MIPS **Floating Point** Architecture

- **Instructions**: Single Precision, Double Precision versions of add, subtract, multiply, divide, compare
  - Single `add.s, sub.s, mul.s, div.s, c.lt.s`
  - Double `add.d, sub.d, mul.d, div.d, c.lt.d`
- **Registers**: MIPS provides 32 32-bit FP reg: $f0, $f1, $f2 …
  - FP data transfers: **l.s**, **l.d**, **s.s**, and **s.d** # **pseudoinstructions**
    - **lwc1** [Load word coprocessor 1], **swc1**
  - Double Precision?
    - Even-odd pair of registers (**$f0#$f1**) act as 64-bit register: **$f0**, **$f2**, **$f4**, …

FP Instructions: Arithmetic and Comparison

```
add.s $f0,$f1,$f2 # $f0=$f1+$f2 FP Add (single)
add.d $f0,$f2,$f4 # $f0=$f2+$f4 FP Add (double)
sub.s $f0,$f1,$f2 # $f0=$f1-$f2 FP Subtract (single)
sub.d $f0,$f2,$f4 # $f0=$f2-$f4 FP Subtract (double)
mul.s $f0,$f1,$f2 # $f0=$f1x$f2 FP Multiply (single)
mul.d $f0,$f2,$f4 # $f0=$f2x$f4 FP Multiply (double)
div.s $f0,$f1,$f2 # $f0=$f1÷$f2 FP Divide (single)
div.d $f0,$f2,$f4 # $f0=$f2÷$f4 FP Divide (double)
c.X.s $f0,$f1     # flag1= $f0 X $f1 FP Compare (single)
c.X.d $f0,$f2     # flag1= $f0 X $f2 FP Compare (double)

# where X is: eq (equal), lt (less than), le (less than
# equal) to test flag value:
# bc1t - floating-point branch true [it's digital '1' in
#      bc1t, NOT letter 'l' in bclt; 't' means true]
# bc1f - floating-point branch false (flag1 was set false)
```

FP Instructions: compare / branch

Hints

Comments on compare / branch with floating number

1) Paired instructions are normally used
2) We can compare floating values and branch depending on the results. The compare is done by coprocessor, which generates a 'status' of true or false. Then the special branch instructions, 'branch on coprocessor condition flag' are used to effect the branch.

Compare	Branch
c.condition.s FR, FR e.g. c.eq.s c.lt.s c.le.s Set flag internally	**bc1t** label #branch if flag is true [note: it's digital '1' in bc1t, not letter 'l' in bclt; 't' means true.] **bc1f** label #branch if flag is false [note: 'f' false]

Handling **FP** immediate numbers

Lecture05_Numbers.pdf
Lecture05 [Supplement]_fpNumbers.pdf

1) From users' input:
   Use syscall service 6 - read_float

```
li $v0,6          #input decimal fraction number and save
syscall           #float input is in $f0
mov.s $f12,$f0    #there's also mov.d instruction
```

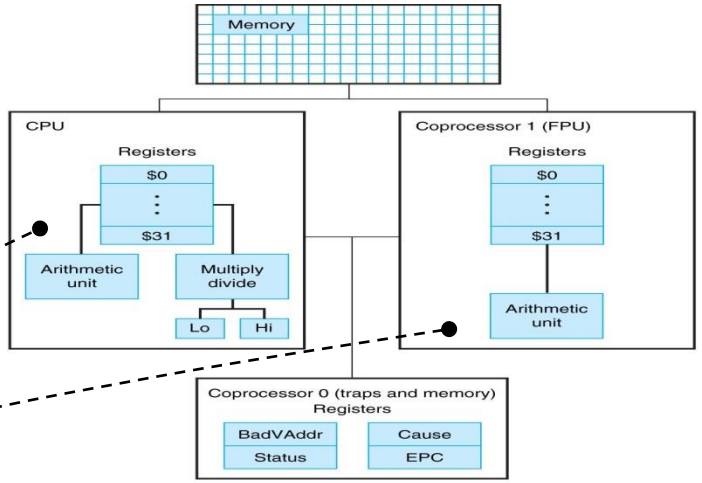
2) Define fp numbers in memory and data transfer between c1 and memory:

```
    .data
pi: .double 3.141592653589793
hundredth: .float 0.01
epsilon: .float 1.0e-7

la $a0, pi
l.d $f12, 0($a0) # pseudoinstructions
```

lwc1 ft, address # Load word coprocessor 1
ldc1 ft, address # Load doublewords coprocessor 1
swc1 ft, address # Store word coprocessor 1
sdc1 ft, address # Store doublewords coprocessor 1
l.s, **l.d**, **s.s**, and **s.d** **# pseudoinstructions for convenience**

'CPU – FPU' data transfer and format conversion



3) li.d:

```
li.s $f7, 1.0
li.d $f8, 2.0
```

Working for **0**: mtc1 $**0**, $f5

But only the **bit pattern**, not the value!
[Working for **0**; e.g. mtc1 $**0**, $f5]

4) FPU to CPU transfer and vice versa:

```
mfc1 dst, fsrc # move from coprocessor 1, dst := fsrc
mtc1 src, fdst # move to fp single reg, fdst := src
mtc1.d src, fdst # move to fp double reg, fdst := src
```

5) Format Conversion: Word pattern to fp number conversion – **value** equivalent

```
cvt.TO.FROM (or vcvt.TO.FROM)
cvt.s.w $f2, $f2   #$f2=convert_from_int_to_sigle($f2)
vcvt.d.w $f2, $f2  #$f2=convert_from_int_to_double($f2)
```

'CPU – FPU' data transfer and format conversion

4) and 5) are used together to transfer and convert an integer in a general purpose register into an equivalent FP value in an FP register.

```
addi    $t1, $0, 2          # Initialise with 2
mtc1    $t1, $f2            # move int to $f2 register
cvt.d.w $f2, $f2            # Convert to fp number
```

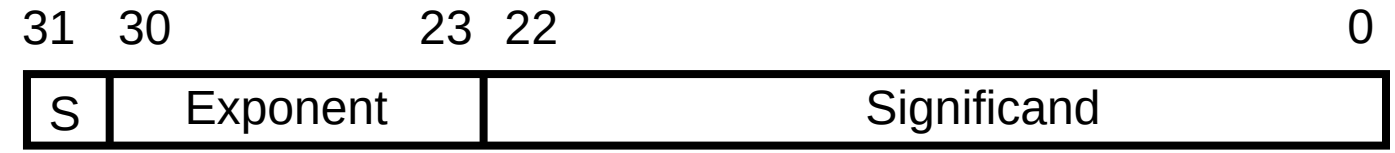
Integer and FP number comparison: Same value but different patterns

Integer and FP representation of 1:

1	Dec	Bin	Hex	Processor
int	$1 = 2^0$	**0**00000000000000000000000000000**1**	0x00000001	CPU
fp.s	$1.0=1.0\times2^0$	**0011111110**0000000000000000000000	0x3f800000	FPU (c1)

Integer and FP representation of 64:

64	Dec	Bin	Hex	Processor
int	$64 = 2^6$	**0**00000000000000000000000**1**000000	0x00000040	CPU
fp.s	$64.0=1.0\times2^6$	**01000010**0000000000000000000000	0x42800000	FPU (c1)



biased-127 representation:

$1.xxxxxxxxx_{two} * 2^{yyyy_{two}}$

Exponent = $yyyy_{two}$ + 127_d