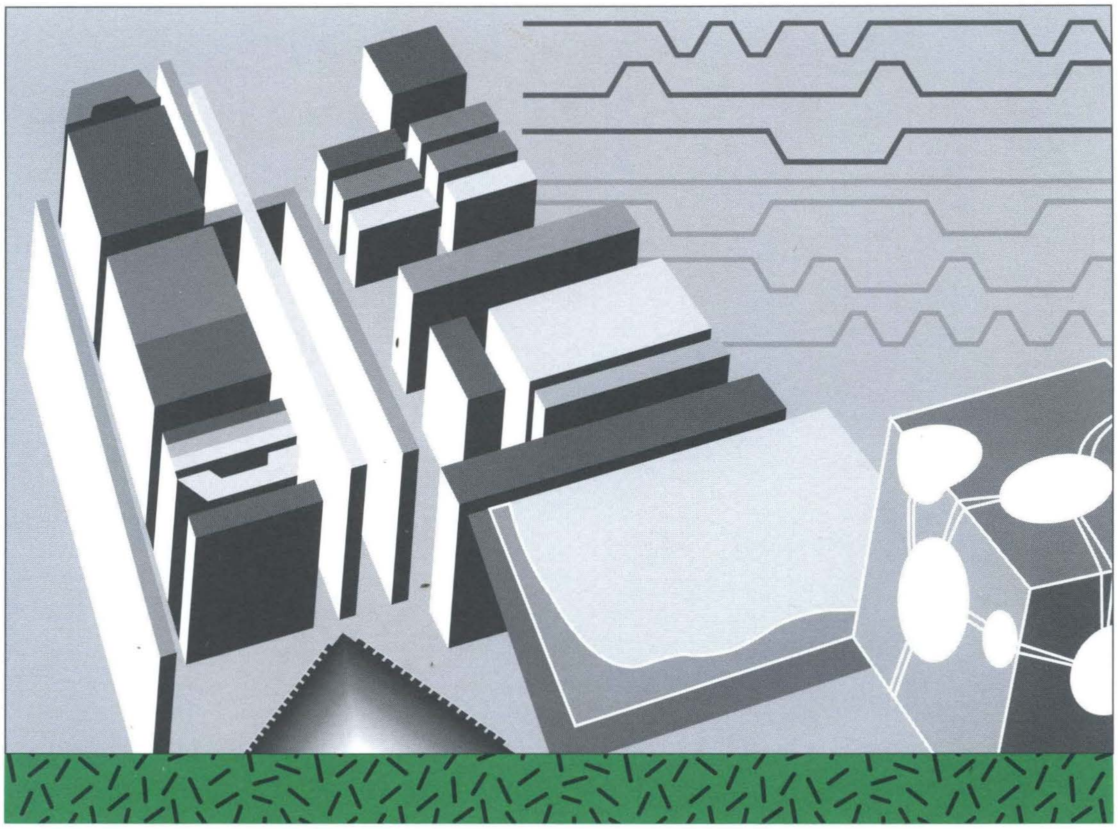
V_R Series™

Errata



MIPS IV

Instruction Set

November 1995

NEC

MIPS IV Instruction Set Errata

November 1995
Document No. 50775E

This document shows the changes made to the "MIPS IV Instruction Set" document between Revision 3.1 and 3.2. It includes the significant changed pages marked, as in the Rev 3.2 document, with change bars.

There are some minor corrections to the instruction descriptions. The main change between these two versions was that the opcode encoding section was expanded. A discussion of instruction decode was added to guide use of the tables. Separate tables of CPU and FPU encodings are provided for each architecture level and for the changes made in each architecture revision.

Changes from previous revision:

Changes are generally marked by change bars in the outer margin of the page – just like the bar to the side of this line. Minor corrections to punctuation and spelling are neither marked with change bars nor noted in this list. Some changes in figures are not marked by change bars due to limitations of the publishing tools.

CVT.D.fmt Instruction

Change the architecture level for the CVT.D.L version of the instruction

from:	
to:	MIPS III

CVT.S.fmt Instruction

Change the architecture level for the CVT.S.L version of the instruction

from:	
to:	MIPS III

LWL Instruction

In the example in Fig. A-4 the sign extension "After executing `LWL $24,2($0)`" should be changed

from:	no cng or sign ext
to:	sign bit (31) extend.

The information in the tables later in the instruction description is correct.

MOVF Instruction

Change the name of the constant value in the function field

from:	MOVC
to:	MOVCI

There is a corresponding change in the FPU opcode encoding table in section 0.4 with opcode=*SPECIAL* and function=MOVC, changing the value to MOVCI.

MOVF.fmt Instruction

Change the name of the constant value in the function field

from:	MOVC
to:	MOVCF

There is a corresponding change in the FPU opcode encoding table in section 0.4 with opcode=*COP1*, fmt = *S* or *D*, and function=MOVC, changing the value to MOVCI.

MOVT Instruction

Change the name of the constant value in the function field

from:	MOVC
to:	MOVCI

There is a corresponding change in the FPU opcode encoding table in section 0.4 with opcode=*SPECIAL* and function=MOVC, changing the value to MOVCI.

MOVT.fmt Instruction

Change the name of the constant value in the function field

from:	MOVC
to:	MOVCF

There is a corresponding change in the FPU opcode encoding table in section 0.4 with opcode=*COP1*, fmt = *S* or *D*, and function=MOVC, changing the value to MOVCI.

CPU Instruction Encoding tables

Revise the presentation of the opcode encoding in section 0 2 for greater clarity when considering different architecture levels or operating a MIPS III or MIPS IV processor in the MIPS II or MIPS III instruction subset modes.

There is a separate encoding table for each architecture level. There is a table of the MIPS IV encodings showing the architecture level at which each opcode was first defined and subsequently modified or extended. There is a separate table for each architecture revision I→II, II→III, and III→IV showing the changes made in that revision.

FPU Instruction Encoding tables

Revise the presentation of the opcode encoding in section 0.4 for greater clarity when considering different architecture levels or operating a MIPS III or MIPS IV processor in the MIPS II or MIPS III instruction subset modes.

There is a separate encoding table for each architecture level. There is a table of the MIPS IV encodings showing the architecture level at which each opcode was first defined and subsequently modified or extended. There is a separate table for each architecture revision I→II, II→III, and III→IV showing the changes made in that revision.

31 26	25 21	20 16	15 0
LWL 100010	base	rt	offset
6	5	5	16

Format: LWL rt, offset(base) **MIPS I**

Purpose: To load the most-significant part of a word as a signed value from an unaligned memory address.

Description: rt ← rt MERGE memory[base+offset]

The 16-bit signed *offset* is added to the contents of GPR *base* to form an effective address (*EffAddr*). *EffAddr* is the address of the most-significant of four consecutive bytes forming a word in memory (*W*) starting at an arbitrary byte boundary. A part of *W*, the most-significant one to four bytes, is in the aligned word containing *EffAddr*. This part of *W* is loaded into the most-significant (left) part of the word in GPR *rt*. The remaining least-significant part of the word in GPR *rt* is unchanged.

If GPR *rt* is a 64-bit register, the destination word is the low-order word of the register. The loaded value is treated as a signed value; the word sign bit (bit 31) is always loaded from memory and the new sign bit value is copied into bits 63..32.

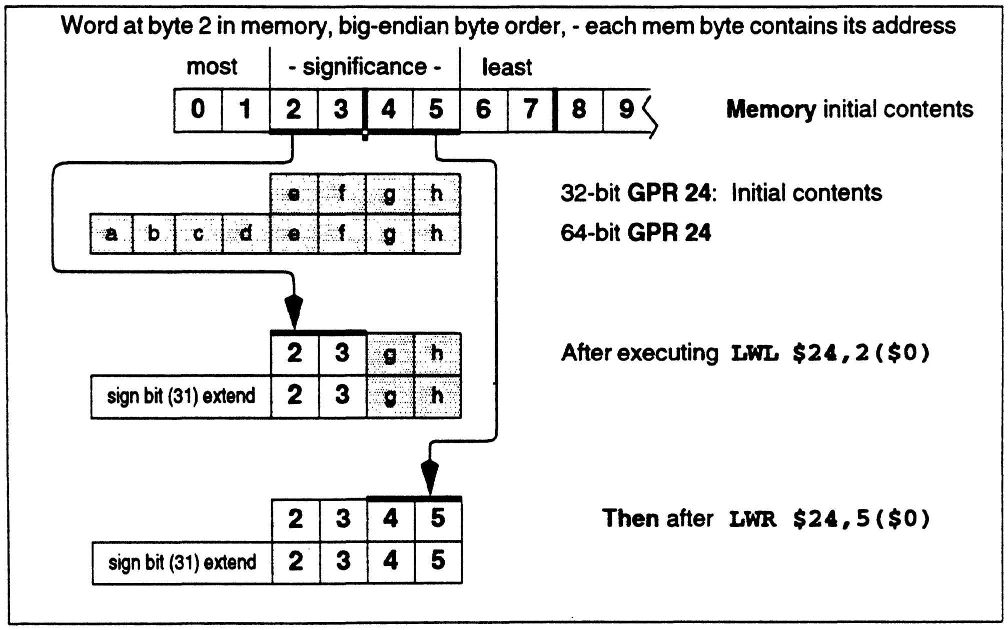

Word at byte 2 in memory, big-endian byte order, - each mem byte contains its address
most
- significance -
least
0 1 2 3 4 5 6 7 8 9
Memory initial contents
e f g h
a b c d e f g h
32-bit GPR 24: Initial contents
64-bit GPR 24
2 3 g h
sign bit (31) extend 2 3 g h
After executing LWL $24,2($0)
2 3 4 5
sign bit (31) extend 2 3 4 5
Then after LWR $24,5($0)

Figure A-2 Unaligned Word Load using LWL and LWR.

The figure above illustrates this operation for big-endian byte ordering for 32-bit and 64-bit registers. The four consecutive bytes in 2..5 form an unaligned word starting at location 2. A part of W, two bytes, is in the aligned word containing the most-significant byte at 2. First, LWL loads these two bytes into the left part of the destination register word and leaves the right part of the destination word unchanged. Next, the complementary LWR loads the remainder of the unaligned word.

The bytes loaded from memory to the destination register depend on both the offset of the effective address within an aligned word, i.e. the low two bits of the address ($vAddr_{1..0}$), and the current byte ordering mode of the processor (big- or little-endian). The table below shows the bytes loaded for every combination of offset and byte ordering.

Table A-28 Bytes Loaded by LWL Instruction

Memory contents and byte offsets

0	1	2	3	← big-endian
I	J	K	L	offset ($vAddr_{1..0}$)
3	2	1	0	← little-endian
most			least	

— significance —

Initial contents of Dest Register

64-bit register

a	b	c	d	e	f	g	h
most		— significance —					least

32-bit register

e	f	g	h

Destination 64-bit register contents after instruction (shaded is unchanged)

Big-endian byte ordering					$vAddr_{1..0}$	Little-endian byte ordering				
sign bit (31) extended	I	J	K	L	0	sign bit (31) extended	L	f	g	h
sign bit (31) extended	J	K	L	h	1	sign bit (31) extended	K	L	g	h
sign bit (31) extended	K	L	g	h	2	sign bit (31) extended	J	K	L	h
sign bit (31) extended	L	f	g	h	3	sign bit (31) extended	I	J	K	L

The word sign (31) is always loaded and the value is copied into bits 63..32.

32-bit register

Big-endian				$vAddr_{1..0}$	Little-endian			
I	J	K	L	0	L	f	g	h
J	K	L	h	1	K	L	g	h
K	L	g	h	2	J	K	L	h
L	f	g	h	3	I	J	K	L

The unaligned loads, LWL and LWR, are exceptions to the load-delay scheduling restriction in the MIPS I architecture. An unaligned load instruction to GPR *rt* that immediately follows another load to GPR *rt* can "read" the loaded data. It will correctly merge the 1 to 4 loaded bytes with the data loaded by the previous instruction.

Restrictions:

MIPS I scheduling restriction: The loaded data is not available for use by the following instruction. The instruction immediately following this one, unless it is an unaligned load (LWL, LWR), may not use GPR *rt* as a source register. If this restriction is violated, the result of the operation is undefined.

Operation: 32-bit processors

```
vAddr ← sign_extend(offset) + GPR[base]
(pAddr, uncached) ← AddressTranslation (vAddr, DATA, LOAD)
pAddr ← $pAddr_{(PSIZE-1)..2}$ || ($pAddr_{1..0}$ xor $ReverseEndian^2$)
if BigEndianMem = 0 then
    pAddr ← $pAddr_{(PSIZE-1)..2}$ || $0^2$
endif
byte ← $vAddr_{1..0}$ xor $BigEndianCPU^2$
memword ← LoadMemory (uncached, byte, pAddr, vAddr, DATA)
GPR[rt] ← $memword_{7+8*byte..0}$ || $GPR[rt]_{23-8*byte..0}$
```

Operation: 64-bit processors

```
vAddr ← sign_extend(offset) + GPR[base]
(pAddr, uncached) ← AddressTranslation (vAddr, DATA, LOAD)
pAddr ← $pAddr_{(PSIZE-1)..3}$ || ($pAddr_{2..0}$ xor $ReverseEndian^3$)
if BigEndianMem = 0 then
    pAddr ← $pAddr_{(PSIZE-1)..3}$ || $0^3$
endif
byte ← 0 || ($vAddr_{1..0}$ xor $BigEndianCPU^2$)
word ← $vAddr_2$ xor BigEndianCPU
memdouble ← LoadMemory (uncached, byte, pAddr, vAddr, DATA)
temp ← $memdouble_{31+32*word-8*byte..32*word}$ || $GPR[rt]_{23-8*byte..0}$
GPR[rt] ← $(temp_{31})^{32}$ || temp
```

Exceptions:

TLB Refill, TLB Invalid
Bus Error
Address Error

Programming Notes:

The architecture provides no direct support for treating unaligned words as unsigned values, i.e. zeroing bits 63..32 of the destination register when bit 31 is loaded. See SLL or SLLV for a single-instruction method of propagating the word sign bit in a register into the upper half of a 64-bit register.

