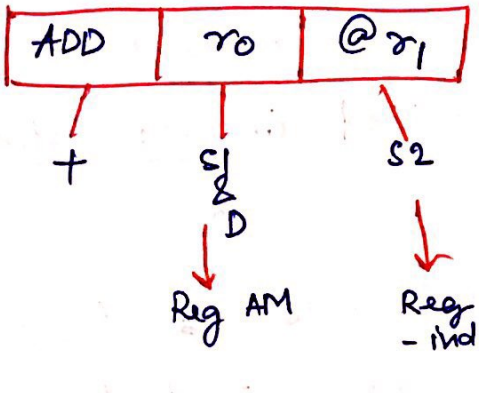



[reg_name]

this means reg contains some special info

$r_0 \leftarrow r_0 + M[[r_1]]$

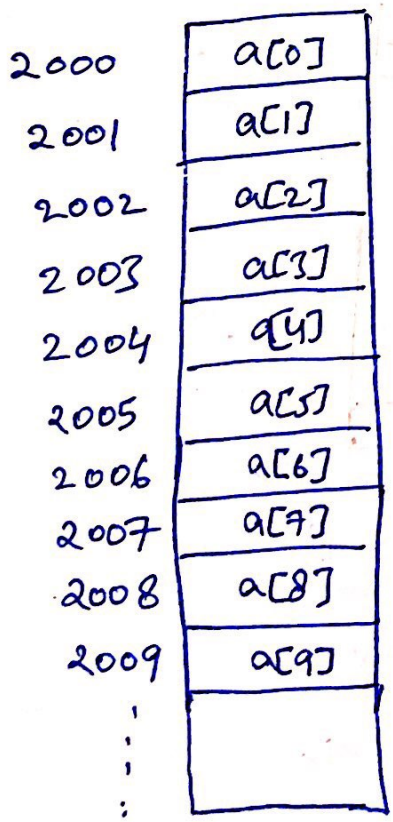
IRR IRR IRR + IMR

## 6. Index addressing Mode :-

**Analysis :-**

```
char a[10];
```

**Memory storage**




Base Address = 2000

Index values [0 to 9]

(a) Random accessing

eg- a[4]

MOV r0 #4

| MOV | r1 | 2000(r0) |
|---|---|---|

"r0 is indix reg"

$r_1 \leftarrow M[2000 + [r_0]]$ (4)

2004

eg:- a[8]

MOV r0 #8

| MOV | r1 | 2000(r0) |
|---|---|---|

$r_1 \leftarrow M[2000 + r_0]$ (8)

2008

(b) Linear accessing

"r0 is base reg"

① starting address onwards i.e a[0], a[1], .....

(Pre Inc)

MOV r0 #1FFF

| MOV | r1 | +(r0) |
|---|---|---|

$r_0 \leftarrow [r_0] + 1$ (1FFF)

2000

$r_1 \leftarrow M[[r_0]]$

2000

(Post Inc)

MOV r0 #2000

| MOV | r1 | (r0)+ |
|---|---|---|

$r_1 \leftarrow M[[r_0]]$

2000

$r_0 \leftarrow [r_0] + 1$

2000

2001

② Ending address onwards i.e a[9], a[8], ---- a[0]

(Pre Dec)

MOV r0 #200A

| MOV | $r_1$ | -(r0) |
|---|---|---|

$r_0 \leftarrow [r_0] - 1$ (200A → 2009)

$r_1 \leftarrow M[[r_0]]$ (2009)

(Post Dec)

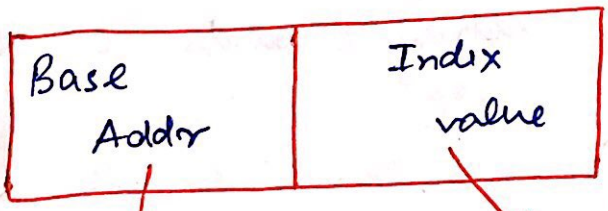
MOV r0 #2009

| MOV | $r_1$ | (r0)- |
|---|---|---|

$r_1 \leftarrow M[[r_0]]$ (2009)

$r_0 \leftarrow [r_0] - 1$ (2009 → 2008)

- Indix AM is used to access the random array elements from the memory. In this style of accessing, to parameters required. i.e (Base add + indix value)




starting addr of an array → Present in the Addr field of an Instⁿ

Position of an array element → Present in the Indix (Ri)

In this mode EA is obtained by adding the constant value to content of an index register i.e

$$EA = \text{Addr field value} + [R_i]$$

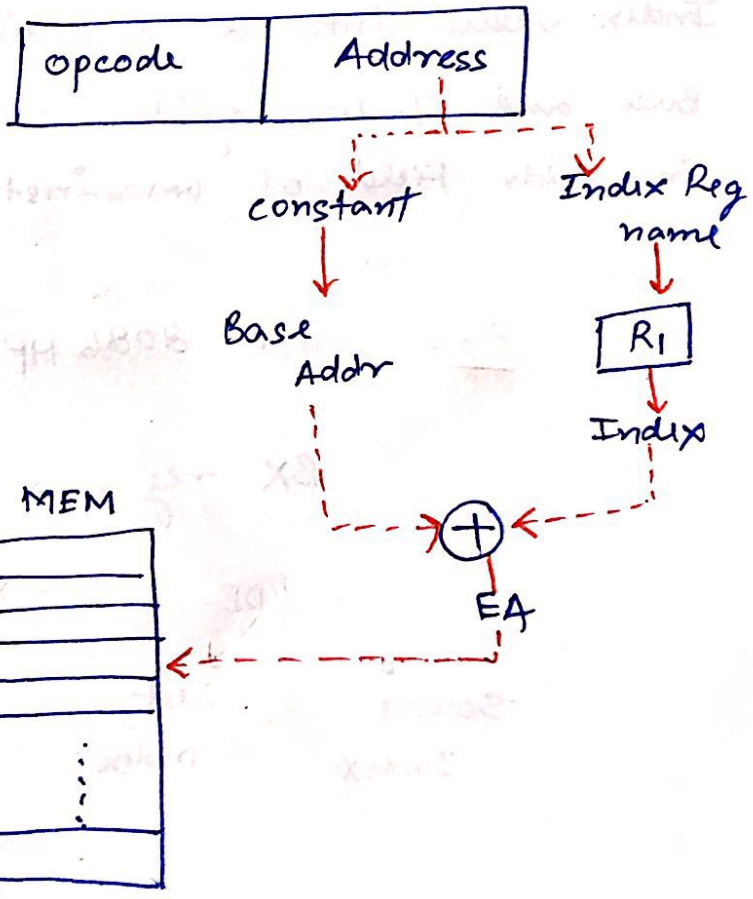
$$Data = [EA]$$

$$= [\text{Addr field value} + [R_i]]$$

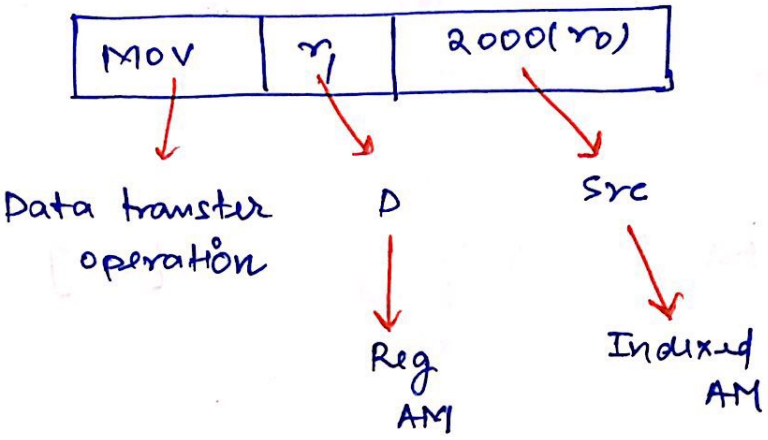
Actions :-

- LRR → to get the index value
- LALV → to calculate the EA
- LMR → to access the 'Data'

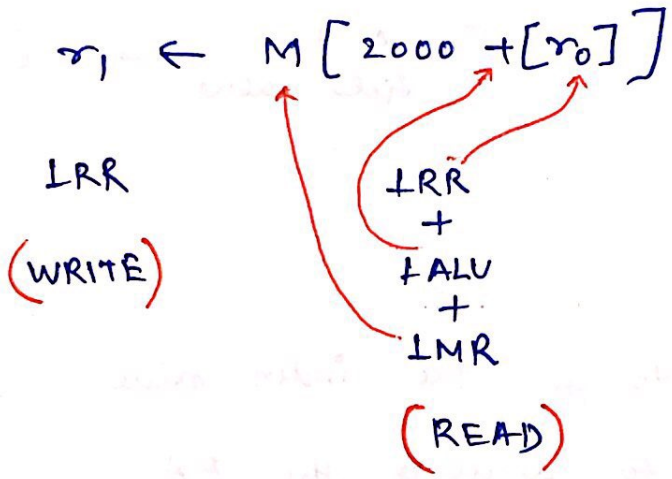
Instⁿ Design :



Eg:-




$r_0$ is a Index register




NOTE :- When the CPU support both Base and Index registers then we can store Base addr & Index value into a respective registers. Later Base and Index register names are maintained in Addr Field of an inst$^n$.

