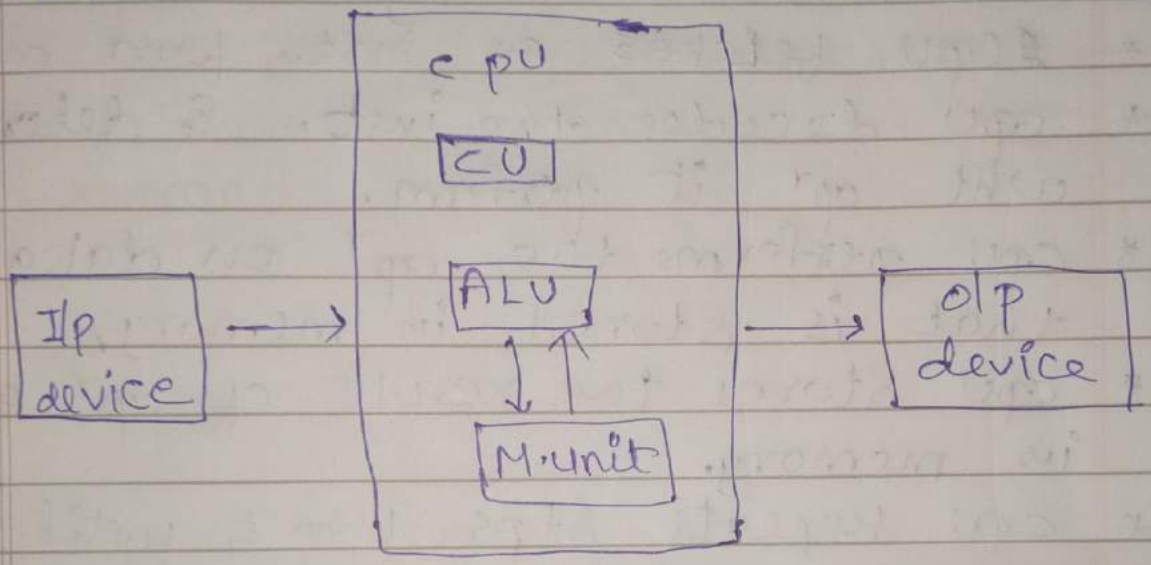
Module - 3

Basic computer organization Design and ~~Archi~~ von Neumann Architecture



The von N. Archi~ consist of following components —

1) cpu :-
Brain of comp. It is responsible for executing the inst~ of a comp prgm.

2) memory :-
It is where the comp stores data & instrn. It is ÷ into 2 —

a) prgm memory & data memory.

b) Input / output devices.
I/o devices allow the comp to

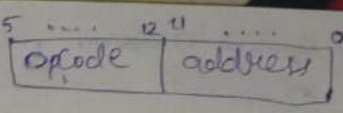
interact with the outside world.
eg of I/O devices - keyboard, mouse, monitors

V.N. Archi - working :

* CPU fetches an instr. frm memory
* cpu decodes the instr. & determines wht opr it perform.
* cpu performs the opr on data that is stored in memory.
* cpu stores the result of opr back in memory.
* cpu repeats stps 1 to 4 until the comp prgm finishes executing.

⇒ Instr. codes =

* It is a grp of bits that instructs the comp to perform a task.
* A comp instr. is binary code thats specifies a sequence micro opr for the components.
* The comp reads each instr. frm memory & places it in a control register.
* The control then interprets the



opcode → opr.
add

inst

binery code of & proceeds to execute it by issuing a sequence of micro opr.

* It is usually ÷ into 2 parts, each having its own interpretation -

1) opr code (opcode)
2) operands

I opr code :

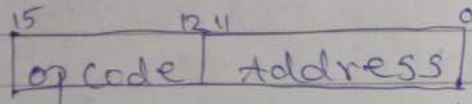
* opcode of an instr. is a grp of bits that define each opr such as add, subtract, multiply, shift & complimnt
* It must consist of atleast n bits for a gvn 2^n distinct opr.
* Suppose we are hving 64 (2^6) opr then the length of opcode will be 6.
* control unit decode the opcode & do the required opr.

II operands :

* opr must be performed on some data stored in processor, register / in memory.
* Every comp has its own Instr. code

format.

