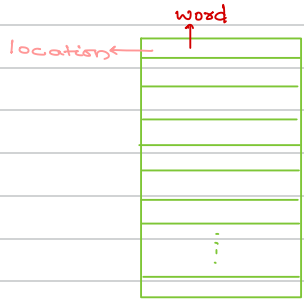


memory is represented as ladder like struc.

$\log_2 n$ (formula)

Eg:- for 2 memory location

for 4 memory location

$1024 \rightarrow 2^{10} \rightarrow 1\ KB$

$1024 \times 1024 \rightarrow 2^{20} \rightarrow 1\ MB$

$\rightarrow 2^{30} \rightarrow 1\ GB$

$\rightarrow 2^{40} \rightarrow 1\ TB$

Q- Find out memory location & address size of 8 GB

Memory location :- $8 \times \underbrace{1024}_{2^{10}} \times \underbrace{1024}_{2^{10}} \times \underbrace{1024}_{2^{10}}$ ($8 = 2^3$)

Address size :- $= 8 \times 1\ GB$

$= 2^3 \times 2^{30}$

$= 2^{33}$

Q- If Address size is 32 . How many GB.

$= 32$

$= 2^{32} = 2^{30} \times 2^2 = 2^{30} \times 4 = 4\ GB$

The amount of bits stored under a unique address is known as word.

or

In a single reference to memory , the amount of bits received is called a word.

Generally word size varies from 1 byte ... 2 byte ... 3 byte ... 8 byte ....

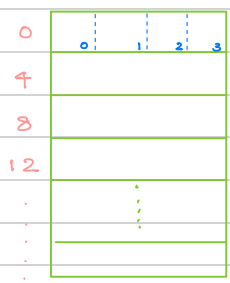
1 Byte = 8 bits

2 Byte = 16 bits

4 Byte = 32 bits → commonly used

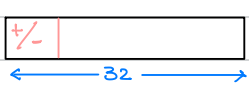
8 Byte = 64 bits → laptops

Byte Addressable Memory - If there is a unique address for every byte of information, then it is called a Byte Addressable Memory.



byte-addressable memory

To store integer :-

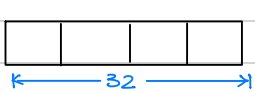


+ → 0

- → 1

rest 31 are to store magnitude.

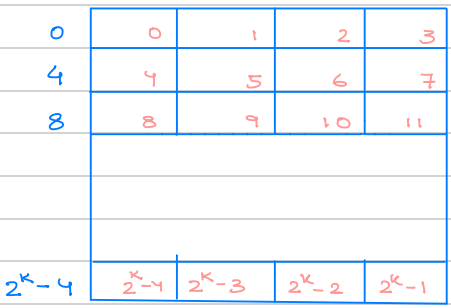
character



*normally data is stored in 2's complement form

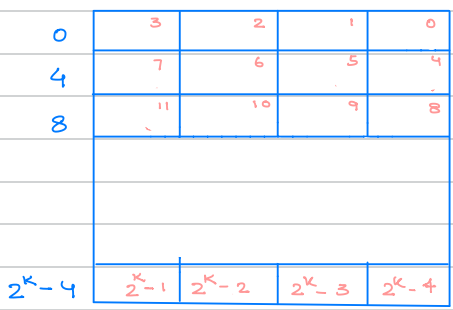
Byte Addressable Memory is of 2 types :-

① Big Endian Notation



M O H I T R A N J A N

② Little Endian Notation



I H O M A R T N A J N

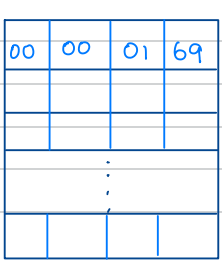
Store +361 , -519 in Byte Addressable.

361 → 101101001

32 bit

00000000 00000000 00000001 01101001

in positive nos there is no 2's complement.



