

ALLEGREX™ FPU Instruction Manual

© 2006 Sony Computer Entertainment Inc.
All Rights Reserved.
SCE Confidential

Table of Contents

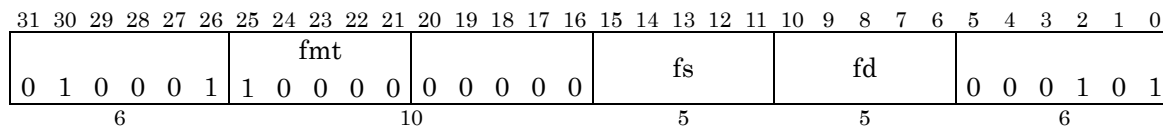
ALLEGREX™ FPU Instructions	4
abs.s	5
add.s	6
bc1f	7
bc1fl	8
bc1t	10
bc1tl	11
c.f.s	13
c.un.s	15
c.eq.s	17
c.ueq.s	19
c.olt.s	21
c.ult.s	23
c.ole.s	25
c.ule.s	27
c.sf.s	29
c.ngle.s	31
c.seq.s	33
c.ngl.s	35
c.lt.s	37
c.nge.s	39
c.le.s	41
c.ngt.s	43
ceil.w.s	45
cfc1	46
ctc1	47
cvt.s.w	49
cvt.w.s	50
div.s	51
floor.w.s	52
lwc1	53
mfc1	54

mov.s	55
mtcl	56
mul.s	57
neg.s	58
round.w.s	59
sqrt.s	60
sub.s	61
swcl	62
trunc.w.s	63
FPU Instruction Set Code Tables	64

ALLEGREX™ FPU Instructions

abs.s

Floating-point Absolute Value



FPU

Syntax:

```
abs.s  fd, fs
```

Description:

The value of the FPU register specified by `fs` is interpreted as being in the format specified by `fmt`, its absolute value is arithmetically calculated, and the result is stored in the FPU register specified by `fd`.

Absolute value is performed as an arithmetic operation. Consequently, an Invalid Operation exception occurs if the operand is not a number (NaN).

This instruction is valid for single-precision floating-point format only.

Operation:

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

Exceptions:

```
Coprocessor Unusable exception
```

```
Floating-Point exception trap
```

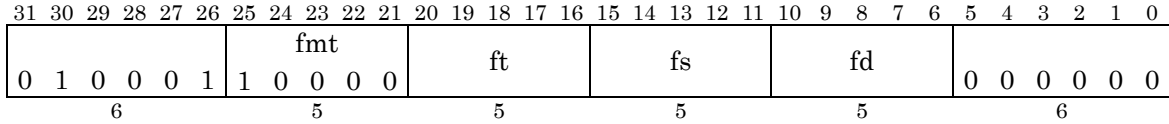
FPU Exceptions:

```
Unimplemented Operation exception(for example, .d)
```

```
Invalid Operation exception
```

add.s

Floating-point Add



FPU

Syntax:

```
add.s  fd, fs, ft
```

Description:

The values of the FPU registers specified by fs and ft are interpreted as being in the format specified by fmt and arithmetically added. The result is calculated to infinite precision and rounded for the specified format according to the current rounding mode. The result is stored in the FPU register specified by fd.

Addition is performed as an arithmetic operation. Consequently, an Invalid Operation exception occurs if the operand is not a number (NaN).

This instruction is valid for single-precision floating-point format only.

Operation:

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

Exceptions:

```
Coprocessor Unusable exception
```

```
Floating-Point exception
```

FPU Exceptions:

```
Unimplemented Operation exception(for example, .d)
```

```
Invalid Operation exception
```

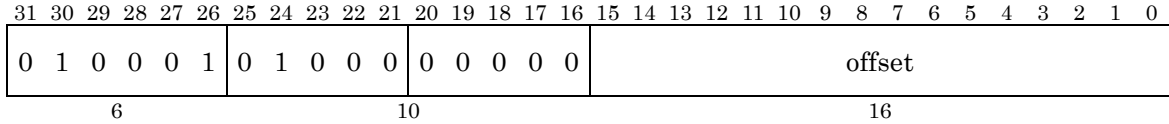
```
Inexact exception
```

```
Overflow exception
```

```
Underflow exception
```

bc1f

Branch on FPU False

MIPS I
FPU**Syntax:**`bc1f offset`**Description:**

When the FPU condition is false, the program branches with a one instruction delay to the branch target address.

The branch target address is the sum of the PC and the 16-bit offset after it has been left-shifted 2 bits and sign-extended to a 32-bit value.

Restrictions:

To change the FPU condition, at least one nop is required between the FPU instruction that changes the condition and the bc1f instruction.

A nop is automatically inserted by the compiler.

The bc1f instruction cannot be placed in the delay slot of a branch/jump instruction.

The ctc1 instruction cannot be placed in the delay slot of the bc1f instruction.

Operation:

```

I-1:
    condition ← notCOC[1]
I:
    target_offset ← sign_extend(offset || 02)
I+1:
    if (condition) then
        PC ← PC +target_offset
    endif

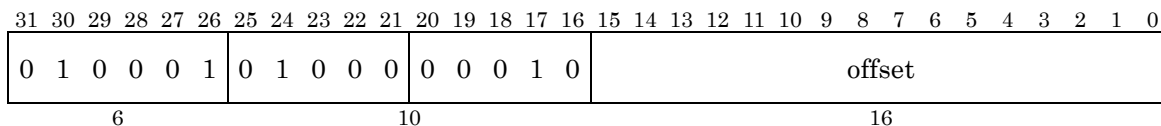
```

Exceptions:

Coprocessor Unusable exception

bc1fl

Branch on FPU False Likely

MIPS II
FPU**Syntax:**`bc1fl offset`**Description:**

When the FPU condition is false, the program branches with a one instruction delay to the branch target address.

The branch target address is the sum of the PC and the 16-bit offset after it has been left-shifted 2 bits and sign-extended to a 32-bit value.

If the branch is not taken, the instruction in the branch delay slot is discarded.

Restrictions:

To change the FPU condition, at least one nop is required between the FPU instruction that changes the condition and the BC1FL instruction.

A nop is automatically inserted by the compiler.

The bc1fl instruction cannot be placed in the delay slot of a branch/jump instruction.

The etc1 instruction cannot be placed in the delay slot of the bc1fl instruction.

Operation:

```

I-1:
    condition ← notCOC[1]
I:
    target_offset ← sign_extend(offset || 02)
I+1:
    if (condition) then
        PC ← PC +target_offset
    else
        NullifyCurrentInstruction
    endif

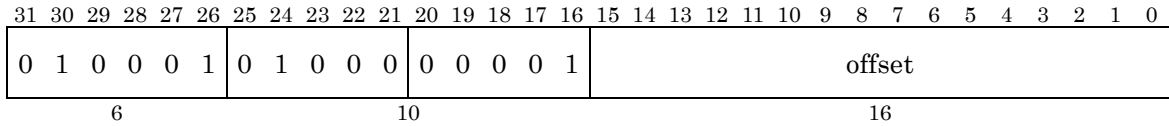
```


Exceptions:

Coprocessor Unusable exception

bc1t

Branch on FPU True

MIPS I
FPU**Syntax:**`bc1t offset`**Description:**

When the FPU condition is true, the program branches with a one instruction delay to the branch target address.

The branch target address is the sum of the PC and the 16-bit offset after it has been left-shifted 2 bits and sign-extended to a 32-bit value.

Restrictions:

To change the FPU condition, at least one nop is required between the FPU instruction that changes the condition and the BC1T instruction.

A nop is automatically inserted by the compiler.

The bc1t instruction cannot be placed in the delay slot of a branch/jump instruction.

The ctc1 instruction cannot be placed in the delay slot of the bc1t instruction.