* Simplest way to organize a comp is to have an instr. code format with 2 parts.
* 1st part specifies the opr to be performed & 2nd part specifies an address.



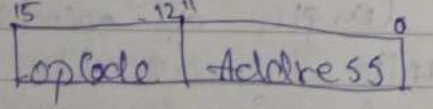
opcode → calculations

64 → 2^6 → 6 bits

op Add > 16 bits

opcode consist of 6 bits

→ Comp instr.. :



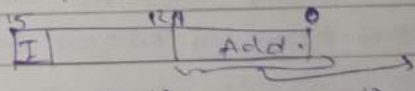
* Basic comp has 3 instr. code format [each format has 16 bits (0-15)].
* opcode part of instr. contains 2 bits & meaning of the remaining 13 bits depends on the opr code encounters.
* 3 instr. code format are -

1) Memory-Refrence instr..
2) Register - " "
3) I/O opr

I Memory-Refrence instr.:

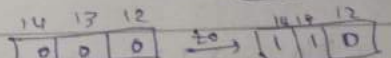
* It uses 12-bits to specify an address & 1 bit to specify the addressing mode 'I'.
* I = 0, for direct addres & to 1, for indirect address.
* opcode = 000 through 110

If I = 0 → direct → direct add
If I = 1 → indirect → indirect refrence



12 bits store address

15 → I



000 to 110

or D/Indirect

m/y-R. (I)

by I.

2 add modes → direct, Indirect

*

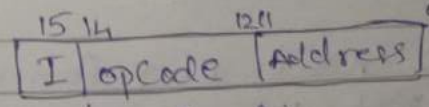


fig: m/y-R. (I)

I → 0, 1

AC → Accumulator

II Register - Reference instr... :

* They are recognized by the op^r code 111 with 0 in the leftmost bit (bit 15) of the instr...
* It specifies an op^r on / a test of the AC register.
* An operand from m/y is not needed, So other 12 bits are used to specify / test to be executed.
*

15	14–12	11–0
0	1 1 1	register opr

(o → direct mode)

fig : R.R. Instr.

(add separate നമ്പറുകൾ ഉപയോഗം ഇല്ല.
ഓരോ op^r ആണോ വേണ്ടത് അതിന് reg opr-ൽ കൊടുക്കും.
12th, 13th, 14th → 111 ആണെങ്കിൽ അത് R.R.I. ആയിരിക്കും. & 15th bit 0 ആകും.)

III Input - output Instr... :

* It does not need a reference to m/y & recognized by the op^r code 111 with a 1 in the leftmost bit of the instr...

* Remaining 12 bits are used to specify the type of I/O op^r / test performed.
*

15	14–12	11–0
1	1 1 1	I/O op^r

fig : I/O instr...

[15th bit always be 1]

→ Instr... Set Completeness :

* The set of instr... is said to be complete if it has following categories -

a) Arithmetic, logical & Shift instr...
b) Instr... for moving to & from m/y & processor registers.
c) Prgm control instr... together with instr... that check status condi...
d) I/p & o/p instr...

→ Comp Registers =

* A register is a very small amount of very ~~small~~ fast m/y that is build into the cpu in order to speed up its op^r by providing quick access 2 commonly used values.

~~IB ACIO~~

(R) — Registers

* Registrs refers to semi cond devices whose contnts can be accessed (read & written to) at extremely high speed but which are held there only frm temporarily.
* Registrs are used to store data temporarily during the execution of prgm.

Register Symbol	Register name	No. of bits	Description
1) AC	Accumulator	16	processor (R)
2) DR	Data (R)	16	Hold m/y data
3) TR	temporary (R)	16	Holds temporary data.
4) IR	Instr... (R)	16	Holds instr... codes.
5) AR	Address (R)	12	Hold m/y address
6) PC	prgm counter	12	Hold addr.. of nxt instr..
7) INPR	Input (R)	8	Hold I/p data.
8) OUTR	output (R)	8	Hold o/p data

[illegible] instr.. [illegible] — Instr.. (R) [illegible]. nxt instr.. addr.. [illegible] → PC (R) [illegible].