Eg:- In 8086 MP

BX reg → Base register

SI | DI reg → Index reg

Source Index

Dest index

```
MOV   BX , #2000
MOV   SI , #4
```

| MOV | AX | BX[SI] |
|---|---|---|

$$AX \leftarrow M[BX + CI]$$

## 7. Indirect - Indexed AM :-

In this mode EA is in the memory, the resp EA is calculated by adding the index value to Base Address.

$$EA = \left[\begin{matrix}\text{Addr field value}\end{matrix} + [Ri]\right]$$

$$Data = [EA]$$

$$= \left[\left[\begin{matrix}\text{Addr field value}\end{matrix} + [Ri]\right]\right]$$

Addr field value → Addr of EA

Addr of EA → EA

EA → Data

Actions :-

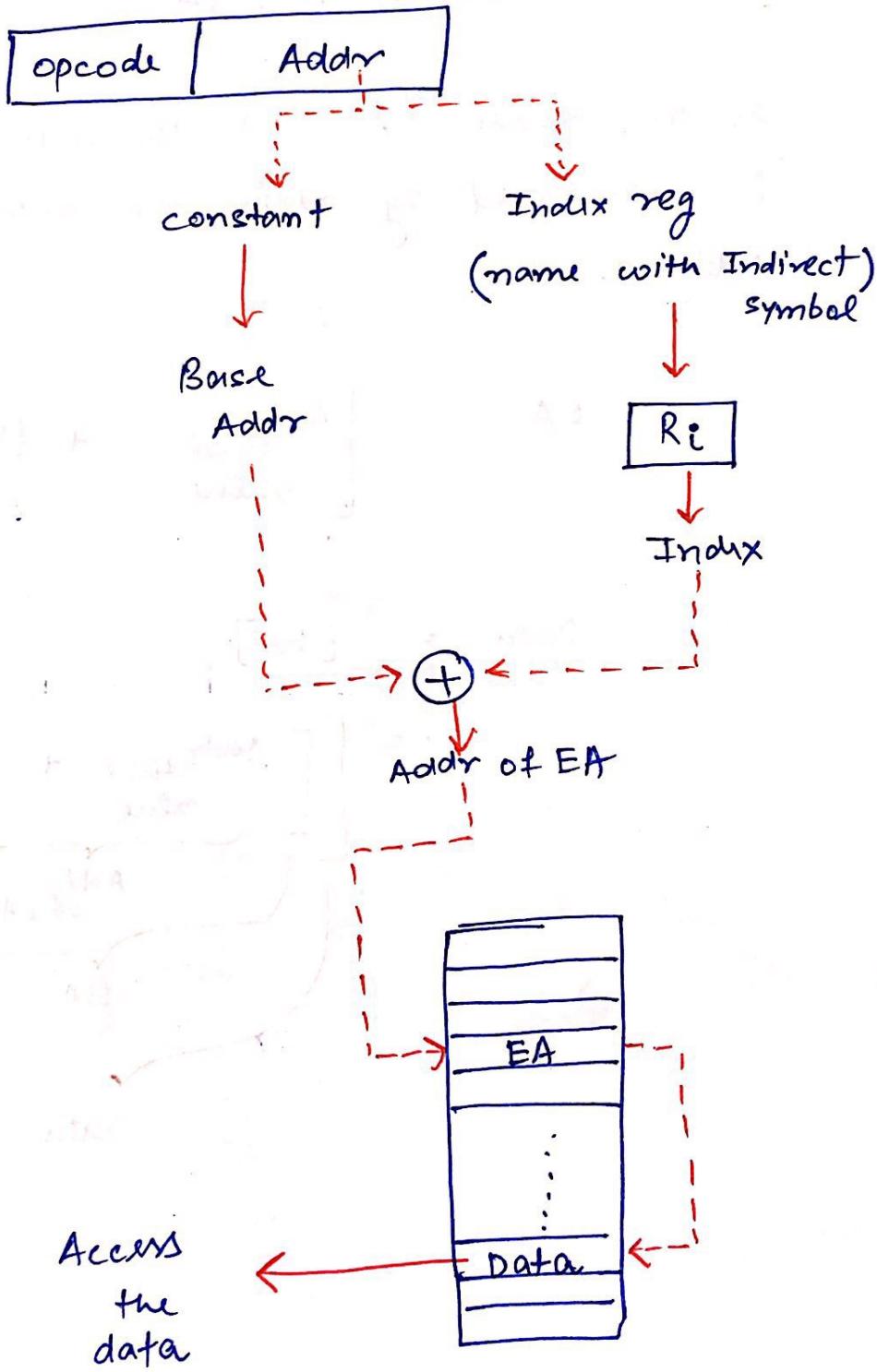
IRR → to get the Indix value

IALU → to calculate address of EA

IMR → to get the EA

IMR → to accus the data

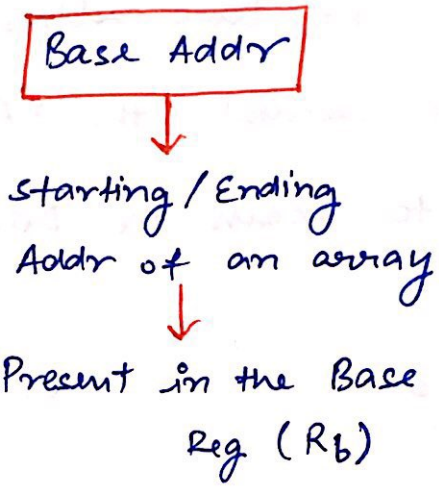
Inst$^{n}$ Design :



* Since this AM has more memory-retrenc (overhead) so it is not in practice.

## 8. Auto - Indixed AM :-

- This mode is used to access the linear array elements from the memory. In this style of accessing only one parameter is required i.e (Base address).




- In this mode EA is obtained either by incrementing or decrementing the base register content with a step size.

- Step size depends on a word-length of a CPU, so it is a fixed constant.

$$(EA = [R_b] +/- step_{size})$$

Data = [EA]

(Pre-increment mode)

= [ [$R_b$] +/- step-size ]

