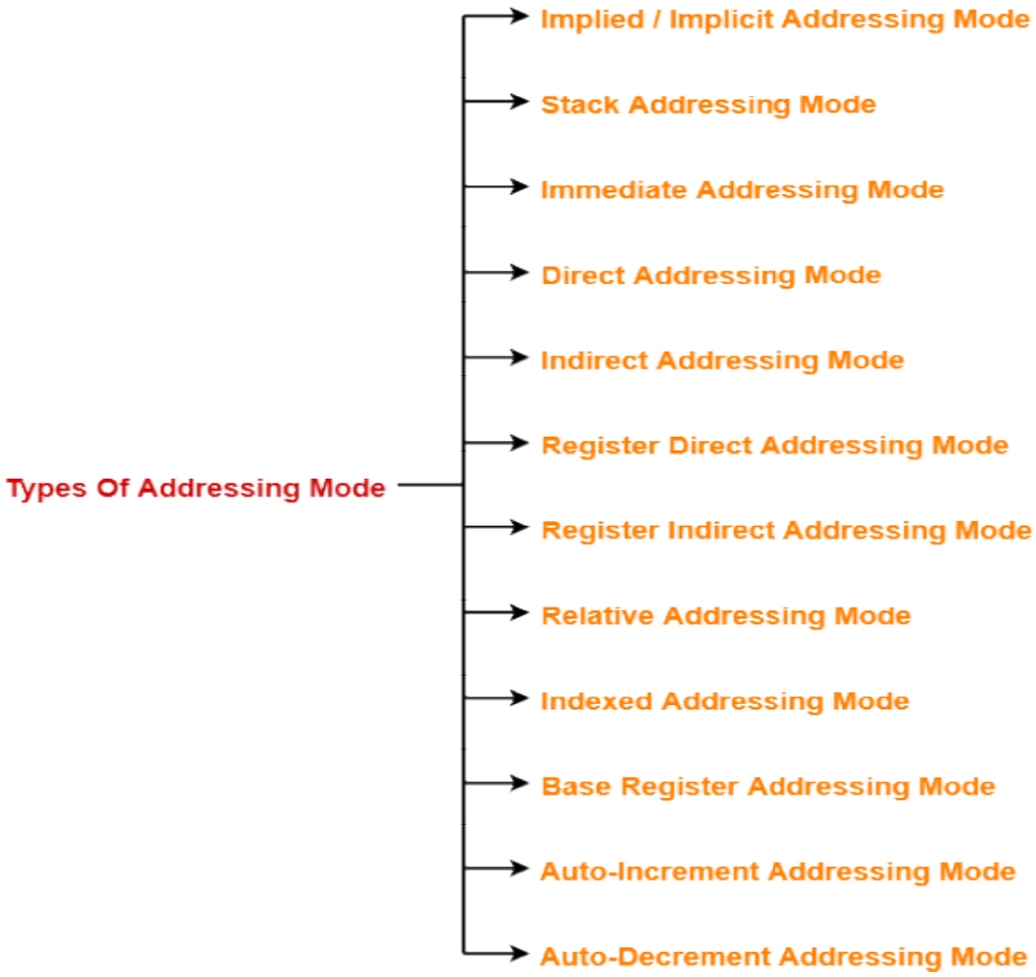
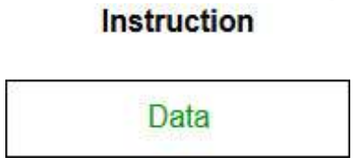
Computer Architecture
Lecture-1

In computer architecture, there are following types of addressing modes-



1. Implied Addressing Mode-

In this addressing mode,
It is also called as **implicit addressing 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.



Example: CLC (used to reset Carry flag to 0)

- ✓ The instruction "Complement Accumulator" is an implied mode instruction.
- ✓ In a stack organized computer, Zero Address Instructions are implied mode instructions.
  (Since operands are always implied to be present on the top of the stack)

2. Stack Addressing Mode-

In this addressing mode,

- ✓ The operand is contained at the top of the stack.