Big Indane (32 bit)

make group of 8 ;

convert in hexadecimal

00000000 00000000 00000001 01101001

0 0 0 0 0 1 6 9

16 bit

Big Indane

00	00
01	69
:	

-519

1 00 00 00 111

1 011 11 11 001 — 2's Complement

make group of 8. in 32 bits

Assembly Language Instruction & Commands :-

In general assembly language instruction perform the foll^{n} operations -:

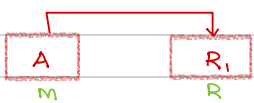
(i) Memory Interaction Instruction

(ii) ALU Instruction

(iii) Branch / Loop Control Instructions

(iv) Input - Output device interaction Instruction

LOAD A, R1



STORE R1, A

MOVE R1, R2

M → Memory

R → Registers

——— → opcode

——— → operands

Types of A

→ can be classified into 4 types based on way it handles the operands :-

(i) 3-Address Instruction

opcode | Source 1 | Source 2 | destination

ADD R1, R2, R3

[R3] ← [R1] + [R2]

ADD A, B, C

[C] ← [A] + [B]

outer one is always destination

R3 & C are destination

$X = (A + B) * (C + D)$ → made for 8085 register

```
ADD  A  B  R1
ADD  C  D  R2
MUL  R1 R2 X
```

} not a good method

correct method :-

LOAD A, R1	[R1] ← [A]
ADD B, R1, R2	[R2] ← [B] + [R1]
LOAD C, R3	[R3] ← [C]
ADD D, R3, R4	[R4] ← [D] + [R3]
MUL R2, R4, R5	[R5] ← [R2] * [R4]
STORE R5, X	[X] ← [R5]

(ii) 2- Address Instruction

opcode source 1 source 2 / destination

↳ initially taken as source & later as destination

ADD R1 R2

[R2] ← [R1] + [R2]

$X = (A+B) * (C+D)$

LOAD A, R1	[R1] ← [A]
ADD B, R1	[R1] ← [B] + [R1]
LOAD C, R2	[R2] ← [C]
ADD D, R2	[R2] ← [D] + [R2]
MUL R1, R2	[R2] ← [R1] * [R2]
STORE R2, X	[X] ← [R3]

(iii) 1-Address Instruction :-

In these instructions only one operand is mentioned along with the opcode.

But as we know atleast 2 operands are reqd. to perform ALU operation ; here the processor uses the default resistor which also holds the result.

This default resistor is known as ACCUMULATOR.
(denoted by A or AC)

ADD B	[AC] ← [B] + [AC]
STORE B	[B] ← [AC]
LOAD B	[AC] ← [B]

(A + B) * (C + D)

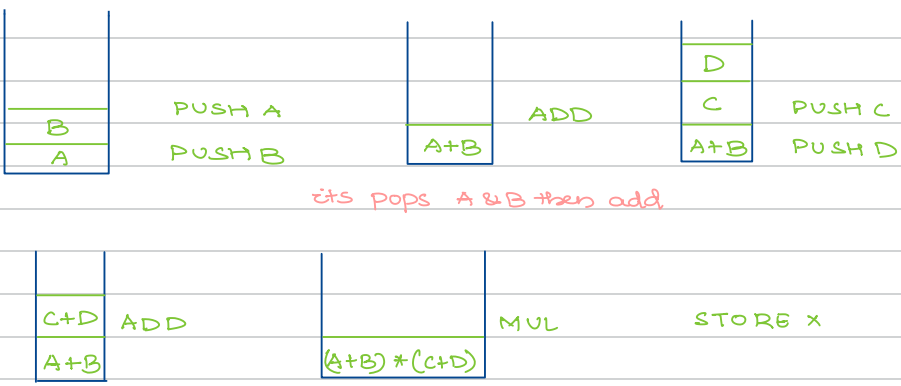
LOAD A	[AC] ← [A]
ADD B	[AC] ← [AC] + [B]
MOVE R1	[R1] ← [AC]
LOAD C	[AC] ← [C]
ADD D	[AC] ← [AC] + [D]
MUL R1	[AC] ← [AC] * [R1]
STORE X	[X] ← [AC]

Accumulator is a Special Purpose Register used as a default register in Assembly Language Instruction.

(iv) 0-Address Instructions :-

Stack is used

$x = (A + B) * (C + D)$



$x = (A - (B/C) * D) - (P/Q * (R + M) + N/P)$

apply BODMAS Rule

in zero address ; convert it to postfix.

