module - 2

Computer Components

Components :—

Computer System contains three fundamental components named as:

CPU, memory, I/O.

CPU organization :—

CPU contain, 3 internal components used to process the instructions name as:

1. Registers
2. ALU
3. Control unit.

Registers :—

Register is a internal storage component in the CPU therefore CPU contain a set of ~~general purpose~~ mendatory Registers. they are:

1) PC : (Program counter) Holds the instruction Address (opcode address)

2) IR : Holds the currently fetched opcode, to decode

3) ACC : Holds the ALU input and ALU o/p.

4.) TR : Holds the ALU 2nd operand.

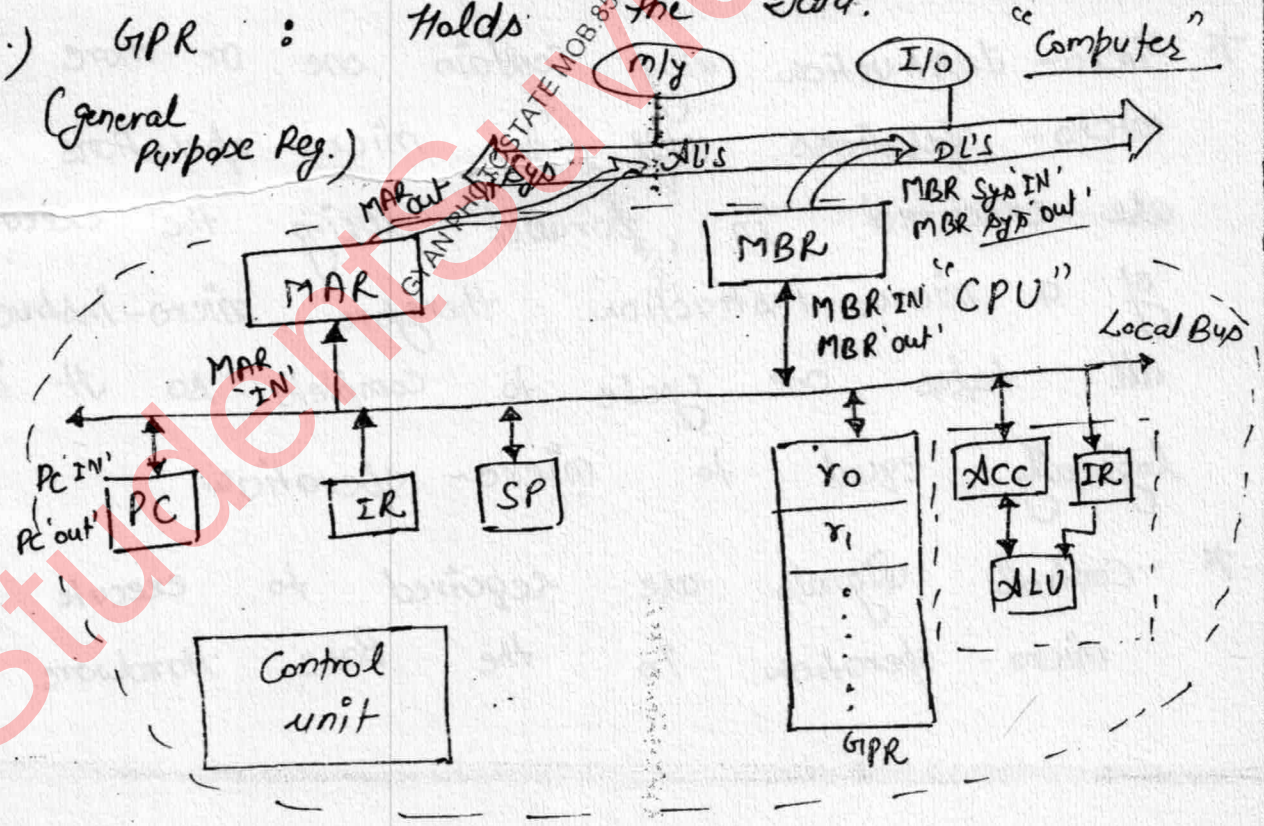
5.) MAR : Holds the m/y Address, Connected to a Address lines of the System bus

6.) MBR : Holds the m/y content.
(m/y Buffer Reg.) Connected to the data lines
(or) of the System Bus.
(MDR)

7.) SP : Holds the Top of the stack Address
(Stack Pointer)

8.) Flag Reg : Holds the Status of an Inst^{n}.

9.) GPR : Holds the Data.
(General Purpose Reg.)



* MBR Contain 4 control Signals.

"Micro operation" :-

Date : 17/08/2017

★ Micro-operation is a premitive operation in the base Hardware.

★ Register to Register transfer operation is a one kind of micro-operation.

★ Micro-operation on average takes 1 cycle to complete.

