Chapter 1

Processors - September 11

1.1 Processor

PC program counter stores memory address of next instruction

IR instruction register stores instruction read from memory

MAR memory adderss to register outputs address to memory

MDR memory data register. Holds data/instructions from memory or going to memory

1.2 Instruction execution

1.2.1 Instruction Fetch(IF)

- Copy PC contents to MAR and assert R/W control signal
- Wait for response from memory and copy MDR contents to IR
- Increment PC

1.2.2 Instruction Decode(ID)

- Interpret bits in IR

1.2.3 Operand Fetch(OF)

- Read data from registers and/or extract constants from IR

1.2.4 Execute(EX)

- Use ALU or read memory(load) or write memory(store)

1.2.5 Writeback(WB)

Write result to a register

Eg Execute Load R2, LOC (memory address label)

1. Always same as above
2. Recognize "Load"
3. Etract LOC from IR
4. Copy LOC to MAR and assert R/W control signal
5. Copy MDR Contents to R2

1.2.6 Homework

```
ADD R4, R2, R3 ($R4 <- [R2] + [R3]$)
 Store R4, LOC
```

1.3 Design Paradigms

1.3.1 CISC

Complex Instruction Set Computer

- Machine instructions can perform complex operations

  E.g. (x86) *movsb* copies an array of bytes
- Instructions are variable length
- Operands come from registers or memory

  E.g *M68K* ADD DO, LOC (mem[LOC] <- [DO] + [mem[LOC])
- Complex addressing modes

  E.g. (M68K) ADD DO, (A0)+
- Smaller object code
- Direct support of High Level Language constructs
- Ease of assembly language programming
- Hardware is difficult to pipeline(speed up)

1.3.2 RISC

Reduced instruction-set computer

- Fewer, simpler instructions
- Load/store architecture
  - only load or store
  - ALU operands only come from registers

Eg (ARM)

```
ldr r1, LOC
add r1, r0, r1
ldr r2=LOC
str r1, [r2]
```

- Object code is larger (by %30)
- Hardwire easier to pipeline

1.4 Register Transfer Notation

(no standard)

- Expresses the semantics of instruction execution as data transfers and control flow(logic)
- Memory locations are assigned labels e.g. LOC, A
- Registers are named R0, R1, PC, IR

x denotes contents of x

E.g.

[LOC] contents of memory at LOC

[R0] contents of register R0

[[R0]] contents of memory at the location specified by contents of R0

',' denotes parallel
';' dnotes sequential

```
ADD R4, R2, R3
R4 <- [R2] + [R3]
```

E.g. instruction fetch

```
MAR <- [PC] , R/W <- 1 , PC <- [PC] + 4
IR <= [MOR]
```