"Transfer of control flow Addressing Modes":-

:- These modes are concentrate on the location of the Next Instruction in the memory.

★ when the program contain control structures then during the Execution of a program, control will be transferred from one place to another place.

★ To Impliment the Control Structures in the Base Hardware, transfer of control Inst^{n} (JMP) {JMP / Branch / skip} is used. This instruction is designed with following Addressing mode, used to calculate the Next Inst^{n} Address:

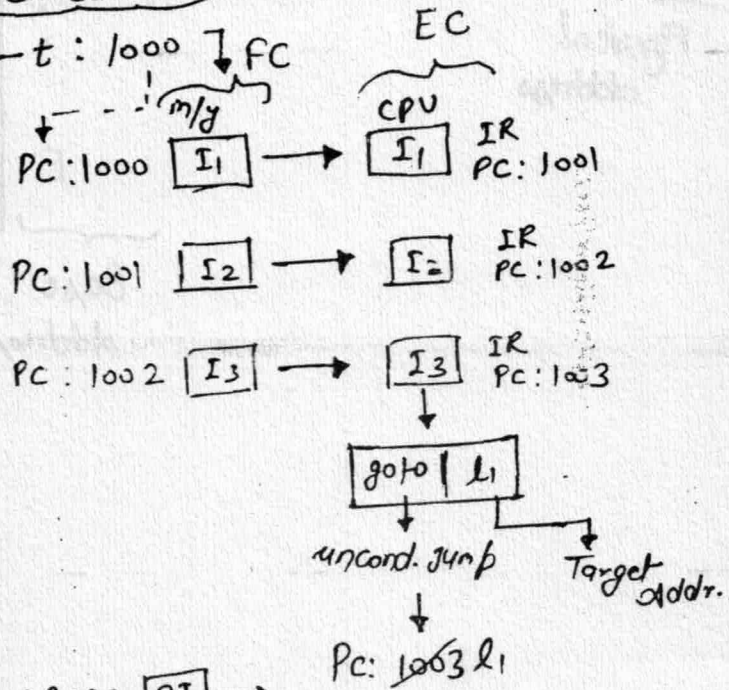
1.) Relative / PC-Relative A.M.

2.) Based / Base-Reg A.M.

Code:

```
1000 : I1
1001 : I2
1002 : I3 (goto l1)
1003 : I4
  .      .
  .      .
(l1) 2000 : BI1
  2001 : BI1
  .      .
  .      .
```

Execution:-



* To Analyze the above Addressing mode, Program Storage in the memory becomes important factor.

:- Let us consider 8086 memory organization to analyze the above Addressing mode :-

:- 8086 supports 1MB physical memory organized into a 16 logical segments of 64 kB.

