPR[rd] ← GPR[rs] – GPR[rt]

Exceptions:

Integer overflow exception
Reserved instruction exception (R4000 in 32-bit mode)

DSUBU Doubleword Subtract Unsigned DSUBU

31–26	25–21	20–16	15–11	10–6	5–0
SPECIAL 0 0 0 0 0 0	rs	rt	rd	0 0 0 0 0 0	DSUBU 1 0 1 1 1 1
6	5	5	5	5	6

Format:

DSUBU rd, rs, rt

Description:

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

The only difference between this instruction and the DSUB instruction is that DSUBU never traps on overflow. No integer overflow exception occurs under any circumstances.

This operation is only defined for the R4000 operating in 64-bit mode. Execution of this instruction in 32-bit mode causes a reserved instruction exception.

Operation:

```
64        T:    GPR[rd] ← GPR[rs] – GPR[rt]
```

Exceptions:

Reserved instruction exception (R4000 in 32-bit mode)

ERET Exception Return ERET

31 … 26	25	24 … 6	5 … 0
COP0 0 1 0 0 0 0	CO 1	0 000 0000 0000 0000 0000	ERET 0 1 1 0 0 0
6	1	19	6

Format:

ERET

Description:

ERET is the R4000 instruction for returning from an interrupt, exception, or error trap. Unlike a branch or jump instruction, ERET does not execute the next instruction.

ERET must not itself be placed in a branch delay slot.

If the processor is servicing an error trap ($SR_2 = 1$), then load the PC from the *ErrorEPC* and clear the *ERL* bit of the *Status* register (SR_2). Otherwise ($SR_2 = 0$), load the PC from the *EPC*, and clear the *EXL* bit of the *Status* register (SR_1).

An ERET executed between a LL and SC also causes the SC to fail.

Operation:

```
32, 64    T:  if SR2 = 1 then
                  PC ← ErrorEPC
                  SR ← SR31...3 || 0 || SR1...0
              else
                  PC ← EPC
                  SR ← SR31...2 || 0 || SR0
              endif
              LLbit ← 0
```

Exceptions:

Coprocessor unusable exception

J — Jump — J

31 — 26	25 — 0
J 0 0 0 0 1 0	target
6	26

Format:

J target

Description:

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

Operation:

32	T:	temp ← target
	T+1:	PC ← $PC_{31...28}$ || temp || 0^2
64	T:	temp ← target
	T+1:	PC ← $PC_{63...28}$ || temp || 0^2

Exceptions:

None

JAL — Jump And Link — JAL

31 … 26	25 … 0
JAL 000011	target
6	26

Format:

JAL target

Description:

The 26-bit target address is shifted left two bits and combined with the high-order bits of the address of the delay slot. The program unconditionally jumps to this 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:

32	T:	temp $\leftarrow$ target
		GPR[31] $\leftarrow$ PC + 8
	T+1:	PC $\leftarrow$ PC $_{31...28}$ || temp || 0^2
64	T:	temp $\leftarrow$ target
		GPR[31] $\leftarrow$ PC + 8
	T+1:	PC $\leftarrow$ PC $_{63...28}$ || temp || 0^2

Exceptions:

None

JALR Jump And Link Register JALR

31–26	25–21	20–16	15–11	10–6	5–0
SPECIAL 0 0 0 0 0 0	rs	0 0 0 0 0 0	rd	0 0 0 0 0 0	JALR 0 0 1 0 0 1
6	5	5	5	5	6

Format:

JALR rs
JALR rd, rs

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, and 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:

32, 64	T:	temp ← GPR [rs]
		GPR[rd] ← PC + 8
	T+1:	PC ← temp

Exceptions:

None

JR Jump Register JR

31 26	25 21	20 6	5 0
SPECIAL 000000	rs	0 000 0000 0000 0000	JR 001000
6	5	15	6

Format:

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:

32, 64	T:	temp ← GPR[rs]
	T+1:	PC ← temp

Exceptions:

None

LB — Load Byte — LB

31 … 26	25 … 21	20 … 16	15 … 0
LB 1 0 0 0 0 0	base	rt	offset
6	5	5	16

Format:

LB rt, offset(base)

Description:

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

Operation:

32 T: $vAddr \leftarrow ((offset_{15})^{16} \,||\, offset_{15...0}) + GPR[base]$

$(pAddr, uncached) \leftarrow AddressTranslation\ (vAddr, DATA)$

$pAddr \leftarrow pAddr_{PSIZE-1...3} \,||\, (pAddr_{2...0}\ xor\ ReverseEndian^{3})$

$mem \leftarrow LoadMemory\ (uncached, BYTE, pAddr, vAddr, DATA)$

$byte \leftarrow vAddr_{2...0}\ xor\ BigEndianCPU^{3}$

$GPR[rt] \leftarrow (mem_{7+8*byte})^{24} \,||\, mem_{7+8*byte...8*byte}$

64 T: $vAddr \leftarrow ((offset_{15})^{48} \,||\, offset_{15...0}) + GPR[base]$

$(pAddr, uncached) \leftarrow AddressTranslation\ (vAddr, DATA)$

$pAddr \leftarrow pAddr_{PSIZE-1...3} \,||\, (pAddr_{2...0}\ xor\ ReverseEndian^{3})$

$mem \leftarrow LoadMemory\ (uncached, BYTE, pAddr, vAddr, DATA)$

$byte \leftarrow vAddr_{2...0}\ xor\ BigEndianCPU^{3}$

$GPR[rt] \leftarrow (mem_{7+8*byte})^{56} \,||\, mem_{7+8*byte...8*byte}$

Exceptions:

TLB refill exception
TLB invalid exception
Bus error exception
Address error exception

LBU — Load Byte Unsigned — LBU

31 … 26	25 … 21	20 … 16	15 … 0
LBU 1 0 0 1 0 0	base	rt	offset
6	5	5	16

Format:

LBU rt, offset(base)

Description:

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

Operation:

