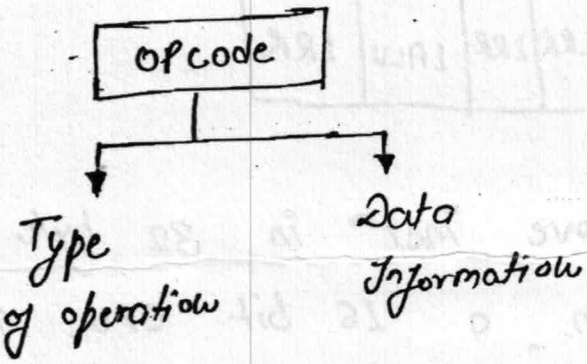
② Implied or Implicit Addressing Mode :-

:- In this Mode; data information is present in the opcode itself.

Instⁿ Design :



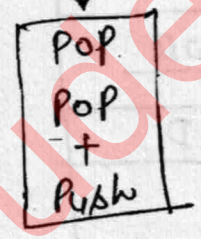
eg: STC ; Set carry

CY = 1

eg: CLC ; clear memory carry

CY = 0

eg: ADD ; on stack CPU

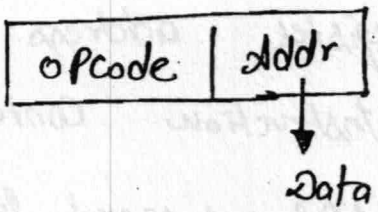


*Note :- All the zero address instructions are designed with a Implied addressing mode.

③ Immediate Addressing Mode :-

:- This mode is used to 'access the constants'. in this mode data is present in address field of a Instⁿ.

Inst^{n} Design :



★ Limitation is Range of constants are always restricted by the size of a Address field.

eg: n-bit Address field supports :

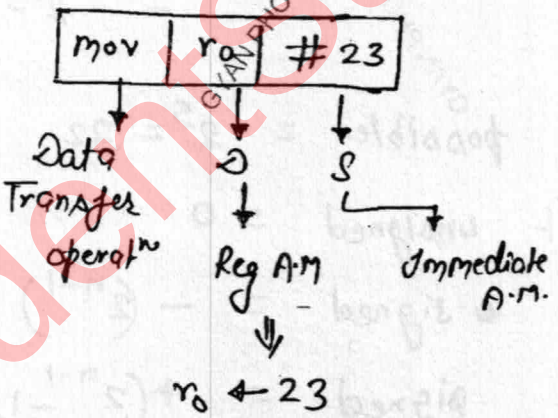
$\{0 - (2^n - 1)\}$ range of unsigned constant

(or)

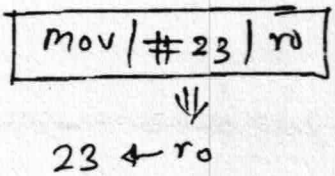
$\{-(2^{n-1}) \text{ to } +(2^{n-1} - 1)\}$ range of signed constant.

★ Immediate Addressing mode is never used as a Destination Addressing mode because constant doesn't have the storage capability.

eg:



eg:



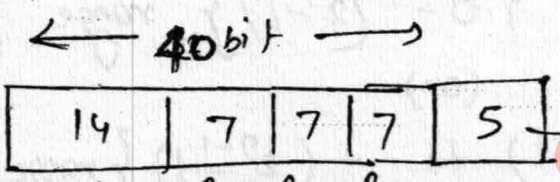
; Invalid

★★

Que: Consider a Hypothetical processor Interfaced with a 5 cells of byte addressible m/y which supports 1 word Long Inst^{n}. Inst^{n}

designed with 3 register address field and immediate field. Instruction Contain 14 bit opcode and CPU supports 128, 1 word long Registers. what is the largest unsigned constant possible in the instth : ?