Example-

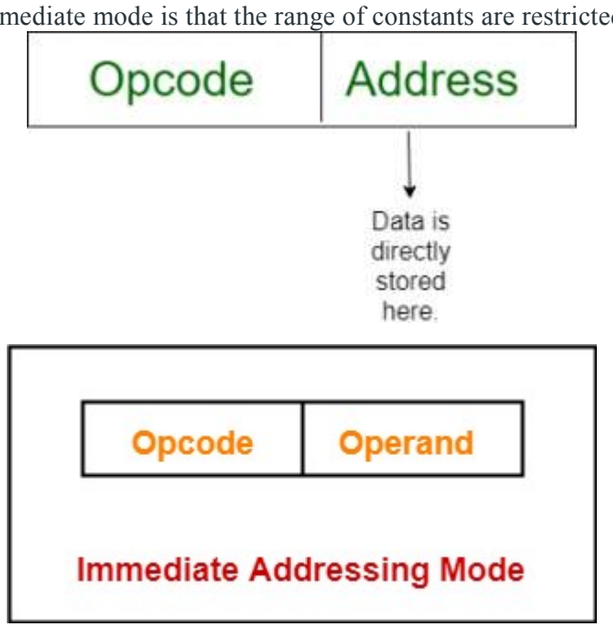
ADD

- ✓ This instruction simply pops out two symbols contained at the top of the stack.
- ✓ The addition of those two operands is performed.
- ✓ The result so obtained after addition is pushed again at the top of the stack.

3. Immediate Addressing Mode-

- ✓ The operand is specified in the instruction explicitly.
- ✓ Instead of address field, an operand field is present that contains the operand.

In this mode data is present in address field of instruction .Designed like one address instruction format.
Note: Limitation in the immediate mode is that the range of constants are restricted by size of address field.



Example:

- ✓ MOV AL, 35H (move the data 35H into AL register)
- ✓ ADD 10 will increment the value stored in the accumulator by 10.
- ✓ MOV R #20 initializes register R to a constant value 20.

```
Solution:   MOV  AX,34F5H        ;AX =34F5H
            ADD  AX,95EBH        ;AX = CAE0H
```

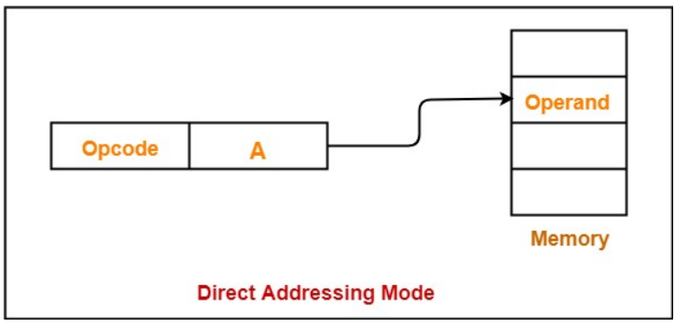
```
  34F5        0011  0100  1111  0101
+ 95EB        1001  0101  1110  1011
  CAE0        1100  1010  1110  0000
```

- ✓

4. Direct Addressing Mode-

The address field of the instruction contains the effective address of the operand.

- ✓ Only one reference to memory is required to fetch the operand.
- ✓ It is also called as **absolute addressing mode**.



