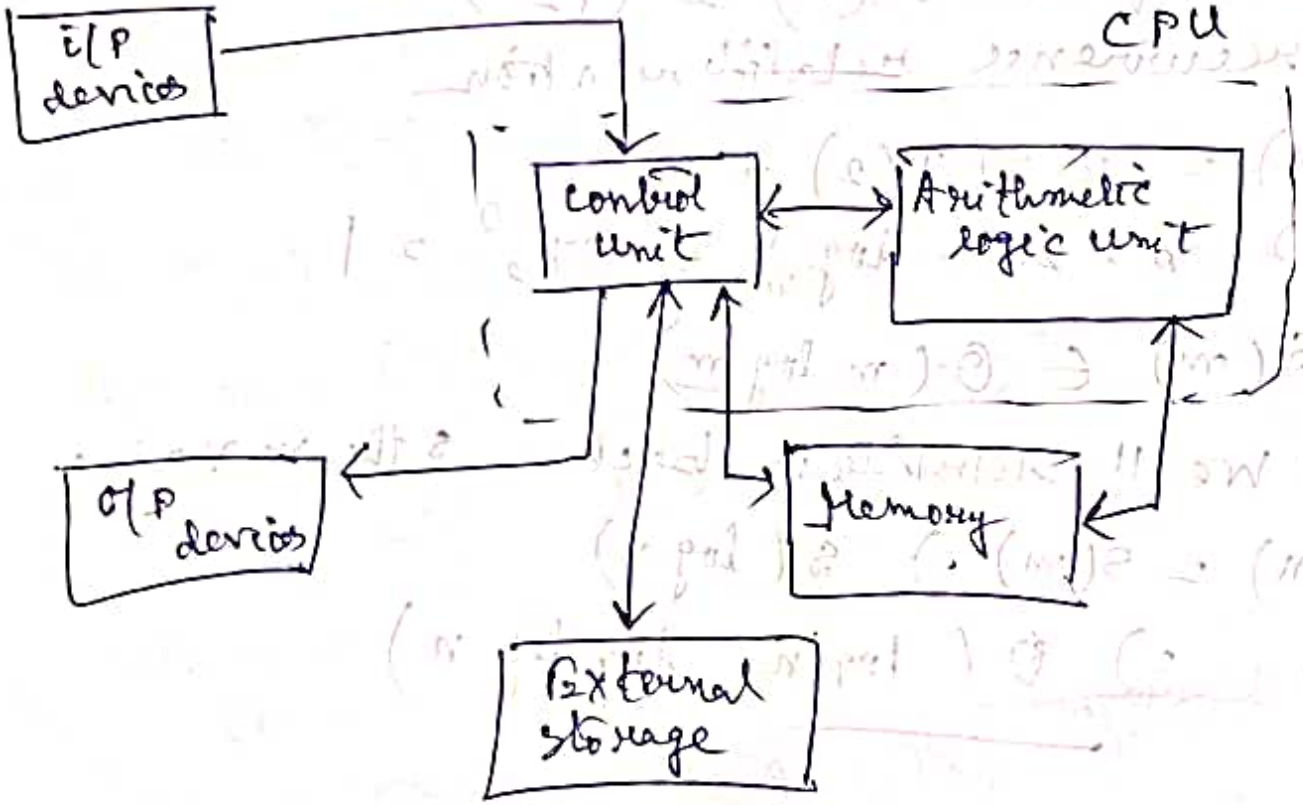
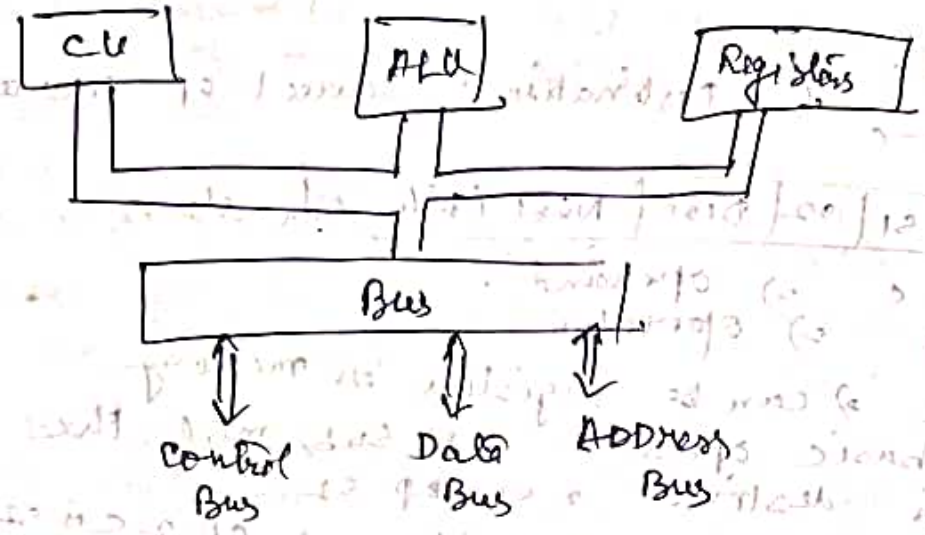
1) BASIC
=> Beginner's All Purpose symbolic Instruction code





