Computer Organization - UNIT 2

Lets us First Understand about what is Opcode -

Every Program in computer weather it is system program like Operating system or Application or user program like a simple c program

Every program ultimately gets converted into machine code or binary form
Now the thing is how processor executes all such program ?

So for basic understanding high level language program gets converted into assembly code or some intermediate code by compiler or interpreter then it is gets converted into executable binary code or Machine code. Machine codes are made up of multiple **machine instructions** every Machine instruction has an Opcode and Operands. For example

ADD R1, R2

add value of R1 register with value of R2 register and result will be transferred from accumulator register to R1 register. (As every ALU result first get stored in accumulator)

OPCODES

An **opcode** is a single instruction that can be executed by the CPU.. In assembly language mnemonic form an **opcode** is a command such as MOV or ADD or JMP. For example.

C code : -

```
For ( int x = 0;   x<=3;   x++)
{
      //Do something!
}
```

Equivalent assembly code:

```
          xor    cx,   cx    ;          cx-register is the counter, set to 0
loop1     //opeartion    ;              Whatever you wanna do goes here, should not change cx
            inc cx       ;                     Increment
            cmp cx,3     ;                Compare cx to the limit
```

```
        jle  loop1      ;                    Loop  while  less  or  equal
```

opcpde tells the computer to do something. Each **machine language instruction** typically has both an opcode and operands.

An example here is
CMP cx , 3

// in this statement **cmp** is opcode, content of cx (it is a register) is operand and will be compared with constant 3 . CMP will return True (which can be a non zero in binary) or False (0 in binary)based on the content of CX is 3 or not.

The opcode uniquely specifies the operation to be performed

***Opcode size** - It is the number of bits occupied by the opcode which is calculated by taking log of instruction set size.

***Operand size** - It is the number of bits occupied by the operand.

***Instruction size** - It is calculated as sum of bits occupied by opcode and operands.

Now this instruction register can store one instruction at a time and processor will execute these.

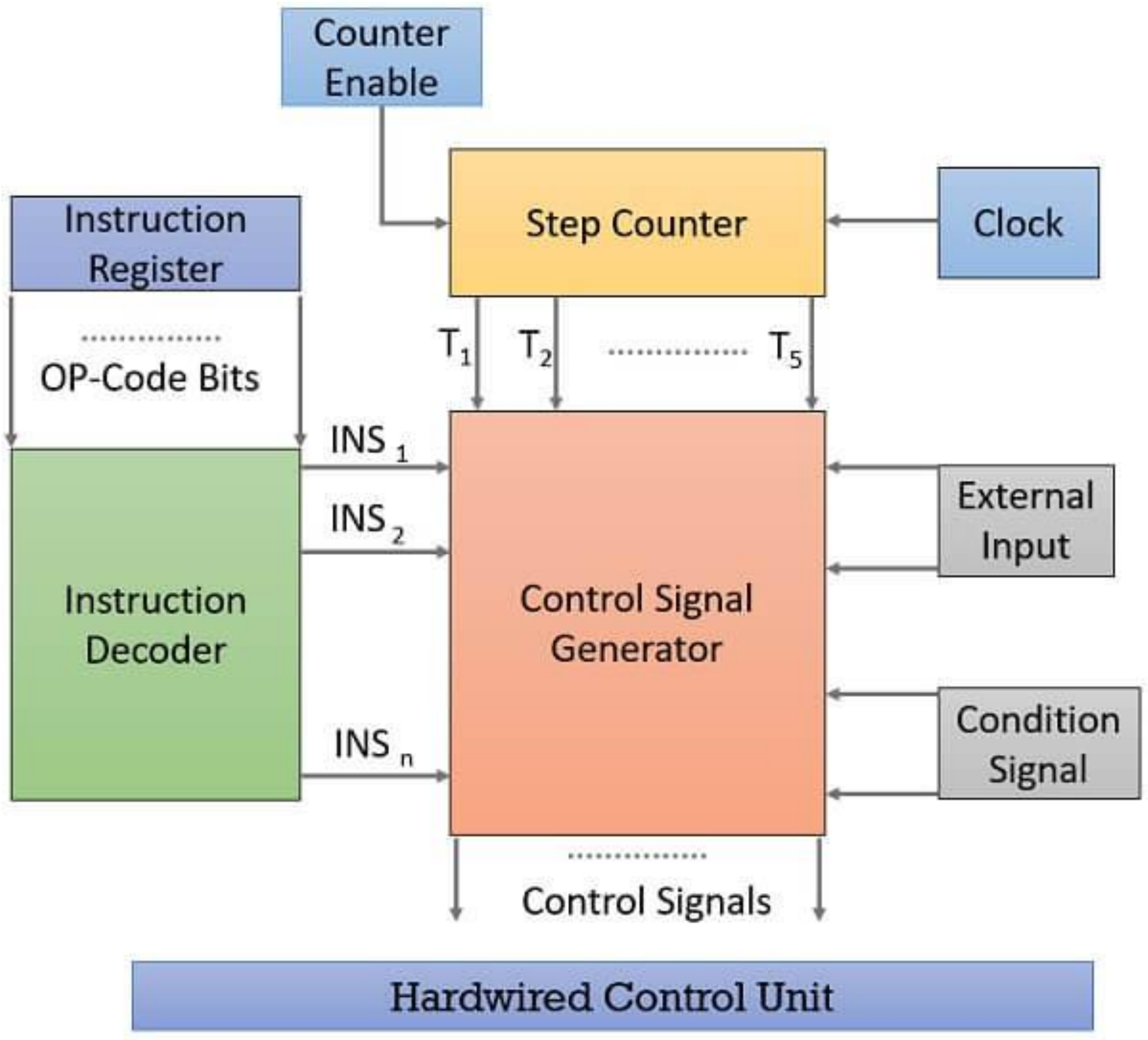
Now how does execution happens

1. There is Program counter register which holds address of instruction to be executed by the processor.
2. The address is given to Address Register which asks the RAM to transfer content which is actually a machine instruction stored at the given address. And at the same time Program counter register is incremented and it starts pointing to next instruction address.
3. When RAM transfer the instruction , it get stored in another register called as Instruction register.
4. Control Unit then decodes what is the opcode or what operation need to be done with the operands.
5. Then after fetching operands processor starts execution of the instruction.