coord size = 40 bit = Inst^{n} size



$2^5 - 1 = 31$

$\{2^5\}$

Range :- $\{0 - 31\}$

max unsigned constant = $\{0 - (2^n - 1)\}$

$= 2^5 - 1 = (31)$ Ans

Constant possible = $2^5 = 32$

smallest unsigned = 0

smallest signed = $-(2^{n-1}) = -16$

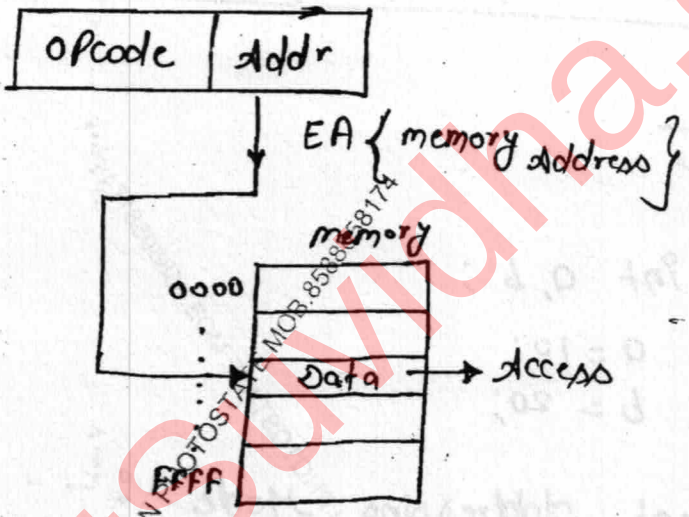
Largest signed = $+(2^{n-1} - 1) = +15$

④ Direct or Absolute addressing mode :-

:- This mode is used to access the Static variables.

★ In this mode Data is present in the memory, corresponding memory Address will be maintained in the address field of a instruction as a effective address.

Instn Design :

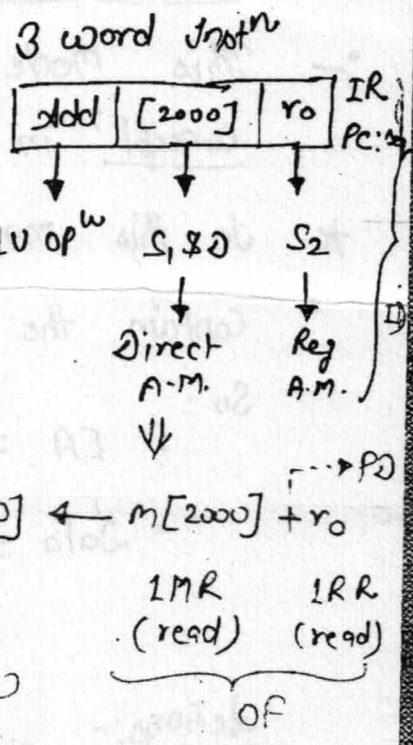


- EA = address field value
  ↓
  memory Addr
- Data = [EA]
  = [memory address]

Actions :-

IMR → To access the Data

-eg:-



* If Instⁿ size is not given, by default 1 word.

IC

FC	EC				
IF	ID	OF		PD	WB
		S_1	S_2		D
1MR	2MR	1MR	1RR	1 ALU	1MR

FC = 1MR
EC: 4MR
IC: 5MR

Analysis :-

Code :

```
int a, b;
a = 10;
b = 20;
```

(5) Indirect Addressing Mode :-

:- This mode is used to 'impliment the Pointer concept' in the base Hardware

* In this mode, Address field of a Instruction Contain the address of a Effective address.

So:

- EA = [Addr. field Name]
- Data = [EA]

$$= [[\underbrace{\text{Addr. field value}}_{EA}]]_{Data}$$

Actions:-

1st Ref. → To get the 'EA'
2nd Ref. → To Access the 'Data'

* Baped on the Avalability of a Effective Address, Indirect mode is two type :

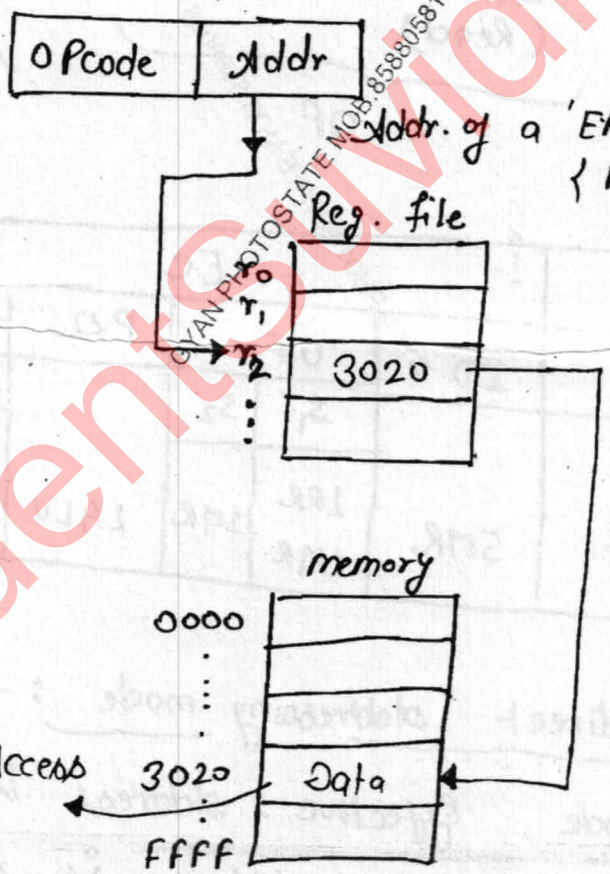
① Registes Indirect A.M.