⇒ Common Bus System = or I/O bus system

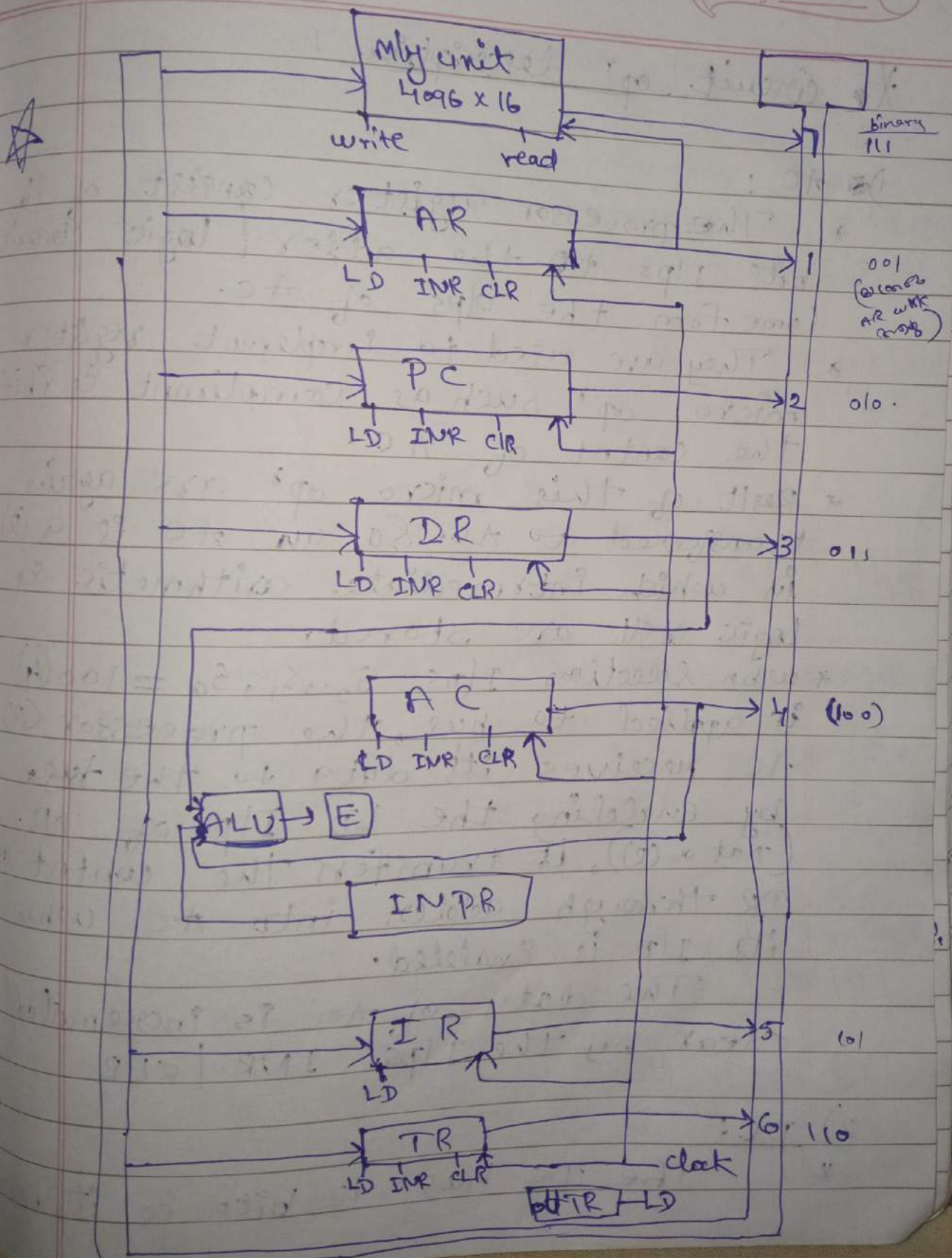
→ Bus :

* A wire / a collection of wires that carry some multi bit info → bus
* Main purpose of bus is to transfer info form 1 system to anthr.

Describing
* The basic comp has 8 (R), [AC, DR, TR, IR, AR, PC, INPR, OUTR] a memory unit & a control unit.
* path must be provided to transfr info frm 1 (R) to anothr & b/w m/y & (R).
* The no. of wires will be excessive if connections are made b/w the o/p of each (R) & I/p of othr (R).
A more efficient scheme is to use a common bus.
* Thus common bus provides a path b/w memory unit & (R).