32 T: $vAddr \leftarrow ((offset_{15})^{16} \,||\, offset_{15...0}) + GPR[base]$
$(pAddr, uncached) \leftarrow AddressTranslation\ (vAddr, DATA)$
$pAddr \leftarrow pAddr_{PSIZE-1...3} \,||\, (pAddr_{2...0}\ xor\ ReverseEndian^3)$
$mem \leftarrow LoadMemory\ (uncached, BYTE, pAddr, vAddr, DATA)$
$byte \leftarrow vAddr_{2...0}\ xor\ BigEndianCPU^3$
$GPR[rt] \leftarrow 0^{24} \,||\, mem_{7+8*byte...8*byte}$

64 T: $vAddr \leftarrow ((offset_{15})^{48} \,||\, offset_{15...0}) + GPR[base]$
$(pAddr, uncached) \leftarrow AddressTranslation\ (vAddr, DATA)$
$pAddr \leftarrow pAddr_{PSIZE-1...3} \,||\, (pAddr_{2...0}\ xor\ ReverseEndian^3)$
$mem \leftarrow LoadMemory\ (uncached, BYTE, pAddr, vAddr, DATA)$
$byte \leftarrow vAddr_{2...0}\ xor\ BigEndianCPU^3$
$GPR[rt] \leftarrow 0^{56} \,||\, mem_{7+8*byte...8*byte}$

Exceptions:

TLB refill exception
TLB invalid exception
Bus error exception
Address error exception

LD — Load Doubleword — LD

31 … 26	25 … 21	20 … 16	15 … 0
LD 1 1 0 1 1 1	base	rt	offset
6	5	5	16

Format:

LD rt, offset(base)

Description:

The 16-bit *offset* is sign-extended and added to the contents of general register *base* to form a virtual address. The contents of the 64-bit doubleword at the memory location specified by the effective address are loaded into general register *rt*.

If any of the three least-significant bits of the effective address are non-zero, an address error exception occurs.

This operation is only defined for the R4000 operating in 64-bit mode. Execution of this instruction in 32-bit mode causes a reserved instruction exception.

Operation:

64	T:	vAddr $\leftarrow$ ((offset$_{15}$)^{48} || offset$_{15...0}$) + GPR[base]
		(pAddr, uncached) $\leftarrow$ AddressTranslation (vAddr, DATA)
		mem $\leftarrow$ LoadMemory (uncached, DOUBLEWORD, pAddr, vAddr, DATA)
		GPR[rt] $\leftarrow$ mem

Exceptions:

TLB refill exception
TLB invalid exception
Bus error exception
Address error exception
Reserved instruction exception (R4000 in 32-bit user mode
R4000 in 32-bit supervisor mode)

LDCz Load Doubleword To Coprocessor LDCz

31 26	25 21	20 16	15 0
LDCz 1 1 0 1 x x*	base	rt	offset
6	5	5	16

Format:

LDCz rt, offset(base)

Description:

The 16-bit *offset* is sign-extended and added to the contents of general register *base* to form a virtual address. The processor reads a doubleword from the addressed memory location and makes the data available to coprocessor unit *z*. The manner in which each coprocessor uses the data is defined by the individual coprocessor specifications.

If any of the three least-significant bits of the effective address are non-zero, an address error exception takes place.

This instruction is not valid for use with CP0.

This instruction is undefined when the least-significant bit of the *rt* field is non-zero.

*See the table “Opcode Bit Encoding” on next page, or “CPU Instruction Opcode Bit Encoding” at the end of Appendix A.

LDCz — Load Doubleword To Coprocessor (continued) — LDCz

Operation:

32	T:	vAddr ← $((\text{offset}_{15})^{16} \,\|\|\, \text{offset}_{15...0})$ + GPR[base] (pAddr, uncached) ← AddressTranslation (vAddr, DATA) mem ← LoadMemory (uncached, DOUBLEWORD, pAddr, vAddr, DATA) COPzLD (rt, mem)
64	T:	vAddr ← $((\text{offset}_{15})^{48} \,\|\|\, \text{offset}_{15...0})$ + GPR[base] (pAddr, uncached) ← AddressTranslation (vAddr, DATA) mem ← LoadMemory (uncached, DOUBLEWORD, pAddr, vAddr, DATA) COPzLD (rt, mem)

Exceptions:

TLB refill exception
TLB invalid exception
Bus error exception
Address error exception
Coprocessor unusable exception

Opcode Bit Encoding:

LDCz

Bit #	31	30	29	28	27	26	0
LDC1	1	1	0	1	0	1	

Bit #	31	30	29	28	27	26	0
LDC2	1	1	0	1	1	0	

Bits 31–28: Opcode; bits 27–26: Coprocessor Unit Number

LDL — Load Doubleword Left — LDL

31 26	25 21	20 16	15 0
LDL 0 1 1 0 1 0	base	rt	offset
6	5	5	16

Format:

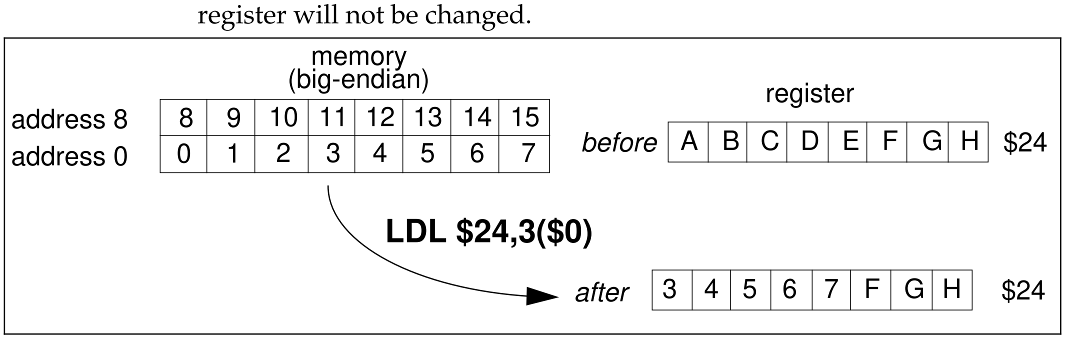
LDL rt, offset(base)

Description:

This instruction can be used in combination with the LDR instruction to load a register with eight consecutive bytes from memory, when the bytes cross a doubleword boundary. LDL loads the left portion of the register with the appropriate part of the high-order doubleword; LDR loads the right portion of the register with the appropriate part of the low-order doubleword.