What is Hardwired Control Unit?

In simple words, the hardwired control unit generates the **control signals** to execute the instructions in a proper sequence and at the correct time. The hardwired control unit is created with the hardware; it is a **circuitry** approach..

A hardwired circuit organization is shown in the figure below. Let us discuss all the components one by one, in order to understand the “generation of control signals” from this circuitry organization.



- The **instruction register** is a processors register that has the ‘instruction’ which is currently in execution. The instruction register generates the **OP-code bits** respective of the **operation** and the **addressing modes** of the operands, mentioned in the instruction.
- **Instruction decoder** receives the Op-code bits generated by the instruction register and **interprets** the operation and addressing modes of the instruction. Now, based

on operation and addressing mode of the instruction in instruction register it set the corresponding **Instruction signal** INS_i to **1**.
Each instruction is executed in step-like, **instruction fetch**, **decode**, **operand fetch**, **ALU**, **memory store**. These steps may vary in different books. But in general, five steps are enough to for the execution of an instruction.

- Now, the control unit must be aware of the current step, the instruction is in. For this, a **Step Counter** is implemented which has signals from T1, …, T5. The step counter sets one of the signals T1 to T5 to 1 on the basis of the step, the instruction is in.
- Here, the question arises how step counter knows the current step of the instruction? For this, a **Clock** is implemented. This clock is designed such that for each step the clock must complete its one clock cycle.
  So, consider if the step counter has set T3 signal 1 then after a clock cycle completes step counter will set T4 to 1.
- What if the execution of instruction has is interrupted due to some reason? Will the clock still continue to trigger step counter? The answer is No. The **Counter Enable** 'disables' the step counter to increment to the next step signal, till the execution of the current step is completed.
- Now, suppose the execution of an instruction depends on some condition or if it is branch instruction. This is determined with the help of the **Condition signals**. The Condition signals generate the signals for the conditions greater than, less than, equal, greater than equal, less than equal etc.
- The remaining is **External inputs**, it acknowledges the control signal generator of **interrupts** which affects the execution of the instruction.

On an, all the Control Signal Generator generates the control signals, based on the inputs obtained by the Instruction register, Step counter, Condition signals and External inputs.

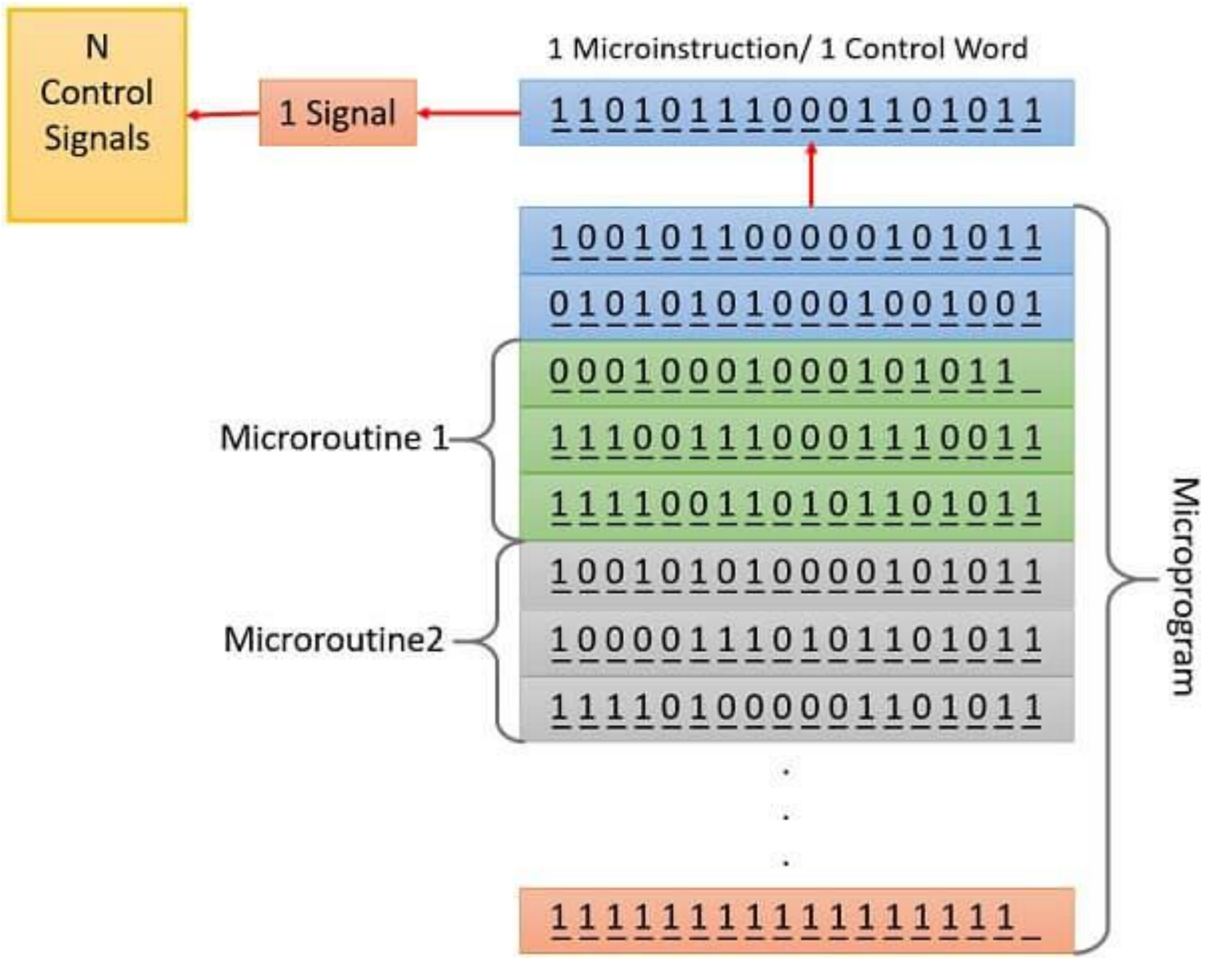
What is Microprogrammed Control Unit?