LD → load
INR → increment
CLR → clear

E → Available



(R) → register

* Circuit op^r description:

1) AC:

* The processor register consist of 16 bits I/ps to the adder / logic circuit come from the o/ps of AC.
* They are used to implement register micro op^r such as compliment & shift the content of AC.
* Result of this micro op^r are again transferred to AC. So, an AC is a (R) in which intermediate arithmetic & logic result are stored.
* When selection I/ps $S_2, S_1, S_0 = 100$ (4) is applied to bus, the processor (R) AC receives its data to the bus by enabling the LD I/p of DR. (Data (R)), it transfers the content of DR through adder into AC when its I/p is enabled.
  The data of AC is incremented / clear by the I/ps INR / CLR

2) PC:

* The PC has 12 bits & it hold address of next instr.
* When selection I/ps $S_2, S_1, S_0 = 010$ is ~~transfers~~ applied to the bus. The PC receives / transfers address from / to the bus when LD I/p is enabled.
* The address of PC is incremented or clear by the I/ps INR / CLR.

3) DR:

* DR holds 16 bits & holds the mly data.
* When selection I/ps $S_2, S_1, S_0 = 011$ is applied to the bus.
* The data of DR is incremented / clr by the I/ps INR / CLR.

4) IR:

* IR holds 16 bits & holds instr. codes.
* When selection I/ps $S_2, S_1, S_0 = 101$ is applied to the bus. IR receives instr. codes from / to the bus when LD I/p is enabled.

5) TR:

* Hold 16 bits & hold temporary data.
* when selection I/ps $S_2, S_1, S_0 = 110$ is applied to the bus.
* The temporary register receives / transferred temporary data from / to the bus.

6) INPR

* I/p (R) consist of 8 bits & hold alpha numeric I/p info.
* The serial info frm the I/p device is shifted into I/p of 8 bit (R) INPR.
* when LD I/p of AC is enabled, the 8 bit info of INPR is transferred to the AC through adder / logic circuit

7) OUTR:

* OUTR receives info frm AC & transferred it to the output device.

→ Timing & Control =

* Timing for all registers in the basic comp is controlled by a master clock genrator.
* The clk pulses are applied to all FF & registers in the system including the FF & registers in the CU.
* cp does not change the state of a register unless the register is enabled by a control signal (LD, INR, CLR).
* There are 2 major types of control organization —

1) Hardwared control
2) Micro prgmed control.

I Hardwared control:

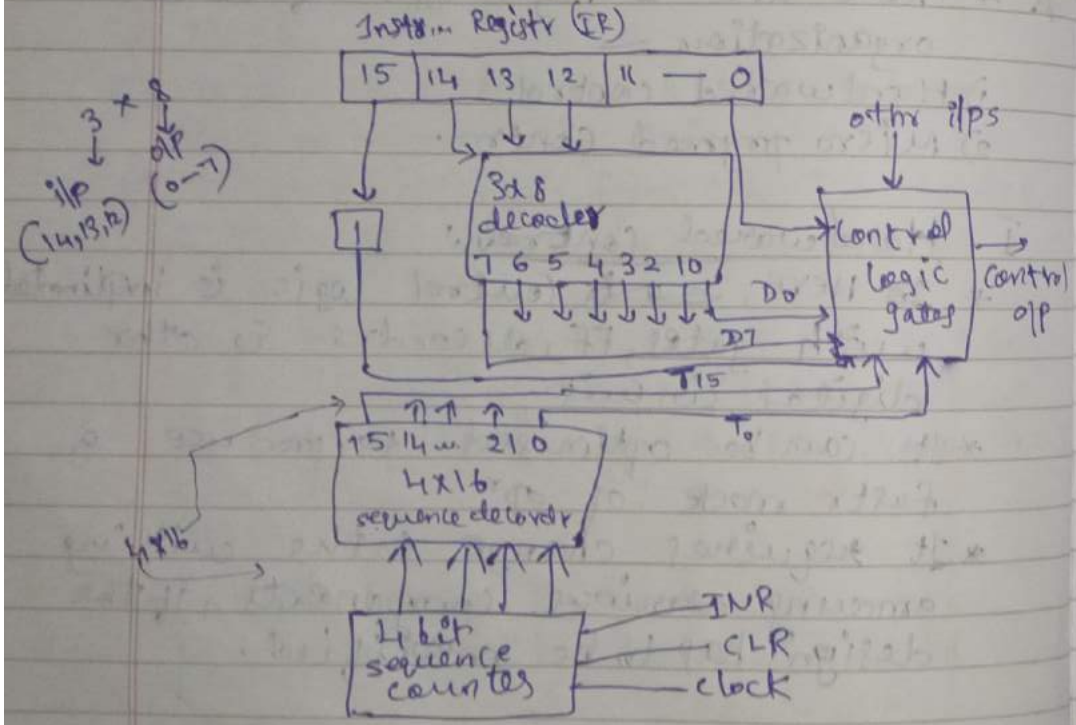
* Here, control logic is implimted with gates, FF, decoders & other digital circuit.
* It can be optimised to produce a fastr mode of op^n.
* It requires change in the wiring among various components, if the design has to be modified.

