CS 223 Computer Architecture and Organization

# Instruction Sets: Characteristics and Functions

**J. K. Deka**

**Professor**

**Department of Computer Science & Engineering**

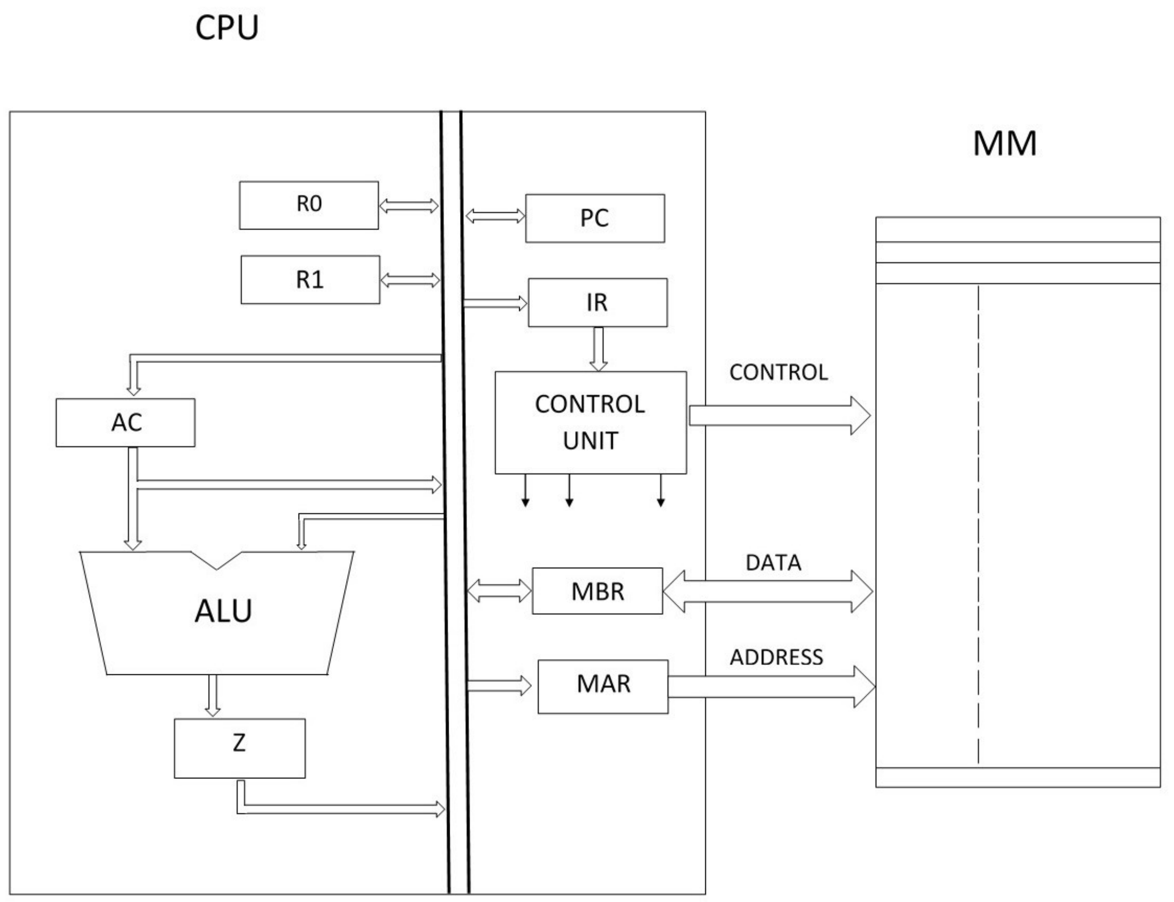
**Indian Institute of Technology Guwahati, Assam.**

## What is an Instruction Set?

- The complete collection of instructions that are understood by a CPU
- Machine Code
  - Binary
- Usually represented by assembly codes

| High Level Code | Assembly Code | Machine Code in HEX (Binary) |
|---|---|---|
| Y = X + Y | LDA 940<br>ADD 941<br>STA 941 | 1940 (0001 1001 0100 0000)<br>5941 (0101 1001 0100 0001)<br>2941 (0010 1001 0100 0001) |

# CPU Organization



Fetch Cycle:

MAR <- PC
Read
PC <- PC+1
IR <- MBR

## What is an Instruction Set?

- Instruction set of a CPU with opcode size of 4 bits.
- CUP has three GPRs: AC, $R_0$ and $R_1$

| OPCODE | Operation | OPCODE | Operation |
|---|---|---|---|
| 0000 | NO OPERATION (NO-OP) | 1000 | MOV $R_0$, AC ($R_0$ = AC) |
| 0001 | LDA M (AC = [M]) | 1001 | MOV $R_1$, AC (R1 = AC) |
| 0010 | STA M ( [M] = AC) | 1010 | MOV AC, $R_0$ (AC = $R_0$) |
| 0011 | DEC AC (AC=AC-1) | 1011 | MOV AC, $R_1$ (AC = R1) |
| 0100 | INC AC (AC=AC+1) | 1100 | SUB M (AC = AC-[M]) |
| 0101 | ADD M (AC=AC+[M]) | 1101 | SUB $R_0$ (AC = AC-$R_0$) |
| 0110 | ADD $R_0$ (AC = AC+$R_0$) | 1110 | SUB $R_1$ (AC = AC-$R_1$) |
| 0111 | ADD $R_1$ (AC = AC+$R_1$) | 1111 | HALT |