Base Addr — constant

EA

Data

Actions :-

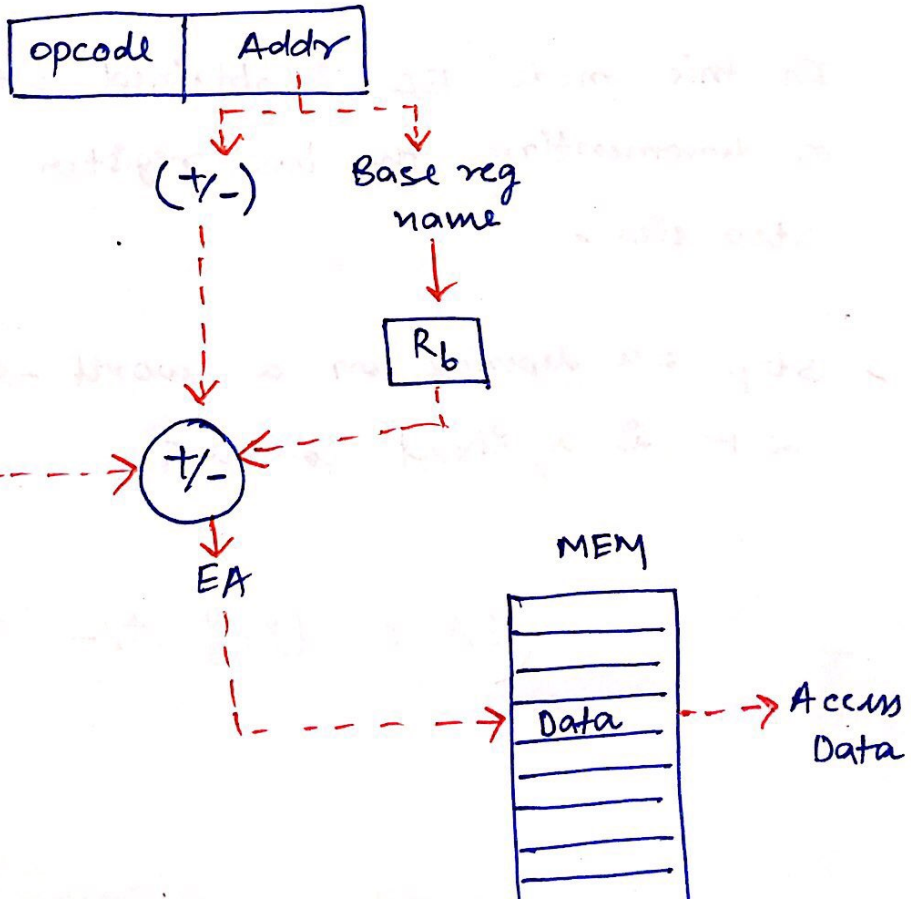
IRR → to get Base address

LALU → to calculate the EA

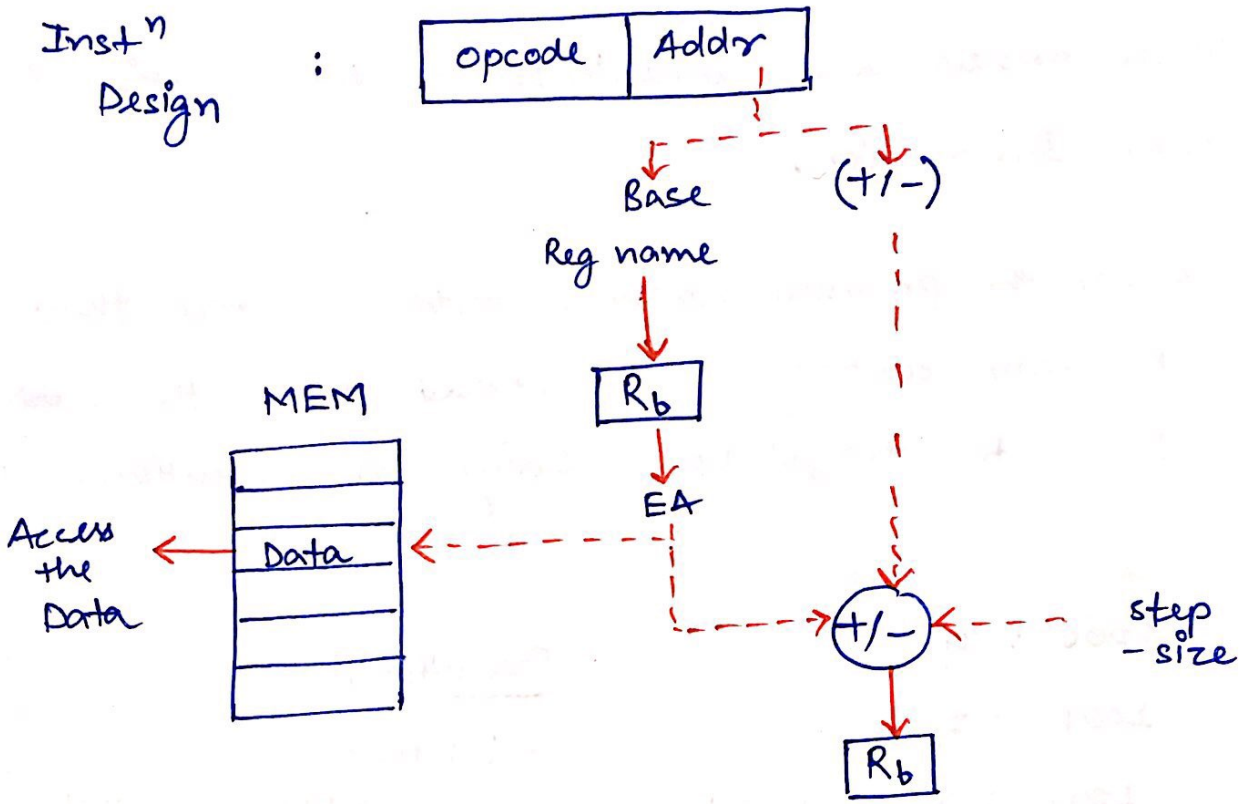
IMR → to access the Data

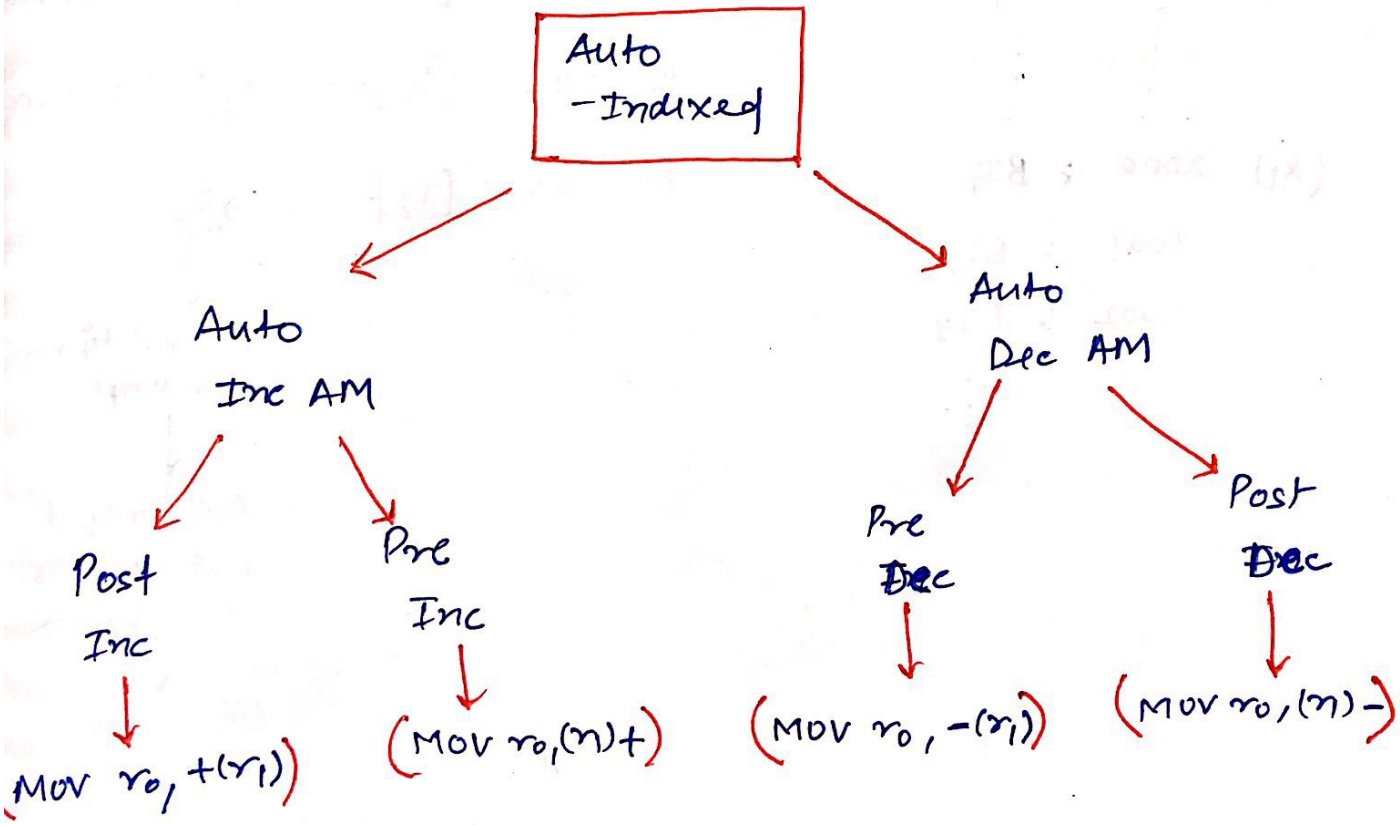
- Pre-Auto Indexed

Instⁿ Design :



## - Post - Auto Indixed




{ $r_1$ is Base register }

# Transfer of Control flow AM :- (TOC)

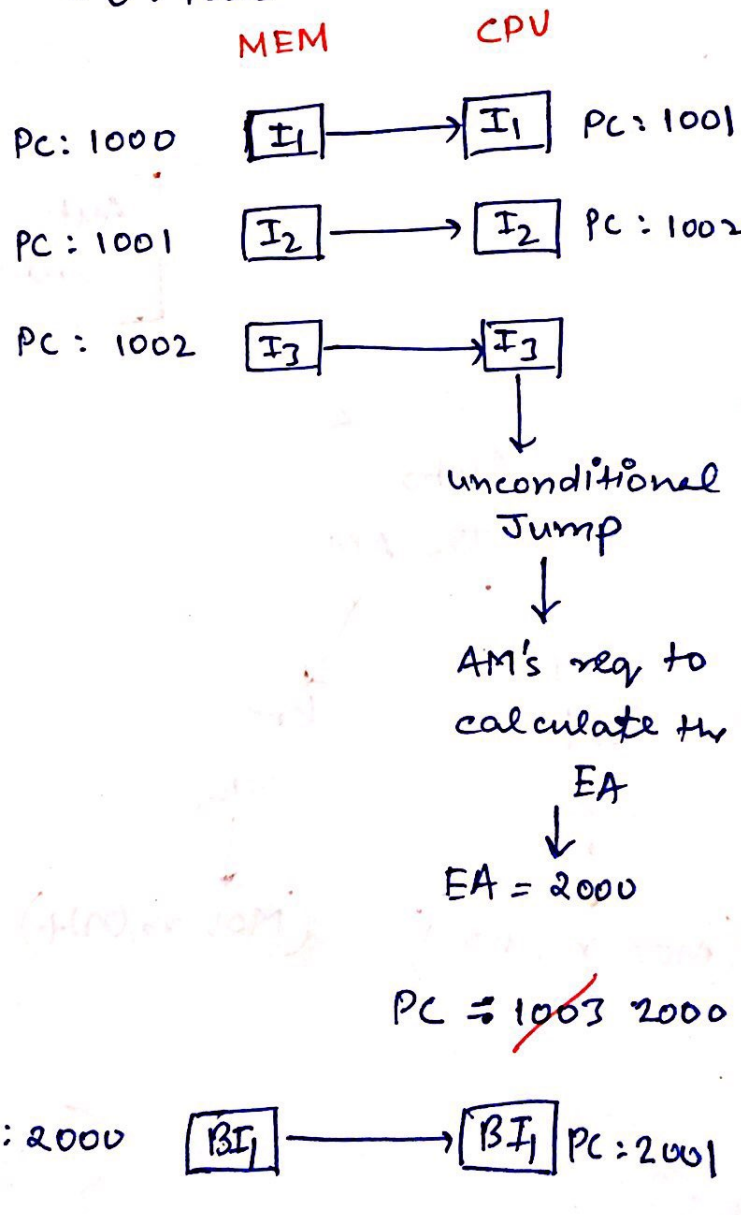
- These modes are concentrate on the Location of next Instruction.

- when the program contains control structures then program control is transfered from the current LOC to target LOC during its execution, i.e

1000 : $I_1$
1001 : $I_2$
1002 : $I_3$ (goto $l_1$)
1003 : $I_4$
⋮ ⋮
($l_1$) 2000 : $BI_1$
2001 : $BI_2$
2002 : $BI_3$
⋮ ⋮

Execution :-

-t : 1000



- Transfer of control inst$^n$ ( Jump or Branch or Skip) are used to implete the control structure in the Base Hardware.

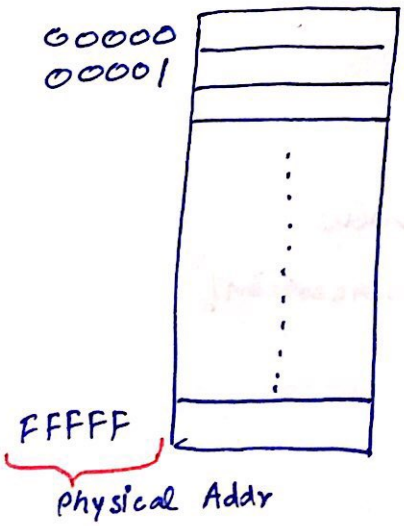
- Transfer of control Inst$^n$ are impliminted with the following addressing mode to calculate the EA (Branch address or target address)


- To analyze the above AM, let us consider the 8086 memory organization as a reference modal. (1MB memory organized into 16 logical segments with a segment size of 64 KB)