Floating-Point Convert to Double Floating-Point **CVT.D.fmt**

31 26	25 21	20 16	15 11	10 6	5 0
COP1 010001	fmt	0 00000	fs	fd	CVT.D 100001
6	5	5	5	5	6

Format:	CVT.D.S	fd, fs	**MIPS I**
	CVT.D.W	fd, fs	
	CVT.D.L	fd, fs	**MIPS III**

Purpose: To convert an FP or fixed-point value to double FP.

Description: fd ← convert_and_round(fs)

The value in FPR *fs* in format *fmt* is converted to a value in double floating-point format rounded according to the current rounding mode in FCSR. The result is placed in FPR *fd*.

If *fmt* is S or W, then the operation is always exact.

Restrictions:

The fields *fs and fd* must specify valid FPRs; *fs* for type *fmt* and *fd* for double floating-point; see **Floating-Point Registers** on page B-6. If they are not valid, the result is undefined.

The operand must be a value in format *fmt*; see section B 7 on page B-24. If it is not, the result is undefined and the value of the operand FPR becomes undefined.

Operation:

```
StoreFPR (fd, D, ConvertFmt(ValueFPR(fs, fmt), fmt, D))
```

Exceptions:

Coprocessor Unusable
Reserved Instruction
Floating-Point
- Invalid Operation
- Unimplemented Operation
- Inexact
- Overflow
- Underflow

31 26	25 21	20 16	15 11	10 6	5 0
COP1 010001	fmt	0 00000	fs	fd	CVT.S 100000
6	5	5	5	5	6

Format: CVT.S.D fd, fs **MIPS I**
CVT.S.W fd, fs
CVT.S.L fd, fs **MIPS III**

Purpose: To convert an FP or fixed-point value to single FP.

Description: fd ← convert_and_round(fs)

The value in FPR *fs* in format *fmt* is converted to a value in single floating-point format rounded according to the current rounding mode in FCSR. The result is placed in FPR *fd*.

Restrictions:

The fields *fs and fd* must specify valid FPRs; *fs* for type *fmt* and *fd* for single floating-point; see **Floating-Point Registers** on page B-6. If they are not valid, the result is undefined.

The operand must be a value in format *fmt*; see section B 7 on page B-24. If it is not, the result is undefined and the value of the operand FPR becomes undefined.

Operation:

```
StoreFPR(fd, S, ConvertFmt(ValueFPR(fs, fmt), fmt, S))
```

Exceptions:

Coprocessor Unusable
Reserved Instruction
Floating-Point
  Invalid Operation
  Unimplemented Operation
  Inexact
  Overflow
  Underflow

MOVF

Move Conditional on FP False

31 26	25 21	20 18	17	16	15 11	10 6	5 0
SPECIAL 000000	rs	cc	0 0	tf 0	rd	0 00000	MOVCI 000001
6	5	3	1	1	5	5	6

Format: MOVF rd, rs, cc **MIPS IV**

Purpose: To test an FP condition code then conditionally move a GPR.

Description: if (cc = 0) then rd ← rs

If the floating-point condition code specified by *cc* is zero, then the contents of GPR *rs* are placed into GPR *rd*.

Restrictions:

None

Operation:

```
active ← FCC[cc] = tf
if active then
    GPR[rd] ← GPR[rs]
endif
```

Exceptions:

Reserved Instruction
Coprocessor Unusable

31 26	25 21	20 18	17	16	15 11	10 6	5 0
COP1 010001	fmt	cc	0 0	tf 0	fs	fd	MOVCF 010001
6	5	3	1	1	5	5	6

Format: MOVF.S fd, fs, cc **MIPS IV**
MOVF.D fd, fs, cc

Purpose: To test an FP condition code then conditionally move an FP value.

Description: if (cc = 0) then fd ← fs

If the floating-point condition code specified by *cc* is zero, then the value in FPR *fs* is placed into FPR *fd*. The source and destination are values in format *fmt*.

If the condition code is not zero, then FPR *fs* is not copied and FPR *fd* contains its previous value in format *fmt*. If *fd* did not contain a value either in format *fmt* or previously unused data from a load or move-to operation that could be interpreted in format *fmt*, then the value of *fd* becomes undefined.

The move is non-arithmetic; it causes no IEEE 754 exceptions.

Restrictions:

The fields *fs* and *fd* must specify FPRs valid for operands of type *fmt*; see **Floating-Point Registers** on page B-6. If they are not valid, the result is undefined.

The operand must be a value in format *fmt*; see section B 7 on page B-24. If it is not, the result is undefined and the value of the operand FPR becomes undefined.

Operation:

```
if FCC[cc] = tf then
    StoreFPR(fd, fmt, ValueFPR(fs, fmt))
else
    StoreFPR(fd, fmt, ValueFPR(fd, fmt))
endif
```

Exceptions:

Coprocessor Unusable
Reserved Instruction
Floating-Point
    Unimplemented operation

Move Conditional on FP True **MOVT**

31 26	25 21	20 18	17	16	15 11	10 6	5 0
SPECIAL 000000	rs	cc	0 0	tf 1	rd	0 00000	MOVCI 000001
6	5	3	1	1	5	5	6

Format: MOVT rd, rs, cc **MIPS IV**

Purpose: To test an FP condition code then conditionally move a GPR.

Description: if (cc = 1) then rd ← rs

If the floating-point condition code specified by *cc* is one then the contents of GPR *rs* are placed into GPR *rd*.

Restrictions:

None

Operation:

```
if FCC[cc] = tf then
    GPR[rd] ← GPR[rs]
endif
```

Exceptions:

Reserved Instruction
Coprocessor Unusable

MOVT.fmt Floating-Point Move Conditional on FP True

31 26	25 21	20 18	17	16	15 11	10 6	5 0
COP1 010001	fmt	cc	0 0	tf 1	fs	fd	MOVCF 010001
6	5	3	1	1	5	5	6

Format: MOVT.S fd, fs, cc **MIPS IV**
MOVT.D fd, fs, cc

Purpose: To test an FP condition code then conditionally move an FP value.

Description: if (cc = 1) then fd ← fs

If the floating-point condition code specified by *cc* is one then the value in FPR *fs* is placed into FPR *fd*. The source and destination are values in format *fmt*.

If the condition code is not one, then FPR *fs* is not copied and FPR *fd* contains its previous value in format *fm*t. If *fd* did not contain a value either in format *fmt* or previously unused data from a load or move-to operation that could be interpreted in format *fmt*, then the value of *fd* becomes undefined.

The move is non-arithmetic; it causes no IEEE 754 exceptions.

Restrictions:

The fields *fs* and *fd* must specify FPRs valid for operands of type *fmt*; see **Floating-Point Registers** on page B-6. If they are not valid, the result is undefined.

The operand must be a value in format *fmt*; see section B 7 on page B-24. If it is not, the result is undefined and the value of the operand FPR becomes undefined.

Operation:

```
if FCC[cc] = tf then
    StoreFPR(fd, fmt, ValueFPR(fs, fmt))
else
    StoreFPR(fd, fmt, ValueFPR(fd, fmt))
endif
```

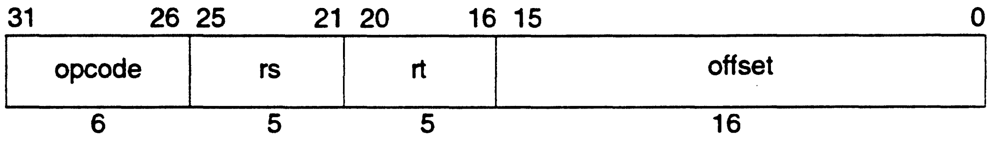
Exceptions:

Coprocessor Unusable
Reserved Instruction
Floating-Point
    Unimplemented operation

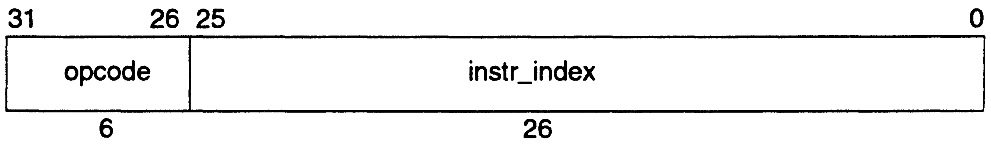
A 7 CPU Instruction Formats

A CPU instruction is a single 32-bit aligned word. The major instruction formats are shown in Figure A-10.

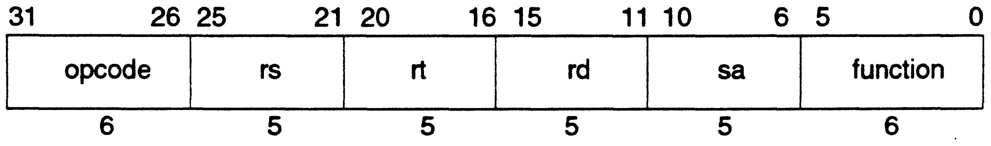
I-Type (Immediate).



J-Type (Jump).



R-Type (Register).



opcode	6-bit primary operation code
rd	5-bit destination register specifier
rs	5-bit source register specifier
rt	5-bit target (source/destination) register specifier or used to specify functions within the primary opcode value *REGIMM*
immediate	16-bit signed immediate used for: logical operands, arithmetic signed operands, load/store address byte offsets, PC-relative branch signed instruction displacement
instr_index	26-bit index shifted left two bits to supply the low-order 28 bits of the jump target address.
sa	5-bit shift amount
function	6-bit function field used to specify functions within the primary operation code value *SPECIAL*.

Figure A-10 CPU Instruction Formats

A 8 CPU Instruction Encoding

This section describes the encoding of user-level, i.e. non-privileged, CPU instructions for the four levels of the MIPS architecture, MIPS I through MIPS IV. Each architecture level includes the instructions in the previous level;† MIPS IV includes all instructions in MIPS I, MIPS II, and MIPS III. This section presents eight different views of the instruction encoding.

- Separate encoding tables for each architecture level.
- A MIPS IV encoding table showing the architecture level at which each opcode was originally defined and subsequently modified (if modified).
- Separate encoding tables for each architecture revision showing the changes made during that revision.

A 8.1 Instruction Decode

Instruction field names are printed in **bold** in this section.

The primary **opcode** field is decoded first. Most **opcode** values completely specify an instruction that has an immediate value or offset. **Opcode** values that do not specify an instruction specify an instruction class. Instructions within a class are further specified by values in other fields. The **opcode** values *SPECIAL* and *REGIMM* specify instruction classes. The *COP0, COP1, COP2, COP3,* and *COP1X* instruction classes are not CPU instructions; they are discussed in section A 8.3.

A 8.1.1 *SPECIAL* Instruction Class

The **opcode**=*SPECIAL* instruction class encodes 3-register computational instructions, jump register, and some special purpose instructions. The class is further decoded by examining the **format** field. The **format** values fully specify the CPU instructions; the *MOVCI* instruction class is not a CPU instruction class.

A 8.1.2 *REGIMM* Instruction Class

The **opcode**=*REGIMM* instruction class encodes conditional branch and trap immediate instructions. The class is further decode, and the instructions fully specified, by examining the **rt** field.

A 8.2 Instruction Subsets of MIPS III and MIPS IV Processors.

MIPS III processors, such as the R4000, R4200, R4300, R4400, and R4600, have a processor mode in which only the MIPS II instructions are valid. The MIPS II encoding table describes the MIPS II-only mode except that the Coprocessor 3 instructions (COP3, LWC3, SWC3, LDC3, SDC3) are not available and cause a Reserved Instruction exception.

† An exception to this rule is that the reserved, but never implemented, Coprocessor 3 instructions were removed or changed to another use starting in MIPS III.

MIPS IV processors, such as the R8000 and R10000, have processor modes in which only the MIPS II or MIPS III instructions are valid. The MIPS II encoding table describes the MIPS II-only mode except that the Coprocessor 3 instructions (COP3, LWC3, SWC3, LDC3, SDC3) are not available and cause a Reserved Instruction exception. The MIPS III encoding table describes the MIPS III-only mode.

A 8.3 Non-CPU Instructions in the Tables

The encoding tables show all values for the field they describe and by doing this they include some entries that are not user-level CPU instructions. The primary opcode table includes coprocessor instruction classes (COP0, COP1, COP2, COP3/COP1X) and coprocessor load/store instructions (LWCx, SWCx, LDCx, SDCx for x=1, 2, or 3). The **opcode**=*SPECIAL* + **function**=*MOVCI* instruction class is an FPU instruction.

A 8.3.1 Coprocessor 0 - *COP0*

COP0 encodes privileged instructions for Coprocessor 0, the System Control Coprocessor. The definition of the System Control Coprocessor is processor-specific and further information on these instructions are not included in this document.

A 8.3.2 Coprocessor 1 - *COP1, COP1X, MOVCI,* and CP1 load/store.