Example-

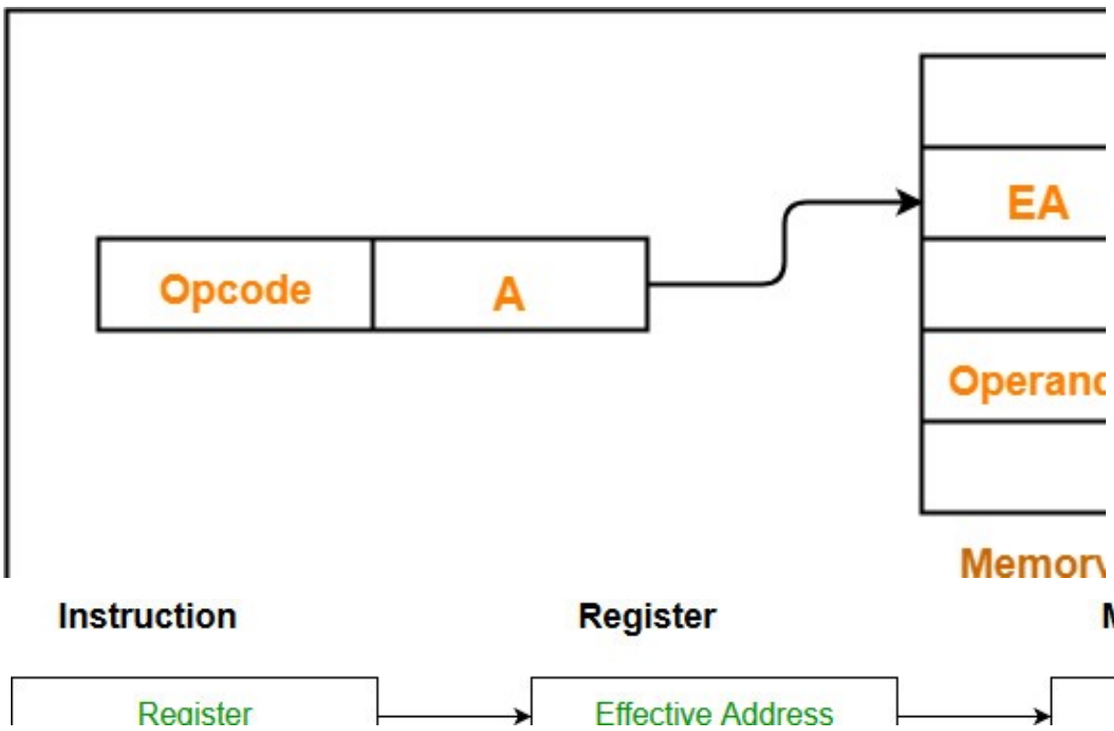
- ✓ ADD X will increment the value stored in the accumulator by the value stored at memory location X.

$$AC \leftarrow AC + [X]$$

5. Indirect Addressing Mode-

The address field of the instruction specifies the address of memory location that contains the effective address of the operand.

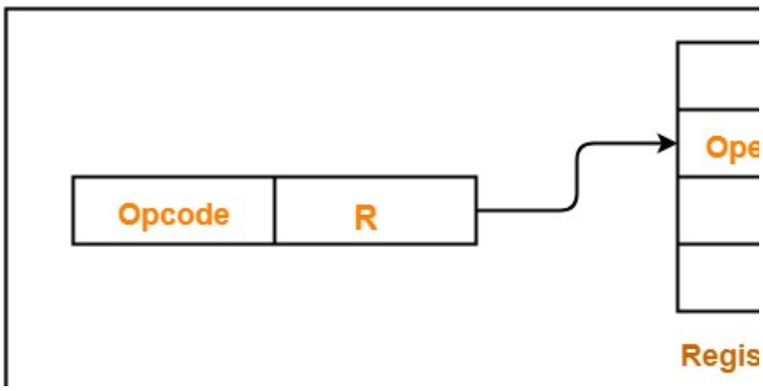
- ✓ Two references to memory are required to fetch the operand.



Example-
MOV AX, [BX](move the contents of memory location s
addressed by the register BX to the register AX)

6. Register Direct Addressing Mode-

- ✓ The operand is contained in a register set.
- ✓ The address field of the instruction refers to a CPU register that contains the operand.
- ✓ No reference to memory is required to fetch the operand.



