FPU Instruction Set Details

B

This appendix provides a detailed description of each floating-point unit (FPU) instruction (refer to Appendix A for a detailed description of the CPU instructions). The instructions are listed alphabetically, and any exceptions that may occur due to the execution of each instruction are listed after the description of each instruction. Descriptions of the immediate causes and the manner of handling exceptions are omitted from the instruction descriptions in this appendix (refer to Chapter 7 for detailed descriptions of floating-point exceptions and handling).

Figure B-3 at the end of this appendix lists the entire bit encoding for the constant fields of the floating-point instruction set; the bit encoding for each instruction is included with that individual instruction.

B.1 Instruction Formats

There are three basic instruction format types:

- I-Type, or Immediate instructions, which include load and store operations
- M-Type, or Move instructions
- R-Type, or Register instructions, which include the two- and three-register floating-point operations.

The instruction description subsections that follow show how these three basic instruction formats are used by:

- Load and store instructions
- Move instructions
- Floating-Point computational instructions
- Floating-Point branch instructions

Floating-point instructions are mapped onto the MIPS coprocessor instructions, defining coprocessor unit number one (CP1) as the floating-point unit.

Each operation is valid only for certain formats. Implementations may support some of these formats and operations through emulation, but they only need to support combinations that are valid (marked *V* in Table B-1). Combinations marked *R* in Table B-1 are not currently specified by this architecture, and cause an unimplemented operation trap. They will be available for future extensions to the architecture.

Table B-1 Valid FPU Instruction Formats

Operation	Source Format			
	Single	**Double**	**Word**	**Longword**
ADD	V	V	R	R
SUB	V	V	R	R
MUL	V	V	R	R
DIV	V	V	R	R
SQRT	V	V	R	R
ABS	V	V	R	R
MOV	V	V		
NEG	V	V	R	R
TRUNC.L	V	V		
ROUND.L	V	V		
CEIL.L	V	V		
FLOOR.L	V	V		
TRUNC.W	V	V		
ROUND.W	V	V		
CEIL.W	V	V		
FLOOR.W	V	V		
CVT.S		V	V	V
CVT.D	V		V	V
CVT.W	V	V		
CVT.L	V	V		
C	V	V	R	R

The coprocessor branch on condition true/false instructions can be used to logically negate any predicate. Thus, the 32 possible conditions require only 16 distinct comparisons, as shown in Table B-2 below.

Table B-2 Logical Negation of Predicates by Condition True/False

Condition			**Relations**				**Invalid Operation Exception If Unordered**
Mnemonic		**Code**	**Greater Than**	**Less Than**	**Equal**	**Unordered**	
True	**False**						
F	T	0	F	F	F	F	No
UN	OR	1	F	F	F	T	No
EQ	NEQ	2	F	F	T	F	No
UEQ	OGL	3	F	F	T	T	No
OLT	UGE	4	F	T	F	F	No
ULT	OGE	5	F	T	F	T	No
OLE	UGT	6	F	T	T	F	No
ULE	OGT	7	F	T	T	T	No
SF	ST	8	F	F	F	F	Yes
NGLE	GLE	9	F	F	F	T	Yes
SEQ	SNE	10	F	F	T	F	Yes
NGL	GL	11	F	F	T	T	Yes
LT	NLT	12	F	T	F	F	Yes
NGE	GE	13	F	T	F	T	Yes
LE	NLE	14	F	T	T	F	Yes
NGT	GT	15	F	T	T	T	Yes

Floating-Point Loads, Stores, and Moves

All movement of data between the floating-point coprocessor and memory is accomplished by coprocessor load and store operations, which reference the floating-point coprocessor *General Purpose* registers. These operations are unformatted; no format conversions are performed and, therefore, no floating-point exceptions can occur due to these operations.

Data may also be directly moved between the floating-point coprocessor and the processor by *move to coprocessor* and *move from coprocessor* instructions. Like the floating-point load and store operations, move to/from operations perform no format conversions and never cause floating-point exceptions.

An additional pair of coprocessor registers are available, called *Floating-Point Control* registers for which the only data movement operations supported are moves to and from processor *General Purpose* registers.

Floating-Point Operations

The floating-point unit operation set includes:

- floating-point add
- floating-point subtract
- floating-point multiply
- floating-point divide
- floating-point square root
- convert between fixed-point and floating-point formats
- convert between floating-point formats
- floating-point compare

These operations satisfy the requirements of IEEE Standard 754 requirements for accuracy. Specifically, these operations obtain a result which is identical to an infinite-precision result rounded to the specified format, using the current rounding mode.

Instructions must specify the format of their operands. Except for conversion functions, mixed-format operations are not provided.

B.2 Instruction Notation Conventions

In this appendix, all variable subfields in an instruction format (such as *fs, ft, immediate*, and so on) are shown in lower-case. The instruction name (such as ADD, SUB, and so on) is shown in upper-case.

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

In some instructions, the instruction subfields *op* and *function* can have constant 6-bit values. When reference is made to these instructions, upper-case mnemonics are used. For instance, in the floating-point ADD instruction we use *op* = COP1 and *function* = ADD. In other cases, a single field has both fixed and variable subfields, so the name contains both upper and lower case characters. Bit encodings for mnemonics are shown in Figure B-3 at the end of this appendix, and are also included with each individual instruction.

In the instruction description examples that follow, the *Operation* section describes the operation performed by each instruction using a high-level language notation.

Instruction Notation Examples

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

Example #1:
$\mathrm{GPR[rt]} \leftarrow \mathrm{immediate} \mathbin{\|\|} 0^{16}$
Sixteen zero bits are concatenated with an immediate value (typically 16 bits), and the 32-bit string (with the lower 16 bits set to zero) is assigned to General Purpose Register *rt*.
Example #2:
$(\mathrm{immediate}_{15})^{16} \mathbin{\|\|} \mathrm{immediate}_{15...0}$
Bit 15 (the sign bit) of an immediate value is extended for 16 bit positions, and the result is concatenated with bits 15 through 0 of the immediate value to form a 32-bit sign extended value.

B.3 Load and Store Instructions

In the R4000 implementation, the instruction immediately following a load may use the contents of the register being loaded. In such cases, the hardware *interlocks*, requiring additional real cycles, so scheduling load delay slots is still desirable, although not required for functional code.

The behavior of the load store instructions is dependent on the width of the *FGR*s.

- When the *FR* bit in the *Status* register equals zero, the *Floating-Point General* registers (*FGR*s) are 32-bits wide.
- When the *FR* bit in the *Status* register equals one, the *Floating-Point General* registers (*FGR*s) are 64-bits wide.

In the load and store operation descriptions, the functions listed in Table B-3 are used to summarize the handling of virtual addresses and physical memory.

Table B-3 Load and Store Common Functions

Function	Meaning
AddressTranslation	Uses the TLB to find the physical address given the virtual address. The function fails and an exception is taken if the required translation is not present in the TLB.
LoadMemory	Uses the cache and main memory to find the contents of the word containing the specified physical address. The low-order two bits of the address and the *Access Type* field indicates which of each of the four bytes within the data word need to be returned. If the cache is enabled for this access, the entire word is returned and loaded into the cache.
StoreMemory	Uses the cache, write buffer, and main memory to store the word or part of word specified as data in the word containing the specified physical address. The low-order two bits of the address and the *Access Type* field indicates which of each of the four bytes within the data word should be stored.

Figure B-1 shows the I-Type instruction format used by load and store operations.

I-Type (Immediate)

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

op	is a 6-bit operation code
base	is the 5-bit base register specifier
ft	is a 5-bit source (for stores) or destination (for loads) FPA register specifier
offset	is the 16-bit signed immediate offset

Figure B-1 Load and Store Instruction Format

All coprocessor loads and stores reference aligned data items. Thus, for word loads and stores, the access type field is always WORD, and the low-order two bits of the address must always be zero.

For doubleword loads and stores, the access type field is always DOUBLEWORD, and the low-order three bits of the address must always be zero.