Before organization

1 MB ⇒ mm

↓

2^{20} cells

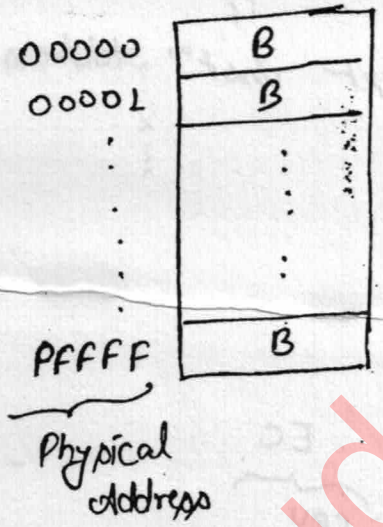
After organization

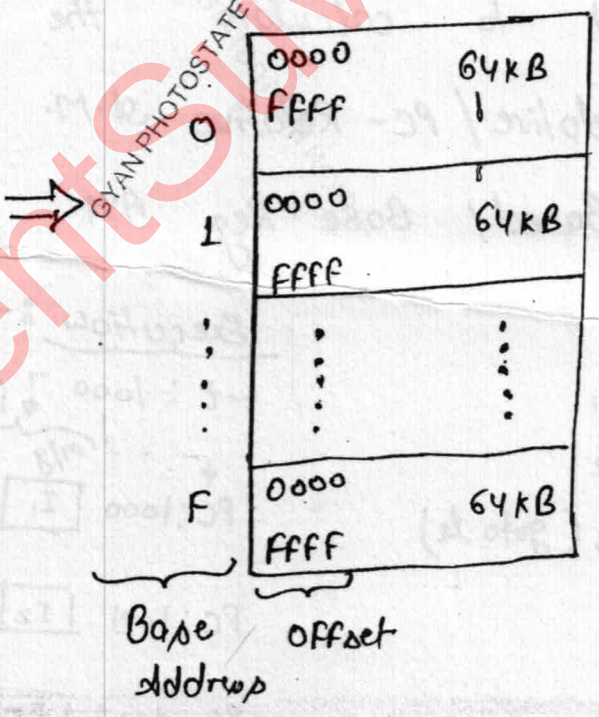
1 MB ⇒ 16 x 64 kB

↓ ↓ ↓

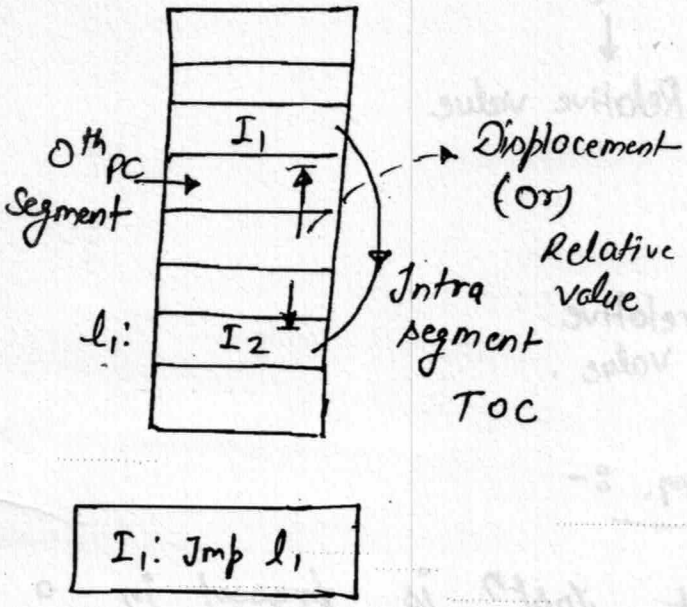
2^4 $2^6 * 2^{10}$

$\Rightarrow 2^{20}$ Bytes



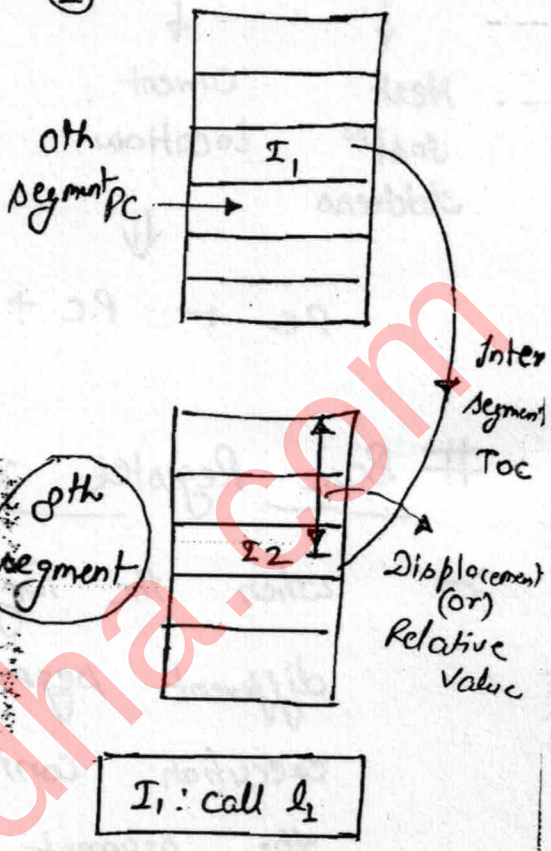


①



I_1: Jmp l_1

②



I_1: call l_1

Relative Addressing Mode :-

★ when the Target instruction is present in the same segment then during the program execution control will be transferred within the segment. Called as intra-segment transfer of control.

★ PC-Relative Addressing Mode is used to implement the Above operation. in this mode effective address is obtained by adding the relative value to a PC.

★ Relative value means distance between the current location to Target location.

★ It is a signed constant.

• EA = PC + Addr. field value

Next Inst^{n} Address — Current Location — Relative value

⇓

PC ← PC + relative value.

Base Register A.M. :-

:- when the Target Inst^{n} is present in a different segment then during the Program execution control will be transferred between the segment, called as Inter segment T.O.C.

★ Base Register Addressing Mode is used to impliment the above operation so,

• EA = [Base Reg.] + Addr. field value

Next Inst^{n} Addr. — Target Segment Base Addr. — Relative value

⇓

PC ← [Base Reg.] + relative value.

★★ Note :- PC-relative and Base-Register both of the addressing modes are suitable for program re-location at run time.

★★ Base-Register A.M. is best suit to write the Position-Independent codes.

8085 - not having segmented m/y
so Base-Register A.M. not possible

Position-Independent Codes ⇒ (Sub programs).

Que:- Consider 8 byte long PC-relative jump Inst^n stored in the word addressible m/y. with a word size of 16 bit. Starting Addr. of a Inst^n is $(2345)_{10}$. Address field of a Inst^n contain Signed constant i.e. (-12). Base Register content is $(4863)_{10}$. Index Register content is $(24)_{10}$.

a) what is the Branch Addr.

b) what is the Branch Addr. If Inst^n designed with base Registes Addr. Mode.