★ When the micro-operation contain <u>m/y reference</u> <u>then</u> it <u>takes</u> <u>'more than one cycle'</u> to complete.

★ Micro-Instruction may contain one or more micro-operations. All the micro-operations are executed in parallel during the execution of a micro-instruction therefore micro-instruction all takes one cycle to complete so it is logically equal to micro-operation.

★ Control signals are required to execute the micro-operation in the Base Hardware.

—: machine Instn:

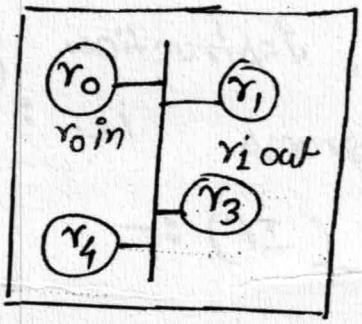
mov r_0, r_1

↓

Register Transfer language (RTL) : $r_0 \leftarrow r_1$

↓

H/w design :



↓

μ opⁿ list :

↓

μ Program

T_1 : r_1 out, r_0 in

Control signals

Add r_0, [2000]

↓

$r_0 \leftarrow r_0 + m[2000]$

↓

H/w design

↓

μ operation list

↓

μ Program

↓

Control Signals

machine Instn:

mul r_0, @2000

↓

$r_0 \leftarrow r_0 * m[[2000]]$

↓

H/w design

↓

μ opⁿ list

↓

μ program

↓

control signals.

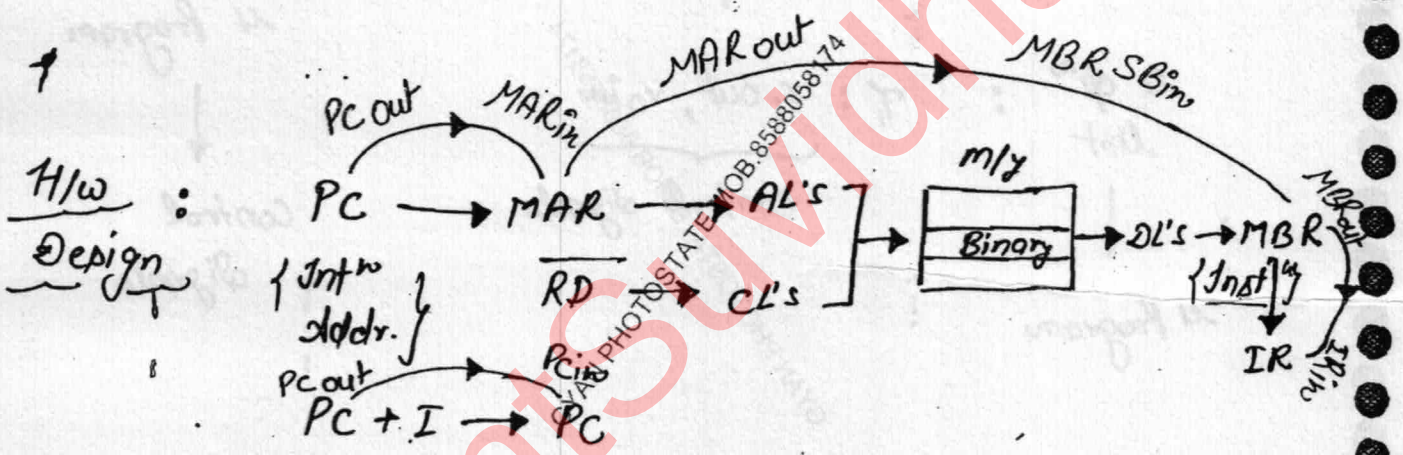
Micro- Program :-

Sequence of a micro-operations used to perform the meaningful action in the base Hardware is called as micro-program.

Let us consider, Instruction cycle to derive the micro-programs i.e. :

1) Instruction fetch (IF) :-

μ-Program is :



μ-Program :

T_1 : PC → MAR ; PCout, MARin

T_2 : M[MAR] → MBR ; MARout, MBRSBin } System Bus
M/y PC + I → PC ; PCout, INC, PCin } Local Bus

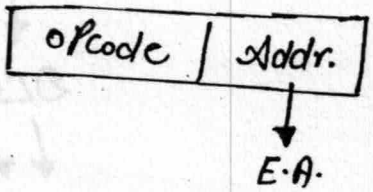
T_3 : MBR → IR ; MBRout, IRin

2.) Operand Fetch :-

Operand fetch using Direct Addressing mode.