Regardless of byte-numbering order (endianness), the address specifies that byte which has the smallest byte-address in the addressed field. For a big-endian machine, this is the leftmost byte; for a little-endian machine, this is the rightmost byte.

B.4 Computational Instructions

Computational instructions include all of the arithmetic floating-point operations performed by the FPU.

Figure B-2 shows the R-Type instruction format used for computational operations.

R-Type (Register)

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

COP1	is a 6-bit operation code
fmt	is a 5-bit format specifier
fs	is a 5-bit source1 register
ft	is a 5-bit source2 register
fd	is a 5-bit destination register
function	is a 6-bit function field

Figure B-2 Computational Instruction Format

The *function* field indicates the floating-point operation to be performed.

Each floating-point instruction can be applied to a number of operand *formats*. The operand format for an instruction is specified by the 5-bit *format* field; decoding for this field is shown in Table B-4.

Table B-4 Format Field Decoding

Code	Mnemonic	Size	Format
16	S	single	Binary floating-point
17	D	double	Binary floating-point
18	Reserved		
19	Reserved		
20	W	single	32-bit binary fixed-point
21	L	longword	64-bit binary fixed-point
22–31	Reserved		

Table B-5 lists all floating-point instructions.

Table B-5 Floating-Point Instructions and Operations

Code (5: 0)	Mnemonic	Operation
0	ADD	Add
1	SUB	Subtract
2	MUL	Multiply
3	DIV	Divide
4	SQRT	Square root
5	ABS	Absolute value
6	MOV	Move
7	NEG	Negate
8	ROUND.L	Convert to 64-bit (long) fixed-point, rounded to nearest/ even
9	TRUNC.L	Convert to 64-bit (long) fixed-point, rounded toward zero
10	CEIL.L	Convert to 64-bit (long) fixed-point, rounded to +∞
11	FLOOR.L	Convert to 64-bit (long) fixed-point, rounded to -∞
12	ROUND.W	Convert to single fixed-point, rounded to nearest/even
13	TRUNC.W	Convert to single fixed-point, rounded toward zero
14	CEIL.W	Convert to single fixed-point, rounded to + ∞
15	FLOOR.W	Convert to single fixed-point, rounded to – ∞
16–31	–	Reserved
32	CVT.S	Convert to single floating-point
33	CVT.D	Convert to double floating-point
34	–	Reserved
35	–	Reserved
36	CVT.W	Convert to 32-bit binary fixed-point
37	CVT.L	Convert to 64-bit (long) binary fixed-point
38–47	–	Reserved
48–63	C	Floating-point compare

In the following pages, the notation *FGR* refers to the 32 *General Purpose* registers *FGR0* through *FGR31* of the FPU, and *FPR* refers to the floating-point registers of the FPU.

- When the *FR* bit in the *Status* register (SR(26)) equals zero, only the even floating-point registers are valid and the 32 *General Purpose* registers of the FPU are 32-bits wide.
- When the *FR* bit in the *Status* register (SR(26)) equals one, both odd and even floating-point registers may be used and the 32 *General Purpose* registers of the FPU are 64-bits wide.

The following routines are used in the description of the floating-point operations to retrieve the value of an FPR or to change the value of an FGR:

value ⟵ ValueFPR(fpr,fmt)

```
if SR26 = 1 then  /* 64-bit wide FGRs */
    case fmt of
        S, W:
            value ⟵ FGR[fpr]31...0
            return
        D, L:
            value ⟵ FGR[fpr]
            return
    endcase
elseif fpr0 = 0 then   /* valid specifier, 32-bit wide FGRs */
    case fmt of
        S, W:
            value ⟵ FGR[fpr]
            return
        D, L:
            value ⟵ FGR[fpr+1] || FGR[fpr]
            return
    endcase
else   /* undefined result for odd 32-bit reg #s */
    value ⟵ undefined
endif
```

StoreFPR(fpr, fmt, value)

```
if SR26 = 1 then  /* 64-bit wide FGRs */
      case fmt of
            S, W:
                  FGR[fpr] ⟵ undefined^32 || value
                  return
            D, L:
                  FGR[fpr] ⟵ value
                  return
      endcase
elseif fpr0 = 0 then   /* valid specifier, 32-bit wide FGRs */
      case fmt of
            S, W:
                  FGR[fpr+1] ⟵ undefined
                  FGR[fpr] ⟵ value
                  return
            D, L:
                  FGR[fpr+1] ⟵ value63...32
                  FGR[fpr] ⟵ value31...0
                  return
      endcase
else   /* undefined result for odd 32-bit reg #s */
      undefined_result
endif
```

ABS.fmt — Floating-Point Absolute Value — ABS.fmt

31–26	25–21	20–16	15–11	10–6	5–0
COP1 0 1 0 0 0 1	fmt	0 0 0 0 0 0	fs	fd	ABS 0 0 0 1 0 1
6	5	5	5	5	6

Format:

ABS.fmt fd, fs

Description:

The contents of the FPU register specified by *fs* are interpreted in the specified format and the arithmetic absolute value is taken. The result is placed in the floating-point register specified by *fd*.

The absolute value operation is arithmetic; a NaN operand signals invalid operation.

This instruction is valid only for single- and double-precision floating-point formats. The operation is not defined if bit 0 of any register specification is set and the *FR* bit in the *Status* register equals zero, since the register numbers specify an even-odd pair of adjacent coprocessor general registers. When the *FR* bit in the *Status* register equals one, both even and odd register numbers are valid.

Operation:

```
T:    StoreFPR(fd, fmt, AbsoluteValue(ValueFPR(fs, fmt)))
```

Exceptions:

Coprocessor unusable exception
Coprocessor exception trap

Coprocessor Exceptions:

Unimplemented operation exception
Invalid operation exception

ADD.fmt Floating-Point Add ADD.fmt

31–26	25–21	20–16	15–11	10–6	5–0
COP1 0 1 0 0 0 1	fmt	ft	fs	fd	ADD 0 0 0 0 0 0
6	5	5	5	5	6

Format:

ADD.fmt fd, fs, ft

Description:

The contents of the FPU registers specified by *fs* and *ft* are interpreted in the specified format and arithmetically added. The result is rounded as if calculated to infinite precision and then rounded to the specified format (*fmt*), according to the current rounding mode. The result is placed in the floating-point register (*FPR*) specified by *fd*.

This instruction is valid only for single- and double-precision floating-point formats. The operation is not defined if bit 0 of any register specification is set and the *FR* bit in the *Status* register equals zero, since the register numbers specify an even-odd pair of adjacent coprocessor general registers. When the *FR* bit in the *Status* register equals one, both even and odd register numbers are valid.

Operation:

```
T:    StoreFPR (fd, fmt, ValueFPR(fs, fmt) + ValueFPR(ft, fmt))
```

Exceptions:

Coprocessor unusable exception
Floating-Point exception

Coprocessor Exceptions:

Unimplemented operation exception
Invalid operation exception
Inexact exception
Overflow exception
Underflow exception

BC1F — Branch On FPA False (Coprocessor 1) — BC1F

31 26	25 21	20 16	15 0
COP1 0 1 0 0 0 1	BC 0 1 0 0 0	BCF 0 0 0 0 0	offset
6	5	5	16

Format:

BC1F offset

Description:

A branch target address is computed from the sum of the address of the instruction in the delay slot and the 16-bit *offset*, shifted left two bits and sign-extended. If the result of the last floating-point compare is false (zero), the program branches to the target address, with a delay of one instruction.

There must be at least one instruction between C.cond.fmt and BC1F.

