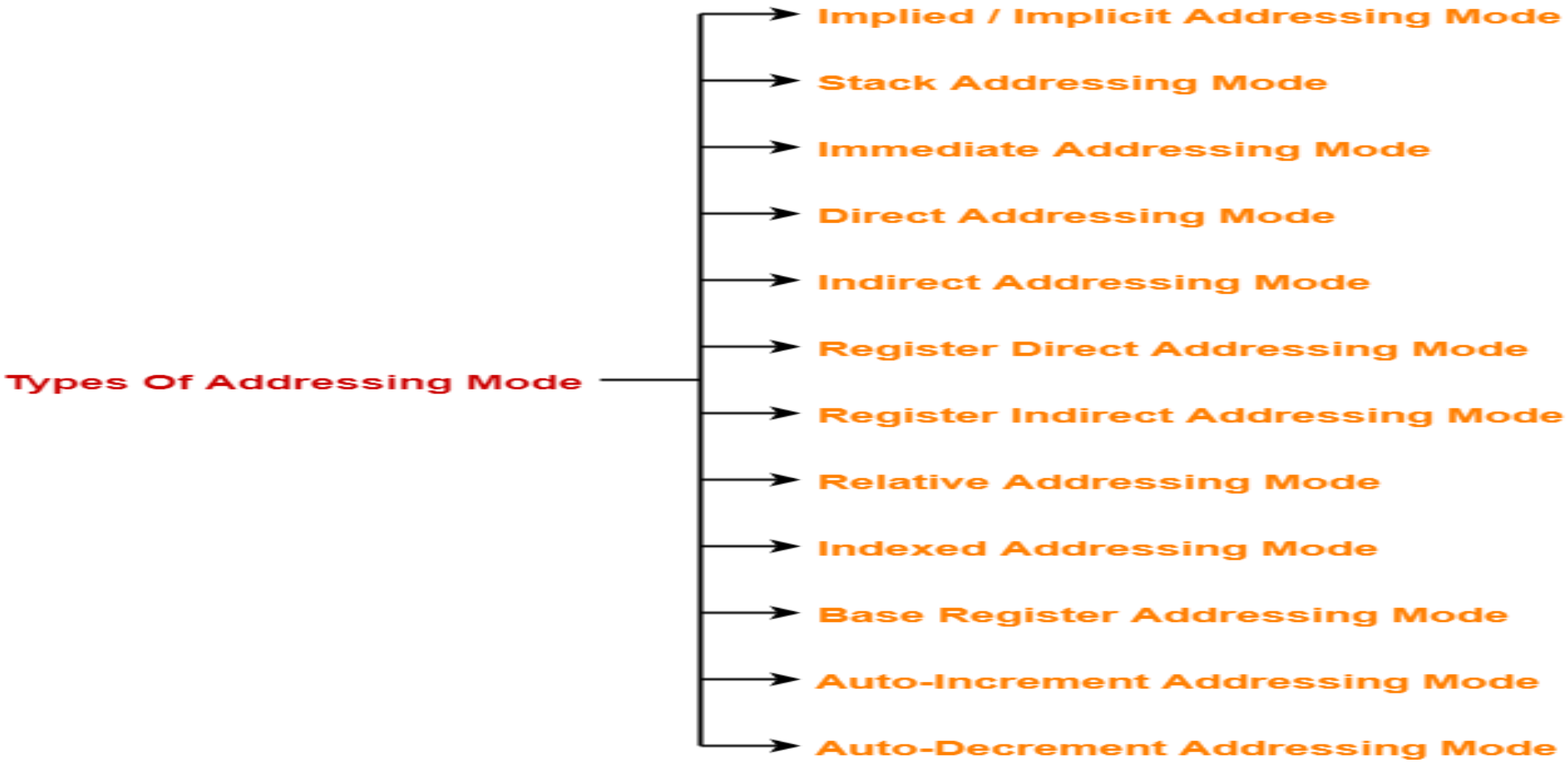
ADDRESSING MODES

The different ways of specifying the location of an operand in an instruction are called as **addressing modes**.

