3/29/20

## Instruction Set Architecture (ISA)

- Programmer's view of computer (functionality)
  instruction set, data types, memory model
  instruction format, addressing modes
- Binary compatibility : $ISA_{old} \subseteq ISA_{new}$
- Instruction set

| Data movement | computation | flow control |
|---|---|---|
| registers ↔ memory | arithmetic | branch (cond) |
| registers ↔ registers | logic | subroutine call |

(Ex

$R[s1] \leftarrow M[R[s\emptyset]]$ $\quad$ $R[s\emptyset] \leftarrow R[s\emptyset] + 4$

lw \$s1, 0(\$s∅) $\quad$ addi \$s∅, \$s∅, 4

if $(R[s0] == R[s3])$, PC ← SOME ADDRESS

beq \$s∅, \$s3, SOMEADDRESS

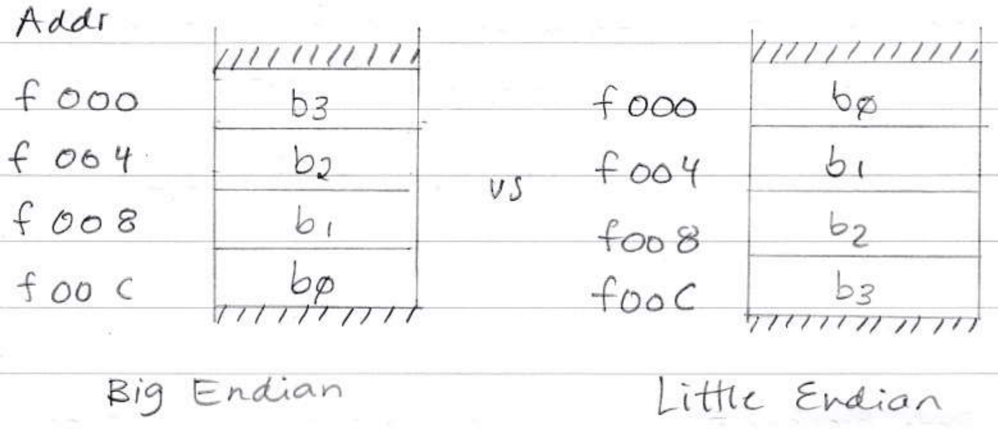
More examples in comming classes (above is MIPS)

- Data types

signed / unsigned integers : byte (1), halfword (2), word (4)

floating-point numbers : float (4), double (8)

* Memory model : word = $b_3\ b_2\ b_1\ b_0$ (bytes)
MSB LSB



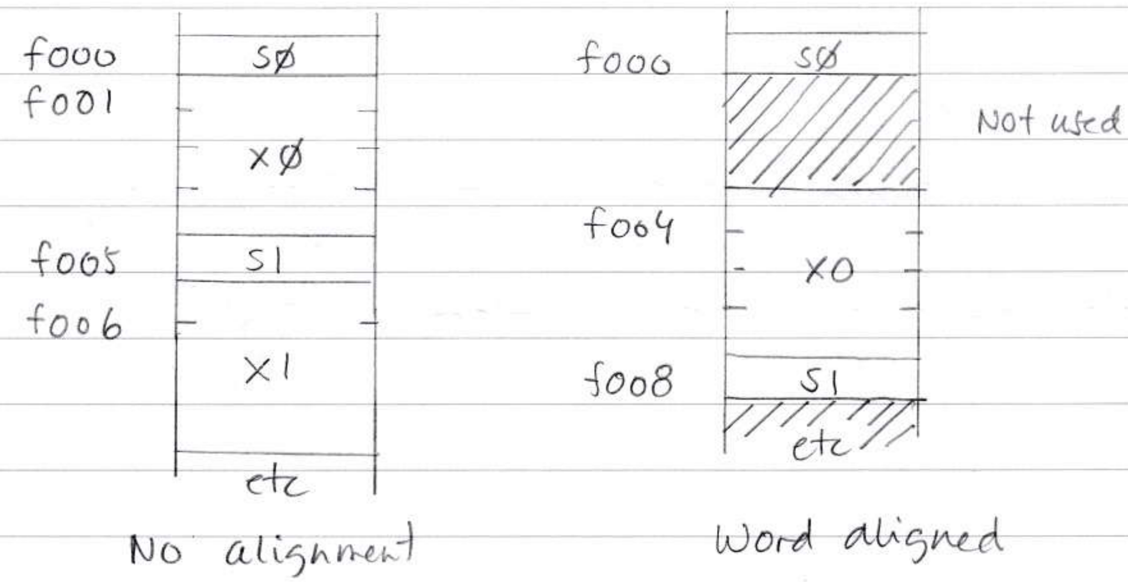
(Ex Mainframes, workstations (SUN SPARC) | Microprocessors (eg Intel IA-32)