Operation:

32	T–1:	condition ← not COC[1]
	T:	target ← $(\text{offset}_{15})^{14}$ || offset || 0^2

	T+1:	if condition then
		    PC ← PC + target
		endif

64	T–1:	condition ← not COC[1]
	T:	target ← $(\text{offset}_{15})^{46}$ || offset || 0^2

	T+1:	if condition then
		    PC ← PC + target
		endif

Exceptions:

Coprocessor unusable exception

BC1FL Branch On FPU False Likely (Coprocessor 1) BC1FL

31–26	25–21	20–16	15–0
COP1 0 1 0 0 0 1	BC 0 1 0 0 0	BCFL 0 0 0 1 0	offset
6	5	5	16

Format:

BC1FL offset

Description:

A branch target address is computed from the sum of the address of the instruction in the delay slot and the 16-bit *offset*, shifted left two bits and sign-extended. If the result of the last floating-point compare is false (zero), the program branches to the target address, with a delay of one instruction. If the conditional branch is not taken, the instruction in the branch delay slot is nullified.

There must be at least one instruction between C.cond.fmt and BC1FL.

Operation:

```
32    T–1:   condition ← not COC[1]
      T:     target ← (offset15)^14 || offset || 0^2
      T+1:   if condition then
                  PC ← PC + target
             else
                  NullifyCurrentInstruction
             endif
64    T–1:   condition ← not COC[1]
      T:     target ← (offset15)^46 || offset || 0^2
      T+1:   if condition then
                  PC ← PC + target
             else
                  NullifyCurrentInstruction
             endif
```

Exceptions:

Coprocessor unusable exception

BC1T — Branch On FPU True (Coprocessor 1) — BC1T

31–26	25–21	20–16	15–0
COP1 0 1 0 0 0 1	BC 0 1 0 0 0	BCT 0 0 0 0 1	offset
6	5	5	16

Format:

BC1T offset

Description:

A branch target address is computed from the sum of the address of the instruction in the delay slot and the 16-bit *offset*, shifted left two bits and sign-extended. If the result of the last floating-point compare is true (one), the program branches to the target address, with a delay of one instruction.

There must be at least one instruction between C.cond.fmt and BC1T.

Operation:

32	T–1:	condition ← COC[1]
	T:	target ← $(\text{offset}_{15})^{14}$ || offset || 0^2

	T+1:	if condition then
		    PC ← PC + target
		endif

64	T–1:	condition ← COC[1]
	T:	target ← $(\text{offset}_{15})^{46}$ || offset || 0^2

	T+1:	if condition then
		    PC ← PC + target
		endif

Exceptions:

Coprocessor unusable exception

BC1TL — Branch On FPU True Likely (Coprocessor 1) — BC1TL

31–26	25–21	20–16	15–0
COP1 0 1 0 0 0 1	BC 0 1 0 0 0	BCTL 0 0 0 1 1	offset
6	5	5	16

Format:

BC1TL offset

Description:

A branch target address is computed from the sum of the address of the instruction in the delay slot and the 16-bit *offset*, shifted left two bits and sign-extended. If the result of the last floating-point compare is true (one), the program branches to the target address, with a delay of one instruction. If the conditional branch is not taken, the instruction in the branch delay slot is nullified.

There must be at least one instruction between C.cond.fmt and BC1TL.

Operation:

```
32   T–1:  condition ← COC[1]
     T:    target ← (offset15)^14 || offset || 0^2
     T+1:  if condition then
               PC ← PC + target
           else
               NullifyCurrentInstruction
           endif
64   T–1:  condition ← COC[1]
     T:    target ← (offset15)^46 || offset || 0^2
     T+1:  if condition then
               PC ← PC + target
           else
               NullifyCurrentInstruction
           endif
```

Exceptions:

Coprocessor unusable exception

C.cond.fmt — Floating-Point Compare — C.cond.fmt

31–26	25–21	20–16	15–11	10–6	5–4	3–0
COP1 0 1 0 0 0 1	fmt	ft	fs	0 0 0 0 0 0	FC*	cond*
6	5	5	5	5	2	4

Format:

C.cond.fmt fs, ft

Description:

The contents of the floating-point registers specified by *fs* and *ft* are interpreted in the specified format, *fmt*, and arithmetically compared.

A result is determined based on the comparison and the conditions specified in the *cond* field. If one of the values is a Not a Number (NaN), and the high-order bit of the *cond* field is set, an invalid operation exception is taken. After a one-instruction delay, the condition is available for testing with branch on floating-point coprocessor condition instructions. There must be at least one instruction between the compare and the branch.

Comparisons are exact and can neither overflow nor underflow. Four mutually-exclusive relations are possible results: less than, equal, greater than, and unordered. The last case arises when one or both of the operands are NaN; every NaN compares unordered with everything, including itself.

Comparisons ignore the sign of zero, so +0 = –0.

This instruction is valid only for single- and double-precision floating-point formats. The operation is not defined if bit 0 of any register specification is set and the *FR* bit in the *Status* register equals zero, since the register numbers specify an even-odd pair of adjacent coprocessor general registers. When the *FR* bit in the *Status* register equals one, both even and odd register numbers are valid.

*See “FPU Instruction Opcode Bit Encoding” at the end of Appendix B.

C.cond.fmt — Floating-Point Compare (continued) — C.cond.fmt

Operation:

```
T:     if NaN(ValueFPR(fs, fmt)) or NaN(ValueFPR(ft, fmt)) then
            less ← false
            equal ← false
            unordered ← true
            if cond3 then
                 signal InvalidOperationException
            endif
       else
            less ← ValueFPR(fs, fmt) < ValueFPR(ft, fmt)
            equal ← ValueFPR(fs, fmt) = ValueFPR(ft, fmt)
            unordered ← false
       endif
       condition ← (cond2 and less) or (cond1 and equal) or
                   (cond0 and unordered)
       FCR[31]23 ← condition
       COC[1] ← condition
```

Exceptions:

Coprocessor unusable
Floating-Point exception

Coprocessor Exceptions:

Unimplemented operation exception
Invalid operation exception

CEIL.L.fmt — Floating-Point Ceiling to Long Fixed-Point Format — CEIL.L.fmt

31 26	25 21	20 16	15 11	10 6	5 0
COP1 0 1 0 0 0 1	fmt	0 0 0 0 0 0	fs	fd	CEIL.L 0 0 1 0 1 0
6	5	5	5	5	6

Format:

CEIL.L.fmt fd, fs

Description:

The contents of the floating-point register specified by *fs* are interpreted in the specified source format, *fmt*, and arithmetically converted to the long fixed-point format. The result is placed in the floating-point register specified by *fd*.

Regardless of the setting of the current rounding mode, the conversion is rounded as if the current rounding mode is round to +∞ (2).

This instruction is valid only for conversion from single- or double-precision floating-point formats. When the *FR* bit in the *Status* register equals one, both even and odd register numbers are valid.

When the source operand is an Infinity, NaN, or the correctly rounded integer result is outside of -2^{63} to $2^{63}-1$, the Invalid operation exception is raised. If the Invalid operation is not enabled then no exception is taken and $2^{63}-1$ is returned.

CEIL.L.fmt — Floating-Point Ceiling to Long Fixed-Point Format (continued) — CEIL.L.fmt

Operation:

```
T:      StoreFPR(fd, L, ConvertFmt(ValueFPR(fs, fmt), fmt, L))
```

Exceptions:

Coprocessor unusable exception
Floating-Point exception

Coprocessor Exceptions:

Invalid operation exception
Unimplemented operation exception
Inexact exception
Overflow exception

CEIL.W.fmt — Floating-Point Ceiling to Single Fixed-Point Format — CEIL.W.fmt

31–26	25–21	20–16	15–11	10–6	5–0
COP1 0 1 0 0 0 1	fmt	0 0 0 0 0 0	fs	fd	CEIL.W 0 0 1 1 1 0
6	5	5	5	5	6

Format:

CEIL.W.fmt fd, fs

Description:

The contents of the floating-point register specified by *fs* are interpreted in the specified source format, *fmt*, and arithmetically converted to the single fixed-point format. The result is placed in the floating-point register specified by *fd*.

Regardless of the setting of the current rounding mode, the conversion is rounded as if the current rounding mode is round to +∞ (2).

This instruction is valid only for conversion from a single- or double-precision floating-point formats. The operation is not defined if bit 0 of any register specification is set and the *FR* bit in the *Status* register equals zero, since the register numbers specify an even-odd pair of adjacent coprocessor general registers. When the *FR* bit in the *Status* register equals one, both even and odd register numbers are valid.