Operation:

```

I-1:
    condition ← COC[1]
I:
    target_offset ← sign_extend(offset || 02)
I+1:
    if (condition) then
        PC ← PC +target_offset
    endif

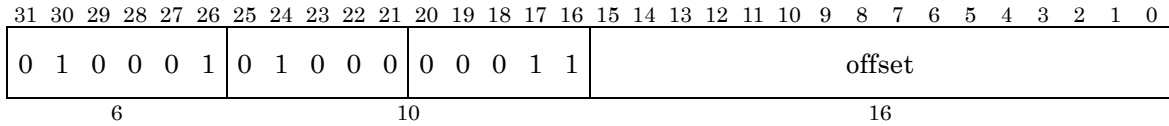
```

Exceptions:

Coprocessor Unusable exception

bc1tl

Branch on FPU True Likely

MIPS II
FPU**Syntax:**`bc1tl offset`**Description:**

When the FPU condition is true, the program branches with a one instruction delay to the branch target address.

The branch target address is the sum of the PC and the 16-bit offset after it has been left-shifted 2 bits and sign-extended to a 32-bit value.

If the branch is not taken, the instruction in the branch delay slot is discarded.

Restrictions:

To change the FPU condition, at least one nop is required between the FPU instruction that changes the condition and the BC1TL instruction.

A nop is automatically inserted by the compiler.

The `bc1tl` instruction cannot be placed in the delay slot of a branch/jump instruction.

The `etc1` instruction cannot be placed in the delay slot of the `bc1tl` instruction.

Operation:

```

I-1:
    condition ← COC[1]
I:
    target_offset ← sign_extend(offset || 02)
I+1:
    if (condition) then
        PC ← PC +target_offset
    else
        NullifyCurrentInstruction
    endif

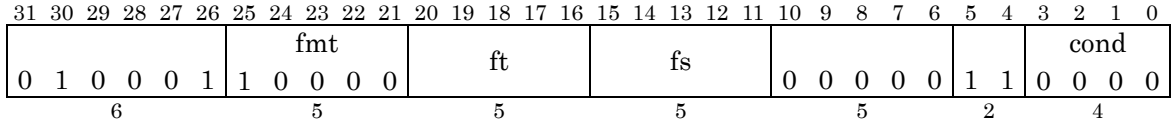
```

Exceptions:

Coprocessor Unusable exception

c.f.s

Floating-point Compare (.f.s)



FPU

Syntax:

```
c.f.s fs, ft
```

Description:

The values of the FPU registers specified by *fs* and *ft* are interpreted as single-precision floating-point numbers and arithmetically compared. The result is obtained by combining this comparison with the condition *cond* specified in the instruction (which is always false). If either operand is not a number (Signaling NaN), an Invalid Operation exception occurs. After a delay of one instruction, a floating-point coprocessor branch instruction test can be performed.

The comparison is exact, and will not generate an overflow or underflow. The result can be one of four mutually exclusive relationships (less than, equal, greater than, or unordered). The last case occurs when one or both of the operands are not a number (NaN). The comparison operation ignores any sign appended to a zero value (in other words, $+0 = -0$). This instruction is valid for single-precision floating-point format only.

Operation:

```
if (NaN (ValueFPR(fs, s)) || NaN (ValueFPR(ft, s))) then
    less ← false
    equal ← false
    unordered ← true
    if (cond3 || SNaN(ValueFPR(fs, s)) || SNaN(ValueFPR(ft, s))) then
        SignalException(InvalidOperation)
    endif
else
    less ← ValueFPR(fs, s) < ValueFPR(ft, s)
    equal ← ValueFPR(fs, s) = ValueFPR(ft, s)
    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 exception

Floating-Point exception

FPU Exceptions:

Unimplemented Operation exception(for example, .d)

Invalid Operation exception

c.un.s

Floating-point Compare (.un.s)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0					
						fmt					ft					fs					0 0 0 0 0		1 1		cond											
0	1	0	0	0	1	1	0	0	0	0																										
6						5					5					5					2		4													

FPU

Syntax:

```
c.un.s fs, ft
```

Description:

The values of the FPU registers specified by *fs* and *ft* are interpreted as single-precision floating-point numbers and arithmetically compared. The result is obtained by combining this comparison with the condition *cond* specified in the instruction (*fs* is not a number or *ft* is not a number). If either operand is not a number (Signaling NaN), an Invalid Operation exception occurs. After a delay of one instruction, a floating-point coprocessor branch instruction test can be performed.

The comparison is exact, and will not generate an overflow or underflow. The result can be one of four mutually exclusive relationships (less than, equal, greater than, or unordered). The last case occurs when one or both of the operands are not a number (NaN). The comparison operation ignores any sign appended to a zero value (in other words, $+0 = -0$). This instruction is valid for single-precision floating-point format only.

Operation:

```
if (NaN (ValueFPR(fs, s)) || NaN (ValueFPR(ft, s))) then
    less ← false
    equal ← false
    unordered ← true
    if (cond3 || SNaN(ValueFPR(fs, s)) || SNaN(ValueFPR(ft, s))) then
        SignalException(InvalidOperation)
    endif
else
    less ← ValueFPR(fs, s) < ValueFPR(ft, s)
    equal ← ValueFPR(fs, s) = ValueFPR(ft, s)
    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 exception

Floating-Point exception

FPU Exceptions:

Unimplemented Operation exception(for example, .d)

Invalid Operation exception

c.eq.s

Floating-point Compare (.eq.s)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0			
						fmt					ft					fs					0 0 0 0 0		1 1		cond									
0	1	0	0	0	1	1	0	0	0	0																								
6						5					5					5					2		4											

FPU

Syntax:

```
c.eq.s fs, ft
```

Description:

The values of the FPU registers specified by *fs* and *ft* are interpreted as single-precision floating-point numbers and arithmetically compared. The result is obtained by combining this comparison with the condition *cond* specified in the instruction (*fs* is equal to *ft*). If either operand is not a number (Signaling NaN), an Invalid Operation exception occurs. After a delay of one instruction, a floating-point coprocessor branch instruction test can be performed.

The comparison is exact, and will not generate an overflow or underflow. The result can be one of four mutually exclusive relationships (less than, equal, greater than, or unordered). The last case occurs when one or both of the operands are not a number (NaN). The comparison operation ignores any sign appended to a zero value (in other words, $+0 = -0$). This instruction is valid for single-precision floating-point format only.

Operation:

```
if (NaN (ValueFPR(fs, s)) || NaN (ValueFPR(ft, s))) then
    less ← false
    equal ← false
    unordered ← true
    if (cond3 || SNaN(ValueFPR(fs, s)) || SNaN(ValueFPR(ft, s))) then
        SignalException(InvalidOperation)
    endif
else
    less ← ValueFPR(fs, s) < ValueFPR(ft, s)
    equal ← ValueFPR(fs, s) = ValueFPR(ft, s)
    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 exception

Floating-Point exception

FPU Exceptions:

Unimplemented Operation exception(for example, .d)

Invalid Operation exception

c.ueq.s

Floating-point Compare (.ueq.s)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0		
						fmt					ft					fs							cond										
0	1	0	0	0	1	1	0	0	0	0																							
6						5					5					5					2		4										

FPU

Syntax:

```
c.ueq.s fs, ft
```

Description:

The values of the FPU registers specified by *fs* and *ft* are interpreted as single-precision floating-point numbers and arithmetically compared. The result is obtained by combining this comparison with the condition *cond* specified in the instruction (*fs* is equal to *ft*, or *fs* is not a number or *ft* is not a number). If either operand is not a number (Signaling NaN), an Invalid Operation exception occurs. After a delay of one instruction, a floating-point coprocessor branch instruction test can be performed.

The comparison is exact, and will not generate an overflow or underflow. The result can be one of four mutually exclusive relationships (less than, equal, greater than, or unordered). The last case occurs when one or both of the operands are not a number (NaN). The comparison operation ignores any sign appended to a zero value (in other words, $+0 = -0$). This instruction is valid for single-precision floating-point format only.

Operation:

```
if (NaN (ValueFPR(fs, s)) || NaN (ValueFPR(ft, s))) then
    less ← false
    equal ← false
    unordered ← true
    if (cond3 || SNaN(ValueFPR(fs, s)) || SNaN(ValueFPR(ft, s))) then
        SignalException(InvalidOperation)
    endif
else
    less ← ValueFPR(fs, s) < ValueFPR(ft, s)
    equal ← ValueFPR(fs, s) = ValueFPR(ft, s)
    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 exception

Floating-Point exception

FPU Exceptions:

Unimplemented Operation exception(for example, .d)

Invalid Operation exception

c.olt.s

Floating-point Compare (.olt.s)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0			
						fmt					ft					fs					0 0 0 0 0		1 1		cond									
0	1	0	0	0	1	1	0	0	0	0																								
6						5					5					5					2		4											

FPU

Syntax:

```
c.olt.s fs, ft
```

Description:

The values of the FPU registers specified by *fs* and *ft* are interpreted as single-precision floating-point numbers and arithmetically compared. The result is obtained by combining this comparison with the condition *cond* specified in the instruction (*fs* is less than *ft*). If either operand is not a number (Signaling NaN), an Invalid Operation exception occurs. After a delay of one instruction, a floating-point coprocessor branch instruction test can be performed.

The comparison is exact, and will not generate an overflow or underflow. The result can be one of four mutually exclusive relationships (less than, equal, greater than, or unordered). The last case occurs when one or both of the operands are not a number (NaN). The comparison operation ignores any sign appended to a zero value (in other words, $+0 = -0$). This instruction is valid for single-precision floating-point format only.

Operation:

```
if (NaN (ValueFPR(fs, s)) || NaN (ValueFPR(ft, s))) then
    less ← false
    equal ← false
    unordered ← true
    if (cond3 || SNaN(ValueFPR(fs, s)) || SNaN(ValueFPR(ft, s))) then
        SignalException(InvalidOperation)
    endif
else
    less ← ValueFPR(fs, s) < ValueFPR(ft, s)
    equal ← ValueFPR(fs, s) = ValueFPR(ft, s)
    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 exception

Floating-Point exception

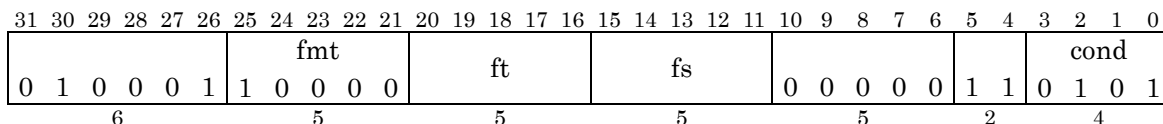
FPU Exceptions:

Unimplemented Operation exception(for example, .d)

Invalid Operation exception

c.ult.s

Floating-point Compare (.ult.s)



FPU

Syntax:

```
c.ult.s fs, ft
```

Description:

The values of the FPU registers specified by *fs* and *ft* are interpreted as single-precision floating-point numbers and arithmetically compared. The result is obtained by combining this comparison with the condition *cond* specified in the instruction (*fs* is less than *ft*, or *fs* is not a number or *ft* is not a number). If either operand is not a number (Signaling NaN), an Invalid Operation exception occurs. After a delay of one instruction, a floating-point coprocessor branch instruction test can be performed.

The comparison is exact, and will not generate an overflow or underflow. The result can be one of four mutually exclusive relationships (less than, equal, greater than, or unordered). The last case occurs when one or both of the operands are not a number (NaN). The comparison operation ignores any sign appended to a zero value (in other words, $+0 = -0$). This instruction is valid for single-precision floating-point format only.

Operation:

```
if (NaN (ValueFPR(fs, s)) || NaN (ValueFPR(ft, s))) then
    less ← false
    equal ← false
    unordered ← true
    if (cond3 || SNaN(ValueFPR(fs, s)) || SNaN(ValueFPR(ft, s))) then
        SignalException(InvalidOperation)
    endif
else
    less ← ValueFPR(fs, s) < ValueFPR(ft, s)
    equal ← ValueFPR(fs, s) = ValueFPR(ft, s)
    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 exception

Floating-Point exception

FPU Exceptions:

Unimplemented Operation exception(for example, .d)

Invalid Operation exception

c.ole.s

Floating-point Compare (.ole.s)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0		
						fmt					ft					fs							cond										
0	1	0	0	0	1	1	0	0	0	0																							
6						5					5					5					2		4										

FPU

Syntax:

```
c.ole.s fs,ft
```

Description:

The values of the FPU registers specified by *fs* and *ft* are interpreted as single-precision floating-point numbers and arithmetically compared. The result is obtained by combining this comparison with the condition *cond* specified in the instruction (*fs* is less than or equal to *ft*). If either operand is not a number (Signaling NaN), an Invalid Operation exception occurs. After a delay of one instruction, a floating-point coprocessor branch instruction test can be performed.

The comparison is exact, and will not generate an overflow or underflow. The result can be one of four mutually exclusive relationships (less than, equal, greater than, or unordered). The last case occurs when one or both of the operands are not a number (NaN). The comparison operation ignores any sign appended to a zero value (in other words, $+0 = -0$). This instruction is valid for single-precision floating-point format only.

Operation:

```
if (NaN (ValueFPR(fs, s)) || NaN (ValueFPR(ft, s))) then
    less ← false
    equal ← false
    unordered ← true
    if (cond3 || SNaN(ValueFPR(fs, s)) || SNaN(ValueFPR(ft, s))) then
        SignalException(InvalidOperation)
    endif
else
    less ← ValueFPR(fs, s) < ValueFPR(ft, s)
    equal ← ValueFPR(fs, s) = ValueFPR(ft, s)
    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 exception

Floating-Point exception

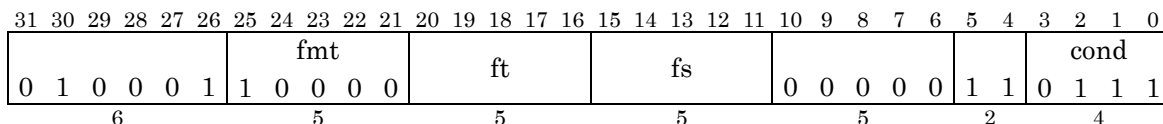
FPU Exceptions:

Unimplemented Operation exception(for example, .d)

Invalid Operation exception

c.ule.s

Floating-point Compare (.ule.s)



FPU

Syntax:

```
c.ule.s fs,ft
```

Description:

The values of the FPU registers specified by *fs* and *ft* are interpreted as single-precision floating-point numbers and arithmetically compared. The result is obtained by combining this comparison with the condition *cond* specified in the instruction (*fs* is less than or equal to *ft*, or *fs* is not a number or *ft* is not a number). If either operand is not a number (Signaling NaN), an Invalid Operation exception occurs. After a delay of one instruction, a floating-point coprocessor branch instruction test can be performed. The comparison is exact, and will not generate an overflow or underflow. The result can be one of four mutually exclusive relationships (less than, equal, greater than, or unordered). The last case occurs when one or both of the operands are not a number (NaN). The comparison operation ignores any sign appended to a zero value (in other words, $+0 = -0$). This instruction is valid for single-precision floating-point format only.

Operation:

```
if (NaN(ValueFPR(fs, s)) || NaN (ValueFPR(ft, s))) then
    less ← false
    equal ← false
    unordered ← true
    if (cond3 || SNaN(ValueFPR(fs, s)) || SNaN(ValueFPR(ft, s))) then
        SignalException(InvalidOperation)
    endif
else
    less ← ValueFPR (fs,s) < ValueFPR (ft,s)
    equal ← ValueFPR (fs,s) = ValueFPR (ft,s)
    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 exception

Floating-Point exception

FPU Exceptions:

Unimplemented Operation exception(for example, .d)

Invalid Operation exception

c.sf.s

Floating-point Compare (.sf.s)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0		
						fmt					ft					fs					cond												
0	1	0	0	0	1	1	0	0	0	0																							
6						5					5					5					2		4										

FPU

Syntax:

```
c.sf.s fs, ft
```

Description:

The values of the FPU registers specified by *fs* and *ft* are interpreted as single-precision floating-point numbers and arithmetically compared. The result is obtained by combining this comparison with the condition *cond* specified in the instruction (which is always false). If either operand is not a number (NaN), an Invalid Operation exception occurs. After a delay of one instruction, a floating-point coprocessor branch instruction test can be performed.

