SPARC Instruction Set

CS 217

Load Instructions

- Move data from memory to a register

```
ld[u|s][b|h|d]{a}   [address], reg
```

- Details
  - fetched byte/halfword is right-justified
  - leftmost bits are zero-filled or sign-extended
  - double-word loaded into register pair; most significant word in *reg* (must be even); least significant in *reg+1*
  - address must be appropriately aligned

Store Instructions

- Move data from a register to memory

```
st[b|h|d]{a}   reg, [address]
```

- Details
  - rightmost bits of byte/halfword are stored
  - leftmost bits of byte/halfword are ignored
  - *reg* must be even when storing double words

Arithmetic Instructions

General form

`add{x}{cc}` *src1, rc2, reg*

`sub{x}{cc}` *src1, src2, reg*

Details

src1 and *reg* must be registers

src2 may be a register or a signed 13-bit immediate

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

Libraries often provide multiply and divide

`.mul .div .rem ...`

Data Movement

- Load a constant into a register

`set` *value, reg*

implemented as

`sethi %hi(`*value***`), `***reg*

`or` *reg*`, %lo(`*value*`), `*reg*

if **`%hi(`***value***`) == 0,`** omit **`sethi`**

if **`%lo(`***value***`) == 0,`** omit **`or`**

Data Movement (cont)

- Example: direct addressing

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

faster alternative

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

Data Movement (cont)

- Clearing registers and memory

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


Synthetic Instructions

- Implemented by assembler with one or more "real" instructions; also called pseudo-instructions

Synthetic	Real
mov *src,dst*	**or %g0,** *src,dst*
clr *reg*	**add %g0,%g0,** *reg*
clr [*addr***]**	**st %g0,[***addr***]**
neg *dst*	**sub %g0,** *dst,dst*
neg *src,dst*	**sub %g0,** src,dst
inc *dst*	**add** *dst*,**1,** *dst*
dec *dst*	**sub** *dst*,**1,** *dst*

Bitwise Logical Instructions

Assembly	Corresponding C
and{cc} *src1,src2,dst*	*dst* = *src1* **&** *src2*
andn{cc} *src1,src2,dst*	*dst* = *src1* **&** **~***src2*
or{cc} *src1,src2,dst*	*dst* = *src1* **|** *src2*
orn{cc} *src1,src2,dst*	*dst* = *src1* **|** **~***src2*
xor{cc} *src1,src2,dst*	*dst* = *src1* **^** *src2*
xnor{cc} *src1,src2,dst*	*dst* = *src1* **^** **~***src2*

Bitwise Logical (cont)

- Complement

  neg *reg* — **sub %g0,***reg,reg* (2's comp)
  not *reg* — **xnor** *reg*,**%g0,***reg* (1's comp)

- 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*

- Example

  btst 0x8,%g1

Shift Instructions

- General form

  s[**l** | **r**][**l** | **a**] *src*, [*reg1* | **0..31**], *reg* (note: no **sla**)

- **sll** and **srl** fill with 0; **sra** fills with sign bit
- For 2's complement numbers

  sra *reg,n,reg* divides *reg* by 2^n
  sll *reg,n,reg* multiplies *reg* by 2^n
  shift instructions do not modify the condition codes

Floating Point Instructions

- Performed by floating point unit (FPU)
- Use 32 floating point registers: **%f0…%f31**
- Load and store instructions

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

- Other instructions are FPU-specific

  fmovs,fsqrt,fadd,fsub,fmul,fdiv,…