When the source operand is an Infinity or NaN, or the correctly rounded integer result is outside of -2^{31} to $2^{31}-1$, the Invalid operation exception is raised. If the Invalid operation is not enabled then no exception is taken and $2^{31}-1$ is returned.

CEIL.W.fmt Floating-Point Ceiling to Single Fixed-Point Format (continued) CEIL.W.fmt

Operation:

```
T:      StoreFPR(fd, W, ConvertFmt(ValueFPR(fs, fmt), fmt, W))
```

Exceptions:

Coprocessor unusable exception
Floating-Point exception

Coprocessor Exceptions:

Invalid operation exception
Unimplemented operation exception
Inexact exception
Overflow exception

CFC1 — Move Control Word From FPU (Coprocessor 1) — CFC1

31–26	25–21	20–16	15–11	10–0
COP1 0 1 0 0 0 1	CF 0 0 0 1 0	rt	fs	0 0 0 0 0 0 0 0 0 0 0 0
6	5	5	5	11

Format:

CFC1 rt, fs

Description:

The contents of the FPU control register *fs* are loaded into general register *rt*.

This operation is only defined when *fs* equals 0 or 31.

The contents of general register *rt* are undefined for the instruction immediately following CFC1.

Operation:

32	T:	temp ← FCR[fs]
	T+1:	GPR[rt] ← temp
64	T:	temp ← FCR[fs]
	T+1:	GPR[rt] ← $(temp_{31})^{32}$ || temp

Exceptions:

Coprocessor unusable exception

CTC1 Move Control Word To FPU (Coprocessor 1) CTC1

31–26	25–21	20–16	15–11	10–0
COP1 0 1 0 0 0 1	CT 0 0 1 1 0	rt	fs	0 0 0 0 0 0 0 0 0 0 0 0
6	5	5	5	11

Format:

CTC1 rt, fs

Description:

The contents of general register *rt* are loaded into FPU control register *fs*. This operation is only defined when *fs* equals 0 or 31.

Writing to *Control Register 31*, the floating-point *Control/Status* register, causes an interrupt or exception if any cause bit and its corresponding enable bit are both set. The register will be written before the exception occurs. The contents of floating-point control register *fs* are undefined for the instruction immediately following CTC1.

Operation:

32	T:	temp $\leftarrow$ GPR[rt]
	T+1:	FCR[fs] $\leftarrow$ temp
		COC[1] $\leftarrow$ FCR[31]_{23}
64	T:	temp $\leftarrow$ $\text{GPR[rt]}_{31...0}$
	T+1:	FCR[fs] $\leftarrow$ temp
		COC[1] $\leftarrow$ FCR[31]_{23}

Exceptions:

Coprocessor unusable exception
Floating-Point exception

Coprocessor Exceptions:

Unimplemented operation exception
Invalid operation exception
Division by zero exception
Inexact exception
Overflow exception
Underflow exception

CVT.D.fmt — Floating-Point Convert to Double Floating-Point Format — CVT.D.fmt

31–26	25–21	20–16	15–11	10–6	5–0
COP1 0 1 0 0 0 1	fmt	0 0 0 0 0 0	fs	fd	CVT.D 1 0 0 0 0 1
6	5	5	5	5	6

Format:

CVT.D.fmt fd, fs

Description:

The contents of the floating-point register specified by *fs* is interpreted in the specified source format, *fmt*, and arithmetically converted to the double binary floating-point format. The result is placed in the floating-point register specified by *fd*.

This instruction is valid only for conversions from single floating-point format, 32-bit or 64-bit fixed-point format.

If the single floating-point or single fixed-point format is specified, the operation is exact. The operation is not defined if bit 0 of any register specification is set and the *FR* bit in the *Status* register equals zero, since the register numbers specify an even-odd pair of adjacent coprocessor general registers. When the *FR* bit in the *Status* register equals one, both even and odd register numbers are valid.

Operation:

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

Exceptions:

Coprocessor unusable exception
Floating-Point exception

Coprocessor Exceptions:

Invalid operation exception
Unimplemented operation exception
Inexact exception
Overflow exception
Underflow exception

CVT.L.fmt — Floating-Point Convert to Long Fixed-Point Format — CVT.L.fmt

31–26	25–21	20–16	15–11	10–6	5–0
COP1 0 1 0 0 0 1	fmt	0 0 0 0 0 0	fs	fd	CVT.L 1 0 0 1 0 1
6	5	5	5	5	6

Format:

CVT.L.fmt fd, fs

Description:

The contents of the floating-point register specified by *fs* are interpreted in the specified source format, *fmt*, and arithmetically converted to the long fixed-point format. The result is placed in the floating-point register specified by *fd*. This instruction is valid only for conversions from single- or double-precision floating-point formats. The operation is not defined if bit 0 of any register specification is set and the *FR* bit in the *Status* register equals zero.

When the source operand is an Infinity, NaN, or the correctly rounded integer result is outside of -2^{63} to $2^{63}-1$, the Invalid operation exception is raised. If the Invalid operation is not enabled then no exception is taken and $2^{63}-1$ is returned.

Operation:

```
T:     StoreFPR (fd, L, ConvertFmt(ValueFPR(fs, fmt), fmt, L))
```

Exceptions:

Coprocessor unusable exception
Floating-Point exception

Coprocessor Exceptions:

Invalid operation exception
Unimplemented operation exception
Inexact exception
Overflow exception

CVT.S.fmt — Floating-Point Convert to Single Floating-Point Format — CVT.S.fmt

31–26	25–21	20–16	15–11	10–6	5–0
COP1 0 1 0 0 0 1	fmt	0 0 0 0 0 0	fs	fd	CVT.S 1 0 0 0 0 0
6	5	5	5	5	6

Format:

CVT.S.fmt fd, fs

Description:

The contents of the floating-point register specified by *fs* are interpreted in the specified source format, *fmt*, and arithmetically converted to the single binary floating-point format. The result is placed in the floating-point register specified by *fd*. Rounding occurs according to the currently specified rounding mode.

This instruction is valid only for conversions from double floating-point format, or from 32-bit or 64-bit fixed-point format. The operation is not defined if bit 0 of any register specification is set and the *FR* bit in the *Status* register equals zero, since the register numbers specify an even-odd pair of adjacent coprocessor general registers. When the *FR* bit in the *Status* register equals one, both even and odd register numbers are valid.

Operation:

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

Exceptions:

Coprocessor unusable exception
Floating-Point exception

Coprocessor Exceptions:

Invalid operation exception
Unimplemented operation exception
Inexact exception
Overflow exception
Underflow exception

CVT.W.fmt — Floating-Point Convert to Fixed-Point Format — CVT.W.fmt

31–26	25–21	20–16	15–11	10–6	5–0
COP1 0 1 0 0 0 1	fmt	0 0 0 0 0 0	fs	fd	CVT.W 1 0 0 1 0 0
6	5	5	5	5	6

Format:

CVT.W.fmt fd, fs

Description:

The contents of the floating-point register specified by *fs* are interpreted in the specified source format, *fmt*, and arithmetically converted to the single fixed-point format. The result is placed in the floating-point register specified by *fd*. This instruction is valid only for conversion from a single- or double-precision floating-point formats. The operation is not defined if bit 0 of any register specification is set and the *FR* bit in the *Status* register equals zero, since the register numbers specify an even-odd pair of adjacent coprocessor general registers. When the *FR* bit in the *Status* register equals one, both even and odd register numbers are valid.

When the source operand is an Infinity or NaN, or the correctly rounded integer result is outside of -2^{31} to $2^{31}-1$, an Invalid operation exception is raised. If Invalid operation is not enabled, then no exception is taken and $2^{31}-1$ is returned.