The comparison is exact, and will not generate an overflow or underflow. The result can be one of four mutually exclusive relationships (less than, equal, greater than, or unordered). The last case occurs when one or both of the operands are not a number (NaN). The comparison operation ignores any sign appended to a zero value (in other words, $+0 = -0$). This instruction is valid for single-precision floating-point format only.

Operation:

```
if (NaN (ValueFPR(fs, s)) || NaN (ValueFPR(ft, s))) then
    less ← false
    equal ← false
    unordered ← true
    if (cond3 || SNaN(ValueFPR(fs, s)) || SNaN(ValueFPR(ft, s))) then
        SignalException(InvalidOperation)
    endif
else
    less ← ValueFPR(fs, s) < ValueFPR(ft, s)
    equal ← ValueFPR(fs, s) = ValueFPR(ft, s)
    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 exception

Floating-Point exception

FPU Exceptions:

Unimplemented Operation exception(for example, .d)

Invalid Operation exception

c.ngle.s

Floating-point Compare (.ngle.s)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
						fmt					ft					fs					cond										
0	1	0	0	0	1	1	0	0	0	0											0	0	0	0	0	1	1	1	0	0	1
6						5					5					5					2		4								

FPU

Syntax:

```
c.ngle.s fs,ft
```

Description:

The values of the FPU registers specified by *fs* and *ft* are interpreted as single-precision floating-point numbers and arithmetically compared. The result is obtained by combining this comparison with the condition *cond* specified in the instruction (*fs* is not greater than *ft*, *fs* is not less than *ft*, or *fs* is not equal to *ft*). If either operand is not a number (NaN), an Invalid Operation exception occurs. After a delay of one instruction, a floating-point coprocessor branch instruction test can be performed.

The comparison is exact, and will not generate an overflow or underflow. The result can be one of four mutually exclusive relationships (less than, equal, greater than, or unordered). The last case occurs when one or both of the operands are not a number (NaN). The comparison operation ignores any sign appended to a zero value (in other words, +0 = -0). This instruction is valid for single-precision floating-point format only.

Operation:

```
if (NaN (ValueFPR(fs, s)) || NaN (ValueFPR(ft, s))) then
    less ← false
    equal ← false
    unordered ← true
    if (cond3 || SNaN(ValueFPR(fs, s)) || SNaN(ValueFPR(ft, s))) then
        SignalException(InvalidOperation)
    endif
else
    less ← ValueFPR(fs, s) < ValueFPR(ft, s)
    equal ← ValueFPR(fs, s) = ValueFPR(ft, s)
    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 exception

Floating-Point exception

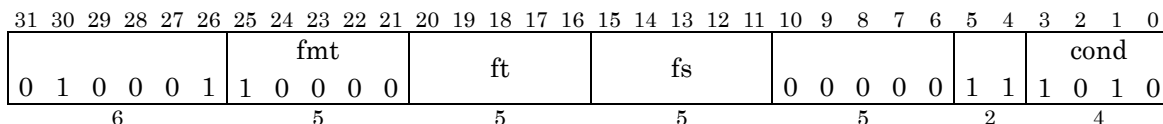
FPU Exceptions:

Unimplemented Operation exception(for example, .d)

Invalid Operation exception

c.seq.s

Floating-point Compare (.seq.s)



FPU

Syntax:

```
c.seq.s fs, ft
```

Description:

The values of the FPU registers specified by *fs* and *ft* are interpreted as single-precision floating-point numbers and arithmetically compared. The result is obtained by combining this comparison with the condition *cond* specified in the instruction (*fs* and *ft* are equal). If either operand is not a number (NaN), an Invalid Operation exception occurs. After a delay of one instruction, a floating-point coprocessor branch instruction test can be performed.

The comparison is exact, and will not generate an overflow or underflow. The result can be one of four mutually exclusive relationships (less than, equal, greater than, or unordered). The last case occurs when one or both of the operands are not a number (NaN). The comparison operation ignores any sign appended to a zero value (in other words, $+0 = -0$). This instruction is valid for single-precision floating-point format only.

Operation:

```
if (NaN (ValueFPR(fs, s)) || NaN (ValueFPR(ft, s))) then
    less ← false
    equal ← false
    unordered ← true
    if (cond3 || SNaN(ValueFPR(fs, s)) || SNaN(ValueFPR(ft, s))) then
        SignalException(InvalidOperation)
    endif
else
    less ← ValueFPR(fs, s) < ValueFPR(ft, s)
    equal ← ValueFPR(fs, s) = ValueFPR(ft, s)
    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 exception

Floating-Point exception

FPU Exceptions:

Unimplemented Operation exception(for example, .d)

Invalid Operation exception

c.ngl.s

Floating-point Compare (.ngl.s)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0			
						fmt					ft					fs					0 0 0 0 0		1 1		cond									
0	1	0	0	0	1	1	0	0	0	0																								
6						5					5					5					2		4											

FPU

Syntax:

```
c.ngl.s fs, ft
```

Description:

The values of the FPU registers specified by *fs* and *ft* are interpreted as single-precision floating-point numbers and arithmetically compared. The result is obtained by combining this comparison with the condition *cond* specified in the instruction (*fs* is not greater than *ft*, or *fs* is not less than *ft*). If either operand is not a number (NaN), an Invalid Operation exception occurs. After a delay of one instruction, a floating-point coprocessor branch instruction test can be performed.

The comparison is exact, and will not generate an overflow or underflow. The result can be one of four mutually exclusive relationships (less than, equal, greater than, or unordered). The last case occurs when one or both of the operands are not a number (NaN). The comparison operation ignores any sign appended to a zero value (in other words, $+0 = -0$). This instruction is valid for single-precision floating-point format only.

Operation:

```
if (NaN (ValueFPR(fs, s)) || NaN (ValueFPR(ft, s))) then
    less ← false
    equal ← false
    unordered ← true
    if (cond3 || SNaN(ValueFPR(fs, s)) || SNaN(ValueFPR(ft, s))) then
        SignalException(InvalidOperation)
    endif
else
    less ← ValueFPR(fs, s) < ValueFPR(ft, s)
    equal ← ValueFPR(fs, s) = ValueFPR(ft, s)
    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 exception

Floating-Point exception

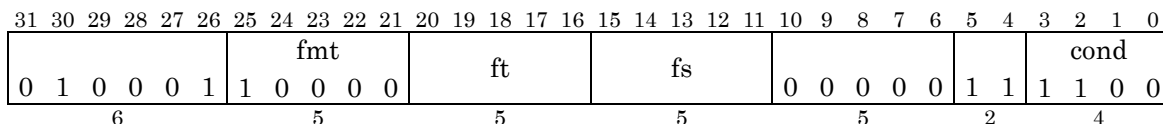
FPU Exceptions:

Unimplemented Operation exception(for example, .d)

Invalid Operation exception

c.lt.s

Floating-point Compare (.lt.s)



FPU

Syntax:

```
c.lt.s fs, ft
```

Description:

The values of the FPU registers specified by *fs* and *ft* are interpreted as single-precision floating-point numbers and arithmetically compared. The result is obtained by combining this comparison with the condition *cond* specified in the instruction (*fs* is less than *ft*). If either operand is not a number (NaN), an Invalid Operation exception occurs. After a delay of one instruction, a floating-point coprocessor branch instruction test can be performed.

The comparison is exact, and will not generate an overflow or underflow. The result can be one of four mutually exclusive relationships (less than, equal, greater than, or unordered). The last case occurs when one or both of the operands are not a number (NaN). The comparison operation ignores any sign appended to a zero value (in other words, $+0 = -0$). This instruction is valid for single-precision floating-point format only.