Before organization

1MB → RAM

$2^{20}$ cells

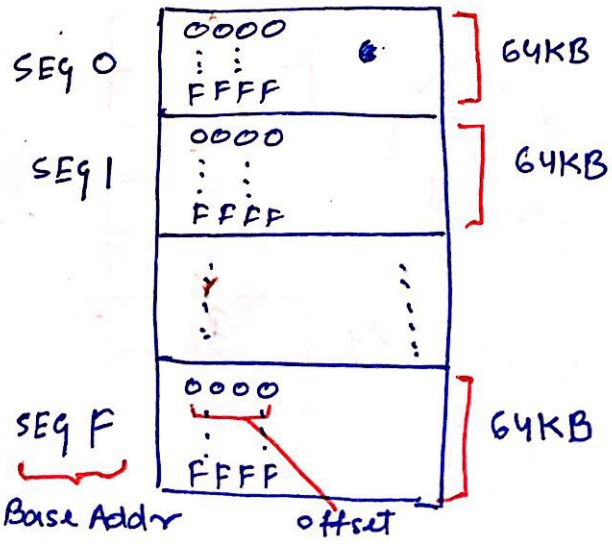



After organization

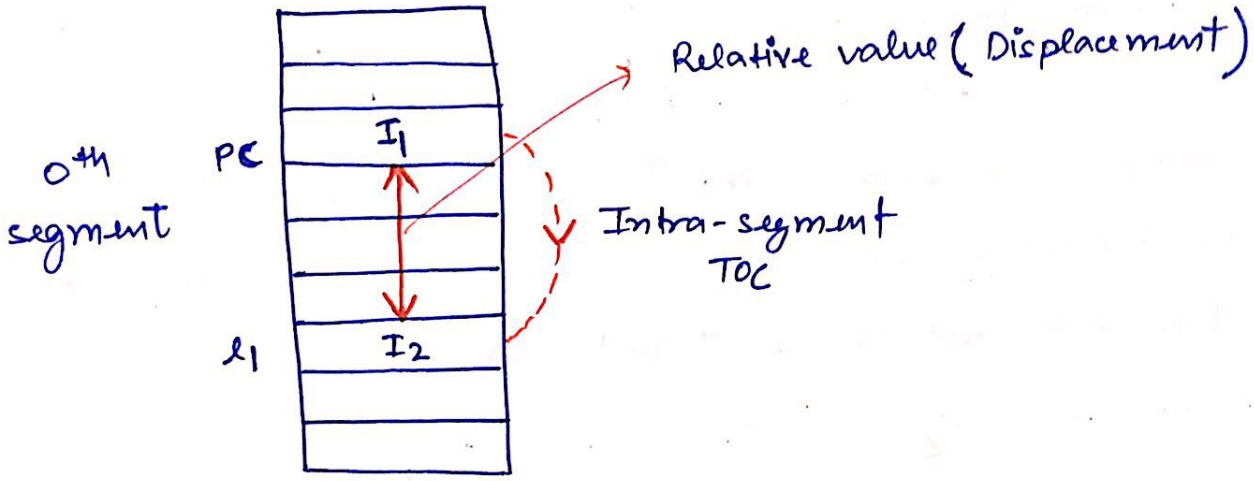
1MB → 16 × 64KB

$2^4$ $2^{16}$

$2^{20}$ cells B

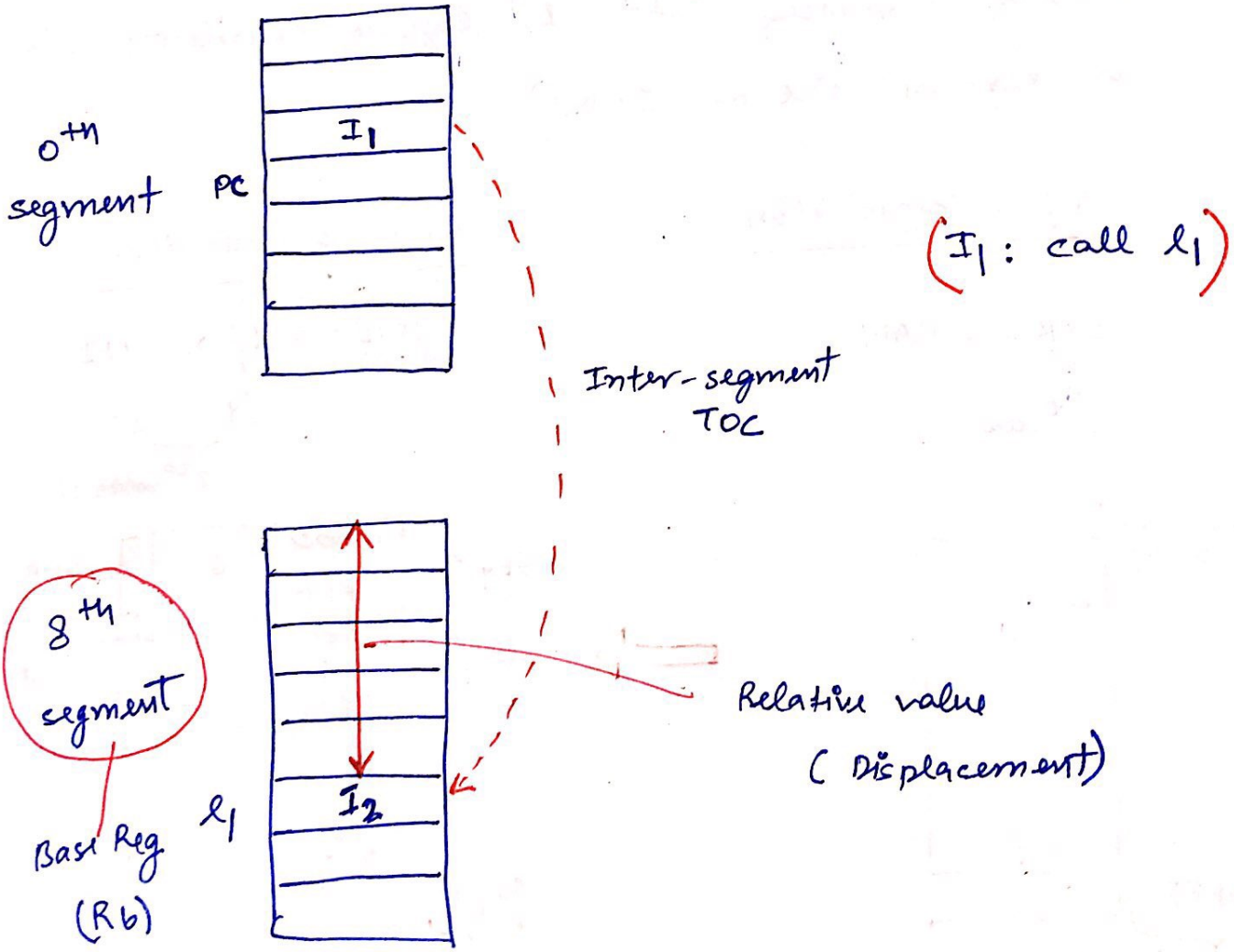


## Case I : Intra segment (TOC)




($I_1$ : JMP $l_1$)

## Case II : Inter segment (TOC)



1. PC - Relative addressing mode :-

- when the target address is present in the same segment then during the program execution control will be transfered within the segment known as Intra segment TOC.

- It is used to implement the above opⁿ.

- In this mode EA is calculated by adding the relative value to PC.

- Relative value means distance b/w the current LOC to target LOC. It is a signed constant, present in the address field of an Instⁿ.

EA = PC + Addr field value

EA → Next Instⁿ LOC

PC → current LOC

Addr field value → Relative value

(PC ← PC + relative value)

2. **Base - Register AM :-**

- When the target Inst$^n$ is present in the different segment, then during the program execution control will be transfer b/w the segments known as Inter-segment TOC.

- It is used to implement the above operation.

- In this mode EA is calculated by adding the relative value to base register.

EA = $[R_b]$ + Addr field value

- EA → Next Inst$^n$ Addr
- $[R_b]$ → Target segment Base Addr
- Addr field value → relative value

$$(PC \leftarrow [R_b] + \text{relative value})$$

**NOTE:-** Both AM's are suitable for prog relocation at Run-time.

Base-Reg AM is best suit to write the position independent code.

Q. Consider a hypothetical CPU which support 4 W inst$^n$ stored in a memory with a starting Address $(400)_{10}$. 1st word of inst$^n$ specify the Opcode and remaining words of the inst$^n$ specify the address part. Word length of CPU is 32-bit. Address field of Inst$^n$ contains (-42). what is the target address when the opcode is designed with a :

(a) relative addressing mode.

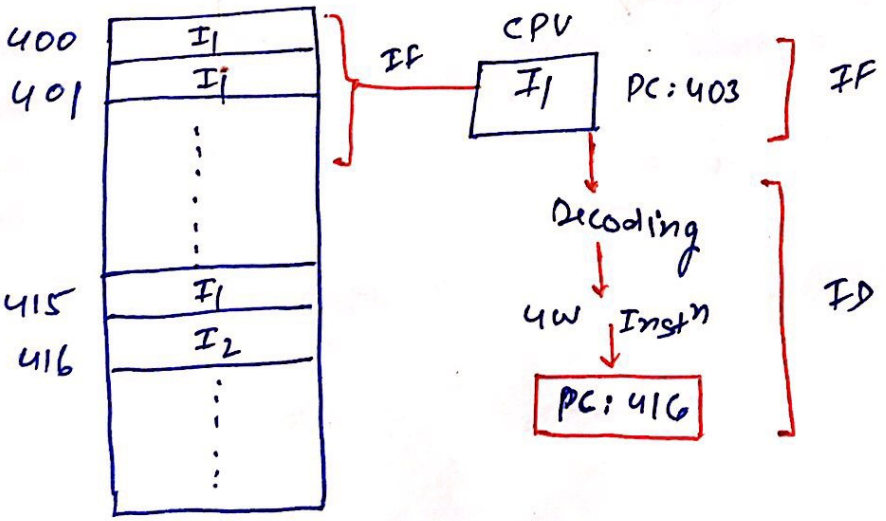
(b) base register addressing mode

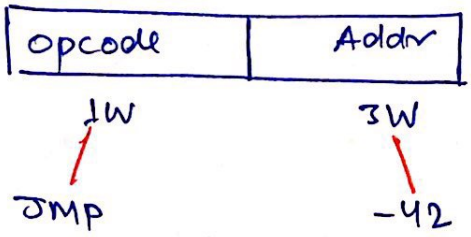
(Assume $R_b = 800$) ($I_1$ is Jump inst$^n$)

MEM : Byte addressable (Default config)

Inst$^n$ size = 4 W = 4 X 32b = 4 X 4B = 16 Bytes

(16 cells of storage required for one inst$^n$)






(a) Relative AM:

$$EA = PC + \text{relative value}$$
$$= 416 + (-42)$$
$$= 374$$

$(PC = 374)$

EC

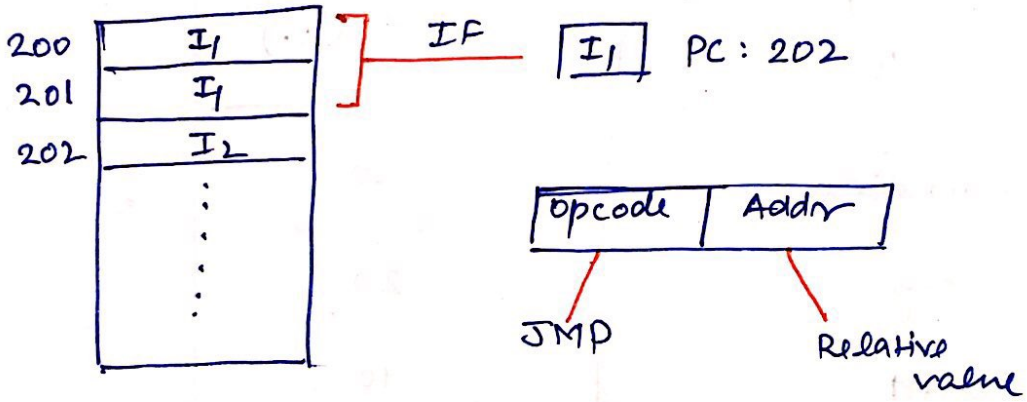
(b) Base - Reg AM:

$$EA = (R_b) + \text{relative - val}$$
$$= 800 + (-42)$$
$$= 758$$

$(PC = 758)$

EC

Q. Consider 16-bit hypothetical CPU which supports 1W long Instⁿ. Instⁿ format contains 8-bit opcode & 8-bit address field. Instⁿ is placed in a a byte-addressable memory with a starting address $(200)_{10}$. Opcode is designed with a PC relative JMP opⁿ. During its exec. control will be transfered to $(142)_{10}$ location. What is the relative value in Hexadecimal.




