**Addressing Modes**

**Addressing Modes**– The term addressing modes refers to the way in which the operand of an instruction is specified. The addressing mode specifies a rule for interpreting or modifying the address field of the instruction before the operand is actually executed.

**Addressing modes for instructions are divided into two categories:**

1) Addressing modes for data

2) Addressing modes for branch

The addressing modes provide flexible access to memory, allowing you to easily access variables, arrays, records, pointers, and other complex data types.  The key to good assembly language programming is the proper use of memory addressing modes.

An assembly language program instruction consists of two parts

[am1](https://media.geeksforgeeks.org/wp-content/cdn-uploads/Addressing_Modes_1.jpg)  
The memory address of an operand consists of two components:

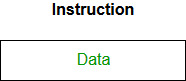
**IMPORTANT TERMS**

* + **Address Field:** Address field may designate a memory address or a processor register.
  + **Starting address** of memory segment.
    - **Effective address**: The effective address is defined to be the memory address obtained from the computation dictated by the given addressing mode. The effective address is the address of the operand in a computational-type instruction.
  + **Offset address:** An offset is determined by adding any combination of three address elements: **displacement, base, and index.**
  + **Displacement:**It is an 8 bit or 16-bit immediate value given in the instruction.
  + **Base**: Contents of base register, BX, or BP.
  + **Index**: Content of index register SI or DI.

According to different ways of specifying an operand in a microprocessor is known as addressing modes.

**Types of Addressing Modes**

1. **Addressing modes for data**
2. **Implied mode:** In implied addressing the operand is specified in the instruction itself. In this mode the data is 8 bits or 16 bits long and data is the part of instruction. Zero address instruction are designed with implied addressing mode.

[](https://media.geeksforgeeks.org/wp-content/cdn-uploads/Addressing_Modes_2.jpg)

For Example:  CLC (used to reset Carry flag to 0), CMA, INC A, RLC, RAR etc.

1. **Immediate addressing mode (symbol #):**
2. In this mode data is present in address field of instruction. Designed like one address instruction format.
3. This instruction has an operand field rather than an address field. The operand field contains the actual operand to be used in conjunction with the operation specified in the instruction.
4. Timeline

   Description automatically generated with medium confidenceThese instructions are useful for initializing register to a constant value.

For Example:  MOV AL, 35H (move the data 35H into AL register)

ADD AX, #5H (A=A+5)

LDA, #25H (Load 25H into the accumulator)

AND AX, 0000H (AX=AX^0000)

**Note:**

* + 1. *Limitation in the immediate mode is that the range of constants are restricted by size of address field.*
    2. *No memory reference.*
    3. *Fast execution.*

|  |
| --- |
|  |
|  |
| Operand |
|  |

1. **Register direct addressing mode:**
   * 1. In this mode, the operands are in registers that reside within the CPU.
     2. The register is selected from the register field in the instruction.

Processor Register

Register

Opcode

For Example: MOV AX, CX (move the contents of CX register to AX register)

ADD BL (AL=AL+BL)

**Note:**

1. *Effective Address (EA) = R*
2. *Here one register reference is required to access the data.*
3. *No memory reference.*
4. *Limited address space*
5. **Register Indirect mode**
   * 1. In this mode the instruction specifies a register in the CPU whose contents give the address of the operand in the memory.
     2. In other words, the selected register contains the address of the operand rather than the operand itself.
     3. Before using a register indirect mode instruction, the programmer must ensure that the memory address of the operand is placed in the processor register with a previous instruction.

Instruction

Register

Opcode

Processor Register

Memory

|  |
| --- |
|  |
|  |
| Operand |
|  |

**Note:**

* + 1. *Effective Address (EA) = (R) or [R]*
    2. *Here two register reference is required to access the data.*
    3. *Large address space.*
    4. *The address field of the instruction uses fewer bits to select a register than would have been required to specify a memory address directly.*
    5. *The CPUs let you access memory indirectly through a register using the register indirect addressing modes.*

For example: MOV AX, [BX] (move the contents of memory location addressed by the register BX to the register AX)

MOV AX, [DI]

ADD AL, [BX]

MOV AX, [SI]

1. **Auto Indexed (increment mode)**: Effective address of the operand is the contents of a register specified in the instruction. After accessing the operand, the contents of this register are automatically incremented to point to the next consecutive memory location **(R1) +**.

*Here one register reference, one memory reference and one ALU operation is required to access the data.*

Example:

Add R1, (R2) +  // OR

R1 = R1 +M[R2]

R2 = R2 + d

*Useful for stepping through arrays in a loop. R2 – start of array d* – size of an element.

1. **Auto indexed (decrement mode)**: Effective address of the operand is the contents of a register specified in the instruction. Before accessing the operand, the contents of this register are automatically decremented to point to the previous consecutive memory location. *–***(R1)**

*Here one register reference, one memory reference and one ALU operation is required to access the data.*

**Example:**

Add R1, -(R2)   //OR

R2 = R2-*d*

R1 = R1 + M[R2]

*Auto decrement mode is same as auto increment mode. Both can also be used to implement a stack as push and pop. Auto increment and Auto decrement modes are useful for implementing “Last-In-First-Out” data structures.*

1. **Direct addressing/ Absolute addressing Mode (symbol []):**
   * 1. In this mode the effective address is equal to the address part of the instruction.
     2. The operand resides in memory and its address is given directly by the address field of the instruction.