:- μ-Program ip :

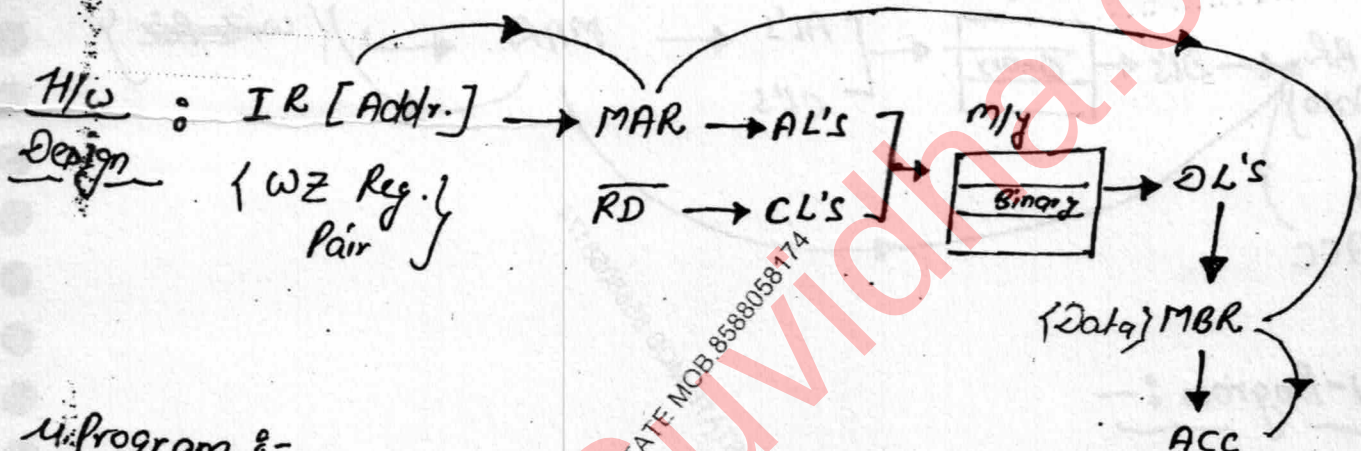
Instn :



eg: Load [2000]

ACC ← m[2000]

H/w Design :



μ-Program :-

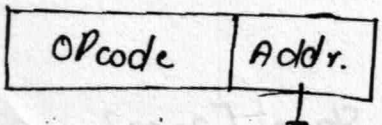
T_1 : IR [Addr.] → MAR ; IR Addr out, MAR in

T_2 : M [MAR] → MBR ; MAR out, MBRSBin } System Bus

T_3 : MBR → ACC ; MBR out, ACC in

:- Operand fetch using "Indirect mode" μ-Program ip :

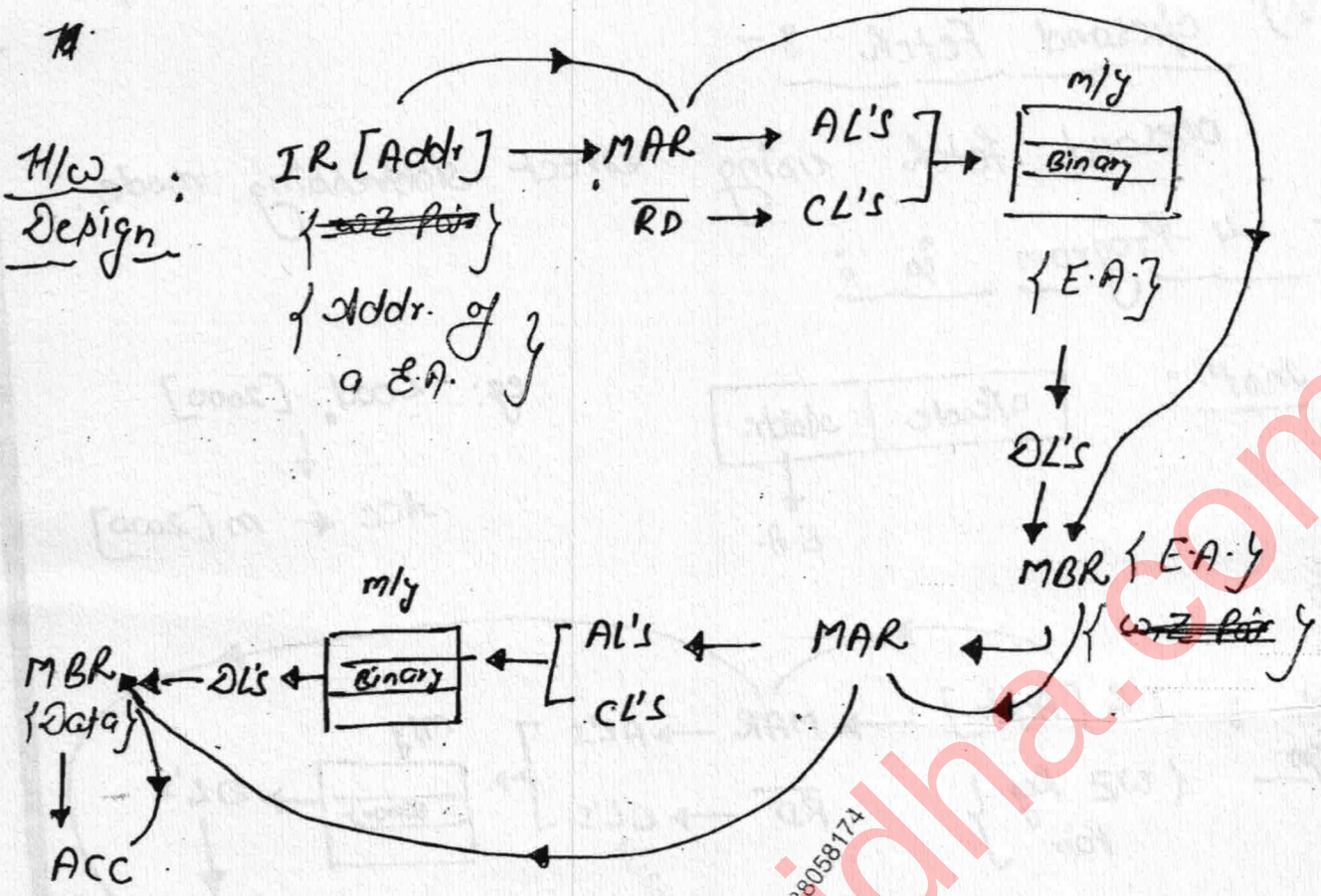
Instn :