Types of ISA :-

(I) Memory - Memory ISA

ADD A, B

(II) Resistor - Resistor ISA

operands are always there in GPR.

ADD R1, R2

(III) Accumulator based ISA

one operand is stored in Accumulator by default.

ADD R1

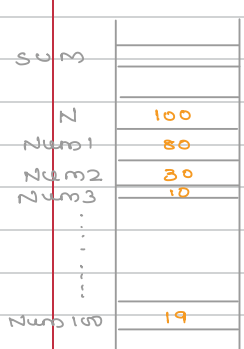
(IV) Stack based ISA

* stack is used
* Zero - Address Instructions

* Very Imp Addressing Modes :-

Instruction Execution

(1) Single Line Sequencing
(2) Branching (loop)



Single line

CLEAR R0 [R0] ← 0
ADD NUM1, R0
ADD NUM2, R0
.
.
.
STORE R0, SUM

Branching

LOAD N, R0
CLEAR R1
Loop
add NUM1 with R1
Change to NUM2
DECREMENT R0
DECREMENT R0
BRANCH NEZ GOTO LOOP
STORE R1, SUM

NEZ → Not equal to zero.

Flag (Condition Code Flag)

* is a one bit information used to keep track of an arithmetical instruction.

→ Multiple such flags are kept under one register called status register.

(i) C → Carry

1 → flag 0
0 → flag 1

```
  10101
+ 10001
-------
 100110
(1) → carry
```

(ii) V → Overflow

1 → flag 1
0 → flag 0

```
  10010101
+ 10010001
----------
 100100110
```

↙ Overflow V = 1

there is a carry but we need 2 registers of 8 bit.

(iii) Z → Zero

0 → flag 1
x → flag 0

(iv) N → Negative

-ve → flag 1
+ve / 0 → flag 0

imp *

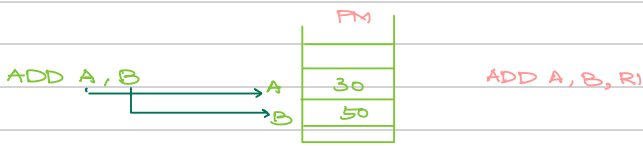
Addressing Modes

→ Addressing Modes refers to the way in which the operands of an instruction are to be specified. (how to deal with operands)

Types :-

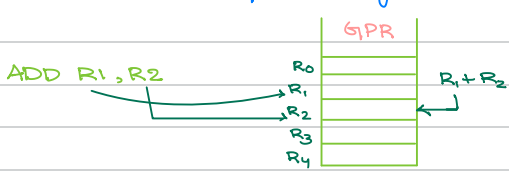
(i) Direct Addressing Mode / Absolute Addressing Mode

* In this mode of addressing, the operand is a memory location. The address of the location is explicitly given in the instruction.



(ii) Register Addressing Mode

* In this mode, the operands are the contents of the CPU register. The name of the register is given in the instruction.



(iii) Immediate Addressing Mode :-

* The operand is given explicitly (directly) in the instruction.

or

→ In this mode, the operand is specified in the instruction itself.

(IV) Indirect Addressing Mode :-

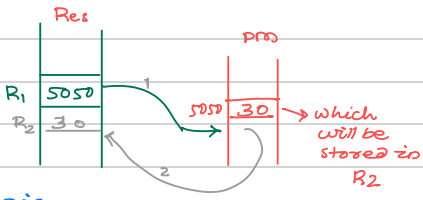
written in paranthesis

MOVE (R1), R2

ADD (R1), (R2)

ADD (A), (B)

* effective address of the operand is the content of the register or a memory location whose address appears in the instruction.



* we denote Indirect Addressing Mode by pressing the mode of memory location inside the paranthesis.

→ The register or memory operation that contains the address of the operand is called pointer.

