General SPARC Information

Registers (general purpose, integer registers):

- Global (%g1 - %g7)
  - Consistent throughout functions call (ie, no sliding window action)
  - %g0 is always the value zero; Read-Only -- Result thrown away when used as destination register
  - %g1 - %g4 are volatile, so use only for temporary values
- Local (%l0 - %l7)
  - Values local to each function (disappear after function returns - RESTORE instruction)
- In(%i0 - %i5)
  - Input parameters to a function; Args passed to this function are accessible via %i0-%i5
  - %i0 is where you should store your return value before you return
  - %i6 (%fp) and %i7 are reserved, so don't mess with them!
- Out(%o0 - %o5)
  - Output arguments to a function (Put args you are passing to a function in %o0-%o5 right before you do the "call")
  - Once a function returns, the return value is waiting in %o0
  - %o6 (%sp) and %o7 are reserved, so don't mess with them!

Common Instructions:

- Set (no +/- 4K restriction)
  - set 12345, %l0 ! %l0 = 12345
- Move (constants between +/- 4K OK)
  - mov -145, %l0 ! %l0 = -145
  - mov %l2, %i5 ! %l0 = 12345
- Simple Arithmetic (add/addcc, sub/subcc)
  - add %o0, %o1, %o2 ! %o2 = %o0 + %o1
- Increment/Decrement (inc/inccc, dec/deccc)
  - inc %l4 ! %l4 = %l4 + 1
- Shifting (sll, srl, sra)
  - sll %o1, 5, %o0 ! %o0 = %o1 << 5
- Load
  - ld [%fp - 4], %i4 ! %i4 = *(%fp - 4)
- Store
  - st %i3, [%fp - 8] ! *(%fp - 8) = %i3
- Compare
  - cmp %o0, %o1 ! Sets condition codes based on %o0 - %o1
- Branch (bg, bge, bl, ble, be, bne, ba, bn) **(NOTE: Requires nop after it)**
  - ble loop2 ! Go to label "loop2" *IF* prior cmp was <=
  - nop
- Call (Args passed to function go in %o0-%o5 before the call) **(NOTE: Requires nop after it)**
  - call foo ! Jump to label "foo"
  - nop
- Multiplication/Division/Modulus Arithmetic (call .mul, .div, .rem)
  - mov %l0, %o0 ! x = x * 5678;
  - set 5678, %o1
  - call .mul
  - nop
  -
  - mov %o0, %l0
- Negating/2's Complement
  - neg %o0, %o0 ! %o0 = -%o0
- Clear register (set reg. contents to 0)
  - clr %l0 ! %l0 = 0
- Bitwise Ops (and/andcc, or/orcc, xor/xorcc)
  - and %l0, %l3, %l0 ! %l0 = %l0 & %l3 (bitmask)

Useful SPARC Floating-Point Information

Floating-point registers:

- %f0 through %f31
- Not windowed (i.e., they do not slide on SAVE/RESTORE operations)

Floating-point I/O:

- To output a single-precision FP number, put it in %f0 and call printFloat() [defined in output.s]
- To input a single-precision FP number, call inputFloat() [defined in input.c] and get result from %f0

Inserting FP constants into your assembly:

- .align 4
- x: .single 0r459.25 [, 0r99.50] (can include a sequence with commas)
- To access these constants, set the label to some register (for example, %l2) and do a ld into an FP register.
- set x, %l2
- ld [%l2], %f0

Promoting an integer to a single-precision floating-point:

- When you want to promote an integer (that is already in an integer registers) into a FP value, you should store it in some temporary memory (*st %l2, [%fp-4]*), reload it into an FP register (*ld [%fp-4], %f2*), and finally convert the underlying bit pattern to single-precision (*fitos %f2, %f2*).