Example-

- ✓ ADD R will increment the value stored in the accumulator by the content of register R.

$$AC \leftarrow AC + [R]$$

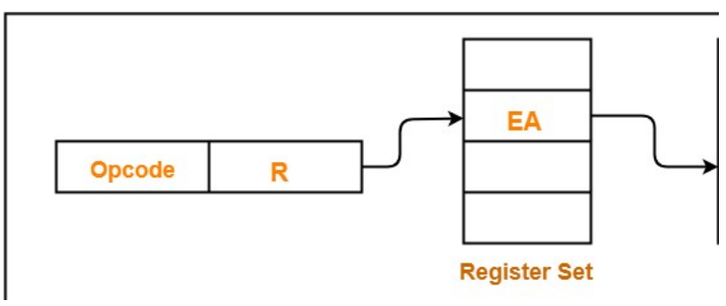
NOTE-
It is interesting to note-

- ✓ This addressing mode is similar to direct addressing mode.
- ✓ The only difference is address field of the instruction refers to a CPU register instead of main memory.

7. Register Indirect Addressing Mode-

The address field of the instruction refers to a CPU register that contains the effective address of the operand.

- ✓ Only one reference to memory is required to fetch the operand.



Example-

- ✓ ADD R will increment the value stored in the accumulator by the content of memory location specified in register R.

$$AC \leftarrow AC + [[R]]$$

- ✓ MOV AX, [BX](move the contents of memory locations addressed by the register BX to the register AX)

NOTE-

It is interesting to note-

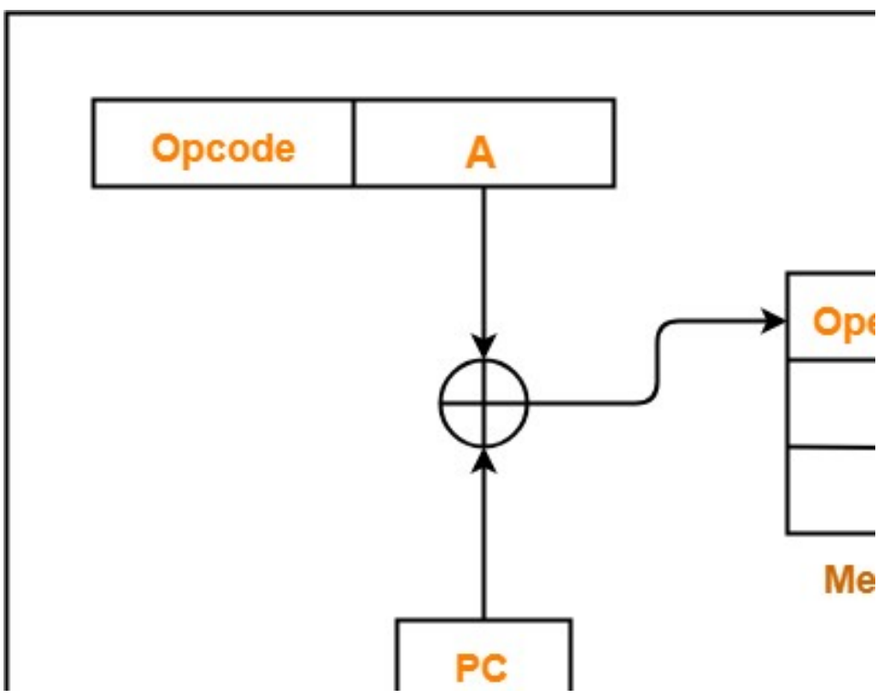
- ✓ This addressing mode is similar to indirect addressing mode.
- ✓ The only difference is address field of the instruction refers to a CPU register.

8. Relative Addressing Mode-

- ✓ Effective address of the operand is obtained by adding the content of program counter with the address part of the instruction.