Microprogrammed **control unit** also produces the control signal but using the **programs**. The program that creates the 'control signals' is called **Microprogram**. This microprogram is placed on the processor chip which is fast memory, it is also called **control memory** or **control store.**

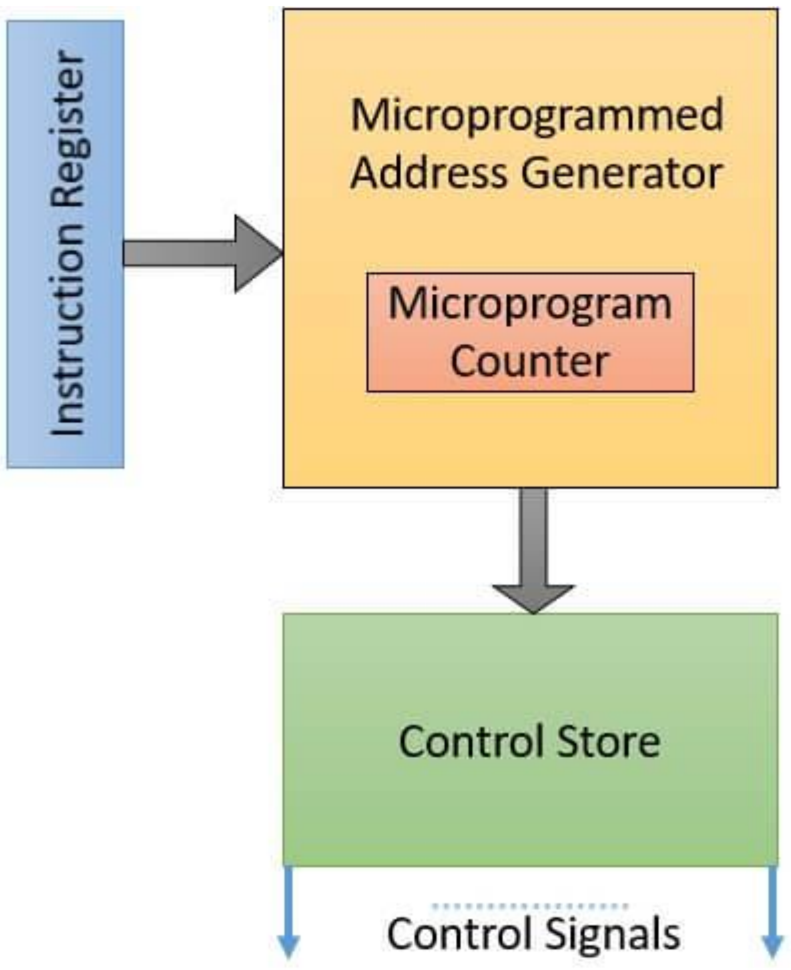
A microprogram has a set of **microinstructions**, or it is also termed as **control word**. Each microinstruction is 'n' bit word. Each control signal differs from other control signal depending on the bit pattern of the control word. Each control word/microinstruction has a different bit pattern.

Instruction execution is performed in steps as we have seen above. So, for each step there is a control word/ microinstruction in the microprogram. A sequence of microinstructions required to execute a particular instruction is called **microroutine**.

In the figure below, you can understand the organization of a **control word/ microinstruction, microroutine** and **microprogram.**



Now let us discuss the **organization** of the **microprogram control unit**. We will discuss the flow of instruction execution in terms of instruction execution steps.



Microprogrammed Control Unit Organization

1. In the **first step** (instruction fetch) the **Microinstruction address generator** would **fetch** the instruction from 'instruction register' (IR).
2. In the **second step**, the microinstruction address generator **decodes** the instruction obtained from IR and retrieves the **starting address** of the **microroutine** required to perform the corresponding operation mentioned in the instruction. It loads that starting address to **microprogram counter**.
3. In the **third step**, the 'control word' corresponding to the 'starting address' of 'microprogram counter' is read and as the execution proceeds, microprogram address generator will **increment** the value of microprogram counter to read the successive control words of the microroutine.
4. In the last microinstruction of a microroutine, there is a bit which we call **end bit**. When end bit is set to 1 it denotes successful execution of that microroutine.
5. After this, the microprogram address generator would return back to **Step 1** again to fetch a new instruction. And the cycle goes on.
6. Control Store or Control memory is a memory used store the microprograms. A microprogram control unit is simple to implement and **flexible** to modify but it is slower than Hardwired control unit.

Key Differences Between Hardwired and Microprogrammed Control Unit (Important for Exam)

- The hardwired control unit is implemented using a **hardware circuit** while a microprogrammed control unit is implemented by **programming**.
- A hardwired control unit is designed for **RISC** style instruction set. On the other hand, a microprogrammed control unit is for **CISC** style instruction set.
- Implementing modification in a microprogrammed control unit is **easier** as it is easy to change the code. But implementing modification in Hardwired control unit is **difficult** as changing the circuitry will add cost.
- The hardwired control unit executes **simple instructions** easily but it finds difficulty in executing complex instruction. The microprogrammed control unit finds, executing **complex instructions** easy.
- Building up a hardwired control unit requires the hardware circuit which is **costly**. Building up microprogrammed control unit is cheaper as it only requires coding.
- The hardwired control unit **does not require** a control memory as here; the control signal is generated by hardware. The microprogrammed control unit **requires** the control memory as the microprograms which are responsible for generating the control signals are stored in control memory.
- Execution is **fast** as everything is in the hardware. But, the microprogram control unit is **slow** as it has to access control memory for the execution of every instruction.

General Register Organization:

When we are using multiple general purpose registers, instead of single accumulator register, in the CPU Organization then this type of organization is known as General register based CPU Organization. In this type of organization, computer uses two or three address fields in their instruction format. Each address field may specify a general register or a memory word. If many CPU registers are available for heavily used variables and intermediate results, we can avoid memory references much of the time, thus vastly increasing program execution speed, and reducing program size.