Types of Addressing Modes



Implied Addressing Mode

- The definition of the instruction itself specify the operands implicitly.
- It is also called as **implicit addressing mode**.
- 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)

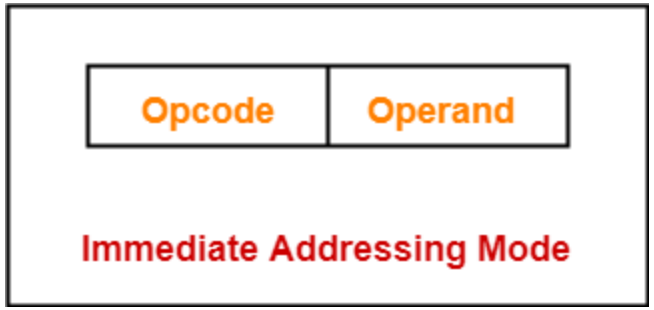
Stack Addressing Mode

- 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.

Immediate Addressing Mode-

In this addressing mode,

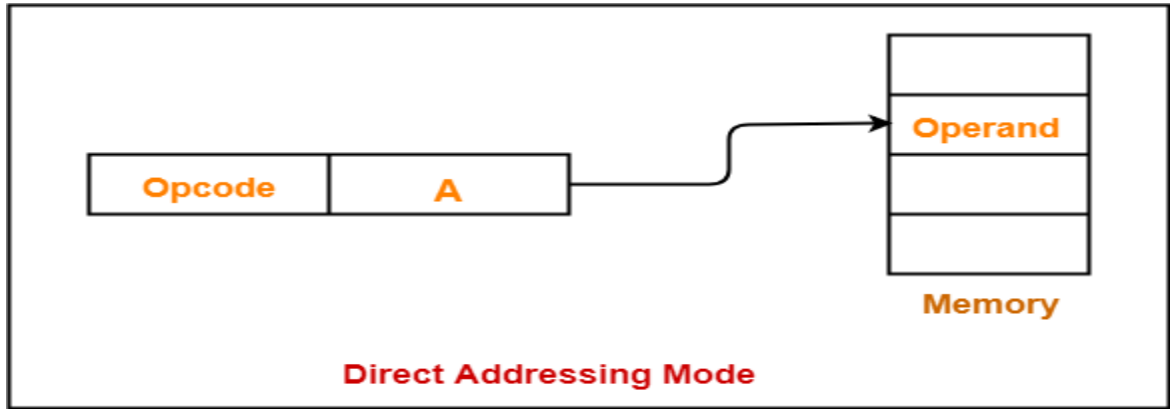
- The operand is specified in the instruction explicitly.
- Instead of address field, an operand field is present that contains the operand.



- **Examples**
- ADD 10 will increment the value stored in the accumulator by 10.
- MOV R #20 initializes register R to a constant value 20.

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**.



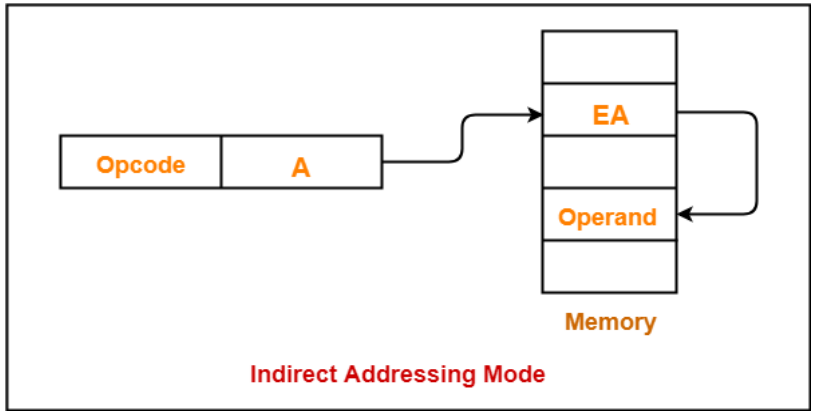
Example-

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

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

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.