The LDL instruction adds its sign-extended 16-bit *offset* to the contents of general register *base* to form a virtual address which can specify an arbitrary byte. It reads bytes only from the doubleword in memory which contains the specified starting byte. From one to eight bytes will be loaded, depending on the starting byte specified.

Conceptually, it starts at the specified byte in memory and loads that byte into the high-order (left-most) byte of the register; then it loads bytes from memory into the register until it reaches the low-order byte of the doubleword in memory. The least-significant (right-most) byte(s) of the register will not be changed.



LDL — Load Doubleword Left (continued) — LDL

The contents of general register *rt* are internally bypassed within the processor so that no NOP is needed between an immediately preceding load instruction which specifies register *rt* and a following LDL (or LDR) instruction which also specifies register *rt*.

No address exceptions due to alignment are possible.

This operation is only defined for the R4000 operating in 64-bit mode. Execution of this instruction in 32-bit mode causes a reserved instruction exception.

Operation:

64	T:	$vAddr \leftarrow ((offset_{15})^{48} \mathbin{\|\|} offset_{15...0}) + GPR[base]$
		$(pAddr, uncached) \leftarrow AddressTranslation\ (vAddr, DATA)$
		$pAddr \leftarrow pAddr_{PSIZE-1...3} \mathbin{\|\|} (pAddr_{2...0}\ xor\ ReverseEndian^3)$
		if BigEndianMem = 0 then
		$\quad pAddr \leftarrow pAddr_{PSIZE-1...3} \mathbin{\|\|} 0^3$
		endif
		$byte \leftarrow vAddr_{2...0}\ xor\ BigEndianCPU^3$
		$mem \leftarrow LoadMemory\ (uncached, byte, pAddr, vAddr, DATA)$
		$GPR[rt] \leftarrow mem_{7+8*byte...0} \mathbin{\|\|} GPR[rt]_{55-8*byte...0}$

LDL — Load Doubleword Left (continued) — LDL

Given a doubleword in a register and a doubleword in memory, the operation of LDL is as follows:

LDL

Register	A	B	C	D	E	F	G	H
Memory	I	J	K	L	M	N	O	P

$vAddr_{2..0}$	BigEndianCPU = 0				BigEndianCPU = 1			
	destination	type	offset		destination	type	offset	
			LEM	BEM			LEM	BEM
0	P B C D E F G H	0	0	7	I J K L M N O P	7	0	0
1	O P C D E F G H	1	0	6	J K L M N O P H	6	0	1
2	N O P D E F G H	2	0	5	K L M N O P G H	5	0	2
3	M N O P E F G P	3	0	4	L M N O P F G H	4	0	3
4	L M N O P F G H	4	0	3	M N O P E F G H	3	0	4
5	K L M N O P G H	5	0	2	N O P D E F G H	2	0	5
6	J K L M N O P H	6	0	1	O P C D E F G H	1	0	6
7	I J K L M N O P	7	0	0	P B C D E F G H	0	0	7

LEM Little-endian memory (BigEndianMem = 0)
BEM BigEndianMem = 1
Type AccessType (see Table 2-1) sent to memory
Offset $pAddr_{2...0}$ sent to memory

Exceptions:

TLB refill exception
TLB invalid exception
Bus error exception
Address error exception
Reserved instruction exception (R4000 in 32-bit mode)

LDR — Load Doubleword Right — LDR

31 … 26	25 … 21	20 … 16	15 … 0
LDR 0 1 1 0 1 1	base	rt	offset
6	5	5	16

Format:

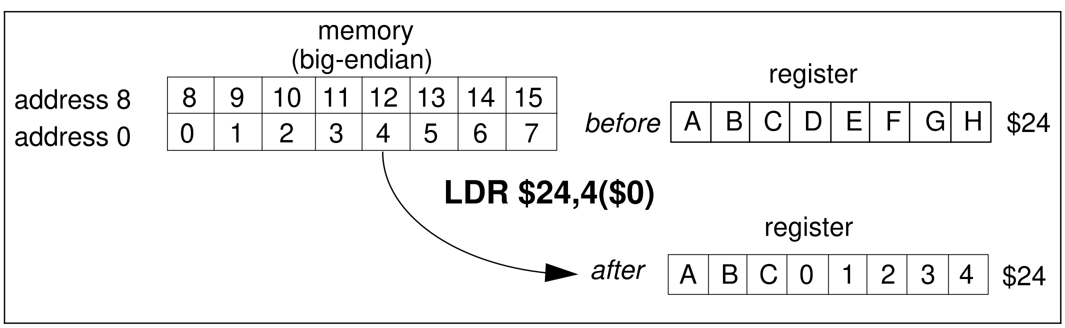
LDR rt, offset(base)

Description:

This instruction can be used in combination with the LDL instruction to load a register with eight consecutive bytes from memory, when the bytes cross a doubleword boundary. LDR loads the right portion of the register with the appropriate part of the low-order doubleword; LDL loads the left portion of the register with the appropriate part of the high-order doubleword.

The LDR instruction adds its sign-extended 16-bit *offset* to the contents of general register *base* to form a virtual address which can specify an arbitrary byte. It reads bytes only from the doubleword in memory which contains the specified starting byte. From one to eight bytes will be loaded, depending on the starting byte specified.

Conceptually, it starts at the specified byte in memory and loads that byte into the low-order (right-most) byte of the register; then it loads bytes from memory into the register until it reaches the high-order byte of the doubleword in memory. The most significant (left-most) byte(s) of the register will not be changed.



LDR — Load Doubleword Right (continued) — LDR

The contents of general register *rt* are internally bypassed within the processor so that no NOP is needed between an immediately preceding load instruction which specifies register *rt* and a following LDR (or LDL) instruction which also specifies register *rt*.

No address exceptions due to alignment are possible.

This operation is only defined for the R4000 operating in 64-bit mode. Execution of this instruction in 32-bit mode causes a reserved instruction exception.