② Memory Indirect A.M. (By default)

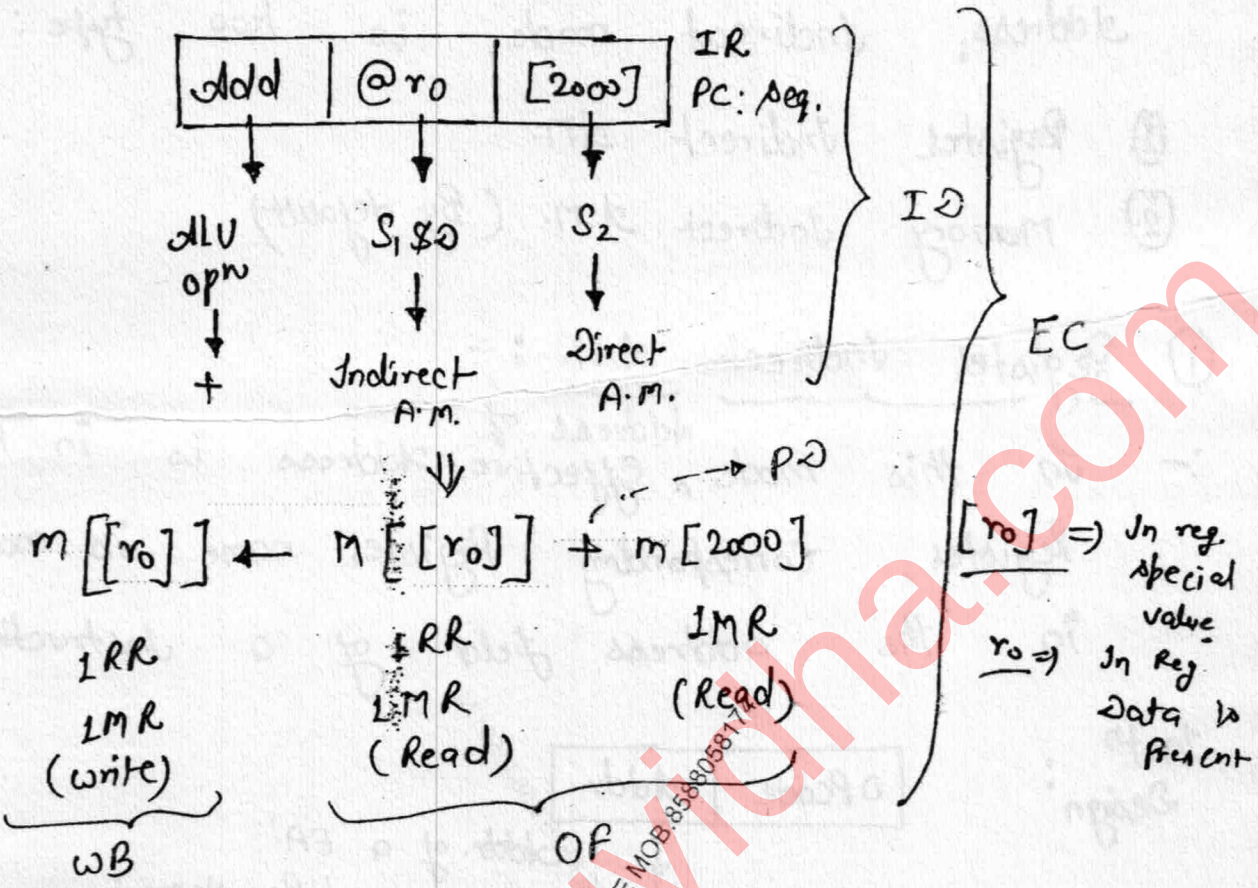
① Registes Indirect A.M. :-

:- In this mode ∧ effective Address is in the Registes, corresponding Registes name is maintained in the address field of a Instruction.

Instn Design:



eg: 6 word Instⁿ



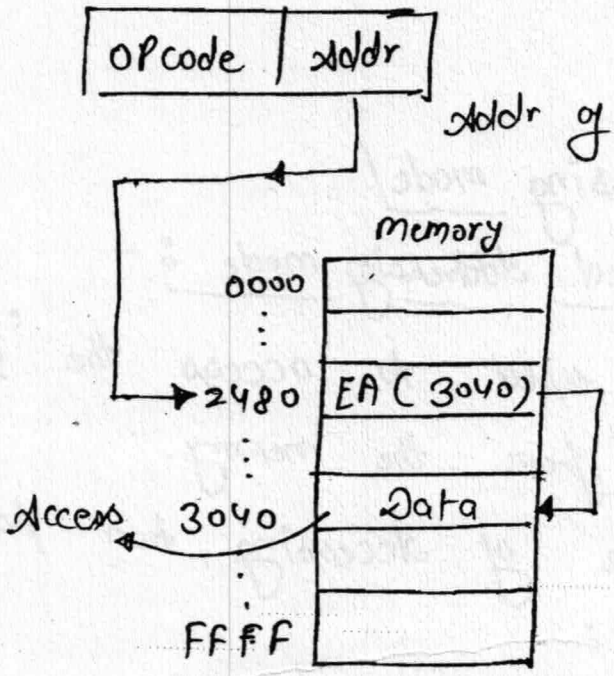
Ic:

FC	EC			
IF	ID	OF	PD	WB
		S1 S2		D
1MR	5MR	1RR 1MR / 1MR	1ALU	1RR 1MR

② Memory Indirect addressing mode :-

:- In this mode, Effective address is in the m/y, the corresponding m/y address is maintained in the address field of a instruction.

Instn Design :



~~Actions~~ :-

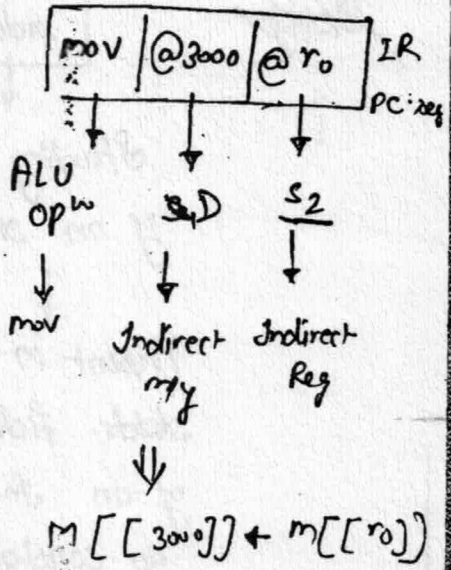
- EA = [Addr. field Name]
  ↓
  m/y address
- Data = [EA]
  = [[m/y addr]]

Actions :-

IMR : → To get the EA

IMR : → To get (access) the Data.

eg: 5 word instn



IC:

FC		EC		
IF	ID	OF	PD	WB
		S		D
1MR	4MR	1RR 1MR	—	2MR

⇒ 1RR 8MR

Date
13/08/2017

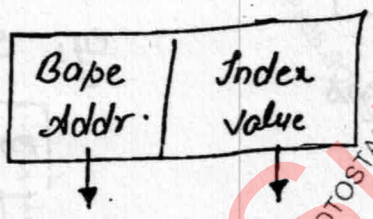
Indexed Addressing mode/

Base / Indexed Addressing mode :-

:- This mode is used to access the 'Random Array Elements' from the memory.
In this style of accessing, two parameters are required :

1) Base Address
2) Index value.

Inst Design :



Base Addr. → Starting Addr of an Array → Present in the Addr. field of an Inst^n as constant

Index value → Position of the Array element → Present in the Index Register

★ In this mode effective address is obtained by adding the constant to the content of the Index Register.

- EA = Addr. field value + [Ri]
- Data = [EA]

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

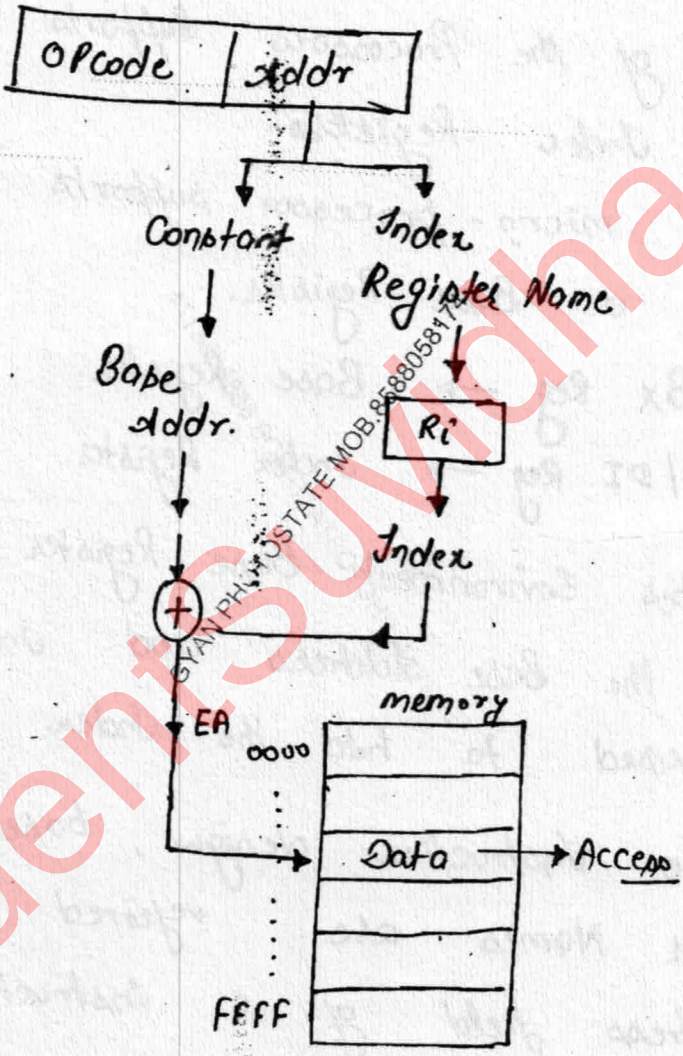
Actions:

1 RR → To get the 'Index'.

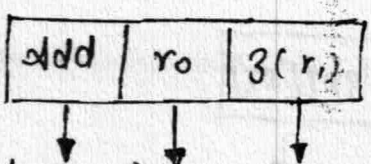
1 Arithmatic → To calculate the 'E.A.'

1 MR → To Access the 'Data.'

Instn Design :



eg:



; r_1 is Index Register.

ALU op → +

S_1 & D → Reg A.M.

S_2 → Index A.M.

$\Rightarrow r_0 \leftarrow r_0 + m[3 + [r_1]]$

1RR (write), 1RR (read), 1RR, 1ALU, 1MR (read)

IC :-

By default Instⁿ size = 1 word.

FC	EC				
IF	ID	OF		PD	WB
		S_1	S_2		D
1MR	—	1RR	1RR 1ALU 1MR	1ALU	1RR

Note :- Some of the Processors supports both base and Index Registers.

Ex: 8086 micro-processor supports 'Bx' register as a Base Register.

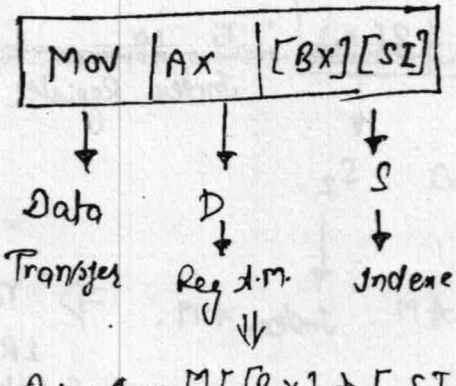
Bx Reg → Base Register

SI/DI Reg → Index Register.

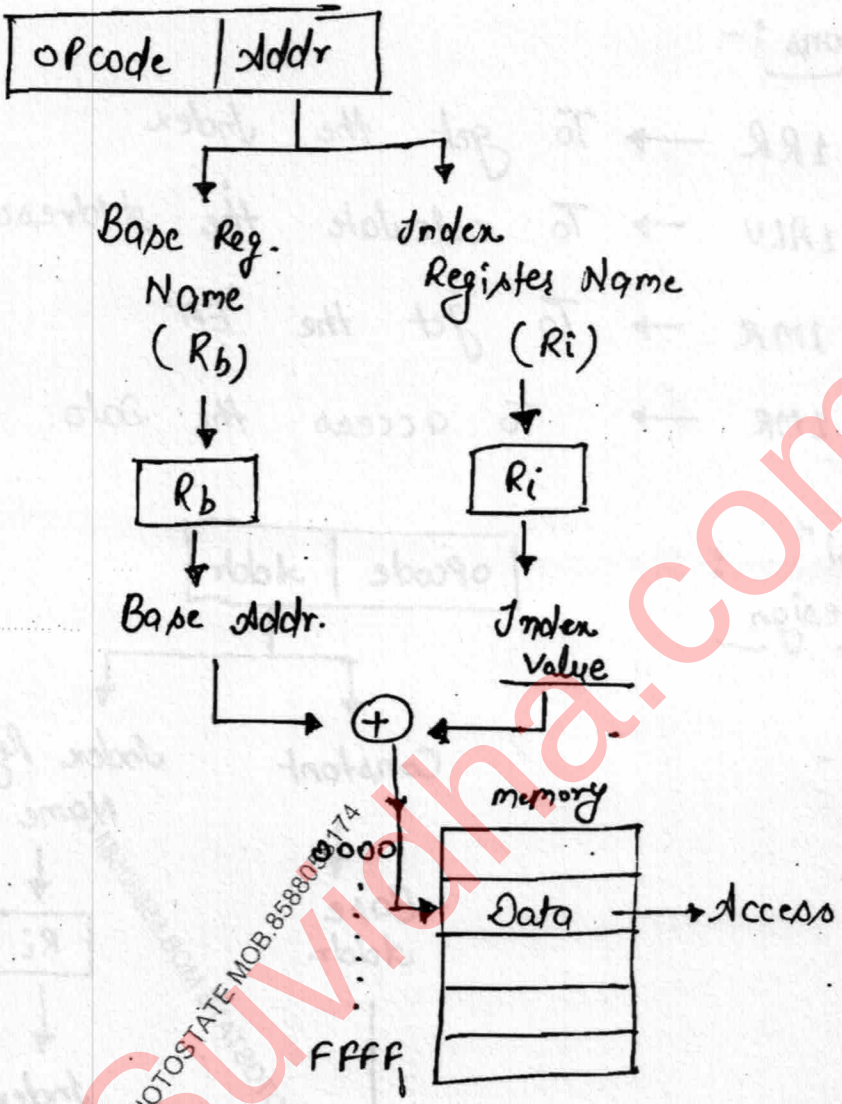
★ In this Environment Base Register is used to hold the Base address and Index Register is used to hold the Index value.