$$PC = PC + \text{rel. value}$$

$$142 = 202 + rel$$

$$rel = -(202 - 142)$$

$$= -(60)$$

Represent 60 into 8-bit binary

00 1 1 1 1 00

2's comp → 1 1 0 0 0 1 0 0

$(C4)_H$

Q. Consider 1 GHz clock-frequency processor, which uses different operand accessing modes discribe below. Assume that MR consumes 5-cycle, ALU computation consumes 3-cycle and 0-cycle consumed when the data is present in register. What is the avg. operand fetch rate of CPU (million words per second).

| Operand Accessing mode | Frequency (%) |
|---|---|
| Immediate | 20 |
| Reg | 20 |
| Direct | 10 |
| In-direct | 20 |
| Reg-indirect | 20 |
| Indixed | 10 |

$$\text{Cycle time} = \frac{1}{\text{clk frequency}}$$

$$= 1\text{ns}$$

Av. time taken to fetch operand (LW) $= \dfrac{\sum (\text{freq of inst}^n) * \text{time-taken}}{\text{Total inst}^n}$

$$T_{av} = \frac{(20\times 0) + (20\times 0) + (10\times 5 \text{ cycles}) + (20\times 10 \text{ cycles}) + (20\times 5 \text{ cycles}) + (10\times 8 \text{ cycles})}{100}$$

$$(T_{av} = 4.3 \text{ ns})$$

Thus 1 word takes 4.3 ns

so in $4.3\times10^{-9}$ s $\longrightarrow$ 1 word

so, in 1s $= \dfrac{1}{4.3\times10^{-9}}$

232.5 million word/sec

Q. 16-bit CPU which support LW inst$^n$ stored in a Byte addressable memory with starting addr $(800)_{10}$ Inst$^n$ contain 8-bit opcode & 8-bit address field.

Processor register $r_0 = 200$ when inst$^n$ is designed with a memory - based AM then address filed contain 100. Memory content of 100 is 150. Calculate EA when the inst$^n$ is designed with:

(a) Immediate addressing mode

(b) Register AM

(c) Direct AM

(d) Indirect AM

(e) Reg - indirect AM

(f) Indixed - AM with $R_0$ as indix register

(g) Pre - auto increament AM with $R_0$ as a base Register.

(h) Post - auto dicreament AM with $R_0$ as base Register.

(i) PC - relative AM

(j) Base - register AM with $R_0$ as base register.

# Instruction Set :-

- In the CPU design, 3 category of inst$^n$ are present name as data - transfer inst$^n$ (MOV, LOAD, STORE, PUSH, POP, IN, OUT), Data manipulation inst$^n$ (ADD, SUB, MUL, DIV, INC, DEC, etc), Logical inst$^n$ (AND, OR, XOR, CMP, etc) and shift & rotate inst$^n$.

NOTE :- During the exec. of a INC inst$^n$, const 1 is added to reg or memory content. when it satisfy the maximum limit then count will be rollback to 0 without changing the carry flag (CY).

Inst$^n$ : | INC | $r_0$ |

$r_0 \leftarrow r_0 + 1$

eg | INC | $r_0$ |

FF

CY

$r_0$ : (1) 1111 1111

+ 1

0 000 0000

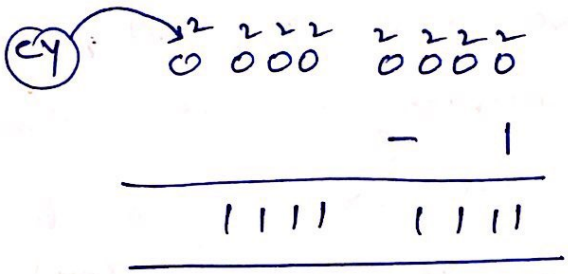
{ CY = 0 and $r_0$ = 00 }

NOTE :- During the exec of DEC inst$^n$ constant 1 is substracted from the register or memory content . when it satisfies the minimum limit then the count will be roll back to maximum value without changing cy .

Inst$^n$ : | DEC | $r_0$ |

00 H

$r_0 \leftarrow r_0 - 1$




$r_0 = F$
$cy = 0$

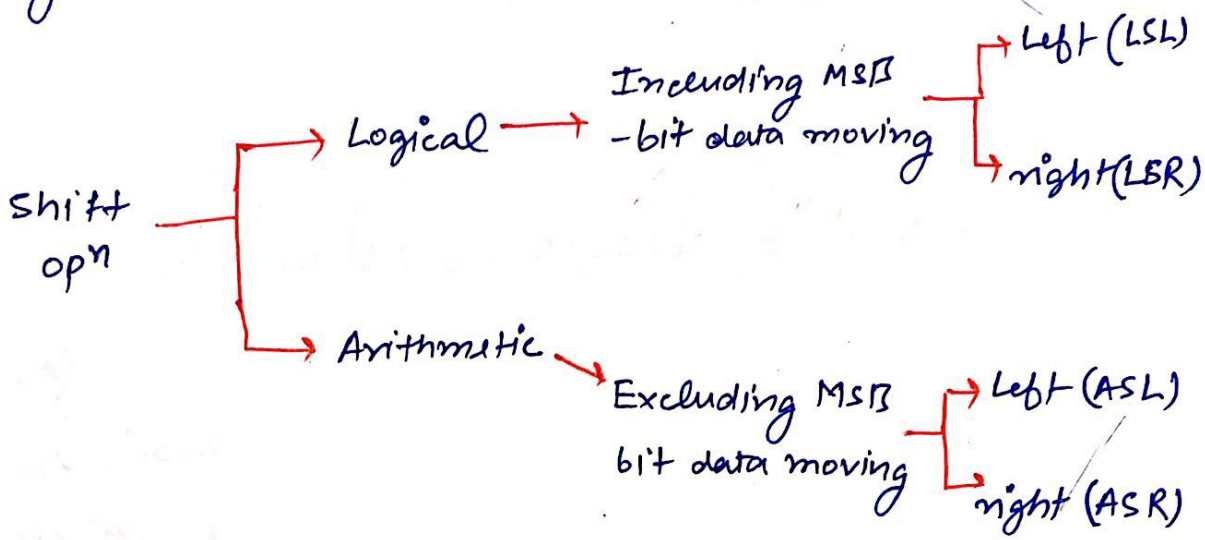
⇛ CMP (compare)

- CMP inst$^n$ is used to compare 2$^{nd}$ operand with 1$^{st}$ operand.
- It invokes SUB operation during its execution.
- After SUB, result will be available in the Zero and carry flags.

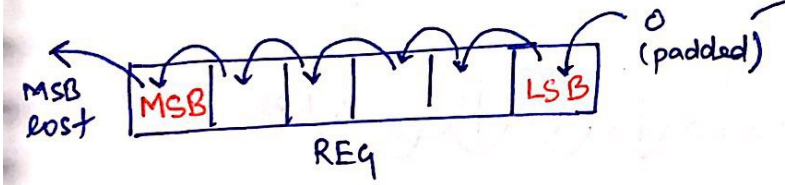
1. Z = 1 , cy = 0 → (op2 = op1)
2. Z = 0 , cy = 1 → (op2 > op1)
3. Z = 0 , cy = 0 → (op2 < op1)

## ⇛ Shift :-

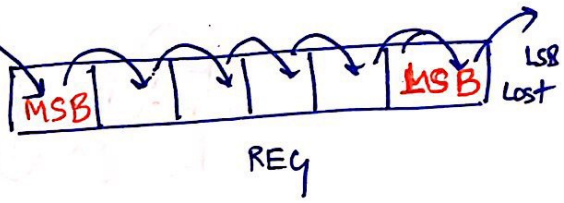
- while execution of this instⁿ data bits are moving in a bitwise sequence with loss of data.




### LSL




### LSR




eg:

| LSL | $r_0$ | #3 |
|---|---|---|

$r_0 = (FC)_H$

so,

1 1 1 1 1 1 0 0

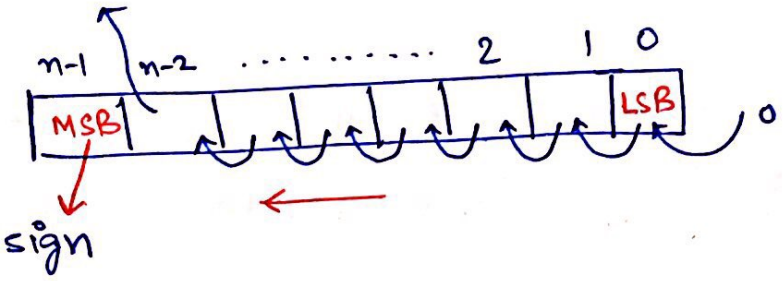
1 1 1 1 1 0 0 0

1 1 1 1 0 0 0 0

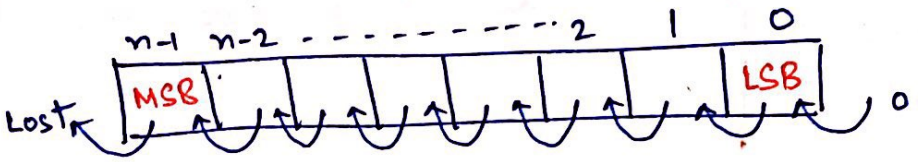
$r_0$ | 1 1 1 0 0 0 0 0 |

E 0

## ASL :-




concept-wise

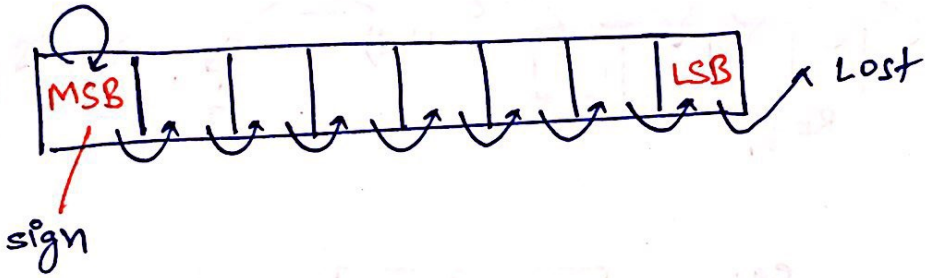



Implement-wise

(ASL = LSL)

means ASL is not practically possible

## ASR :-




eg :-

| ASR | $r_0$ | #3 |
|---|---|---|

$r_0$ : FC (H)