For example:

```
MULT R1, R2, R3
```

This is an instruction of an arithmatic multiplication written in assembly language. It uses three address fields R1, R2 and R3. The meaning of this instruction is:

```
R1 <-- R2 * R3
```

This instruction also can be written using only two address fields as:

```
MULT R1, R2
```

In this instruction, the destination register is the same as one of the source registers. This means the operation

```
R1  <--  R1  *  R2
```

The use of large number of registers results in short program with limited instructions.

Some examples of General register based CPU Organization are **IBM 360 and PDP- 11**.

The advantages of General register based CPU organization –

- Efficiency of CPU increases as there are large number of registers are used in this organization.
- Less memory space is used to store the program since the instructions are written in compact way.

The disadvantages of General register based CPU organization –

- Care should be taken to avoid unnecessary usage of registers. Thus, compilers need to be more intelligent in this aspect.
- Since large number of registers are used, thus extra cost is required in this organization.

Stack Organization

Stack is a storage structure that stores information in such a way that the last item stored is the first item retrieved. It is based on the principle of LIFO (Last-in-first-out). The stack in digital computers is a group of memory locations with a register that holds the address of top of element. This register that holds the address of top of element of the stack is called ***Stack Pointer***.

Stack Operations

The two operations of a stack are:

1. **Push:** Inserts an item on top of stack.
2. **Pop:** Deletes an item from top of stack.

Implementation of Stack

In digital computers, stack can be implemented in two ways:

1. Register Stack
2. Memory Stack

Register Stack

A stack can be organized as a collection of finite number of registers that are used to store temporary information during the execution of a program. The stack pointer (SP) is a register that holds the address of top of element of the stack.

Memory Stack

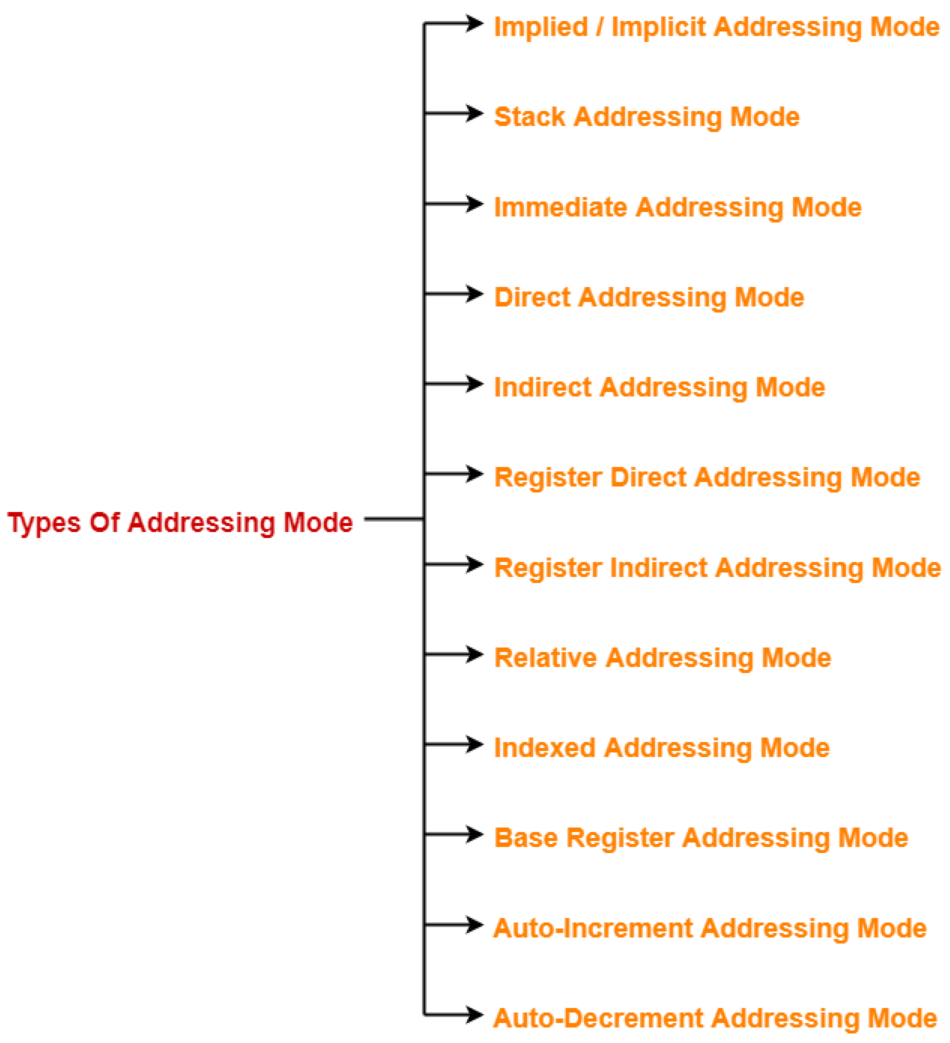
A stack can be implemented in a random access memory (RAM) attached to a CPU. The implementation of a stack in the CPU is done by assigning a portion of memory to a stack operation and using a processor register as a stack pointer. The starting memory location of the stack is specified by the processor register as *stack pointer*.

Addressing Modes-

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

Types of Addressing Modes-

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


Types Of Addressing Mode
Implied / Implicit Addressing Mode
Stack Addressing Mode
Immediate Addressing Mode
Direct Addressing Mode
Indirect Addressing Mode
Register Direct Addressing Mode
Register Indirect Addressing Mode
Relative Addressing Mode
Indexed Addressing Mode
Base Register Addressing Mode
Auto-Increment Addressing Mode
Auto-Decrement Addressing Mode

1. Implied / Implicit Addressing Mode
2. Stack Addressing Mode
3. Immediate Addressing Mode
4. Direct Addressing Mode
5. Indirect Addressing Mode
6. Register Direct Addressing Mode
7. Register Indirect Addressing Mode
8. Relative Addressing Mode
9. Indexed Addressing Mode
10. Base Register Addressing Mode
11. Auto-Increment Addressing Mode
12. Auto-Decrement Addressing Mode

1. Implied Addressing Mode-

In this addressing mode,

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

Examples-

- 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-

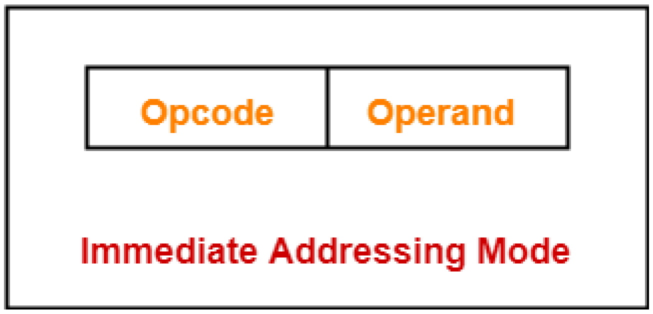
ADD

- 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-

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.