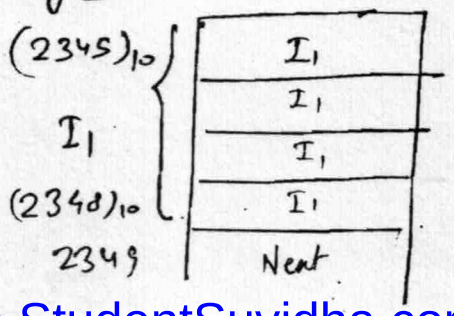
★c) what is the Branch Addr. If Inst^n designed with base-indexed addr mode.

Inst^n size = 8 byte → 4 cells ⇒ 4w

word size = 16 bit = 2 Byte.

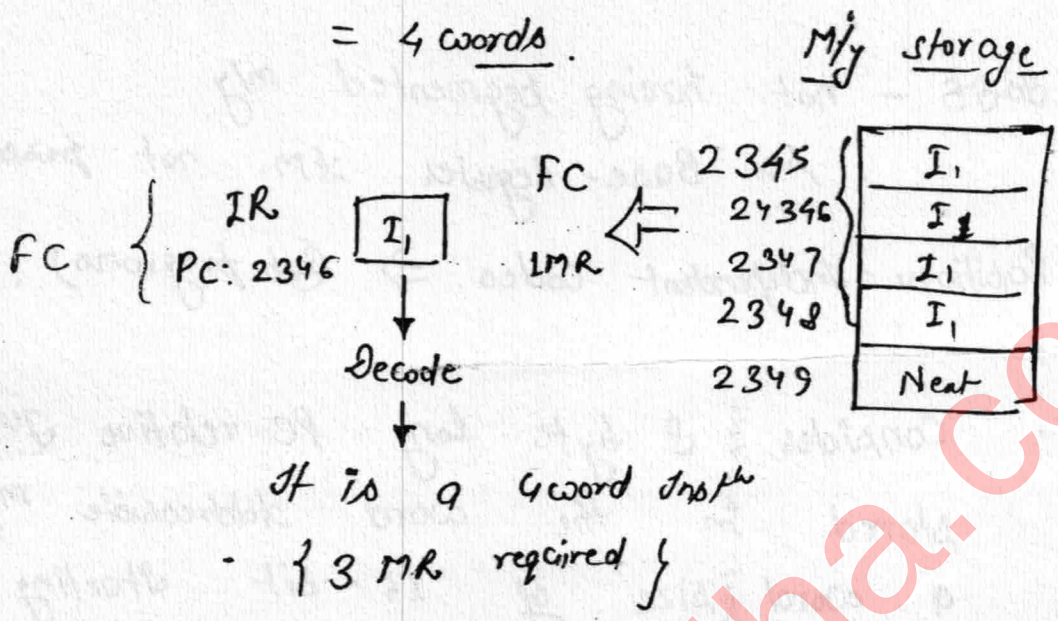
PC = 2345 - 12

= 2333

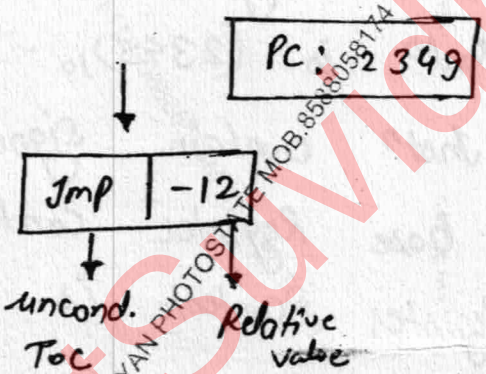


Word size = 16 bit
= 2 B

Instr size = 8 B
= 4 words.



If is a 4 word instr
- {3 MR required}



PC-relative A.M.

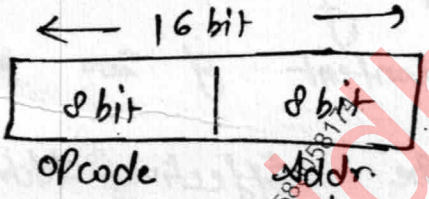
EA = PC + relative value
= 2349 + (-12)
= 2337

PC: ~~2349~~ 2337

(b) E.A = [Base Reg.] + relative value
= (4863) + (-12)
= 4851

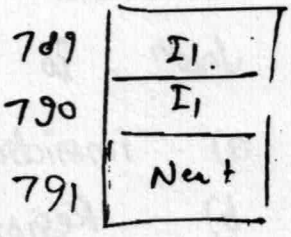
*(c) EA = [Base Reg.] + [Index Reg.] + Displacement
= 4863 + 24 + (-12)
= 4875.

Que:- Consider A Hypothetical Processor which supports instⁿ format with 8 bit opcode and 8 bit Address field, opcode is designed with a Pc- relative jump operation. Instⁿ is Stored in the m/y with a Starting addr. of $(789)_{10}$. during the execution control will be transferred to $(752)_{10}$ location what is the displacement in Hexadecimal ?