Indirect Ad. Modes are of 2 types :-

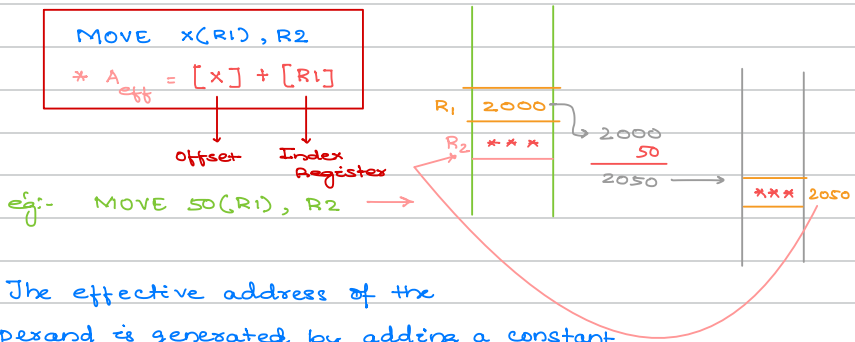
- Register Reference Indirect Addressing
- Memory Reference indirect addressing

MOVE (R1), R2

MOVE (A), (B) ; total no of addressing modes :-

* fetch
* 2 + 2
* add
* store

(V) Index Addressing Mode :-



* The effective address of the operand is generated by adding a constant value to a content of a register. The register used here can be either a SPR (Special Purpose Register) or more commonly it may be any one of the GPR.

→ The register is referred as a index register.

→ The effective address we get after adding the offset with the index register is valid for that instruction only. The content of the index register remains unchanged.

(VI) Relative Addressing Mode :-

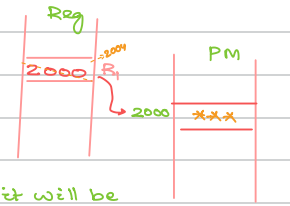
* This is similar as Index Addressing Mode only exception is here the program counter is used as the default index register.

* This Addressing Mode is used in branch instruction to calc the branch target address.

⑦ Auto Increment Addressing Mode :-

First Use, then Increment

(R1)+

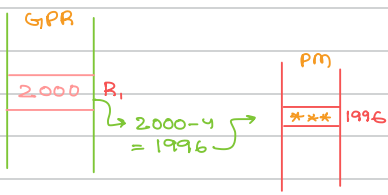


then it will be incremented by +4 ; 2004
first use then increment

⑧ Auto Decrement Addressing Mode :-

First Decrease then Use

-(R1)



first decrement, then use

⑨ Implicit / Implied Addressing Mode :-

* In this mode, the operands are specified implicitly in the instruction.

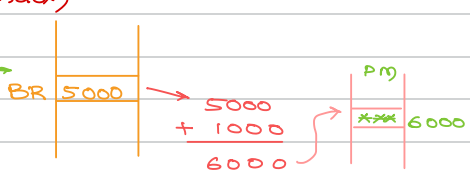
* All register reference instruction that uses accumulator & all zero address instructions are implied addressing mode.

COM → COMPLEMENT OF CONTENT OF ACCUMULATOR

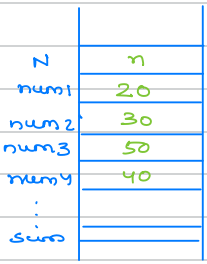
⑩ Base Register Addressing Mode :-

Base Register [BR] is a SPR. Here Base Register is use by default. (same as Index)

ADD 1000 (BR), R2



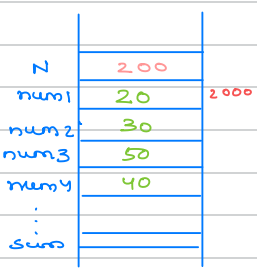
Q- Write a Assembly Language code to add 'n' numbers.

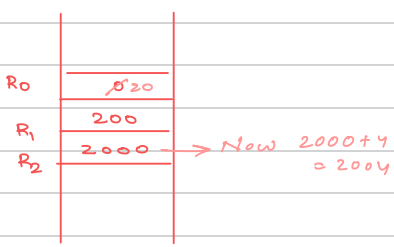



```
MOVE   N    R1
MOVE   #NUM1, R2      ← address of NUM1 is taken in R2
CLEAR  R0
ADD    (R2), R0       ┐
ADD    #4, R2         │ Loop
DCE    R1             ┘
BRANCH >0  LOOP       ← depends on previous statement
MOVE   R0  SUM
```

ex N = 200

DCE → Decrement





199 > 0 → again add (R2), R0

18-01-25

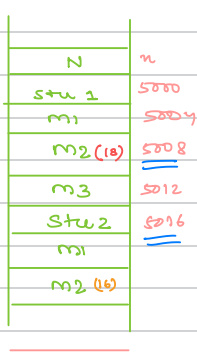
H.W → WAP to find the sum of Test 2 marks of all the students?

