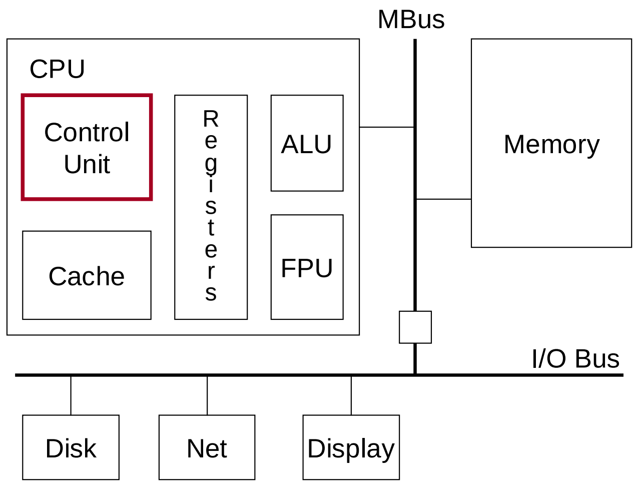
Sparc Architecture

CS 217

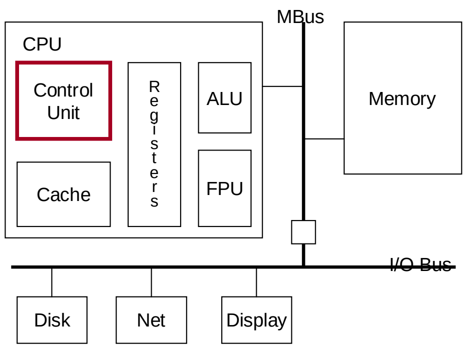
Machine Architecture



Instruction Execution

- CPU's control unit executes a program

```
PC ß memory location of first instruction
while (PC != last_instr_addr)
    execute(MEM[PC]);
```



Instruction Execution

- CPU's control unit executes a program

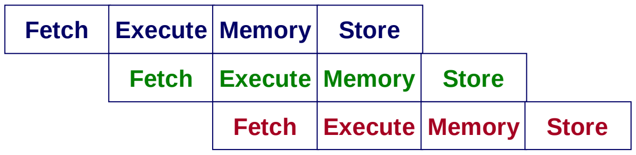
```
PC ß memory location of first instruction
while (PC != last_instr_addr)
    execute(MEM[PC]);
```

- Multiple phases…
  - fetch: instruction fetch; increment PC
  - execute: arithmetic instructions, compute branch target address, compute memory addresses
  - memory access: read/write memory
  - store: write results to registers

Fetch	Execute	Memory	Store	Fetch	Execute	Memory	Store

Instruction Pipelining

- Pipeline



- PC is incremented by 4 at the Fetch stage to retrieve the next instruction

Instructions

- High-level language

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

- Assembly language

```
ld a, %r8
ld b, %r9
add %r8, %r9, %r10
st %r10, x
```

Symbolic Representation

- Machine language

```
10010100000000010000000000001001
```

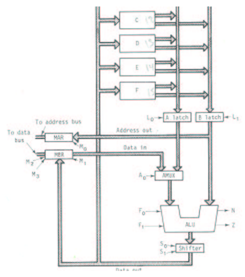
Bit-encoded Representation

Machine Language

- Each 32-bit word is a code for one instruction
- Example:

```
10  01010  000000  01000  0  00000000  01001
```

01010	000000	01000	01001
Destination Register %r10	Add	Source Register %r8	Source Register %r9



Assembly Language

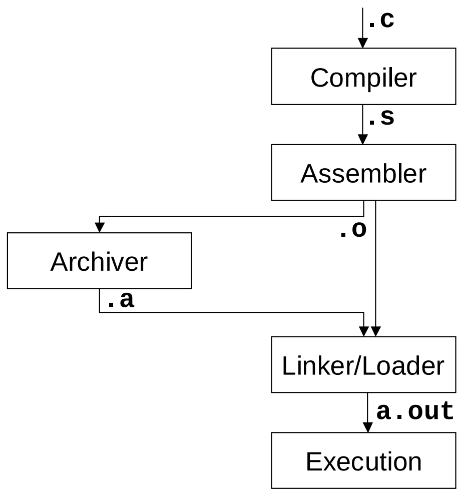
- Symbolic representation of machine language
  - Most assembly language instructions map directly to machine language counterparts
  - Also have symbolic names, macros, address labels, etc.

```
    #define i %l0
    #define n %l1
    clr i
L1: cmp i,n
    bge L2; nop
    . . .
    inc i
    ba L1; nop
L2:
```

Compilation Pipeline

- Compiler (`gcc`): `.c` à `.s`
  - translates high-level language to assembly language
- Assembler (`as`): `.s` à `.o`
  - translates assembly language to machine language
- Archiver (`ar`): `.o` à `.a`
  - collects object files into a single library
- Linker (`ld`): `.o` + `.a` à `a.out`
  - builds an executable file from a collection of object files
- Execution (`execlp`)
  - loads an executable file into memory and starts it

Compilation Pipeline



Sparc Instructions

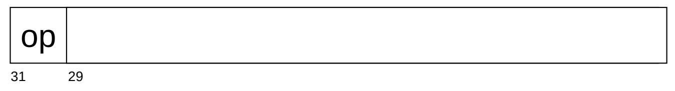
- Each machine instruction is composed of…
  - opcode: operation to be performed
  - operand: data that is operated upon
- Each machine supports a few formats…
  - *opcode*
  - *opcode dst*
  - *opcode src dst*
  - *opcode src1 src2 dst*

Sparc Instruction Set

- Instruction groups
  - integer arithmetic (**`add`**, **`sub`**, ...)
  - bit-wise logical (**`and`**, **`or`**, **`xor`**, ...)
  - bit-wise shift (**`sll`**, **`srl`**, ...)
  - load/store (**`ld`**, **`st`**, ...)
  - integer branch (be, bne, **`bl`**, **`bg`**, ...)
  - Trap (**`ta`**, **`te`**, ...)
  - control transfer (**`call`**, **`save`**, ...)
  - floating point (**`ldf`**, **`stf`**, **`fadds`**, **`fsubs`**, ...)
  - floating point branch (**`f`**be, **`fb`**ne, **`fbl`**, **`fbg`**, ...)