eg: Load @ 2000

Acc ← m[[2000]]

H/w Design :



μ-Program :-

T_1 : IR [Addr.] → MAR

T_2 : M [MAR] → MBR

T_3 : MBR → MAR

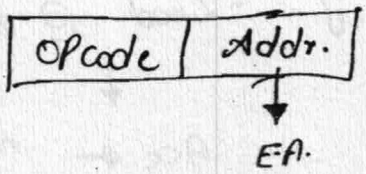
T_4 : M[MAR] → MBR

T_5 : MBR → ACC

(4) Operand store using, Direct Addressing mode

μ-Program is :

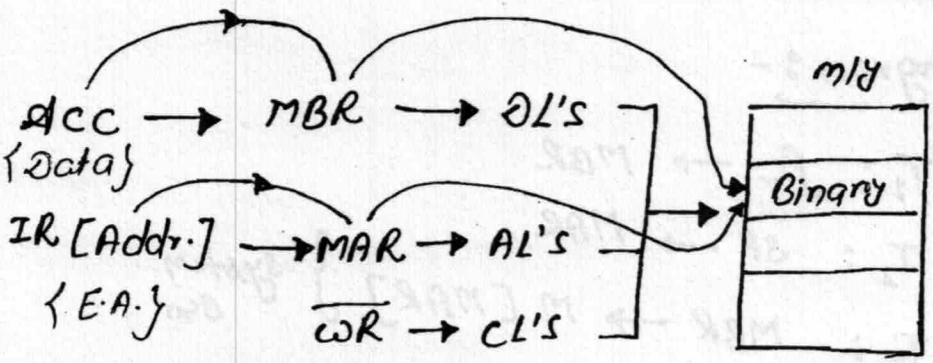
Instn :



eg: Store [2000]

M[2000] ← ACC.

H/w Design :



μ-Program :

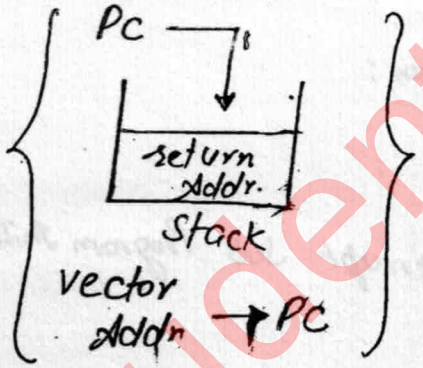
T_1 : ACC → MBR ; ACCout, MBRiw

T_2 : IR [Addr.] → MAR ; IR Addout, MARiw

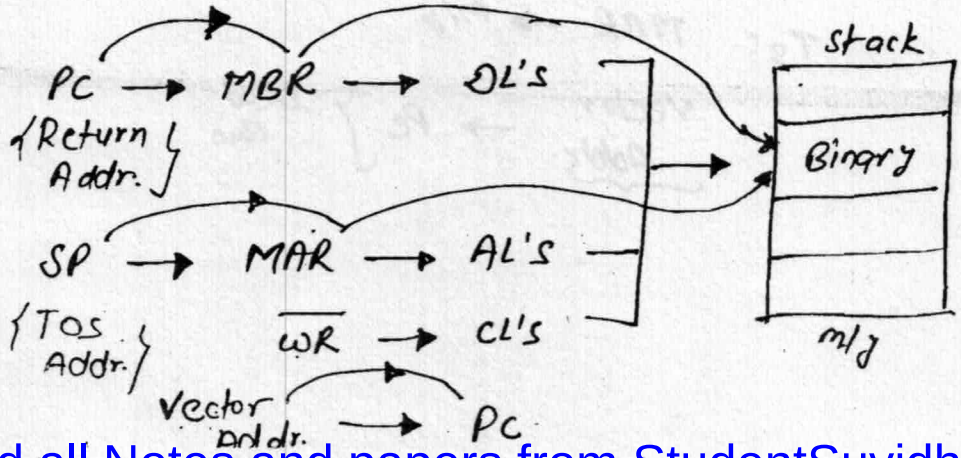
T_3 : MBR → M[MAR] ; MBRSBout, MARout
m/y