We're going to discuss ~~the~~ about one of the imp. aspect of any computer organization / comp. Architecture. We're going to discuss comp. instructions. Before we start our discussion we must know what are the components that our computer is made of.

As we know our computer means our CPU and some peripheral devices, some input/o/p devices. or our memory. How they interact to each other is the basic study of any comp. orga. / comp. Arch.

CPU is made of 3 things —

1) ALU 2) CU 3) Registers.

ALU is responsible for doing any arithmetic and logical operation in any computer.

2) CU is responsible for controlling different parts of the comp. It is basically used to issue some signals to different devices. It is also responsible for controlling internal elements of CPU.

3) Registers is a small memory unit that is responsible for holding diff. instructions as well as data that are CPU is currently working on.

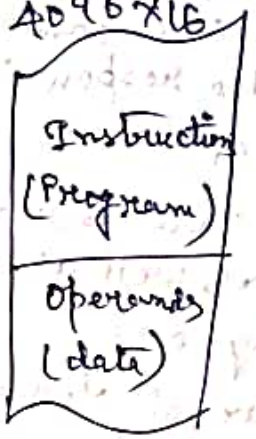
To interact with other devices like external storage devices like CD ROM, Hard disk, PD & i/p devices like keyboard, mouse & o/p devices Printer we use some buses called control bus, data bus and address bus. These buses are used to communicate betn CPU & other devices.

Now we're going to discuss how any computer instruction works.

$A = B - C$

op code	S1	S2	Dest	Next instruction address

A, B, & C ⇒ a) operand :
=, + ⇒ a) operator
A, B, C ⇒ can be registers or memory.
For any basic operation Add, Sub, mul. the general is destination S1 OP S2
Here, A is dest, B is S1 & + OP & C is S2
To execute these type instructions we need Basic computer inst. format.
For any instruction we've to do add, sub, mul, or any mathematical / logical operation. For doing any operation we need some operators. We're specifying op code / operation code here.
After execution of any particular instruction we must go to next instruction. After completion of inst. A = B+C, comp. will go to next inst. location
A = B - C.
We've another thing to store is next instruction address.



From this dia. we can easily understand that is RAM has 4096 no. of locations and in each location it has 16 bits. so, anything we can store with in this RAM can max. 16 bits.

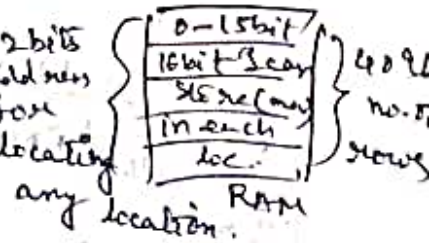
Now $2^n = 4096$ $n = 12$ so in this comb. we need 12 bits address for locating instruction.

∴ 12 bits are required to locate any address in RAM. we must have 12 bits Address bus because we know address bus responsible for holding any memory location address. And we need 16 bits data bus, because on each location we can have READ/ Write 16 bits

We can assume RAM as a table also where we can have 4096 no. of rows available. and in each location it has 16 bits.

So assume 4096 rows and each row 16 bits.
In any row I can store upto 16 bits. (max)
If we apply binary operation $2^n = 4096$, $n = 12$
so, we need 12 bits address for locating any location.

If we have 12 bits to locate, we can locate in any loc. in this RAM.



Address Bus => responsible for holding any memory location.
Data Bus => responsible for holding any value that is address bus currently locating..
Control bus => is responsible for instructing the Read/ write ope^n.

Opcode	S1	S2	Dest	Next inst
4 bit	12	12	12	12

Total = 52 bits.

If we've ope like -
A = B+C / A = B-C.

If comp. has 16 operations available the 4 bits are required i.e $2^4 = 16$ as op-code.
Now for locating any location in the memory we need 12 bits for S1, S2, dest & Next inst. address. each & every bits has 12 bits. so, totally we have 52 bits for field locating any location / executing any instruction.