Coprocessor 1 is the floating-point unit in the MIPS architecture. *COP1, COP1X,* and the (**opcode**=*SPECIAL* + **function**=*MOVCI*) instruction classes encode floating-point instructions. LWC1, SWC1, LDC1, and SDC1 are floating-point loads and stores. The FPU instruction encoding is documented in section B.12.

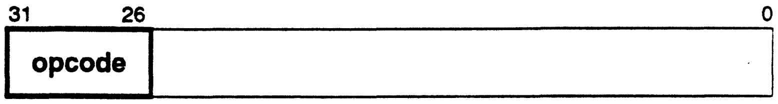
A 8.3.3 Coprocessor 2 - *COP2* and CP2 load/store.

Coprocessor 2 is optional and implementation-specific. No standard processor from MIPS has implemented coprocessor 2, but MIPS' semiconductor licensees may have implemented it in a product based on one of the standard MIPS processors. At this time the standard processors are: R2000, R3000, R4000, R4200, R4300, R4400, R4600, R6000, R8000, and R10000.

A 8.3.4 Coprocessor 3 - *COP3* and CP3 load/store.

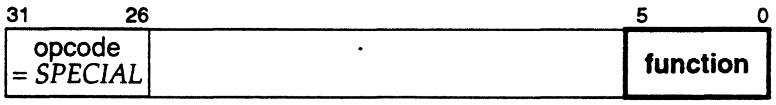
Coprocessor 3 is optional and implementation-specific in the MIPS I and MIPS II architecture levels. It was removed from MIPS III and later architecture levels. Note that in MIPS IV the *COP3* primary opcode was reused for the *COP1X* instruction class. No standard processor from MIPS has implemented coprocessor 2, but MIPS' semiconductor licensees may have implemented it in a product based on one of the standard MIPS processors. At this time the standard processors are: R2000, R3000, R4000, R4200, R4300, R4400, R4600, R6000, R8000, and R10000.

Table A-37 CPU Instruction Encoding - MIPS I Architecture



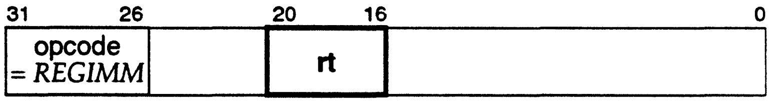
opcode bits 28..26 — Instructions encoded by **opcode** field.

bits 31..29	0 000	1 001	2 010	3 011	4 100	5 101	6 110	7 111
0 000	*SPECIAL* δ	*REGIMM* δ	J	JAL	BEQ	BNE	BLEZ	BGTZ
1 001	ADDI	ADDIU	SLTI	SLTIU	ANDI	ORI	XORI	LUI
2 010	*COP0* δ,π	*COP1* δ,π	*COP2* δ,π	*COP3* δ,π,κ	*	*	*	*
3 011	*	*	*	*	*	*	*	*
4 100	LB	LH	LWL	LW	LBU	LHU	LWR	*
5 101	SB	SH	SWL	SW	*	*	SWR	*
6 110	*	LWC1 π	LWC2 π	LWC3 π,κ	*	*	*	*
7 111	*	SWC1 π	SWC2 π	SWC3 π,κ	*	*	*	*



function bits 2..0 — Instructions encoded by **function** field when opcode field = SPECIAL.

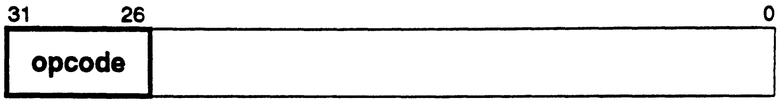
bits 5..3	0 000	1 001	2 010	3 011	4 100	5 101	6 110	7 111
0 000	SLL	*	SRL	SRA	SLLV	*	SRLV	SRAV
1 001	JR	JALR	*	*	SYSCALL	BREAK	*	*
2 010	MFHI	MTHI	MFLO	MTLO	*	*	*	*
3 011	MULT	MULTU	DIV	DIVU	*	*	*	*
4 100	ADD	ADDU	SUB	SUBU	AND	OR	XOR	NOR
5 101	*	*	SLT	SLTU	*	*	*	*
6 110	*	*	*	*	*	*	*	*
7 111	*	*	*	*	*	*	*	*



rt bits 18..16 — Instructions encoded by the **rt** field when opcode field = REGIMM.

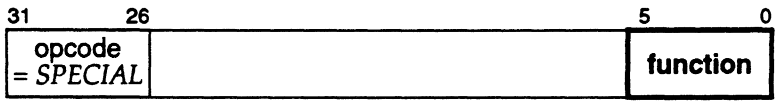
bits 20..19	0 000	1 001	2 010	3 011	4 100	5 101	6 110	7 111
0 00	BLTZ	BGEZ	†	†	†	†	†	†
1 01	†	†	†	†	†	†	†	†
2 10	BLTZAL	BGEZAL	†	†	†	†	†	†
3 11	†	†	†	†	†	†	†	†

Table A-38 CPU Instruction Encoding - MIPS II Architecture



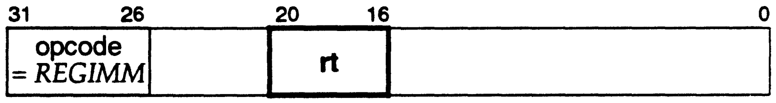
opcode bits 28..26 Instructions encoded by **opcode** field.

bits	0	1	2	3	4	5	6	7
31..29	000	001	010	011	100	101	110	111
0 000	*SPECIAL* δ	*REGIMM* δ	J	JAL	BEQ	BNE	BLEZ	BGTZ
1 001	ADDI	ADDIU	SLTI	SLTIU	ANDI	ORI	XORI	LUI
2 010	*COP0* δ,π	*COP1* δ,π	*COP2* δ,π	*COP3* δ,π,κ	BEQL	BNEL	BLEZL	BGTZL
3 011	*	*	*	*	*	*	*	*
4 100	LB	LH	LWL	LW	LBU	LHU	LWR	*
5 101	SB	SH	SWL	SW	*	*	SWR	ρ
6 110	LL	LWC1 π	LWC2 π	LWC3 π,κ	*	LDC1 π	LDC2 π	LDC3 π,κ
7 111	SC	SWC1 π	SWC2 π	SWC3 π,κ	*	SDC1 π	SDC2 π	SDC3 π,κ



function bits 2..0 Instructions encoded by **function** field when opcode field = SPECIAL.

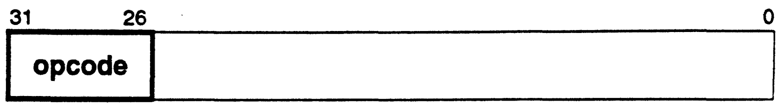
bits	0	1	2	3	4	5	6	7
5..3	000	001	010	011	100	101	110	111
0 000	SLL	*	SRL	SRA	SLLV	*	SRLV	SRAV
1 001	JR	JALR	*	*	SYSCALL	BREAK	*	SYNC
2 010	MFHI	MTHI	MFLO	MTLO	*	*	*	*
3 011	MULT	MULTU	DIV	DIVU	*	*	*	*
4 100	ADD	ADDU	SUB	SUBU	AND	OR	XOR	NOR
5 101	*	*	SLT	SLTU	*	*	*	*
6 110	TGE	TGEU	TLT	TLTU	TEQ	*	TNE	*
7 111	*	*	*	*	*	*	*	*



rt bits 18..16 Instructions encoded by the **rt** field when opcode field = REGIMM.

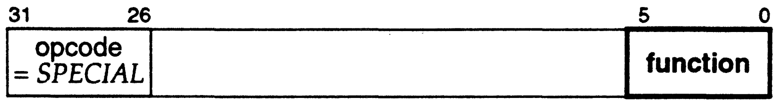
bits	0	1	2	3	4	5	6	7
20..19	000	001	010	011	100	101	110	111
0 00	BLTZ	BGEZ	BLTZL	BGEZL	*	*	*	*
1 01	TGEI	TGEIU	TLTI	TLTIU	TEQI	*	TNEI	*
2 10	BLTZAL	BGEZAL	BLTZALL	BGEZALL	*	*	*	*
3 11	*	*	*	*	*	*	*	*

Table A-39 CPU Instruction Encoding - MIPS III Architecture



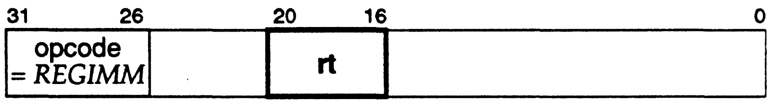
opcode bits 28..26 — Instructions encoded by **opcode** field.

bits 31..29	0 000	1 001	2 010	3 011	4 100	5 101	6 110	7 111
0 000	*SPECIAL* δ	*REGIMM* δ	J	JAL	BEQ	BNE	BLEZ	BGTZ
1 001	ADDI	ADDIU	SLTI	SLTIU	ANDI	ORI	XORI	LUI
2 010	*COP0* δ,π	*COP1* δ,π	*COP2* δ,π	*	BEQL	BNEL	BLEZL	BGTZL
3 011	DADDI	DADDIU	LDL	LDR	*	*	*	*
4 100	LB	LH	LWL	LW	LBU	LHU	LWR	LWU
5 101	SB	SH	SWL	SW	SDL	SDR	SWR	ρ
6 110	LL	LWC1 π	LWC2 π	*	LLD	LDC1 π	LDC2 π	LD
7 111	SC	SWC1 π	SWC2 π	*	SCD	SDC1 π	SDC2 π	SD



function bits 2..0 — Instructions encoded by **function** field when opcode field = SPECIAL.

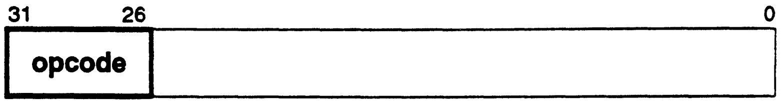
bits 5..3	0 000	1 001	2 010	3 011	4 100	5 101	6 110	7 111
0 000	SLL	*	SRL	SRA	SLLV	*	SRLV	SRAV
1 001	JR	JALR	*	*	SYSCALL	BREAK	*	SYNC
2 010	MFHI	MTHI	MFLO	MTLO	DSLLV	*	DSRLV	DSRAV
3 011	MULT	MULTU	DIV	DIVU	DMULT	DMULTU	DDIV	DDIVU
4 100	ADD	ADDU	SUB	SUBU	AND	OR	XOR	NOR
5 101	*	*	SLT	SLTU	DADD	DADDU	DSUB	DSUBU
6 110	TGE	TGEU	TLT	TLTU	TEQ	*	TNE	*
7 111	DSLL	*	DSRL	DSRA	DSLL32	*	DSRL32	DSRA32



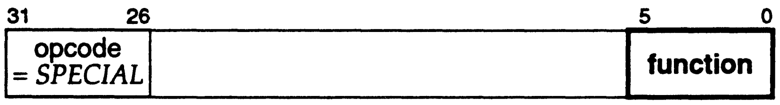
rt bits 18..16 — Instructions encoded by the **rt** field when opcode field = REGIMM.

bits 20..19	0 000	1 001	2 010	3 011	4 100	5 101	6 110	7 111
0 00	BLTZ	BGEZ	BLTZL	BGEZL	*	*	*	*
1 01	TGEI	TGEIU	TLTI	TLTIU	TEQI	*	TNEI	*
2 10	BLTZAL	BGEZAL	BLTZALL	BGEZALL	*	*	*	*
3 11	*	*	*	*	*	*	*	*

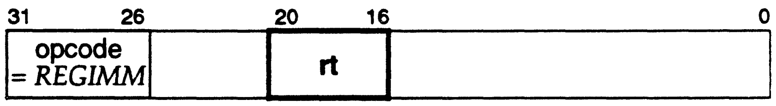
Table A-40 CPU Instruction Encoding - MIPS IV Architecture



opcode	bits 28..26		Instructions encoded by **opcode** field.					
bits	0	1	2	3	4	5	6	7
31..29	000	001	010	011	100	101	110	111
0 000	*SPECIAL* δ	*REGIMM* δ	J	JAL	BEQ	BNE	BLEZ	BGTZ
1 001	ADDI	ADDIU	SLTI	SLTIU	ANDI	ORI	XORI	LUI
2 010	*COP0* δ,π	*COP1* δ,π	*COP2* δ,π	*COP1X* δ,π	BEQL	BNEL	BLEZL	BGTZL
3 011	DADDI	DADDIU	LDL	LDR		*	*	*
4 100	LB	LH	LWL	LW	LBU	LHU	LWR	LWU
5 101	SB	SH	SWL	SW	SDL	SDR	SWR	ρ
6 110	LL	LWC1 π	LWC2 π	PREF	LLD	LDC1 π	LDC2 π	LD
7 111	SC	SWC1 π	SWC2 π	*	SCD	SDC1 π	SDC2 π	SD



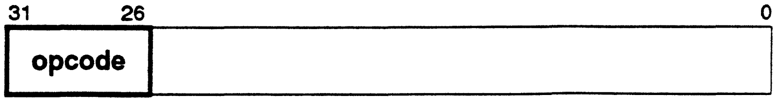
function	bits 2..0		Instructions encoded by **function** field when opcode field = SPECIAL.					
bits	0	1	2	3	4	5	6	7
5..3	000	001	010	011	100	101	110	111
0 000	SLL	*MOVCI* δ,μ	SRL	SRA	SLLV	*	SRLV	SRAV
1 001	JR	JALR	MOVZ	MOVN	SYSCALL	BREAK	*	SYNC
2 010	MFHI	MTHI	MFLO	MTLO	DSLLV	*	DSRLV	DSRAV
3 011	MULT	MULTU	DIV	DIVU	DMULT	DMULTU	DDIV	DDIVU
4 100	ADD	ADDU	SUB	SUBU	AND	OR	XOR	NOR
5 101	*	*	SLT	SLTU	DADD	DADDU	DSUB	DSUBU
6 110	TGE	TGEU	TLT	TLTU	TEQ	*	TNE	*
7 111	DSLL	*	DSRL	DSRA	DSLL32	*	DSRL32	DSRA32



rt	bits 18..16		Instructions encoded by the **rt** field when opcode field = REGIMM.					
bits	0	1	2	3	4	5	6	7
20..19	000	001	010	011	100	101	110	111
0 00	BLTZ	BGEZ	BLTZL	BGEZL	*	*	*	*
1 01	TGEI	TGEIU	TLTI	TLTIU	TEQI	*	TNEI	*
2 10	BLTZAL	BGEZAL	BLTZALL	BGEZALL	*	*	*	*
3 11	*	*	*	*	*	*	*	*

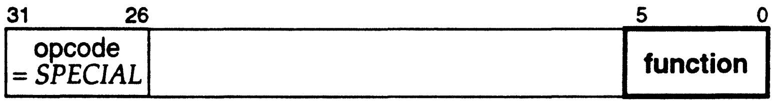
Table A-41 Architecture Level in Which CPU Instructions are Defined or Extended.