★ In the Instruction design, base & Index Register Names are referred in the address field of a Instruction.

eg:- 8086 MP :-



Instⁿ design :



Indirect Indexed Addressing Mode :-

:- In this mode Effective Address is present in the memory, the corresponding memory address is calculated by adding the index value to base address. So,

- $EA = [\text{Addr. field value} + [Ri]]$
- $\text{Data} = [EA]$
  $= [[\text{Addr. field value} + [Ri]]]$

Actions :-

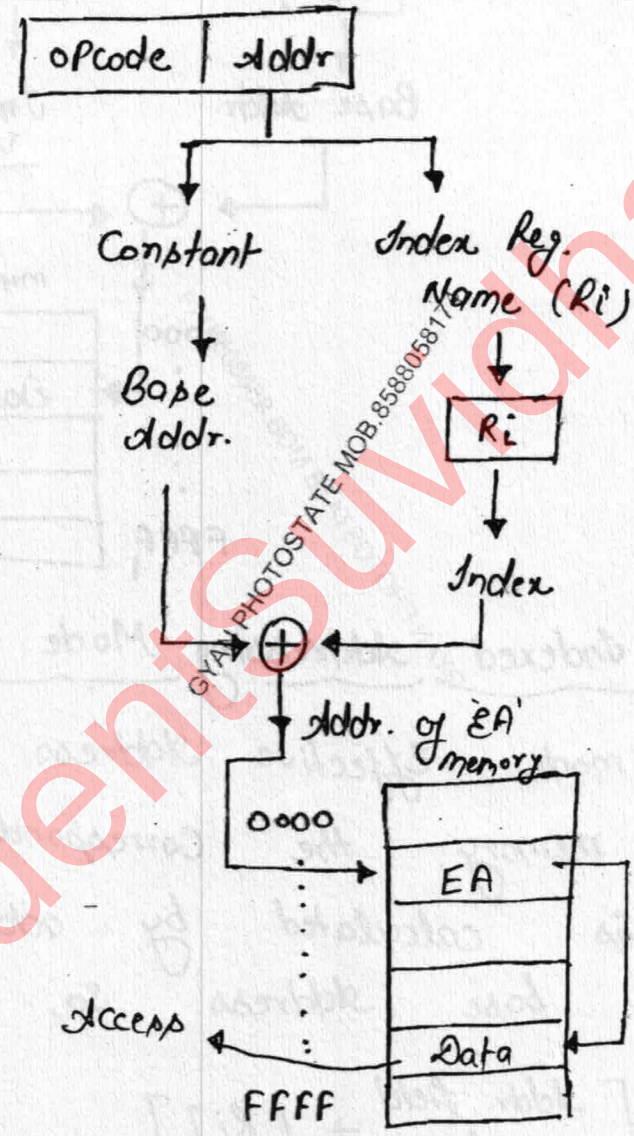
IRR → To get the Index

IALU → To calculate the address of EA.

IMR → To get the 'EA'

IMR → To access the Data.

Instⁿ Design :



* This mode takes more cycles, to Access the Data, therefore this mode is not in the practice

Auto Indexed Addressing mode :-

:- 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'

Base Address

↓

Starting / Ending Address of an Array

↓

Present in the Base Register (R_b)

★ In this mode Effective Address is obtained either by increamenting or decreamenting the base Register content with a 'Step size'

★ Step-size is a fixed constant depends on the word-length of a CPU.