Opcode	S1	~~S2~~	~~Dest~~	~~Next inst~~

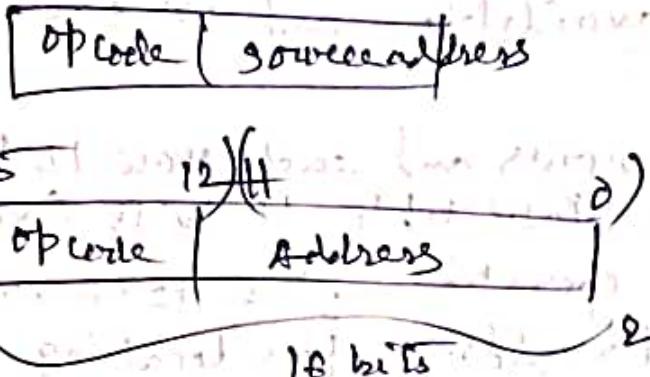
A = A+B / B = B-C Ac ← AC + S1

We're introducing a special register call accumulator that we will use to hold both source & Destination address.

A = A+B A is present in both left & right side of the expression Here we can have acc & S1 same. S1 & Destination is same.
Ac will work as S1
Now discarding S2 & dest. & keeping 1 source so, the instruction format will become like Ac ← Ac ⊕ S1 /-/+/1

Now we're introducing PC/ Program counter is a special Purpose register that we will use to hold next instruction address. / We're also getting free of 12 bits

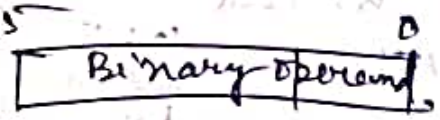
new instruction will become like this



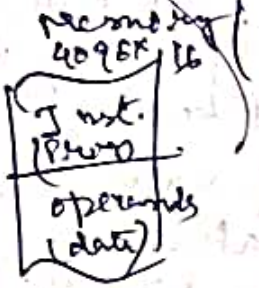
4 bits for opcode (12–15)

0–11 12 bits for address

16 bits Instruction format



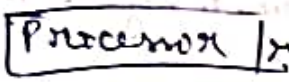
16 bits instruction (0–15)



How any comp. program is made of?

⇒ Any comp. program is made of 2 things

1) ~~Program~~ Instruction
2) data

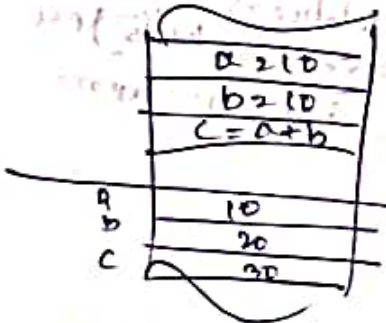


Whenever any prog. resides in RAM, comp. divides it in 2 parts. 1) Inst. 2) data.

Ex:

```
inst. { int a, b, c;
        a = 10;
        b = 20;
        c = a+b
```

Basic 3 instruction. compiler will assign 3 memory location in data part. 3 empty spaces in our RAM.



When comp. executes a = 10 instruction then this 10 value will come in the empty space allocated for a.

For this architecture our ~~comp.~~ has 16 bit information system. The binary operand, means a, b, c has 16 bit data also.

55, 22, 88, 33, 11, 44

88

88

55, 22, 88

44

33, 11, 44

55

88

33

44

- Evaluate the following arithmetic expression using 0, 1, 2, 3 address instruction:-

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

Three address instruction

```
ADD   R1, A, B  ; R1 ← M[A] + M[B]
MULT  R2, C, D  ; R2 ← M[C] * M[D]
DIV   X, R1, R2 ; X ← R1 / R2
```

Two address

```
LOAD  R1, A   ; R1 ← M[A]
ADD   R1, B   ; R1 ← R1 + M[B]
LOAD  R2, C   ; R2 ← M[C]
MULT  R2, D   ; R2 ← R2 * M[D]
DIV   R1, R2  ; R1 ← R1 / R2
STORE X, R1   ; X ← R1
```

One address

```
LOAD  C   ; AC ← M[C]
MULT  D   ; AC ← AC * M[D]
STORE T   ; T ← AC
LOAD  A   ; AC ← M[A]
ADD   B   ; AC ← AC + M[B]
DIV   T   ; AC ← AC / M[T]
STORE X   ; X ← AC
```

Zero address :-

```
PUSH  A   ; TOS ← A
PUSH  B   ; TOS ← B
ADD       ; TOS ← (A+B)
PUSH  C   ; TOS ← C
PUSH  D   ; TOS ← D
MULT      ; TOS ← (C*D)
DIV       ; TOS ← (A+B) / (C+D)
POP   X   ; X ← TOS
```

Instruction Length

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

LOAD :) Symbolic op-code used for transferring data to register from memory.