The architecture level in which each MIPS IVencoding was defined is indicated by a subscript 1, 2, 3, or 4 (for architecture level I, II, III, or IV). If an instruction or instruction class was later extended, the extending level is indicated after the defining level.



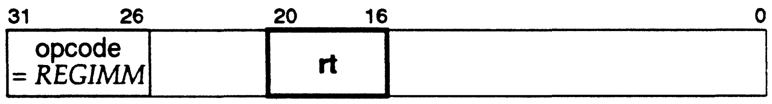
opcode bits 31..29	bits 28..26: 0 000	1 001	2 010	3 011	4 100	5 101	6 110	7 111
0 000	*SPECIAL*$_{1\text{-}4}$	*REGIMM*$_{1,2}$	J$_1$	JAL$_1$	BEQ$_1$	BNE$_1$	BLEZ$_1$	BGTZ$_1$
1 001	ADDI$_1$	ADDIU$_1$	SLTI$_1$	SLTIU$_1$	ANDI$_1$	ORI$_1$	XORI$_1$	LUI$_1$
2 010	*COP0*$_1$	*COP1*$_{1,2,3,4}$	*COP2*$_1$	*COP1X*$_4$	BEQL$_2$	BNEL$_2$	BLEZL$_2$	BGTZL$_2$
3 011	DADDI$_3$	DADDIU$_3$	LDL$_3$	LDR$_3$	*$_1$	*$_1$	*$_1$	*$_1$
4 100	LB$_1$	LH$_1$	LWL$_1$	LW$_1$	LBU$_1$	LHU$_1$	LWR$_1$	LWU$_3$
5 101	SB$_1$	SH$_1$	SWL$_1$	SW$_1$	SDL$_3$	SDR$_3$	SWR$_1$	ρ_2
6 110	LL$_2$	LWC1$_1$	LWC2$_1$	PREF$_4$	LLD$_3$	LDC1$_2$	LDC2$_2$	LD$_3$
7 111	SC$_2$	SWC1$_1$	SWC2$_1$	*$_3$	SCD$_3$	SDC1$_2$	SDC2$_2$	SD$_3$

Instructions encoded by **opcode** field.



function bits 5..3	bits 2..0: 0 000	1 001	2 010	3 011	4 100	5 101	6 110	7 111
0 000	SLL$_1$	*MOVCI*$_4$	SRL$_1$	SRA$_1$	SLLV$_1$	*$_1$	SRLV$_1$	SRAV$_1$
1 001	JR$_1$	JALR$_1$	MOVZ$_4$	MOVN$_4$	SYSCALL$_1$	BREAK$_1$	*$_1$	SYNC$_2$
2 010	MFHI$_1$	MTHI$_1$	MFLO$_1$	MTLO$_1$	DSLLV$_3$	*$_1$	DSRLV$_3$	DSRAV$_3$
3 011	MULT$_1$	MULTU$_1$	DIV$_1$	DIVU$_1$	DMULT$_3$	DMULTU$_3$	DDIV$_3$	DDIVU$_3$
4 100	ADD$_1$	ADDU$_1$	SUB$_1$	SUBU$_1$	AND$_1$	OR$_1$	XOR$_1$	NOR$_1$
5 101	*$_1$	*$_1$	SLT$_1$	SLTU$_1$	DADD$_3$	DADDU$_3$	DSUB$_3$	DSUBU$_3$
6 110	TGE$_2$	TGEU$_2$	TLT$_2$	TLTU$_2$	TEQ$_2$	*$_1$	TNE$_2$	*$_1$
7 111	DSLL$_3$	*$_1$	DSRL$_3$	DSRA$_3$	DSLL32$_3$	*$_1$	DSRL32$_3$	DSRA32$_3$

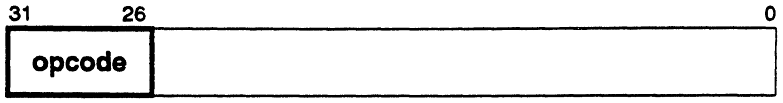
Instructions encoded by **function** field when opcode field = SPECIAL.



rt bits 20..19	bits 18..16: 0 000	1 001	2 010	3 011	4 100	5 101	6 110	7 111
0 00	BLTZ$_1$	BGEZ$_1$	BLTZL$_2$	BGEZL$_2$	*$_1$	*$_1$	*$_1$	*$_1$
1 01	TGEI$_2$	TGEIU$_2$	TLTI$_2$	TLTIU$_2$	TEQI$_2$	*$_1$	TNEI$_2$	*$_1$
2 10	BLTZAL$_1$	BGEZAL$_1$	BLTZALL$_2$	BGEZALL$_2$	*$_1$	*$_1$	*$_1$	*$_1$
3 11	*$_1$	*$_1$	*$_1$	*$_1$	*$_1$	*$_1$	*$_1$	*$_1$

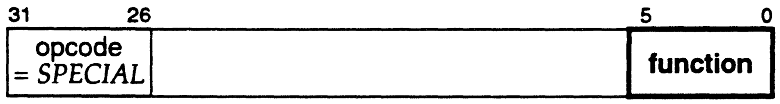
Instructions encoded by the **rt** field when opcode field = REGIMM.

Table A-42 CPU Instruction Encoding Changes - MIPS II Revision.

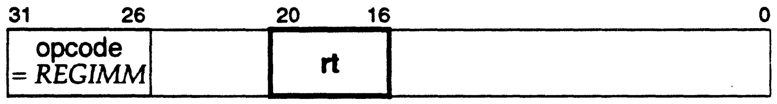


An instruction encoding is shown if the instruction is added in this revision.

opcode bits 28..26	Instructions encoded by opcode field.							
bits 31..29	0 000	1 001	2 010	3 011	4 100	5 101	6 110	7 111
0 000								
1 001								
2 010					BEQL	BNEL	BLEZL	BGTZL
3 011								
4 100								
5 101								ρ
6 110	LL					LDC1 π	LDC2 π	LDC3 π
7 111	SC					SDC1 π	SDC2 π	SDC3 π

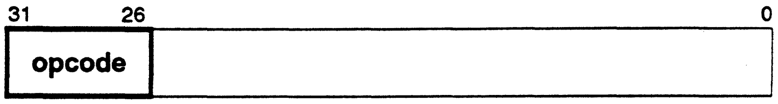


function bits 2..0	Instructions encoded by function field when opcode field = SPECIAL.							
bits 5..3	0 000	1 001	2 010	3 011	4 100	5 101	6 110	7 111
0 000								
1 001								SYNC
2 010								
3 011								
4 100								
5 101								
6 110	TGE	TGEU	TLT	TLTU	TEQ		TNE	
7 111								



rt bits 18..16	Instructions encoded by the rt field when opcode field = REGIMM.							
bits 20..19	0 000	1 001	2 010	3 011	4 100	5 101	6 110	7 111
0 00			BLTZL	BGEZL				
1 01	TGEI	TGEIU	TLTI	TLTIU	TEQI		TNEI	
2 10			BLTZALL	BGEZALL				
3 11								

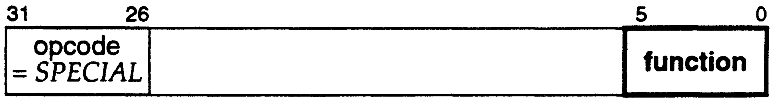
Table A-43 CPU Instruction Encoding Changes - MIPS III Revision.



An instruction encoding is shown if the instruction is added or modified in this revision.

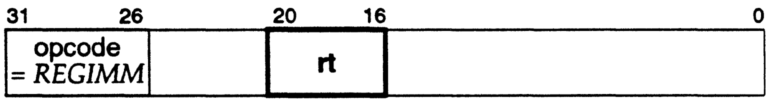
opcode bits 28..26 Instructions encoded by **opcode** field.

bits 31..29	0 000	1 001	2 010	3 011	4 100	5 101	6 110	7 111
0 000								
1 001								
2 010				* (was *COP3*)				
3 011	DADDI	DADDIU	LDL	LDR				
4 100								LWU
5 101					SDL	SDR		
6 110				* (was LWC3)	LLD			LD (was LDC3)
7 111				* (was SWC3)	SCD			SD (was SDC3)



function bits 2..0 Instructions encoded by **function** field when opcode field = SPECIAL.

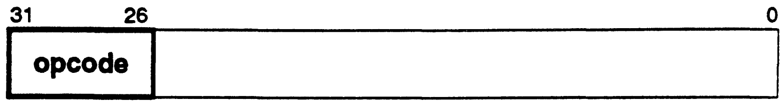
bits 5..3	0 000	1 001	2 010	3 011	4 100	5 101	6 110	7 111
0 000								
1 001								
2 010					DSLLV		DSRLV	DSRAV
3 011					DMULT	DMULTU	DDIV	DDIVU
4 100								
5 101					DADD	DADDU	DSUB	DSUBU
6 110								
7 111	DSLL		DSRL	DSRA	DSLL32		DSRL32	DSRA32



rt bits 18..16 Instructions encoded by the **rt** field when opcode field = REGIMM.

bits 20..19	0 000	1 001	2 010	3 011	4 100	5 101	6 110	7 111
0 00								
1 01								
2 10								
3 11								

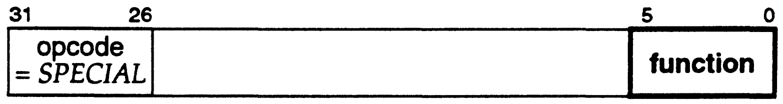
Table A-44 CPU Instruction Encoding Changes - MIPS IV Revision.



An instruction encoding is shown if the instruction is added or modified in this revision.

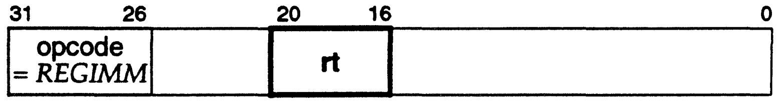
opcode bits 28..26 Instructions encoded by **opcode** field.

bits	0	1	2	3	4	5	6	7
31..29	000	001	010	011	100	101	110	111
0 000								
1 001								
2 010				*COP1X* δ,π				
3 011								
4 100								
5 101								
6 110				PREF				
7 111								



function bits 2..0 Instructions encoded by **function** field when opcode field = SPECIAL.

bits	0	1	2	3	4	5	6	7
5..3	000	001	010	011	100	101	110	111
0 000		*MOVCI* δ,μ						
1 001			MOVZ	MOVN				
2 010								
3 011								
4 100								
5 101								
6 110								
7 111								



rt bits 18..16 Instructions encoded by the **rt** field when opcode field = REGIMM.

bits	0	1	2	3	4	5	6	7
20..19	000	001	010	011	100	101	110	111
0 00								
1 01								
2 10								
3 11								

Key to notes in CPU instruction encoding tables:

*	This opcode is reserved for future use. An attempt to execute it causes a Reserved Instruction exception.
†	This opcode is reserved for future use. An attempt to execute it produces an undefined result. The result may be a Reserved Instruction exception but this is not guaranteed.
δ	(also *italic* opcode name) This opcode indicates an instruction class. The instruction word must be further decoded by examing additional tables that show values for another instruction field.
π	This opcode is a coprocessor operation, not a CPU operation. If the processor state does not allow access to the specified coprocessor, the instruction causes a Coprocessor Unusable exception. It is included in the table because it uses a primary opcode in the instruction encoding map.
κ	This opcode is removed in a later revision of the architecture. If a MIPS III or MIPS IV processor is operated in MIPS II-only mode this opcode will cause a Reserved Instruction exception.
μ	This opcode indicates a class of coprocessor 1 instructions. If the processor state does not allow access to coprocessor 1, the opcode causes a Coprocessor Unusable exception. It is included in the table because the encoding uses a location in what is otherwise a CPU instruction encoding map. Further encoding information for this instruction class is in the FPU Instruction Encoding tables.
ρ	This opcode is reserved for Coprocessor 0 (System Control Coprocessor) instructions that require base+offset addressing. If the instruction is used for COP0 in an implementation, an attempt to execute it without Coprocessor 0 access privilege will cause a Coprocessor Unusable exception. If the instruction is not used in an implementation, it will cause a Reserved Instruction exception.

