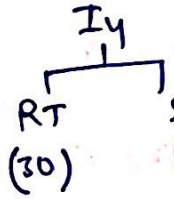
Q. Consider a hypothetical CPU which supports 4-interrupts (I_1, I_2, I_3, I_4) with the resp. service time of (20ns, 40ns, 60ns & 30ns). Response time of an interrupt is 2ns. I_1 has the highest priority. I_4 has the least priority. What is the range of time required to service I_4 when the interrupts may or may not occurs simuntaneously.

Total Time = Response time (RT) + Service time (ST)

Minimum time required for I_4 (without simuntaneous occurrence) = only I_4 intr present

i.e



$= 32ns$

Maximum time required for I_4 (simuntaneous occurrence) = All 4 Interrupts present

$= I_1 + I_2 + I_3 + I_4$

$= (2+20) + (2+40) + (2+60) + (2+30)$

$= 158ns$

[Rang = 32ns to 158ns]

Q. Consider 32-bit hypothetical CPU operated with a 500 MHz clock. Following program is executed in the given CPU. Program is stored in the memory with a starting address $(1000)_{10}$. System supports word addressable memory. Memory refrence consumes 4-cycle. ALU op^n consumes 3-cycle. And register refrence (RR) consumes 0-cycles.

Inst^n	Meaning	size in Bytes
MOV r0, (2000)	r0 ← M[[2000]]	12 (3w)
MOV r1, [3000]	r1 ← M[3000]	8 (2w)
Add @4000, r1	M[[4000]] ← M[[4000]] + r1	12 (3w)
MUL r0, r1	r0 ← r0 * r1	8 (2w)
MOV 3(r0), r1	M[3 + [r0]] ← r1	8 (2w)
Halt	M/c halts	4 (1w)

(a) If an interrupt occurs during the execution of Halt inst^n. what could be the return Address pushed into a STACK.

(b) Program execution time ?

(a) Mem storage

I_1 : 1000 — 1002

I_2 : 1003 — 1004

I_3 : 1005 — 1007

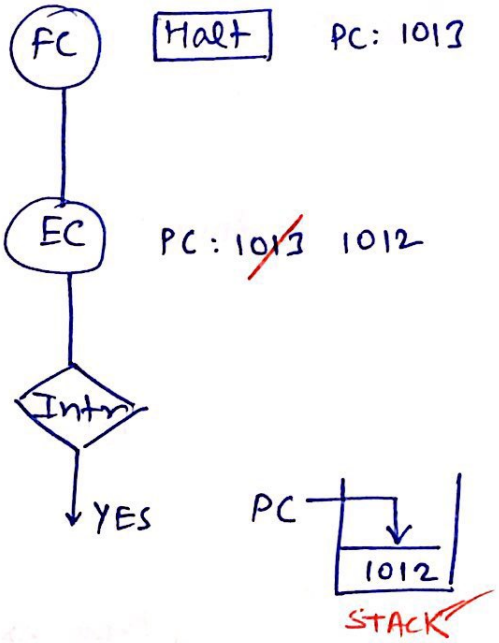
I_4 : 1008 — 1009

I_5 : 1010 — 1011

I_6 : 1012

1013

⋮



(b)

	FC	EC				
	IF	ID	OF		PD	WB
			S1	S2		D
I_1	1MR	2MR	2MR	X	X	1RR
I_2	1MR	1MR	1MR	X	X	1RR
I_3	1MR	2MR	2MR	1RR	1ALU	2MR
I_4	1MR	1MR	1RR	1RR	1ALU	1RR
I_5	1MR	1MR	1RR	X	X	1RR + 1ALU + 1MR
I_6	1MR	X	X	X	X	X

Program Exec. time = $[(21MR * 4C) + (3ALU * 3C)]$

Cycle time (C) = $\frac{1}{\text{Clk freq (Hz)}} = \frac{1}{500MHz}$

$C = 2ns$

so, (Prog exe time = 186 ns)

⇛ RISC Vs CISC

RISC (Reduced Inst^n set computer) characteristics –

- supports more register
- reg-to-reg rebrence CPU
- support less addressing modes
- support fixed-length Inst^n
- Implement Pipelining successfully
- CPI (cycles per Inst^n) = 1
- It is a super computer

- It is used in real time application
- It is an Expensive computer
- It contains smaller Instruction set
- It uses hardwired control unit

Eg:- Motorola processors, Power-PC processors, ARM-processor

CISC (complex Inst^n set computer) charactmistics:-

- Supports less registers
- supports more Addressing modes
- It support variable length inst^n.
- Pipeline is not implemented
- CPI $\neq 1$
- It is a general purpose computer, used for personal application
- Less Expensive processor
- Contains larger Inst^n set

- It uses micro-programmed CU

Eg :- Intel Pentium series computer

Register organization in RISC CPU :-

- RISC procusor supports more registers categorized into 4 groups :

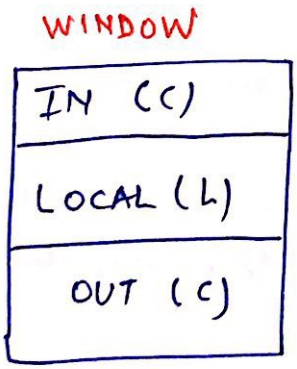
(i) Global Registers (G)

(ii) Local Registers (L)

(iii) IN - Registers (C) — common

(iv) OUT- Registers (C)

- In the RISC CPU register window is formed by grouping the IN, LOCAL & OUT Registers.

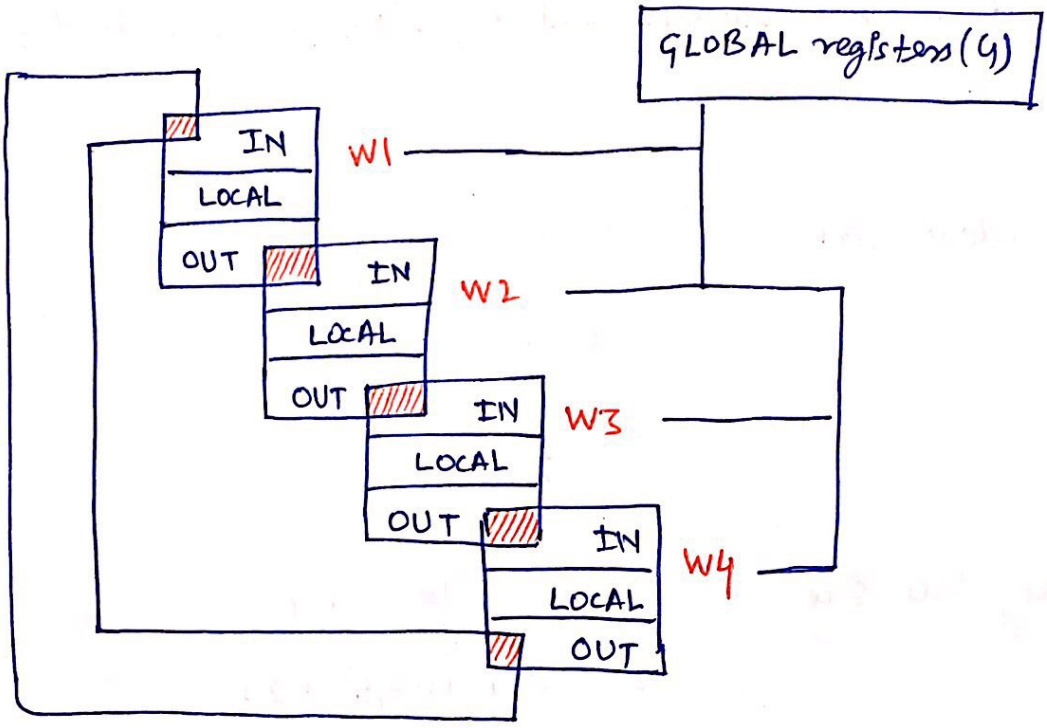


- Global registers are accessible by all the windows therefore no. of reg present in the window is -