II micro prgmed control:

* In micro prgmed organiz^n the control

(R) → register.

Info is stored in a control m/y.
* control m/y is prgmed to initiate the required sequence of opr.
* If the design is modified, the micro prgm in control m/y has to be updated.
* control unit for basic comp—

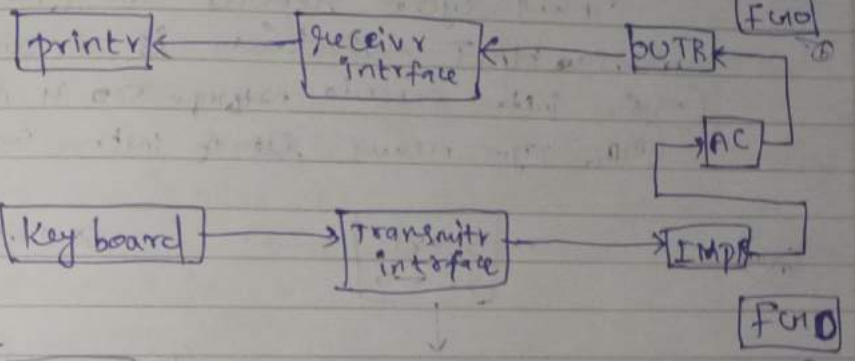


[Control logic gates → op* perform arithmetic control o/p signals]

Fig: Hardware Control

* I/O configration:

I/O terminal Serial communication interface — comp registrs & FF



→ Instruction cycle =

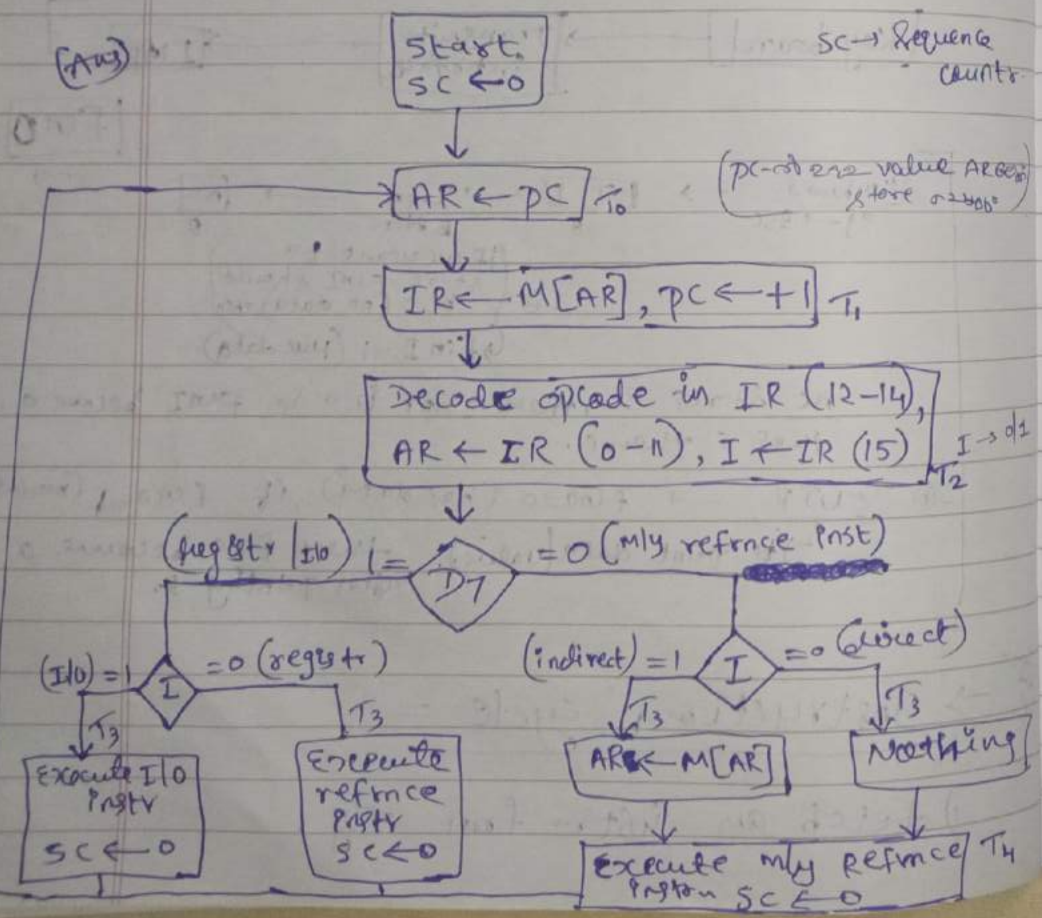
1) Fetch an instr.. from m/y.