Operation:

```
if (NaN (ValueFPR(fs, s)) || NaN (ValueFPR(ft, s))) then
    less ← false
    equal ← false
    unordered ← true
    if (cond3 || SNaN(ValueFPR(fs, s)) || SNaN(ValueFPR(ft, s))) then
        SignalException(InvalidOperation)
    endif
else
    less ← ValueFPR(fs, s) < ValueFPR(ft, s)
    equal ← ValueFPR(fs, s) = ValueFPR(ft, s)
    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 exception

Floating-Point exception

FPU Exceptions:

Unimplemented Operation exception(for example, .d)

Invalid Operation exception

c.nge.s

Floating-point Compare (.nge.s)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0			
						fmt					ft					fs					0 0 0 0 0		1 1		cond									
0	1	0	0	0	1	1	0	0	0	0																								
6						5					5					5					2		4											

FPU

Syntax:

```
c.nge.s fs, ft
```

Description:

The values of the FPU registers specified by *fs* and *ft* are interpreted as single-precision floating-point numbers and arithmetically compared. The result is obtained by combining this comparison with the condition *cond* specified in the instruction (*fs* is not greater than *ft*, or *fs* is not equal to *ft*). If either operand is not a number (NaN), an Invalid Operation exception occurs. After a delay of one instruction, a floating-point coprocessor branch instruction test can be performed.

The comparison is exact, and will not generate an overflow or underflow. The result can be one of four mutually exclusive relationships (less than, equal, greater than, or unordered). The last case occurs when one or both of the operands are not a number (NaN). The comparison operation ignores any sign appended to a zero value (in other words, $+0 = -0$). This instruction is valid for single-precision floating-point format only.

Operation:

```
if (NaN (ValueFPR(fs, s)) || NaN (ValueFPR(ft, s))) then
    less ← false
    equal ← false
    unordered ← true
    if (cond3 || SNaN(ValueFPR(fs, s)) || SNaN(ValueFPR(ft, s))) then
        SignalException(InvalidOperation)
    endif
else
    less ← ValueFPR(fs, s) < ValueFPR(ft, s)
    equal ← ValueFPR(fs, s) = ValueFPR(ft, s)
    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 exception

Floating-Point exception

FPU Exceptions:

Unimplemented Operation exception(for example, .d)

Invalid Operation exception

c.le.s

Floating-point Compare (.le.s)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0		
						fmt					ft					fs					0 0 0 0 0		1 1		cond								
0	1	0	0	0	1	1	0	0	0	0																							
6						5					5					5					2		4										

FPU

Syntax:

```
c.le.s fs, ft
```

Description:

The values of the FPU registers specified by *fs* and *ft* are interpreted as single-precision floating-point numbers and arithmetically compared. The result is obtained by combining this comparison with the condition *cond* specified in the instruction (*fs* is less than or equal to *ft*). If either operand is not a number (NaN), an Invalid Operation exception occurs. After a delay of one instruction, a floating-point coprocessor branch instruction test can be performed.

The comparison is exact, and will not generate an overflow or underflow. The result can be one of four mutually exclusive relationships (less than, equal, greater than, or unordered). The last case occurs when one or both of the operands are not a number (NaN). The comparison operation ignores any sign appended to a zero value (in other words, $+0 = -0$). This instruction is valid for single-precision floating-point format only.

Operation:

```
if (NaN (ValueFPR(fs, s)) || NaN (ValueFPR(ft, s))) then
    less ← false
    equal ← false
    unordered ← true
    if (cond3 || SNaN(ValueFPR(fs, s)) || SNaN(ValueFPR(ft, s))) then
        SignalException(InvalidOperation)
    endif
else
    less ← ValueFPR(fs, s) < ValueFPR(ft, s)
    equal ← ValueFPR(fs, s) = ValueFPR(ft, s)
    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 exception

Floating-Point exception

FPU Exceptions:

Unimplemented Operation exception(for example, .d)

Invalid Operation exception

c.ngt.s

Floating-point Compare (.ngt.s)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
						fmt					ft					fs							cond									
0	1	0	0	0	1	1	0	0	0	0																	1	1	1	1	1	1
6						5					5					5					2		4									

FPU

Syntax:

```
c.ngt.s fs, ft
```

Description:

The values of the FPU registers specified by *fs* and *ft* are interpreted as single-precision floating-point numbers and arithmetically compared. The result is obtained by combining this comparison with the condition *cond* specified in the instruction (*fs* is not greater than *ft*). If either operand is not a number (NaN), an Invalid Operation exception occurs. After a delay of one instruction, a floating-point coprocessor branch instruction test can be performed.

The comparison is exact, and will not generate an overflow or underflow. The result can be one of four mutually exclusive relationships (less than, equal, greater than, or unordered). The last case occurs when one or both of the operands are not a number (NaN). The comparison operation ignores any sign appended to a zero value (in other words, $+0 = -0$). This instruction is valid for single-precision floating-point format only.

Operation:

```
if (NaN (ValueFPR(fs, s)) || NaN (ValueFPR(ft, s))) then
    less ← false
    equal ← false
    unordered ← true
    if (cond3 || SNaN(ValueFPR(fs, s)) || SNaN(ValueFPR(ft, s))) then
        SignalException(InvalidOperation)
    endif
else
    less ← ValueFPR(fs, s) < ValueFPR(ft, s)
    equal ← ValueFPR(fs, s) = ValueFPR(ft, s)
    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 exception

Floating-Point exception

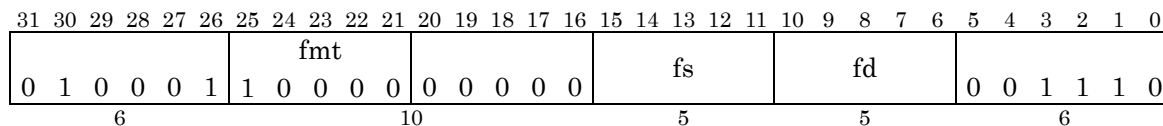
FPU Exceptions:

Unimplemented Operation exception(for example, .d)

Invalid Operation exception

ceil.w.s

Ceiling to Word from Single



FPU

Syntax:

```
ceil.w.s  fd, fs
```

Description:

The value of the FPU register specified by fs is interpreted as a single-precision 32-bit binary floating-point number and arithmetically converted to a 32-bit fixed-point number. The result is rounded towards $+\infty$, regardless of the current rounding mode, and stored in the FPU register specified by fd. When the source operand is infinitely large or not a number (NaN), or outside the range $-2^{31} \sim 2^{31}-1$, an Invalid Operation exception occurs. If Invalid Operation exceptions are disabled, no exception will occur. If the result of the operation is a NaN, or if it is beyond the range of positive numbers, $2^{31}-1$ is returned. If the result is beyond the range of negative numbers, -2^{31} is returned. This instruction is valid for single-precision floating-point format only.

Operation:

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

Exceptions:

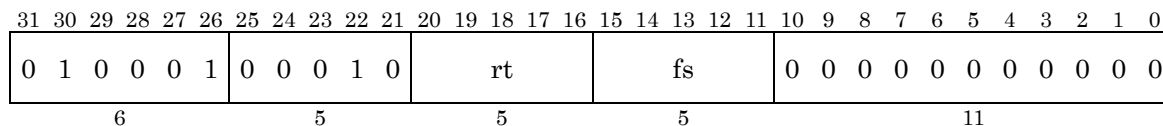
```
Coprocessor Unusable exception
Floating-Point exception
```

FPU Exceptions:

```
Unimplemented Operation exception(for example, .d)
Invalid Operation exception
Inexact exception
```

cfc1

Move Control from FPU

MIPS I
FPU**Syntax:**`cfc1 rt, fs`**Description:**

The contents of the FPU control register `fs` are transferred to the CPU register `rt`. This instruction is only defined when `fs` is 0 or 31.