Window size = L + 2C + G

L → LOCAL
C → IN + OUT
G → GLOBAL

- In the RISC cpu register windows are organized in a overlapping order. One window OUT reg are used as IN-register in another window.



* no. of register in CPU = W(L+C) + G

where W is no. of windows

L " LOCAL REG

C no. of IN or OUT (anyone)

G no. of GLOBAL Reg

[* since IN and OUT reg are overlapping so count them as 1 not as 2]

Q. Consider a hypothetical CPU which supports 32-register windows organized in a overlapping order. CPU contain 20 global reg, 10 local reg, 15-IN reg, 15-OUT reg. What is window size and register file size in the CPU.

Window size = $L + 2C + G$

$= 10 + 2(15) + 20$

$= \underline{60}$

Reg-file size = $W(L+C) + G$

$= 32(10+15) + 20$

$= \underline{820}$

(COMPUTER ORGANIZATION)

Computer system contains 3 - fundamental components named as - CPU, memory and I/O.

CPU organization :-

CPU contains 3 internal components used to process the instⁿ named as registers, ALU, control unit (CU).

Register :-

- It is a internal storage component in the CPU.

Mandatory registers are -

(i) PC (program counter) : holds the Instⁿ addr

(ii) IR (Instⁿ registers) : Holds the Instⁿ (opcode)

(iii) ACC (Accumulator) : Holds I/p & O/p of ALU

(iv) TR (Temp reg) : Hold source 2 (not visible to user)

(v) MAR (~~Holds Reg~~ Mem Address Reg / ~~mem Addr~~) : Holds the memory address, connected to Address BUS.

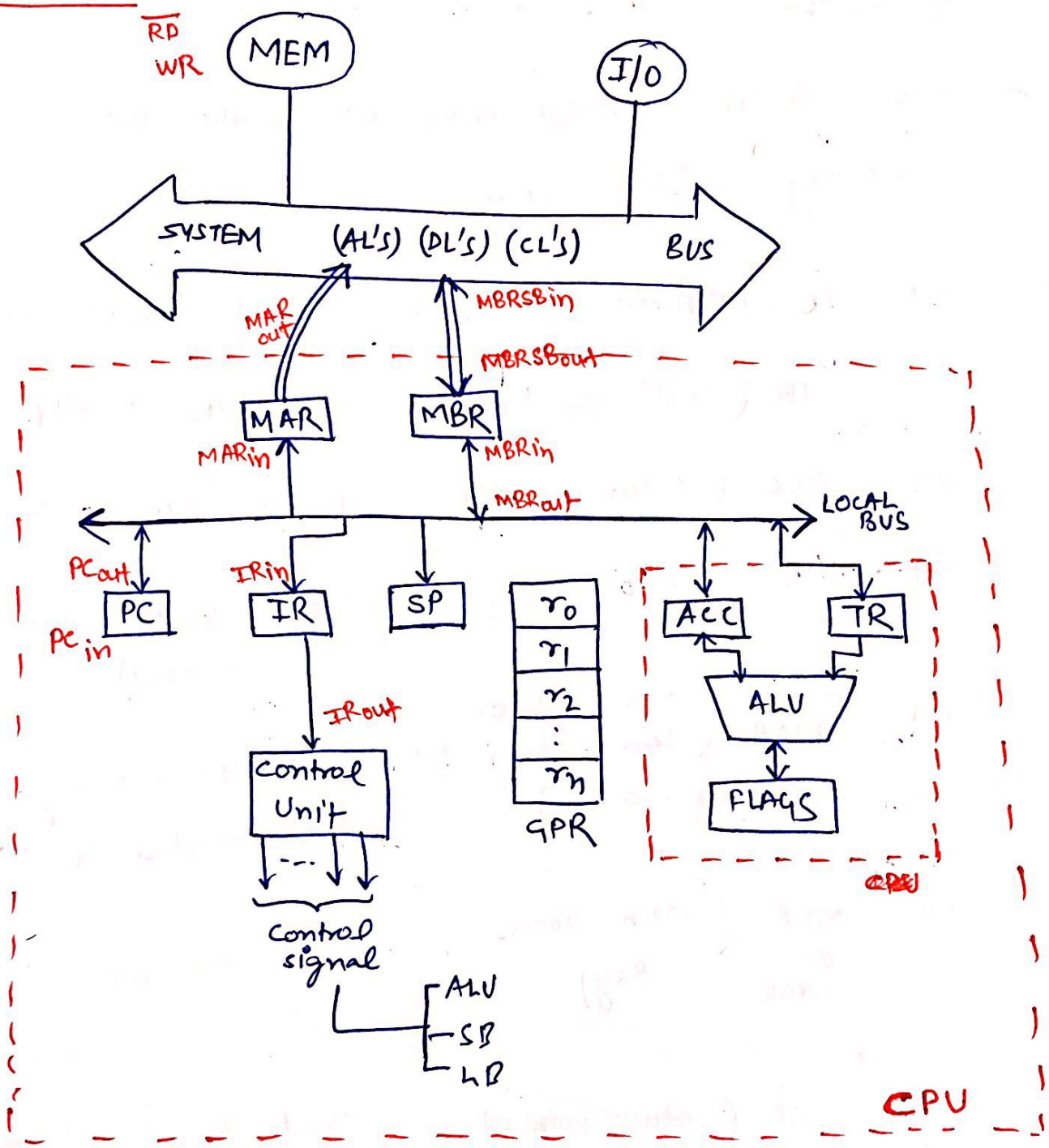
(vi) MBR or MDR (Mem Data Reg) : Holds the mem content

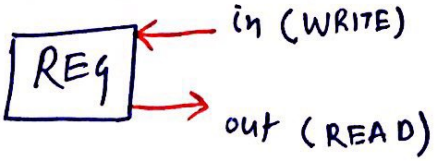
(vii) SP (stack pointer) : Holds the Top of stack address

(viii) Flag registers : Holds the Instⁿ status

(ix) GPR (general purpose reg) : Holds the Data

Organization -





Micro- Operation :-

- Micro-op^n is a primitive or fundamental or elementary op^n in the base h/w.
- Op^n which is directly executed in the base H/W without Further decomposition is called as micro-op^n.
- Op^n which completes its execution within 1-cycle is known as micro- op^n.
- Micro-op^n design is the responsibility of the designer. because designer only knows the control signal in the base H/W.
- User -point of view Reg-to- Reg transfer op^n is a one-kind of micro -op^n.
- Micro -inst^n is designed by the designer which contain one -or -more micro-op^n still it takes 1-cycle to complete because all the micro -op^n which are defined in the micro--inst^n is running in parallel.

$$\mu \text{ Inst}^n \longrightarrow 1 \text{ or } \mu\text{-op}^n \longrightarrow 1\text{-cycle}$$

$$\mu \text{ Inst}^n \longrightarrow 1000\ \mu\text{-op}^n \longrightarrow 1000\text{-cycle}$$

Micro-prog is a sequence of micro-op^n used to perform the meaningful action in a base H/w.