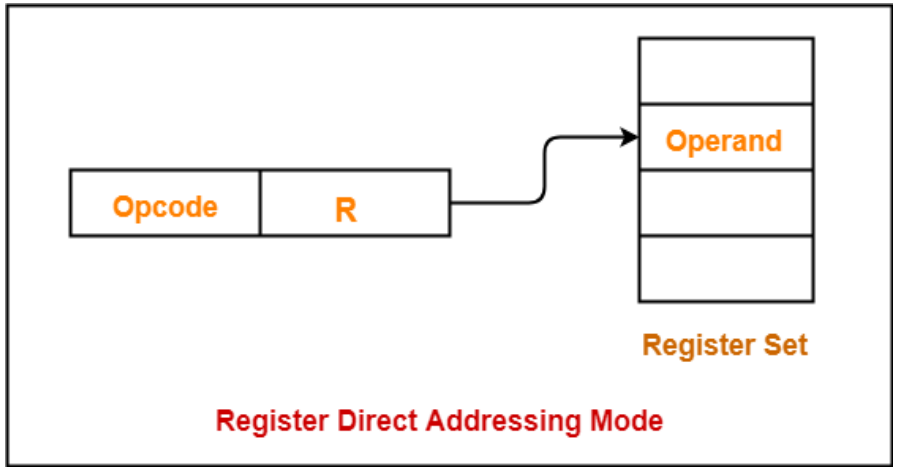


Indirect Addressing Mode

- **Example**
- ADD X will increment the value stored in the accumulator by the value stored at memory location specified by X.
  - AC ← AC + [[X]]

Register Direct Addressing Mode

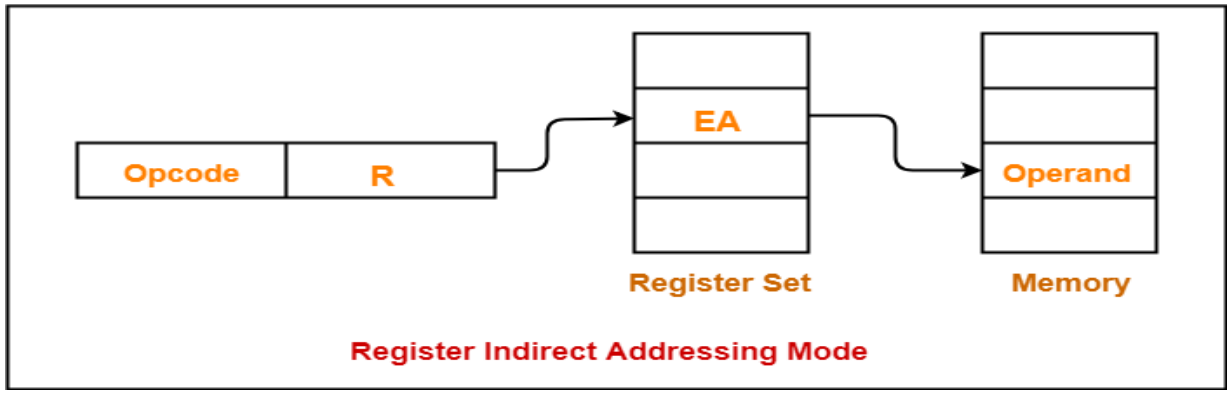
- In this addressing mode,
- The operand is contained in a register set.
- The address field of the instruction refers to a CPU register that contains the operand.
- No reference to memory is required to fetch the operand.



- **Example-**
- ADD R will increment the value stored in the accumulator by the content of register R. AC ← AC + [R]
- 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.