Restrictions:

Only 0 or 31 is valid for `fs`.

Operation:

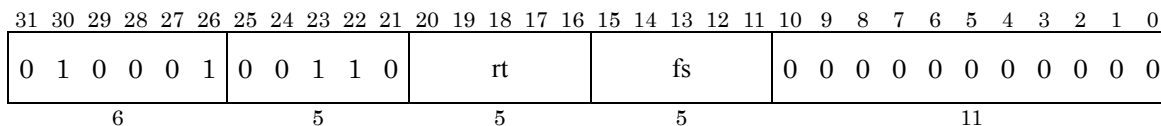
I:
temp ← FCR[fs]
I+1:
GPR[rt] ← temp

Exceptions:

Coprocessor Unusable exception

ctc1

Move control to FPU

MIPS I
FPU**Syntax:**`ctc1 rt, fs`**Description:**

Transfers the contents of CPU register `rt` to FPU control register `fs`.

This instruction is only defined for cases in which `fs` is 31.

If a 1 is moved both to one of the Cause bits and to its corresponding Enable bit, then an interrupt or an exception will occur. After this load instruction, the FPU pipeline stalls until the result is reflected by control.

Restrictions:

For `fs`, only 31 is valid.

After a `ctc1` instruction which causes an exception, another consecutive `ctc1` instruction cannot be executed.

After execution of a `ctc1` instruction, at least one `nop` is necessary until an instruction which references `FCR31` is executed.

If a Cause bit changes from 1 to 0 and the corresponding Enable bit changes from 0 to 1, an unexpected FPU exception will be raised. This can be prevented by dividing the `CTC1` instruction into two instances when setting `FCR31`. In the first instance of `ctc1`, zero out `FCR31`, and in the second instance of `ctc1`, set it to the target value.

Operation:

```

I :
    temp ← GPR[rt]
I + 1 :
    FCR[fs] ← temp
    COC[1] ← FCR[fs]23

```

Exceptions:

Coprocessor Unusable exception

Floating-Point exception

FPU exceptions:

Unimplemented Operation exception

Invalid Operation exception

Division By Zero exception

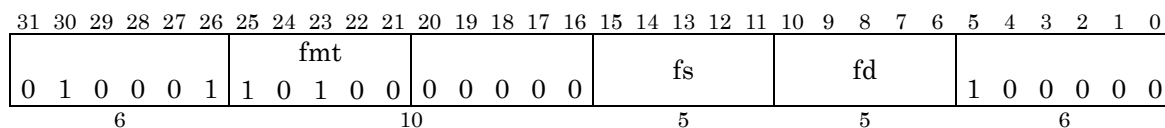
Inexact exception

Overflow exception

Underflow exception

cvt.s.w

Convert to Single from Word



FPU

Syntax:

```
cvt.s.w fd, fs
```

Description:

The value of the FPU register specified by `fs` is interpreted as being a single binary fixed-point number and arithmetically converted to a single-precision floating-point number. The result is rounded according to the current rounding mode and stored in the FPU register specified by `fd`.

This instruction is valid for 32-bit fixed-point format only.

Operation:

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

Exceptions:

```
Coprocessor Unusable exception
```

```
Floating-Point exception
```

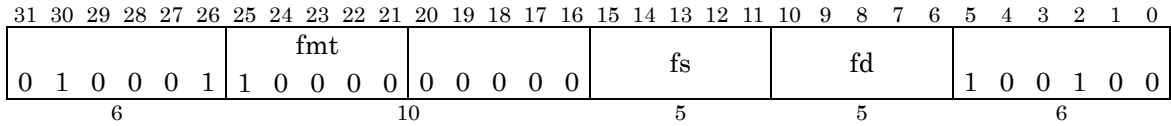
FPU Exceptions:

```
Unimplemented Operation exception(for example, .d)
```

```
Inexact exception
```

cvt.w.s

Convert to Word from Single



FPU

Syntax:

```
cvt.w.s  fd, fs
```

Description:

The value of the FPU register specified by fs is interpreted as being a single-precision 32-bit binary floating-point number and arithmetically converted to a 32-bit fixed-point number. The result is rounded according to the current rounding mode, and stored in the FPU register specified by fd.

When the source operand is infinitely large or not a number (NaN), or the rounded result is outside the range $-2^{31} \sim 2^{31}-1$, an Invalid Operation exception occurs. If Invalid Operation exceptions are disabled, no exception will occur. If the result of the operation is a NaN, or if it is beyond the range of positive numbers, $2^{31}-1$ is returned. If the result is beyond the range of negative numbers, -2^{31} is returned.

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

Operation:

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

Exceptions:

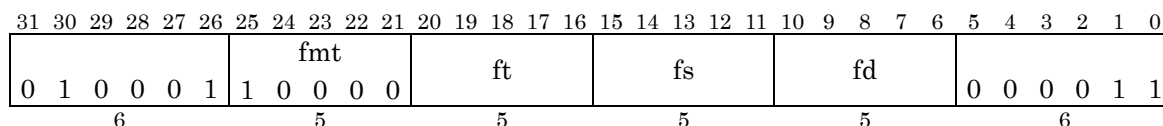
```
Coprocessor Unusable exception
Floating-Point exception
```

FPU Exceptions:

```
Unimplemented Operation exception(for example, .d)
Invalid Operation exception
Inexact exception
```

div.s

Floating-point Divide



FPU

Syntax:

```
div.s  fd, fs, ft
```

Description:

The values of the FPU registers specified by fs and ft are interpreted as being single-precision 32-bit binary floating-point numbers and arithmetically divided. The result is calculated as if the precision were infinite and rounded for the specified format according to the current rounding mode, then stored in the FPU register specified by fd. Division is performed as an arithmetic operation. Consequently, an Invalid Operation exception occurs if the operand is not a number (NaN).

This instruction is valid for single-precision floating-point format only.

Operation:

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

Exceptions:

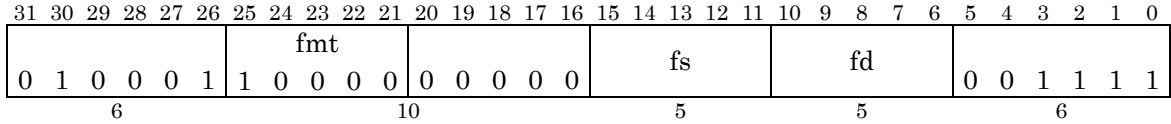
```
Coprocessor Unusable exception
Floating-Point exception
```

FPU Exceptions:

```
Unimplemented Operation exception(for example, .d)
Invalid Operation exception
Division By Zero exception
Inexact exception
Overflow exception
Underflow exception
```

floor.w.s

Floor to Word from Single



FPU

Syntax:

```
floor.w.s fd, fs
```

Description:

The value of the FPU register specified by fs is interpreted as a single-precision 32-bit binary floating-point number and arithmetically converted to a 32-bit fixed-point number. The result is rounded towards $-\infty$ regardless of the current rounding mode, and stored in the FPU register specified by fd.

When the source operand is infinitely large or not a number (NaN), or the rounded result is outside the range $-2^{31} \sim 2^{31}-1$, an Invalid Operation exception occurs. If Invalid Operation exceptions are disabled, no exception will occur. If the result of the operation is a NaN, or if it is beyond the range of positive numbers, $2^{31}-1$ is returned. If the result is beyond the range of negative numbers, -2^{31} is returned.

This instruction is valid for single-precision floating-point format only.

