07/02/18

Addressing mode - by which cell operand is reference to the address. during the execution of prog.

various types -

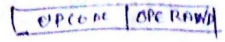
1) Implied Addressing Mode (AM)
2) Immediate " "
3) Direct " "
4) Indirect " "
5) Register " "
6) Register Indirect " "
7) Displacement " " — Relative AM
   Base Register AM
   Indexed AM
8) Stack Addressing mode
9) Auto increment/decrement addressing mode

1) Implied / Implicit
- operand absent, presence of operand null
- the instruct^n format is CMA (complement accumulator)
  operand in accumulator is complimented

RAL (Rotate accumulator Left) / LShA (left shift Accumulator)
the operand implicitly present within

- to get operand, no memory required.
- So no memory reference for operand
  one " " for opcode

2. Immediate



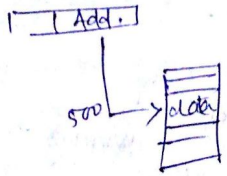
- operand immediately present within the instruct^{n}.
- direct data is given within instruct^{n}

MVI A, 05 H

- no memory required for operand, if operand is not coming from memory.

3. Direct

- address part gives the address of operand.

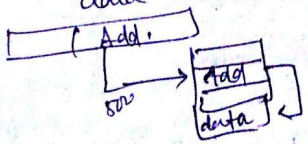


- directly the add. is present with in the instruct^{n}.

eg - LOAD Addr. AC to loc^{n} Locn.
STOR Addr.

4. Indirect

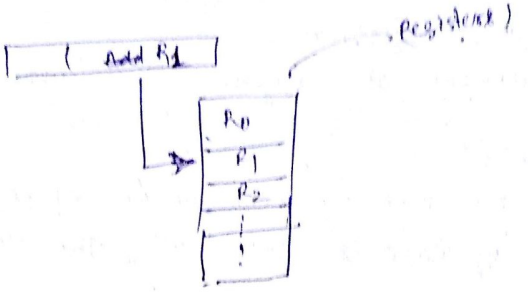
- address of operand calculates indirectly.
- add. part of instruction gives address of data



LXI H, Add - to initialize add.
MOV B, M

5) Register

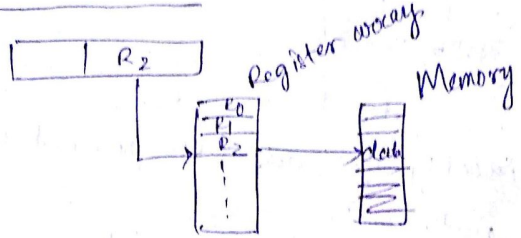
- the add. part is a register which contains the data



operand : data present in R_1

MOV R_0, R_1

6) Register Indirect



content of register is add. of operand

LXI H, Add : content of H is add of data

* Q) What is the advantage of Indirect over direct Reg. II over Reg 4!

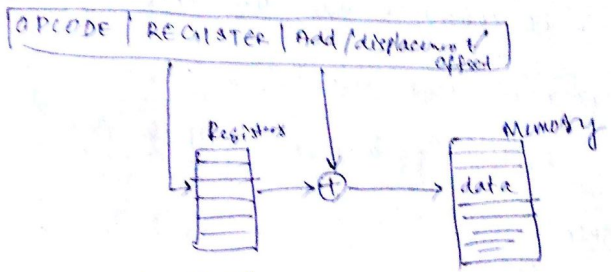
7) Displacement

Opcode	Register	Add /Displacement/offset

which

ct {
?

2) Displacement



should be in 2's complement
↑
- displacement value - may be +ve or -ve

particular address of data

3) Relative

If this register is PCounter, then this is relative AM.

eg - PC = 850 (depends on bytes of instruct^n)
displacement = +45
here, 1 byte
add. of data = 898 + 1
= 899 ∴ (PC = PC + 1)

4) Base Register

If this register is base register, then this is base register AM.

- for relocat^n of prog. where are reg. required.
- only by changing base register value it can go for one to another add.

Register operat^n is faster than memory operat^n.

5) Indexed Register

If this register is indexed reg, then this is Indexed reg. AM.

- incrementing the reg. content

Index reg = Index reg + 1.

- System performance will be slow

8. Stack

PUSH
PUSH
ADD — implied.

9. Auto increment / decrement

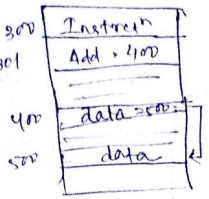
INR R_3 - content of reg. increments by 1.