B.11 FPU Instruction Formats

An FPU instruction is a single 32-bit aligned word. The distinct FP instruction layouts are shown in Figure B-16. Variable information is in lower-case labels, such as "offset". Upper-case labels and any numbers indicate constant data. A table follows all the layouts that explains the fields used in them. Note that the same field may have different names in different instruction layout pictures. The field name is mnemonic to the function of that field in the instruction layout. The opcode tables and the instruction decode discussion use the canonical field names: opcode, fmt, nd, tf, and function. The other fields are not used for instruction decode.

Figure B-16 FPU Instruction Formats

Immediate: load/store using register + offset addressing.

31 26	25 21	20 16	15 0
opcode	base	ft	offset
6	5	5	16

Register: 2-register and 3-register formatted arithmetic operations.

31 26	25 21	20 16	15 11	10 6	5 0
COP1	fmt	ft	fs	fd	function
6	5	5	5	5	6

Register Immediate: data transfer -- CPU ↔ FPU register.

31 26	25 21	20 16	15 11	10 0
COP1	sub	rt	fs	0
6	5	5	5	11

Condition code, Immediate: conditional branches on FPU cc using PC + offset.

31 26	25 21	20 18	17	16	15 0
COP1	BC	cc	nd	tf	offset
6	5	3	1	1	16

Register to Condition Code: formatted FP compare.

31 26	25 21	20 16	15 11	10 8	7 6	5 0
COP1	fmt	ft	fs	cc	0	function
6	5	5	5	3	2	4

Condition Code, Register FP: FPU register move-conditional on FP cc.

31 26	25 21	20 18	17	16	15 11	10 6	5 0
COP1	fmt	cc	0	tf	fs	fd	MOVCF
6	5	5	1	1	5	5	6

Register-4: 4-register formatted arithmetic operations.

31 26	25 21	20 16	15 11	10 6	5 3	2 0
					function	
COP1X	fr	ft	fs	fd	op4	fmt3
6	5	5	5	5	3	3

Register Index: Load/store using register + register addressing.

31 26	25 21	20 16	15 11	10 6	5 0
COP1X	base	index	0	fd	function
6	5	5	5	5	6

Register Index hint: Prefetch using register + register addressing.

31 26	25 21	20 16	15 11	10 6	5 0
COP1X	base	index	hint	0	PREFX
6	5	5	5	5	6

Condition Code, Register Integer: CPU register move-conditional on FP cc.

31 26	25 21	20 18	17	16	15 11	10 6	5 0
SPECIAL	rs	cc	0	tf	rd	0	MOVCI
6	5	5	1	1	5	5	6

BC	Branch Conditional instruction subcode (op=COP1)
base	CPU register: base address for address calculations
COP1	Coprocessor 1 primary opcode value in op field.
COP1X	Coprocessor 1 eXtended primary opcode value in op field.
cc	condition code specifier. For architecture levels prior to MIPS IV it must be zero.
fd	FPU register: destination (arithmetic, loads, move-to) or source (stores, move-from)
fmt	destination and/or operand type ("format") specifier
fr	FPU register: source
fs	FPU register: source
ft	FPU register: source (for stores, arithmetic) or destination (for loads)
function	function field specifying a function within a particular op operation code.
function: op4 + fmt3	op4 is a 3-bit function field specifying which 4-register arithmetic operation for COP1X, fmt3 is a 3-bit field specifying the format of the operands and destination. The combinations are shown as several distinct instructions in the opcode tables.
hint	hint field made available to cache controller for prefetch operation
index	CPU register, holds index address component for address calculations
MOVC	Value in function field for conditional move. There is one value for the instruction with op=COP1, another for the instruction with op=SPECIAL.
nd	nullify delay. If set, branch is Likely and delay slot instruction is not executed. This must be zero for MIPS I.
offset	signed offset field used in address calculations
op	primary operation code (COP1, COP1X, LWC1, SWC1, LDC1, SDC1, SPECIAL)
PREFX	Value in function field for prefetch instruction for op=COP1X
rd	CPU register: destination
rs	CPU register: source
rt	CPU register: source / destination
SPECIAL	SPECIAL primary opcode value in op field.
sub	Operation subcode field for COP1 register immediate mode instructions.
tf	true/false. The condition from FP compare is tested for equality with tf bit.

B.12 FPU (CP1) Instruction Opcode Bit Encoding

This section describes the encoding of the Floating-Point Unit (FPU) instructions for the four levels of the MIPS architecture, MIPS I through MIPS IV. Each architecture level includes the instructions in the previous level;[†] MIPS IV includes all instructions in MIPS I, MIPS II, and MIPS III. This section presents eight different views of the instruction encoding.

- Separate encoding tables for each architecture level.
- A MIPS IV encoding table showing the architecture level at which each opcode was originally defined and subsequently modified (if modified).
- Separate encoding tables for each architecture revision showing the changes made during that revision.

B 12.1 Instruction Decode

Instruction field names are printed in **bold** in this section.

The primary **opcode** field is decoded first. The **opcode** values LWC1, SWC1, LDC1, and SDC1 fully specify FPU load and store instructions. The **opcode** values *COP1*, *COP1X*, and *SPECIAL* specify instruction classes. Instructions within a class are further specified by values in other fields.

B 12.1.1 COP1 Instruction Class

The **opcode**=*COP1* instruction class encodes most of the FPU instructions. The class is further decoded by examining the **fmt** field. The **fmt** values fully specify the CPU $\leftrightarrow$ FPU register move instructions and specify the *S, D, W, L*, and *BC* instruction classes.

The **opcode**=*COP1* + **fmt**=*BC* instruction class encodes the conditional branch instructions. The class is further decoded, and the instructions fully specified, by examining the **nd** and **tf** fields.

The **opcode**=*COP1* + **fmt**=(*S, D, W*, or *L*) instruction classes encode instructions that operate on formatted (typed) operands. Each of these instruction classes is further decoded by examining the **function** field. With one exception the **function** values fully specify instructions. The exception is the *MOVCF* instruction class.

The **opcode**=*COP1* + **fmt**=(*S* or *D*) + **function**=*MOVCF* instruction class encodes the MOVT.*fmt* and MOVF.*fmt* conditional move instructions (to move FP values based on FP condition codes). The class is further decoded, and the instructions fully specified, by examining the **tf** field.

[†] An exception to this rule is that the reserved, but never implemented, Coprocessor 3 instructions were removed or changed to another use starting in MIPS III.

B 12.1.2 COP1X Instruction Class

The **opcode**=*COP1X* instruction class encodes the indexed load/store instructions, the indexed prefetch, and the multiply accumulate instructions. The class is further decoded, and the instructions fully specified, by examining the **function** field.

B 12.1.3 SPECIAL Instruction Class

The **opcode**=*SPECIAL* instruction class is further decoded by examining the **function** field. The only **function** value that applies to FPU instruction encoding is the *MOVCI* instruction class. The remainder of the **function** values encode CPU instructions.

The **opcode**=*SPECIAL* + **function**=*MOVCI* instruction class encodes the MOVT and MOVF conditional move instructions (to move CPU registers based on FP condition codes). The class is further decoded, and the instructions fully specified, by examining the **tf** field.

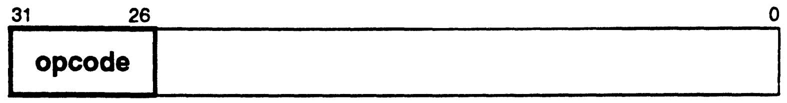
B 12.2 Instruction Subsets of MIPS III and MIPS IV Processors.

MIPS III processors, such as the R4000, R4200, R4300, R4400, and R4600, have a processor mode in which only the MIPS II instructions are valid. The MIPS II encoding table describes the MIPS II-only mode.

MIPS IV processors, such as the R8000 and R10000, have processor modes in which only the MIPS II or MIPS III instructions are valid. The MIPS II encoding table describes the MIPS II-only mode. The MIPS III encoding table describes the MIPS III-only mode.

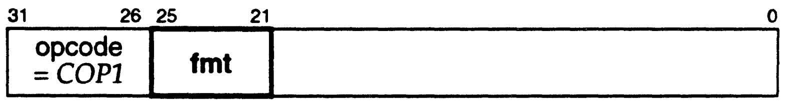
Table B-23 FPU (CP1) Instruction Encoding - MIPS I Architecture

Instructions encoded by the **opcode** field.



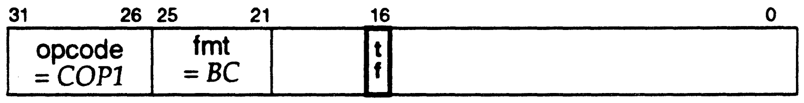
opcode bits 31..29	bits 28..26 0 000	1 001	2 010	3 011	4 100	5 101	6 110	7 111
0 000								
1 001								
2 010		*COP1* δ						
3 011								
4 100					χ			
5 101								
6 110		LWC1						
7 111		SWC1						

Instructions encoded by the **fmt** field when opcode=*COP1*.



fmt bits 25..24	bits 23..21 0 000	1 001	2 010	3 011	4 100	5 101	6 110	7 111
0 00	MFC1	*	CFC1	*	MTC1	*	CTC1	*
1 01	*BC* δ	*	*	*	*	*	*	*
2 10	*S* δ	*D* δ	*	*	*W* δ	*	*	*
3 11	*	*	*	*	*	*	*	*

Instructions encoded by the **tf** field when opcode=*COP1* and fmt=*BC*.



t f bit 16	
0	1
BC1F	BC1T

Table B-23 (cont.) FPU (CP1) Instruction Encoding - MIPS I Architecture

Instructions encoded by the **function** field when opcode=*COP1* and fmt = *S, D,* or *W*

encoding when fmt = *S*

31 26	25 21		0
opcode = *COP1*	fmt = *S*		**function**

function	bits 2..0	0	1	2	3	4	5	6	7
bits 5..3		000	001	010	011	100	101	110	111
0	000	ADD	SUB	MUL	DIV	*	ABS	MOV	NEG
1	001	*	*	*	*	*	*	*	*
2	010	*	*	*	*	*	*	*	*
3	011	*	*	*	*	*	*	*	*
4	100	*	CVT.D	*	*	CVT.W	*	*	*
5	101	*	*	*	*	*	*	*	*
6	110	C.F α	C.UN α	C.EQ α	C.UEQ α	C.OLT α	C.ULT α	C.OLE α	C.ULE α
7	111	C.SF α	C.NGLE α	C.SEQ α	C.NGL α	C.LT α	C.NGE α	C.LE α	C.NGT α

encoding when fmt = *D*

31 26	25 21		0
opcode = *COP1*	fmt = *D*		**function**

function	bits 2..0	0	1	2	3	4	5	6	7
bits 5..3		000	001	010	011	100	101	110	111
0	000	ADD	SUB	MUL	DIV	*	ABS	MOV	NEG
1	001	*	*	*	*	*	*	*	*
2	010	*	*	*	*	*	*	*	*
3	011	*	*	*	*	*	*	*	*
4	100	CVT.S	*	*	*	CVT.W	*	*	*
5	101	*	*	*	*	*	*	*	*
6	110	C.F α	C.UN α	C.EQ α	C.UEQ α	C.OLT α	C.ULT α	C.OLE α	C.ULE α
7	111	C.SF α	C.NGLE α	C.SEQ α	C.NGL α	C.LT α	C.NGE α	C.LE α	C.NGT α

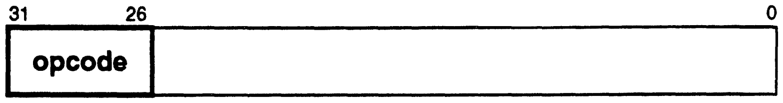
encoding when fmt = *W*

31 26	25 21		0
opcode = *COP1*	fmt = *W*		**function**

function	bits 2..0	0	1	2	3	4	5	6	7
bits 5..3		000	001	010	011	100	101	110	111
0	000	*	*	*	*	*	*	*	*
1	001	*	*	*	*	*	*	*	*
2	010	*	*	*	*	*	*	*	*
3	011	*	*	*	*	*	*	*	*
4	100	CVT.S	CVT.D	*	*	*	*	*	*
5	101	*	*	*	*	*	*	*	*
6	110	*	*	*	*	*	*	*	*
7	111	*	*	*	*	*	*	*	*

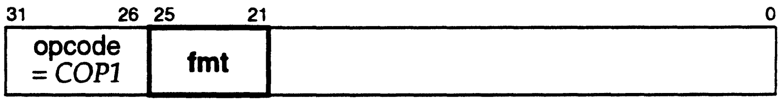
Table B-24 FPU (CP1) Instruction Encoding - MIPS II Architecture

Instructions encoded by the **opcode** field.