- $EA = [R_b]$ (+/−) Step size
- Data = [EA]
  $= [[R_b]$ (+/−) Step size $]$

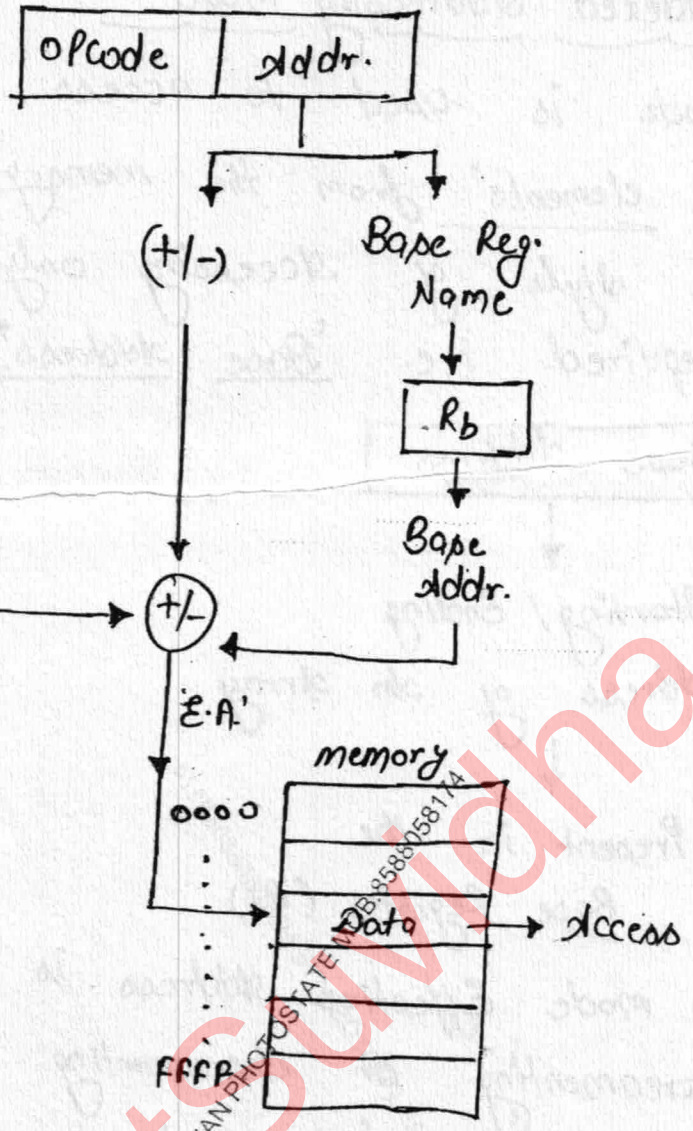
Actions :

IRR → To get the Base addr.

LALU → To calculate the 'EA'

IMR → To access the Data

Instn Size :-



Auto Indexed addressing Mode

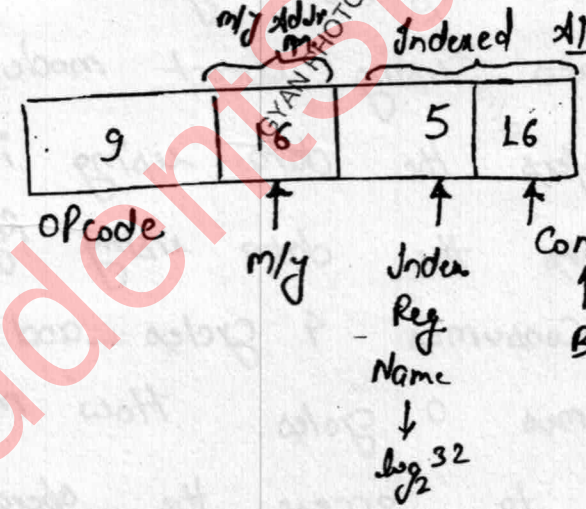
- Auto Inc. A.M.
  - Pre Auto Inc. AM. eg: Mov r0, +(r1);
  - Post Auto Inc. AM, eg: Mov. r0, (r1)+;
- Auto Dec. AM.
  - Pre Auto Dec. AM. eg: Mov r0, -(r1);
  - Post Auto Dec AM. eg: Mov r0, (r1)-;

:- where 'r1' is a Base Register

Que:- Consider a Hypothetical CPU, which supports the 16 bit main memory address space. Instn is designed with 2 m/y operands. operand 1 uses Direct Addressing mode and operand 2 uses Indexed Addressing mode. Addressing mode Information is implemented in the opcode itself. CPU Instn set size is 400. Register Bank size is 32 including index register. How many bits are required to encode the instructions.