Operation:

```
T:      StoreFPR(fd, W, ConvertFmt(ValueFPR(fs, fmt), fmt, W))
```

Exceptions:

Coprocessor unusable exception
Floating-Point exception

Coprocessor Exceptions:

Invalid operation exception
Unimplemented operation exception
Inexact exception
Overflow exception

DIV.fmt — Floating-Point Divide — DIV.fmt

31–26	25–21	20–16	15–11	10–6	5–0
COP1 0 1 0 0 0 1	fmt	ft	fs	fd	DIV 0 0 0 0 1 1
6	5	5	5	5	6

Format:

DIV.fmt fd, fs, ft

Description:

The contents of the floating-point registers specified by *fs* and *ft* are interpreted in the specified *format* and the value in the *fs* field is divided by the value in the *ft* field. The result is rounded as if calculated to infinite precision and then rounded to the specified format, according to the current rounding mode. The result is placed in the floating-point register specified by *fd*.

This instruction is valid for only single or double precision floating-point formats.

The operation is not defined if bit 0 of any register specification is set and the *FR* bit in the *Status* register equals zero, since the register numbers specify an even-odd pair of adjacent coprocessor general registers. When the *FR* bit in the *Status* register equals one, both even and odd register numbers are valid.

Operation:

```
T:      StoreFPR (fd, fmt, ValueFPR(fs, fmt)/ValueFPR(ft, fmt))
```

Exceptions:

Coprocessor unusable exception
Floating-Point exception

Coprocessor Exceptions:

Unimplemented operation exception
Division-by-zero exception
Overflow exception
Invalid operation exception
Inexact exception
Underflow exception

DMFC1 — Doubleword Move From Floating-Point Coprocessor — DMFC1

31–26	25–21	20–16	15–11	10–0
COP1 0 1 0 0 0 1	DMF 0 0 0 0 1	rt	fs	0 0 0 0 0 0 0 0 0 0 00
6	5	5	5	11

Format:

DMFC1 rt, fs

Description:

The contents of register *fs* from the floating-point coprocessor is stored into processor register *rt*.

The contents of general register *rt* are undefined for the instruction immediately following DMFC1.

The *FR* bit in the *Status* register specifies whether all 32 registers of the R4000 are addressable. When *FR* equals zero, this instruction is not defined when the least significant bit of *fs* is non-zero. When *FR* is set, *fs* may specify either odd or even registers.

Operation:

```
64    T:      if SR26 = 1 then   /* 64-bit wide FGRs */
                  data ← FGR[fs]
              elseif  fs0 = 0 then   /* valid specifier, 32-bit wide FGRs */
                  data ← FGR[fs+1] || FGR[fs]
              else   /* undefined for odd 32-bit reg  #s */
                  data ⟵ undefined^64
              endif

      T+1:    GPR[rt] ← data
```

Exceptions:

Coprocessor unusable exception

Coprocessor Exceptions:

Unimplemented operation exception

DMTC1 — Doubleword Move To Floating-Point Coprocessor — DMTC1

31–26	25–21	20–16	15–11	10–0
COP1 0 1 0 0 0 1	DMT 0 0 1 0 1	rt	fs	0 0 0 0 0 0 0 0 0 0 00
6	5	5	5	11

Format:

DMTC1 rt, fs

Description:

The contents of general register *rt* are loaded into coprocessor register *fs* of the CP1.

The contents of floating-point register *fs* are undefined for the instruction immediately following DMTC1.

The *FR* bit in the *Status* register specifies whether all 32 registers of the R4000 are addressable. When *FR* equals zero, this instruction is not defined when the least significant bit of *fs* is non-zero. When *FR* equals one, *fs* may specify either odd or even registers.

Operation:

```
64    T:      data ← GPR[rt]
      T+1:    if SR26 = 1 then   /* 64-bit wide FGRs */
                   FGR[fs] ← data
              elseif  fs0 = 0 then   /*valid specifier, 32-bit wide valid FGRs */
                   FGR[fs+1] ← data63...32
                   FGR[fs] ← data31...0
              else   /* undefined result for odd 32-bit reg #s */
                   undefined_result
              endif
```

Exceptions:

Coprocessor unusable exception

Coprocessor Exceptions:

Unimplemented operation exception

FLOOR.L.fmt — Floating-Point Floor to Long Fixed-Point Format — FLOOR.L.fmt

31 … 26	25 … 21	20 … 16	15 … 11	10 … 6	5 … 0
COP1 0 1 0 0 0 1	fmt	0 0 0 0 0 0	fs	fd	FLOOR.L 0 0 1 0 1 1
6	5	5	5	5	6

Format:

FLOOR.L.fmt fd, fs

Description:

The contents of the floating-point register specified by *fs* are interpreted in the specified source format, *fmt*, and arithmetically converted to the long fixed-point format. The result is placed in the floating-point register specified by *fd*.

Regardless of the setting of the current rounding mode, the conversion is rounded as if the current rounding mode is round to -∞ (3).

This instruction is valid only for conversion from single- or double-precision floating-point formats.

When the source operand is an Infinity, NaN, or the correctly rounded integer result is outside of -2^{63} to $2^{63}-1$, the Invalid operation exception is raised. If the Invalid operation is not enabled then no exception is taken and $2^{63}-1$ is returned.

FLOOR.L.fmt — Floating-Point Floor to Long Fixed-Point Format (continued) — FLOOR.L.fmt

Operation:

```
T:      StoreFPR(fd, L, ConvertFmt(ValueFPR(fs, fmt), fmt, L))
```

Exceptions:

Coprocessor unusable exception
Floating-Point exception

Coprocessor Exceptions:

Invalid operation exception
Unimplemented operation exception
Inexact exception
Overflow exception

FLOOR.W.fmt Floating-Point Floor to Single Fixed-Point Format FLOOR.W.fmt

31 26	25 21	20 16	15 11	10 6	5 0
COP1 0 1 0 0 0 1	fmt	0 0 0 0 0 0	fs	fd	FLOOR.W 0 0 1 1 1 1
6	5	5	5	5	6

Format:

FLOOR.W.fmt fd, fs

Description:

The contents of the floating-point register specified by *fs* are interpreted in the specified source format, *fmt*, and arithmetically converted to the single fixed-point format. The result is placed in the floating-point register specified by *fd*.

Regardless of the setting of the current rounding mode, the conversion is rounded as if the current rounding mode is round to $-\infty$ (RM = 3).

This instruction is valid only for conversion from a single- or double-precision floating-point formats. The operation is not defined if bit 0 of any register specification is set and the *FR* bit in the *Status* register equals zero, since the register numbers specify an even-odd pair of adjacent coprocessor general registers. When the *FR* bit in the *Status* register equals one, both even and odd register numbers are valid.

When the source operand is an Infinity or NaN, or the correctly rounded integer result is outside of -2^{31} to $2^{31}-1$, an Invalid operation exception is raised. If Invalid operation is not enabled, then no exception is taken and $2^{31}-1$ is returned.

FLOOR.W.fmt Floating-Point Floor to Single Fixed-Point Format (continued) FLOOR.W.fmt

Operation:

```
T:      StoreFPR(fd, W, ConvertFmt(ValueFPR(fs, fmt), fmt, W))
```

Exceptions:

Coprocessor unusable exception
Floating-Point exception

Coprocessor Exceptions:

Invalid operation exception
Unimplemented operation exception
Inexact exception
Overflow exception

LDC1 — Load Doubleword to FPU (Coprocessor 1) — LDC1

31 … 26	25 … 21	20 … 16	15 … 0
LDC1 1 1 0 1 0 1	base	ft	offset
6	5	5	16

Format:

LDC1 ft, offset(base)

Description:

The 16-bit *offset* is sign-extended and added to the contents of general register *base* to form an unsigned effective address.

In 32-bit mode, the contents of the doubleword at the memory location specified by the effective address is loaded into registers *ft* and *ft+1* of the floating-point coprocessor. This instruction is not valid, and is undefined, when the least significant bit of *ft* is non-zero.

In 64-bit mode, the contents of the doubleword at the memory location specified by the effective address are loaded into the 64-bit register *ft* of the floating point coprocessor.