Effective Address

= Content of Program Counter + Address part of the instruction



NOTE-

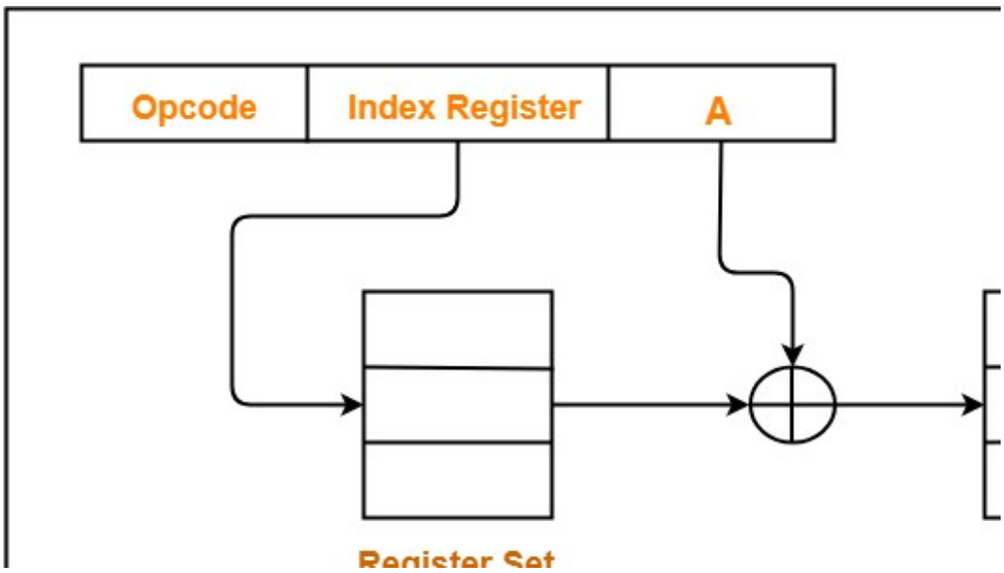
- ✓ **Program counter** (PC) always contains the address of the next instruction to be executed.
- ✓ After fetching the address of the instruction, the value of program counter immediately increases.
- ✓ The value increases irrespective of whether the fetched instruction has completely executed or not.

9. Indexed Addressing Mode-

- ✓ Effective address of the operand is obtained by adding the content of index register with the address part of the instruction.

Effective Address

= Content of Index Register + Address part of the instruction

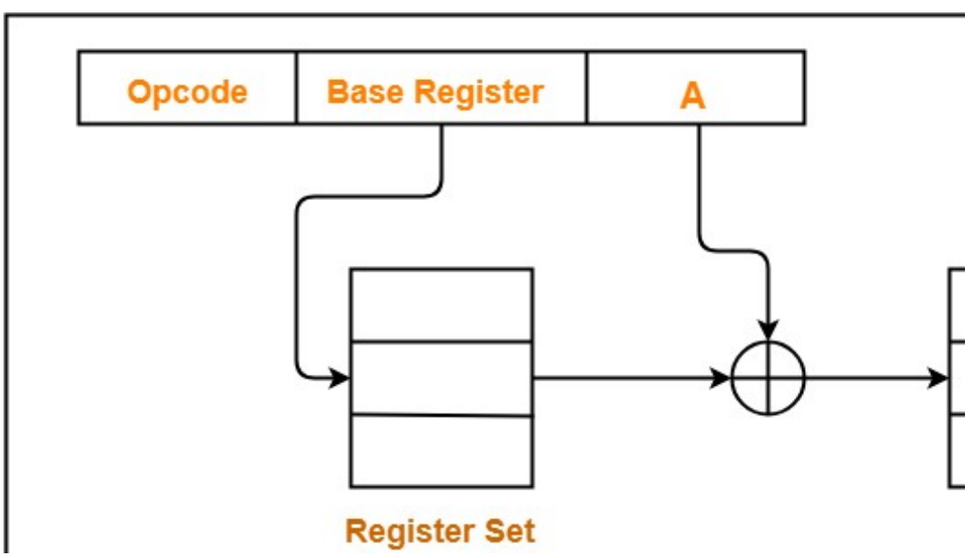


10. Base Register Addressing Mode-

- ✓ Effective address of the operand is obtained by adding the content of base register with the address part of the instruction.