Network protocols (eg TCP)

Most modern ISA support both : Bi - Endian
(eg ARM, MIPS)
Intel IA-64

* Word alignment (data at address divisible by its size)

struct { char s; float x; } A[N];



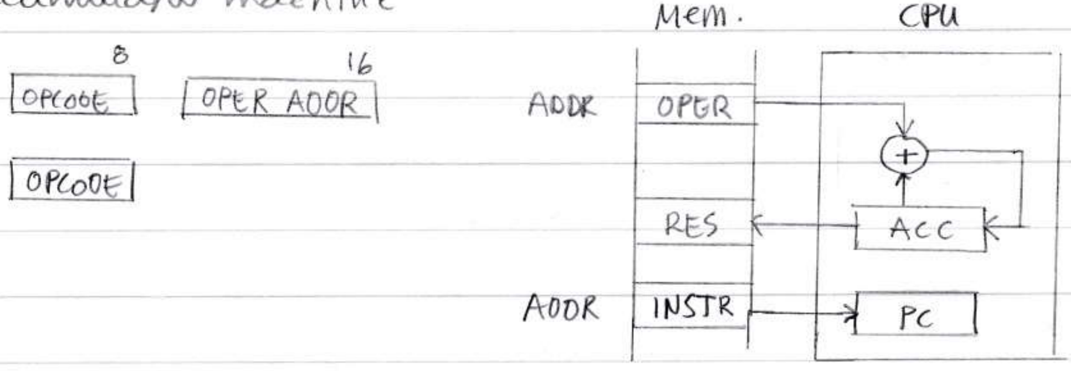
☆ Instruction format: opcode

Bit layout of executable instruction (more next class)

| Variable width | Fixed width |
|---|---|
| + Flexible | − Not very flexible |
| − Slow decode / execution | + fast decode / execution |
| (Ex Intel / AMD | ARM, MIPS, SPARC |