The *FR* bit of the *Status* register (SR_{26}) specifies whether all 32 registers of the R4000 are addressable. If *FR* equals zero, this instruction is not defined when the least significant bit of *ft* is non-zero. If *FR* equals one, *ft* may specify either odd or even registers.

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

LDC1 Load Doubleword to FPU (Coprocessor 1) (continued) LDC1

Operation:

32	T:	vAddr ← $((\text{offset}_{15})^{16} \parallel \text{offset}_{15...0}) + \text{GPR[base]}$
64	T:	vAddr ← $((\text{offset}_{15})^{48} \parallel \text{offset}_{15...0}) + \text{GPR[base]}$
32, 64		(pAddr, uncached) ← AddressTranslation (vAddr, DATA)
		data ← LoadMemory(uncached, DOUBLEWORD, pAddr, vAddr, DATA)
		if SR_{26} = 1 then /* 64-bit wide FGRs */
		FGR[ft] ← data
		elseif ft_0 = 0 then /* valid specifier, 32-bit wide FGRs */
		FGR[ft+1] ← $\text{data}_{63...32}$
		FGR[ft] ← $\text{data}_{31...0}$
		else /* undefined result if odd */
		undefined_result
		endif

Exceptions:

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

LWC1 — Load Word to FPU (Coprocessor 1) — LWC1

31 … 26	25 … 21	20 … 16	15 … 0
LWC1 1 1 0 0 0 1	base	ft	offset
6	5	5	16

Format:

LWC1 ft, offset(base)

Description:

The 16-bit *offset* is sign-extended and added to the contents of general register *base* to form an unsigned effective address. The contents of the word at the memory location specified by the effective address is loaded into register *ft* of the floating-point coprocessor.

The *FR* bit of the *Status* register specifies whether all 64-bit *Floating-Point* registers are addressable. If *FR* equals zero, LWC1 loads either the high or low half of the 16 even *Floating-Point* registers. If *FR* equals one, LWC1 loads the low 32-bits of both even and odd *Floating-Point* registers.

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

LWC1 — Load Word to FPU (Coprocessor 1) (continued) — LWC1

Operation:

32	T:	$vAddr \leftarrow ((offset_{15})^{16} \mid\mid offset_{15...0}) + GPR[base]$
64	T:	$vAddr \leftarrow ((offset_{15})^{48} \mid\mid offset_{15...0}) + GPR[base]$

32, 64		(pAddr, uncached) ← AddressTranslation (vAddr, DATA)
		$pAddr \leftarrow pAddr_{PSIZE-1...3} \mid\mid (pAddr_{2...0}$ xor (ReverseEndian $\mid\mid 0^2$))
		mem ← LoadMemory(uncached, WORD, pAddr, vAddr, DATA)
		$byte \leftarrow vAddr_{2...0}$ xor (BigEndianCPU $\mid\mid 0^2$)
		/* “mem” is aligned 64-bits from memory. Pick out correct bytes. */
		if SR_{26} = 1 then /* 64-bit wide FGRs */
		    $FGR[ft] \leftarrow undefined^{32} \mid\mid mem_{31+8*byte...8*byte}$
		else /* 32-bit wide FGRs */
		    $FGR[ft] \leftarrow mem_{31+8*byte...8*byte}$
		endif

Exceptions:

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

MFC1 — Move From FPU (Coprocessor 1) — MFC1

31 — 26	25 — 21	20 — 16	15 — 11	10 — 0
COP1 0 1 0 0 0 1	MF 0 0 0 0 0	rt	fs	0 000 0000 0000
6	5	5	5	11

Format:

MFC1 rt, fs

Description:

The contents of register *fs* from the floating-point coprocessor are stored into processor register *rt*.

The contents of register *rt* are undefined for the instruction immediately following MFC1.

The *FR* bit of the *Status* register specifies whether all 32 registers of the R4000 are addressable. If *FR* equals zero, MFC1 stores either the high or low half of the 16 even *Floating-Point* registers. If *FR* equals one, MFC1 stores the low 32-bits of both even and odd *Floating-Point* registers.

Operation:

32 T:		data $\leftarrow$ FGR[fs]$_{31...0}$
	T+1:	GPR[rt] $\leftarrow$ data
64 T:		data $\leftarrow$ FGR[fs]$_{31...0}$
	T+1:	GPR[rt] $\leftarrow$ (data$_{31}$)^{32} || data

Exceptions:

Coprocessor unusable exception

MOV.fmt Floating-Point Move MOV.fmt

31–26	25–21	20–16	15–11	10–6	5–0
COP1 0 1 0 0 0 1	fmt	0 0 0 0 0 0	fs	fd	MOV 0 0 0 1 1 0
6	5	5	5	5	6

Format:

MOV.fmt fd, fs

Description:

The contents of the FPU register specified by *fs* are interpreted in the specified *format* and are copied into the FPU register specified by *fd*.

The move operation is non-arithmetic; no IEEE 754 exceptions occur as a result of the instruction.

This instruction is valid only for single- or double-precision floating-point formats.

The operation is not defined if bit 0 of any register specification is set and the *FR* bit in the *Status* register equals zero, since the register numbers specify an even-odd pair of adjacent coprocessor general registers. When the *FR* bit in the *Status* register equals one, both even and odd register numbers are valid.

Operation:

```
T:    StoreFPR(fd, fmt, ValueFPR(fs, fmt))
```

Exceptions:

Coprocessor unusable exception
Floating-Point exception

Coprocessor Exceptions:

Unimplemented operation exception

MTC1 — Move To FPU (Coprocessor 1) — MTC1

31 26	25 21	20 16	15 11	10 0
COP1 0 1 0 0 0 1	MT 0 0 1 0 0	rt	fs	0 000 0000 0000
6	5	5	5	11

Format:

MTC1 rt, fs

Description:

The contents of register *rt* are loaded into the FPU general register at location *fs*.

The contents of floating-point register *fs* is undefined for the instruction immediately following MTC1.

The *FR* bit of the *Status* register specifies whether all 32 registers of the R4000 are addressable. If *FR* equals zero, MTC1 loads either the high or low half of the 16 even *Floating-Point* registers. If *FR* equals one, MTC1 loads the low 32-bits of both even and odd *Floating-Point* registers.

Operation:

```
32,64   T:      data ← GPR[rt]31...0
        T+1:    if SR26 = 1 then   /* 64-bit wide FGRs */
                     FGR[fs] ← undefined^32 || data
                else   /* 32-bit wide FGRs */
                     FGR[fs] ← data
                endif
```

Exceptions:

Coprocessor unusable exception

MUL.fmt **Floating-Point Multiply** MUL.fmt

31–26	25–21	20–16	15–11	10–6	5–0
COP1 0 1 0 0 0 1	fmt	ft	fs	fd	MUL 0 0 0 0 1 0
6	5	5	5	5	6

Format:

MUL.fmt fd, fs, ft

Description:

The contents of the floating-point registers specified by *fs* and *ft* are interpreted in the specified *format* and arithmetically multiplied. The result is rounded as if calculated to infinite precision and then rounded to the specified *format*, according to the current rounding mode. The result is placed in the floating-point register specified by *fd*.

This instruction is valid only for single- or double-precision floating-point formats.

The operation is not defined if bit 0 of any register specification is set and the *FR* bit in the *Status* register equals zero, since the register numbers specify an even-odd pair of adjacent coprocessor general registers. When the *FR* bit in the *Status* register equals one, both even and odd register numbers are valid.

Operation:

```
T:      StoreFPR (fd, fmt, ValueFPR(fs, fmt) * ValueFPR(ft, fmt))
```

Exceptions:

Coprocessor unusable exception
Floating-Point exception

Coprocessor Exceptions:

Unimplemented operation exception
Invalid operation exception
Inexact exception
Overflow exception
Underflow exception

NEG.fmt Floating-Point Negate NEG.fmt

31–26	25–21	20–16	15–11	10–6	5–0
COP1 0 1 0 0 0 1	fmt	0 0 0 0 0 0	fs	fd	NEG 0 0 0 1 1 1
6	5	5	5	5	6

Format:

NEG.fmt fd, fs

Description:

The contents of the FPU register specified by *fs* are interpreted in the specified *format* and the arithmetic negation is taken (polarity of the sign-bit is changed). The result is placed in the FPU register specified by *fd*.