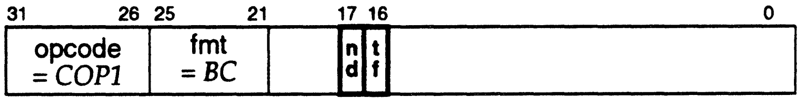
opcode bits 31..29 \ bits 28..26	0 000	1 001	2 010	3 011	4 100	5 101	6 110	7 111
0 000								
1 001								
2 010		*COP1 δ*						
3 011								
4 100				χ				
5 101								
6 110		LWC1				LDC1		
7 111		SWC1				SDC1		

Instructions encoded by the **fmt** field when opcode=*COP1*.



fmt bits 25..24 \ bits 23..21	0 000	1 001	2 010	3 011	4 100	5 101	6 110	7 111
0 00	MFC1	*	CFC1	*	MTC1	*	CTC1	*
1 01	*BC δ*	*	*	*	*	*	*	*
2 10	*S δ*	*D δ*	*	*	*W δ*	*	*	*
3 11	*	*	*	*	*	*	*	*

Instructions encoded by the **nd** and **tf** fields when opcode=*COP1* and fmt=*BC*.



nd bit 17 \ tf bit 16	0	1
0	BC1F	BC1T
1	BC1FL	BC1TL

Table B-24 (cont.) FPU (CP1) Instruction Encoding - MIPS II Architecture

Instructions encoded by the **function** field when opcode=*COP1* and fmt = *S, D,* or *W*

encoding when fmt = *S*

31 26	25 21		0
opcode = *COP1*	fmt = *S*		**function**

function bits 5..3	bits 2..0: 0 000	1 001	2 010	3 011	4 100	5 101	6 110	7 111
0 000	ADD	SUB	MUL	DIV	SQRT	ABS	MOV	NEG
1 001	*	*	*	*	ROUND.W	TRUNC.W	CEIL.W	FLOOR.W
2 010	*	*	*	*	*	*	*	*
3 011	*	*	*	*	*	*	*	*
4 100	*	CVT.D	*	*	CVT.W	*	*	*
5 101	*	*	*	*	*	*	*	*
6 110	C.F α	C.UN α	C.EQ α	C.UEQ α	C.OLT α	C.ULT α	C.OLE α	C.ULE α
7 111	C.SF α	C.NGLE α	C.SEQ α	C.NGL α	C.LT α	C.NGE α	C.LE α	C.NGT α

encoding when fmt = *D*

31 26	25 21		0
opcode = *COP1*	fmt = *D*		**function**

function bits 5..3	bits 2..0: 0 000	1 001	2 010	3 011	4 100	5 101	6 110	7 111
0 000	ADD	SUB	MUL	DIV	SQRT	ABS	MOV	NEG
1 001	*	*	*	*	ROUND.W	TRUNC.W	CEIL.W	FLOOR.W
2 010	*	*	*	*	*	*	*	*
3 011	*	*	*	*	*	*	*	*
4 100	CVT.S	*	*	*	CVT.W	*	*	*
5 101	*	*	*	*	*	*	*	*
6 110	C.F α	C.UN α	C.EQ α	C.UEQ α	C.OLT α	C.ULT α	C.OLE α	C.ULE α
7 111	C.SF α	C.NGLE α	C.SEQ α	C.NGL α	C.LT α	C.NGE α	C.LE α	C.NGT α

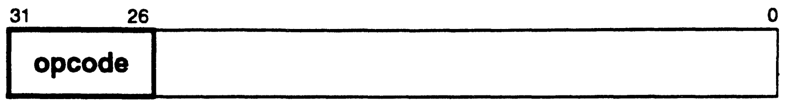
encoding when fmt = *W*

31 26	25 21		0
opcode = *COP1*	fmt = *W*		**function**

function bits 5..3	bits 2..0: 0 000	1 001	2 010	3 011	4 100	5 101	6 110	7 111
0 000	*	*	*	*	*	*	*	*
1 001	*	*	*	*	*	*	*	*
2 010	*	*	*	*	*	*	*	*
3 011	*	*	*	*	*	*	*	*
4 100	CVT.S	CVT.D	*	*	*	*	*	*
5 101	*	*	*	*	*	*	*	*
6 110	*	*	*	*	*	*	*	*
7 111	*	*	*	*	*	*	*	*

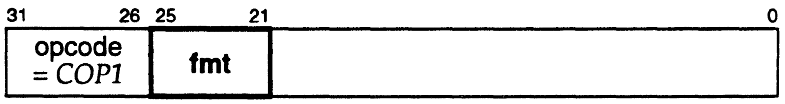
Table B-25 FPU (CP1) Instruction Encoding - MIPS III Architecture

Instructions encoded by the **opcode** field.



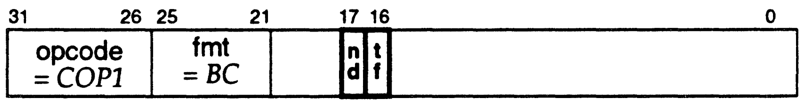
opcode bits 31..29 \ bits 28..26	0 000	1 001	2 010	3 011	4 100	5 101	6 110	7 111
0 000								
1 001								
2 010		*COP1 δ*						
3 011								
4 100				χ				
5 101								
6 110		LWC1				LDC1		
7 111		SWC1				SDC1		

Instructions encoded by the **fmt** field when opcode=*COP1*.



fmt bits 25..24 \ bits 23..21	0 000	1 001	2 010	3 011	4 100	5 101	6 110	7 111
0 00	MFC1	DMFC1	CFC1	*	MTC1	DMTC1	CTC1	*
1 01	*BC δ*	*	*	*	*	*	*	*
2 10	*S δ*	*D δ*	*	*	*W δ*	*L δ*	*	*
3 11	*	*	*	*	*	*	*	*

Instructions encoded by the **nd** and **tf** fields when opcode=*COP1* and fmt=*BC*.



nd bit 17 \ tf bit 16	0	1
0	BC1F	BC1T
1	BC1FL	BC1TL

Table B-25 (cont.) FPU (CP1) Instruction Encoding - MIPS III Architecture

Instructions encoded by the **function** field when opcode=*COP1* and fmt = *S, D, W,* or *L*

encoding when fmt = *S*

31 26	25 21		0
opcode = *COP1*	fmt = *S*		**function**

function bits 2..0 / bits 5..3	0 000	1 001	2 010	3 011	4 100	5 101	6 110	7 111
0 000	ADD	SUB	MUL	DIV	SQRT	ABS	MOV	NEG
1 001	ROUND.L	TRUNC.L	CEIL.L	FLOOR.L	ROUND.W	TRUNC.W	CEIL.W	FLOOR.W
2 010	*	*	*	*	*	*	*	
3 011	*	*	*	*	*	*	*	*
4 100	*	CVT.D	*	*	CVT.W	CVT.L	*	*
5 101	*	*	*	*	*	*	*	*
6 110	C.F α	C.UN α	C.EQ α	C.UEQ α	C.OLT α	C.ULT α	C.OLE α	C.ULE α
7 111	C.SF α	C.NGLE α	C.SEQ α	C.NGL α	C.LT α	C.NGE α	C.LE α	C.NGT α

encoding when fmt = *D*

31 26	25 21		0
opcode = *COP1*	fmt = *D*		**function**

function bits 2..0 / bits 5..3	0 000	1 001	2 010	3 011	4 100	5 101	6 110	7 111
0 000	ADD	SUB	MUL	DIV	SQRT	ABS	MOV	NEG
1 001	ROUND.L	TRUNC.L	CEIL.L	FLOOR.L	ROUND.W	TRUNC.W	CEIL.W	FLOOR.W
2 010	*	*	*	*	*	*	*	
3 011	*	*	*	*	*	*	*	*
4 100	CVT.S	*	*	*	CVT.W	CVT.L	*	*
5 101	*	*	*	*	*	*	*	*
6 110	C.F α	C.UN α	C.EQ α	C.UEQ α	C.OLT α	C.ULT α	C.OLE α	C.ULE α
7 111	C.SF α	C.NGLE α	C.SEQ α	C.NGL α	C.LT α	C.NGE α	C.LE α	C.NGT α

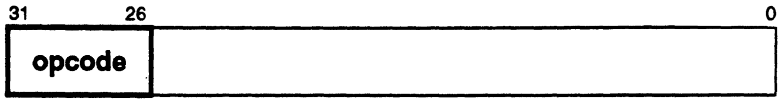
encoding when fmt = *W* or *L*

31 26	25 21		0
opcode = *COP1*	fmt = *W, L*		**function**

function bits 2..0 / bits 5..3	0 000	1 001	2 010	3 011	4 100	5 101	6 110	7 111
0 000	*	*	*	*	*	*	*	*
1 001	*	*	*	*	*	*	*	*
2 010	*	*	*	*	*	*	*	*
3 011	*	*	*	*	*	*	*	*
4 100	CVT.S	CVT.D	*	*	*	*	*	*
5 101	*	*	*	*	*	*	*	*
6 110	*	*	*	*	*	*	*	*
7 111	*	*	*	*	*	*	*	*

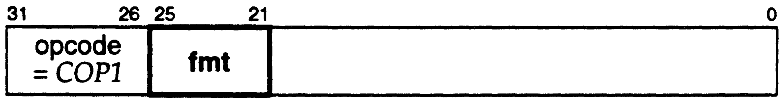
Table B-26 FPU (CP1) Instruction Encoding - MIPS IV Architecture

Instructions encoded by the **opcode** field.



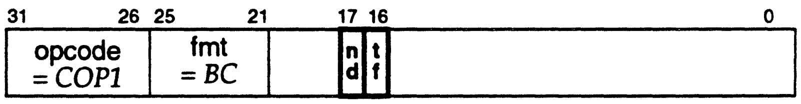
opcode bits 31..29 \ bits 28..26		0 000	1 001	2 010	3 011	4 100	5 101	6 110	7 111
0	000	*SPECIAL* δ, β							
1	001								
2	010		*COP1* δ		*COP1X* δ,λ				
3	011						χ		
4	100								
5	101								
6	110		LWC1				LDC1		
7	111		SWC1				SDC1		

Instructions encoded by the **fmt** field when opcode=*COP1*.



fmt bits 25..24 \ bits 23..21		0 000	1 001	2 010	3 011	4 100	5 101	6 110	7 111
0	00	MFC1	DMFC1	CFC1	*	MTC1	DMTC1	CTC1	*
1	01	*BC* δ	*	*	*	*	*	*	*
2	10	*S* δ	*D* δ	*	*	*W* δ	*L* δ	*	*
3	11	*	*	*	*	*	*	*	*

Instructions encoded by the **nd** and **tf** fields when opcode=*COP1* and fmt=*BC*.



nd bit 17 \ tf bit 16	0	1
0	BC1F	BC1T
1	BC1FL	BC1TL

Table B-26 (cont.) FPU (CP1) Instruction Encoding - MIPS IV Architecture

Instructions encoded by the **function** field when opcode=*COP1* and fmt = *S, D, W,* or *L*

encoding when fmt = *S*

31 26	25 21		0
opcode = *COP1*	fmt = *S*		**function**

function bits 5..3	bits 2..0 0 000	1 001	2 010	3 011	4 100	5 101	6 110	7 111
0 000	ADD	SUB	MUL	DIV	SQRT	ABS	MOV	NEG
1 001	ROUND.L	TRUNC.L	CEIL.L	FLOOR.L	ROUND.W	TRUNC.W	CEIL.W	FLOOR.W
2 010	*	*MOVCF* δ	MOVZ	MOVN	*	RECIP	RSQRT	
3 011	*	*	*	*	*	*	*	*
4 100	*	CVT.D	*	*	CVT.W	CVT.L	*	*
5 101	*	*	*	*	*	*	*	*
6 110	C.F α	C.UN α	C.EQ α	C.UEQ α	C.OLT α	C.ULT α	C.OLE α	C.ULE α
7 111	C.SF α	C.NGLE α	C.SEQ α	C.NGL α	C.LT α	C.NGE α	C.LE α	C.NGT α

encoding when fmt = *D*

31 26	25 21		0
opcode = *COP1*	fmt = *D*		**function**

function bits 5..3	bits 2..0 0 000	1 001	2 010	3 011	4 100	5 101	6 110	7 111
0 000	ADD	SUB	MUL	DIV	SQRT	ABS	MOV	NEG
1 001	ROUND.L	TRUNC.L	CEIL.L	FLOOR.L	ROUND.W	TRUNC.W	CEIL.W	FLOOR.W
2 010	*	*MOVCF* δ	MOVZ	MOVN	*	RECIP	RSQRT	
3 011	*	*	*	*	*	*	*	*
4 100	CVT.S	*	*	*	CVT.W	CVT.L	*	*
5 101	*	*	*	*	*	*	*	*
6 110	C.F α	C.UN α	C.EQ α	C.UEQ α	C.OLT α	C.ULT α	C.OLE α	C.ULE α
7 111	C.SF α	C.NGLE α	C.SEQ α	C.NGL α	C.LT α	C.NGE α	C.LE α	C.NGT α

encoding when fmt = *W* or *L*

31 26	25 21		0
opcode = *COP1*	fmt = *W, L*		**function**

function bits 5..3	bits 2..0 0 000	1 001	2 010	3 011	4 100	5 101	6 110	7 111
0 000	*	*	*	*	*	*	*	*
1 001	*	*	*	*	*	*	*	*
2 010	*	*	*	*	*	*	*	*
3 011	*	*	*	*	*	*	*	*
4 100	CVT.S	CVT.D	*	*	*	*	*	*
5 101	*	*	*	*	*	*	*	*
6 110	*	*	*	*	*	*	*	*
7 111	*	*	*	*	*	*	*	*

Table B-26 (cont.) FPU (CP1) Instruction Encoding - MIPS IV Architecture

Instructions encoded by the **function** field when opcode=*COP1X*.

31 26		5 0
opcode = *COP1X*		**function**

function	bits 2..0							
bits 5..3	0 000	1 001	2 010	3 011	4 100	5 101	6 110	7 111
0 000	LWXC1	LDXC1	*	*	*	*	*	*
1 001	SWXC1	SDXC1	*	*	*	*	*	PREFX
2 010	*	*	*	*	*	*	*	*
3 011	*	*	*	*	*	*	*	*
4 100	MADD.S	MADD.D	*	*	*	*	*	*
5 101	MSUB.S	MSUB.D	*	*	*	*	*	*
6 110	NMADD.S	NMADD.D	*	*	*	*	*	*
7 111	NMSUB.S	NMSUB.D	*	*	*	*	*	*

Instructions encoded by the **tf** field when opcode=*COP1*, fmt = *S* or *D*, and function=*MOVCF*.

