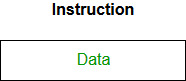
**Types of Addressing Modes in 8086 Microprocessor**

Addressing modes in computer architecture define how the operands (data) are accessed by an instruction. The **8086 microprocessor** supports multiple addressing modes to enhance flexibility and efficiency in executing instructions.

**1. Implied Addressing Mode**

* The operand is **implicitly specified** in the instruction itself.
* The data is either **8-bit or 16-bit** and is part of the instruction.
* This mode is mainly used for instructions that **do not require an explicit operand**.



**Example:**

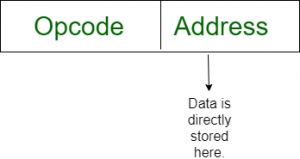
* CLC → Clears the **Carry Flag (CF)** (sets CF = 0).

💡 **Use Case:**

* Used in flag operations, accumulator-based instructions, and stack operations.

**2. Immediate Addressing Mode (Symbol: #)**

* The operand (data) is provided **directly in the instruction**.
* The instruction contains the actual value to be used.
* Limited by the **size of the address field**, so only small constants can be used.



**Example:**

* MOV AL, 35H → Move the value **35H** directly into the **AL register**.

💡 **Use Case:**

* Used when assigning a constant value to a register.

**3. Register Addressing Mode**

* The operand is stored in a **general-purpose register** (8-bit or 16-bit).
* The instruction specifies which register holds the data.

am3

**Example:**

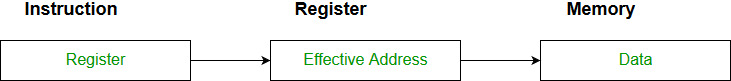
* MOV AX, CX → Copy the contents of **CX** into **AX**.

💡 **Use Case:**

* Fastest mode as no memory access is required.

**4. Register Indirect Addressing Mode**

* The operand is stored in **memory**, but the memory address is held in a **register**.
* The effective address is stored in **BX, BP, SI, or DI registers**.



**Example:**

* MOV AX, [BX] → Load **AX** with the contents of the memory location pointed to by **BX**.

💡 **Use Case:**

* Useful for working with **arrays, pointers, and tables**.

**5. Auto-Indexed Addressing Mode**

**Auto-Increment Mode**

* The effective address of the operand is stored in a **register**.
* After accessing the operand, the register is **automatically incremented**.

**Example:**

ADD R1, (R2)+

Equivalent to:

R1 = R1 + M[R2]

R2 = R2 + d

💡 **Use Case:**

* **Loops and arrays** (useful for sequential memory access).

**Auto-Decrement Mode**

* The effective address of the operand is stored in a **register**.
* Before accessing the operand, the register is **automatically decremented**.

**Example:**

ADD R1, -(R2)

Equivalent to:

R2 = R2 - d

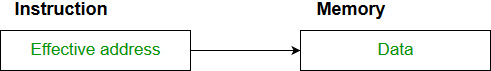
R1 = R1 + M[R2]

💡 **Use Case:**

* Used in **stack operations** (Last-In-First-Out behavior).

**6. Direct Addressing Mode (Absolute Addressing) (Symbol: [ ])**

* The memory address of the operand is **explicitly mentioned** in the instruction.
* The instruction contains a **16-bit memory address**.



**Example:**

* ADD AL, [0301] → Add the value at memory address **0301H** to **AL**.

💡 **Use Case:**

* Useful for **accessing global variables** stored at a fixed location in memory.

**7. Indirect Addressing Mode (Symbol: @ or ())**

* The instruction contains the **address of the effective address**.
* Requires **two memory accesses**:
  1. First to **fetch** the effective address.
  2. Second to **fetch** the actual operand.

**Types of Indirect Addressing:**

**a) Register Indirect**

* The effective address is stored in a **register**.
* Requires **one register reference + one memory reference**.

**Example:**

* MOV A, @R0 → Move the contents of memory location pointed by **R0** into **A**.

**b) Memory Indirect**

* The effective address is stored **in memory**.
* Requires **two memory references**.

**Example:**

* MOV AX, [[5000H]] → Move the contents of memory location **5000H** into **AX**.

💡 **Use Case:**

* Useful in **pointer-based operations**.

**8. Indexed Addressing Mode**

* The effective address is obtained by **adding an index register** (SI or DI) **to a displacement**.

**Example:**

* MOV AX, [SI + 05] → Move the value from **(SI + 5) memory location** into **AX**.

💡 **Use Case:**

* Used in **array processing**.

**9. Based Indexed Addressing Mode**

* The effective address is the sum of a **base register (BX)** and an **index register (SI or DI)**.

**Example:**

* ADD AX, [BX + SI] → Add the contents of memory location **(BX + SI)** to **AX**.

💡 **Use Case:**

* Used in **multi-dimensional arrays** and **structured data access**.

**10. PC Relative Addressing Mode**

* The effective address is determined by **adding a displacement to the Program Counter (PC)**.
* Used for **intra-segment control transfers**.

**Example:**

EA = PC + Address Field Value

PC = PC + Relative Value

💡 **Use Case:**

* Used for **jump instructions (JMP, JZ, JNZ, etc.)**.

**11. Base Register Addressing Mode**

* The effective address is obtained by **adding a base register value** to an **address field value**.
* Used for **inter-segment control transfers**.

**Example:**

EA = Base Register + Address Field Value

PC = Base Register + Relative Value

💡 **Use Case:**

* Used for **segmentation** in memory.

**Advantages of Addressing Modes (Summary)**

* Enable advanced techniques like **pointers** and **loop counters**.
* Simplify memory access for **arrays** and **complex data structures**.
* Support **program relocation** during runtime.
* Optimize **instruction size**, improving efficiency.