direct address → 0
Indirect " → 1

2) Decode the instr..
3) Read the effective address from m/y if the instr.. has an indirect address.
4) Execute the instr..

This cycle repeats indefinitly unless a HALT detection is encountered.
([illegible] instr execute [illegible] 4 steps [illegible]
(Prgm means set of instrn [illegible])

(Ans)



(Q) Explain the type of instr.. with a suitable diagram?

→ **Addressing Modes** = (oprand-[illegible] loc [illegible])

* Refers to the way in which the oprand of an instr.. is specified.
* A.modes specifies a rule for intrpreting or modifying the address field of the instr.. bfre the oprand is actually executed.
* Benefits :
  * To give programing versatality.
  * To reduce no. of bits.
* Type of A.modes:

I. Implied | Implicit A.M
II. Immediate A.M
III. Registr-direct A.M
IV. Registr - indirect A.M
V. Direct A.M
VI. Indirect A.M
VII. Stack A.M
VIII. Relative A.M
IX. Indexed A.M

X Base registr A·M

XI Auto Increment A·M

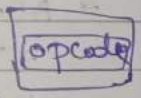
XII Auto Decremnt A·M

I Implied / ~~Implied~~ Implicit A·M =

* oprand are Specified implicitly in the defnition of the instrn.
* eg →
  - The instrn Complemnt Accumulater is an Implied mode instrn.

② ~~address~~ zero address instrn

(no oprands, only opcode) ←



II Immediate A·M =

* oprand is Specified instrn explicitly.
* Instead of address field, an oprand field is prsnt that contains the oprand.

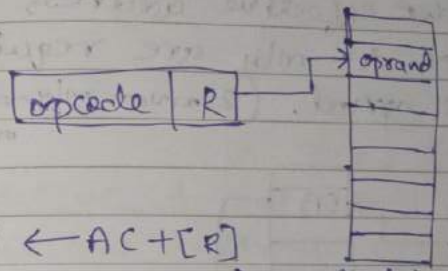
shr → | opcode | oprand |

eg:- | ADD | 10 |

III Registr direct A·M =

* oprand is contained in a register set

* Address field of instrn refrs to a CPU registr that contains oprand.
* NO refrnce to mly is required to fetch the oprand.

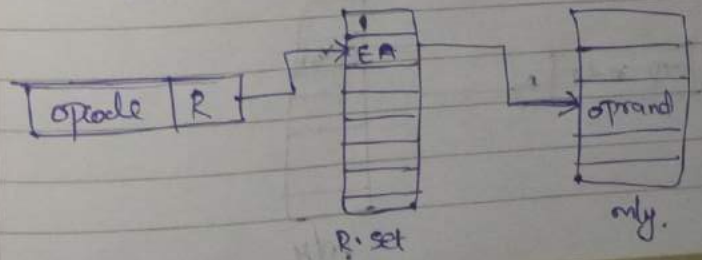


eg → AC ← AC + [R]

(Ac-ml 2nd value + registr value.