The negate operation is arithmetic; an NaN operand signals invalid operation.

This instruction is valid only for single- or double-precision floating-point formats. The operation is not defined if bit 0 of any register specification is set and the *FR* bit in the *Status* register equals zero, since the register numbers specify an even-odd pair of adjacent coprocessor general registers. When the *FR* bit in the *Status* register equals one, both even and odd register numbers are valid.

Operation:

```
T:    StoreFPR(fd, fmt, Negate(ValueFPR(fs, fmt)))
```

Exceptions:

Coprocessor unusable exception
Floating-Point exception

Coprocessor Exceptions:

Unimplemented operation exception
Invalid operation exception

ROUND.L.fmt Floating-Point Round to Long Fixed-Point Format ROUND.L.fmt

31 26	25 21	20 16	15 11	10 6	5 0
COP1 0 1 0 0 0 1	fmt	0 0 0 0 0 0	fs	fd	ROUND.L 0 0 1 0 0 0
6	5	5	5	5	6

Format:

ROUND.L.fmt fd, fs

Description:

The contents of the floating-point register specified by *fs* are interpreted in the specified source format, *fmt*, and arithmetically converted to the long fixed-point format. The result is placed in the floating-point register specified by *fd*.

Regardless of the setting of the current rounding mode, the conversion is rounded as if the current rounding mode is round to nearest/even (0).

This instruction is valid only for conversion from single- or double-precision floating-point formats.

When the source operand is an Infinity, NaN, or the correctly rounded integer result is outside of -2^{63} to $2^{63}-1$, the Invalid operation exception is raised. If the Invalid operation is not enabled then no exception is taken and $2^{63}-1$ is returned.

ROUND.L.fmt **Floating-Point Round to Long Fixed-Point Format (continued)** ROUND.L.fmt

Operation:

```
T:    StoreFPR(fd, L, ConvertFmt(ValueFPR(fs, fmt), fmt, L))
```

Exceptions:

Coprocessor unusable exception
Floating-Point exception

Coprocessor Exceptions:

Invalid operation exception
Unimplemented operation exception
Inexact exception
Overflow exception

ROUND.W.fmt Floating-Point Round to Single Fixed-Point Format ROUND.W.fmt

31–26	25–21	20–16	15–11	10–6	5–0
COP1 0 1 0 0 0 1	fmt	0 0 0 0 0 0	fs	fd	ROUND.W 0 0 1 1 0 0
6	5	5	5	5	6

Format:

ROUND.W.fmt fd, fs

Description:

The contents of the floating-point register specified by *fs* are interpreted in the specified source format, *fmt*, and arithmetically converted to the single fixed-point format. The result is placed in the floating-point register specified by *fd*.

Regardless of the setting of the current rounding mode, the conversion is rounded as if the current rounding mode is round to the nearest/even (RM = 0).

This instruction is valid only for conversion from a single- or double-precision floating-point formats. The operation is not defined if bit 0 of any register specification is set and the *FR* bit in the *Status* register equals zero, since the register numbers specify an even-odd pair of adjacent coprocessor general registers. When the *FR* bit in the *Status* register equals one, both even and odd register numbers are valid.

When the source operand is an Infinity or NaN, or the correctly rounded integer result is outside of -2^{31} to $2^{31}-1$, an Invalid operation exception is raised. If Invalid operation is not enabled, then no exception is taken and $2^{31}-1$ is returned.

ROUND.W.fmt Floating-Point Round to Single Fixed-Point Format (continued) ROUND.W.fmt

Operation:

```
T:      StoreFPR(fd, W, ConvertFmt(ValueFPR(fs, fmt), fmt, W))
```

Exceptions:

Coprocessor unusable exception
Floating-Point exception

Coprocessor Exceptions:

Invalid operation exception
Unimplemented operation exception
Inexact exception
Overflow exception

SDC1 — Store Doubleword from FPU (Coprocessor 1) — SDC1

31–26	25–21	20–16	15–0
SDC1 1 1 1 1 0 1	base	ft	offset
6	5	5	16

Format:

SDC1 ft, offset(base)

Description:

The 16-bit *offset* is sign-extended and added to the contents of general register *base* to form an unsigned effective address.

In 32-bit mode, the contents of registers *ft* and *ft+1* from the floating-point coprocessor are stored at the memory location specified by the effective address. This instruction is not valid, and is undefined, when the least significant bit of *ft* is non-zero.

In 64-bit mode, the 64-bit register *ft* is stored to the contents of the doubleword at the memory location specified by the effective address. The *FR* bit of the *Status* register (SR_{26}) specifies whether all 32 registers of the R4000 are addressable. When FR equals zero, this instruction is not defined if the least significant bit of *ft* is non-zero. If FR equals one, *ft* may specify either odd or even registers.

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

SDC1 — Store Doubleword from FPU (Coprocessor 1) (continued) — SDC1

Operation:

32	T:	vAddr ← $(\text{offset}_{15})^{16} \,\|\|\, \text{offset}_{15...0}$) + GPR[base]
64	T:	vAddr ← $(\text{offset}_{15})^{48} \,\|\|\, \text{offset}_{15...0}$) + GPR[base]
32,64		(pAddr, uncached) ← AddressTranslation (vAddr, DATA)
		if SR_{26} = 1 /* 64-bit wide FGRs */
		data ← FGR[ft]
		elseif ft_0 = 0 then /* valid specifier, 32-bit wide FGRs */
		data ← FGR[ft+1] || FGR[ft]
		else /* undefined for odd 32-bit reg #s */
		data ← undefined^{64}
		endif
		StoreMemory(uncached, DOUBLEWORD, data, pAddr, vAddr, DATA)

Exceptions:

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

SQRT.fmt — Floating-Point Square Root — SQRT.fmt

31–26	25–21	20–16	15–11	10–6	5–0
COP1 0 1 0 0 0 1	fmt	0 0 0 0 0 0	fs	fd	SQRT 0 0 0 1 0 0
6	5	5	5	5	6

Format:

SQRT.fmt fd, fs

Description:

The contents of the floating-point register specified by *fs* are interpreted in the specified *format* and the positive arithmetic square root is taken. The result is rounded as if calculated to infinite precision and then rounded to the specified *format*, according to the current rounding mode. If the value of *fs* corresponds to –0, the result will be –0. The result is placed in the floating-point register specified by *fd.*

This instruction is valid only for single- or double-precision floating-point formats.

The operation is not defined if bit 0 of any register specification is set and the *FR* bit in the *Status* register equals zero, since the register numbers specify an even-odd pair of adjacent coprocessor general registers. When the *FR* bit in the *Status* register equals one, both even and odd register numbers are valid.

Operation:

```
T:    StoreFPR(fd, fmt, SquareRoot(ValueFPR(fs, fmt)))
```

Exceptions:

Coprocessor unusable exception
Floating-Point exception

Coprocessor Exceptions:

Unimplemented operation exception
Invalid operation exception
Inexact exception

SUB.fmt Floating-Point Subtract SUB.fmt

31–26	25–21	20–16	15–11	10–6	5–0
COP1 0 1 0 0 0 1	fmt	ft	fs	fd	SUB 0 0 0 0 0 1
6	5	5	5	5	6

Format:

SUB.fmt fd, fs, ft

Description:

The contents of the floating-point registers specified by *fs* and *ft* are interpreted in the specified *format* and the value in the *ft* field is subtracted from the value in the *fs* field. The result is rounded as if calculated to infinite precision and then rounded to the specified *format*, according to the current rounding mode. The result is placed in the floating-point register specified by *fd*. This instruction is valid only for single- or double-precision floating-point formats.

The operation is not defined if bit 0 of any register specification is set and the *FR* bit in the *Status* register equals zero, since the register numbers specify an even-odd pair of adjacent coprocessor general registers. When the *FR* bit in the *Status* register equals one, both even and odd register numbers are valid.