Interrupt Subprogram Initialization :-

μ-Program is :



H/w Design :



4- Program :-

T_1 : PC → MBR

T_2 : SP → MAR

T_3 : MBR → M[MAR] } System Bus

Vector Addr. → PC } Local Bus

Gate-2013

Que :- Consider the following micro-Program :-

T_1 : PC → MBR

T_2 : (X) → MAR

T_3 : MBR → M/y

(Y) → PC

which one of the following operation is satisfied by the above micro-Program :

a) ~~OF~~ IF b) OF

c) Conditional Jump d) Interrupt Sub Program Initialization

T_2 : SP → MAR

T_3 : MBR → M/y

Vector Addr. → PC } local Bus

"Control unit Design":-

★ Control unit design, 'brain' of the CPU.

★ Pre-Requirements of a control unit Design is :-

(a) How many control lines in the Base H/w.

(b) How many machine - Inst^{n} are implemented in the Base Hardware.

(c) How many micro-operations are required for each machine Inst^{n}.

(d) what are the control signals required for each micro-operation, for each machine Instruction.

After finalize the above requirements, Control unit is implimented by using the following approaches:-

1.) Hardwired approach.

2.) Micro-Programmed Approach.

"Hardwired control unit Design":-

★ In this design control signal is expressed in a Sum of Product (SOP) format.

★ SOP Expression is directly realized by the Independent Hardware called as "Hardwired control unit".

★ It is fastest control unit.

★ It is used in the Real-time Applications.

★ Even a Minor Modification requires Re-design and Re-Connection of a control signals Hence It is not flexible.

★ It is not suitable in the Design and Testing Places.

★ RISC control unit is a Hard-wired control unit.

Sample CU :-

① # CS's in the H/w : $\{S_0, S_1, S_2, S_3\}$

② # Instn in the CPU : $\{I_1, I_2, I_3\}$ → Add, mul, Div

③ # u opn / Instn : $\{T_1, T_2, T_3, T_4\}$

④

u opw / Instw	I_1	I_2	I_3
T_1	S_0, S_1, S_2	S_0, S_3, S_1	S_2, S_1
T_2	S_0, S_3	S_2, S_3	S_3, S_1
T_3	S_0, S_2	S_1, S_2	S_2, S_3, S_0
T_4	S_0, S_1	S_1, S_3	S_0, S_3

:- Hard-wired CU :-

[CS: SOP format]

$$\begin{cases} S_0 : T_1(I_1+I_2) + T_2 I_1 + T_3(I_1+I_3) + T_4(I_1+I_3) \\ S_1 : \underbrace{T_1(I_1+I_2+I_3)}_{T_1} + T_2 I_3 + T_3 I_2 + T_4(I_1+I_2) \\ S_2 : T_1(I_1+I_3) + T_2 I_2 + T_3 \\ S_3 : T_1 I_2 + T_2 + T_3 I_3 + T_4(I_2+I_3) \end{cases}$$