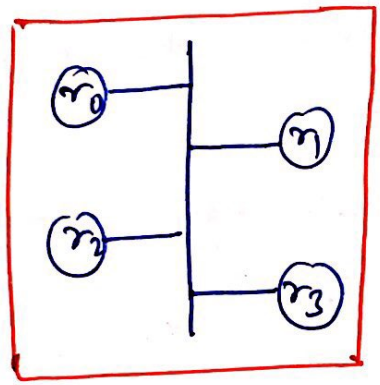
Machine inst : MOV r_0, r_1

↓

Register transfer Lang (RTL) : $r_0 \leftarrow r_1$

↓

H/w Design :



↓

Mop^n list : T_1 : $r_{1\,out}$, $r_{0\,in}$

control signals

↓

M-prog

Add r_0, [2000]

↓

$r_0 \leftarrow r_0 + M[2000]$

↓

H/w Design

↓

Mop^n List

↓

M-prog

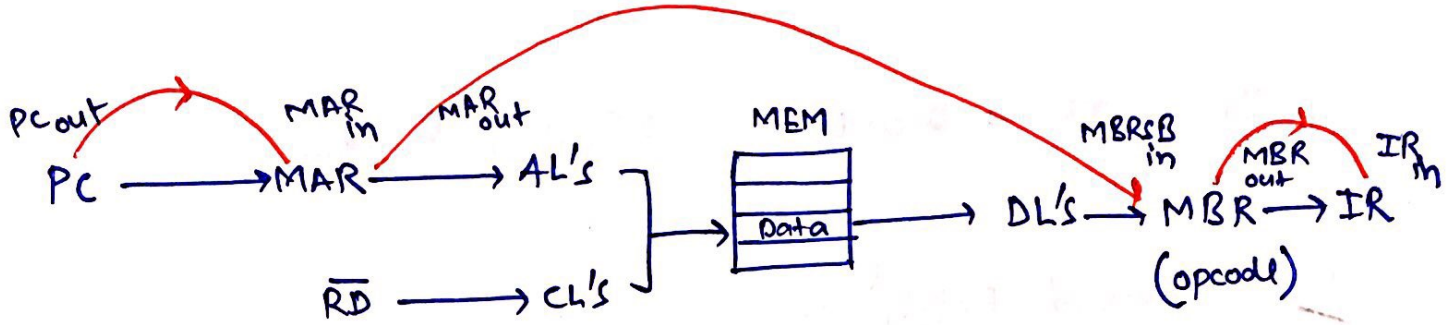
↓

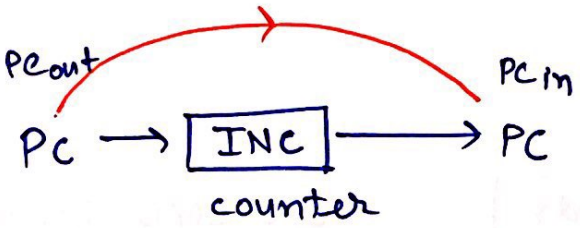
sequence of control signals

Micro - Program :-

Common M-prog used in the inst^{n} execution is,

(i) Instruction - Fetch (IF) :-





M-Prog

T_1 : PC $\rightarrow$ MAR ; PC_{out} , MAR_{in}

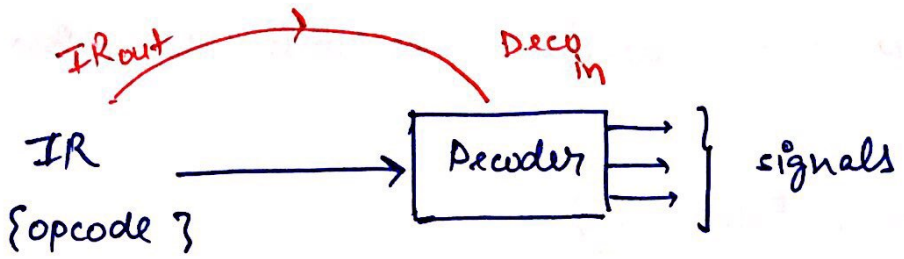
T_2 : M[MAR] $\rightarrow$ MBR ; MAR_{out} , $MBRSB_{in}$ } SYS BUS

PC + I $\rightarrow$ PC ; PC_{out} , INC , PC_{in} } LOCAL BUS

(constant)

T_3 : MBR $\rightarrow$ IR ; MBR_{out} , IR_{in}

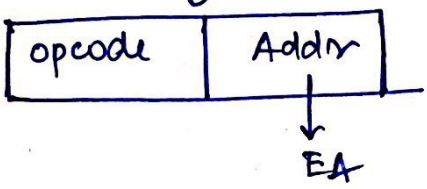
(ii) Instruction - Decode (ID) :-



T_1 : IR → Decoder ; IR_{out} , $Decod_{in}$

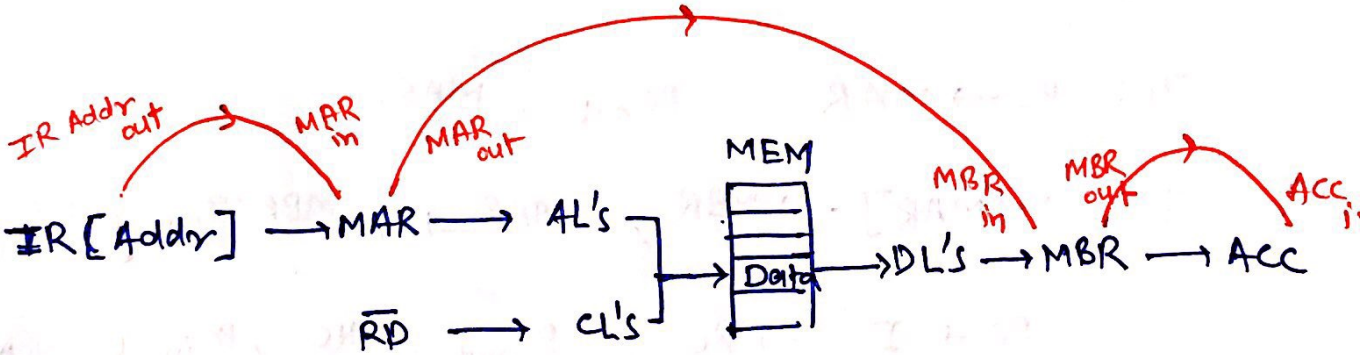
(iii) Operand Fetch (OF) :-

(a) Direct Addressing



eg:- LOAD [2000]

