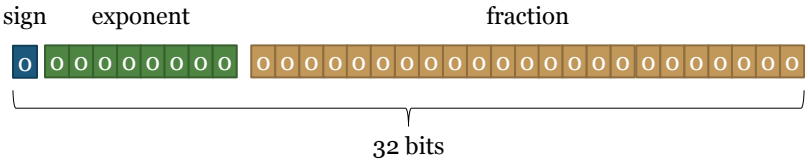
MIPS Floating Point Instructions

CS/COE 447

Why Floating Point?

- Sometimes need very small, or very large numbers? Non-integers?
  - "1.1" or "2.99792E10"
- Not always precise. Not all numbers can be represented
  - Repeating digits
    - E.g., in base 10: 1/3 = 0.33333…
  - Lack of precision
    - E.g., 1.23456789012345678901234567890 may not "fit" in the storage space allocated for the floating point number
- Single precision: 32-bits used to represent a number.
  - "float" in C
- Double precision: 64-bits used to represent a number.
  - "double" in C
- IEEE 754 standard

Single Precision Floating Point Format



- **Sign**: whether # is positive or negative
- **Exponent**: makes value large or small
- **Fraction**: the actual "number"
- Value: $-1^{\mathbf{sign}} \cdot 1.\mathbf{fraction} \cdot 2^{(\mathbf{exponent-127})}$
  - Special values exist for ±∞, NaN (not a number)
  - There are some other exceptions/issues

Overview of MIPS Floating Point Instructions

- MIPS provides several instructions for floating point numbers
  - Arithmetic
  - Data movement (memory and registers)
  - Conditional jumps
- FP instructions work with a different bank of registers
  - Registers are named $f0 to $f31
  - $f0 is not special (can hold any value, not just zero)
  - "Coprocessor 1" tab in MARS
- There are instructions for single precision and double precision numbers (we will only use single precision)
  - Double precision numbers use only even numbered registers
  - Single precision instructions end with ".s" (e.g. add.s)
  - There is generally a corresponding double precision instruction, which ends with ".d"

Arithmetic Instructions

• add.s $f0, $f1, $f2	$f0 := $f1 + $f2
• sub.s $f0, $f1, $f2	$f0 := $f1 - $f2
• mul.s $f0, $f1, $f2	$f0 := $f1 * $f2
• div.s $f0, $f1, $f2	$f0 := $f1 / $f2
• abs.s $f0, $f1	$f0 := \|$f1|
• neg.s $f0, $f1	$f0 := -$f1

Data Movement Instructions

- Memory Transfer Instructions
  - l.s $f0, 100($t2) load word into $f0 from address $t2+100
  - s.s $f0, 100($t2) store word from $f0 into address $t2+100
- Data Movement between registers
  - mov.s $f0, $f2 move between FP registers
  - mfc1 $t1, $f2 move from FP registers (no conversion)
  - mtc1 $t1, $f2 move to FP registers (no conversion)
- Data conversion
  - cvt.w.s $f2, $f4 convert from single precision FP to integer
  - cvt.s.w $f2, $f4 convert from integer to single precision FP

Conditional Jumps

- Conditional jumps are performed in two stages
  1. Comparison of FP values sets a code in a special register
  2. Branch instructions jump depending on the value of the code
- Comparison
  - c.eq.s $f2, $f4 if $f2 == $f4 then code = 1 else code = 0
  - c.le.s $f2, $f4 if $f2 <= $f4 then code = 1 else code = 0
  - c.lt.s $f2, $f4 if $f2 < $f4 then code = 1 else code = 0
- Branches
  - bc1f label if code == 0 then jump to label
  - bc1t label if code == 1 then jump to label