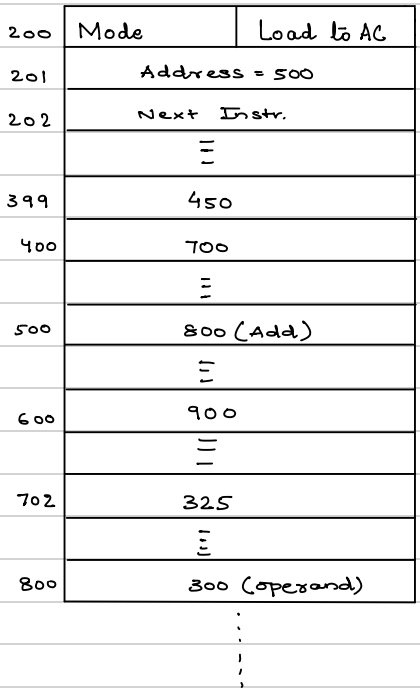

```
MOVE   N, R1
MOVE   #STUDID, R2     or #5000
CLEAR  R0
ADD    8(R2), R0       offset
ADD    #16, R2         ┐ Loop
DCE    R1              │
BRANCH >0  LOOP        ┘
MOVE   R0, SUM
```

R_1 = ~~00~~ 79

R_2 = ~~5000~~ 5016

R_0 = ~~0~~ 18+16





PC 200

R1 400

*R 100 (Index Register)

BR 202 (Base Register)

AC ?

Find the content of AC by all addressing modes

① Direct AM

LOAD 500 to AC

M[500] = 800

A ← 800

② Register AM

LOAD R1 to A

A ← 400

③ Register Indirect AM

LOAD (R1) to A

M[400] → 700

A ← 700

④ Memory I A M

LOAD 500 to A

M[M[500]] → M[800] → 300

A ← 300

⑤ Immediate AM

LOAD #500 to A

A ← 500

⑥ Index AM

LOAD #500(*R) to A

500 + 100

M[600] → 900

A ← 900

⑦ Relative AM

LOAD #500(PC)

↓

202

+ 500

M[702]

↓

325

A ← 325

⑧ Auto Increment AM

LOAD (R1)+ to AC

AC ← 700

R1 ← 401

⑨ Auto Increment AM

LOAD -(R1) to AC

↓

R1[399]

↓

450

A ← 450

⑩ Base Register AM

LOAD 500(BR) to AC

↓

M[500 + 202]

↓

M[702]

↓

325

A ← 325

EA → effective address

W	opcode
W+1	Y
Z	?
W+2	Next Inst

Direct AM

EA of Z = Y

Indirect AM

EA of Z = [Y]

Relative AM

EA of Z = Add. part + [PC]

= Y + W+2

= Y + W+2

Index AM

EA of Z = Add. part + [IR]

= Y + X

Q- A computer has 32 bit instruction, 12 bit address. If there are 250, 2 address instruction, then how many one address insts. can be formulated?

256 − 250 = 6

250 are already occupied

opcode	operands
0	0_1 0_2

8 bit

8 bit for one opcode

no of opcode possible = 2^8 = 256

12 bit

no of opcode possible = 2^{12} = 4096

No of combination possible = 4096 × 6 = 24576

Total 1 address instru.

Q- What must be the add. field of an index add. mode inst^{n} to be made so that it is same as the register indirect addressing mode ins^{v}?

index

MOVE x(R1), R2

?