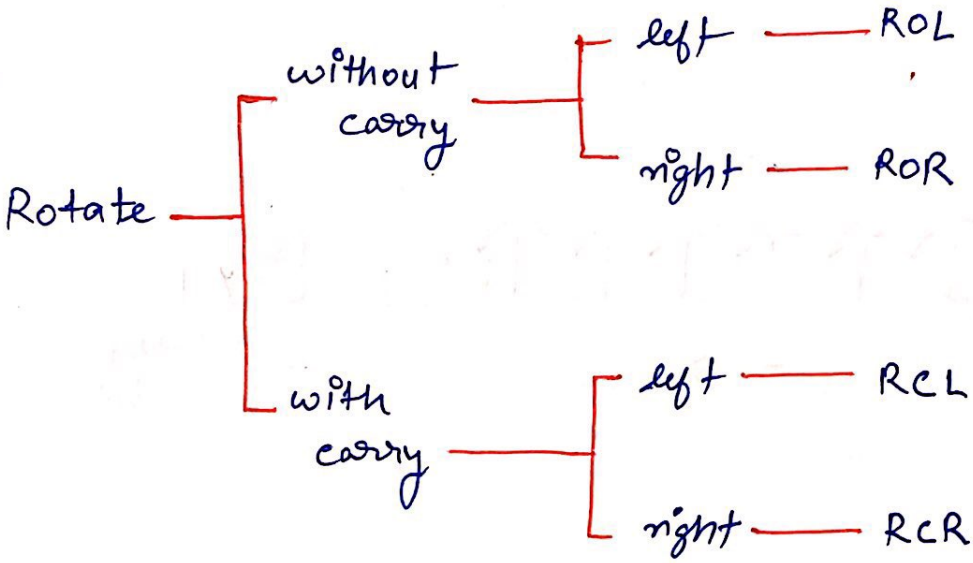
$r_0$ : 1111 1100

3(ASR)

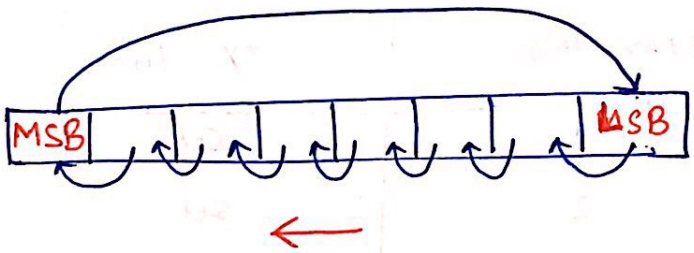
$r_0$ : 1111 1111

## ⇒ Rotate :-

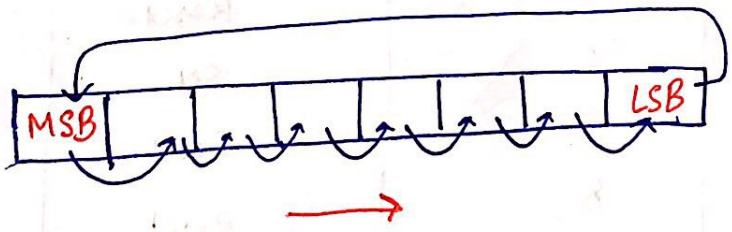
- During execution of this instⁿ data bits are moving in a bitwise sequence without loss.



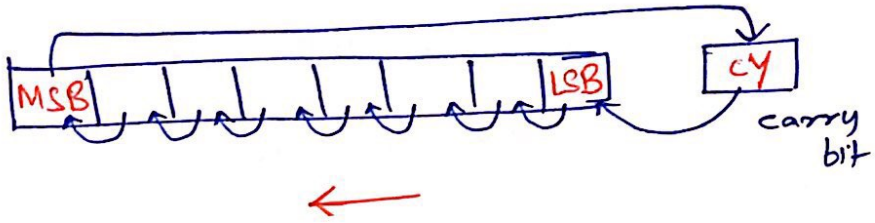

ROL :-



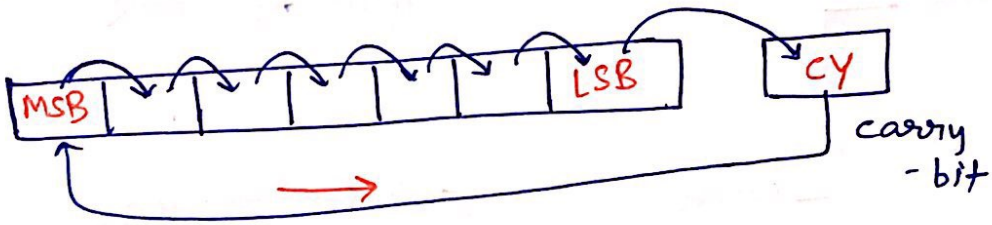

ROR :-



RCL :-




RCR :-



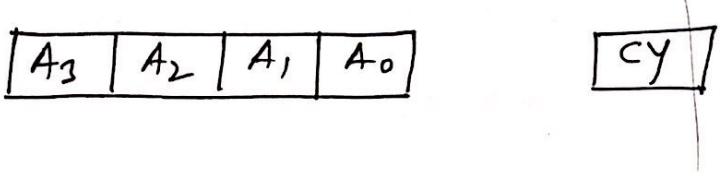

Ex :- RCL o/p status is

| Iteration | CY status |
|---|---|
| 1 | Set |
| 2 | Set |
| 3 | Reset |
| 4 | Set |
| 5 | Reset |
| 6 | Set |
| 7 | Reset |
| 8 | Reset |

what are the I/p values ?

I/p = 1101 0100

Eg:- consider 4-bit reg




ROR

| Iteration | Reg value |
|---|---|
| 1 | $A_0\ A_3\ A_2\ A_1$ |
| 2 | $A_1\ A_0\ A_3\ A_2$ |
| 3 | $A_2\ A_1\ A_0\ A_3$ |
| 4 | $A_3\ A_2\ A_1\ A_0$ |

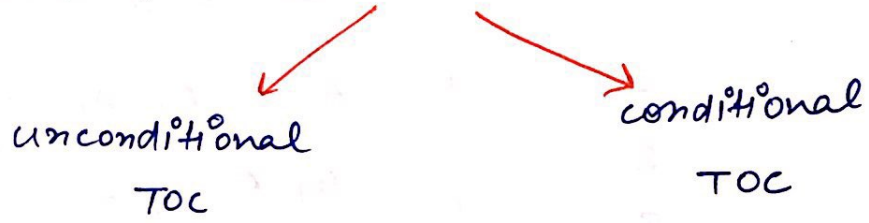
So, for n-bit register after n itiration register value come back to its I/p value

RCR

| Iteration | Reg value | CY |
|---|---|---|
| 1 | $CY\ A_3\ A_2\ A_1$ | $A_0$ |
| 2 | $A_0\ CY\ A_3\ A_2$ | $A_1$ |
| 3 | $A_1\ A_0\ CY\ A_3$ | $A_2$ |
| 4 | $A_2\ A_1\ A_0\ CY$ | $A_3$ |
| 5 | $A_3\ A_2\ A_1\ A_0$ | CY |

so, in case of RCR for n-bit reg, reg comes back to its I/p form after (n+1) iterations

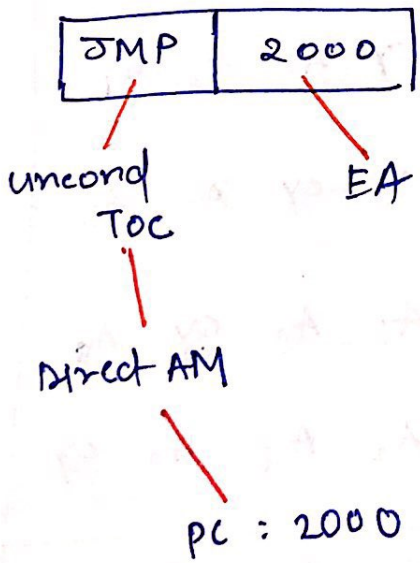
⇒ Transfer of control (TOC) :-




Unconditional TOC –

Control of program flow will be transffered to target location without checking the condition.

Eg :- (JMP 2000)




Inst$^{n}$ format

- JMP inst$^n$ is used to implement the goto statement and the machine control inst$^n$ (halt)

- HALT is a machine control inst$^n$. It invokes the uncond JMP inst$^n$ with starting add of halt as target address i.e

(LL: JMP LL)

while exec. of this inst$^n$, control will be transfered to same location so processor will be enter into a infinite loop there fore RESET action is required to run the new program.

Code:-

1000 : $I_1$

1001 : $I_2$

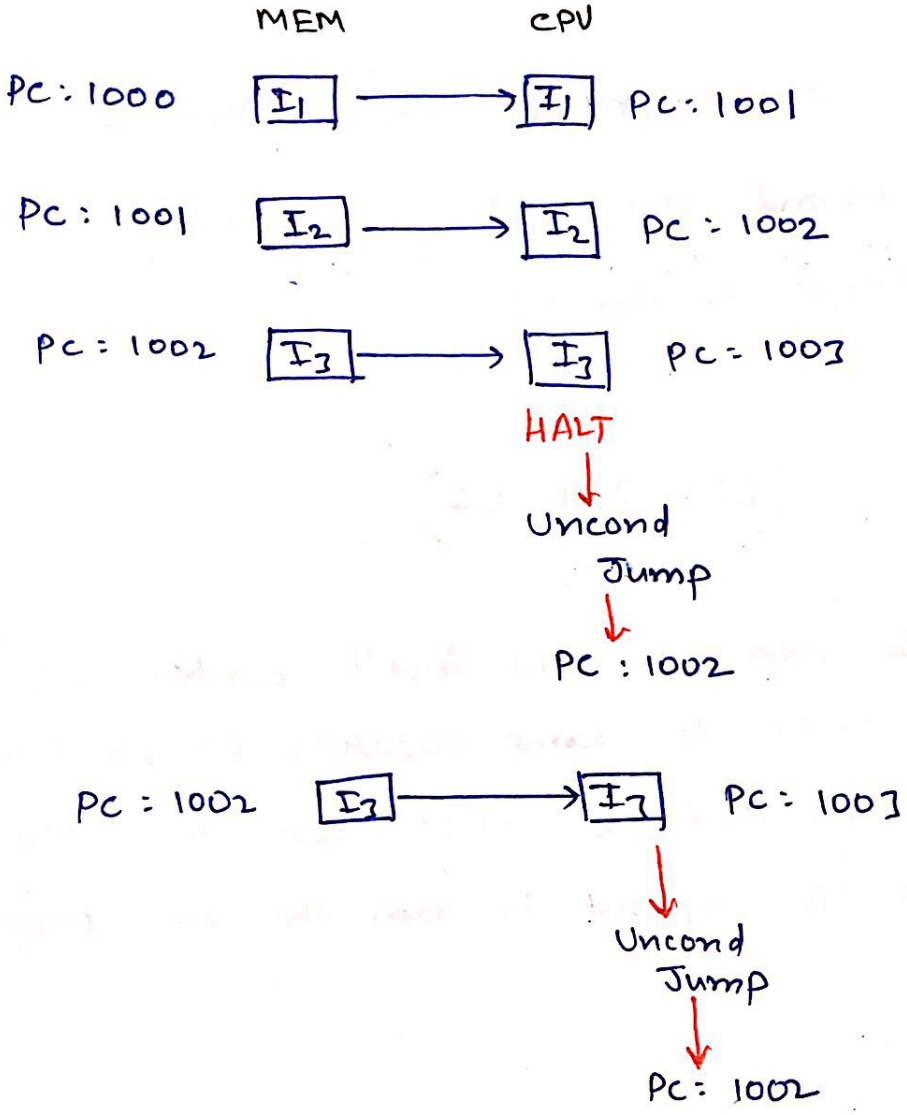
1002 : $I_3$ (HALT)