Operation:

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

Exceptions:

```
Coprocessor Unusable exception
```

```
Floating-Point exception
```

FPU Exceptions:

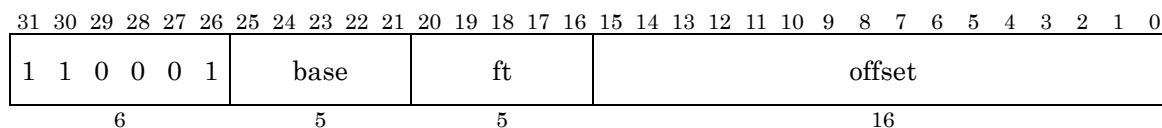
```
Unimplemented Operation exception (for example, .d)
```

```
Invalid Operation exception
```

```
Inexact exception
```

lwc1

Load Word to FPU

MIPS I
FPU**Syntax:**

```
lwc1 ft,offset(base)
```

Description:

The indicated 16-bit offset is sign-extended and added to the contents of the base register to generate an address. The contents of the memory word at this address are stored in FPU register ft.

If the low-order 2-bits of the address are not zero, an Address Error exception occurs.

Operation:

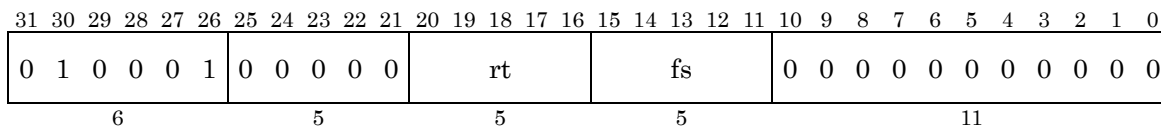
```
vAddr ← sign_extend(offset) + GPR[base];
if (vAddr1,0 ≠ 02) then
    SignalException(AddressError)
endif
(pAddr, CCA) ← AddressTranslation(vAddr, DATA, LOAD)
memword ← LoadMemory(CCA, WORD, pAddr, vAddr, DATA)
StoreFPR(ft, UNINTERPRETED_WORD, memword)
```

Exceptions:

```
Coprocessor Unusable exception
Address Error exception
Bus Error exception
```

mfc1

Move From FPU

MIPS I
FPU**Syntax:**

```
mfc1 rt, fs
```

Description:

The contents of the FPU general-purpose register fs are transferred to the CPU register rt.

Operation:

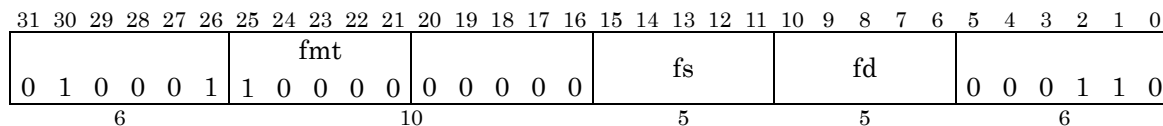
```
I:
    data ← ValueFPR(fs, UNINTERPRETED_WORD)
I+1:
    GPR[rt] ← data
```

Exceptions:

Coprocessor Unusable exception

mov.s

Floating-point Move



FPU

Syntax:

```
mov.s fd, fs
```

Description:

The value of the FPU register specified by fs is interpreted as being a single precision 32-bit binary floating-point number and stored in the FPU register specified by fd.

The mov.s instruction is not an arithmetic operation. Therefore, no IEEE 754 floating-point exceptions will be generated.

This instruction is valid for single-precision floating-point format only.

Operation:

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

Exceptions:

```
Coprocessor Unusable exception
```

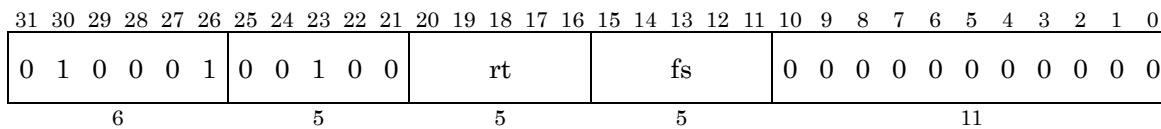
```
Floating-Point exception
```

FPU Exceptions:

```
Unimplemented Operation exception(for example, .d)
```

mtc1

Move To FPU

MIPS I
FPU**Syntax:**

```
mtc1 rt, fs
```

Description:

The contents of the CPU register `rt` are transferred to the FPU general-purpose register `fs`.

Operation:

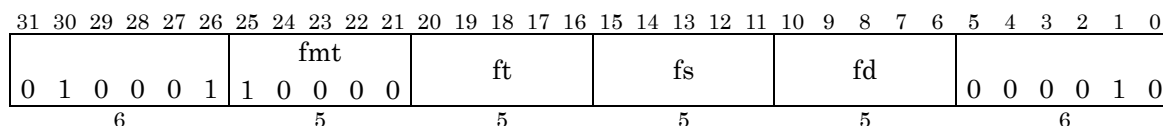
```
I:
    data ← GPR[rt]31..0
I+1:
    StoreFPR(fs, UNINTERPRETED_WORD, data)
```

Exceptions:

Coprocessor Unusable exception

mul.s

Floating-point Multiply



FPU

Syntax:

```
mul.s  fd, fs, ft
```

Description:

The values of the FPU registers specified by fs and ft are interpreted as being in the format specified by fmt and arithmetically multiplied. The result is calculated as if the precision were infinite and rounded for the specified format according to the current rounding mode, then stored in the FPU register specified by fd.

Multiplication is performed as an arithmetic operation. Consequently, an Invalid Operation exception occurs if the operand is not a number (NaN).

This instruction is valid for single-precision floating-point format only.

Operation:

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

Exceptions:

```
Coprocessor Unusable exception
```

```
Floating-Point exception
```

FPU Exceptions:

```
Unimplemented Operation exception(for example, .d)
```

```
Invalid Operation exception
```

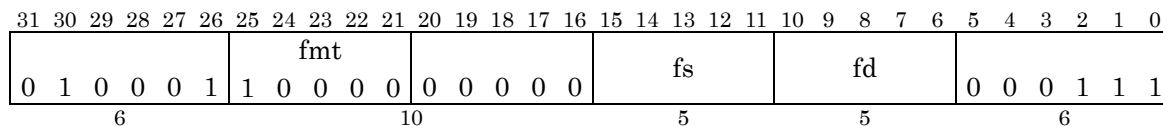
```
Inexact exception
```

```
Overflow exception
```

```
Underflow exception
```

neg.s

Floating-point Negate



FPU

Syntax:

```
neg.s  fd, fs
```

Description:

The value of the FPU register specified by `fs` is interpreted as being a single precision 32-bit binary floating-point number and arithmetically negated. The result is stored in the FPU register specified by `fd`.

Negation is performed as an arithmetic operation. Consequently, an Invalid Operation exception occurs if the operand is not a number (NaN).

This instruction is valid for single-precision floating-point format only.

Operation:

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

Exceptions:

```
Coprocessor Unusable exception
```

```
Floating-Point exception
```

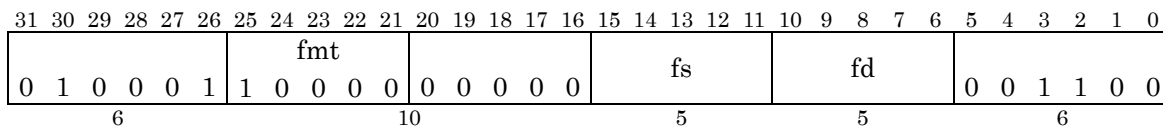
FPU Exceptions:

```
Unimplemented Operation exception(for example, .d)
```

```
Invalid Operation exception
```

round.w.s

Round to Word from Single



FPU

Syntax:

```
round.w.s  fd, fs
```

Description:

The value of the FPU register specified by fs is interpreted as being in the format specified by fmt and arithmetically converted to a 32-bit fixed-point number. The result is rounded to an approximate value or to an even number (when it is an intermediate value) regardless of the current rounding mode, then stored in the FPU register specified by fd.

When the source operand is infinitely large or not a number (NaN), or the rounded result is outside the range $-2^{31} \sim 2^{31}-1$, an Invalid Operation exception occurs. If Invalid Operation exceptions are disabled, no exception will occur. If the result of the operation is a NaN, or if it is beyond the range of positive numbers, $2^{31}-1$ is returned. If the result is beyond the range of negative numbers, -2^{31} is returned. This instruction is valid for single-precision floating-point format only.