Register Indirect Addressing Mode

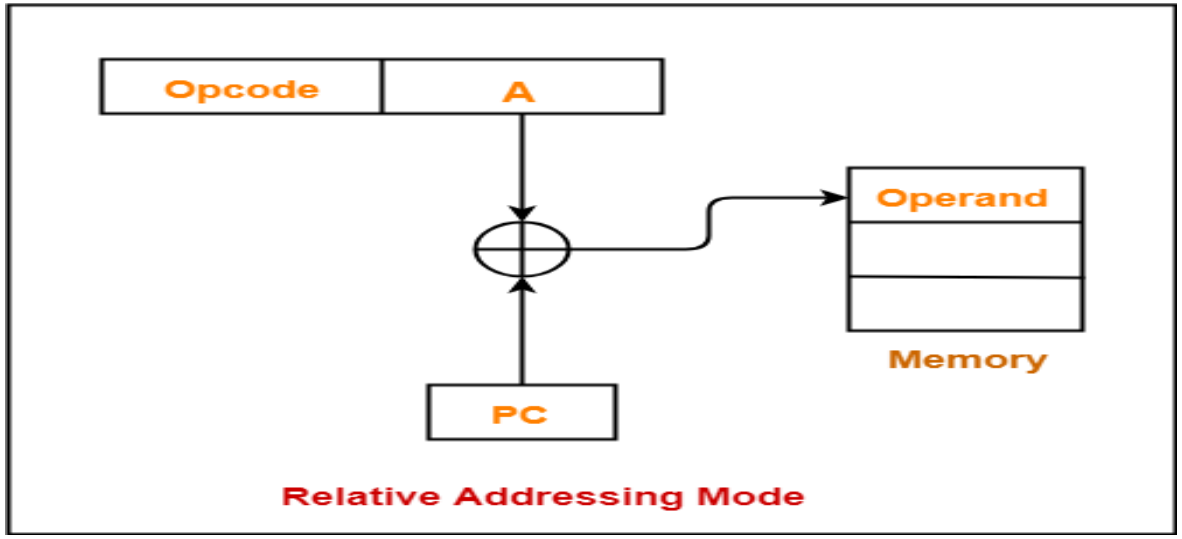
- The address field of the instruction refers to a CPU register that contains the effective address of the operand.
- Only one reference to memory is required to fetch the operand.



- ADD R will increment the value stored in the accumulator by the content of memory location specified in register R. AC ← AC + [[R]]
- This addressing mode is similar to indirect addressing mode.
- The only difference is address field of the instruction refers to a CPU register.