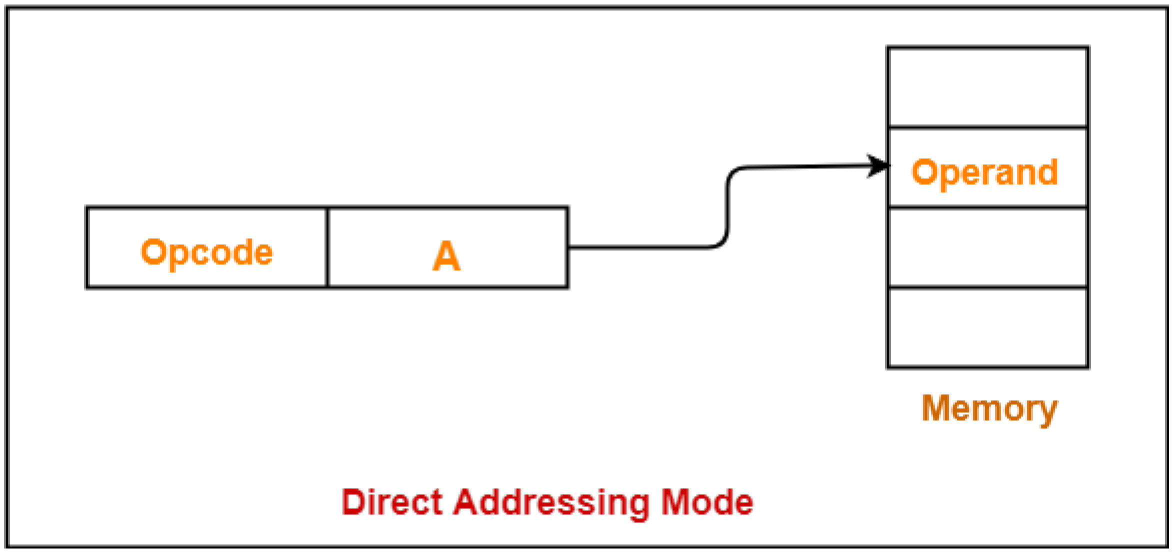
Examples-

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

4. Direct Addressing Mode-

In this 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-

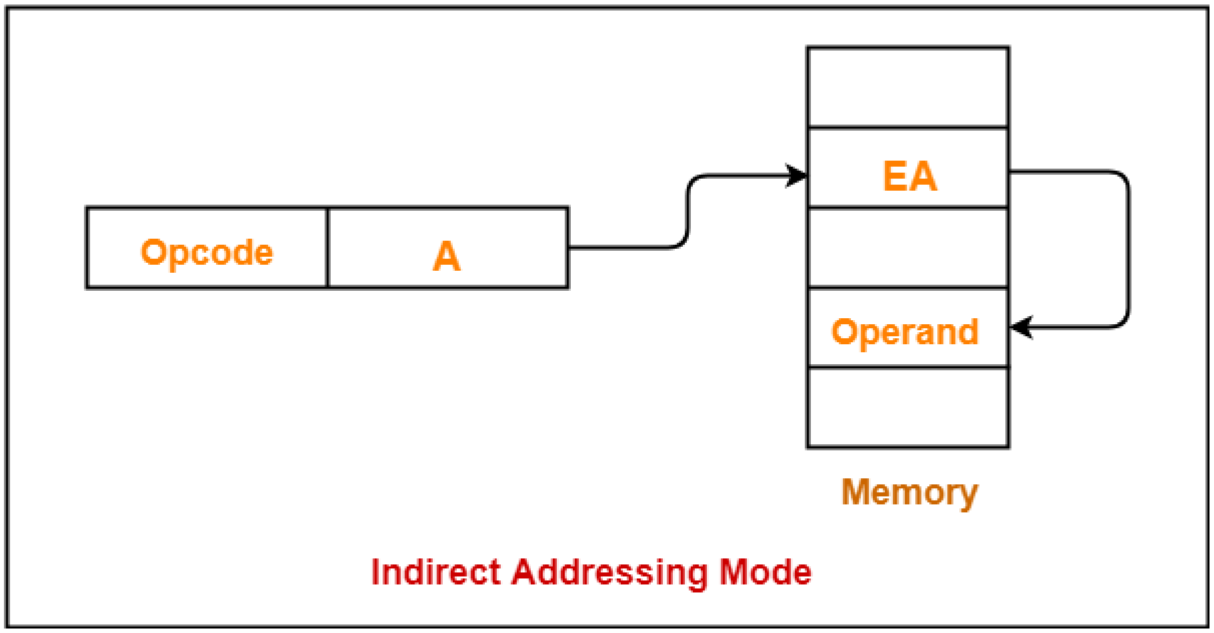
- ADD X will increment the value stored in the accumulator by the value stored at memory location X.