31 26	25 21		16		5 0
opcode = *COP1*	fmt = *S, D*		tf		function = *MOVCF*

tf	bit 16	0	1
		MOVF (fmt)	MOVT (fmt)

These are the MOVF.fmt and MOVT.fmt instructions. They should not be confused with MOVF and MOVT.

Instruction class encoded by the **function** field when opcode=*SPECIAL*.

31 26		5 0
opcode = *SPECIAL*		**function**

function	bits 2..0							
bits 5..3	0 000	1 001	2 010	3 011	4 100	5 101	6 110	7 111
0 000		*MOVCI* δ						
...					χ			
7 111								

Instructions encoded by the **tf** field when opcode = *SPECIAL* and function=*MOVCI*.

31 26		16		5 0
opcode = *SPECIAL*		tf		function = *MOVCI*

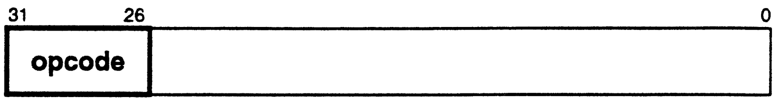
tf	bit 16	0	1
		MOVF	MOVT

These are the MOVF and MOVT instructions. They should not be confused with MOVF.fmt and MOVT.fmt.

Table B-27 Architecture Level In Which FPU Instructions are Defined or Extended.

The architecture level in which each MIPS IVencoding was defined is indicated by a subscript 1, 2, 3, or 4 (for architecture level I, II, III, or IV). If an instruction or instruction class was later extended, the extending level is indicated after the defining level.

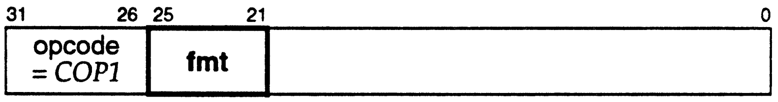
Instructions encoded by the **opcode** field.



opcode bits 28..26 Architecture level is shown by a subscript 1, 2, III, or 4.

bits		0	1	2	3	4	5	6	7
31..29		000	001	010	011	100	101	110	111
0	000	*SPECIAL* β_{4}							
1	001								
2	010		*COP1* $_{1,2,3,4}$		*COP1X* $_{4}$				
3	011						χ		
4	100								
5	101								
6	110		LWC1 $_{1}$				LDC1 $_{2}$		
7	111		SWC1 $_{1}$				SDC1 $_{2}$		

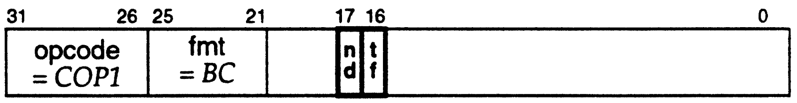
Instructions encoded by the **fmt** field when opcode=*COP1*.



fmt bits 23..21 Architecture level is shown by a subscript 1, 2, 3, or 4.

bits		0	1	2	3	4	5	6	7
25..24		000	001	010	011	100	101	110	111
0	00	MFC1 $_{1}$	DMFC1 $_{3}$	CFC1 $_{1}$	$*_{1}$	MTC1 $_{1}$	DMTC1 $_{3}$	CTC1 $_{1}$	$*_{1}$
1	01	*BC* $_{1,2,4}$	$*_{1}$	$*_{1}$	$*_{1}$	$*_{1}$	$*_{1}$	$*_{1}$	$*_{1}$
2	10	*S* $_{1,2,3,4}$	*D* $_{1,2,3,4}$	$*_{1}$	$*_{1}$	*W* $_{1,2,3,4}$	*L* $_{3,4}$	$*_{1}$	$*_{1}$
3	11	$*_{1}$	$*_{1}$	$*_{1}$	$*_{1}$	$*_{1}$	$*_{1}$	$*_{1}$	$*_{1}$

Instructions encoded by the **nd** and **tf** fields when opcode=*COP1* and fmt=*BC*.



Architecture level is shown by a subscript 1, 2, 3, or 4.

tf bit 16 / **nd** bit 17	0	1
0	BC1F $_{1,4}$	BC1T $_{1,4}$
1	BC1FL $_{2,4}$	BC1TL $_{2,4}$

Table B-27 (cont.) Architecture Level (I-IV) In Which FPU Instructions are Defined or Extended

Instructions encoded by the **function** field when opcode=*COP1* and fmt = *S, D, W,* or *L*

encoding when fmt = *S*

31 26	25 21		0
opcode = *COP1*	fmt = *S*		**function**

function bits 2..0 Architecture level is shown by a subscript 1, 2, 3, or 4.

bits	0	1	2	3	4	5	6	7
5..3	000	001	010	011	100	101	110	111
0 000	ADD$_1$	SUB$_1$	MUL$_1$	DIV$_1$	SQRT$_2$	ABS$_1$	MOV$_1$	NEG$_1$
1 001	ROUND.L$_3$	TRUNC.L$_3$	CEIL.L$_3$	FLOOR.L$_3$	ROUND.W$_2$	TRUNC.W$_2$	CEIL.W$_2$	FLOOR.W$_2$
2 010	*$_1$	*MOVCF*$_4$	MOVZ$_4$	MOVN$_4$	*$_1$	RECIP$_4$	RSQRT$_4$	*$_1$
3 011	*$_1$	*$_1$	*$_1$	*$_1$	*$_1$	*$_1$	*$_1$	*$_1$
4 100	*$_1$	CVT.D$_{1,3}$	*$_1$	*$_1$	CVT.W$_1$	CVT.L$_3$	*$_1$	*$_1$
5 101	*$_1$	*$_1$	*$_1$	*$_1$	*$_1$	*$_1$	*$_1$	*$_1$
6 110	C.F$_{1,4}$	C.UN$_{1,4}$	C.EQ$_{1,4}$	C.UEQ$_{1,4}$	C.OLT$_{1,4}$	C.ULT$_{1,4}$	C.OLE$_{1,4}$	C.ULE$_{1,4}$
7 111	C.SF$_{1,4}$	C.NGLE$_{1,4}$	C.SEQ$_{1,4}$	C.NGL$_{1,4}$	C.LT$_{1,4}$	C.NGE$_{1,4}$	C.LE$_{1,4}$	C.NGT$_{1,4}$

encoding when fmt = *D*

31 26	25 21		0
opcode = *COP1*	fmt = *D*		**function**

function bits 2..0 Architecture level is shown by a subscript 1, 2, 3, or 4.

bits	0	1	2	3	4	5	6	7
5..3	000	001	010	011	100	101	110	111
0 000	ADD$_1$	SUB$_1$	MUL$_1$	DIV$_1$	SQRT$_2$	ABS$_1$	MOV$_1$	NEG$_1$
1 001	ROUND.L$_3$	TRUNC.L$_3$	CEIL.L$_3$	FLOOR.L$_3$	ROUND.W$_2$	TRUNC.W$_2$	CEIL.W$_2$	FLOOR.W$_2$
2 010	*$_1$	*MOVCF*$_4$	MOVZ$_4$	MOVN$_4$	*$_1$	RECIP$_4$	RSQRT$_4$	*$_1$
3 011	*$_1$	*$_1$	*$_1$	*$_1$	*$_1$	*$_1$	*$_1$	*$_1$
4 100	CVT.S$_{1,3}$	*$_1$	*$_1$	*$_1$	CVT.W$_1$	CVT.L$_3$	*$_1$	*$_1$
5 101	*$_1$	*$_1$	*$_1$	*$_1$	*$_1$	*$_1$	*$_1$	*$_1$
6 110	C.F$_{1,4}$	C.UN$_{1,4}$	C.EQ$_{1,4}$	C.UEQ$_{1,4}$	C.OLT$_{1,4}$	C.ULT$_{1,4}$	C.OLE$_{1,4}$	C.ULE$_{1,4}$
7 111	C.SF$_{1,4}$	C.NGLE$_{1,4}$	C.SEQ$_{1,4}$	C.NGL$_{1,4}$	C.LT$_{1,4}$	C.NGE$_{1,4}$	C.LE$_{1,4}$	C.NGT$_{1,4}$

encoding when fmt = *W* or *L*

31 26	25 21		0
opcode = *COP1*	fmt = *W, L*		**function**

function bits 2..0 Architecture level is shown by a subscript 1, 2, 3, or 4.

bits	0	1	2	3	4	5	6	7
5..3	000	001	010	011	100	101	110	111
0 000	*$_1$	*$_1$	*$_1$	*$_1$	*$_1$	*$_1$	*$_1$	*$_1$
1 001	*$_1$	*$_1$	*$_1$	*$_1$	*$_1$	*$_1$	*$_1$	*$_1$
2 010	*$_1$	*$_1$	*$_1$	*$_1$	*$_1$	*$_1$	*$_1$	*$_1$
3 011	*$_1$	*$_1$	*$_1$	*$_1$	*$_1$	*$_1$	*$_1$	*$_1$
4 100	CVT.S$_{1,3}$	CVT.D$_{1,3}$	*$_1$	*$_1$	*$_1$	*$_1$	*$_1$	*$_1$
5 101	*$_1$	*$_1$	*$_1$	*$_1$	*$_1$	*$_1$	*$_1$	*$_1$
6 110	*$_1$	*$_1$	*$_1$	*$_1$	*$_1$	*$_1$	*$_1$	*$_1$
7 111	*$_1$	*$_1$	*$_1$	*$_1$	*$_1$	*$_1$	*$_1$	*$_1$

Table B-27 (cont.) Architecture Level (I-IV) In Which FPU Instructions are Defined or Extended

Instructions encoded by the **function** field when opcode=*COP1X*.

31 26		5 0
opcode = *COP1X*		**function**

Architecture level is shown by a subscript 1, 2, 3, or 4.

function bits 5..3 \ bits 2..0	0 000	1 001	2 010	3 011	4 100	5 101	6 110	7 111
0 000	LWXC1$_4$	LDXC1$_4$	$*_4$	$*_4$	$*_4$	$*_4$	$*_4$	$*_4$
1 001	SWXC1$_4$	SDXC1$_4$	$*_4$	$*_4$	$*_4$	$*_4$	$*_4$	PREFX$_4$
2 010	$*_4$	$*_4$	$*_4$	$*_4$	$*_4$	$*_4$	$*_4$	$*_4$
3 011	$*_4$	$*_4$	$*_4$	$*_4$	$*_4$	$*_4$	$*_4$	$*_4$
4 100	MADD.S$_4$	MADD.D$_4$	$*_4$	$*_4$	$*_4$	$*_4$	$*_4$	$*_4$
5 101	MSUB.S$_4$	MSUB.D$_4$	$*_4$	$*_4$	$*_4$	$*_4$	$*_4$	$*_4$
6 110	NMADD.S$_4$	NMADD.D$_4$	$*_4$	$*_4$	$*_4$	$*_4$	$*_4$	$*_4$
7 111	NMSUB.S$_4$	NMSUB.D$_4$	$*_4$	$*_4$	$*_4$	$*_4$	$*_4$	$*_4$

Instructions encoded by the **tf** field when opcode=*COP1*, fmt = *S* or *D*, and function=*MOVCF*.

31 26	25 21		16		5 0
opcode = *COP1*	fmt = *S, D*		tf		function = *MOVCF*

tf bit 16	0	1
	MOVF (fmt)$_4$	MOVT (fmt)$_4$

These are the MOVF.fmt and MOVT.fmt instructions. They should not be confused with MOVF and MOVT.

Instruction class encoded by the **function** field when opcode=*SPECIAL*.

31 26		5 0
opcode = *SPECIAL*		**function**

Architecture level is shown by a subscript 1, 2, 3, or 4.

function bits 5..3 \ bits 2..0	0 000	1 001	2 010	3 011	4 100	5 101	6 110	7 111
0 000		*MOVCI*$_4$						
...					χ			
7 111								

Instructions encoded by the **tf** field when opcode = *SPECIAL* and function=*MOVCI*.

31 26		16		5 0
opcode = *SPECIAL*		tf		function = *MOVCI*

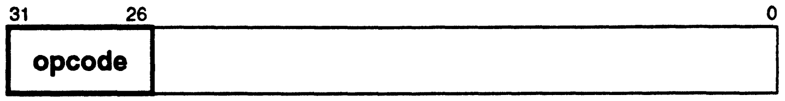
tf bit 16	0	1
	MOVF$_4$	MOVT$_4$

These are the MOVF and MOVT instructions. They should not be confused with MOVF.fmt and MOVT.fmt.

Table B-28 FPU Instruction Encoding Changes - MIPS II Architecture Revision.

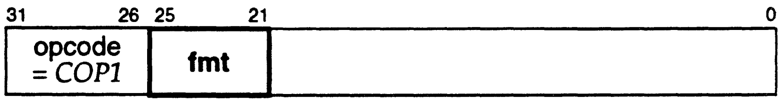
An instruction encoding is shown if the instruction is added or extended in this architecture revision. An instruction class, like COP1, is shown if the instruction class is added in this architecture revision.

Instructions encoded by the **opcode** field.