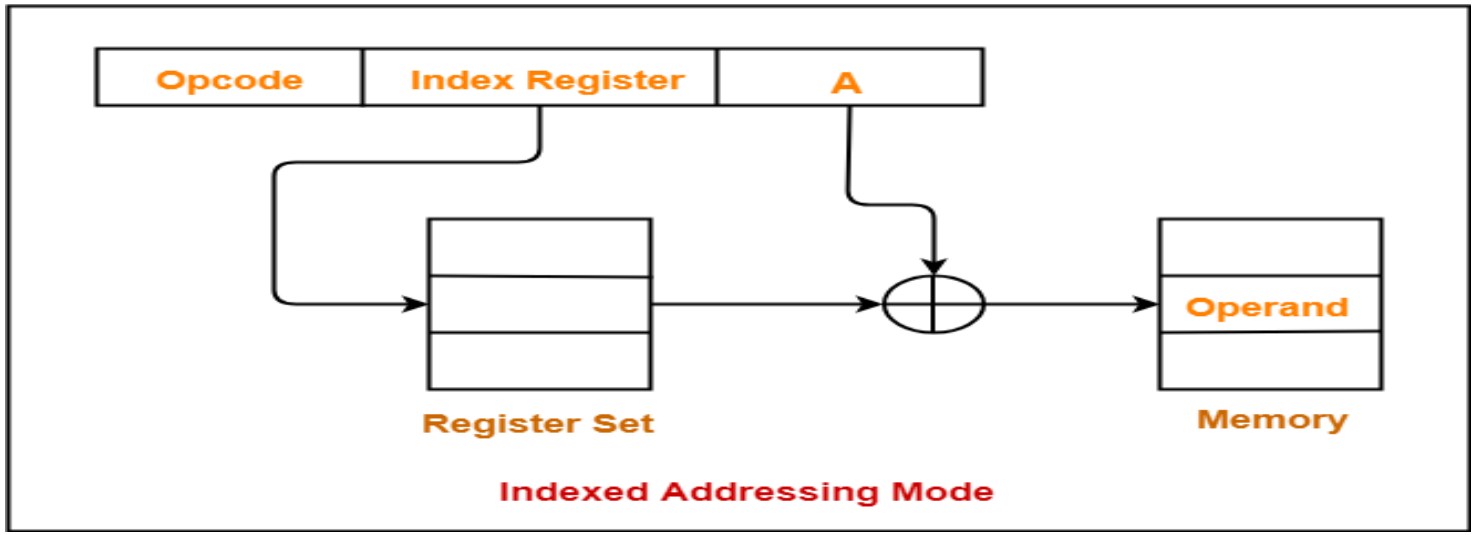
Relative Addressing Mode

- 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**
- **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.