AC ← AC + [X]

5. Indirect Addressing Mode-

In this 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-

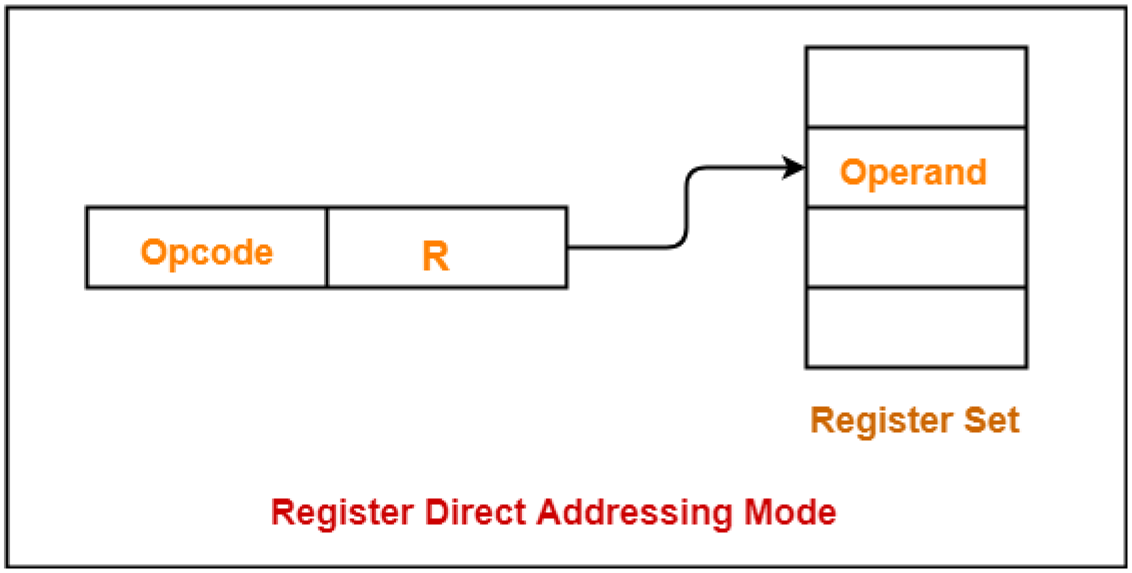
- ADD X will increment the value stored in the accumulator by the value stored at memory location specified by X.

AC ← AC + [[X]]

6. 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]

NOTE-

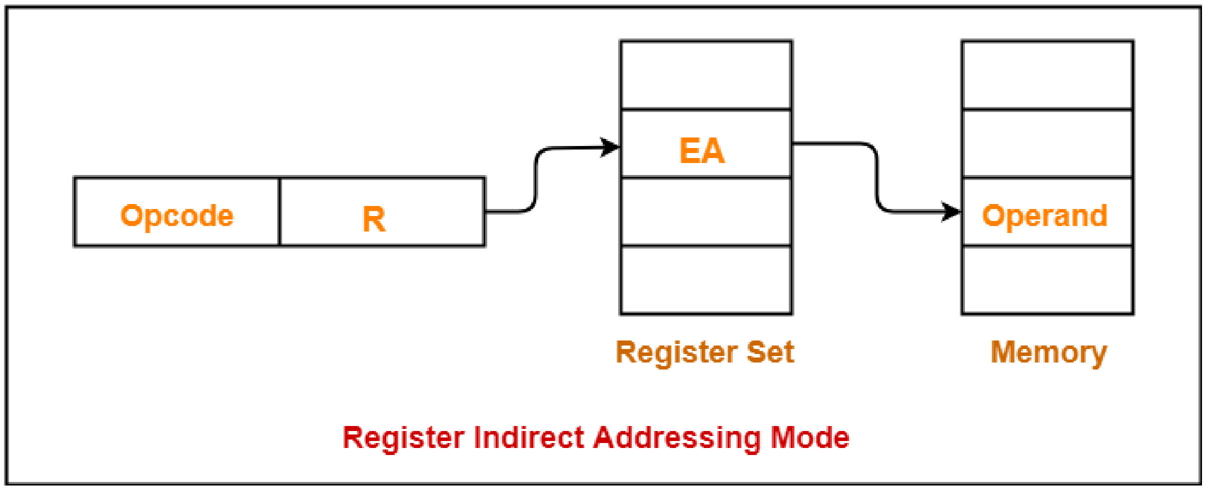
It is interesting to note-

- 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-

In this 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.

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

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.

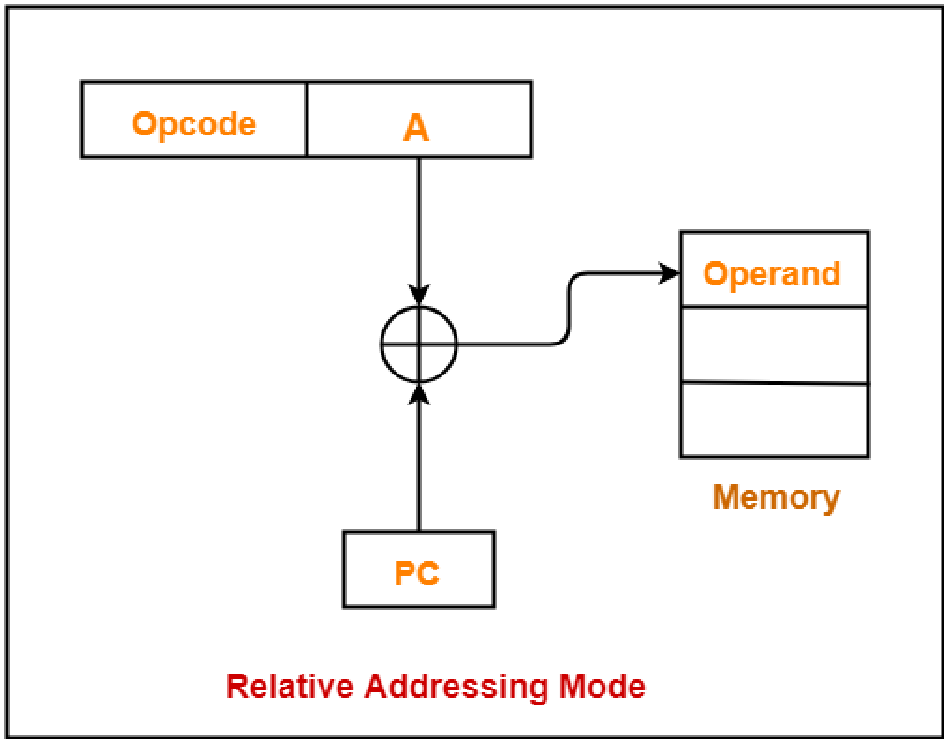
8. Relative Addressing Mode-

In this 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.