1003 : $I_4$

Expected o/p sequence = $I_1 \rightarrow I_2 \rightarrow I_3$

## Inst$^{n}$ Execution :-

-t : 1000



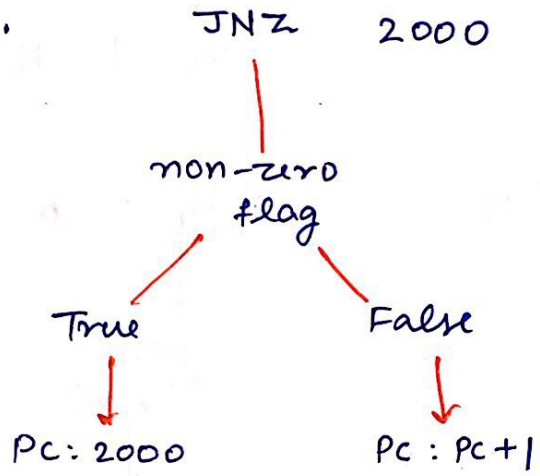

So, Actual O/p sequence $= I_1 \rightarrow I_2 \rightarrow I_3 \rightarrow I_3 \rightarrow I_3 \cdots$

## Conditional TOC :-

- while exec. of this Inst$^n$, associative condition is evaluated based on the status of a previous inst$^n$ (flag register).

- when the condition is true, then control will be transfered to target location otherwise control will follow sequential flow.

eg.

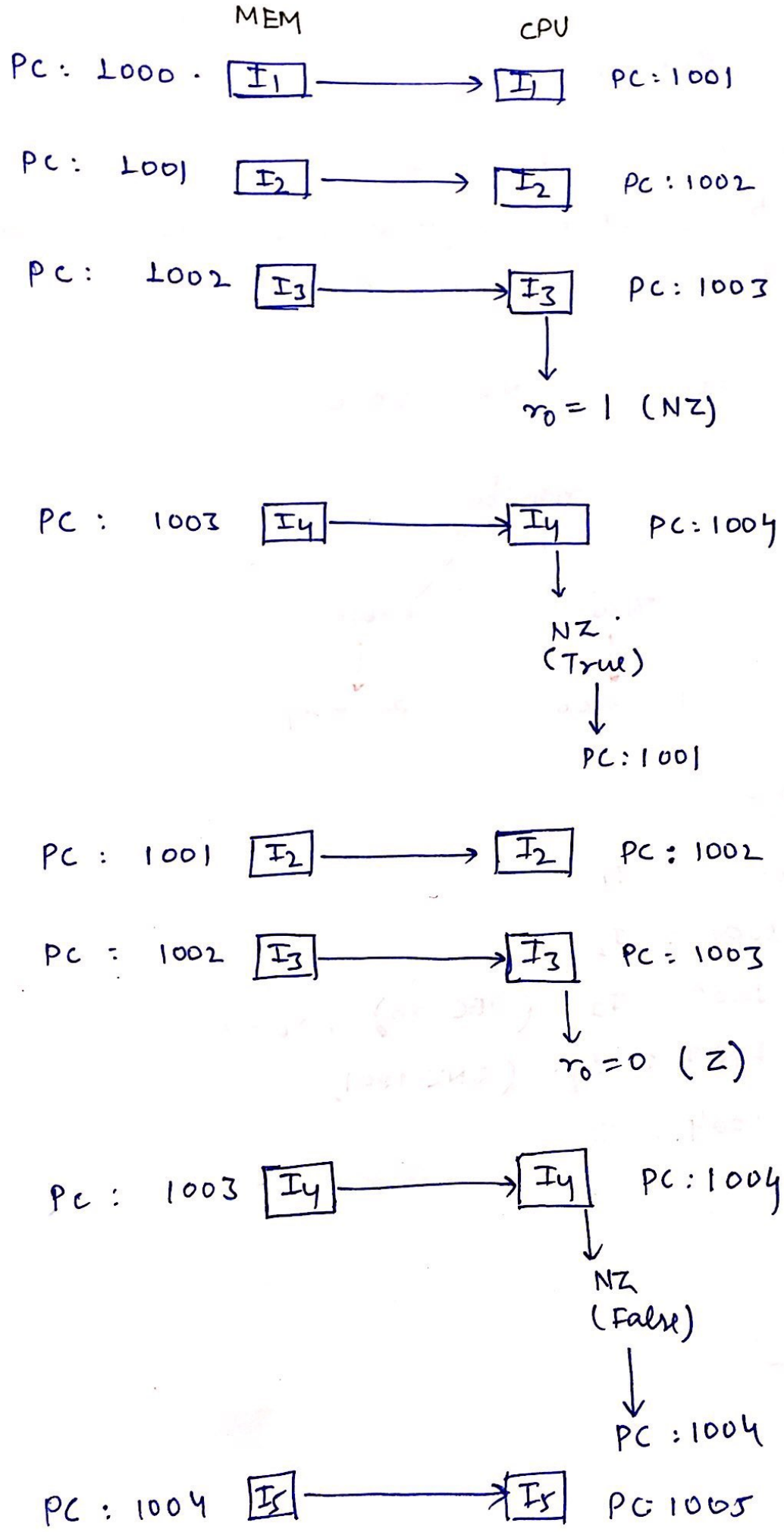



Code :-

```
1000 : I1
1001 : I2
1002 : I3  (DEC r0)  ; r0 = 2
1003 : I4  (JNZ 1001)
1004 : I5
```

Exec :-

- t : 1000


MEM
CPU
PC : 1000
I1
I1
PC : 1001
PC : 1001
I2
I2
PC : 1002
PC : 1002
I3
I3
PC : 1003
r0 = 1 (NZ)
PC : 1003
I4
I4
PC : 1004
NZ
(True)
PC : 1001
PC : 1001
I2
I2
PC : 1002
PC : 1002
I3
I3
PC : 1003
r0 = 0 (Z)
PC : 1003
I4
I4
PC : 1004
NZ
(False)
PC : 1004 (No change)
PC : 1004
I5
I5
PC : 1005

- These instn is used to implement the conditional selection statements (if, switch) and various iterative statements (for, while, do).

## Sub - Program :-

- Sub-Program is a re-usable prog that means, repeatedly occured functionality in the application is developed as a subprogram only once later, refer it in the application as many times.

### Characteristic :-

(i) Single Entry / Single Exit point

(ii) Main prog is suspended during the execution of a sub-program.

(iii) Control will be transfer back to main program after the completion of a sub program.

### Implementation :-

It is implemented in the base HLW using the paired instruction i.e

CALL - RET

CALL — RET

CALL → Entry point → Present in the main program → Invokes PUSH opⁿ to save the PC value into a stack later transfer the control to sub prog

RET → Exit point → Present in the sub program → Invokes POP opⁿ to restore the return address into a PC.

- In this implementation process STACK is used to store the return addresses.
- "Return address" is next instⁿ address after the CALL instⁿ.

Code:-

```
1000 : I1
1001 : I2
1002 : I3 (call 2000)
1003 : I4
  :    :
```

```
2000 : BI1
2001 : BI2
2002 : BI3
2003 : BI4 (RET)
2004 : BI5
  :    :
```

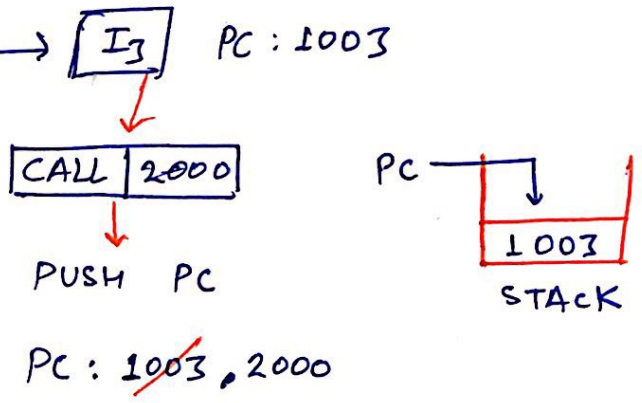
## Instⁿ Execution :-

- t : 1000

PC : 1000

PC : 1001

PC : 1002 $I_3$ ⟶ $I_3$ PC : 1003




PC : ~~1003~~, 2000

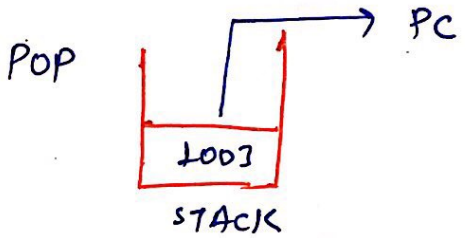
PC : 2000

PC : 2001

PC : 2002.