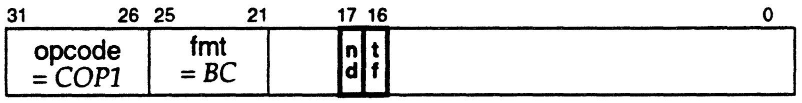
opcode bits 31..29 \ bits 28..26	0 000	1 001	2 010	3 011	4 100	5 101	6 110	7 111
0 000								
1 001								
2 010								
3 011								
4 100								
5 101								
6 110						LDC1		
7 111						SDC1		

Instructions encoded by the **fmt** field when opcode=*COP1*.



fmt bits 25..24 \ bits 23..21	0 000	1 001	2 010	3 011	4 100	5 101	6 110	7 111
0 00								
1 01								
2 10								
3 11								

Instructions encoded by the **nd** and **tf** fields when opcode=*COP1* and fmt=*BC*.



nd bit 17 \ **tf** bit 16	0	1
0		
1	BC1FL	BC1TL

Table B-28 (cont.) FPU Instruction Encoding Changes - MIPS II Revision.

Instructions encoded by the **function** field when opcode=*COP1* and fmt = *S, D,* or *W*

encoding when fmt = *S*

31 26	25 21		0
opcode = *COP1*	fmt = *S*		**function**

function	bits 2..0							
bits	0	1	2	3	4	5	6	7
5..3	000	001	010	011	100	101	110	111
0 000					SQRT			
1 001					ROUND.W	TRUNC.W	CEIL.W	FLOOR.W
2 010								
3 011								
4 100								
5 101								
6 110								
7 111								

encoding when fmt = *D*

31 26	25 21		0
opcode = *COP1*	fmt = *D*		**function**

function	bits 2..0							
bits	0	1	2	3	4	5	6	7
5..3	000	001	010	011	100	101	110	111
0 000					SQRT			
1 001					ROUND.W	TRUNC.W	CEIL.W	FLOOR.W
2 010								
3 011								
4 100								
5 101								
6 110								
7 111								

encoding when fmt = *W*

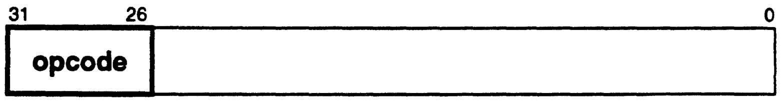
31 26	25 21		0
opcode = *COP1*	fmt = *W*		**function**

function	bits 2..0							
bits	0	1	2	3	4	5	6	7
5..3	000	001	010	011	100	101	110	111
0 000								
1 001								
2 010								
3 011								
4 100								
5 101								
6 110								
7 111								

Table B-29 FPU Instruction Encoding Changes - MIPS III Revision.

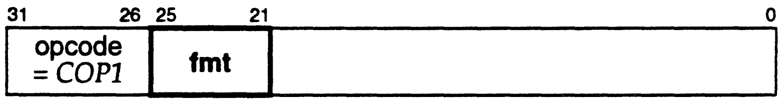
An instruction encoding is shown if the instruction is added or extended in this architecture revision. An instruction class, like COP1, is shown if the instruction class is added in this architecture revision.

Instructions encoded by the **opcode** field.



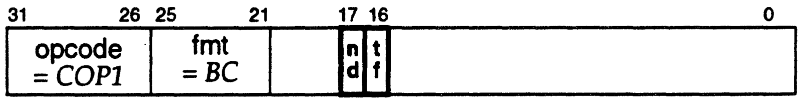
opcode bits 31..29	bits 28..26: 0 000	1 001	2 010	3 011	4 100	5 101	6 110	7 111
0 000								
1 001								
2 010								
3 011								
4 100								
5 101								
6 110								
7 111								

Instructions encoded by the **fmt** field when opcode=*COP1*.



fmt bits 25..24	bits 23..21: 0 000	1 001	2 010	3 011	4 100	5 101	6 110	7 111
0 00		DMFC1				DMTC1		
1 01								
2 10						*L* δ		
3 11								

Instructions encoded by the **nd** and **tf** fields when opcode=*COP1* and fmt=*BC*.



nd bit 17	**tf** bit 16: 0	1
0		
1	BC1FL	BC1TL

Table B-29 (cont.) FPU Instruction Encoding Changes - MIPS III Revision.

Instructions encoded by the **function** field when opcode=*COP1* and fmt = *S*, *D*, or *L*.

encoding when fmt = *S*

31 ... 26	25 ... 21		... 0
opcode = *COP1*	fmt = *S*		**function**

function bits 2..0 bits 5..3	0 000	1 001	2 010	3 011	4 100	5 101	6 110	7 111
0 000								
1 001	ROUND.L	TRUNC.L	CEIL.L	FLOOR.L				
2 010								
3 011								
4 100						CVT.L		
5 101								
6 110								
7 111								

encoding when fmt = *D*

31 ... 26	25 ... 21		... 0
opcode = *COP1*	fmt = *D*		**function**

function bits 2..0 bits 5..3	0 000	1 001	2 010	3 011	4 100	5 101	6 110	7 111
0 000								
1 001	ROUND.L	TRUNC.L	CEIL.L	FLOOR.L				
2 010								
3 011								
4 100						CVT.L		
5 101								
6 110								
7 111								

encoding when fmt = *L*

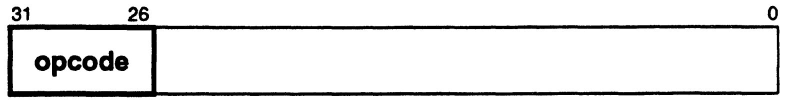
31 ... 26	25 ... 21		... 0
opcode = *COP1*	fmt = *L*		**function**

function bits 2..0 bits 5..3	0 000	1 001	2 010	3 011	4 100	5 101	6 110	7 111
0 000	*	*	*	*	*	*	*	*
1 001	*	*	*	*	*	*	*	*
2 010	*	*	*	*	*	*	*	*
3 011	*	*	*	*	*	*	*	*
4 100	CVT.S	CVT.D	*	*	*	*	*	*
5 101	*	*	*	*	*	*	*	*
6 110	*	*	*	*	*	*	*	*
7 111	*	*	*	*	*	*	*	*

Table B-30 FPU Instruction Encoding Changes - MIPS IV Revision.

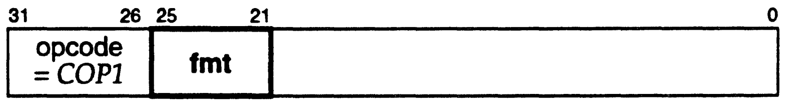
An instruction encoding is shown if the instruction is added or extended in this architecture revision. An instruction class, like COP1X, is shown if the instruction class is added in this architecture revision.

Instructions encoded by the **opcode** field.



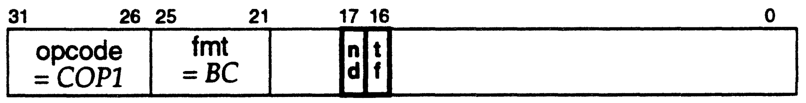
opcode bits 31..29		bits 28..26 0 000	1 001	2 010	3 011	4 100	5 101	6 110	7 111
0	000								
1	001								
2	010				*COP1X* δ				
3	011								
4	100								
5	101								
6	110								
7	111								

Instructions encoded by the **fmt** field when opcode=*COP1*.



fmt bits 25..24		bits 23..21 0 000	1 001	2 010	3 011	4 100	5 101	6 110	7 111
0	00								
1	01								
2	10								
3	11								

Instructions encoded by the **nd** and **tf** fields when opcode=*COP1* and fmt=*BC*.



nd bit 17	**tf** bit 16 0	1
0	BC1F	BC1T
1	BC1FL	BC1TL

Table B-30 (cont.) FPU Instruction Encoding Changes - MIPS IV Revision.

Instructions encoded by the **function** field when opcode=*COP1* and fmt = *S, D, W,* or *L.*

encoding when fmt = *S*

31 26	25 21		0
opcode = *COP1*	fmt = *S*		**function**

function bits 5..3 \ bits 2..0	0 000	1 001	2 010	3 011	4 100	5 101	6 110	7 111
0 000								
1 001								
2 010		*MOVCF* δ	MOVZ	MOVN		RECIP	RSQRT	
3 011								
4 100								
5 101								
6 110	C.F	C.UN	C.EQ	C.UEQ	C.OLT	C.ULT	C.OLE	C.ULE
7 111	C.SF	C.NGLE	C.SEQ	C.NGL	C.LT	C.NGE	C.LE	C.NGT

encoding when fmt = *D*

31 26	25 21		0
opcode = *COP1*	fmt = *D*		**function**

function bits 5..3 \ bits 2..0	0 000	1 001	2 010	3 011	4 100	5 101	6 110	7 111
0 000								
1 001								
2 010		*MOVCF* δ	MOVZ	MOVN		RECIP	RSQRT	
3 011								
4 100								
5 101								
6 110	C.F	C.UN	C.EQ	C.UEQ	C.OLT	C.ULT	C.OLE	C.ULE
7 111	C.SF	C.NGLE	C.SEQ	C.NGL	C.LT	C.NGE	C.LE	C.NGT

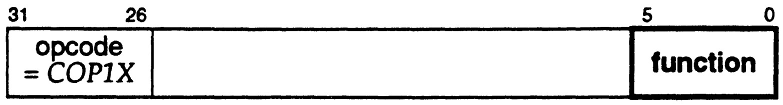
encoding when fmt = *W* or *L*

31 26	25 21		0
opcode = *COP1*	fmt = *W, L*		**function**

function bits 5..3 \ bits 2..0	0 000	1 001	2 010	3 011	4 100	5 101	6 110	7 111
0 000								
1 001								
2 010								
3 011								
4 100								
5 101								
6 110								
7 111								

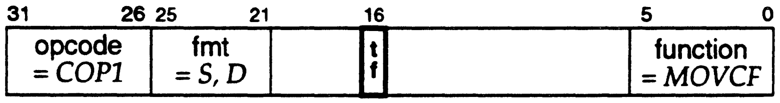
Table B-30 (cont.) FPU Instruction Encoding Changes - MIPS IV Revision.

Instructions encoded by the **function** field when opcode=*COP1X*.



function bits 5..3	bits 2..0: 0 000	1 001	2 010	3 011	4 100	5 101	6 110	7 111
0 000	LWXC1	LDXC1	*	*	*	*	*	*
1 001	SWXC1	SDXC1	*	*	*	*	*	PREFX
2 010	*	*	*	*	*	*	*	*
3 011	*	*	*	*	*	*	*	*
4 100	MADD.S	MADD.D	*	*	*	*	*	*
5 101	MSUB.S	MSUB.D	*	*	*	*	*	*
6 110	NMADD.S	NMADD.D	*	*	*	*	*	*
7 111	NMSUB.S	NMSUB.D	*	*	*	*	*	*

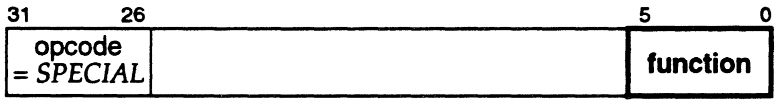
Instructions encoded by the **tf** field when opcode=*COP1*, fmt = *S* or *D*, and function=*MOVCF*.



tf bit 16	0	1
	MOVF (fmt)	MOVT (fmt)

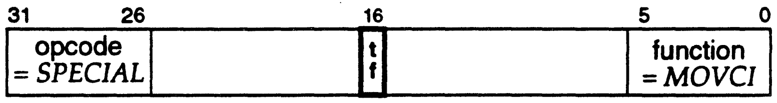
These are the MOVF.fmt and MOVT.fmt instructions. They should not be confused with MOVF and MOVT.

Instruction class encoded by the **function** field when opcode=*SPECIAL*.



function bits 5..3	bits 2..0: 0 000	1 001	2 010	3 011	4 100	5 101	6 110	7 111
0 000		*MOVCI* δ						
...						χ		
7 111								

Instructions encoded by the **tf** field when opcode = *SPECIAL* and function=*MOVCI*.



tf bit 16	0	1
	MOVF	MOVT

These are the MOVF and MOVT instructions. They should not be confused with MOVF.fmt and MOVT.fmt.

Key to all FPU (CP1) instruction encoding tables:

* This opcode is reserved for future use. An attempt to execute it causes either a Reserved Instruction exception or a Floating Point Unimplemented Operation Exception. The choice of exception is implementation specific.

α The table shows 16 compare instructions with values named C.condition where "condition" is a comparison condition such as "EQ". These encoding values are all documented in the instruction description titled "C.cond.fmt".

β The SPECIAL instruction class was defined in MIPS I for CPU instructions. An FPU instruction was first added to the instruction class in MIPS IV.

δ (also *italic* opcode name) This opcode indicates an instruction class. The instruction word must be further decoded by examing additional tables that show values for another instruction field.

λ The *COP1X* opcode in MIPS IV was the COP3 opcode in MIPS I and II and a reserved instruction in MIPS III.

χ These opcodes are not FPU operations. For further information on them, look in the CPU Instruction Encoding information section A 8.

(fmt) This opcode is a conditional move of formatted FP registers - either MOVF.D, MOVF.S, MOVT.D, or MOVT.S. It should not be confused with the similarly-named MOVF or MOVT instruction that moves CPU registers.

NEC Electronics Inc.

CORPORATE HEADQUARTERS

475 Ellis Street
P.O. Box 7241
Mountain View, CA 94039
TEL 415-960-6000

For literature, call toll-free 7 a.m. to 6 p.m. Pacific time: **1-800-366-9782**
or FAX your request to: **1-800-729-9288**