MIPS mul div, and MIPS floating point instructions

Multiply and Division Instructions

- `mul rd, rs, rt`
  - puts the result of `rs times rt` in `rd`
- `div rd, rs, rt`
  - A pseudo instruction
  - puts the quotient of `rs/rt` into `rd`

`hi` and `lo`

- Special 'addressable' registers
  - you can not use these directly, you have to use special move instructions

- `mult rs,rt`
  - put the high word in `hi` and low word in `lo`.
- `div rs, rt`
  - put the remainder in `hi` and quotient in `lo`.

- `mfhi rd`
  - copies the value from `hi` and stores it in `rd`
- `mflo rd`
  - copies the value from `lo` and stores it in `rd`

Floating Point

- Floating point registers and the instructions that operate on them are on a separate chip referred to as coprocessor 1
- As a result floating point instructions typically can not use regular registers directly, you need to move the values into floating point registers
- Uses special registers $f0-$f31 (32 just like the number of main registers)
- Each register can store a single precision floating point number
- Pairs of registers can store a double precision floating point number
  - Ex. Storing a double precision number into $f0 uses both $f0 and $f1
  - Because of this, can only store to even numbered f registers
- When using functions, treat f registers like s registers
  - That is store them onto the stack if you need to use them within a function

Load and Store (single precision)

- Load or store from a memory location. Just load the 32 bits into the register.
  - `lwc1 $f0, 0($t0)`
    - `l.s $f0, 0($t0)`
  - `swc1 $f0, 0($t0)`
    - `s.s $f0, 0($t0)`

Load and Store (double precision)

- Load or store from a memory location. Just load the 64 bits into the register.
  - `ldc1 $f0, 0($t0)`
    - `l.d $f0, 0($t0)`
  - `sdc1 $f0, 0($t0)`
    - `s.d $f0, 0($t0)`

Load and Store (immediate)

- Load immediate number (pseudoinstruction)
  - `li.s  $f0, 0.5`
  - `li.d  $f0, 0.5`

Print and Read (single precision)

- Print:
  - `li $v0, 2`
  - `li.s $f12, 0.5`
  - `syscall`
- Read
  - `li $v0, 6`
  - `syscall`
  - (the read will be in `$f0`)

Print and Read (double precision)

- Print:
  - `li $v0, 3`
  - `li.d $f12, 0.5`
  - `syscall`
- Read
  - `li $v0, 7`
  - `syscall`
  - (the read will be in `$f0`)

Arithmetic Instructions

- Single Precision
- abs.s $f0, $f1
- add.s $f0, $f1, $f2
- sub.s $f0, $f1, $f2
- mul.s $f0, $f1, $f2
- div.s $f0, $f1, $f2
- neg.s $f0, $f1

- Double Precision
- abs.d $f0, $f2
- add.d $f0, $f2, $f4
- sub.d $f0, $f2, $f4
- mul.d $f0, $f2, $f4
- div.d $f0, $f2, $f4
- neg.d $f0, $f2

Data move

- `mov.s $f0, $f1`

  copy `$f1` to `$f0`.
- `mov.d $f0, $f2`

  copy `$f2` to `$f0`.

- `mfc1 $t0, $f0`

  copy `$f0` to `$t0`. Note the ordering
- `mtc1 $t0, $f0`

  copy `$t0` to `$f0`. Note the ordering

Convert to integer and from integer

- `cvt.s.w $f0, $f1`
  - convert the 32 bits in `$f1` currently representing an integer to float of the same value and store in `$f0`
- `cvt.w.s $f0, $f1`
  - the reverse

- `cvt.d.w $f0, $f2`
  - convert the 64 bits in `$f2` currently representing an integer to float of the same value and store in `$f0`
- `cvt.w.d $f0, $f2`
  - the reverse

Branch instructions

- `bc1t L1`
  - branch to L1 if the flag is set
- `bc1f L1`
  - branch to L1 if the flag is not set

Comparison instructions (single precision)

- `c.lt.s $f0,$f1`
  - set a flag in coprocessor 1if $f0 < $f1, else clear it. The flag will stay until set or cleared next time
- `c.le.s $f0,$f1`
  - set flag if $f0 <= $f1, else clear it
- `c.gt.s $f0,$f1`
  - set flag if $f0 > $f1, else clear it
- `c.ge.s $f0,$f1`
  - set flag if $f0 >= $f1, else clear it
- `c.eq.s $f0,$f1`
  - set flag if $f0 == $f1, else clear it
- `c.ne.s $f0,$f1`
  - set flag if $f0 != $f1, else clear it

Comparison instructions (double precision)

- `c.lt.d $f0,$f2`
  - set a flag in coprocessor 1if $f0 < $f2, else clear it. The flag will stay until set or cleared next time
- `c.le.d $f0,$f2`
  - set flag if $f0 <= $f2, else clear it
- `c.gt.d $f0,$f2`
  - set flag if $f0 > $f2, else clear it
- `c.ge.d $f0,$f2`
  - set flag if $f0 >= $f2, else clear it
- `c.eq.d $f0,$f2`
  - set flag if $f0 == $f2, else clear it
- `c.ne.d $f0,$f2`
  - set flag if $f0 != $f2, else clear it

Computing the square root of a number n

- The Newton's method

  $x' = (x + n/x)/2$

  - For any n, guess an initial value of x as the sqrt of n and keep on updating x until is the difference between the two updates are very close.
  - The idea is that $x' = x - f(x)/f'(x)$, where $f(x)$ is $x^2 - n = 0$.

```
            .data
#val1:      .float 0.6

            .text
            .globl main
main:
            li.s $f0, 361.0
            mfc1 $a0, $f0
            jal calsqrt

done:
            mtc1 $v0, $f12
            li $v0,2
            syscall

exit:
            li $v0,10
            syscall

            # calsqrt:
            # calculating the square root of n
            # using the formula x'=(x+n/x)/2
            # loop until |x'-x| < 0.001

calsqrt:
            addi $sp, $sp, -24  # store f
            swc1 $f0, 20($sp)   # registers
            swc1 $f1, 16($sp)   # onto stack
            swc1 $f2, 12($sp)
            swc1 $f3, 8($sp)
            swc1 $f20, 4($sp)
            swc1 $f21, 0($sp)

            mtc1 $a0, $f0        # $f0 gets n
            li.s $f20, 2.0       # $f20 = 2 for dividing
            li.s $f21, 0.001     # $f21 = 0.001 for comparison
            div.s $f1, $f0, $f20 # $f1 = x = n/2

calsqrtloop:
            div.s $f2, $f0, $f1  # $f2 = n/x
            add.s $f2, $f2, $f1  # $f2 = n/x + x
            div.s $f2, $f2, $f20 # $f2 = x' = (n/x + x)/2
            sub.s $f3, $f2, $f1  # $f3 = x'-x
            abs.s $f3, $f3       # $f3 = |x'-x|
            c.lt.s $f3, $f21     # set flag if |x'-x| < 0.001
            bc1t calsqrtdone
            mov.s $f1, $f2
            j calsqrtloop

calsqrtdone:
            mfc1 $v0, $f2

            lwc1 $f0, 20($sp)    # restore f
            lwc1 $f1, 16($sp)    # registers
            lwc1 $f2, 12($sp)    # from stack
            lwc1 $f3, 8($sp)
            lwc1 $f20, 4($sp)
            lwc1 $f21, 0($sp)
            addi $sp, $sp, 24

            jr $ra
```