Operation:

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

Exceptions:

```
Coprocessor Unusable exception
```

```
Floating-Point exception
```

FPU Exceptions:

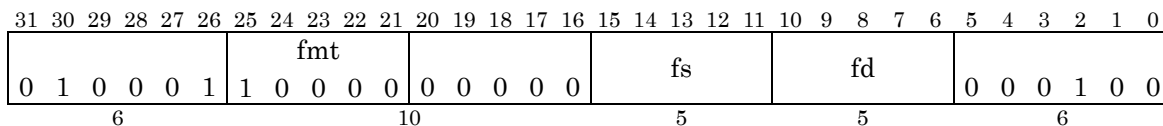
```
Unimplemented Operation exception(for example, .d)
```

```
Invalid Operation exception
```

```
Inexact exception
```

sqrt.s

Floating-point Square Root



FPU

Syntax:

```
sqrt.s  fd, fs
```

Description:

The value of the FPU register specified by fs is interpreted as a single-precision 32-bit binary floating-point number and its positive square root is calculated. The result is calculated as if the precision were infinite and rounded for the specified format according to the current rounding mode, then stored in the FPU register specified by fd. If the value of fs is -0, the result will also be -0.

Square root is performed as an arithmetic operation. Consequently, an Invalid Operation exception occurs if the operand is not a number (NaN).

This instruction is valid for single-precision floating-point format only.

Operation:

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

Exceptions:

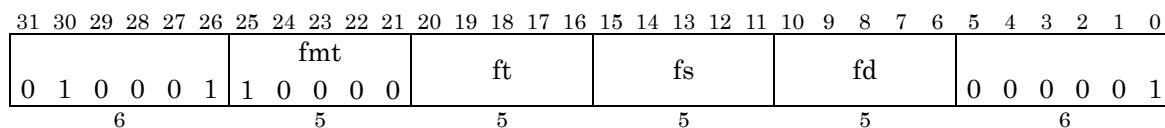
```
Coprocessor Unusable exception
Floating-Point exception
```

FPU Exceptions:

```
Unimplemented Operation exception(for example, .d)
Invalid Operation exception
Inexact exception
```

sub.s

Floating-point Subtract



FPU

Syntax:

```
sub.s  fd, fs, ft
```

Description:

The values of the FPU registers specified by `fs` and `ft` are interpreted as a single-precision 32-bit binary floating-point number and arithmetically subtracted. The result is calculated as if the precision were infinite and rounded for the specified format according to the current rounding mode, then stored in the FPU register specified by `fd`. Subtraction is performed as an arithmetic operation. Consequently, an Invalid Operation exception occurs if the operand is not a number (NaN). This instruction is valid for single-precision floating-point format only.

Operation:

$$\text{StoreFPR}(\text{fd}, \text{fmt}, \text{ValueFPR}(\text{fs}, \text{fmt}) - \text{ValueFPR}(\text{ft}, \text{fmt}))$$
Exceptions:

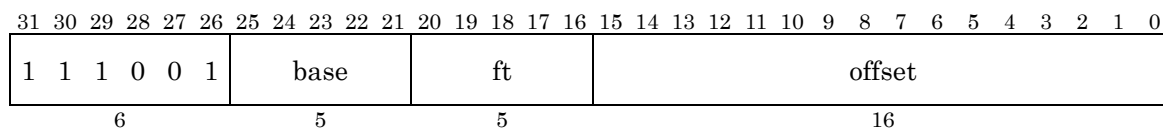
```
Coprocessor Unusable exception
Floating-Point exception
```

FPU Exceptions:

```
Unimplemented Operationexception(for example, .d)
Invalid Operation exception
Inexact exception
Overflow exception
Underflow exception
```

swc1

Store Word from FPU

MIPS I
FPU**Syntax:**

```
swc1 ft,offset(base)
```

Description:

The indicated 16-bit offset is sign-extended and added to the contents of the base register to generate an address. The contents of the FPU register ft are stored in memory at this address.

Bus error exceptions will not be reported because writing is done via a buffer. If a bus error occurs, it is handled by the system as an interrupt.

If the low-order 2-bits of the address are not zero, an Address Error exception occurs.

Operation:

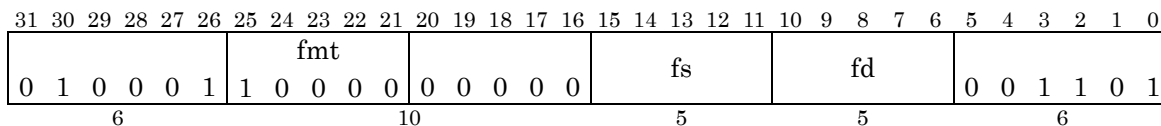
```
vAddr ← sign_extend(offset) + GPR[base]
if (vAddr1:0 ≠ 02) then
    SignalException(AddressError)
endif
(pAddr, CCA) ← AddressTranslation(vAddr, DATA, STORE)
dataword ← ValueFPR(ft, UNINTERPRETED_WORD)
StoreMemory(CCA, WORD, dataword, pAddr, vAddr, DATA)
```

Exceptions:

```
Coprocessor Unusable exception
Address Error exception
```

trunc.w.s

Truncate to Word from Single



FPU

Syntax:

```
trunc.w.s  fd, fs
```

Description:

The value of the FPU register specified by fs is interpreted as a single-precision 32-bit binary floating-point number and arithmetically converted to a 32-bit fixed-point number. The result is rounded towards 0 regardless of the current rounding mode, and is stored in the FPU register specified by fd.

When the source operand is infinitely large or is not a number (NaN), or the rounded result is outside the range $-2^{31} \sim 2^{31}-1$, an Invalid Operation exception occurs. If Invalid Operation exceptions are disabled, no exception will occur. If the result of the operation is a NaN, or if it is beyond the range of positive numbers, $2^{31}-1$ is returned. If the result is beyond the range of negative numbers, -2^{31} is returned.

This instruction is valid for single-precision floating-point format only.

Operation:

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

Exceptions:

```
Coprocessor Unusable exception
```

```
Floating-Point exception
```

FPU Exceptions:

```
Unimplemented Operation exception (for example, .d)
```

```
Invalid Operation exception
```

```
Inexact exception
```

FPU Instruction Set Code Tables

Opcode	Bit 31-29	Bit 28-26							
		0	1	2	3	4	5	6	7
0		SPECIAL	REGIM M	J	JAL	BEQ	BNE	BLEZ	BGTZ
1		ADDI	ADDIU	SLTI	SLTIU	ANDI	ORI	XORI	LUI
2		COP0	COP1	COP2	*	BEQL	BNEL	BLEZL	BGTZL
3		VFPU0	VFPU1	*	VFPU3	SPECIAL2	*	*	SPECIAL3
4		LB	LH	LWL	LW	LBU	LHU	LWR	*
5		SB	SH	SWL	SW	*	*	SWR	CACHE
6		LL	LWC1	LWC2	*	VFPU4	LQUC2	LQC2	VFPU5
7		SC	SWC1	SWC2	*	VFPU6	SQUC2	SQC2	VFPU7

COP1 rs	Bit 25-24	Bit 23-21							
		0	1	2	3	4	5	6	7
0		MFC1		CFC1		MTC1		CTC1	
1		BC							
2		S				W			
3									

COP1 rt (BC)	Bit 20-19	Bit 18-16							
		0	1	2	3	4	5	6	7
0		BC1F	BC1T	BC1FL	BC1TL				
1									
2									
3									

SPECIAL2 func	Bit 5-3	Bit 2-0							
		0	1	2	3	4	5	6	7
0		ADD	SUB	MUL	DIV	SQRT	ABS	MOV	NEG
1						ROUND.W	TRUNC.W	CEIL.W	FLOOR.W
2									
3									
4		CVT.S				CVT.W			
5									
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

	MIPS-I			MIPS-II
	FPU			ALLEGREX™