- Reg direct → $R_3 + 1$ = data
- Reg indirect → add of $R_3 + 1$

DCR R_3

- direct → $R_3 - 1$ = data
- indirect → add. of $(R_3 - 1)$

Q) An instruction is stored at loc^n 300 with add field at loc^n 301. The add. field has the value 400. The processor reg. R_1 contents the no. 200. Evaluate the eff add (add of data) of add. mode of instruction is direct, indirect, immediate, relative, and indexed with R_1 as a indexed register.?



i) direct : 400 (add. of data)

ii) indirect = 500

iii) Relative = A_1 = PC (301) 400 = displacement

PC = 302

= 400 (2's complement)

702

iv) immediate = 301

v) index = 200 + 400 — displacement

= 600

8/2/18

To complent operⁿ of instructⁿ = instructⁿ cycle.

↓

CPU passes through diff phases till completion process.

Suboperⁿ that perform by CPU to complete the operⁿ of instructⁿ

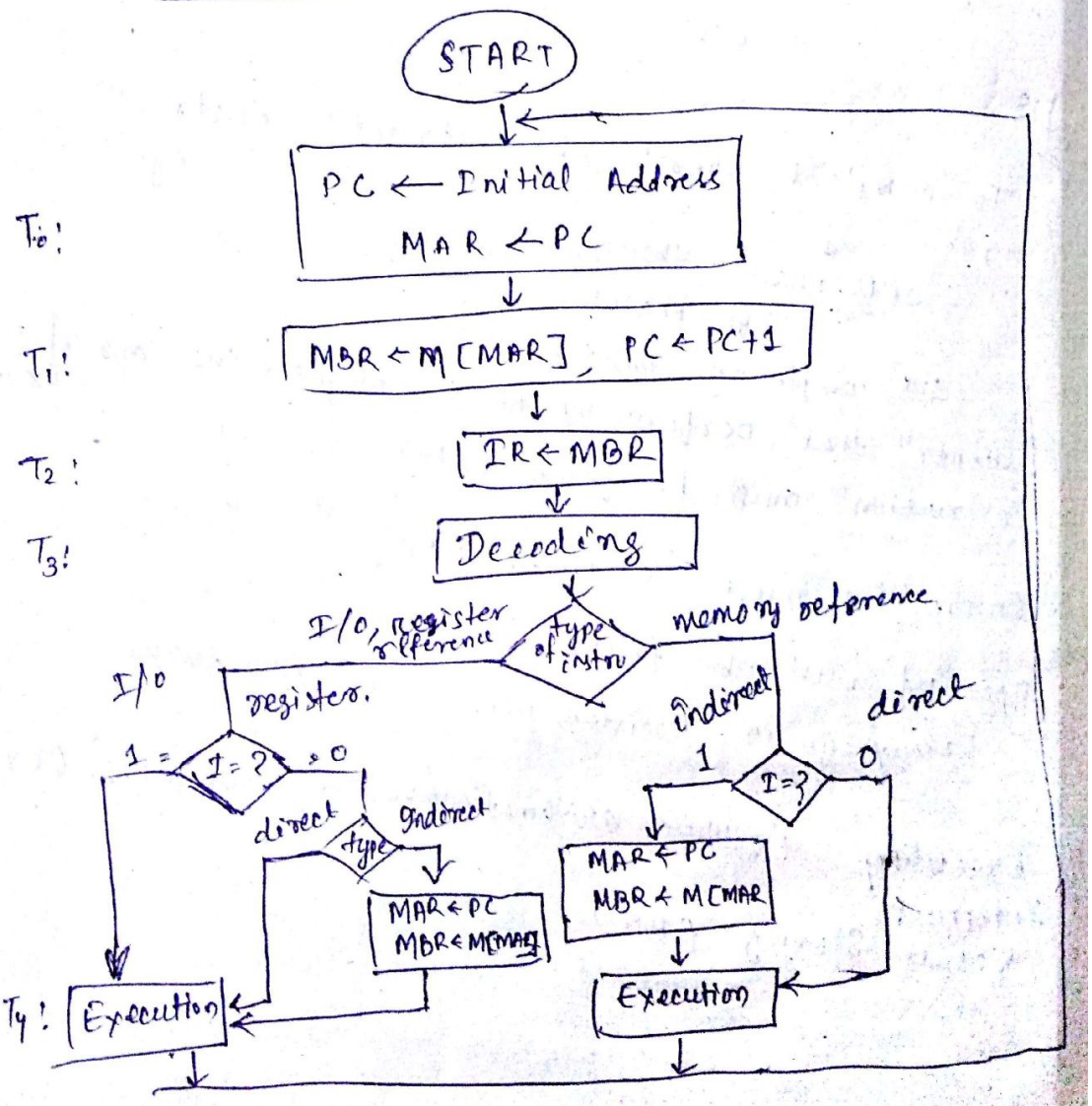
1. Instruction must be fetch (IF)
2. Instruction must be decoded (ID)
3. Operand must be fetched (OF)
   - may be in register, memory loc, Input devices.
4. Execution (either arithmatic & logical computation) (EX)
5. Interrupt
6. Result Storing (RS)