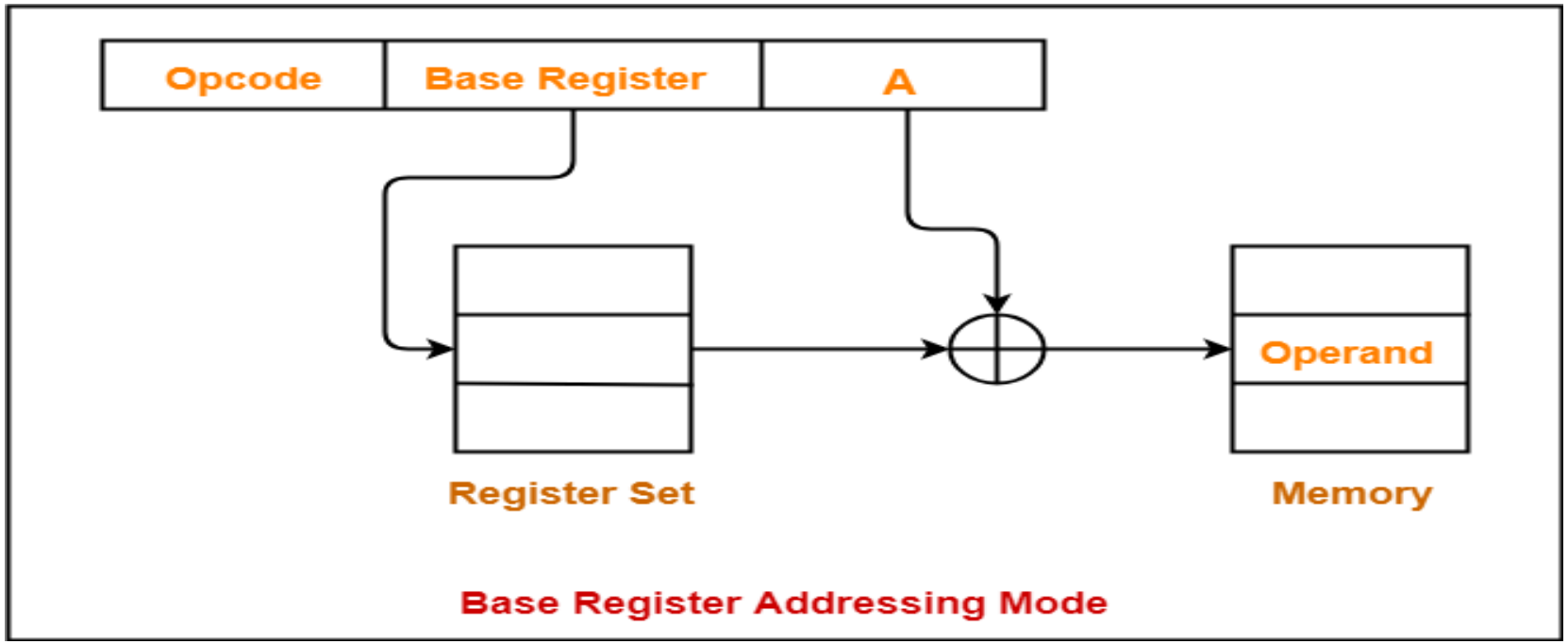
Indexed Addressing Mode

- 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**



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**



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.

Assume operand size = 2 bytes.

Here,

After fetching the operand 6B, the instruction register R_{AUTO} will be automatically incremented by 2.

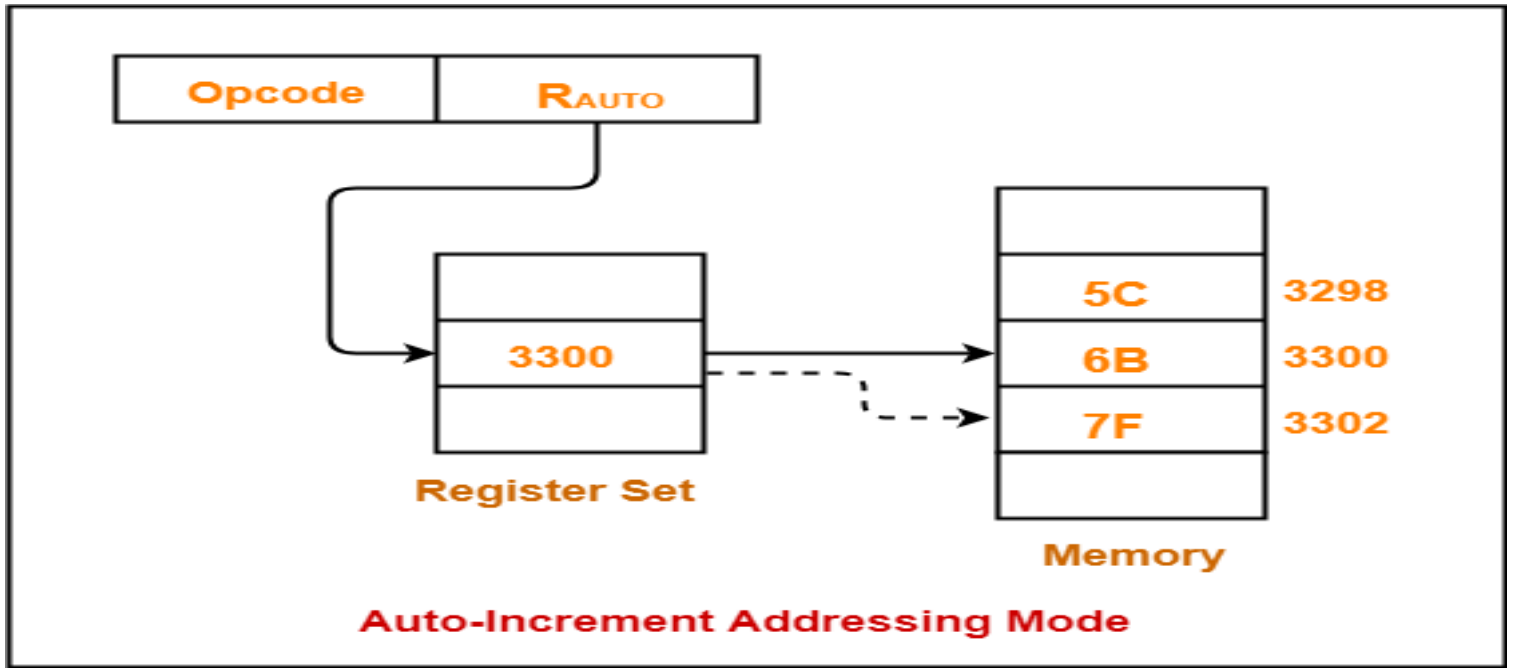
Then, updated value of R_{AUTO} will be 3300 + 2 = 3302.