STORE :) Symbolic op-code used for transferring data to memory from register.

The symbolic opcodes ADD and SUB are used for the arithmetic operation addition and subtraction respectively. Assume that the respective operands are in memory address A, B, C, D and the result must be stored in the memory at address X.

Three address Instructions:

The general register organized computers use three address instructions. Each address field may specify either a processor register or a memory operand. The program to evaluate $X = (A+B) - (C+D)$ in assembly language is shown below.

(add V). generates short prog. & long instructio (dis ad

```
ADD R1, A, B  : R1 ← M[A] + M[B]
ADD R2, C, D  : R2 ← M[C] + M[D]
SUB X, R1, R2 : X ← R1 - R2
```

Two address Instructions:-

The most popular instructio in commercial computers are two-address instruction

```
LOAD  R1, A   ;  R1 ← M[A]
ADD   R1, B   ;  R1 ← R1 + M[B]
LOAD  R2, C   ;  R2 ← M[C]
ADD   R2, D   ;  R2 ← R2 + M[D]
SUB   R1, R2  ;  R1 ← R1 - R2
STORE X, R1   ;  X ← R1
```

One Address Instructions:-

The single accumulator based computers use one-address instructions. Here all instructions use an implied accumulator (AC) register.

```
LOAD   C  ;  AC ← M[C]
ADD    D  ;  AC ← AC + M[D]
STORE  T  ;  T ← AC
LOAD   A  ;  AC ← M[A]
ADD    B  ;  AC ← AC + M[B]
SUB    T  ;  AC ← AC - M[T]
STORE  X  ;  X ← AC
```

storing the intermedia result

T is the temporary memory location required for s

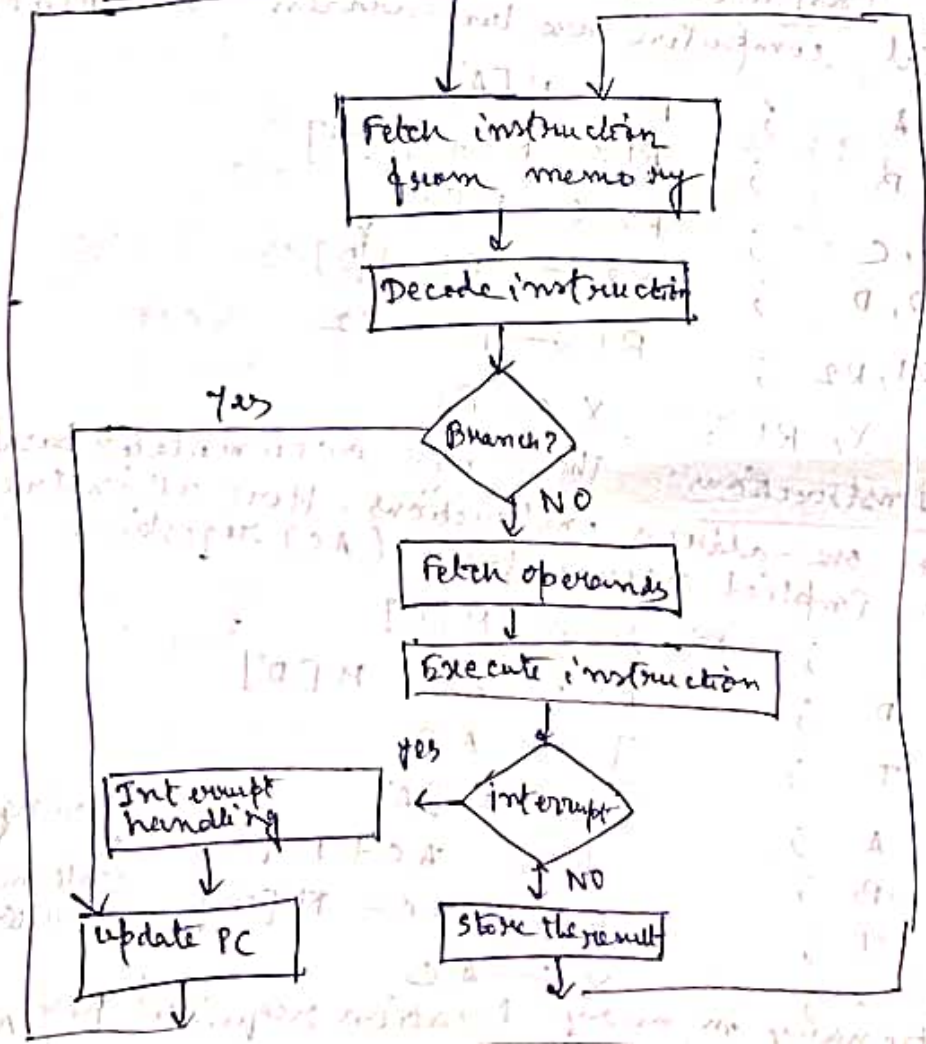
Zero address instructions:- It is used by stack organized computers, which do not use any address field for the operation-type instructions. The name "zero-address" is given to this type of computer because of the absence of an address field in the computational instructions.

