Arithmetic Instructions

- General form

$$\begin{bmatrix}\texttt{add}\\ \texttt{sub}\end{bmatrix}\{\texttt{x}\}\ \{\texttt{cc}\}\qquad src_1, src_2, reg$$

use carry bit from previous condition code (more later)

set condition code

- src_1 and reg must be registers; src_2 may be a register or a signed 13-bit number

```
add %o1,%o2,%g3
sub %i1,2,%g3
```

- Some SPARCs have ***no*** multiply or divide instructions

  see Appendix E of the SPARC Architecture Manual, §4.10 in Paul

- Standard run-time library provides multiply and divide routines

  `.mul .rem .div` signed arithmetic

  `.umul .urem .udiv` unsigned arithmetic

Data Movement

- Load a constant into a register

  set *value*, *reg* **sethi** **%hi(***value***)**, *reg*
  or *reg*, **%lo(***value***)**, *reg*

- if **%hi(***value***) == 0**, omit **sethi**, if **%lo(***value***) == 0**, omit **or**
- **sethi** instruction

0	*reg*	4	*imm22*
31 29	28	24	21

e.g., direct addressing

```
set a,%g1             sethi %hi(a),%g1
ld [%g1],%g2          or %lo(a),%g1
                      ld [%g1],%g2
```

faster alternative (2 instead of 3 ticks):

```
sethi %hi(a),%g1
ld [%g1+%lo(a)],%g2
```

- Clearing registers and memory; note the use of **%g0** to stand for 0

```
add %g0,%g0,%o1
st %g0,[%i1]
stb %g0,[%i1]
```


Synthetic Instructions

- ***Synthetic instructions*** or ***pseudo-instructions*** are implemented by the ***assembler*** by one or more "real" instructions

move register to register

mov *src*, *dst*	**or %g0,** *src*, *dst*

clear register, memory

clr *reg*	**add %g0,%g0,** *reg*
clr [*address***]**	**st %g0,[***address***]**

negate

neg *dst*	**sub %g0,** *dst*, *dst*
neg *src*, *dst*	**sub %g0,** *src*, *dst*

increment/decrement

inc *dst*	**add** *dst*, **1**, *dst*
dec *dst*	**sub** *dst*, **1**, *dst*

- See page 85 in the SPARC Architecture Manual and Appendix C in Paul

Bitwise Logical Instructions

- General form

$$\begin{bmatrix}\texttt{and}\\ \texttt{andn}\\ \texttt{or}\\ \texttt{orn}\\ \texttt{xor}\\ \texttt{xnor}\end{bmatrix}\{\texttt{cc}\}\qquad src_1, src_2, dst$$

- Corresponding C bitwise operators; src_2 is a register or a signed 13-bit immediate (number)

and	$dst = src_1 \ \& \ src_2$
andn	$dst = src_1 \ \& \ \sim src_2$
or	$dst = src_1 \ \vert \ src_2$
orn	$dst = src_1 \ \vert \ \sim src_2$
xor	$dst = src_1 \ \hat{} \ src_2$
xnor	$dst = src_1 \ \hat{} \ \sim src_2$

Bitwise Logical Instructions, cont'd

- Complement

2's complement	**neg** *reg*	**sub %g0,***reg***,***reg*
1's complement	**not** *reg*	**xnor** *reg***,%g0,***reg*

- Synthetic instructions

btst *bits***,***reg*	**andcc** *reg***,***bits***,%g0**
bset *bits***,***reg*	**or** *reg***,***bits***,***reg*
bclr *bits***,***reg*	**andn** *reg***,***bits***,***reg*
btog *bits***,***reg*	**xor** *reg***,***bits***,***reg*

e.g.,

btst 0x8,%g1

Shift Instructions

- General form

$$\mathtt{s}\begin{bmatrix}\mathtt{l}\\ \mathtt{r}\end{bmatrix}\begin{bmatrix}\mathtt{l}\\ \mathtt{a}\end{bmatrix}\quad src, \begin{bmatrix} reg_1 \\ \mathtt{0..31}\end{bmatrix}, reg$$

left or right shift

arithmetic or logical shift (no `sla`)

shift amount

- Instruction format

10	*reg*	**sll**=*100101* **srl**=*100110* **sra**=*100111*	*src*	0	00000000	*reg*$_1$
10	*reg*	"	*src*	1	00000000	0..31
31	29	24	18	13	12	4

- Vacated bits: **sll** or **srl** fill with 0s, **sra** fills with sign bit
- For 2's complement numbers

  sra *reg***,***n***,***reg* divides *reg* by 2^n

  sla *reg***,***n***,***reg* multiplies *reg* by 2^n

  shift instructions do ***not*** modify the condition codes

Floating Point Instructions

- Floating point instructions are performed using the floating point unit (FPU);
- 32 floating point registers: **%f0** — **%f31**
- Floating point load and store instructions:

  ld [*address***],***freg*
  ldd [*address***],***freg*

  st *freg***,[***address***]**
  std *freg***,[***address***]**

  doubles use even-odd register pair

- Other instructions; ***src***, ***src1***, ***src2***, ***dst*** denote floating point registers

fmovs	*src***,***dst*	move; double/quad takes 2/4 **fmovs**
fnegs	*src***,***dst*	negate; double/quad takes 1/3 **fmovs**
fabss	*src***,***dst*	absolute value; double/quad takes 1/3 **fmovs**
fsqrt[sdq]	*src***,***dst*	square root
fadd[sdq]	*src1***,***src2***,***dst*	addition
fsub[sdq]	*src1***,***src2***,***dst*	subtraction
fmul[sdq]	*src1***,***src2***,***dst*	multiplication
fdiv[sdq]	*src1***,***src2***,***dst*	division

Floating Point Instructions, cont'd

- Comparison and branching

fcmp[sdq]	*src1***,***src2*	floating point compare

- 4 floating point condition codes

  E equal
  L less than
  G greater than
  U unordered

- Use these condition codes with floating point conditional branches

  see page 38 in SPARC Architecture Manual, §11.5 in Paul

- Floating point conversions

f[sdq]toi	*src1***,***src2*	convert single/double/quad to signed integer
fito[sdq]	*src1***,***src2*	convert integer to single/double/quad

fito*x* rounds to "even", **f*x*toi** rounds toward 0; register holds an integer

f[sdq]to[sdq]	*src1***,***src2*	convert between floating point formats