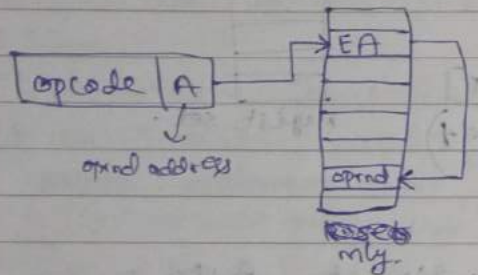
IV Registr Indirect A·M =

* Address field of instrn refers to cpu register that contains the effective address of the oprnd.
* only 1 refrnce to mly is required to fetch the oprand.



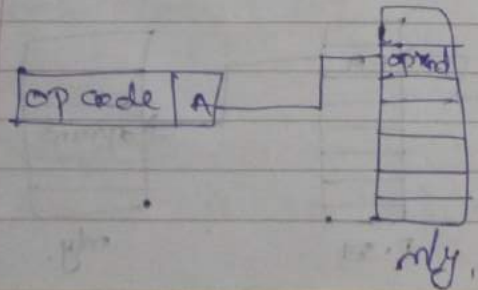
V Indirect A.M =

* Address field of the instr. Specify the address of the mly location that contains the effective address of oprnd.
* 2 refrnces to mly are required to fetch the oprand. (2 mem mly access).



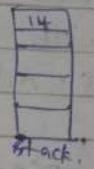
VI Direct A.M =

* Address field of the instr. contains the effective address of the oprnd.
* only 1 refrnce to mly is required.
* Also → opSolute A.M.



⇒ Stack Addressing Mode =

* Here, the oprand is contained at the top of the stack



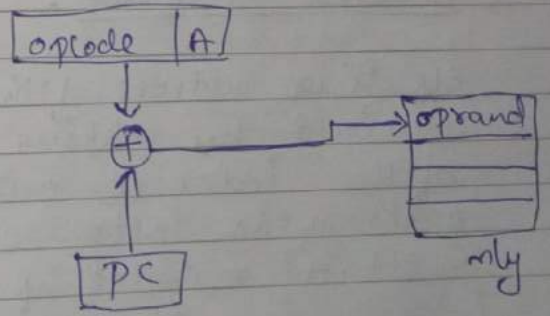
ADD 8,6.

✳ Relative A. modes :

Effective Address of oprand is obtained by adding the cntnt of pc with the address part of the instr..

Eff Address = Cntnt of pc + Ad. part of instr..

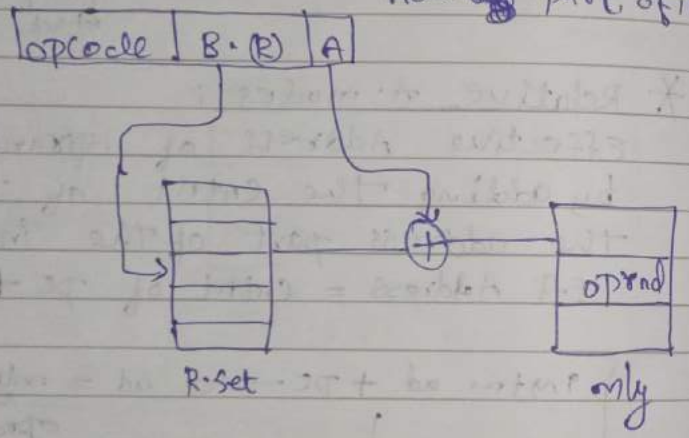
[Instr ad + pc - ad = mly ad oprand ad]



* Base Register A·modes :

Effective address of oprand is obtained by adding cntnt of Base (R) with address part of instr.