At memory address 3302, the next operand will be found.

In auto-increment addressing mode,

First, the operand value is fetched.

Then, the instruction register R_{AUTO} value is incremented by step size 'd'.



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.

Assume operand size = 2 bytes.

Here,

First, the instruction register R_{AUTO} will be decremented by 2.

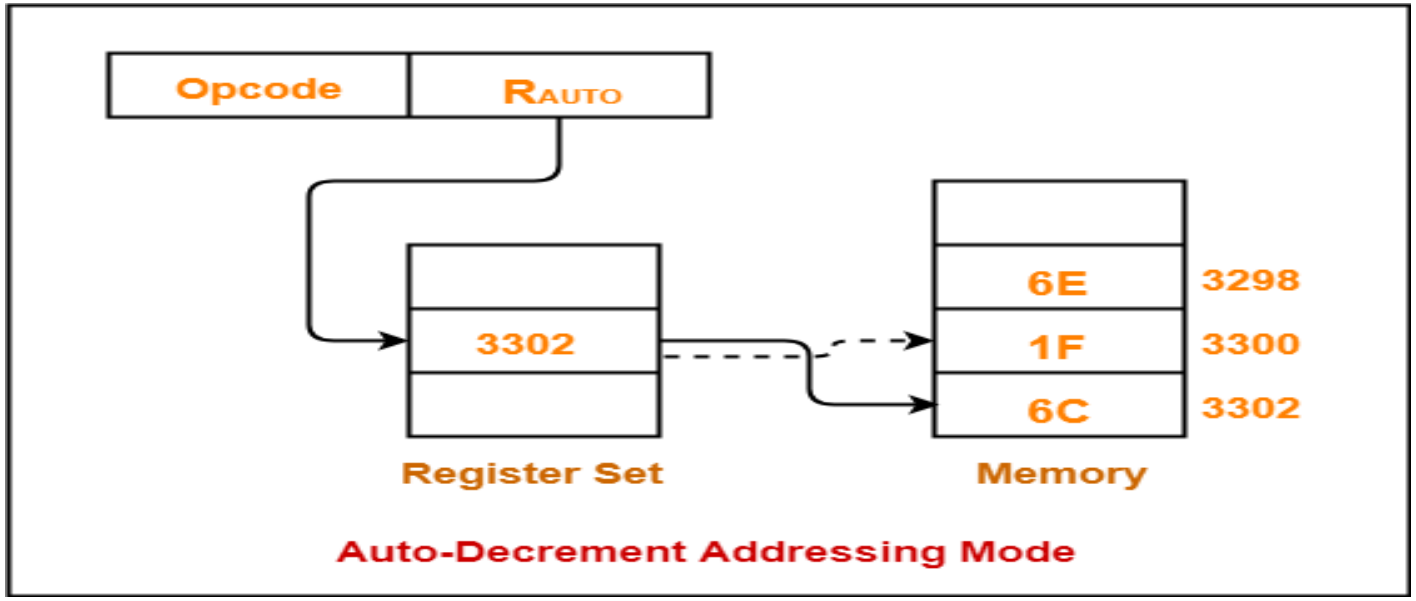
Then, updated value of R_{AUTO} will be 3302 – 2 = 3300.

At memory address 3300, the operand will be found.

In auto-decrement addressing mode,

First, the instruction register R_{AUTO} value is decremented by step size ‘d’.

Then, the operand value is fetched.