Single-precision operations:

- fadds %fx, %fy, %fz Add register fx to fy, place result in fz
- fsubs %fx, %fy, %fz Subtract register fy from fx, place result in fz
- fmuls %fx, %fy, %fz Multiply register fx by fy, place result in fz
- fdivs %fx, %fy, %fz Divide register fx by fy, place result in fz
- fsqrts %fx, %fy Compute square root of fx
- fcmps %fx, %fy Compare register fx to fy, set fcc bits
- fitos %fx, %fy Convert integer to single precision
- fstoi %fx, %fy Convert single precision to integer

Special purpose operations:

- fmovs %fx, %fy Copy contents of fx to fy
- fnegs %fx, %fy Complement upper bit of fx and copy to fy
- fabss %fx, %fy Clear upper bit of fx and copy to fy
- fstod %fx, %fy Convert single to double precision **(NOTE: you should not need to use this instruction)**
- fdtos %fx, %fy Convert double to single precision **(NOTE: you should not need to use this instruction)**

Floating-point memory:

- ld [adr], %fx Load single-precision value from memory
- st %fx, [adr] Store single-precision value into memory

Floating-point Branching:

- For conditional branches, just add an "f" before the equivalent integer branch instructions list above
  - fbe, fbne, fbg, fbge, fbl, fble, etc.
- **NOTE: You need a nop *or* a simple integer instruction between *fcmps* and the FP branch instruction!!!**
  - fcmps %f0, %f1
  - nop
  -

    - fbg .L3
    - nop
- Also, note that when you compare floating point values, since we are only displaying 2 decimal places, don't get confused with something like 16.75 != 16.75 (almost impossible to do equality for floating-points)
- FP values are imprecise and cannot be exactly represented except for a few values. So, it is typically good programming practice to avoid writing code that uses explicit equality checks (instead, one should do things like >= or <=, since direct == will typically result in false due to imprecision).

Example SPARC Floating-Point Program

```
        .section        ".data"
        .align          4
NL:     .asciz          "\n"
y:      .asciz          "YES\n"
n:      .asciz          "NO\n"
        .align          4
fpc:    .single         0r420.25, 0r-23.75      ! Two compile-time constants

        .section        ".text"
        .align          4
        .global         foo                     ! foo ( FLOAT )
foo:
        save            %sp, -(92 + 4) & -8, %sp

        st              %i0, [%fp-4]
        ld              [%fp-4], %f0            ! Put FLOAT param into fp reg
        ! Alternatively, you could pass FLOATs directly in the fp regs

        call            printFloat
        nop

        ret
        restore


        .section        ".text"
        .align          4
        .global         main
main:
        save            %sp, -(92 + 4) & -8, %sp

        set             fpc, %l0
        ld              [%l0], %f0
        fadds           %f0, %f0, %f0

!       .section        ".data"                 ! Same as below, but with floating const
!       .align          4                       ! Shows you can insert constants were needed
!tmp1:  .single         0r3
!       .section        ".text"
!       .align          4
!       set             tmp1, %l0
!       ld              [%l0], %f1

        mov             3, %l0                  ! Using integer
        st              %l0, [%fp-4]
        ld              [%fp-4], %f1
        fitos           %f1, %f1                ! Convert bit pattern from int to float
        fdivs           %f0, %f1, %f0
        fsqrts          %f0, %f0

        call            printFloat              ! Result already in %f0
        nop

        set             NL, %o0
        call            printf
        nop
```

```
        call            inputFloat
        nop                                     ! Result in %f0

        set             fpc, %l0
        ld              [%l0+4], %f1            ! get 2nd constant (hence +4)

        fcmps           %f0, %f1                ! Compare input w/ 2nd const
        nop                                     ! Needed in between FP compare and FP branch!!!
        fbg             L1
        nop

        set             y, %o0
        call            printf
        nop

        ba              skip
        nop

L1:
        set             n, %o0
        call            printf
        nop
skip:
        call            inputFloat
        nop                                     ! Result in %f0

        st              %f0, [%fp-4]
        ld              [%fp-4], %o0            ! Send using sliding int reg
        ! Alternatively can send directly with non-sliding fp reg

        call            foo
        nop

        ret
        restore
```