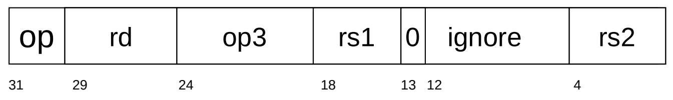
Sparc Instruction Set

- Instruction formats
  - Format 1 (op = 1) -- e.g., call
  - Format 2 (op = 0): -- e.g., branches
  - Format 3 (op = 2 or 3): -- e.g., add



Sparc Instruction Set

- Format 3 (op = 2 or 3):



```
add %i1,%i2,%o2
```

Sparc Instruction Set

- Format 3 (op = 2 or 3):

op	rd	op3	rs1	0	ignore	rs2

OR

op	rd	op3	rs1	1	simm13	
31	29	24	18	13	12	4

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

simm13 is a signed constant within +-4096

Example

- Assembly Language
  `add %i1,360,%o2`
- Machine language

2	10	0	25	1	360	(decimal)
2	12	0	31	1	550	(octal)
31	29	24	18	13	12	

```
10010100000001100110000101101000
```

Other Sparc Instructions

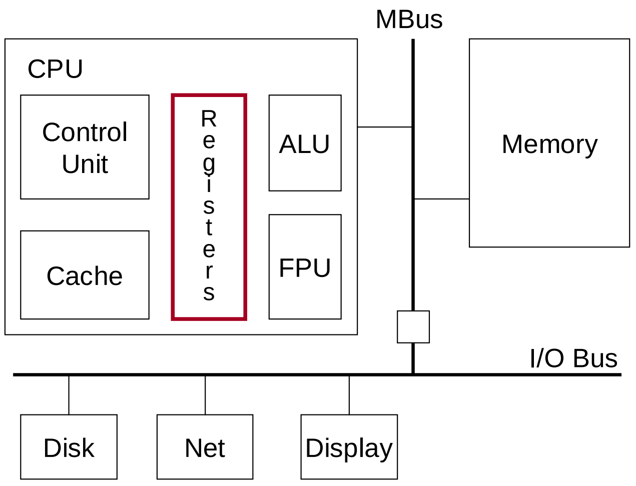
- Format 1 (op = 1) -- e.g., **`call`**

op	disp30
31	29

- Format 2 (op = 0) -- e.g., branches

op		rd	op2	imm22
op	a	cond	op2	disp22
31	29	28	24	21

Machine Architecture



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

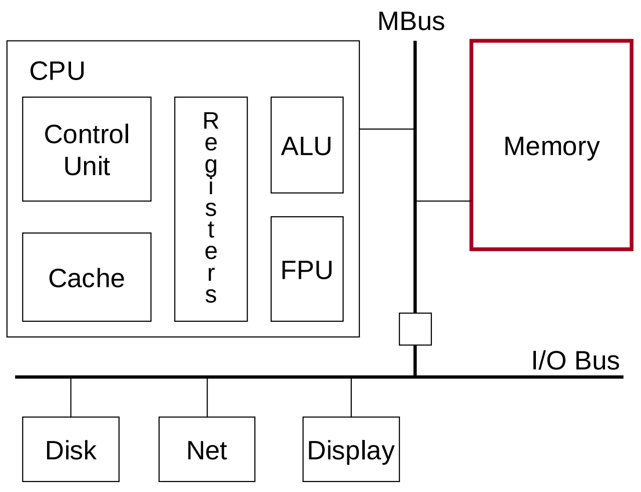
- Some registers have dedicated uses

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

Sparc Registers (cont)

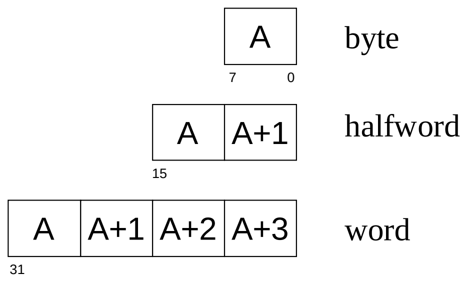
- Special-purpose registers
  - o manipulated by special instructions
- Examples
  - o floating point registers **`(%f0 … %f31)`**
  - o program counter **`(PC)`**
  - o next program counter **`(NPC)`**
  - o integer codition codes **`(PSR)`**
  - o trap base register **`(TBR)`**
  - o window **`(WIM)`**
  - o etc.

Machine Architecture



Addressing Memory

- 8-bit byte is the smallest addressable unit
- 32-bit addresses; thus 32-bit address space
- Can load and store doublewords too
- Sparc is big-endian



Addressing Memory

- 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
    ```

Upcoming Lectures ...

- Instruction set
- Number systems
- Branching condition codes
- Procedure calls
- Assembler
- Linker
- etc.