## What is an Instruction Set?

- Instruction set of a CPU with opcode size of 4 bits.
- CPU has three GPRs: AC, $R_0$ and $R_1$

| OPCODE | Operation |
|---|---|
| 1000 | MOV $R_0$, AC ($R_0$ = AC) |
| 1001 | MOV $R_1$, AC (R1 = AC) |
| 1010 | MOV AC, $R_0$ (AC = $R_0$) |
| 1011 | MOV AC, $R_1$ (AC = R1) |

May have different Format!!

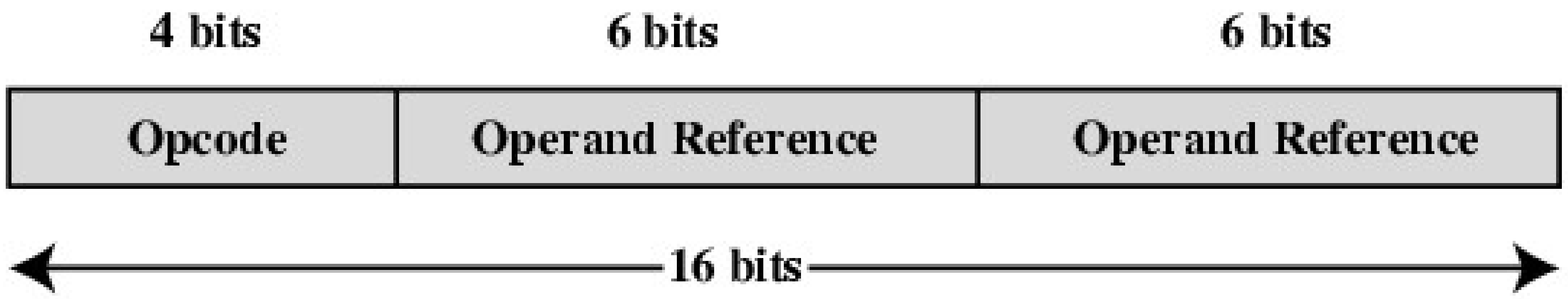
## Elements of an Instruction

- Operation code (Op code)
  - Do this
- Source Operand reference
  - To this
- Result Operand reference
  - Put the answer here
- Next Instruction Reference
  - When you have done that, do this...

## Instruction Representation

- In machine code each instruction has a unique bit pattern
- For human consumption (well, programmers anyway) a symbolic representation is used
  - e.g. ADD, SUB, LOAD
- Operands can also be represented in this way
  - ADD A,B

## Simple Instruction Format



## Instruction Types

- Data processing
- Data storage (main memory)
- Data movement (I/O)
- Program flow control

# Number of Addresses (a)

- 3 addresses
  - Operand 1, Operand 2, Result (2 sources, one destination)
  - a = b + c;
  - ADD a, b, c
  - May be a forth - next instruction (usually implicit)
  - Not common
  - Needs very long words to hold everything