Effective Address

= Content of Base Register + Address part of the instruction



11. Auto-Increment Addressing Mode-

- ✓ This addressing mode is a special case of Register Indirect Addressing Mode where-

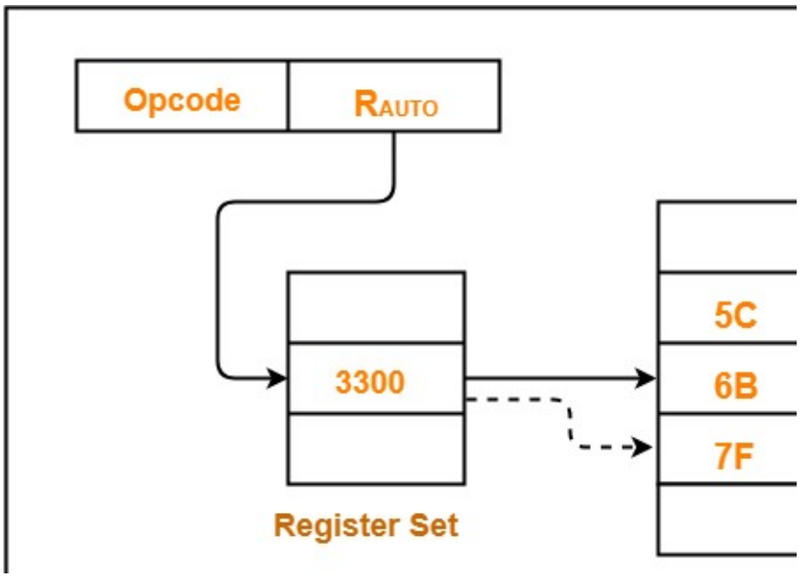
Effective Address of the Operand

= Content of Register

In this addressing mode,

- ✓ After accessing the operand, the content of the register is automatically incremented by step size 'd'.
- ✓ Step size 'd' depends on the size of operand accessed.
- ✓ Only one reference to memory is required to fetch the operand.

Example-



Assume operand size = 2 bytes.

Here,

- ✓ After fetching the operand 6B, the instruction register R_{AUTO} will be automatically incremented by 2.
- ✓ Then, updated value of R_{AUTO} will be 3300 + 2 = 3302.
- ✓ At memory address 3302, the next operand will be found.

NOTE-

In auto-increment addressing mode,

- ✓ First, the operand value is fetched.
- ✓ Then, the instruction register R_{AUTO} value is incremented by step size 'd'.

12. Auto-Decrement Addressing Mode-

- ✓ This addressing mode is again a special case of Register Indirect Addressing Mode where-