Operation:

64 T: $vAddr \leftarrow ((offset_{15})^{48} \,||\, offset_{15...0}) + GPR[base]$

$(pAddr, uncached) \leftarrow AddressTranslation\ (vAddr, DATA)$

$pAddr \leftarrow pAddr_{PSIZE-1...3} \,||\, (pAddr_{2...0}\ \text{xor}\ ReverseEndian^3)$

if BigEndianMem = 1 then

$\quad pAddr \leftarrow pAddr_{31...3} \,||\, 0^3$

endif

$byte \leftarrow vAddr_{2...0}\ \text{xor}\ BigEndianCPU^3$

$mem \leftarrow LoadMemory\ (uncached, byte, pAddr, vAddr, DATA)$

$GPR[rt] \leftarrow GPR[rt]_{63...64-8*byte} \,||\, mem_{63...8*byte}$

LDR — Load Doubleword Right (continued) — LDR

Given a doubleword in a register and a doubleword in memory, the operation of LDR is as follows:

LDR

Register	A	B	C	D	E	F	G	H
Memory	I	J	K	L	M	N	O	P

$vAddr_{2..0}$	BigEndianCPU = 0				BigEndianCPU = 1			
	destination	type	offset		destination	type	offset	
			LEM	BEM			LEM	BEM
0	I J K L M N O P	7	0	0	A B C D E F G I	0	7	0
1	A I J K L M N O	6	1	0	A B C D E F I J	1	6	0
2	A B I J K L M N	5	2	0	A B C D E I J K	2	5	0
3	A B C I J K L M	4	3	0	A B C D I J K L	3	4	0
4	A B C D I J K L	3	4	0	A B C I J K L M	4	3	0
5	A B C D E I J K	2	5	0	A B I J K L M N	5	2	0
6	A B C D E F I J	1	6	0	A I J K L M N O	6	1	0
7	A B C D E F G I	0	7	0	I J K L M N O P	7	0	0

LEM	Little-endian memory (BigEndianMem = 0)
BEM	BigEndianMem = 1
Type	AccessType (see Table 2-1) sent to memory
Offset	$pAddr_{2...0}$ sent to memory

Exceptions:

TLB refill exception
TLB invalid exception
Bus error exception
Address error exception
Reserved instruction exception (R4000 in 32-bit mode)

LH Load Halfword LH

31 … 26	25 … 21	20 … 16	15 … 0
LH 1 0 0 0 0 1	base	rt	offset
6	5	5	16

Format:

LH rt, offset(base)

Description:

The 16-bit *offset* is sign-extended and added to the contents of general register *base* to form a virtual 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.

Operation:

32 T: $vAddr \leftarrow ((offset_{15})^{16} \| offset_{15...0}) + GPR[base]$
$(pAddr, uncached) \leftarrow AddressTranslation\ (vAddr, DATA)$
$pAddr \leftarrow pAddr_{PSIZE-1...3} \| (pAddr_{2...0}\ xor\ (ReverseEndian \| 0))$
$mem \leftarrow LoadMemory\ (uncached, HALFWORD, pAddr, vAddr, DATA)$
$byte \leftarrow vAddr_{2...0}\ xor\ (BigEndianCPU^2 \| 0)$
$GPR[rt] \leftarrow (mem_{15+8*byte})^{16} \| mem_{15+8*byte...8*byte}$

64 T: $vAddr \leftarrow ((offset_{15})^{48} \| offset_{15...0}) + GPR[base]$
$(pAddr, uncached) \leftarrow AddressTranslation\ (vAddr, DATA)$
$pAddr \leftarrow pAddr_{PSIZE-1...3} \| (pAddr_{2...0}\ xor\ (ReverseEndian \| 0))$
$mem \leftarrow LoadMemory\ (uncached, HALFWORD, pAddr, vAddr, DATA)$
$byte \leftarrow vAddr_{2...0}\ xor\ (BigEndianCPU^2 \| 0)$
$GPR[rt] \leftarrow (mem_{15+8*byte})^{48} \| mem_{15+8*byte...8*byte}$

Exceptions:

TLB refill exception
TLB invalid exception
Bus error exception
Address error exception

LHU — Load Halfword Unsigned — LHU

31–26	25–21	20–16	15–0
LHU 100101	base	rt	offset
6	5	5	16

Format:

LHU rt, offset(base)

Description:

The 16-bit *offset* is sign-extended and added to the contents of general register *base* to form a virtual 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.

Operation:

32	T:	$vAddr \leftarrow ((offset_{15})^{16} \parallel offset_{15...0}) + GPR[base]$
		$(pAddr, uncached) \leftarrow AddressTranslation\ (vAddr, DATA)$
		$pAddr \leftarrow pAddr_{PSIZE-1...3} \parallel (pAddr_{2...0}\ xor\ (ReverseEndian^{2} \parallel 0))$
		$mem \leftarrow LoadMemory\ (uncached, HALFWORD, pAddr, vAddr, DATA)$
		$byte \leftarrow vAddr_{2...0}\ xor\ (BigEndianCPU^{2} \parallel 0)$
		$GPR[rt] \leftarrow 0^{16} \parallel mem_{15+8*byte...8*byte}$
64	T:	$vAddr \leftarrow ((offset_{15})^{48} \parallel offset_{15...0}) + GPR[base]$
		$(pAddr, uncached) \leftarrow AddressTranslation\ (vAddr, DATA)$
		$pAddr \leftarrow pAddr_{PSIZE-1...3} \parallel (pAddr_{2...0}\ xor\ (ReverseEndian^{2} \parallel 0))$
		$mem \leftarrow LoadMemory\ (uncached, HALFWORD, pAddr, vAddr, DATA)$
		$byte \leftarrow vAddr_{2...0}\ xor\ (BigEndianCPU^{2} \parallel 0)$
		$GPR[rt] \leftarrow 0^{48} \parallel mem_{15+8*byte...8*byte}$

Exceptions:

TLB refill exception
TLB invalid exception
Bus Error exception
Address error exception

LL Load Linked LL

31–26	25–21	20–16	15–0
LL 1 1 0 0 0 0	base	rt	offset
6	5	5	16

Format:

LL rt, offset(base)

Description:

The 16-bit *offset* is sign-extended and added to the contents of general register *base* to form a virtual address. The contents of the word at the memory location specified by the effective address are loaded into general register *rt*. In 64-bit mode, the loaded word is sign-extended.

The processor begins checking the accessed word for modification by other processor and devices.