However, two basic instructions in stack PUSH and POP require an address field to specify the destination or source of operand. (LIFO)

TOS :) Top of the stack.

```
PUSH A  ;  TOS ← A
PUSH B  ;  TOS ← B
ADD     ;  TOS ← (A+B)
PUSH C  ;  TOS ← C
PUSH D  ;  TOS ← D
ADD     ;  TOS ← (C+D)
SUB     ;  TOS ← (A+B) - (C+D)
POP X   ;  X ← TOS.
```

Instruction cycle:-



Before discussing different addressing modes, it is important to get the basic idea about instruction cycle of computers. (The processing required for a single instruction is called instruction cycle.) The control unit's task is to go through an instruction cycle. That can be divided five major phases.

1) Fetch the instruction from the memory.
2) Decode the instruction.
3) Fetch the operands from memory or register.
4) Execute the whole instruction
5) Store the output result to the memory/register

Step1 is basically performed using a special register in the CPU called program counter (PC) that holds the address of the next instruction to be executed. If the current instruction is simple arithmetic/logic or load/store type the PC is automatically incremented. Otherwise, PC is loaded with the address dictated by the currently executing instruction.

Step2 Decoding is done in step2 determines the operation to be performed and the addressing mode of the instruction for calculation of address of operands. After getting the information about the address of operands, the

Step3 the CPU fetches the operands in step3 from memory or registers and stores them in registers

Step4 the ALU of processor executes the instruction on the stored operands in registers

Step5 After the execution of instruction in phase 5, the result is stored back in memory or register and returns to step 1 to fetch the next instruction in sequence. All these sub-operations are controlled and synchronized by the control unit.

Instruction cycle is :— (FDEW)

a) Fetch – decode – execution
b) Fetch – execution – decode
c) Fetch – encode – execution
d) Fetch – execution – encode

There is an operation field in the instruction format.

Operⁿ field => It basically specifies the operation to be performed. Operation to be performed on such data that is the operands; These operands are needed to be selected carefully, which depends upon the addressing mode.

Addressing Modes :- (defⁿ) These are the rules specifies for interpreting the address field of the instruction before the actual operands is referrenced. We can get operands in many different ways that depend upon the addressing mode that we are selected.

There are 10 addressing modes :-

① Implied Mode :- In this addressing mode the operands are specified implicitely.

RAL - Rotate the content of acc through carry

eg: CMA :- " complement the accumulator "

In this instruction Acc is containing the operand and it is specified here itself in the instruction; therefore it is the implied mode. (This is popular for 8 bit microprocessors explicitely)

② Immediate Mode :- The instruction format is like

Opcode	Address

operand field instead of address field operand field is present where operand, it specifies itself here

eg 1) MVI A, 06; loads equivalent binary value of 06 to the accumulator

2) ADI 05; Adds the equivalent binary value of 05 to the content of Ac.

[Operand is specified in the instruction itself. We need not to specify the any address for operand]

[To initialize the constant value to register, this mode is used]

③ Register (Direct) Mode :- In this mode the operands are stored in the registers.

certain registers are present in the CPU. We'll refer to the particular register and that register will hold the value of the operands. (Direct because the register is holding the operand value directly.)

ADD R1, R2; Adds contents of registers R1 and R2 and

This mode is useful to a long program in storing the intermediate results in the register than memory. This will result in fast execution, since register accessing is much faster than the memory accessing.

④ Register indirect Mode :- Where register will contain an address and that particular address will refer to the operand.

→ register → contain add
↓
operand.

effective address is that address that contain the operand.

Diff bet^n R(Di) & R(indi) :-

In this mode the register will contain the operand itself.

→ Here register is going to contain an address which can also be reffered as effective address (contain the operand).

