Unit 5

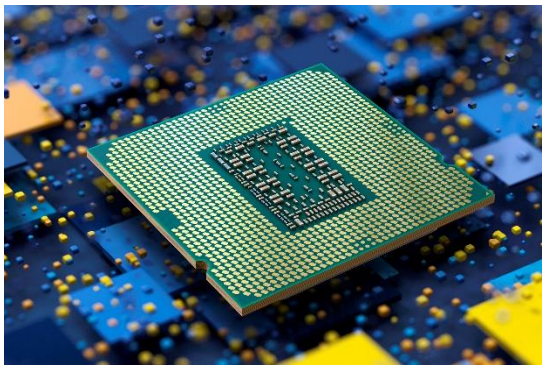
Central Processing Unit

5.1 CPU Organizations

5.2 Instruction Formats

5.3 Addressing Modes

Sanjeev Thapa, Er. DevOps, SRE, CKA, RHCSA, RHCE, RHCSA-Openstack, MTCNA, MTCTCE, UBSRS, HEv6, Research Evangelist



The **Central Processing Unit (CPU)** is the core component of a computer system.
It fetches instructions, decodes them, executes operations, and controls all system activities.

Main parts of CPU:

- **ALU (Arithmetic Logic Unit)**
- **Control Unit (CU)**
- **Register Set**

5.1 CPU ORGANIZATIONS

CPU organization refers to **how registers, ALU, and memory are arranged** and how instructions operate on data.

Types of CPU Organizations

---

1. Accumulator-Based Organization

- Uses a single register called **Accumulator (AC)**
- Most operations use AC implicitly

Instruction Format:

ADD X → AC ← AC + M[X]

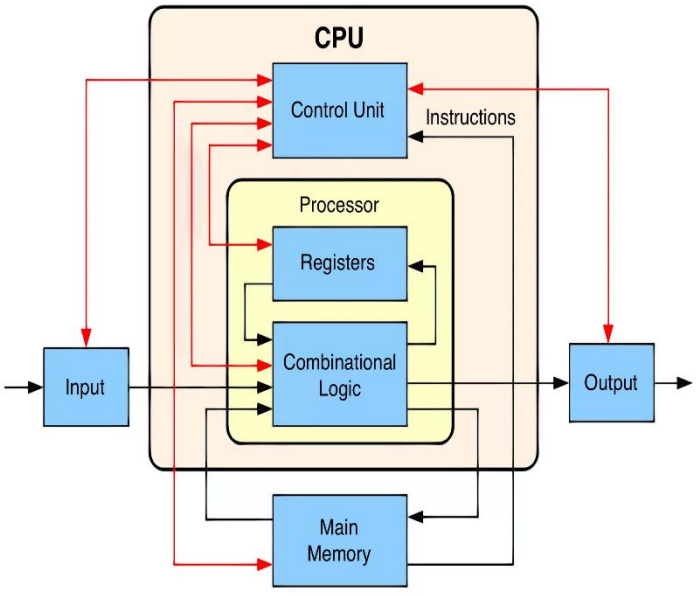
Example

AC = 10

Memory[20] = 5

ADD 20

AC = 15



Advantages

- Simple hardware
- Short instructions

Disadvantages

- More memory accesses
- Slower execution

---

2. General Register Organization

- Uses multiple general-purpose registers (R1, R2, R3…)
- Faster than accumulator-based

Instruction Format **ADD R1, R2, R3 → R1 ← R2 + R3**

Example

R2 = 8

R3 = 6

ADD R1, R2, R3

R1 = 14

Advantages

- Fast execution
- Fewer memory accesses

Disadvantages

- More complex instruction format

---

3. Stack-Based Organization

- Uses stack (LIFO structure)
- Operands are implicit

Instruction Format **PUSH, POP, ADD**

Example

PUSH 4

PUSH 6

ADD

Execution

4 and 6 popped

4 + 6 = 10 pushed back

Advantages

- No address fields needed
- Compact instructions

Disadvantages

- Limited flexibility
- Harder to optimize

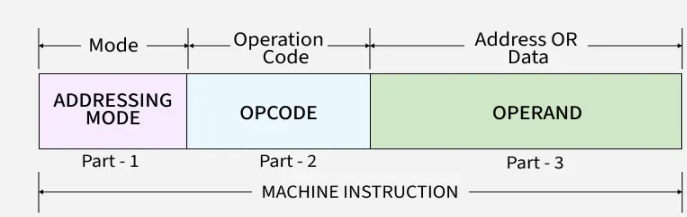
Comparison Table

Organization	Registers Used	Example Instruction
Accumulator	AC only	ADD X
General Register	Multiple registers	ADD R1, R2, R3
Stack	Stack	ADD

Instruction format defines **how an instruction is structured** in memory.

Basic Instruction Format

[Opcode | Operand / Address]



Types of Instruction Formats

---

1. Zero-Address Format

(Stack-based)

Example

ADD

Execution

- Pops two stack values
- Adds them
- Pushes result

---

2. One-Address Format

(Accumulator-based)

Example

ADD 30

Execution

AC ← AC + M[30]

---

3. Two-Address Format

Example

ADD R1, R2

Execution

R1 ← R1 + R2

---

4. Three-Address Format

Example

ADD R1, R2, R3

Execution

R1 ← R2 + R3

Instruction Format Comparison

Format	Addresses	Example
Zero	0	ADD
One	1	ADD X
Two	2	ADD R1, R2
Three	3	ADD R1, R2, R3

5.3 ADDRESSING MODES

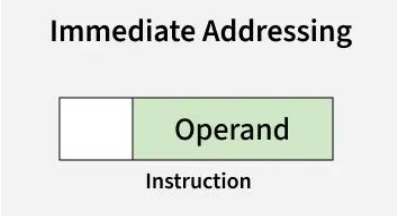
Addressing modes define **how the operand of an instruction is accessed**.

