

# Floating Point Instructions

# Floating Point

- There are several coprocessors in the MIPS system. Where instructions can operate on more than one coprocessor, z is used.
- The floating point coprocessor is 1 (the digit one not the letter l)

# Floating Point Instructions

- Cautions on converting
  - For FP operands, MIPS expects registers to be FP registers
  - For example

```
cvt.s.w fd,fs    cvt.s.w $f6,$f8
```

converts the integer in floating point register f8 to a single precision float

```
cvt.s.d fd,fs
```

converts the double in fs to a single precision

# Floating Point Instructions

`cvt.w.s fd,fs`      `cvt.w.s $f6,$f8`

converts the single precision float in `fs` to an integer in `fd`

`cvt.w.d fd,fs`      `cvt.w.d $f6,$f8`

converts the double in `fs` to an integer in `fd`

# Floating Point Instructions

- To get an int into a float register, can move it from an integer register to a float register, or put it in memory then use a l.s (there is also lwcl \$f#,address : note that it's the letter l on the front and the digit 1 on the rear)

mtc1 \$5,\$f6 # \$f6=\$5 mt=> move to

Note: this is the only instruction with the source on the left

*or*

```
sw    $5,int_loc
```

```
l.s   $f6,int_loc
```

# Floating Point Instructions

```
mfcz  rt,rd  # coprocessor z (normally 1)
        # mf=>move from
        #register rd is moved into CPU
        # register rt
```

```
mfc1  $5,$f6    # $5=$f6
```

# Floating Point Instructions

`mfc1.d rdest,rsrc`

move floating point registers `rsc` & `rsc+1`  
into CPU registers `rdest` & `rdest+1`

# FP Compares

- Comparisons are

c.x.s and c.x.d

x = eq, neq, lt, le, gt, and ge (for >=)

Sets floating point (processor 1) condition  
flag to a 1 on true

# FP Branches

bczt label

bczf label

Branch to label if processor z's condition flag is 1(t) or 0(f). Note FP register uses z=1

# FP Compares

- Let's say we wanted to branch to NXT if the contents in single precision fp registers 6 and 10 are equal

```
c.eq.s $f6,$f10
```

```
bc1t    NXT
```

# Question

- What will the following program output?

```
.data  
flt: .space 4 # declare some space  
nl: .ascii "\n"
```

# These next two lines always start the main program

```
    .text  
    .globl main  
main:    la    $6,flt  
        add   $5,$0,$0
```

```
addi $5,$5,-0x8000
lui  $5,0xC080
sw   $5,0($6)
l.s  $f12,0($6)
li   $v0,2
syscall
li   $v0,4
la   $a0,nl           # print a new line
syscall

li   $v0, 10 # system call code for exit
syscall
```

- The instruction `addi$5,$5,-0x8000` puts -32768 in R5, but the `lui` puts 0xc080 in the upper 16 bits and zeros out the lower 16 so the `addi` has no effect
- -4.000000000

# Question

- For the small change shown, what happens (the addi was swapped with the lui.)

```
lui    $5,0xC080
addi   $5,$5,-0x8000

sw     $5,0($6)
l.s    $f12,0($6)
li     $v0,2
syscall

li     $v0,4
la     $a0,nl                # print a new line
syscall

li     $v0, 10 # system call code for exit
syscall
```

- The instruction lui no longer affects the addi  
\$5,\$5,-0x8000  
so now after 0x0c80 is loaded into r5, ffff8000  
is added. This gives 0xc07f8000 or
- -3.99.....

# Question

- What will the following program output?

.data

flt: .float 57.89

nl:.asciiz "\n"

main: li.d \$f12,1.3

li \$v0,3

syscall

li \$v0,4

la \$a0,nl # print a new line

syscall

li \$v0,2

syscall

li \$v0,4

la \$a0,nl # print a new line

syscall

```
l.d $f12,flt
li $v0,3
syscall
li $v0,4
la $a0,nl      # print a new line
syscall
li $v0,2
syscall
li $v0,4
la $a0,nl      # print a new line
syscall
li $v0, 10     # system call code for exit
syscall
```

```
1.3  
-107374184.0000000000000000000000  
2.1770388127595243e-313  
57.889999389648438000  
|
```



# Question

- What will the following program output?

.data

```
int:  .word 2147483647      # a large (maximum) int
lint: .word 2147483648    # maximum + 1
rlint: .word 5679812465455 # even larger
flt:  .float 123456.99      # large single float
lflt: .float 123456789012345.99 # really large
dbl:  .double 12345678901234.99 # a double
ldbl: .double 123456789012345678901234.99
      # a really long double
nl:   .ascii "\n"
```

main:

```
jal newln
```

```
li $v0,1
```

```
lw $a0,int # prints 2147483647, i.e.  $2^{31} - 1$ 
```

```
syscall
```

```
jal newln
```

```
li $v0,1
```

```
lw $a0,int # -2147483647, i.e.  $-2^{31}$ 
```

```
syscall
```

```
jal newln
```

```
li $v0,1
```

```
lw $a0,rhint # 1865700143, i.e. GARBAGE
```

```
syscall
```

```
jal newln
```

```
li $v0,2
```

```
l.s $f12,flt # prints 123456.992188
```

```
syscall
```

```
jal newln
li $v0,2
l.s $f12,lflt # prints 123456788103168.000000
syscall
jal newln
li $v0,3
l.d $f12,dbl # prints 12345678901234.990234
syscall
jal newln
li $v0,3
l.d $f12,ldbl # prints
123456789012345686000000.000000

syscall
```

# Question

- What does the following program output?

```
.data
```

```
float1:    .float 7.5,6.6           # declare some
```

```
float (single precisions)
```

```
result:    .space 4                # allocate space for a
```

```
float result
```

```
int:      .space 4
```

```
nl:      .ascii "\n"
```

.text

.globl main

main:

```
l.s  $f2,float1      # this loads $f2 with 7.5
l.s  $f8,float1+4    # this loads $f8 with 6.6
add.s      $f10,$f8,$f2  # this loads $f10 with $f8+$f2 = 14.1
s.s  $f10,result     # store the result in location result
l.s  $f12,result     # as a check, get the value back
li    $v0,2          # now print it (will output 14.1)
syscall
li    $v0,4
la    $a0,nl         # print a new line
syscall
li    $6,10
sw   $6,int         # store a 10 in memory location int
l.s  $f12,int        # load the 10 into a float register
li    $v0,2         # now print it (will output garbage - why?)
                                # remember $f12 is where the float is to be on
```

```

syscall                # a print_float
li    $v0,4
la    $a0,nl          # print a new line
syscall
cvt.s.w    $f12,$f12 # convert the integer 10 to a float
10.0
li    $v0,2          # now print it (will output 10.0)
syscall
li    $v0,4
la    $a0,nl          # print a new line
syscall
cvt.w.s    $f8,$f8          # convert 6.6 to an
                        #integer (6), notice it truncates

```

```

# la    $5,result      # these next 3 inst's are equiv. to the mfc1
# s.s   $f8,12($5)     # save it in memory
# lw    $a0,12($5)     # get it into a0 to print as an integer
mfc1   $a0,$f8         # preceding two instructions also
                                     #work. This one is move from
                                     # the floating point coprocessor (1) to general
                                     # purpose register $a0

li     $v0,1
syscall

li     $v0,4
la     $a0,nl          # print a new line
syscall

li     $v0, 10         # system call code for exit
syscall

```

# Console



```
2147483647  
-2147483648  
1865700143  
123456.99218750000000000000  
123456788103168.00000000000000000000  
12345678901234.99  
1.2345678901234569e+023
```