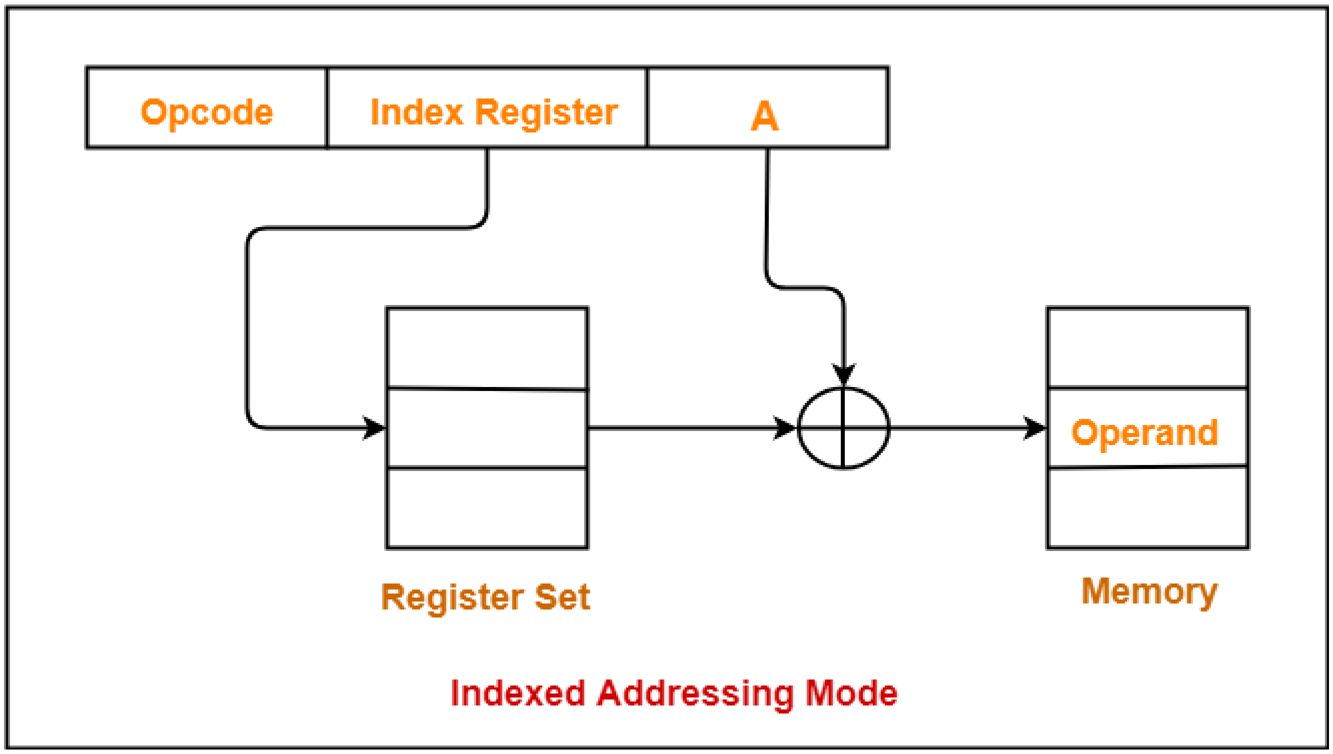
9. Indexed Addressing Mode-

In this 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



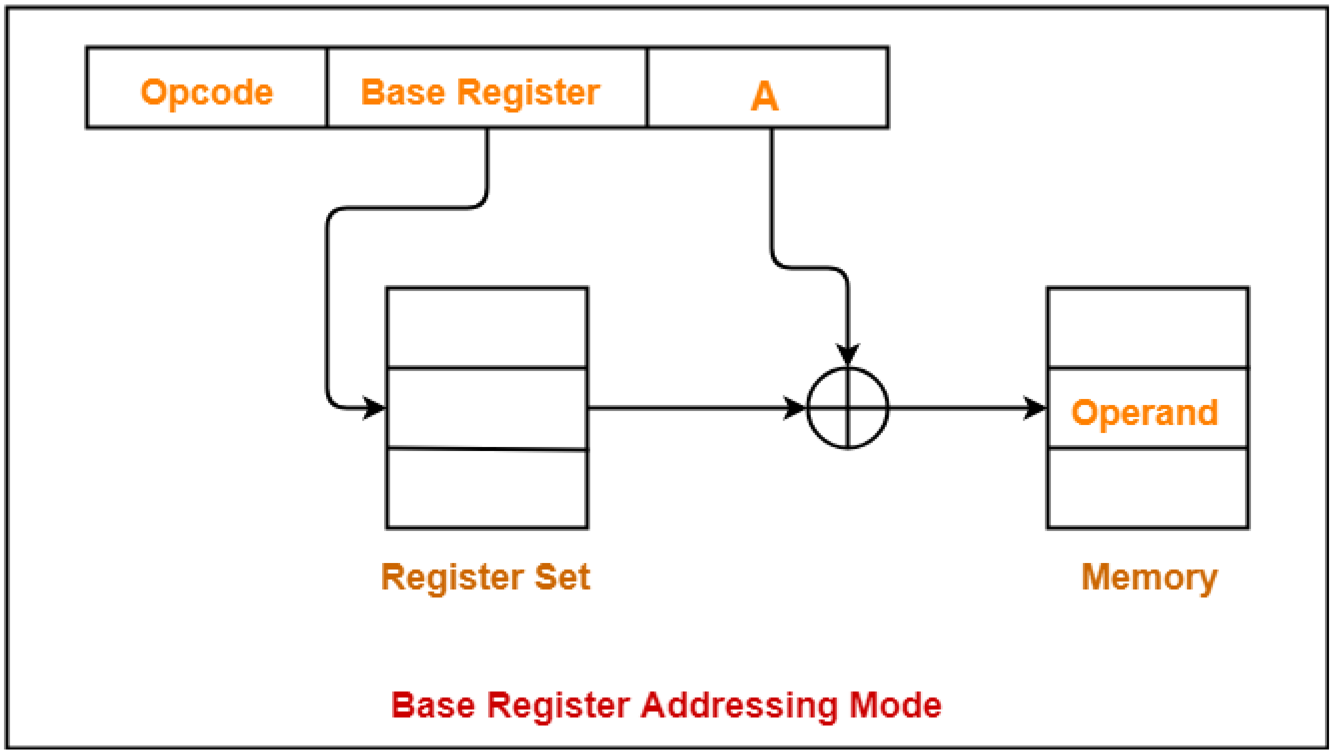
10. Base Register Addressing Mode-

In this 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-