1. Immediate Addressing Mode

Operand is inside instruction.

Example

MOV A, #5



Meaning

A ← 5

---

2. Direct Addressing Mode

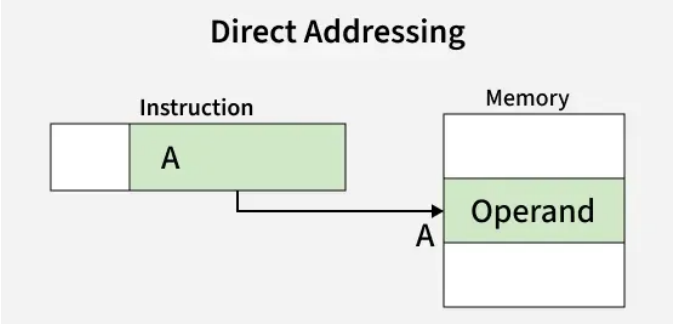
Address of operand is specified.

Example

LOAD 2000H

Meaning

AC ← Memory[2000H]



---

3. Indirect Addressing Mode

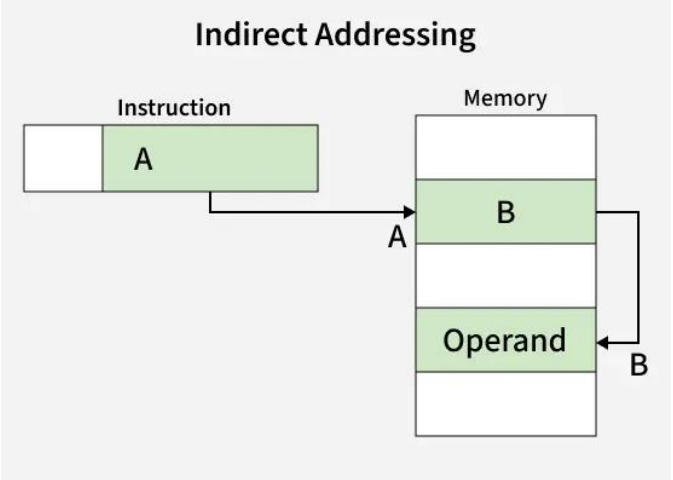
Instruction points to address that holds operand address.

Example

LOAD (2000H)

Meaning

AC ← Memory[Memory[2000H]]



---

4. Register Addressing Mode

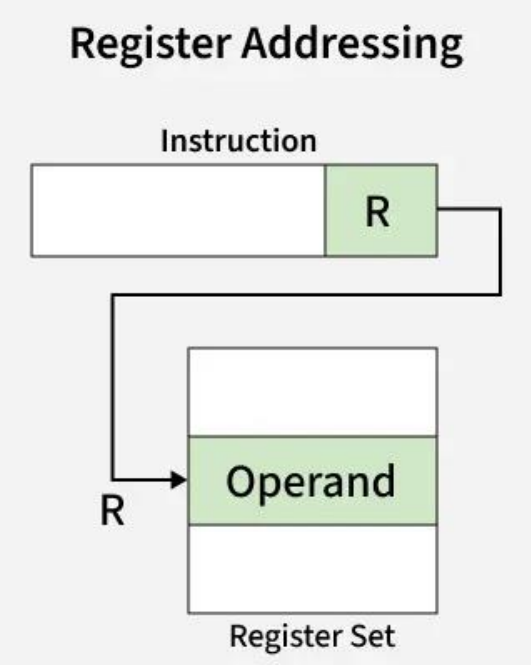
Operand is in register.

Example

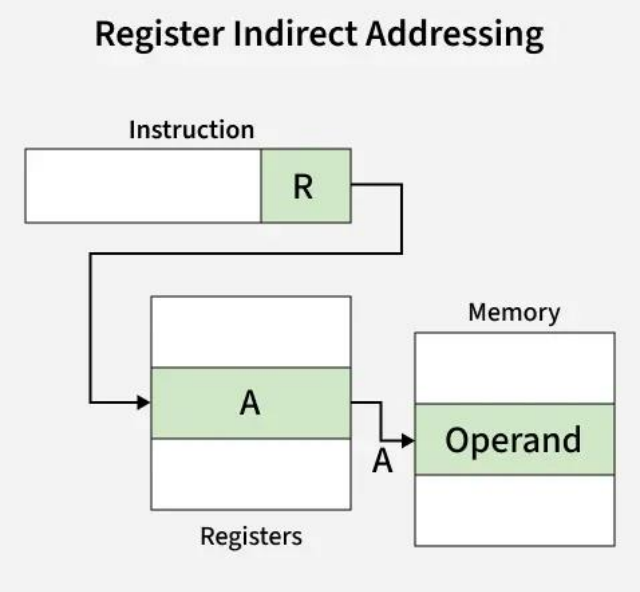
ADD B

Meaning

A ← A + B



5. Register Indirect Addressing Mode



Register holds memory address.

Example

MOV A, M (8085)

Meaning

A ← Memory[HL]

---

6. Indexed Addressing Mode

Effective address = Base + Index

Example

LOAD ARRAY[R1]

Meaning

AC ← Memory[ARRAY + R1]

Addressing Modes Summary Table

Mode	**Operand Location**	**Example**
Immediate	Inside instruction	MOV A, #5
Direct	Memory address	LOAD 2000H
Indirect	Address of address	LOAD (2000H)

Mode	**Operand Location**	**Example**
Register	Register	ADD B
Register Indirect	Memory via register	MOV A, M
Indexed	Base + index	LOAD A[R1]