Micro-Programmed Control unit :-

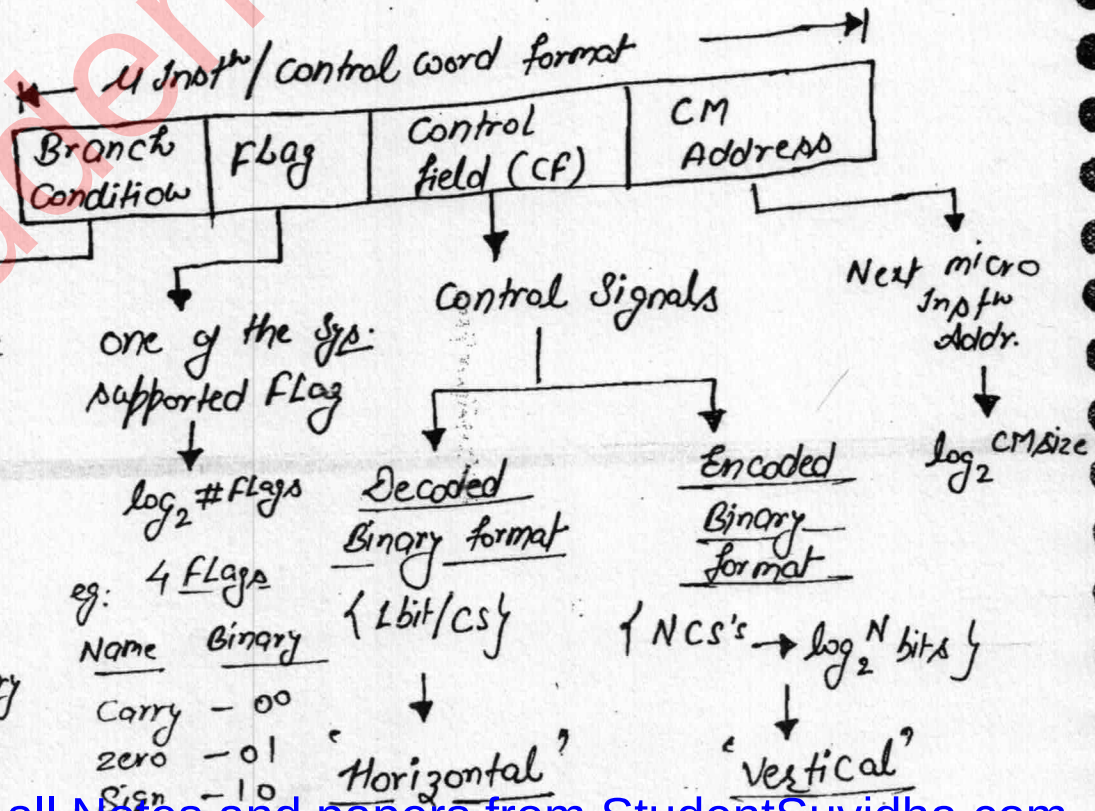
★ In this design control memory is programmed with a micro-program.

★ Control memory is a permanent memory i.e. ROM

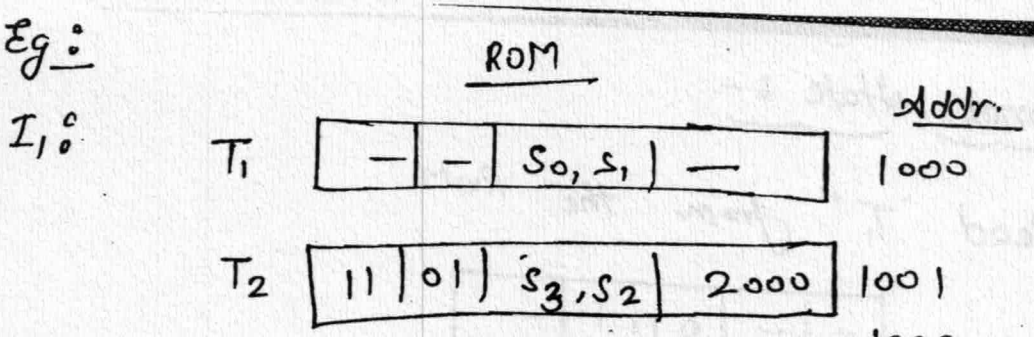
★ In this design micro-sequencer unit is present, used to generate the micro-instruction address in a sequence.

★ Control memory associated with a CAR and CDR. registers used to hold the control m/y address and control m/y content respectively.