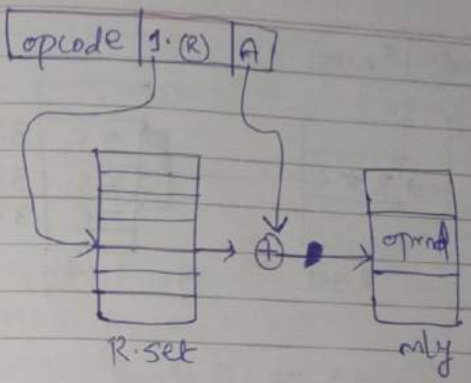
Eff. Ad = cntnt of B.(R) + Ad. part of inst.



* Indexed A·modes :

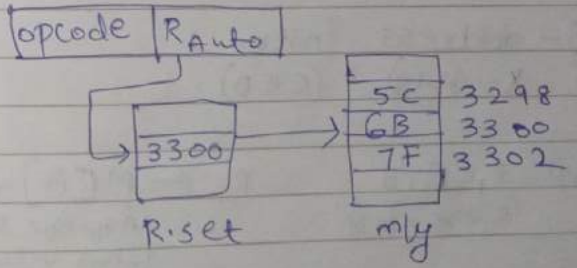
Effective address of the oprand is obtained by adding the contnt of the index (R) with the address part of the instr.

Eff. Ad = contnt of Index (R) + Ad part of ~~index~~ instr.



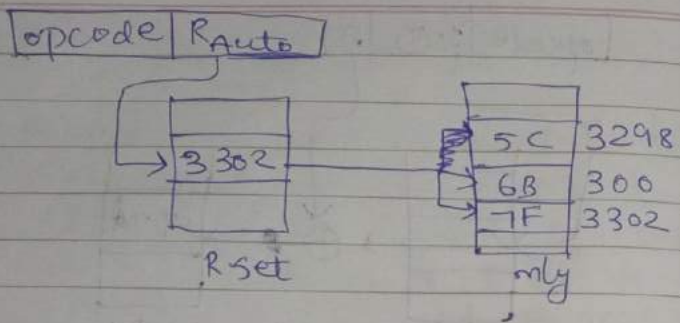
* Auto-Increment A·modes : (Automatic ~~ad~~ increment mode)

It is spl case of (R) - Indirect A·mode where : effective ad of oprnd = contnt of (R).



* Auto-Decrement A·modes :

It is a spl case of (R)-indirect A·mode where : effective ad of the oprnd = cntnt of (R) - Stp Size.



⇒ Instr.. Types =

→ No~ of address in an instr:

1) 3-address instr.. (to fetch 3 ad)
2) 2 " " "
3) 1 " " "
4) 0 " " (no ad, ಬೆ stack is used by push/pop)

[1] 3- address instr.. :

* eg→ $X = (A+B) * (C+D)$

1st (A+B) add, 2nd (C+D) add, 3rd * 2ರ

ADD R_1, A, B — $R_1 \leftarrow M[A] + M[B]$
(R1 ಅಲ್ಲಿ A & B) (A ಅಥವಾ B ಅಡ್ರಸ್ಸಲ್ಲಿ ಇರೋ value R1 ಗೆ ಬರುತ್ತೆ)

ADD R_2, C, D — $R_2 \leftarrow M[C] + M[D]$

MUL X, R_1, R_2 — $M[X] \leftarrow R_1 * R_2$.
(3)

* here we are Fetching 3 ad instr- (in this eg) R_1 R_2 X.

[2] 2-address instr :

eg→ $X = (A+B) * (C+D)$

MOV R_1, A — $R_1 \leftarrow M[A]$ (Ans mly-ಅಲ್ಲಿ ಇರೋ value R_1 ಗೆ ಬರುತ್ತೆ MOV)

ADD R_1, B — $R_1 \leftarrow R_1 + M[B]$ (ಆಗ R_1 = A value)

MOV R_2, C — $R_2 \leftarrow M[C]$

ADD R_2, D — $R_2 \leftarrow R_2 + M[D]$

MUL R_1, R_2 — $R_1 \leftarrow R_1 * R_2$
(R_1 ಅಲ್ಲಿ ಇರೋ value ಮತ್ತು 2 ad ಇರೋ R_1 & R_2 ಗಳು) (R_1 = (A+B) value, R_2 = (C+D) value)

MOV X, R_1 (2) — $M[X] \leftarrow [R_1]$
(R_1 ಅಲ್ಲಿ ಇರೋ store ಆಗುತ್ತೆ)