⑤ Auto increment / Auto decrement Mode :-

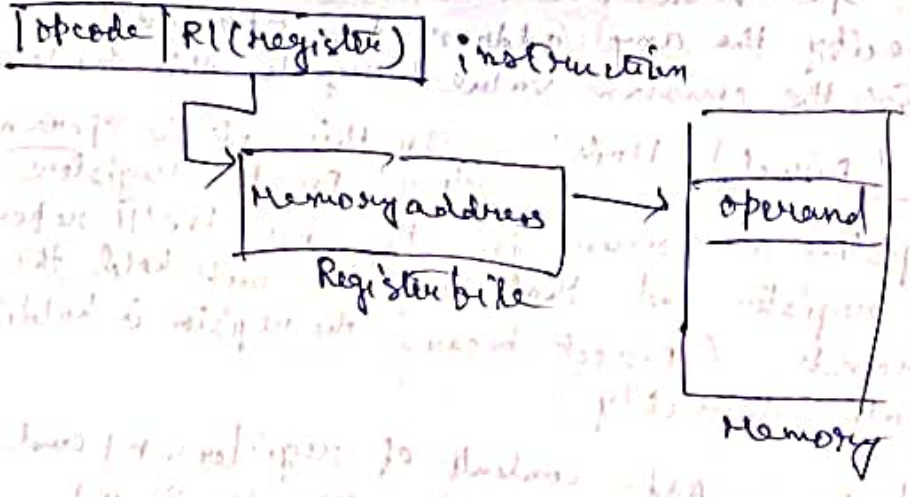
by this mode the value of the register will be incremented by 1, after the execution of the instruction

(R) → ↑ 1

by this mode the value of the register will be decremented by 1

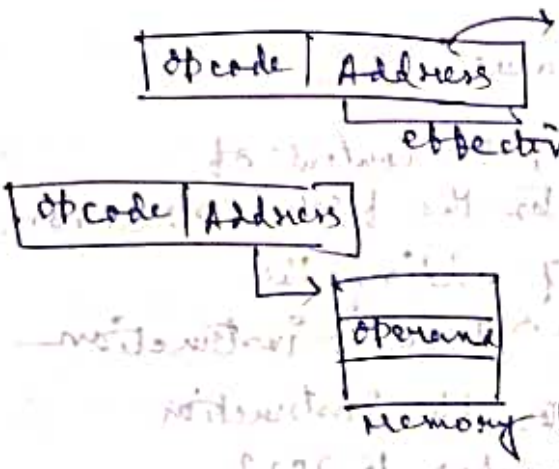
R → ↓ 1 before the execution of the instruction.

④' Register Indirect



Effective address:- sometimes the instruction directly gives the address of the operand in its format. Sometimes the instruction does not give the operand or its address explicitly. Instead of it specifies the information from which the memory address of the operand can be determined. This address is referred to as effective address.

⑥ Direct (Absolute) address Mode:-

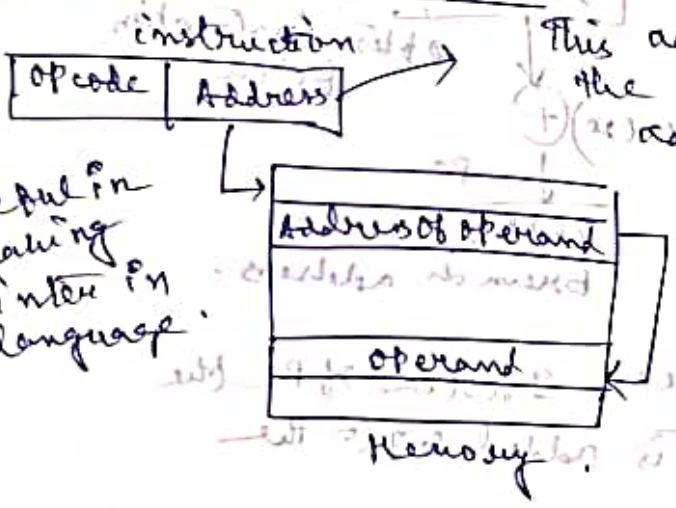


This address will directly referred to the operand.

eg: STA 2500H; stores the content of the Acc in the memory location 2500H.

LDA 2500H; Loads the Acc with content of the memory location 2500H.

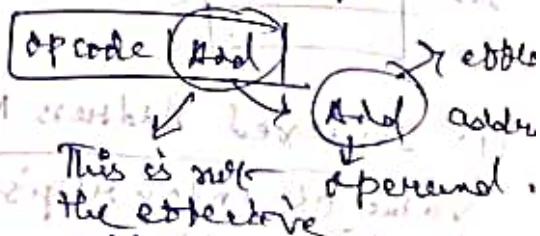
⑦ Indirect Address Mode:-



useful in making pointer in C language.