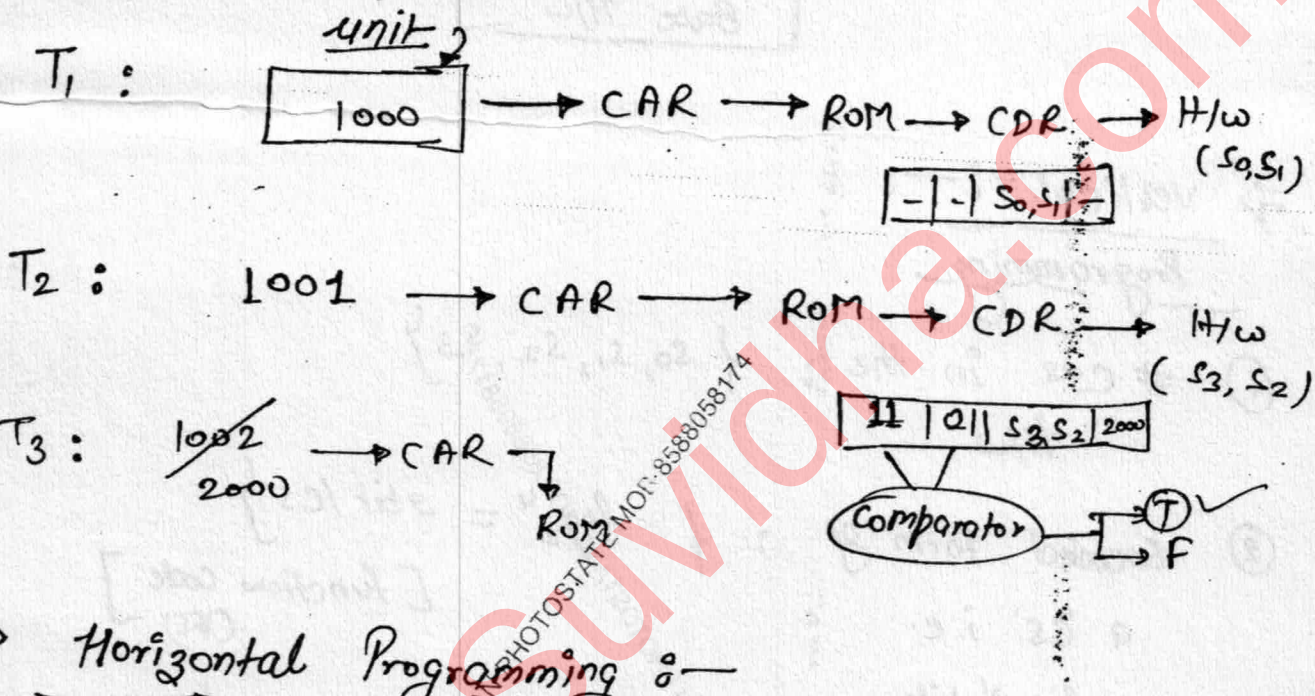
★ In the control memory micro-instruction is stored in following format :



Eg:



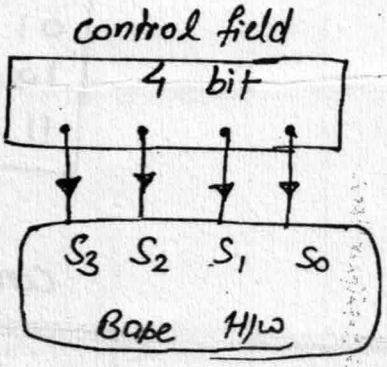
Execution :- μ-sequencer



⇒ Horizontal Programming :-

① # CS's in the H/w : { S0, S1, S2, S3 }

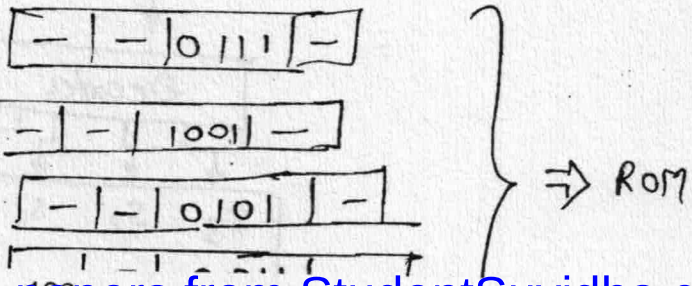
② Decoded form of a CS' i.e. 1 bit / CS :



1 bit → 0 : Disable
1 : Enable

③ μ Instn Design :

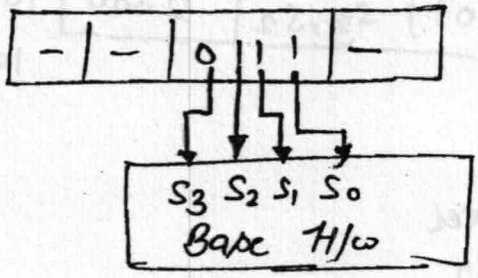
(1) T1 : { S0, S1, S2 }
T2 : { S0, S3 }
T3 : { S0, S2 }
T4 : { ... }



④ Operational State :-

(I) Read 'T_1' from the ROM

T_1 :



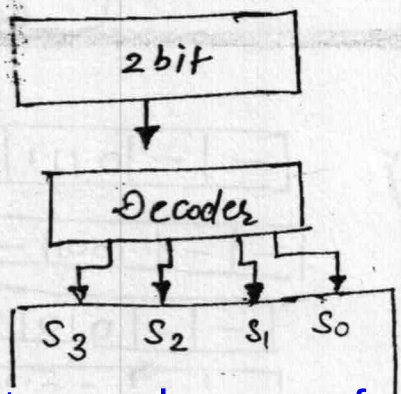
⇒ Vertical :- Programming

① # CS's in the H/w : { S_0, S_1, S_2, S_3 }

② Encoded form of a 'CS' i.e. $\log_2 N$ bit : { $\log_2 4 = 2$ bit /CS } [function code (FC)]

FC	CS
00	S_0
01	S_1
10	S_2
11	S_3

Control field



③ μ Instn Design :