Instⁿ size = 2w

Pc: 791



752 = 791 + rel value

752 - 791 = rel. value

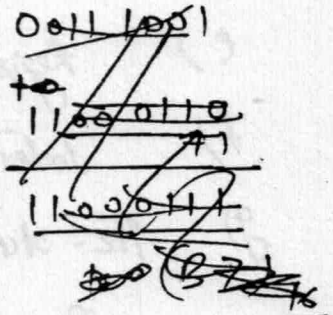
(-39) = rel value

(-39) :- 0001 0011

1101 1000 +1

1101 1001

$(D9)_{16}$

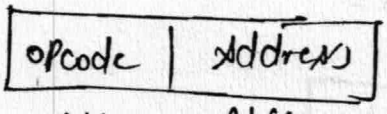


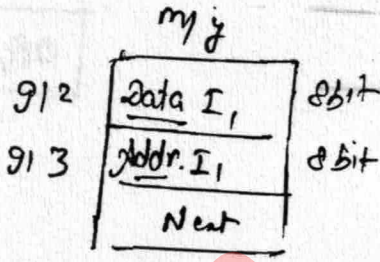
Date
14/08/2017

Que:- Consider a Hypothetical processor which supports instruction format with 2 fields i.e. 8 bit opcode and 8 bit Address. Instruction is stored in the memory with a starting address of $(912)_{10}$. Processor supports 1 Register name as R0. It contain 222 value address field of instruction contain 200 value ~~m/y~~ m/y content of 200 ~~contain~~ is 100.

calculate the Effective Address when the Instn is designed using:

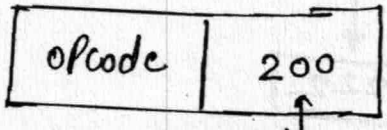
a) immidiate Addressing mode
b) Register Address mode
c) direct A.M.
d) Indirect A.M.
e) Register Indirect A.M.
f) Indexed A.M. with R0 as Index Register
g) Pre-Auto increament Addressing mode with R0 as a base Register and word length is 16 bit.
h) Pc-Relative A.M.
i) Base Register A.M. with R0 as base Register.

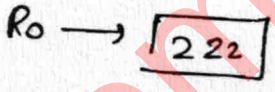




*

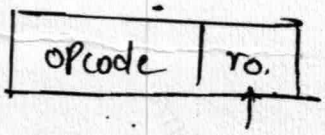
a) Immediate Addressing mode





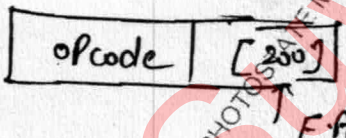
E.A. :- 913 (Data location)

b) Register A.M.



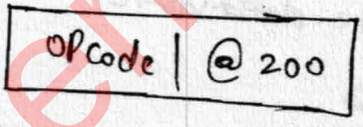
E.A. = [R0] = 222 ⇒ EA. = R0

c) Direct A.M.



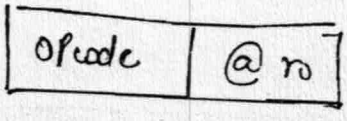
E.A. = [200] = 100 EA. = 200

d) Indirect. A.M.



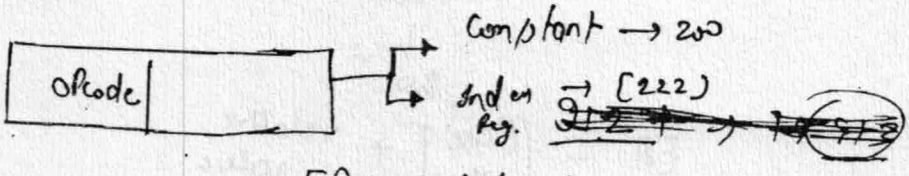
EA. = 100

e) Register Indirect A.M.



EA. = 222

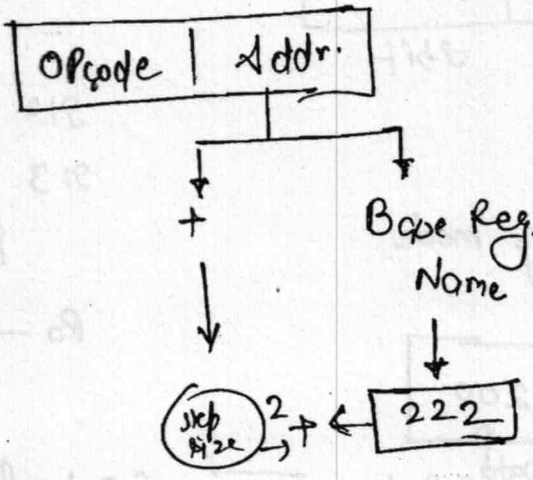
f) Indexed A.M.



E.A. = Index Reg Base Reg

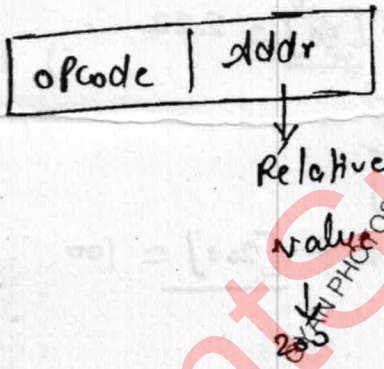
222 + 200 = 422

g) Pre Auto Inc A.M.