Load Linked and Store Conditional can be used to atomically update memory locations as shown:

```
L1:
    LL      T1, (T0)
    ADD     T2, T1, 1
    SC      T2, (T0)
    BEQ     T2, 0, L1
    NOP
```

This atomically increments the word addressed by T0. Changing the ADD to an OR changes this to an atomic bit set. This instruction is available in User mode, and it is not necessary for CP0 to be enabled.

The operation of LL is undefined if the addressed location is uncached and, for synchronization between multiple processors, the operation of LL is undefined if the addressed location is noncoherent. A cache miss that occurs between LL and SC may cause SC to fail, so no load or store operation should occur between LL and SC, otherwise the SC may never be successful. Exceptions also cause SC to fail, so persistent exceptions must be avoided. If either of the two least-significant bits of the effective address are non-zero, an address error exception takes place.

LL — Load Linked (continued) — LL

Operation:

32	T:	$vAddr \leftarrow ((offset_{15})^{16} \| \| offset_{15...0}) + GPR[base]$
		$(pAddr, uncached) \leftarrow AddressTranslation\ (vAddr, DATA)$
		$pAddr \leftarrow pAddr_{PSIZE-1...3} \| \| (pAddr_{2...0}\ xor\ (ReverseEndian \| \| 0^2))$
		$mem \leftarrow LoadMemory\ (uncached, WORD, pAddr, vAddr, DATA)$
		$byte \leftarrow vAddr_{2...0}\ xor\ (BigEndianCPU \| \| 0^2)$
		$GPR[rt] \leftarrow mem_{31+8*byte...8*byte}$
		$LLbit \leftarrow 1$
64	T:	$vAddr \leftarrow ((offset_{15})^{48} \| \| offset_{15...0}) + GPR[base]$
		$(pAddr, uncached) \leftarrow AddressTranslation\ (vAddr, DATA)$
		$pAddr \leftarrow pAddr_{PSIZE-1...3} \| \| (pAddr_{2...0}\ xor\ (ReverseEndian \| \| 0^2))$
		$mem \leftarrow LoadMemory\ (uncached, WORD, pAddr, vAddr, DATA)$
		$byte \leftarrow vAddr_{2...0}\ xor\ (BigEndianCPU \| \| 0^2)$
		$GPR[rt] \leftarrow (mem_{31+8*byte})^{32} \| \| mem_{31+8*byte...8*byte}$
		$LLbit \leftarrow 1$

Exceptions:

TLB refill exception
TLB invalid exception
Bus error exception
Address error exception

LLD Load Linked Doubleword LLD

31 26	25 21	20 16	15 0
LLD 1 1 0 1 0 0	base	rt	offset
6	5	5	16

Format:

LLD rt, offset(base)

Description:

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

The processor begins checking the accessed word for modification by other processor and devices.

Load Linked Doubleword and Store Conditional Doubleword can be used to atomically update memory locations:

```
L1:
    LLD     T1, (T0)
    ADD     T2, T1, 1
    SCD     T2, (T0)
    BEQ     T2, 0, L1
    NOP
```

This atomically increments the word addressed by T0. Changing the ADD to an OR changes this to an atomic bit set.

LLD — Load Linked Doubleword (continued) — LLD

The operation of LLD is undefined if the addressed location is uncached and, for synchronization between multiple processors, the operation of LLD is undefined if the addressed location is noncoherent. A cache miss that occurs between LLD and SCD may cause SCD to fail, so no load or store operation should occur between LLD and SCD, otherwise the SCD may never be successful. Exceptions also cause SCD to fail, so persistent exceptions must be avoided.

This instruction is available in User mode, and it is not necessary for CP0 to be enabled.

If any of the three least-significant bits of the effective address are non-zero, an address error exception takes place.

This operation is only defined for the R4000 operating in 64-bit mode. Execution of this instruction in 32-bit mode causes a reserved instruction exception.

Operation:

64	T:	vAddr ← $((\text{offset}_{15})^{48} \parallel \text{offset}_{15...0})$ + GPR[base]
		(pAddr, uncached) ← AddressTranslation (vAddr, DATA)
		mem ← LoadMemory (uncached, DOUBLEWORD, pAddr, vAddr, DATA)
		GPR[rt] ← mem
		LLbit ← 1

Exceptions:

TLB refill exception
TLB invalid exception
Bus error exception
Address error exception
Reserved instruction exception (R4000 in 32-bit mode)

LUI — Load Upper Immediate — LUI

31 … 26	25 … 21	20 … 16	15 … 0
LUI 0 0 1 1 1 1	0 0 0 0 0 0	rt	immediate
6	5	5	16

Format:

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*. In 64-bit mode, the loaded word is sign-extended.

Operation:

32	T:	$GPR[rt] \leftarrow immediate \,\|\|\, 0^{16}$
64	T:	$GPR[rt] \leftarrow (immediate_{15})^{32} \,\|\|\, immediate \,\|\|\, 0^{16}$

Exceptions:

None

LW — Load Word — LW

31 … 26	25 … 21	20 … 16	15 … 0
LW 1 0 0 0 1 1	base	rt	offset
6	5	5	16

Format:

LW rt, offset(base)

Description:

The 16-bit *offset* is sign-extended and added to the contents of general register *base* to form a virtual address. The contents of the word at the memory location specified by the effective address are loaded into general register *rt*. In 64-bit mode, the loaded word is sign-extended. If either of the two least-significant bits of the effective address is non-zero, an address error exception occurs.