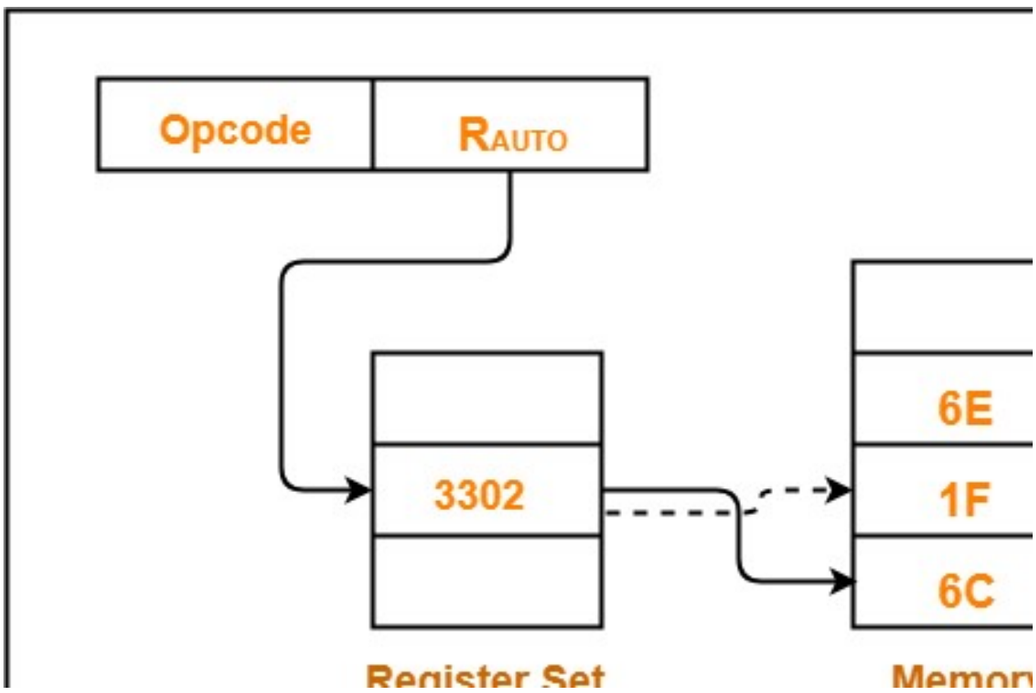
Effective Address of the Operand

= Content of Register – Step Size

In this addressing mode,

- ✓ First, the content of the register is decremented by step size 'd'.
- ✓ Step size 'd' depends on the size of operand accessed.
- ✓ After decrementing, the operand is read.
- ✓ Only one reference to memory is required to fetch the operand.

Example-



Assume operand size = 2 bytes.

Here,

- First, the instruction register R_{AUTO} will be decremented by 2.
- Then, updated value of R_{AUTO} will be 3302 – 2 = 3300.
- At memory address 3300, the operand will be found.

NOTE-

In auto-decrement addressing mode,

- First, the instruction register R_{AUTO} value is decremented by step size 'd'.
- Then, the operand value is fetched.

Also Read- Practice Problems On Addressing Modes

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

Problem-01:

Which of the following system bus is used to designate the source or destination of the data on the bus itself?

a) Control bus
b) Data bus
c) Address bus
d) System bus

Solution-

The correct option is (C) Address bus.

Address bus carries the source or destination address of data i.e. where to store or from where to retrieve the data.

Problem-02:

The bus which is used to transfer data from main memory to peripheral device is-

a) Data bus
b) Input bus
c) DMA bus
d) Output bus

Solution-

The correct option is (A) Data bus.

Data bus carries data / instruction from CPU to memory/IO and vice-versa.

Problem-03:

How many memory locations a system with a 32-bit address bus can address?

a) 2^8
b) 2^{16}
c) 2^{32}
d) 2^{64}

Solution-

The correct option is (C) 2^{32}.

2^{32} memory locations can be addressed by a 32-bit address bus.

Problem-04:

How many bits can be transmitted at a time using a bus with 32 data lines?

a) 8 bits
b) 16 bits
c) 32 bits
d) 1024 bits

Solution-

Each line carries one bit. So, a bus with 32 data lines can transmit 32 bits at a time.

Problem-05:

A microprocessor has a data bus with 64 lines and an address bus with 32 lines. The maximum number of bits that can be stored in memory is-

a) 32×2^{12}
b) 32×2^{64}

c) 64×2^{32}
d) 64×2^{64}

Solution-
The correct option is (C) 64×2^{32}.
The amount of blocks that could be located is 2^{32}. Now, since data bus has 64 lines, so each block is 64 bits. Thus, maximum number of bits stored in memory is 2^{32} x 64 bits.

Problem-06:
The address bus with a ROM of size 1024 x 8 bits is-

a) 8 bits
b) 10 bits
c) 12 bits
d) 16 bits

Solution-
The correct option is (B) 10 bits.

The size of the ROM is $1024 \times 8 = 2^{10} \times 8$. Here, 10 indicates the address bus and 8 indicates the data bus width.

Problem-07:
The data bus width of a ROM of size 2048 x 8 bits is-

a) 8
b) 10
c) 12
d) 16

Solution-
The correct option is (A) 8.

The size of the ROM is $2048 \times 8 = 2^{11} \times 8$. Here, 11 indicate the address bus and 8 indicates the data bus width.

Problem-01:
The most appropriate matching for the following pairs is-

Column-1:
X: Indirect addressing

Y: Immediate addressing

Z: Auto decrement addressing

Column-2:
1. Loops

2. Pointers

3. Constants

a) X-3, Y-2, Z-1
b) X-1, Y-3, Z-2
c) X-2, Y-3, Z-1
d) X-3, Y-1, Z-2

Solution-
Option (C) is correct.