Instruction Cycle
- Fetch cycle
- Execution Cycle or Computation cycle
  - Instruct^n cycle :- when control decoding the instr
  - Indirect/direct cycle
  - Interrupt cycle

any interrupt generated either by external instruct^n.

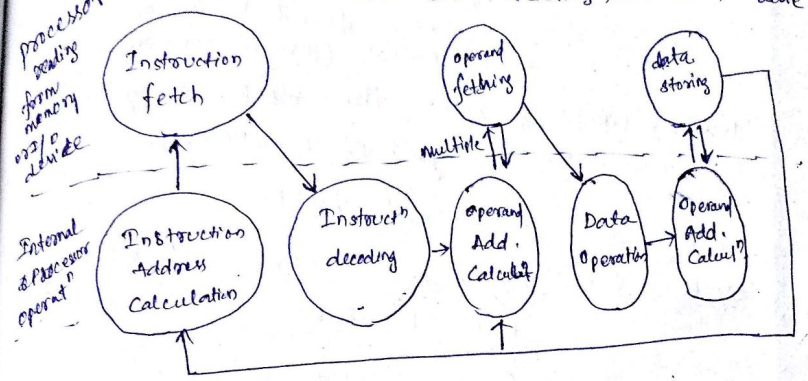
eg - sthg. divided by zero

INSTRUCTION FLOW DIAGRAM



INSTRUCTION FLOW STATE DIAGRAM

~~CPU either reading from memory I/O device~~

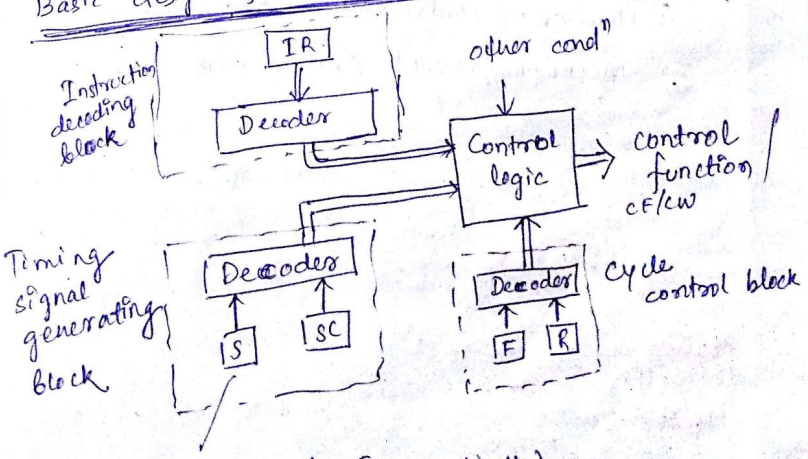


9/2/18

Control Unit - of body - nerve system.

* Synchronization is done by control unit, by producing diff. control signals.
* practically it is ~~done~~ distributed to every where of the system.

Basic design of control unit -



- instructions are in register
- instruction decoded by decoder
- what signal pass to control logic

Timing signal - provides the clock to system to synchronize everything
- start-stop f/f FF

Cycle control - cycles are controlled by 2 F/F 'F' & 'R'