Instruction

Address

Opcode

|  |
| --- |
|  |
|  |
| Operand |
|  |

Main Memory

Effective Address (EA) = A

Advantage: Simple. Disadvantage: limited address field

For Example: ADD AL, [0301H] //add the operand of offset address 0301H to AL

MOV AX, [0500H] //move the operand of offset address 0500H to AX

**Note:**

1. *Here only one memory reference operation is required to access the data.*
2. *Simple.*
3. *Limited address Field*
4. **Indirect addressing Mode (symbol @ or ()):**

In this mode address field of instruction contains the address of effective address. Here two references are required.   
 a) 1st reference to get effective address.   
 b) 2nd reference to access the data.

Based on the availability of Effective address, Indirect mode is of two kinds:

1. Register Indirect: In this mode effective address is in the register, and corresponding register name will be maintained in the address field of an instruction.

***Note: Here one register reference, one memory reference is required to access the data.***

* + - * 1. Memory Indirect: In this mode effective address is in the memory, and corresponding memory address will be maintained in the address field of an instruction.

***Note: Here two memory reference is required to access the data.***

For example: MOV AX, (A)

ADD AX, @BX.

1. **Based on Transfer of control**
   * + 1. **PC relative addressing mode:** PC relative addressing mode is used to implement intra segment transfer of control. In this mode effective address is obtained by adding displacement to PC.

EA= PC + Address field value

PC= PC + Relative value.

**Note:**

1. *Used for program control instructions.*
   * + 1. **Base register addressing mode:** Base register addressing mode is used to implement inter segment transfer of control. In this mode effective address is obtained by adding base register value to address field value.

EA= Base register + Address field value.

PC= Base register + Relative value.

For example: MOV DX, [BX+04]

ADD CL, [BX+08]

**Note:**

1. *PC relative and based register both addressing modes are suitable for program relocation at runtime.*
2. *Based register addressing mode is best suitable to write position independent codes.*
3. *The base register addressing mode is used in computers to facilitate the relocation of programs in memory i.e., when programs and data are moved from one segment of memory to another.*
   * + 1. **Indexed addressing mode**: The operand’s offset is the sum of the content of an index register SI or DI and an 8 bit or 16-bit displacement.

For example: MOV AX, [SI +05]

ADD AL, [DI+16]

**Note**: 1) Use to access or implement array efficiently.

2) Multiple registers required to implement.

3) Any element can be accessed without changing instruction.

* + - 1. **Based Indexed Addressing:**The operand’s offset is sum of the content of a base register BX or BP and an index register SI or DI.

For Example: ADD AX, [BX+SI]

MOV AX, [SI+2000]

MOV AL, [DI+3000]

**Applications of Addressing Modes-**

|  |  |
| --- | --- |
| **Addressing Modes** | **Applications** |
| **Immediate Addressing Mode** | * To initialize registers to a constant value |
| **Direct Addressing Mode**  **and**  **Register Direct Addressing Mode** | * To access static data * To implement variables |
| **Indirect Addressing Mode**  **and**  **Register Indirect Addressing Mode** | * To implement pointers because pointers are memory locations that store the address of another variable * To pass array as a parameter because array name is the base address and pointer is needed to point the address |
| **Relative Addressing Mode** | * For program relocation at run time i.e. for position independent code * To change the normal sequence of execution of instructions * For branch type instructions since it directly updates the program counter |
| **Index Addressing Mode** | * For array implementation or array addressing * For records implementation |
| **Base Register Addressing Mode** | * For writing relocatable code i.e. for relocation of program in memory even at run time * For handling recursive procedures |
| **Auto-increment Addressing Mode**  **and**  **Auto-decrement Addressing Mode** | * For implementing loops * For stepping through arrays in a loop * For implementing a stack as push and pop |

**Sample Question**

Q. Match each of the high-level language statements given on the left-hand side with the most natural addressing mode from those listed on the right-hand side.

1. A [1] = B[J];     a. Indirect addressing

2. while [\*A++];   b. Indexed addressing

3. int temp = \*x;   c. Autoincrement

**(A**) (1, c), (2, b), (3, a)

**(B)** (1, a), (2, c), (3, b)

**(C)** (1, b), (2, c), (3, a)

**(D)** (1, a), (2, b), (3, c)

**Answer:** **(C)**

**Explanation:**

List 1                           List 2

1) A [1] = B[J];      b) Index addressing

Here indexing is used

2) while [\*A++];     c) auto increment

The memory locations are automatically incremented

3) int temp = \*x;    a) Indirect addressing

Here temp is assigned the value of int type stored

at the address contained in X

Hence (C) is correct solution.