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

Applications of Addressing Modes

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

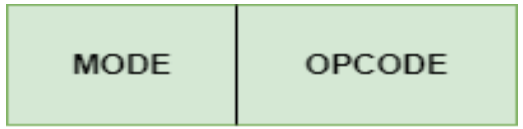
Instruction Formats

- The instruction format also defines the layout of the bits for an instruction. It can be of variable lengths with multiple numbers of addresses. These address fields in the instruction format vary as per the organization of the registers in the CPU. Depending on the multiple address fields, the instruction is categorized as follows:

- Three address instruction
- Two address instruction
- One address instruction
- Zero address instruction

Zero Address Instruction

- This instruction does not have an operand field, and the location of operands is implicitly represented. The stack-organized computer system supports these instructions. To evaluate the arithmetic expression, it is required to convert it into reverse polish notation.



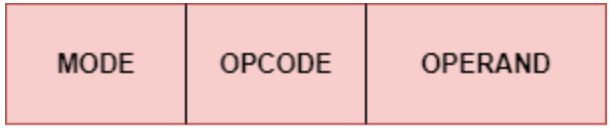
- **Example:** Consider the below operations, which shows how X = (A + B) * (C + D) expression will be written for a stack-organized computer.

TOS: Top of the Stack

PUSH	A	TOS ← A
PUSH	B	TOS ← B
ADD		TOS ← (A + B)
PUSH	C	TOS ← C
PUSH	D	TOS ← D
ADD		TOS ← (C + D)
MUL		TOS ← (C + D) * (A + B)
POP	X	M [X] ← TOS

One Address Instruction

- This instruction uses an implied accumulator for data manipulation operations. An accumulator is a register used by the CPU to perform logical operations. In one address instruction, the accumulator is implied, and hence, it does not require an explicit reference. For multiplication and division, there is a need for a second register. However, here we will neglect the second register and assume that the accumulator contains the result of all the operations.



- **Example:** The program to evaluate X = (A + B) ∗ (C + D) is as follows:

LOAD	A	AC ← M [A]
ADD	B	AC ← A [C] + M [B]
STORE	T	M [T] ← AC
LOAD	C	AC ← M [C]
ADD	D	AC ← AC + M [D]
MUL	T	AC ← AC * M [T]
STORE	X	M [X] ← AC

- All operations are done between the accumulator(AC) register and a memory operand.
- M[ ] is any memory location.
- M[T] addresses a temporary memory location for storing the intermediate result.
- This instruction format has only one operand field. This address field uses two special instructions to perform data transfer, namely:
- LOAD: This is used to transfer the data to the accumulator.
- STORE: This is used to move the data from the accumulator to the memory.

Two Address Instructions

- This instruction is most commonly used in commercial computers. This address instruction format has three operand fields. The two address fields can either be memory addresses or registers.

- **Example:** The program to evaluate X = (A + B) ∗ (C + D) is as follows:
- The MOV instruction transfers the operands to the memory from the processor registers. R1, R2 registers.

MOV	R1, A	R1 ← M [A]
ADD	R1, B	R1 ← R1 + M [B]
MOV	R2, C	R2 ← M [C]
ADD	R2, D	R2 ← R2 + M [D]
MUL	R1, R2	R1 ← R1∗R2
MOV	X, R1	M [X] ← R1

Three Address Instruction

- The format of a three address instruction requires three operand fields. These three fields can be either memory addresses or registers.

- **Example:** The program in assembly language X = (A + B) ∗ (C + D) Consider the instructions given below that explain each instruction's register transfer operation.
- Two processor registers, R1 and R2.
- The symbol M [A] denotes the operand at memory address symbolized by A. The operand1 and operand2 contain the data or address that the CPU will operate. Operand 3 contains the result's address.

ADD	R1, A, B	R1 ← M [A] + M [B]
ADD	R2, C, D	R2 ← M [C] + M [D]
MUL	X, R1, R2	M [X] ← R1 ∗ R2

References

- https://www.gatevidyalay.com/addressing-modes/
- https://www.codingninjas.com/codestudio/library/instruction-formats