Problem-02:

In the absolute addressing mode,

a) The operand is inside the instruction
b) The address of the operand is inside the instruction
c) The register containing the address of the operand is specified inside the instruction
d) The location of the operand is implicit

Solution-
Option (B) is correct.

Problem-03:

Which of the following addressing modes are suitable for program relocation at run time?

1. Absolute addressing
2. Base addressing
3. Relative addressing
4. Indirect addressing
    a) 1 and 4
    b) 1 and 2
    c) 2 and 3
    d) 1, 2 and 4

Solution-

Option (C) is correct.

Problem-04:

What is the most appropriate match for the items in the first column with the items in the second column-

Column-1:

X: Indirect addressing

Y: Indexed addressing

Z: Base register addressing

Column-2:

1. Array implementation

2. Writing relocatable code

3. Passing array as parameter

    a) X-3, Y-1, Z-2
    b) X-2, Y-3, Z-1
    c) X-3, Y-2, Z-1
    d) X-1, Y-3, Z-2

Solution-

Option (A) is correct.

Problem-05:

Which of the following addressing modes permits relocation without any change whatsoever in the code?

    a) Indirect addressing
    b) Indexed addressing
    c) Base register addressing
    d) PC relative addressing

Solution-

Option (C) is correct.

Problem-08: important

The memory locations 1000, 1001 and 1020 have data values 18, 1 and 16 respectively before the following program is executed.

MOVI	R_s, 1	Move immediate
LOAD	R_d, 1000(R_s)	Load from memory
ADDI	R_d, 1000	Add immediate
STOREI	0(R_d), 20	Store immediate

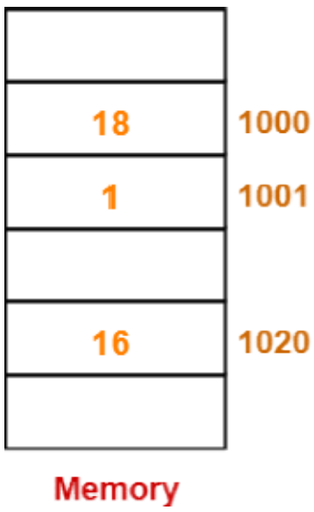
Which of the statements below is TRUE after the program is executed?

    a) Memory location 1000 has value 20
    b) Memory location 1020 has value 20
    c) Memory location 1021 has value 20

d) Memory location 1001 has value 20

Solution-

Before the execution of program, the memory is-



Now, let us execute the program instructions one by one-

Instruction-01: MOVI R_s, 1

- ✓ This instruction uses immediate addressing mode.
- ✓ The instruction is interpreted as Rs ← 1.
- ✓ Thus, value = 1 is moved to the register R_s.

Instruction-02: LOAD R_d, 1000(R_s)

- ✓ This instruction uses displacement addressing mode.
- ✓ The instruction is interpreted as $R_d \leftarrow [1000 + [R_s]]$.
- ✓ Value of the operand = $[1000 + [R_s]] = [1000 + 1] = [1001] = 1$.
- ✓ Thus, value = 1 is moved to the register R_d.

Instruction-03: ADDI R_d, 1000

- ✓ This instruction uses immediate addressing mode.
- ✓ The instruction is interpreted as $R_d \leftarrow [R_d] + 1000$.
- ✓ Value of the operand = $[R_d] + 1000 = 1 + 1000 = 1001$.
- ✓ Thus, value = 1001 is moved to the register R_d.

Instruction-04: STOREI 0(R_d), 20

- ✓ This instruction uses displacement addressing mode.
- ✓ The instruction is interpreted as $0 + [R_d] \leftarrow 20$.
- ✓ Value of the destination address = $0 + [R_d] = 0 + 1001 = 1001$.
- ✓ Thus, value = 20 is moved to the memory location 1001.

Thus,

- ✓ After the program execution is completed, memory location 1001 has value 20.
- ✓ Option (D) is correct.