PC : 2003 $BI_4$ ⟶ $BI_4$ PC : 2004

RET




PC : ~~2004~~ 1003

PC : 1003 $I_4$ ⟶ $I_4$

O/p sequence = $I_1 - I_2 - I_3 - BI_1 - BI_2 - BI_3 - BI_4 - I_4 - I_5$

# TOC Instⁿ :-

Direct TOC Instⁿ

- JMP 2000
- CALL 2000
- JNZ 2000
- CALLNZ 2000
- JZ 2000
- CALLZ 2000

Indirect TOC instⁿ

- RET
- RETNZ
- RETZ

etc

Only the PC will affected

* Direct TOC instⁿ

(EA is in Addr field of an Instⁿ)

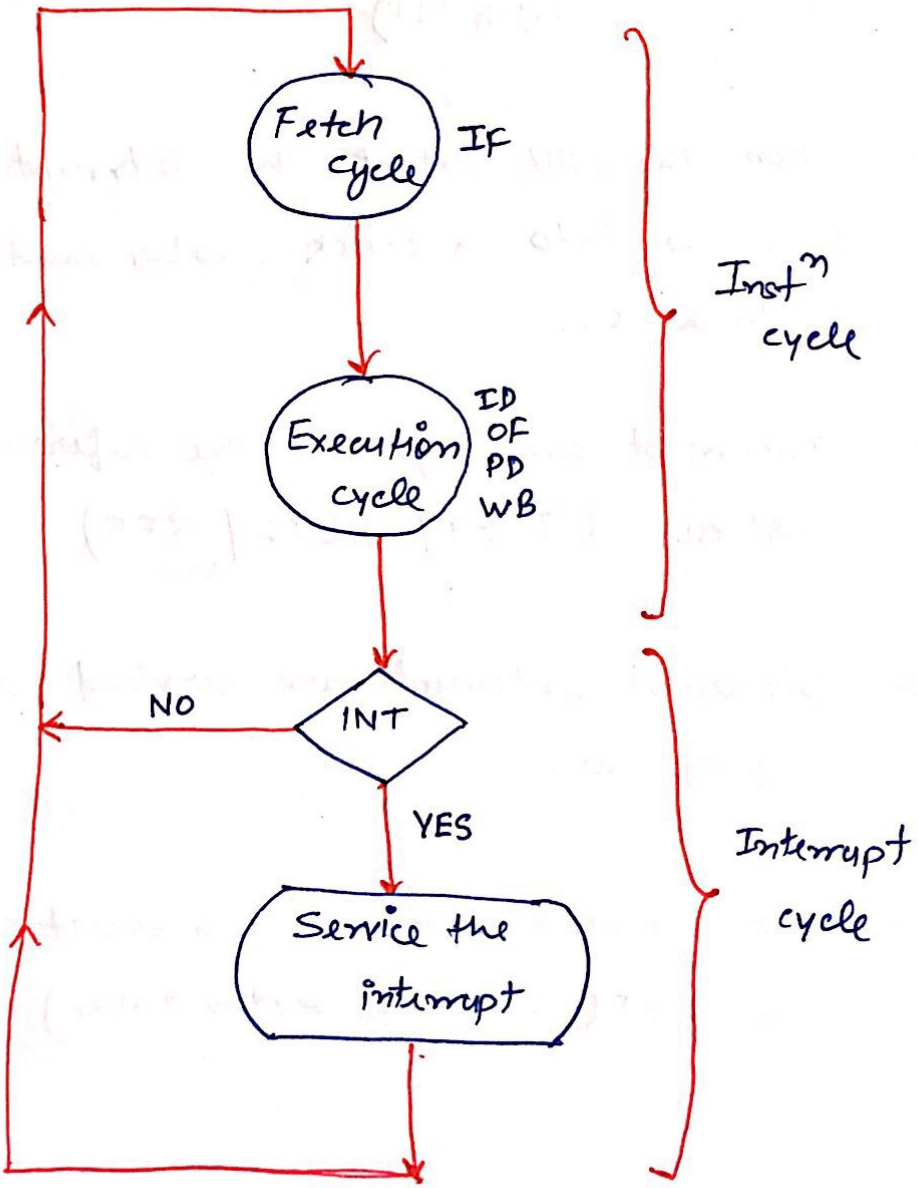
* Indirect TOC instⁿ

( EA is in Memory (STACK) )

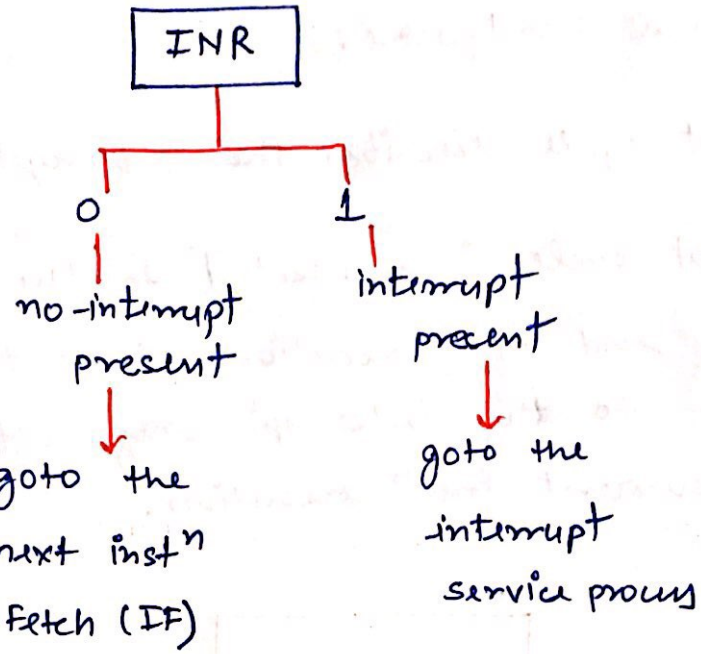
| TOC | unconditional | conditional |
|---|---|---|
| Direct | JMP 2000<br>CALL 2000<br>HALT, goto, etc | JNZ, JZ,<br>CALLNZ,<br>CALLZ,<br>If, switch, for<br>while, etc |
| Indirect | RET | RETZ,<br>RETNZ,<br>etc |

# Interrupt cycle :-

- Interrupt is a signal which is generated by Internal or External hardware.
- Interrupt cycle describes the interrupt handling process.
- Interrupt cycle is connected to the inst'n cycle at the end of execution cycle so that, CPU will respond to the interrupt only after the completion of a current inst$^{n}$ execution.



- After completion of a every inst$^n$ execution, CPU reads the status of an interrupt pin to detect the interrupts. i.e




- when the CPU detect the interrupt then it saves the PC value into a STACK, later loading the vector address into a PC.

- Interrupt sub-programs are referred with a vector address, ( IRET / RETI / RFE)

- Different interrupts are serviced with different sub-programs.

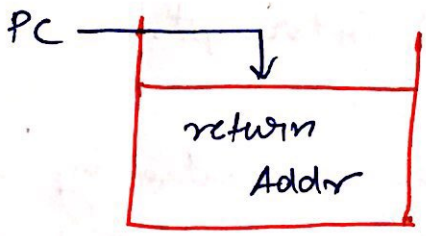
- All the vector addresses present in the memory called as IVT ( Interrupt vector table)

- During the execution of a interrupt sub-program when the CPU encounters the IRET instⁿ, then it perform or invokes the POP opⁿ to restore the return address into PC.

- After servicing the interrupt control will be transffered to user program.

**NOTE** :- Objective of a interrupt cycle is the initialization of interrupt sub-program. In this process PC value will be saved into a STACK, later vector address is loaded into PC.

IVT (Interrupt vector Table)




PC ← vector Addr

- CPU services the interrupt based on the priority assignment (suspend the interrupt based on high priority interrupt arrival) called as nested Interrupt processing mechanism.

Types of Interrupts :-

Hardware interrupt | Software interrupt

Hardware interrupt :-

- It is a signal which is generated by the External & or internal hardware components. So it may be an internal interrupt or external interrupt.

- External interrupt are generated by external devices
  Eg- Power supply, Basic I/O ( keyboard, printer)

- Internal interrupt is a signal which is generated by internal hardware present in the motherboard.
  Eg - Temperature sensor, timer, critical sensor, Invalid opcode, divide by zero, stack overflow.

- In the CPU hardware pins are reserved to hold the hardware interrupts. Eg- In 8085 MP

RST 4.5, RST 7.5, RST 6.5, RST 5.5, INTR


Priority

In 8086 MP

| | |
|---|---|
| INTR | Low priority |
| NMI | high priority |

Software interrupt :-

- It is an inst$^{n}$, used to execute the pre-difined I/O functions in the processor area.

Eg - system calls

Maskable interrupt :-

- It is a low priority hardware interrupt so CPU may or may not service this interrupt when it occurs.

Non- maskable Interrupt:-

It is a high priority hardware interrupt. So CPU compulsory service this interrupt when it occurs.

Vectored Interrupt :-

This interrupt contains vector information so vector addresses is calculated based on the interrupt vector

Eg :- RST 4.5

Non - vector Interrupt:-

This interrupt doesn't contains the vector information. So CPU will be sending the ACK signal and wait until the interrupt source supplies the vector address.

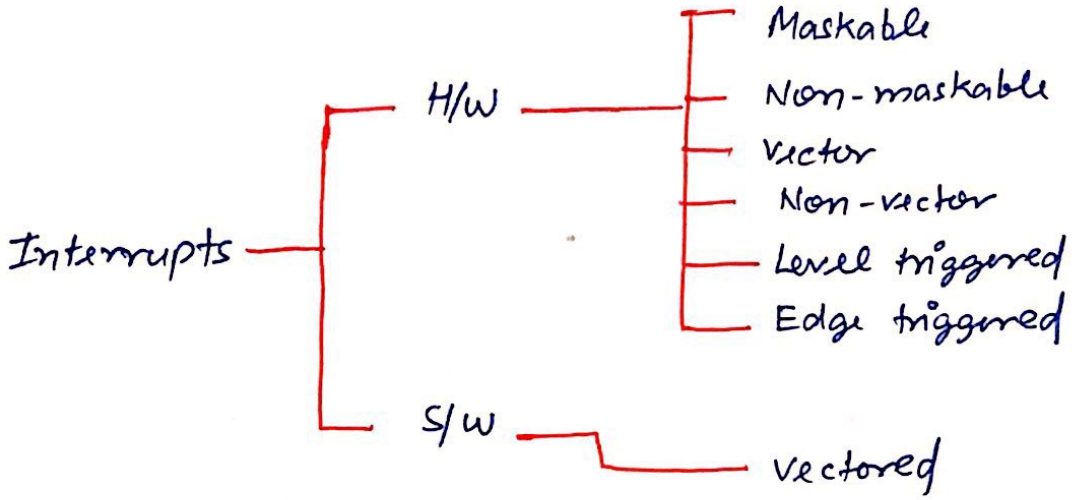
Eg :- INTR, $\overline{\text{INTA}}$

Level-Triggered interrupt :-

This interrupt operates on low level and high level transition of CLK signal.

Edge- Triggered interrupt:-

This interrupt is operated based on the raising edge and falling edge transition of a CLK signal.


Interrupts
H/W
S/W
Maskable
Non-maskable
Vector
Non-vector
Level triggered
Edge triggered
Vectored