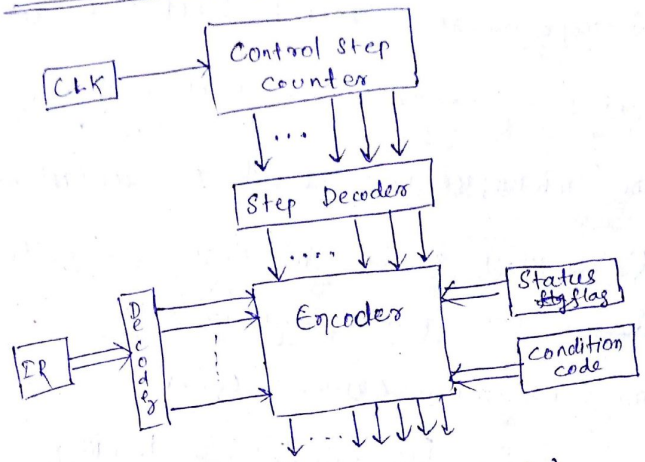
FF	
00	fetch cycle
01	direct / indirect cycle
10	execute cycle
11	Interrupt cycle

Control logic - designs what are the control signals to be generate & which component to be communicate

Control Unit

2 types of control unit -
1) Hardwired control unit - Risk Processor
2) Microprogrammed Control unit.

Hardwired Control Unit



Clock provides clock signal to control step counter.

Counter track of all instructions.

Acc. to instruct^{n} act^{n} it provides signal to step decoder. It gives neccessary signal to encoder & the decoded signal enters to encoder by decoder. Any cond^{n} from ext. agent passed to encoder. flag register is changed. Taking all these in an account, either one or more than one control signal will active. (decides which one will be active)

This is very fast operat^{n}.

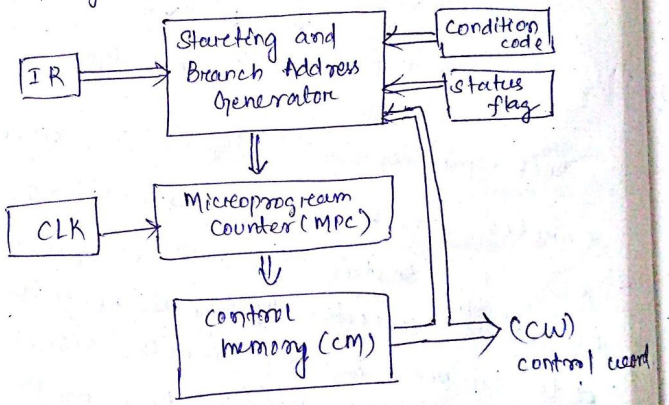
If we will go for processing part - time is very critical.

(each & every fract^{n} of time is counted)

15/02/18

Microprogrammed Control Unit

- set of micro instruct^n - microprogram.
- The microprogram reside in control memory
- Using that control memory we will design the micro program i.e microprogram control unit.



- control word (CW) ⇒ comb^n of control signals.
- If cond^n satisfied, then only it will branch address of
If cond^n, not " , then next instruct^n will executed.
- Main / Macro instruct^n.

Required control signal generated by control memory.

- Starting & br
cond^n, status
- Generally
- V. Wilkes
Microprog

Wilkel m

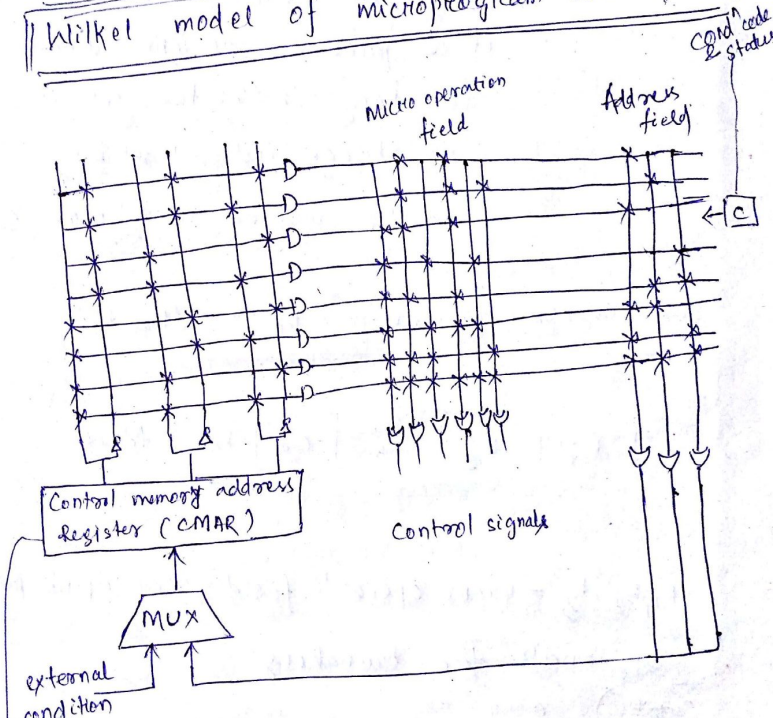
Control mem
Register (

external condition
↓
i/p requir
for

gener

- Starting & branch address generator depends on cond^n, status flags & prev. microprog. instruct^n.
- Generally it is a ROM.
- V. Wilkes proposed Wilkel model of Microprog. control unit.

