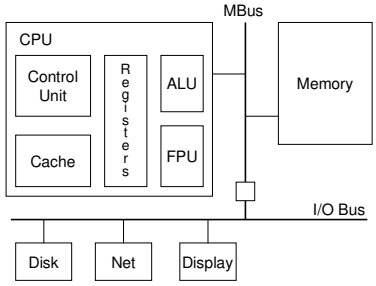
Machine Architecture

CS 217

Computer Organization



Memory Hierarchy

- Registers
  - ~128, 1-5ns access time (CPU cycle time)
- Cache
  - 1KB – 4MB, 20-100ns (multiple levels)
- Memory
  - 64MB – 1GB, 200ns
- Disk
  - 1GB – 20GB, 10ms
- Long-term Storage
  - 1TB, 1-10s

Source to Binary

- Source code in some high-level language

```
x = a + b;
```

- Assembly language

```
ld a, %r1
ld b, %r2
add %r1, %r2, %r3
st %r3, x
```

- Machine language

  32-bit instructions (bit patterns) executed by the machine

Instruction Formats

- Each machine instruction is composed of…

  opcode: operation to be performed

  operand(s): data that is operated upon

- Each machine supports a few formats…

  opcode

  opcode dst

  opcode src dst

  opcode src1 src2 dst

Instruction Execution

- CPU's control unit executes a loop

  fetch: fetch at PC; increment PC

  decode: interpret instruction format

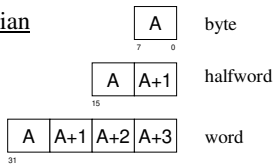
  operand fetch: load operands into registers

  execute: perform instruction opcode

  store: write results to memory

Addressing Memory

- 8-bit byte is the smallest addressable unit
- 32-bit addresses; thus 32-bit address space
- Doubleword too
- Sparc is <u>big-endian</u>



Sparc Registers

- 32 x 32-bit general-purpose registers
  `%r0 … %r31`
- Register map

`%g0 … %g7`	`%r0 … %r7`	global
`%o0 … %o7`	`%r8 … %r15`	output
`%l0 … %l7`	`%r16 … %r23`	local
`%i0 … %i7`	`%r24 … %r31`	input

- Some registers have dedicated uses

`%sp (%r14, %o6)`	stack pointer
`%fp (%r30, %i6)`	frame pointer
`%r15`	temporary
`%r31`	return address
`%g0 (%r0)`	always `0`

Sparc Registers (cont)

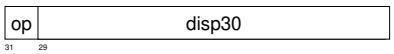
- Special-purpose registers
  - manipulated by special instructions
  - floating point registers `(%f0 … %f31)`
  - program counter `(PC)`
  - next program counter `(nPC)`
  - `PSR, TBR, WIM, Y`

Sparc Instruction Set

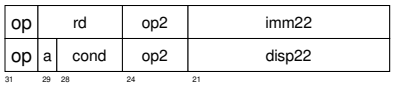
- Instruction groups
  - load/store instructions
  - integer arithmetic and bit-wise logical instructions
  - control transfer instructions
  - special instructions (used by OS)
  - floating point arithmetic

Instruction Set (cont)

- Format 1 (op = 1): **`call`**

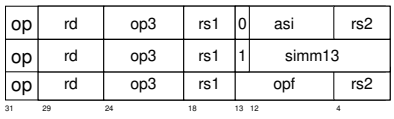


- Format 2 (op = 0): **`sethi`** and branches



Instruction Set (cont)

- Format 3 (op = 2 or 3): remaining instructions



Assembly vs Machine Language

- Machine language is the bit patterns that represent instructions
- Assembly language is a symbolic representation of machine language
- Assemblers translate from assembly to machine language
  - mapping is 1-to-1
- Compilers map from source to assembly
  - mapping is 1-to-many

Assembly vs Machine (cont)

- Example

```
add %i1,360,%o2
```

2	10	0	25	1	360	(decimal)
2	12	0	31	1	550	(octal)

31 29 24 18 13 12

```
10010100000001100110000101101000
```


Addressing Modes

- Two modes to yield effective address
  - add contents of two registers
    - `ld [%o1],%o2` register indirect
    - `st %o1,[%o2,%o3]` register indexed
  - add contents of register and immediate
    - `ld [%o1+10],%o2` base displacement

Addressing Modes (cont)

- Assembly language syntax

Address	Synonym
reg	reg + %g0
reg + reg	
reg + N	
N + reg	reg + *N*
N	%g0 + *N*

where *N* is a 13-bit, signed, integer constant