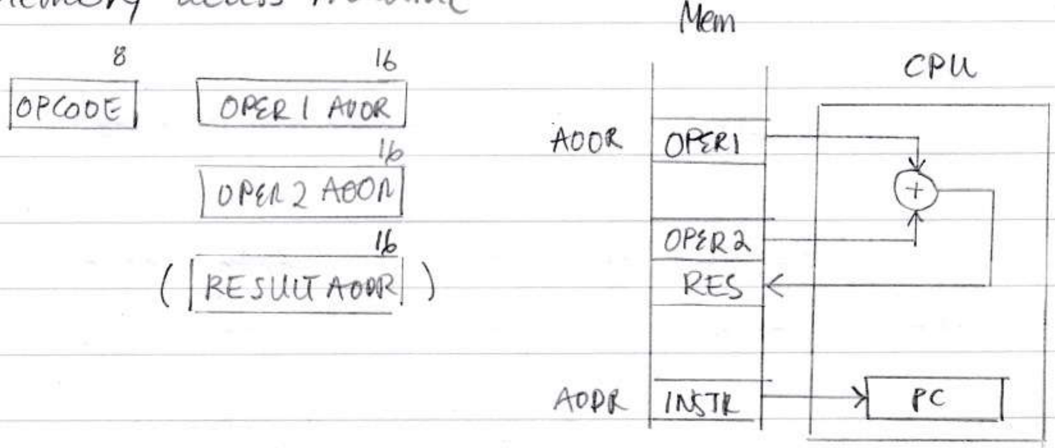
☆ Instruction format: operands

− Accumulator machine



(Ex "Early day microprocessors" eg Intel 8080 / MC6800

− Memory access machine



(Ex Minicomputers, mainframes, later Intel/AMD processors

CISC: Complex Instruction Set Computer

- General register machine

| OPCODE | Ra | OPER ADDR |
|---|---|---|

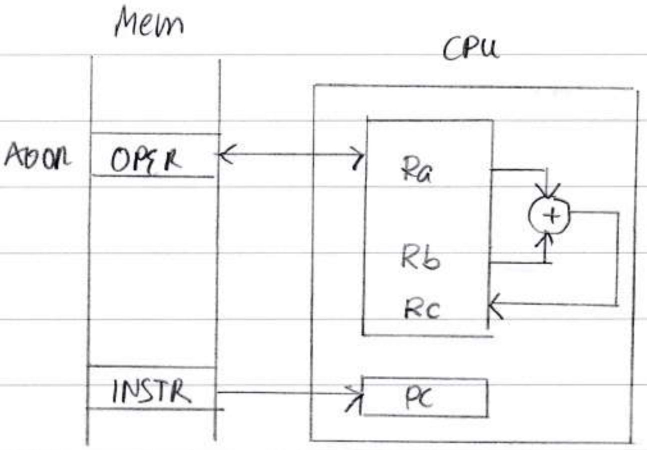
32

load/store

| OPCODE | Ra | Rb | Rc | |
|---|---|---|---|---|

32

Computation



(Ex Mid 1980s idea : MIPS + ARM, SPARC
RISC : Reduced Instruction Set Computer

△ Addressing modes : Effective address = operand address

- Immediate R[ra] ← const

| OPCODE | Ra | ± CONST |
|---|---|---|

- direct R[ra] ← M[const]

- indirect R[ra] ← M[ M[const] ]

Note: Constant must either be "full address width"
or automatically expanded to such a width in
combination with some other instruction that adds
missing information (eg (ADDR1 << 16) | ADDR2)

Q: Which addressing mode is equiv. to data variable?
How about pointer and pointer-to-pointer?

Q: Which is really bad for pipelined CPU?

- Register

const?

$R[ra] \leftarrow R[rb] + R[rc]$

| OPCODE | Ra | Rb | Rc | X |
|---|---|---|---|---|

- Register direct $R[ra] \leftarrow M[\,R[rb]\,]$

- Base plus offset $R[ra] \leftarrow M[\,R[rb] + \text{Const}\,]$
  (aka displacement)

- Base plus index $R[ra] \leftarrow M[\,R[rb] + R[rc]\,]$

Q: How could the above be used to access an array?

p=A, *p++ A[0], A[1] $A[i] \equiv *(A+i)$

- PC relative $R[ra] \leftarrow M[PC + \text{const}]$

| OPCODE | ra | ±const |
|---|---|---|

$PC \leftarrow M[PC + \text{Const}]$

| OPCODE | ± const |
|---|---|

(Bigger displacement)

- Auto increment / decrement + Before / after use

*p++: $R[ra] \leftarrow M[\,R[rb]\,]$, $R[rb] \leftarrow R[rb] + 4$

*(++p): $R[rb] \leftarrow R[rb] + 4$, $R[ra] \leftarrow M[\,R[rb]\,]$

## Modern ISA Examples

(Dominant market)

↓

- Intel / AMD IA-32 / IA-64 (x86) PCs, Laptops, Servers
  CISC

- MIPS (Microproc w/o interlocked pipelined stage)
  RISC
  Embedded systems
  (Ex Automotive, home entertainment
  network routers, gaming devices

- ARM (Acorn → Advanced RISC Machine)
  RISC (CISC)
  Smart phones, tablets

## MIPS overview

- See mips_reference 1, 2

- MARS Simulator / Tools / MIPS x-ray