(I) T_1 : { S_0, S_1, S_2 }

–	–	00	01	10	–

T_2 : { S_0, S_3 }

–	–	00	11	–

T_3 : { S_0, S_2 }

–	–	00	10	–

T_4 : { S_0, S_1 }

–	–	00	01	–

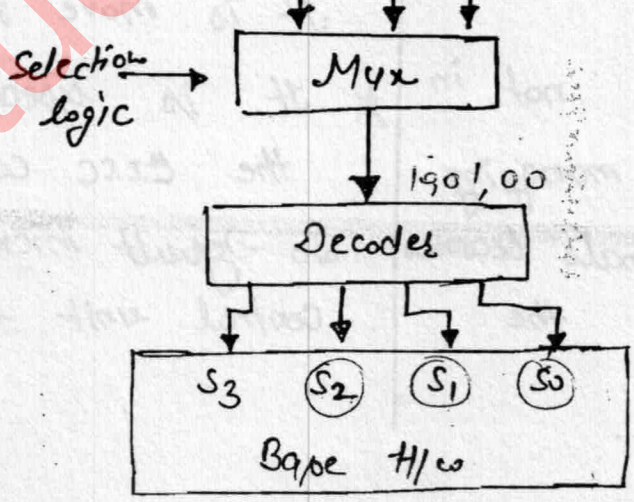
⇒ ROM

④ operational state :-

(I) Read 'T_1' from the ROM

T_1 :

–	–	00	01	10	–



Horizontal vs vertical :-

Horizontal	Vertical
★ In this design control signal is expressed in a decoded binary format i.e. 1 bit / c.s.	★ In this design, control signal is expressed in a encoded binary format. i.e. $\log_2 N$ bit format.
★ It supports longer control word with a fixed control field.	★ It supports shorter control field.
★ No need of a external decoders to generate the control signal, so it is faster than vertical	★ Need of a external decoders to generate the control signals so, it is slower than Horizontal.
★ It allows high degree of parallelism i.e. None or more than 1.	★ It allows low degree of parallelism i.e. None / 1. (None or 1).
★ It is bit flexible compared with Hard-wired control unit.	★ It allows easy implementation of new instructions so it is more flexible.
★ This concept is not in practice because managing the decoding signals becomes very complex in the Hardware.	★ It is used in the CISC computers. So Default micro-programmed control unit is vertical.

★ Note :- 'Ascending order' of a control unit design :-

a) In terms of speed :

vertical < Horizontal < Hard-wired

b) In terms of flexibility :

Hard-wired < Horizontal < vertical.

Que :- Consider a Hypothetical control unit which supports 32k control word m/y. Hardware contain 64 control signals and 8 Flags. what is the size of a control word in bits and control memory in bytes using

a) Horizontal Programming.

b) vertical Programming.

Soln :-

Horizontal :-

- # CS's in the H/w : 64
- Decoded form of a 'cs' : 64 bits
  i.e. 1 bit/CS
- μ Instn Design [Default with 1 'CS']

BC	flag	CF	CM Addr.

— $\log_2 8 = 3$ bit, 64 bit, $\log_2 32K = 15$ bit

* HCW size = 82 bits

* Horizontal control m/y size = 32K CW

↓

32K * CW

↓

32K * 82 bits

↓

$\left(\frac{32K \times 82 \text{ bits}}{8}\right)$ Bytes.

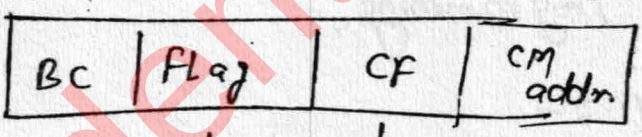
Vertical :-

CS's in the H/W = 64

Encoded form of a C.S. = $\log_2 64$

= 6 bit/C.S. [FC]

* u inst^n Design [Default with 1 CS]



BC → $\log_2 8$ → 3bit

Flag → 1CS → 1FC → 6bits

CF → $\log_2 32K$

* VCW size = 24 bits

* VCM size = 32 kCW

↓

32 * CW = 32K * 24 bits = $\left[\frac{32K \times 24}{8}\right]$ Bytes

Que:- A Micro-Programmed Control unit supports 400 Control Signals. Micro Instⁿ is designed in such a way that 4 signals are Active what is the size of a Control field in the micro-Instⁿ.

Solⁿ:-

Micro-Programmed Control unit

By default = {vertical}

400 CS's so :- $\log_2 400$ = 9 bits required

4 Signals are Active

so :- 4 FC (Functⁿ code) Needed.

= 9×4 = 36 bits in the Control field (CF).

Que:- Consider a CPU which supports 600 Control lines, uses the Horizontal micro-Programmed Control unit to design the signals. Micro-Instruction is designed with 40 signals what is the size of a Control field in the micro-Instⁿ.

Horizontal micro Programmed CU

Decoded form of CS's = 1 bit/CS

so 600 bits.

| BC | Flag | CF | CM Add |