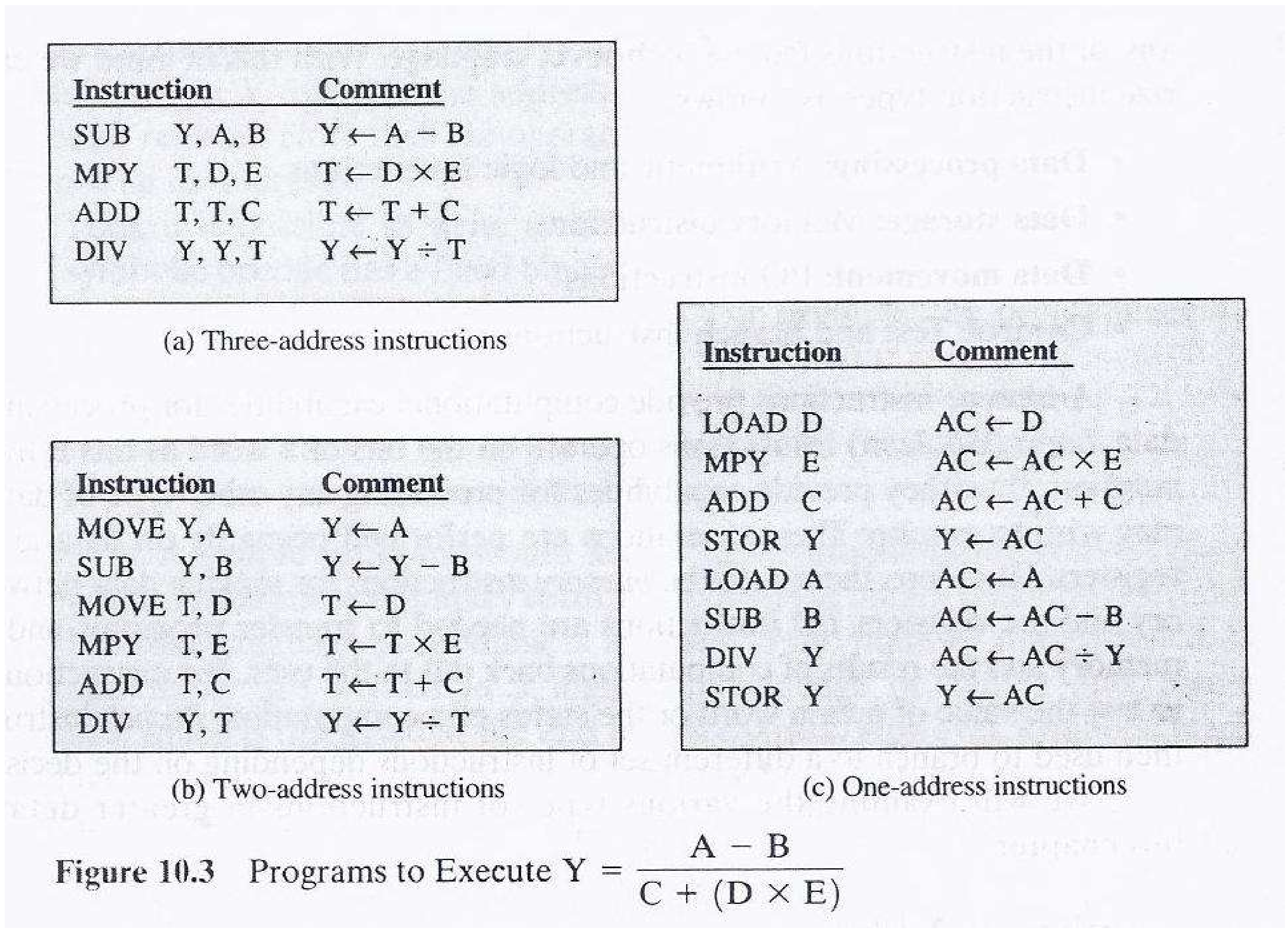
## Number of Addresses (b)

- 2 addresses
  - One address doubles as operand and result (source as well as destination)
  - a = a + b
  - ADD a, b
  - Reduces length of instruction
  - Requires some extra work
    - Temporary storage to hold some results

## Number of Addresses (c)

- 1 address
  - Implicit address of one operand
  - Usually a register (accumulator)
  - Common on early machines

# Number of Addresses



| Instruction | | Comment |
|---|---|---|
| SUB | Y, A, B | $Y \leftarrow A - B$ |
| MPY | T, D, E | $T \leftarrow D \times E$ |
| ADD | T, T, C | $T \leftarrow T + C$ |
| DIV | Y, Y, T | $Y \leftarrow Y \div T$ |

(a) Three-address instructions

| Instruction | | Comment |
|---|---|---|
| MOVE | Y, A | $Y \leftarrow A$ |
| SUB | Y, B | $Y \leftarrow Y - B$ |
| MOVE | T, D | $T \leftarrow D$ |
| MPY | T, E | $T \leftarrow T \times E$ |
| ADD | T, C | $T \leftarrow T + C$ |
| DIV | Y, T | $Y \leftarrow Y \div T$ |

(b) Two-address instructions

| Instruction | | Comment |
|---|---|---|
| LOAD | D | $AC \leftarrow D$ |
| MPY | E | $AC \leftarrow AC \times E$ |
| ADD | C | $AC \leftarrow AC + C$ |
| STOR | Y | $Y \leftarrow AC$ |
| LOAD | A | $AC \leftarrow A$ |
| SUB | B | $AC \leftarrow AC - B$ |
| DIV | Y | $AC \leftarrow AC \div Y$ |
| STOR | Y | $Y \leftarrow AC$ |

(c) One-address instructions

**Figure 10.3** Programs to Execute $Y = \dfrac{A - B}{C + (D \times E)}$

## Number of Addresses (d)

- 0 (zero) addresses
  - All addresses implicit
  - Uses a stack
  - e.g. push a
  - push b
  - add
  - pop c

  - c = a + b

## How Many Addresses

- More addresses
  - More complex (powerful?) instructions
  - More registers
    - Inter-register operations are quicker
  - Fewer instructions per program
- Fewer addresses
  - Less complex (powerful?) instructions
  - More instructions per program
  - Faster fetch/execution of instructions

## Design Decisions (1)

- Operation repertoire
  - How many ops?
  - What can they do?
  - How complex are they?
- Data types
- Instruction formats
  - Length of op code field
  - Number of addresses

## Design Decisions (2)

- Registers
  - Number of CPU registers available
  - Which operations can be performed on which registers?
- Addressing modes

## Reference

Computer Organization and Architecture –
Designing for Performance
William Stallings

Chapter 10: Page no. 334 – 342 (Seventh Edition)
Page No.: 349 – 356 (Eighth Edition)