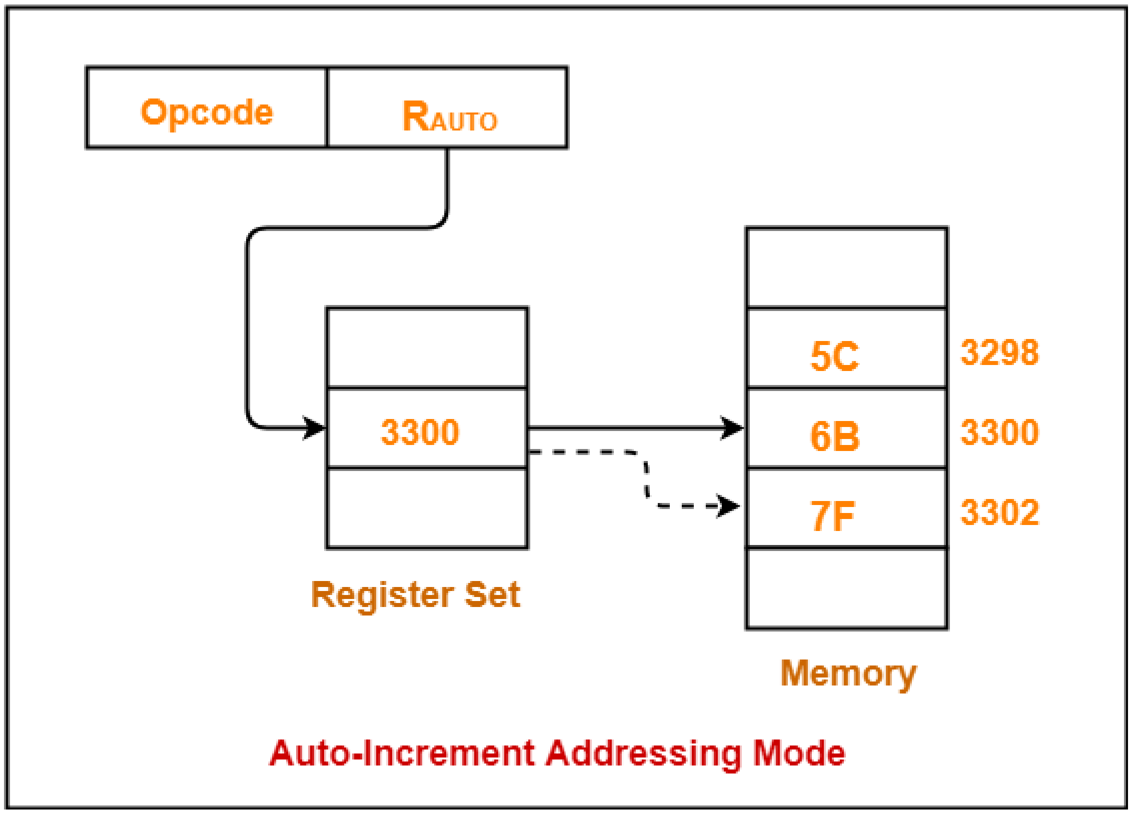
Effective Address of the Operand

= 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-

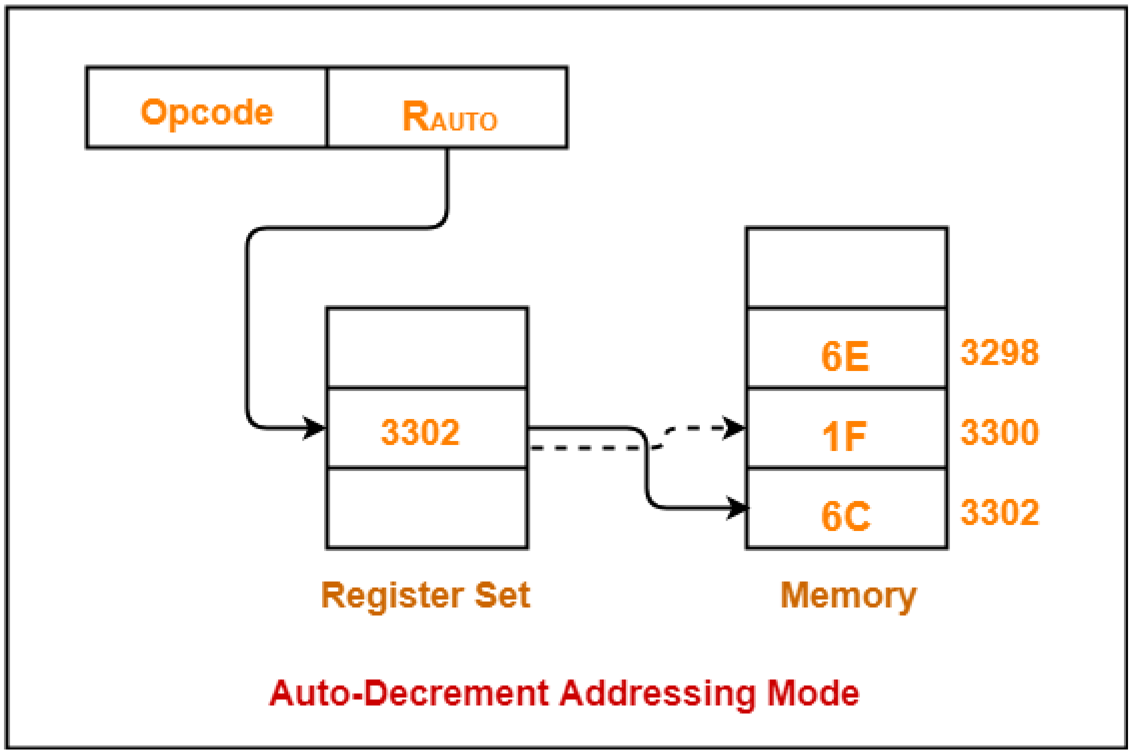
Effective Address of the Operand

= 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