Acc ← M[2000]



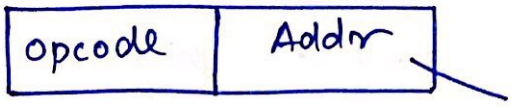
Mprog :-

T_1 : IR[Addr] → MAR ; $IR\ Addr_{out}$, MAR_{in}

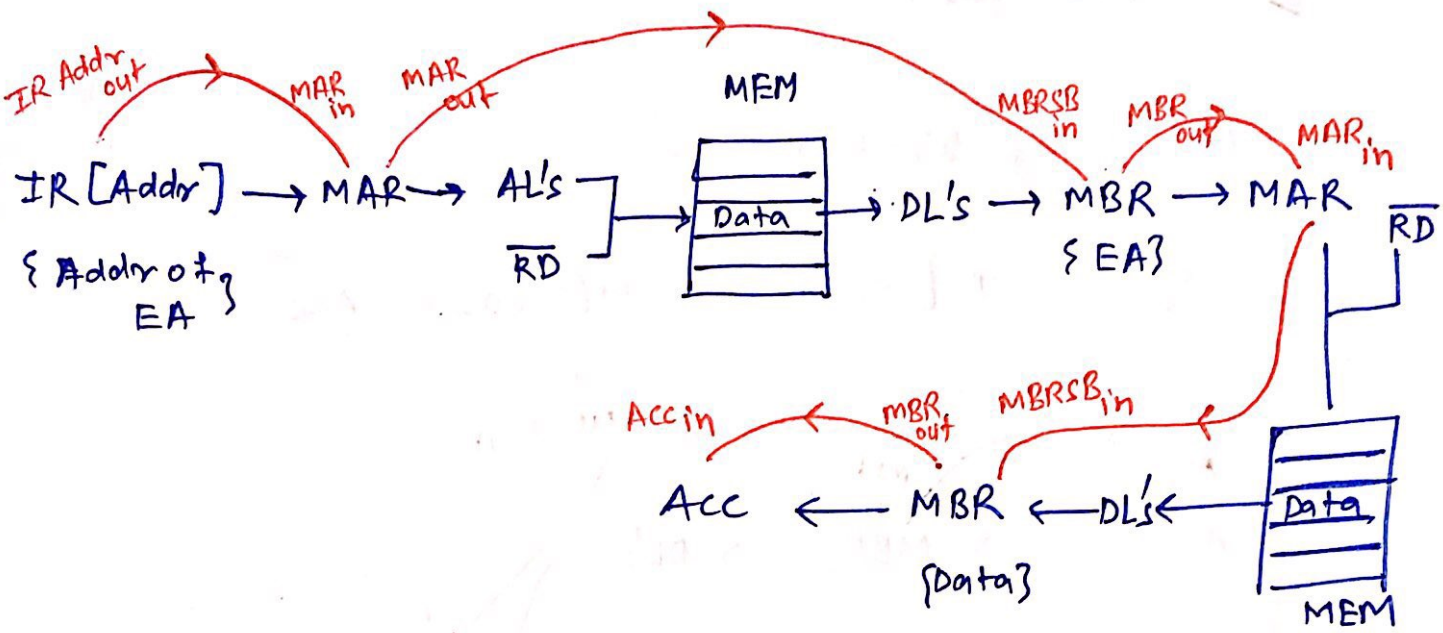
T_2 : M[MAR] → MBR ; MAR_{out} , $MBRSB_{in}$

T_2 : MBR → ACC ; MBR_{out} , ACC_{in}

(b) Indirect Mode :-



Eg :- LOAD @2000 ; → ACC ← M[[2000]]



M-prog :-

T_1 : IR[Addr] → MAR ; $IR_{Addr\ out}$, MAR_{in}

T_2 : M[MAR] → MBR ; MAR_{out} , $MBRSB_{in}$

T_3 : MBR → MAR ; MBR_{out} , MAR_{in}

T_4 : M[MAR] → MBR ; MAR_{out} , $MBRSB_{in}$

T_5 : MBR → ACC ; MBR_{out} , ACC_{in}

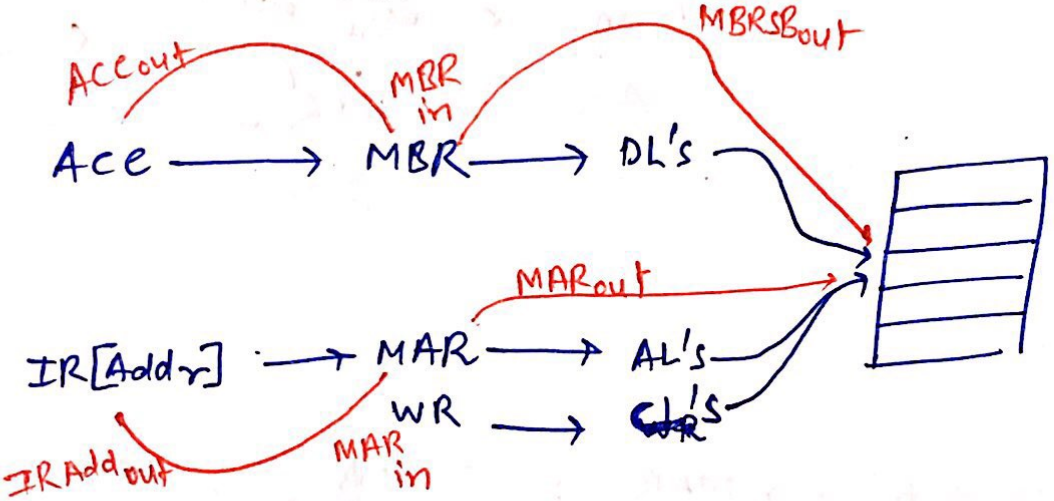
(iv) **Process Data** :- μ-prog varies depends on opcode

so, it is not a common μprog.