E.A. = 224

h) Relative A.M. :-

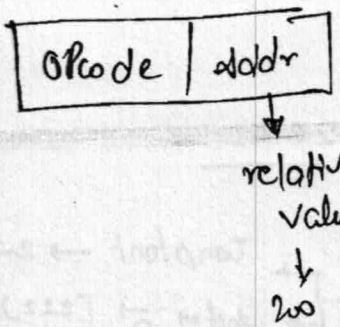


EA. = PC + relative ~~Addr~~ value

= 914 + 200

= 1114

gi) Base A. Reg A.M. :-



EA. = [Base Reg] + relative value

[ra] + 200 = 222 + 200 = 422.

"Instruction Set" :-

(Types of operation
(or)
Types of Instructions)

Every Processor supports three category of the Instructions name as :

① "Data Transfer Instruction" :-

[Mov, Load, Store, Push
Pop, In, Out etc.]

② "Data manipulation Instructions" :-

(2.1) Arithmatic Instruction :-

[ADD, SUB, MUL, DIV,
INC, DEC, etc.]

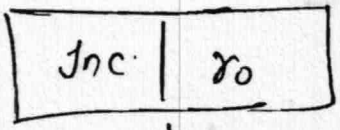
(2.2) Logical Instruction :-

[AND, OR, XOR, ~~Not~~,
CMP etc]

(2.3) Shift & Rotate Instruction :-

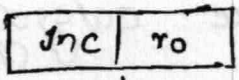
★ Note :- During the execution of Increament Inst^{n} constant '1' is added to a register or memory content. when the Register satisfies the maximum limit then all 1's are rollback to 0 without change the carry flag.

Instⁿ:



$r_0 \leftarrow r_0 + 1$

eg:

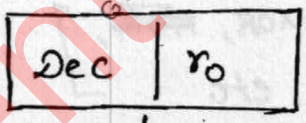


r_0 : FFH

r_0: 1111 1111
+1
r_0: 0000 0000

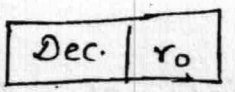
★ During the Execution of decreament instruction Constant '1' is subtracted from the Register (or) memory content. when it satisfies the minimum limit then all '0's are rolled-back to 1's without change the carry flag.

Instⁿ:



$r_0 \leftarrow r_0 - 1$

eg:



r_0 : 00 H

r_0: 0000 0000
−1
r_0: 1111 1111

Borrow : Base

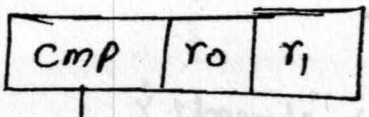
Extra point :- In subtraction :- carry discarded
In addition :- carry counting
} difference in Sub & Add

:- **CMP :-** This Instⁿ is used to compare the 2nd operand with 1st operand.

★ It invokes subtraction operation during the execution. ~~His~~

★ This Instruction execution status is maintained in the zero and carry flag of a Flag Registes.

Eg:



; r_0 : 80H
r_1 : 80H

Sub

$(r_0 - r_1)$

r_0 : 1000 0000

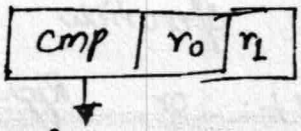
r_1 : 1000 0000

0000 0000

cy = 0, z = 1

{ operand 2 = operand 1 }

eg:



r_0 = 80H
r_1 = 40H

Sub

$(r_0 - r_1)$

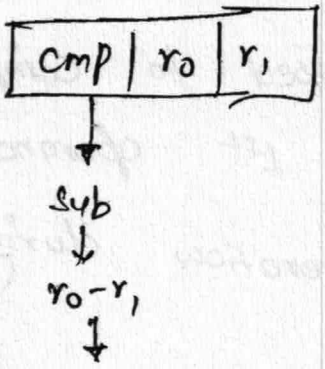
r_0 : 1000 0000

r_1 : 0100 0000

0100 0000

cy = 0, z = 0 { operand 2 < operand 1 }

eg:



$r_0 = 40H$

$r_1 = 80H$

r_0 : 0100 0000

r_1 : 1000 0000

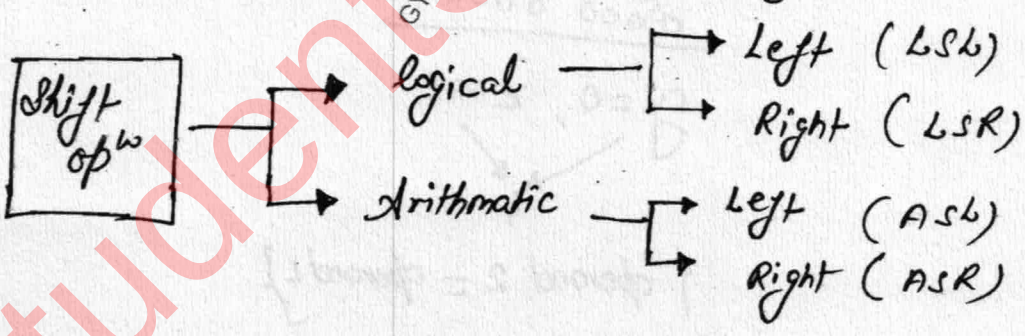
1100 0000

$y = 1$ $\&Z = 0$

{operand 2 > operand 1}

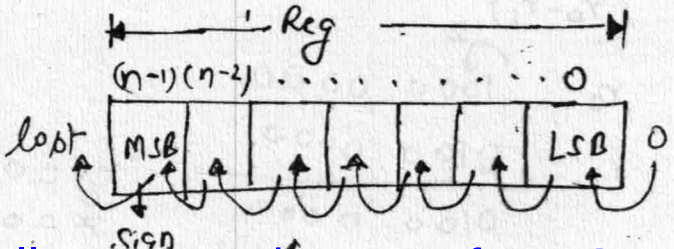
Shift :-

* while execution of this instruction, data bits are moving to left or Right direction. in a bit wise sequence with loss of data.

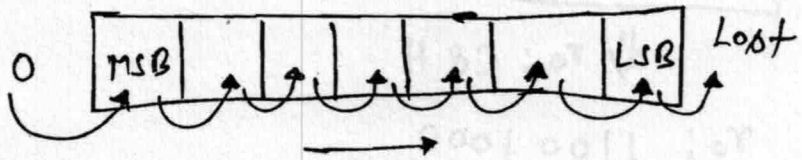


* In the Logical Shift Operation data bits are moving to left or Right direction bit by bit including sign bit.

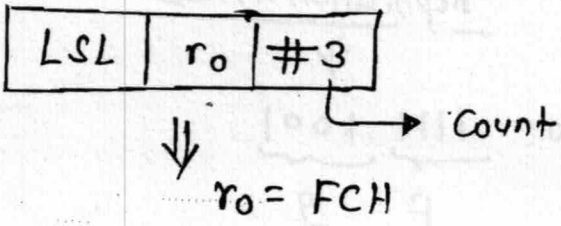
LSL :-



LSR :-

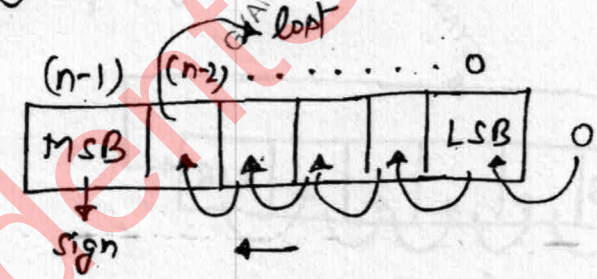


Eg :-

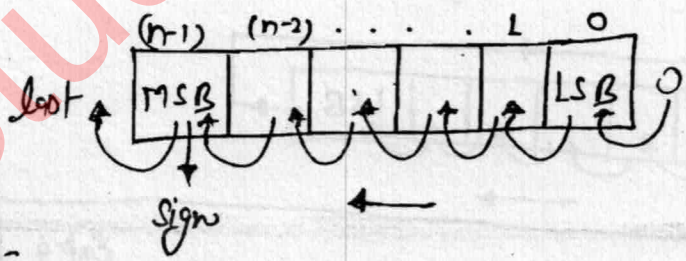


* In Arithmatic shift operation Data bits are moving to left or Right direction except the sign bit.

ASL :



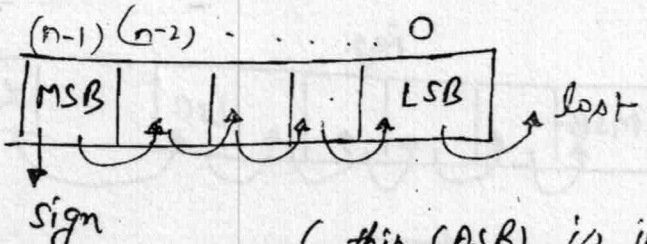
; conceptwise.



, Implementation wise

{ ASL = LSL }

ASR :-



(MSB will not lost.

(this (ASR) is in Practice } not padding zero).

eg:

ASR	r_0	#3

$\Downarrow$ r_0 : C8 H

r_0 : 1100 1000

Replication of MSB

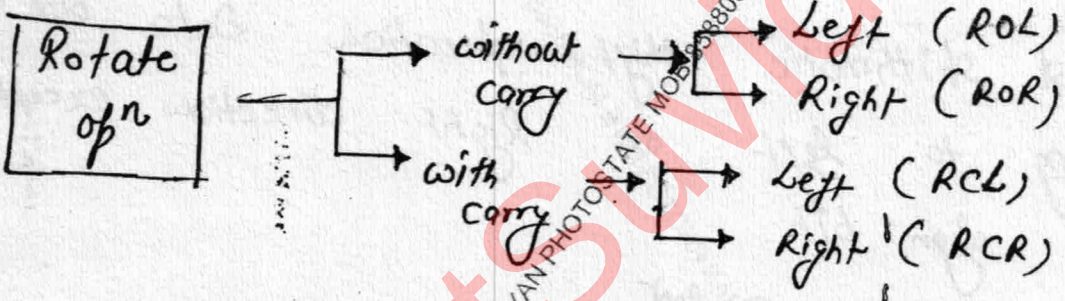
$\Downarrow$

r_0 : 1111 1001

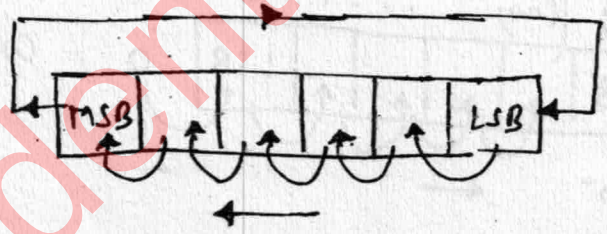
F 9

**Rotate :-

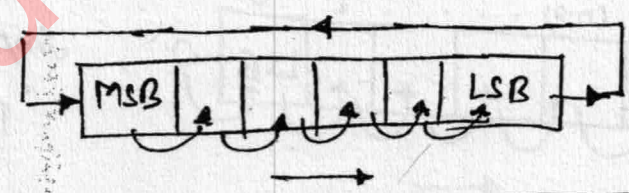
* while Execution of this instⁿ, Data bits are moving to left or Right direction bit by bit without Data loss.



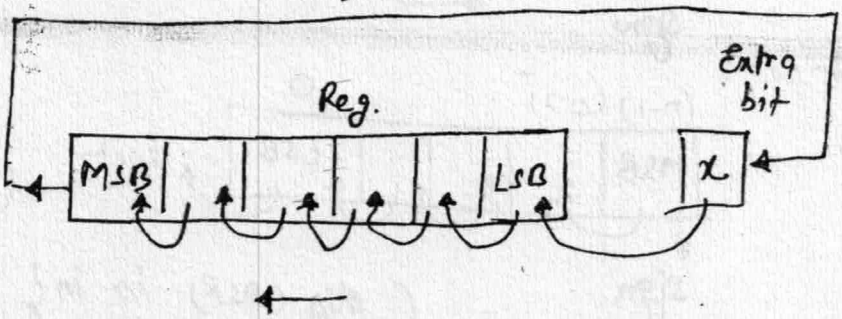
ROL :-



ROR :-

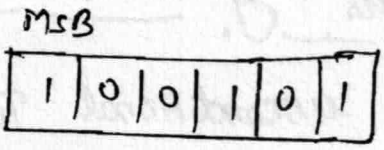


RCL :-

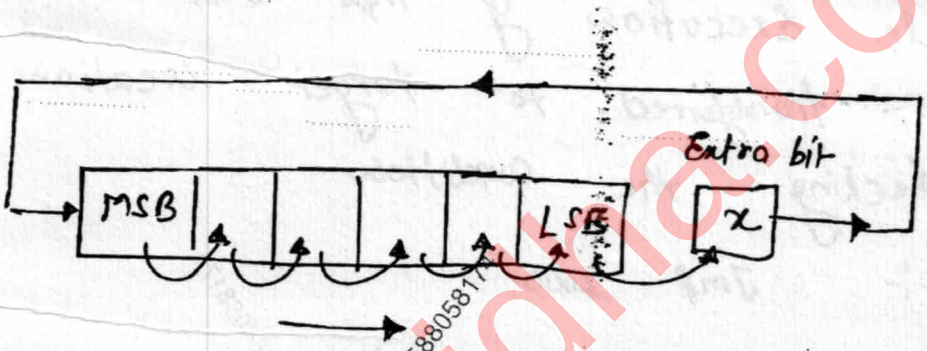


If 6 time op^{n} :-

	Carry
1st time :	Set
2nd time :	Reset
3rd time :	reset
4th time :	set
5th time :	Reset
6th time :	Set



RCR :-



Eg :-

Initial :

	Reg	extra bit	Carry flag
	1 0 0 1	x	
1st time :	x 1 0 0	1	Set
2nd time :	1 x 1 0	0	Reset
3rd time :	0 1 x 1	0	Reset
4th time :	0 0 1 x	1	Set

∴ Apply one more RCR to get the initial data

1	0	0	1	x

~~Data Retrieved~~

③ Transfer of control Instⁿ :-

③.1 unconditional ToC

③.2 conditional ToC

"unconditional ToC" :-

★ while Execution of this Instruction, Control will be transferred to target location without checking the condition.