Operation:

32 T: $vAddr \leftarrow ((offset_{15})^{16} \,||\, offset_{15...0}) + GPR[base]$
$(pAddr, uncached) \leftarrow AddressTranslation\ (vAddr, DATA)$
$pAddr \leftarrow pAddr_{PSIZE-1...3} \,||\, (pAddr_{2...0}\ xor\ (ReverseEndian \,||\, 0^2))$
$mem \leftarrow LoadMemory\ (uncached, WORD, pAddr, vAddr, DATA)$
$byte \leftarrow vAddr_{2...0}\ xor\ (BigEndianCPU \,||\, 0^2)$
$GPR[rt] \leftarrow mem_{31+8*byte...8*byte}$

64 T: $vAddr \leftarrow ((offset_{15})^{48} \,||\, offset_{15...0}) + GPR[base]$
$(pAddr, uncached) \leftarrow AddressTranslation\ (vAddr, DATA)$
$pAddr \leftarrow pAddr_{PSIZE-1...3} \,||\, (pAddr_{2...0}\ xor\ (ReverseEndian \,||\, 0^2))$
$mem \leftarrow LoadMemory\ (uncached, WORD, pAddr, vAddr, DATA)$
$byte \leftarrow vAddr_{2...0}\ xor\ (BigEndianCPU \,||\, 0^2)$
$GPR[rt] \leftarrow (mem_{31+8*byte})^{32} \,||\, mem_{31+8*byte...8*byte}$

Exceptions:

TLB refill exception
TLB invalid exception
Bus error exception
Address error exception

LWCz Load Word To Coprocessor LWCz

31 26	25 21	20 16	15 0
LWCz 1 1 0 0 x x*	base	rt	offset
6	5	5	16

Format:

LWCz rt, offset(base)

Description:

The 16-bit *offset* is sign-extended and added to the contents of general register *base* to form a virtual address. The processor reads a word from the addressed memory location, and makes the data available to coprocessor unit *z*.

The manner in which each coprocessor uses the data is defined by the individual coprocessor specifications.

If either of the two least-significant bits of the effective address is non-zero, an address error exception occurs.

This instruction is not valid for use with CP0.

*See the table “Opcode Bit Encoding” on next page, or “CPU Instruction Opcode Bit Encoding” at the end of Appendix A.

LWCz Load Word To Coprocessor (continued) LWCz

Operation:

32	T:	$vAddr \leftarrow ((offset_{15})^{16} \parallel offset_{15...0}) + GPR[base]$ $(pAddr, uncached) \leftarrow AddressTranslation\ (vAddr, DATA)$ $pAddr \leftarrow pAddr_{PSIZE-1...3} \parallel (pAddr_{2...0}\ xor\ (ReverseEndian \parallel 0^2))$ $mem \leftarrow LoadMemory\ (uncached, WORD, pAddr, vAddr, DATA)$ $byte \leftarrow vAddr_{2...0}\ xor\ (BigEndianCPU \parallel 0^2)$ COPzLW (byte, rt, mem)
64	T:	$vAddr \leftarrow ((offset_{15})^{48} \parallel offset_{15...0}) + GPR[base\}$ $(pAddr, uncached) \leftarrow AddressTranslation\ (vAddr, DATA)$ $pAddr \leftarrow pAddr_{PSIZE-1...3} \parallel (pAddr_{2...0}\ xor\ (ReverseEndian \parallel 0^2))$ $mem \leftarrow LoadMemory\ (uncached, WORD, pAddr, vAddr, DATA)$ $byte \leftarrow vAddr_{2...0}\ xor\ (BigEndianCPU \parallel 0^2)$ COPzLW (byte, rt, mem)

Exceptions:

TLB refill exception
TLB invalid exception
Bus error exception
Address error exception
Coprocessor unusable exception

Opcode Bit Encoding:

LWCz

Bit #	31	30	29	28	27	26	0
LWC1	1	1	0	0	0	1	

Bit #	31	30	29	28	27	26	0
LWC2	1	1	0	0	1	0	
	Opcode				Coprocessor Unit Number		

LWL — Load Word Left — LWL

31 … 26	25 … 21	20 … 16	15 … 0
LWL 1 0 0 0 1 0	base	rt	offset
6	5	5	16

Format:

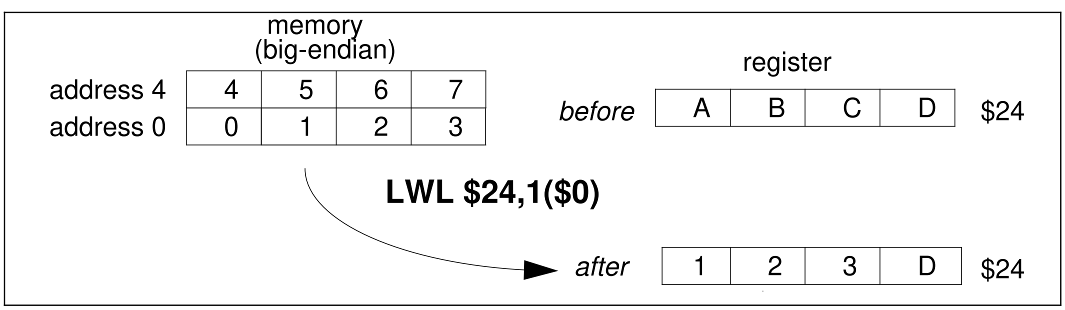
LWL rt, offset(base)

Description:

This instruction can be used in combination with the LWR instruction to load a register with four consecutive bytes from memory, when the bytes cross a word boundary. LWL loads the left portion of the register with the appropriate part of the high-order word; LWR loads the right portion of the register with the appropriate part of the low-order word.

The LWL instruction adds its sign-extended 16-bit *offset* to the contents of general register *base* to form a virtual address which can specify an arbitrary byte. It reads bytes only from the word in memory which contains the specified starting byte. From one to four bytes will be loaded, depending on the starting byte specified. In 64-bit mode, the loaded word is sign-extended.

Conceptually, it starts at the specified byte in memory and loads that byte into the high-order (left-most) byte of the register; then it loads bytes from memory into the register until it reaches the low-order byte of the word in memory. The least-significant (right-most) byte(s) of the register will not be changed.



LWL — Load Word Left (continued) — LWL

The contents of general register *rt* are internally bypassed within the processor so that no NOP is needed between an immediately preceding load instruction which specifies register *rt* and a following LWL (or LWR) instruction which also specifies register *rt*. No address exceptions due to alignment are possible.