(v) **Write Back** :-

(a) Using **Direct Mode** -

eg :- STA [2000] ; M[2000] ← ACC



T_1 : ACC → MBR ; ACC_{out}, MBR_{in}

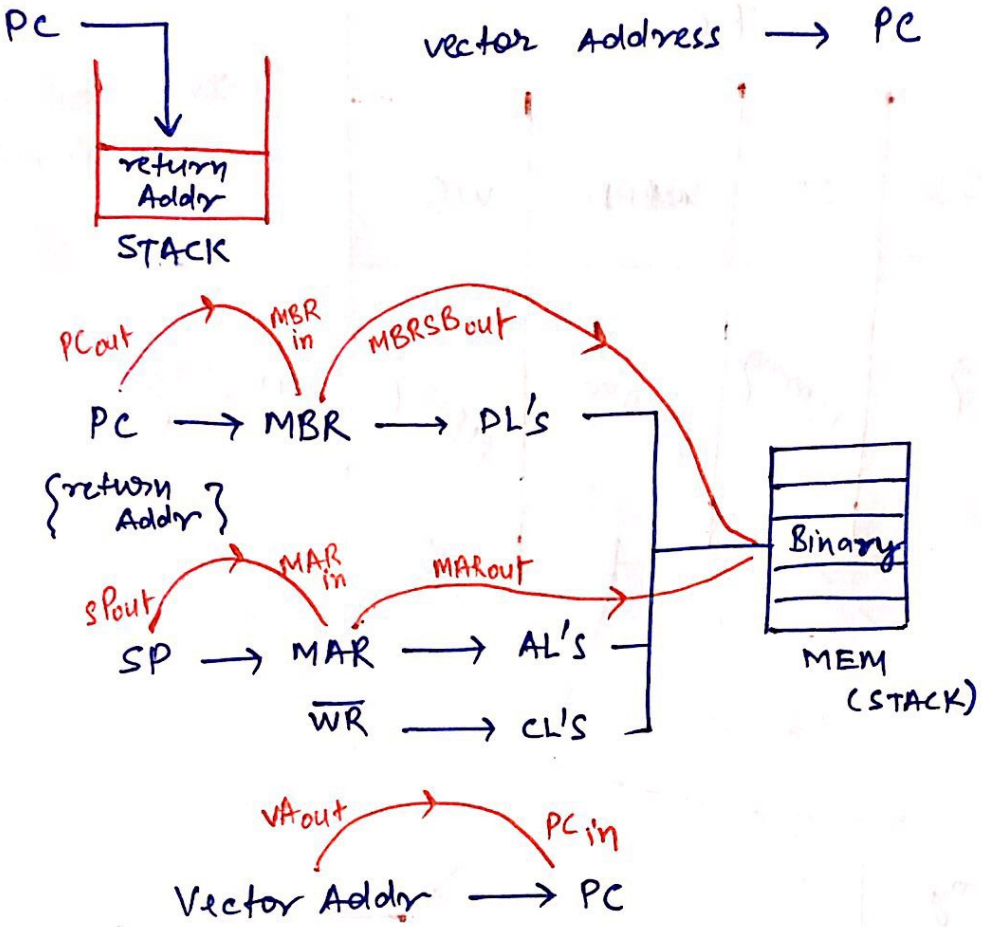
T_2 : IR[Addr] → MAR ; $IR\ Addr_{out}$, MAR_{in}

T_3 : MBR → M[MAR] ; $MBRSB_{out}$, MAR_{out}

(M[MAR] = Mem)

T_1 & T_2 can be interchanged

(vi) Interrupt Subprogram :-



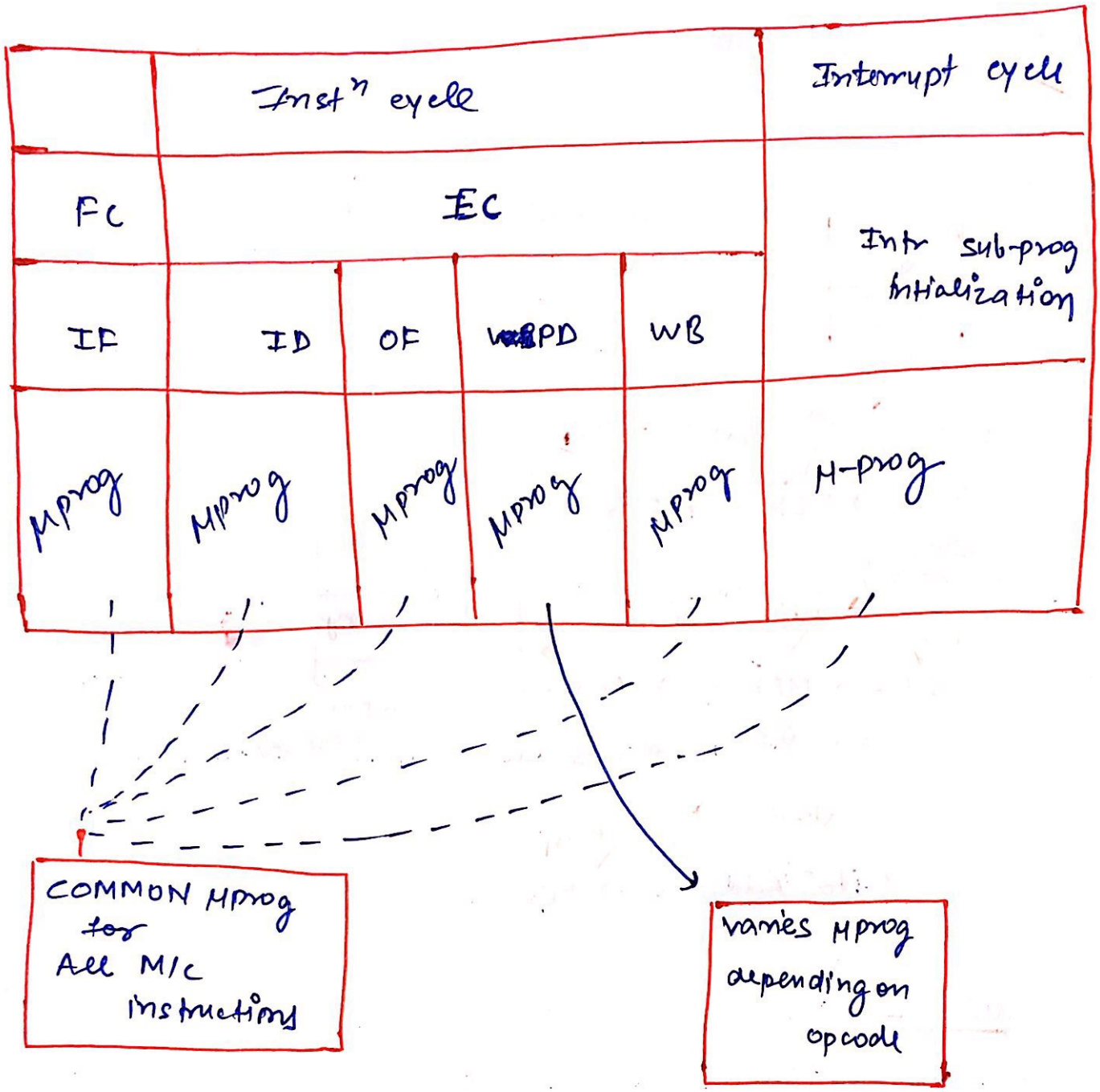
M-Prog -

T_1 : PC → MBR ; PC_{out} , MBR_{in}

T_2 : SP → MAR ; SP_{out} , MAR_{in}

T_3 : MBR → M[MAR] (SYS-BUS) ; $MBRSB_{out}$, MAR_{out}

Vector Addr → PC (LOCAL BUS) ; VA_{out} , PC_{in}



Q. Consider the following micro-program:

T_1: PC → MBR

T_2: X → MAR

T_3: MBR → memory

Y → PC

which one the following is satisfy by above Mprog ?