Ex :- Jmp 2000

Program :-

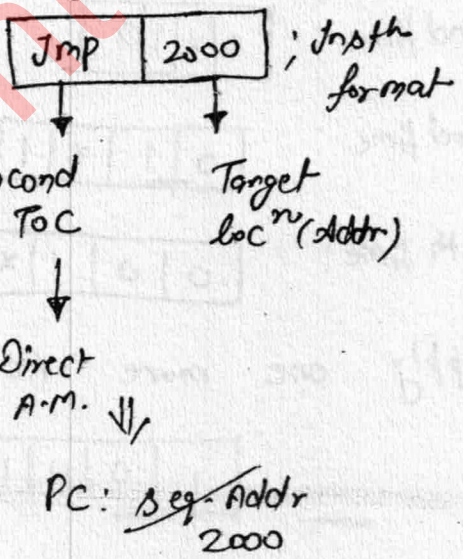
Instⁿ : Jmp 2000 ; memory

↓

Jmp 2000 ; CPU

PC : seg Addr. } F.C.

↓



★ This Instⁿ is used to implement the uncond. selection statement (goto) and various machine control Instructions (Halt).

HALT :-

★ It is the machine control instruction.

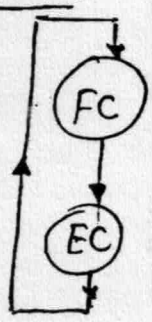
★ It invokes the unconditional jump during its execution

★ Target address of the jump is starting address of a Halt instruction

★ During the Execution of the HALT, CPU enters into a Infinite Loop, so user point of view

● Program execution is suspended. (Stopped)

Halt :-



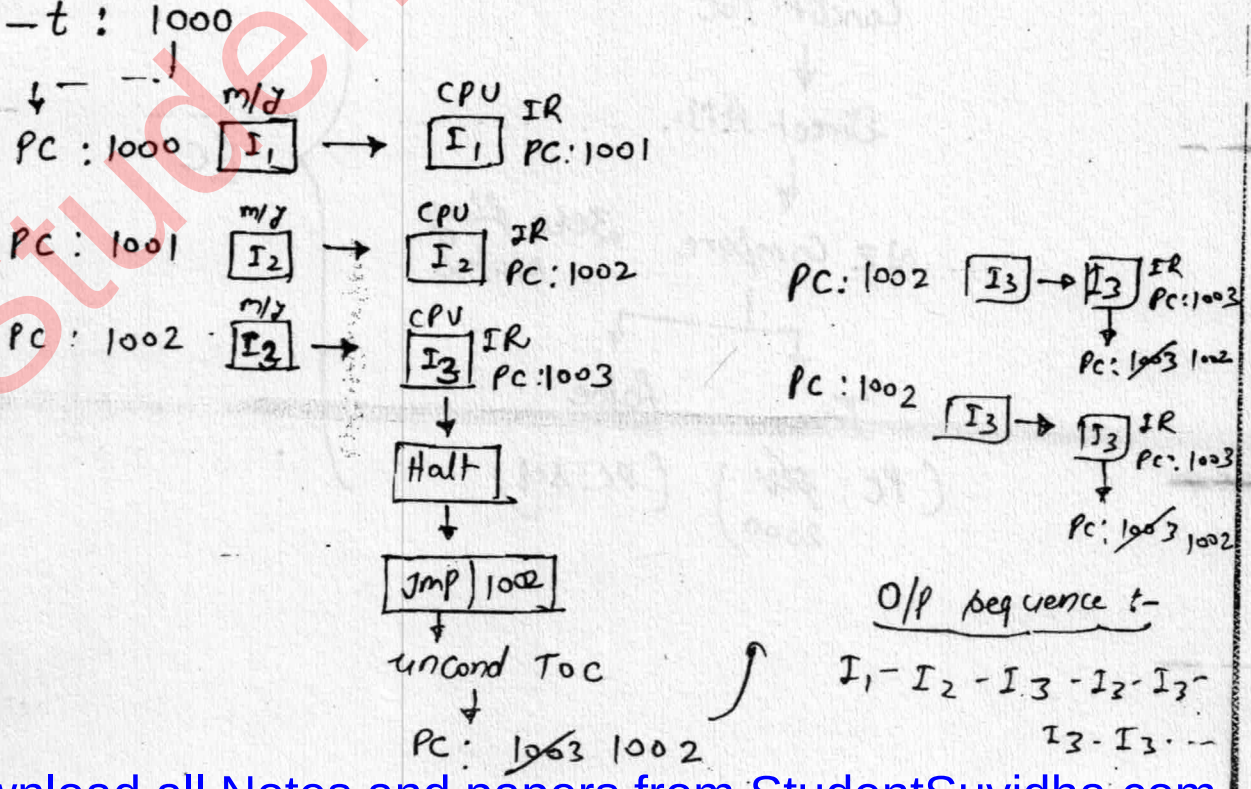
HALT :

```
li: Jmp li
```

Program :-

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

Execution :-



O/p sequence :-

$I_1 - I_2 - I_3 - I_3 - I_3 - I_3 - I_3 \cdots$

⇒ "Conditional ToC" :-

* while Execution of this instruction, Associated condition is evaluated based on the status of a previous instruction.
* when the condition is true, then control will be transferred to target location, otherwise sequential instruction in the memory will be executed.

Eg :- JNZ 2000

Program :-