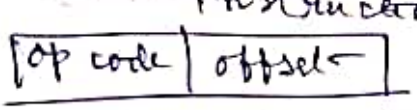
Point

This address part will refer to the another address and that address will refer to the operand. instead of operand.



eg MOV R1,(X); content of the location whose address is given in X is loaded into register R1.

⑧ Relative address / PC relative Mode:-



Operand Address = content of PC + offset

The value of PC + address field value of instruction (This address will point to the effective address.

eg JR 20; Branch to a location relative to the value 20 (offset)

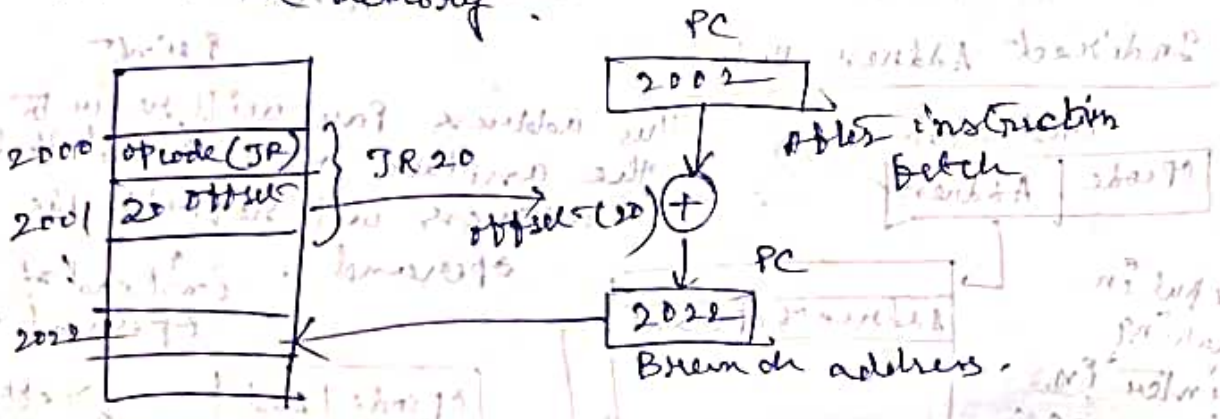
Jump to the address in register.

The branch location is computed by adding the offset value 20 with the current value of PC.

This instruction (JR 20) requires 2 bytes, one for the opcode JR and another for its offset value 20.

Since the instruction is 2 byte, the content of PC is 2000+2 = 2002 after the fetch instruction. The branch is calculated by adding the content of PC with the address part of instruction (offset).

2002 + 20 = 2022 After the instruction execution, the Program branches to 2022 address in memory.



(9) Indexed Address Mode :- Instead of PC the value of index register is added to get the effective address.

Index. Reg (XR) + addr. field = effective add.

(10) Base add. Mode

Base reg (BR) + address field = effective address.

The index reg. is a special CPU register that contains an index value for the operand.

The index value = (starting address − address of the operand) distance betn

eg The operand array starts at memory address 1000 and assume XR contains the value 0002

Now consider load instruction

LDA 1000

Eff. addr = 1000 + contents of XR = 1002

10 Base It is used for relocation of the program in memory

A base register holds the starting address of a memory array of operands and the address part of the instruction gives a displacement/offset relative to this starting address.

Relocation is a technique of moving program/data segments from one part of memory to another part of memory.

Week

① What value remains on the stack after the following sequence of instructions?

PUSH #6 (Symbol # indicates direct value of the number)

PUSH #8

PUSH #4

ADD

PUSH #12

SUB

MULT

4	12	
8	12	0
6	6	6
	0	

$\frac{4}{8} + \frac{12}{12} - \frac{10}{6} = 60$

$\frac{12}{6}$

$\frac{8}{6}$

② A relative mode branch instruction is stored in memory location 530 (decimal). The branch is made to the location 30 (decimal). What is the effective address?

532 + 30 = 562

③

		Mode
2000	LDA	
2001	Address = 2500	
2002	Next Instruction	
2299	2400	
2300	2450	
2500	2700	
2520	2800	
2700	2250	
4502	3400	

content of PC = 2000
content of RI = 2300
content of XR = 20
content of AC = ??

Example of addressing modes.

Addressing Mode	Effective Address	content of AC
Immediate	2001	2500
Register	—	2300
Register Indirect	2300	2450
Auto decrement	2299	2400
Direct	2500	2700
Indirect	2700	2250
	4502	3400

4) There are 50 processor registers, 7 addressing modes and 16K × 32 main memory. State the instruction format and size of each field if each instruction supports one register operand and one address operand.