(a) IF

(b) OF

(c) cond JMP

✓(d) Interupt

↳ Control Unit Design :-

Pre-requirements :-

↳ How many control signals are present in Basic H/W

↳ How many inst^n are implemented in Basic H/W

↳ How many M-op^n are required for each inst^n

↳ what are the control signals required for each M-op^n, for each instruction.

After finalizing the above requirements CU is implemented using the following approaches :

↳ Hardwired - approach

↳ Micro-programmed approach

Hardwired CU :-

- In this design control signal is expressed in a sum of product expression format.
- Control expression is directly realized by the independent hardware.
- It is a fastest CU.
- It is used in Real-time application.
- Even minor modification requires, redesign & reconnection of a control signal, hence it is not flexible.
- It is not suitable in the design & testing places.
- RISC control unit is hardwired control unit.

Sample CU:-

(i) control signal's in H/W = $\{ S_0, S_1, S_2, S_3 \}$

(ii) Inst^n in H/W = $\{ I_1, I_2, I_3 \}$
$\downarrow \quad \downarrow \quad \downarrow$
$+ \quad * \quad /$

opcode = $\log_2 3$ = 2-bit

00 — undefined
01 — I_1 (+)
10 — I_2 (*)
11 — I_3 (/)

(iii) no. of M_{op}^{n} per $inst^{n}$ = { T_1, T_2, T_3, T_4 }

i.e All $inst^{n}$ (I_1, I_2, I_3) takes 4-cycles

(iv)

$M\text{-}op^{n}$ / $Inst^{n}$	I_1	I_2	I_3
T_1	S_0, S_1, S_2	S_0, S_1	S_1, S_2, S_3
T_2	S_0, S_1	S_2, S_3	S_3, S_1
T_3	S_0, S_2	S_2, S_1	S_3, S_0, S_2
T_4	S_3, S_0	S_3, S_2, S_1	S_0, S_3

Now in Hardwired - CU

(CS : SOP Format)

$S_0 = T_1(I_1 + I_2) + T_2(I_1) + T_3(I_1 + I_3) + T_4(I_1 + I_3)$

$S_1 = T_1(\underbrace{I_1 + I_2 + I_3}_{1}) + T_2(I_1 + I_3) + T_3(I_2) + T_4(I_2)$

$S_2 = T_1(I_1 + I_3) + T_2(I_2) + T_3(\underbrace{I_1 + I_2 + I_3}_{1}) + T_4(I_2)$

$S_3 = T_1(I_3) + T_2(I_2) + T_3(I_3) + T_4(\underbrace{I_1 + I_2 + I_3}_{1})$

Q. Consider the following control expressions used to generate CS on CU. During the execution of a I_4 instⁿ, In which μ-opⁿ CS is disabled in the H/W.

$$X_{in} = T_1 + T_2(I_2+I_3) + T_3 + T_4(I_1+I_4)$$

$$\cdot \begin{bmatrix} I_1 \\ I_2 \\ I_3 \\ (I_4) \end{bmatrix} \quad \cdot \begin{bmatrix} I_2 \\ I_3 \end{bmatrix} \quad \begin{bmatrix} I_1 \\ I_2 \\ I_3 \\ (I_4) \end{bmatrix} \quad \begin{bmatrix} I_1 \\ (I_4) \end{bmatrix} \cdot$$

I_4 is not present in Cycle-2 or T_2 μ-opⁿ

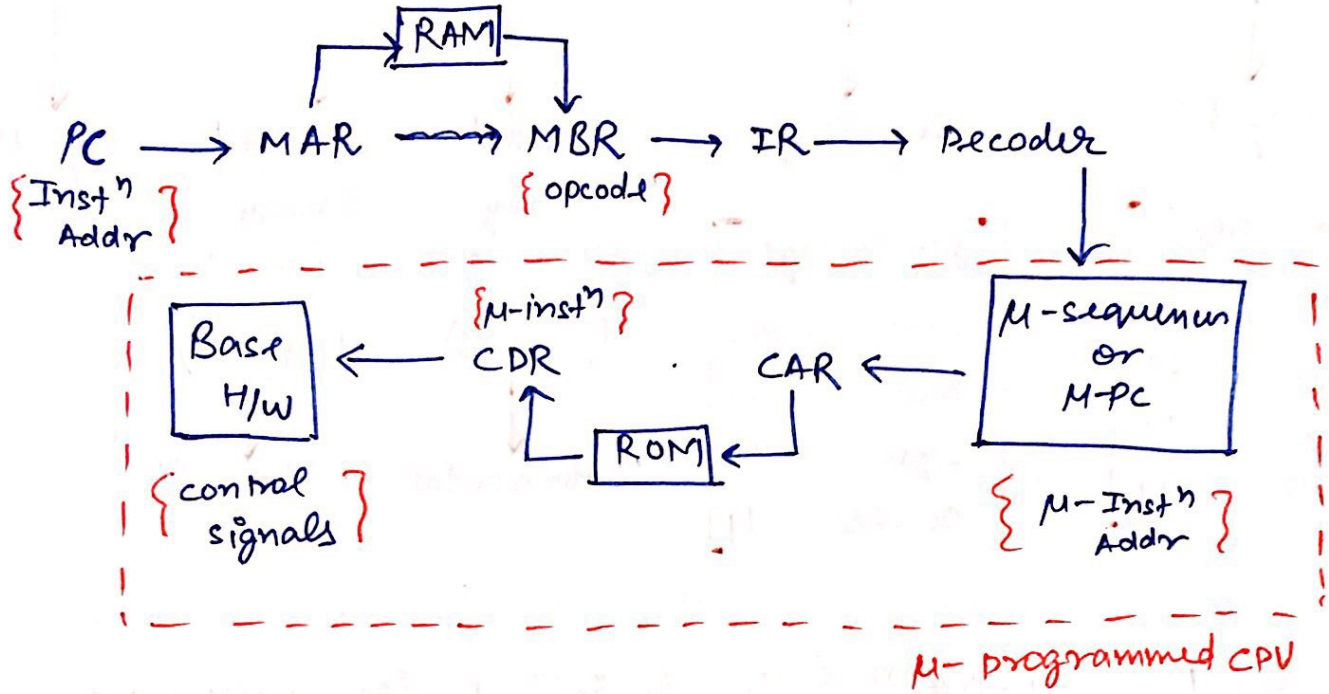
So, CS of I_4 is disabled in T_2

Micro-programmed CU :-

- In this design control memory is programmed with μ-programs.
- Control memory is a permanent memory i.e ROM
- In this design micro-sequencer unit is present used to generate the μ-instⁿ address in a sequence.

- Control memory is associate with a CAR and CDR register used to hold the control memory address & content respectively.