Operation:

32	T:	$vAddr \leftarrow ((offset_{15})^{16} \mid\mid offset_{15...0}) + GPR[base]$
		$(pAddr, uncached) \leftarrow AddressTranslation (vAddr, DATA)$
		$pAddr \leftarrow pAddr_{PSIZE-1...3} \mid\mid (pAddr_{2...0}\ xor\ ReverseEndian^3)$
		if BigEndianMem = 0 then
		    $pAddr \leftarrow pAddr_{PSIZE-1...2} \mid\mid 0^2$
		endif
		$byte \leftarrow vAddr_{1...0}\ xor\ BigEndianCPU^2$
		$word \leftarrow vAddr_2\ xor\ BigEndianCPU$
		$mem \leftarrow LoadMemory (uncached, 0 \mid\mid byte, pAddr, vAddr, DATA)$
		$temp \leftarrow mem_{32*word+8*byte+7...32*word} \mid\mid GPR[rt]_{23-8*byte...0}$
		$GPR[rt] \leftarrow temp$

64	T:	$vAddr \leftarrow ((offset_{15})^{48} \mid\mid offset_{15...0}) + GPR[base]$
		$(pAddr, uncached) \leftarrow AddressTranslation (vAddr, DATA)$
		$pAddr \leftarrow pAddr_{PSIZE-1...3} \mid\mid (pAddr_{2...0}\ xor\ ReverseEndian^3)$
		if BigEndianMem = 0 then
		    $pAddr \leftarrow pAddr_{PSIZE-1...2} \mid\mid 0^2$
		endif
		$byte \leftarrow vAddr_{1...0}\ xor\ BigEndianCPU^2$
		$word \leftarrow vAddr_2\ xor\ BigEndianCPU$
		$mem \leftarrow LoadMemory (uncached, 0 \mid\mid byte, pAddr, vAddr, DATA)$
		$temp \leftarrow mem_{32*word+8*byte+7...32*word} \mid\mid GPR[rt]_{23-8*byte...0}$
		$GPR[rt] \leftarrow (temp_{31})^{32} \mid\mid temp$

LWL — Load Word Left (continued) — LWL

Given a doubleword in a register and a doubleword in memory, the operation of LWL is as follows:

LWL

Register	A	B	C	D	E	F	G	H
Memory	I	J	K	L	M	N	O	P

$vAddr_{2..0}$	BigEndianCPU = 0				BigEndianCPU = 1			
	destination	type	offset		destination	type	offset	
			LEM	BEM			LEM	BEM
0	S S S S P F G H	0	0	7	S S S S I J K L	3	4	0
1	S S S S O P G H	1	0	6	S S S S J K L H	2	4	1
2	S S S S N O P H	2	0	5	S S S S K L G H	1	4	2
3	S S S S M N O P	3	0	4	S S S S L F G H	0	4	3
4	S S S S L F G H	0	4	3	S S S S M N O P	3	0	4
5	S S S S K L G H	1	4	2	S S S S N O P H	2	0	5
6	S S S S J K L H	2	4	1	S S S S O P G H	1	0	6
7	S S S S I J K L	3	4	0	S S S S P F G H	0	0	7

LEM	Little-endian memory (BigEndianMem = 0)
BEM	BigEndianMem = 1
Type	AccessType (see Table 2-1) sent to memory
Offset	$pAddr_{2...0}$ sent to memory
S	sign-extend of $destination_{31}$

Exceptions:

TLB refill exception
TLB invalid exception
Bus error exception
Address error exception

LWR Load Word Right LWR

31 26	25 21	20 16	15 0
LWR 1 0 0 1 1 0	base	rt	offset
6	5	5	16

Format:

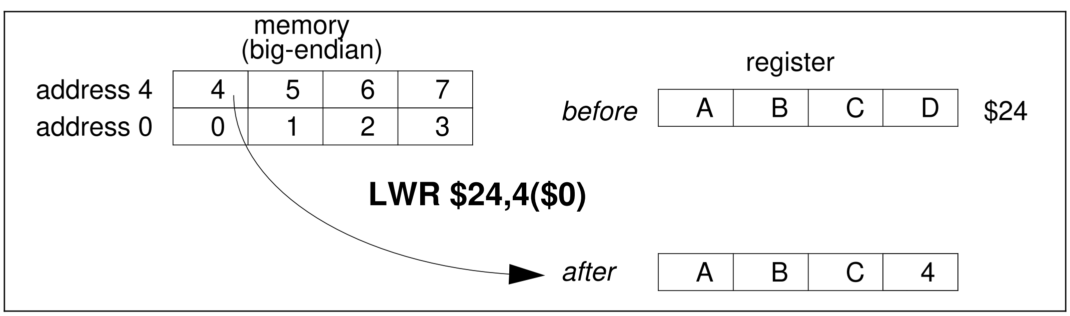
LWR rt, offset(base)

Description:

This instruction can be used in combination with the LWL instruction to load a register with four consecutive bytes from memory, when the bytes cross a word boundary. LWR loads the right portion of the register with the appropriate part of the low-order word; LWL loads the left portion of the register with the appropriate part of the high-order word.

The LWR instruction adds its sign-extended 16-bit *offset* to the contents of general register *base* to form a virtual address which can specify an arbitrary byte. It reads bytes only from the word in memory which contains the specified starting byte. From one to four bytes will be loaded, depending on the starting byte specified. In 64-bit mode, if bit 31 of the destination register is loaded, then the loaded word is sign-extended.

Conceptually, it starts at the specified byte in memory and loads that byte into the low-order (right-most) byte of the register; then it loads bytes from memory into the register until it reaches the high-order byte of the word in memory. The most significant (left-most) byte(s) of the register will not be changed.



LWR — Load Word Right (continued) — LWR

The contents of general register *rt* are internally bypassed within the processor so that no NOP is needed between an immediately preceding load instruction which specifies register *rt* and a following LWR (or LWL) instruction which also specifies register *rt*. No address exceptions due to alignment are possible.