Wilkel model of microprogram control unit



↓
i/p required for starting address.

→ generate address – depending on this particular microinstruct^n will selected.

After execut^n, if the cond^n will satisfy, add. field will give you add. of next instruct^n.

If, add. field not satisfy, then the next micro op^n will be executed.

Hardwired CU is faster. (Adv.)

- hardwired complexity high
- you can't change instruction set

Adv. of Microprog. CU

- flexible
- hardwired complexity less.
- if a processor designer wants to change instruction set, one can change instruction set by updating writable control memory (WCS)

Basic things present in Micro instruction:-

F_1	F_2	F_3	CD	BR	Address

F_1, F_2, F_3 - micro operation field (like opcode part)

CD - condn for branching (2 bit size)

	CD	
1 =	0 0	- unconditional branch (continue)
	0 1	- indirect address
	1 0	- sign bit Accumulator
	1 1	- Zero in Accumulator

(not set)

BR - branch

0 0 - JUMP (to this address)

0 1 - CALL (which is present in this address)

1 0 - RET (return when branch to, return to a return address)

1 1 - MAP (maping - from micro oper^n field, sometimes to be destined)

when CD = 00 → 1 (set cond^n).

at that time address part will give next address

CMAR = CMAR + 1

@ MAP - form microinstruct^n to next microinstruct^n

Assignment

What do you mean by horizontal micro instruct^n, vertical microinstruct^n?

16/02/18

STACK

- contiguous memory location specified by the user.
- stack pointer - always points to top of stack.
- LIFO (last inserted data can be retired first)
- Generally from highest memory loc^n stack is defined.

eg - In 8085 processor

- various types of stack - ascending stack
  descending "

Ascending stack - stack pointer in ascending stack

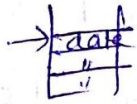
stack pointer = stack pointer + 1

Descending stack

stack ptr = stack ptr - 1

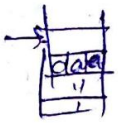
- full stack — stack ptr points to top of stack where data is present
  Empty stack — stack ptr points to top empty loc^n where data can be inserted.

Ascending



(full stack)

Sp = sp + 1
data ← enter



(empty stack)

data ← enter
Sp = sp - 1

Descending

(full stack)

sp = sp + 1
data enter

(empty stack)

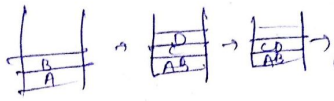
data - enter.
sp = sp - 1.

A*B+C*D

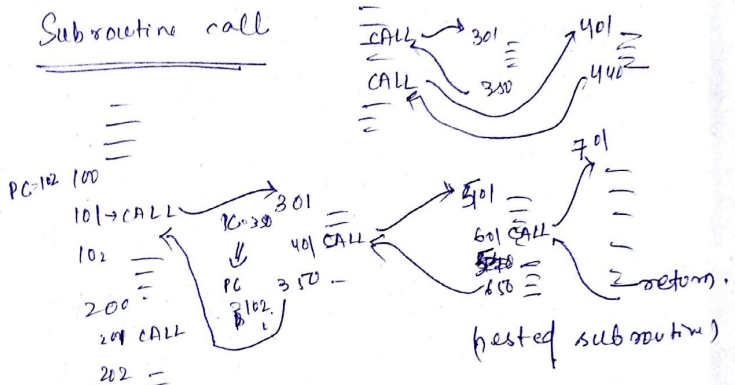
+*AB*CD (prefix / polish notation) PN

AB*CD*+ (postfix / Reverse Polish) RPN notatn

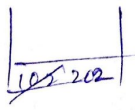
evaluation



Subroutine call



(nested subroutine)



Nested subroutine

eg

PC = 102

~~301~~

~~402~~

~~501~~

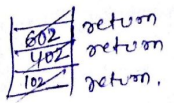
~~602~~

~~701~~

~~602~~ 602

402

102



PC = 102