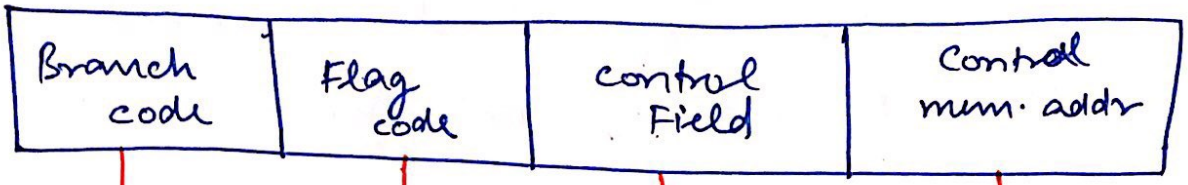
- Binding Process is



- In the control memory, μ-instⁿ is stored in the following format :

μ-instⁿ ⟶ combination of control signals

$(S_0, S_1, S_2, \cdots)$



Branch logic → $\log_2$ BC

[eg:- 4BC's
C - 00
NC - 01
Z - 10
NZ - 11]

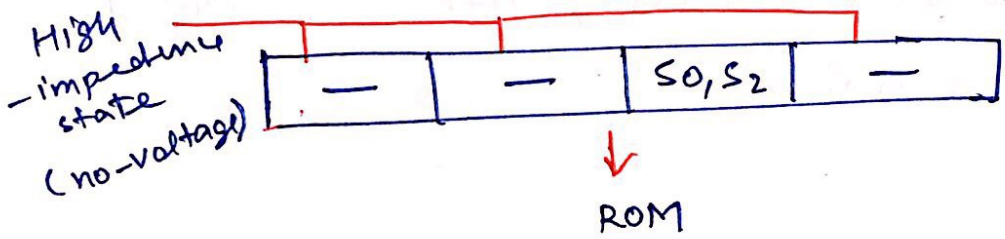
Flag Information → $\log_2$ flags

[eg:- 4Flags
carry - 00
Zero - 01
sign - 10
Overflow - 11]

control signals → Decoded Binary Format (1bit/cs) → Horizontal
control signals → Encoded Binary format ($\log_2 N$ bit) → Vertical

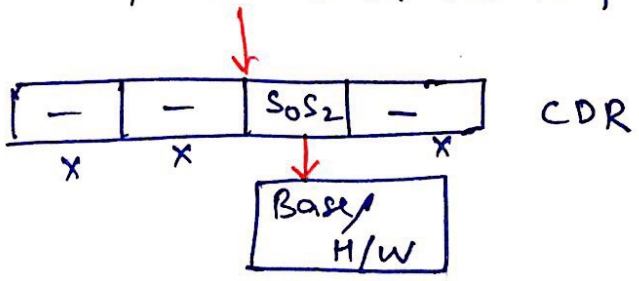
Next μ-inst^n addr → $\log_2$ CM size

Eg-1: μ-inst^n { T_1 : S_0, S_2 } Design without branch Logic -



operational state

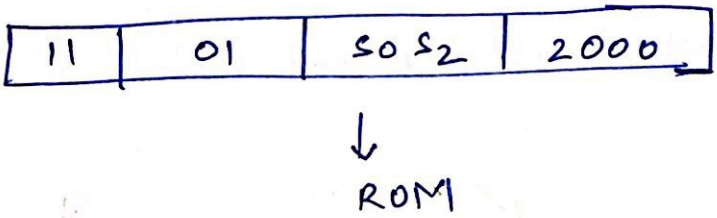
Fetch the μ-inst^n from the ROM



no branch logic given

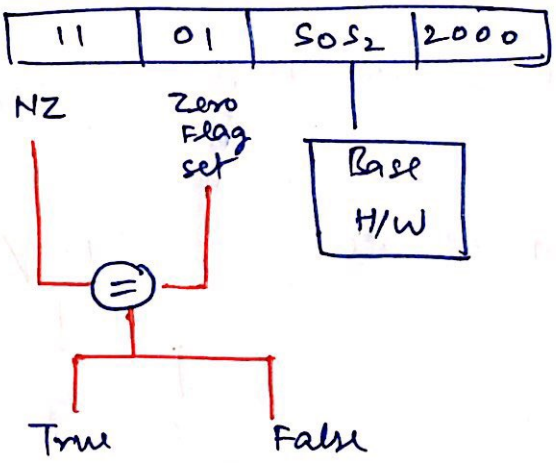
M-sequencer unit → CAR

Eg-2: M-inst^n { T_1: S_0, S_2 } design with 'NZ' branch logic



operational state

Fetch the M-inst^n from the ROM



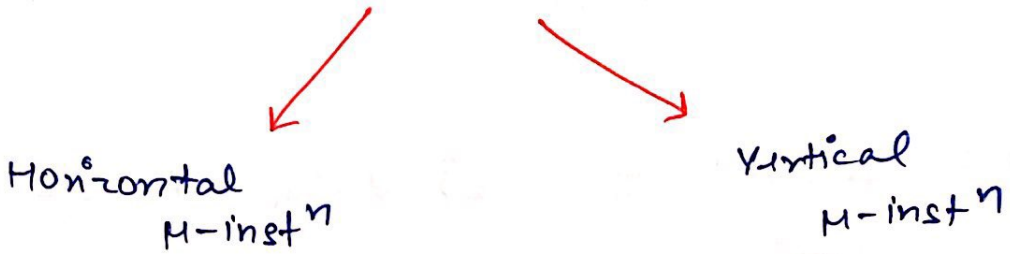
2000 → M-seq

MSeq → CAR

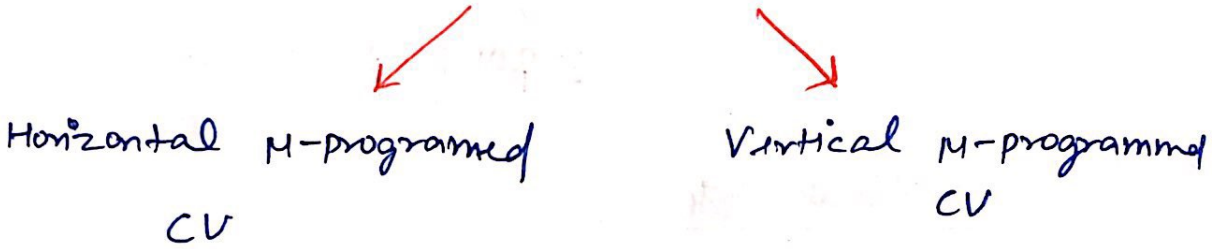
No change in M-seq

MSeq → CAR

- Based on the style of representing the control signal μ-instⁿ is of two kinds :



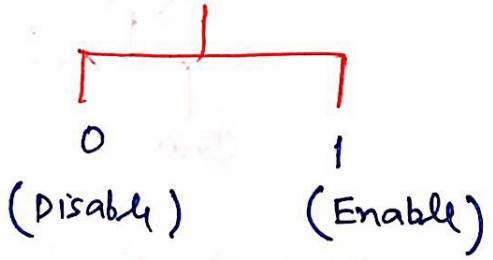
- Based on the kind of μ-instⁿ program in the control memory (CM), control unit (CU) is of two types :