Operation:

32 T: $vAddr \leftarrow ((offset_{15})^{16} \mid\mid offset_{15...0}) + GPR[base]$
$(pAddr, uncached) \leftarrow AddressTranslation\ (vAddr, DATA)$
$pAddr \leftarrow pAddr_{PSIZE-1...3} \mid\mid (pAddr_{2...0}\ xor\ ReverseEndian^3)$
if BigEndianMem = 1 then
$\quad pAddr \leftarrow pAddr_{PSIZE-31...3} \mid\mid 0^3$
endif
$byte \leftarrow vAddr_{1...0}\ xor\ BigEndianCPU^2$
$word \leftarrow vAddr_2\ xor\ BigEndianCPU$
$mem \leftarrow LoadMemory\ (uncached, 0 \mid\mid byte, pAddr, vAddr, DATA)$
$temp \leftarrow GPR[rt]_{31...32-8*byte} \mid\mid mem_{31+32*word...32*word+8*byte}$
$GPR[rt] \leftarrow temp$

64 T: $vAddr \leftarrow ((offset_{15})^{48} \mid\mid offset_{15...0}) + GPR[base]$
$(pAddr, uncached) \leftarrow AddressTranslation\ (vAddr, DATA)$
$pAddr \leftarrow pAddr_{PSIZE-1...3} \mid\mid (pAddr_{2...0}\ xor\ ReverseEndian^3)$
if BigEndianMem = 1 then
$\quad pAddr \leftarrow pAddr_{PSIZE-31...3} \mid\mid 0^3$
endif
$byte \leftarrow vAddr_{1...0}\ xor\ BigEndianCPU^2$
$word \leftarrow vAddr_2\ xor\ BigEndianCPU$
$mem \leftarrow LoadMemory\ (uncached, 0 \mid\mid byte, pAddr, vAddr, DATA)$
$temp \leftarrow GPR[rt]_{31...32-8*byte} \mid\mid mem_{31+32*word...32*word+8*byte}$
$GPR[rt] \leftarrow (temp_{31})^{32} \mid\mid temp$

LWR — Load Word Right (continued) — LWR

Given a word in a register and a word in memory, the operation of LWR is as follows:

LWR

Register	A	B	C	D	E	F	G	H
Memory	I	J	K	L	M	N	O	P

vAddr$_{2..0}$	BigEndianCPU = 0 destination	type	offset LEM	offset BEM	BigEndianCPU = 1 destination	type	offset LEM	offset BEM
0	S S S S M N O P	0	0	4	X X X X E F G I	0	7	0
1	X X X X E M N O	1	1	4	X X X X E F I J	1	6	0
2	X X X X E F M N	2	2	4	X X X X E I J K	2	5	0
3	X X X X E F G M	3	3	4	S S S S I J K L	3	4	0
4	S S S S I J K L	0	4	0	X X X X E F G M	0	3	4
5	X X X X E I J K	1	5	0	X X X X E F M N	1	2	4
6	X X X X E F I J	2	6	0	X X X X E M N O	2	1	4
7	X X X X E F G I	3	7	0	S S S S M N O P	3	0	4

LEM	Little-endian memory (BigEndianMem = 0)
BEM	BigEndianMem = 1
Type	AccessType (see Table 2-1) sent to memory
Offset	pAddr$_{2...0}$ sent to memory
S	sign-extend of destination$_{31}$
X	either unchanged or sign-extend of destination$_{31}$

Exceptions:

TLB refill exception
TLB invalid exception
Bus error exception
Address error exception

LWU Load Word Unsigned LWU

31 … 26	25 … 21	20 … 16	15 … 0
LWU 1 0 0 1 1 1	base	rt	offset
6	5	5	16

Format:

LWU rt, offset(base)

Description:

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

If either of the two least-significant bits of the effective address is non-zero, an address error exception occurs.

This operation is only defined for the R4000 operating in 64-bit mode. Execution of this instruction in 32-bit mode causes a reserved instruction exception.

Operation:

64 T: $vAddr \leftarrow ((offset_{15})^{48} \,||\, offset_{15...0}) + GPR[base]$
$(pAddr, uncached) \leftarrow AddressTranslation\ (vAddr, DATA)$
$pAddr \leftarrow pAddr_{PSIZE-1...3} \,||\, (pAddr_{2...0}\ xor\ (ReverseEndian \,||\, 0^2))$
$mem \leftarrow LoadMemory\ (uncached, WORD, pAddr, vAddr, DATA)$
$byte \leftarrow vAddr_{2...0}\ xor\ (BigEndianCPU \,||\, 0^2)$
$GPR[rt] \leftarrow 0^{32} \,||\, mem_{31+8*byte...8*byte}$

Exceptions:

TLB refill exception
TLB invalid exception
Bus error exception
Address error exception
Reserved instruction exception (R4000 in 32-bit mode)

MFC0 Move From System Control Coprocessor MFC0

31–26	25–21	20–16	15–11	10–0
COP0 0 1 0 0 0 0	MF 0 0 0 0 0	rt	rd	0 000 0000 0000
6	5	5	5	11

Format:

MFC0 rt, rd

Description:

The contents of coprocessor register *rd* of the CP0 are loaded into general register *rt*.

Operation:

32	T:	data ← CPR[0,rd]
	T+1:	GPR[rt] ← data
64	T:	data ← CPR[0,rd]
	T+1:	GPR[rt] ← $(data_{31})^{32}$ || $data_{31...0}$

Exceptions:

Coprocessor unusable exception

MFCz — Move From Coprocessor — MFCz

31–26	25–21	20–16	15–11	10–0
COPz 0 1 0 0 x x*	MF 0 0 0 0 0	rt	rd	0 000 0000 0000
6	5	5	5	11

Format:

MFCz rt, rd

Description:

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

Operation:

32	T:	data $\leftarrow$ CPR[z,rd]
	T+1:	GPR[rt] $\leftarrow$ data
64	T:	if rd_0 = 0 then
		data $\leftarrow$ CPR[z,$rd_{4...1}$ || $0]_{31...0}$
		else
		data $\leftarrow$ CPR[z,$rd_{4...1}$ || $0]_{63...32}$
		endif
	T+1:	GPR[rt] $\leftarrow (data_{31})^{32}$ || data

Exceptions:

Coprocessor unusable exception

*See the table “Opcode Bit Encoding” on next page, or “CPU Instruction Opcode Bit Encoding” at the end of Appendix A.

MFCz — Move From Coprocessor (continued) — MFCz

Opcode Bit Encoding:

MFCz

	Bit # 31	30	29	28	27	26	25	24	23	22	21	0
MFC0	0	1	0	0	0	0	0	0	0	0	0	
MFC1	0	1	0	0	0	1	0	0	0	0	0	
MFC2	0	1	0	0	1	0	0	0	0	0	0	

Opcode (bits 31–28) · Coprocessor Unit Number (bits 27–26) · Coprocessor Suboperation (bits 25–21)