Instn set size = 400 = 9 bits

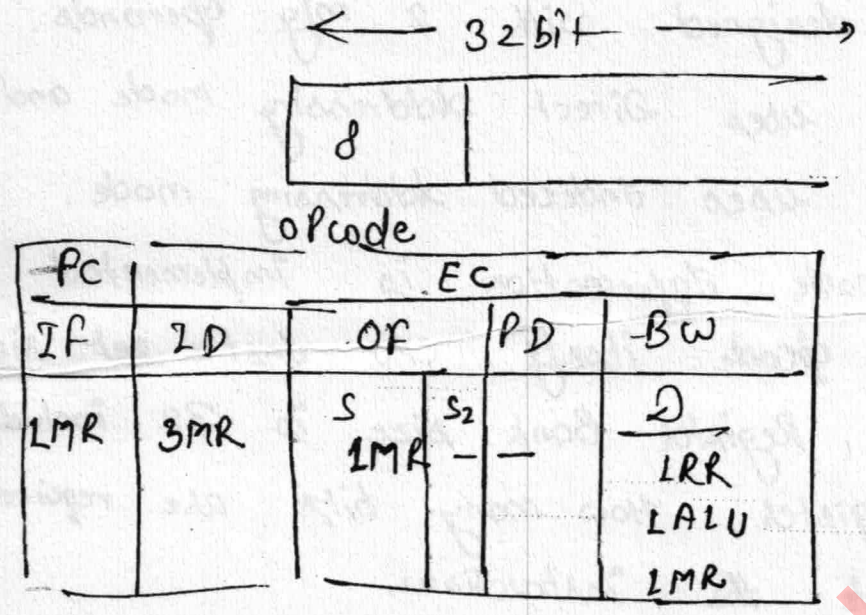
2 m/y operand = 16 bit + 16 bit



Inst. = 46 bits

Que:- Conside 4 byte long Instn running on a 8 bit Hypothetical CPU. Instn contain 1 word opcode which tells the type of operation as a data Transfer and source addressing mode is a Direct and Destination A.M is a Indexed.

How many # of m/y references are required to complete the instn.



Que:- ~~Cons~~ A Hypothetical processor uses different operand accessing modes i.e. Register, Direct and Indirect. In the Program 70 instn refers the data using direct mode, 40 instn refers the data using indirect mode. & 50 instn refers the data using Register. m/y reference Consumes 4 cycles and Register Reference consumes 0 cycles. How many cycles are required to access the operand during the program execution.

LMR = 4 cycles

70 - direct → 70 MR → 280 cycle

40 - Indirect → 80 MR → 320 cycles

50 - Registers → 0 MR / 1RR → 0 cycles.

600 cycles

**Que:- A Hypothetical processor which is operating with a 500 MHz clock frequency uses the different operand accessing modes:

Operand Access mode	frequency (%)	
Reg	30	→ RR
Direct	22	→ 1MR = 22MR
Indirect	17	→ 2MR = 34MR
Reg. Indirect	11	→ 1RR, 1MR = 11MR
Indexed	20	→ 1RR, 1AL, 1MR = 20MR + 20ALU

Assume that 4 cycles consumed for m/y reference, 2 cycles consumed for ALU op^{w} and 0 cycle consumed for Register Reference. what is the Avg operand fetch Rate of the processor in million words per seconds.

$$f = \frac{1}{T} \Rightarrow T = \frac{1}{f} = \frac{1}{500 \times 10^6}$$

T = cycles/sec

f = cycles/sec

$$\text{cycles} = \frac{22\times4 + 34\times4 + 11\times4 + 20\times4 + 90\times2}{100}$$

$$= \frac{88 + 136 + 44 + 80 + 40}{100}$$

$$= \frac{388}{100} = 3.88 \text{ cycles}$$

(percentage → 100)

$$= 3.88 \times 500 \times 10^6 \text{ (Bmillion)} = 1940 \text{ million words/sec}$$

Avg = $\frac{\text{Sum}}{\text{Count}}$ Count = 100.

$\therefore$ Avg = Sum

Avg of time = $[(0.3)*0 + (0.22*4) + (0.17*8) + (0.11*4) + (0.2*6)]$

= 3.88 cycles

Cycles time $\propto \frac{1}{\text{Clock freq}}$

cycle time = $\frac{1}{500 \text{ MHz}}$

= 2 ns

Avg. of time = (3.88 * 2ns)

= 7.76 ns

1 operand → 7.76 ns

operands (words) → 1 sec

$\Rightarrow \frac{1 \text{ operand}}{7.76 \text{ ns}}$

$\Rightarrow \frac{1}{7.76 \times 10^{-9}}$ words / sec.

$\Rightarrow \frac{1}{7.76} * 10^9$ words/sec

= 128.8 million words / sec.