→ The processor has 50 registers, so, 6 bit used to specify each reg. uniquely. (because $32 < 50 < 64$)
No. of addressing modes is 7, so 3-bit is used in mode field in the instruction format (because $7 < 8$)
The main memory size is 16K × 32; so to access each word of 32 bit in memory 14-bit address is generated.
Therefore, the op-code field requires $32 - 3 - 6 - 14$
$= 9$ bit.

The instruction format will be as:

Mode (3-bit)	Opcode (9-bit)	Registeradd. (6)	Mem Add. (14 bit)

5) There are 54 processor registers, 5 addressing modes and 8K × 32 main memory. State the instruction format and size of each field if each instruction supports one register operand and one address operand.

→ 54 processor register. $(32 < 54 < 64)$
2^6 — 6 bits used to specify each reg. uniquely
8K × 32 Main memory $\Rightarrow 2^3 \times 2^{10} = 2^{13}$ address Bus.
32 ⇒ Data bus.
5 addressing modes. $5 < 8$ / 2^3 bits used in mode field in instruction format.

mode 3 bit	Opcode 10 bits	Reg. Add (6 bits)	Memory Add. (13 bits)

$32 - (3 + 6 + 13) = 32 - 22$
$= 10$ bits

Q) A two-byte relative mode branch instruction is stored in memory location 1000. The branch is made to the location 87. What is the effective address?

⇒ As the instruction is two byte. The content of PC is 1000+2 = 1002. after the instruction fetch. The address will be evaluated by adding the content of PC with address part of the instruction which is 87.

∴ The effective address = 1002 + 87 = 1089.

Thus, after the instruction execution, the program branches to 1089 address in memory.

Q) Suppose it takes 1 ns to read an instruction from memory, 3 ns to decode the instruction, 5 ns to read the operands from register file, 2 ns to perform the computation of the instruction and 4 ns to write the result into the reg. What is the max. clock rate of the processor?

(1+3+5+2+4) = 21 ns

∴ cycle time = $(1/21 \times 10^{-9})$

= 1000/21 MHz

= 47.62 MHz

Week-5

1) Represent the decimal numbers ~~0.75~~ in IEEE single Precision format and convert it into Hexa-decimal numbers.

(i) 0.75 -1.1×2^{-1} $(BF400000)_{16}$

(ii) -1.75 $(17FC00000)_{16}$ -1.11×2^{0}

(iii) 21 $(BFE00000)_{16}$

(iv) -7.75 $(41A80000)_{16}$ 1.0101×2^{4}

(v) -0.125 $(COF80000)_{16}$ -1.1111×2^{2}

$(BE000000)_{16}$

$= -(0.001)_2$

$= 1.0 \times 2^{-3}$

1) A floating point number system uses 16 bits for representing a number. The most significant bit is the sign bit. The least-significant nine bits represent the mantisa and remaining 6 bits represent the exponent. Assume that the nos are stored in the normalised format with the one hidden bit.

a) -1.6×10^3

b) 0 000100 11 0000000

a)

Sign	6 bits exponent	9 bits Mantisa
1		

$-1.6 \times 10^3 = -1600$ binary of 10 (B)

$= -11001\ 000000 \times = -1.1001000000 \times 2^{10}$

1	001010	1001 000000

b)

S	E	M
0	000100	11 0000000

+4 384 $2^7 + 2^8 = 128 + 256 = 384$

$+1.M \times 2^E = +1.384 \times 2^4$

$= 22.144$

2) If each register is specified by 3 bits and instruction ADD R1, R2, R3 is 2 byte long then what is the length of the op-code field

→ The opcode field can have $16 - 3 - 3 - 3 = 7$ bits

opcode 7 bits	R1 3 bits	R2 3 bits	R3 3 bits

3) What is the maximum no. of 0-address, 1-address and 2-address instructions if the instruction size is of 32 bit and 10 bit address field?

→ 0 Address no. address field is used. So all 32 bit of the instruction size can be used as the size of op-code field. Max no. of 0-address instructions is 2^{32}

1 Address instructions, 1 address field is used whose size is given 10 bits. So for opcode field $32 - 10 = 22$ bits can be used.
Max. no. of 1-address instruction 2^{22}.

2 address instruction instructions. 20 bits are occupied for address field.
opcode = $32 - 20 = 12$ bits
Max. no. of 2 address instruction 2^{12}.

4) A 2 byte long assembly language instruction BR 09 (branch instruction) stored at location 1000 (all no. are in Hex). The PC holds the address.

a) 1002 b) 1000 c) 1009 d) 100B