0 // Ans

reg. indirect

MOVE (R1), R2

OR

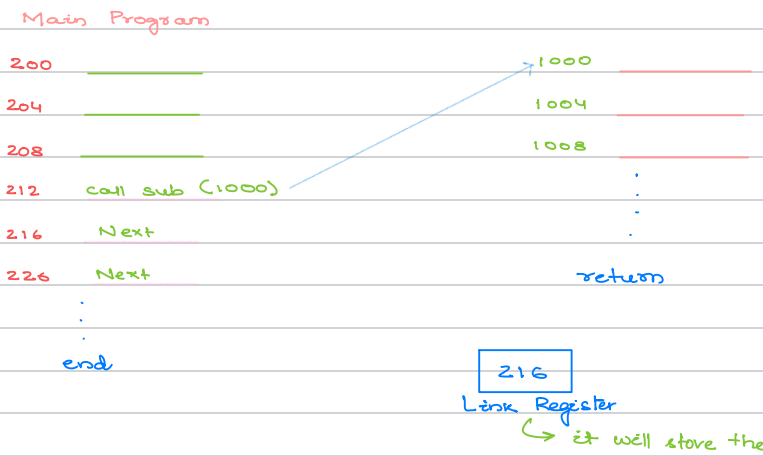
IRM $\Rightarrow$ EA = X + [Ri]

IR $\Rightarrow$ EA = [Ri]

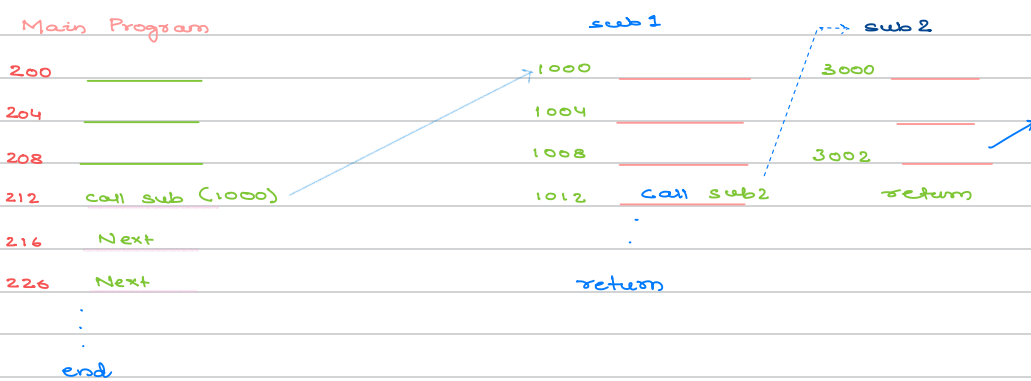
X + [Ri] = [Ri]

$\boxed{X = 0}$

Subroutine

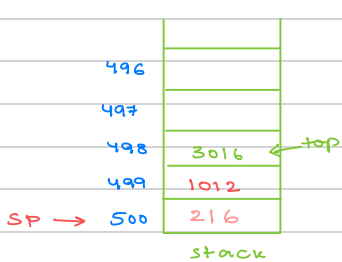


Nested



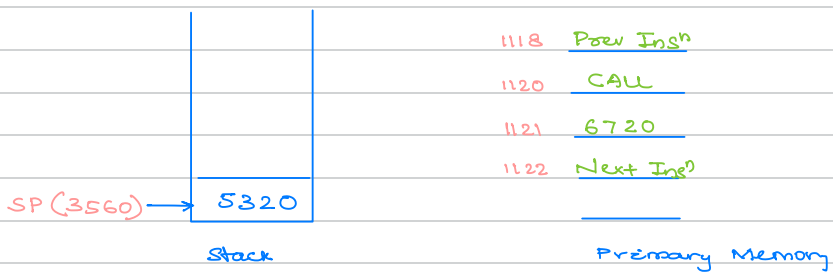
(it will call again & again)

We will use stack instead of link register



When we will return, pop the stack.

Q- The content of the top of a stack is



(i) Before Calling Subroutine :-

Value of SP → 3560

TOS → 5320

PC → 1120

TOS → top of stack

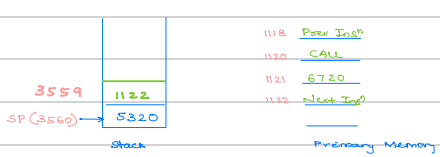
PC → Program counter

(ii) After subroutine call

Value of SP → 3559

TOS → 1122

PC → 6720



(iii) After return

Value of SP → 3560

TOS → 5320

PC → 1122