Operation:

```
T:    StoreFPR (fd, fmt, ValueFPR(fs, fmt) – ValueFPR(ft, fmt))
```

Exceptions:

Coprocessor unusable exception
Floating-Point exception

Coprocessor Exceptions:

Unimplemented operation exception
Invalid operation exception
Inexact exception
Overflow exception
Underflow exception

SWC1 — Store Word from FPU (Coprocessor 1) — SWC1

31–26	25–21	20–16	15–0
SWC1 1 1 1 0 0 1	base	ft	offset
6	5	5	16

Format:

SWC1 ft, offset(base)

Description:

The 16-bit *offset* is sign-extended and added to the contents of general register *base* to form an unsigned effective address. The contents of register *ft* from the floating-point coprocessor are stored at the memory location specified by the effective address.

The *FR* bit of the *Status* register specifies whether all 64-bit floating-point registers are addressable.

If *FR* equals zero, SWC1 stores either the high or low half of the 16 even floating-point registers.

If *FR* equals one, SWC1 stores the low 32-bits of both even and odd floating-point registers.

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

SWC1 — Store Word from FPU (Coprocessor 1) (continued) — SWC1

Operation:

32	T:	$vAddr \leftarrow ((offset_{15})^{16} \,\|\|\, offset_{15...0}) + GPR[base]$
64	T:	$vAddr \leftarrow ((offset_{15})^{48} \,\|\|\, offset_{15...0}) + GPR[base]$

32, 64		$(pAddr, uncached) \leftarrow AddressTranslation\ (vAddr, DATA)$
		$pAddr \leftarrow pAddr_{PSIZE-1...3} \,\|\|\, (pAddr_{2...0}\ xor\ (ReverseEndian \,\|\|\, 0^2))$
		$byte \leftarrow vAddr_{2...0}\ xor\ (BigEndianCPU \,\|\|\, 0^2)$
		/* the bytes of the word are put in the correct byte lanes in
		 * “data” for a 64-bit path to memory */
		if SR_{26} = 1 then /* 64-bit wide FGRs */
		    $data \leftarrow FGR[ft]_{63-8*byte...0} \,\|\|\, 0^{8*byte}$
		else /* 32-bit wide FGRs */
		    $data \leftarrow 0^{32-8*byte} \,\|\|\, FGR[ft] \,\|\|\, 0^{8*byte}$
		endif
		StoreMemory (uncached, WORD, data, pAddr, vAddr, DATA)

Exceptions:

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

TRUNC.L.fmt — Floating-Point Truncate to Long Fixed-Point Format — TRUNC.L.fmt

31–26	25–21	20–16	15–11	10–6	5–0
COP1 0 1 0 0 0 1	fmt	0 0 0 0 0 0	fs	fd	TRUNC.L 0 0 1 0 01
6	5	5	5	5	6

Format:

TRUNC.L.fmt fd, fs

Description:

The contents of the floating-point register specified by *fs* are interpreted in the specified source format, *fmt*, and arithmetically converted to the long fixed-point format. The result is placed in the floating-point register specified by *fd*.

Regardless of the setting of the current rounding mode, the conversion is rounded as if the current rounding mode is round toward zero (1).

This instruction is valid only for conversion from single- or double-precision floating-point formats.

When the source operand is an Infinity, NaN, or the correctly rounded integer result is outside of -2^{63} to $2^{63}-1$, the Invalid operation exception is raised. If the Invalid operation is not enabled then no exception is taken and $2^{63}-1$ is returned.

TRUNC.L.fmt Floating-Point Truncate to Long Fixed-Point Format (continued) TRUNC.L.fmt

Operation:

```
T:      StoreFPR(fd, L, ConvertFmt(ValueFPR(fs, fmt), fmt, L))
```

Exceptions:

Coprocessor unusable exception
Floating-Point exception

Coprocessor Exceptions:

Invalid operation exception
Unimplemented operation exception
Inexact exception
Overflow exception

TRUNC.W.fmt Floating-Point Truncate to Single Fixed-Point Format TRUNC.W.fmt

31–26	25–21	20–16	15–11	10–6	5–0
COP1 0 1 0 0 0 1	fmt	0 0 0 0 0 0	fs	fd	TRUNC.W 0 0 1 1 0 1
6	5	5	5	5	6

Format:

TRUNC.W.fmt fd, fs

Description:

The contents of the FPU register specified by *fs* are interpreted in the specified source format *fmt* and arithmetically converted to the single fixed-point format. The result is placed in the FPU register specified by *fd*.

Regardless of the setting of the current rounding mode, the conversion is rounded as if the current rounding mode is round toward zero (RM = 1).

This instruction is valid only for conversion from a single- or double-precision floating-point formats. The operation is not defined if bit 0 of any register specification is set and the *FR* bit in the *Status* register equals zero, since the register numbers specify an even-odd pair of adjacent coprocessor general registers. When the *FR* bit in the *Status* register equals one, both even and odd register numbers are valid.

When the source operand is an Infinity or NaN, or the correctly rounded integer result is outside of -2^{31} to 2^{31-1}, an Invalid operation exception is raised. If Invalid operation is not enabled, then no exception is taken and -2^{31} is returned.

TRUNC.W.fmt Floating-Point Truncate to Single Fixed-Point Format (continued) TRUNC.W.fmt

Operation:

```
T:    StoreFPR(fd, W, ConvertFmt(ValueFPR(fs, fmt), fmt, W))
```

Exceptions:

Coprocessor unusable exception
Floating-Point exception

Coprocessor Exceptions:

Invalid operation exception
Unimplemented operation exception
Inexact exception
Overflow exception

FPU Instruction Opcode Bit Encoding

Opcode

31...29 \ 28...26	0	1	2	3	4	5	6	7
0								
1								
2		COP1						
3								
4								
5								
6		LWC1				LDC1		
7		SWC1				SDC1		

sub

25...24 \ 23...21	0	1	2	3	4	5	6	7
0	MF	DMFη	CF	δ	MT	DMTη	CT	δ
1	BC	δ	δ	δ	δ	δ	δ	δ
2	S	D	δ	δ	W	Lη	δ	δ
3	δ	δ	δ	δ	δ	δ	δ	δ

br

20...19 \ 18...16	0	1	2	3	4	5	6	7
0	BCF	BCT	BCFL	BCTL	γ	γ	γ	γ
1	γ	γ	γ	γ	γ	γ	γ	γ
2	γ	γ	γ	γ	γ	γ	γ	γ
3	γ	γ	γ	γ	γ	γ	γ	γ

Figure B-3 Bit Encoding for FPU Instructions

5...3 \ 2...0	0	1	2	3	4	5	6	7
0	ADD	SUB	MUL	DIV	SQRT	ABS	MOV	NEG
1	ROUND.Lη	TRUNC.Lη	CEIL.Lη	FLOOR.Lη	ROUND.W	TRUNC.W	CEIL.W	FLOOR.W
2	δ	δ	δ	δ	δ	δ	δ	δ
3	δ	δ	δ	δ	δ	δ	δ	δ
4	CVT.S	CVT.D	δ	δ	CVT.W	CVT.Lη	δ	δ
5	δ	δ	δ	δ	δ	δ	δ	δ
6	C.F	C.UN	C.EQ	C.UEQ	C.OLT	C.ULT	C.OLE	C.ULE
7	C.SF	C.NGLE	C.SEQ	C.NGL	C.LT	C.NGE	C.LE	C.NGT

(function)

Figure B-3 (cont.) Bit Encoding for FPU Instructions

Key:

γ Operation codes marked with a gamma cause a reserved instruction exception. They are reserved for future versions of the architecture.

δ Operation codes marked with a delta cause unimplemented operation exceptions in all current implementations and are reserved for future versions of the architecture.

η Operation codes marked with an eta are valid only when MIPS III instructions are enabled. Any attempt to execute these without MIPS III instructions enabled causes an unimplemented operation exception.