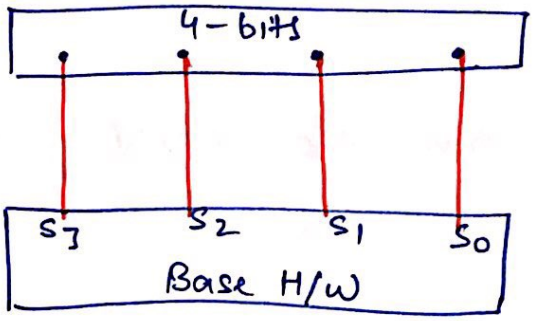
Horizontal μ-programmed CPU :-

① control signal (CS) in the H/W = $\{S_0, S_1, S_2, S_3\}$

② Decoded form of CS = 1-bit / CS



control field of CW (control word)



(3) M-Instⁿ design -

I_1 : $T_1 = \{ S_0, S_1, S_2 \}$

–	–	0111	–

$T_2 = \{ S_0, S_1 \}$

–	–	0011	–

$T_3 = \{ S_0, S_2 \}$

–	–	0101	–

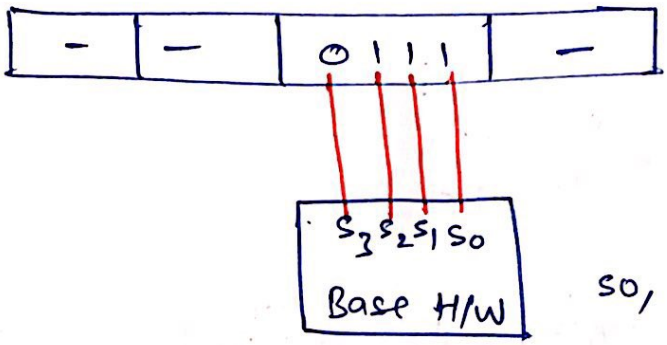
$T_4 = \{ S_0, S_4 \}$

–	–	1001	–

Stored into ROM

④ operational state -

Fetch the T_1 from the ROM (control memory)



No- branch logic

M-seq unit ⟶ CAR

Then T_2 will be fetch and so on -

Vertical μ-programmed CU :-

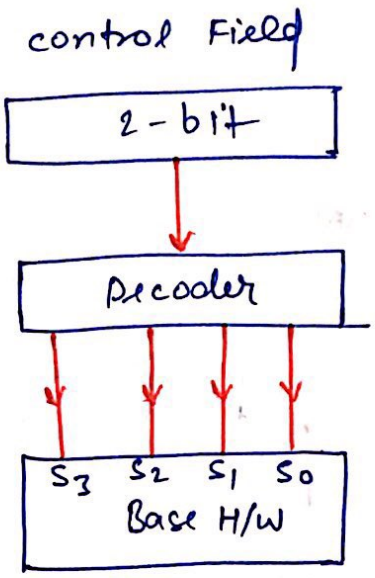
① CS in the H/W : $\{ S_0, S_1, S_2, S_3 \}$

② Encoded form of a CS = $\log_2 4$ = 2-bits / CS

↓

Function code (FC)

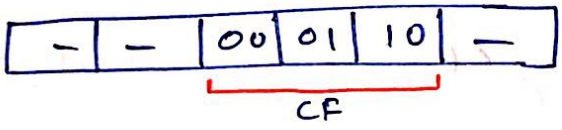
FC	CS
00	S_0
01	S_1
10	S_2
11	S_3



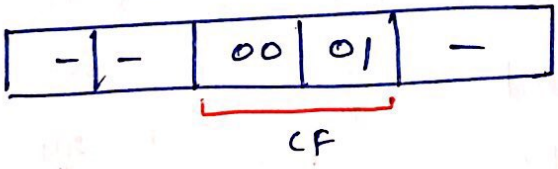
(3) M-Instⁿ Design uses multiple func codes in the control field to represent the multiple signals.

I_1 :

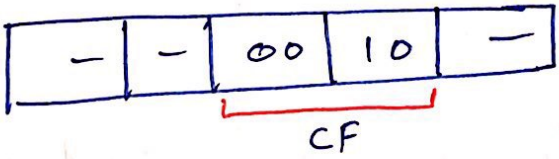
$T_1 = \{S_0, S_1, S_2\}$



$T_2 = \{S_0, S_1\}$



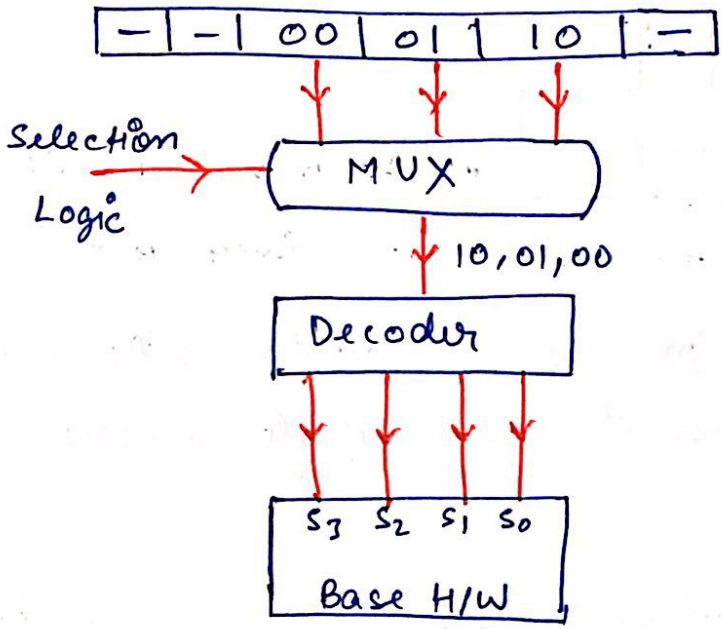
$T_3 = \{S_0, S_2\}$



Stored in ROM

④ Operational state

T_1: Fetch T_1 from the ROM



Horizontal μ-prog CU VS Vertical μ-prog CU

Horizontal μ-prog CU	Vertical μ-prog CU
- In this design control signal expressed in decoded binary format i.e 1-bit/CS	- In this design, CS is expressed in a encoded binary format. i.e - $\log_2 N$.
- It support longer control word	- It support shorter CW.
- No - need of external decoders to generate the CS, so it is faster than vertical version	- Need of a external Decoders to generate CS, so it is slower then horizontal

- It allow high degree of parallelizn i.e (0 or more than 0)
- It is a bit flexible compared with hard-wired control unit.
- It is very complex to implement therefore this concept is not in practise

- It allows low degree of parallelism i.e (0 or 1)
- It is more flexible
- It is used in CISC computers ∴ default μ-programmed CU is vertical-control unit.

NOTE:- Ascending order of a CU design in term of

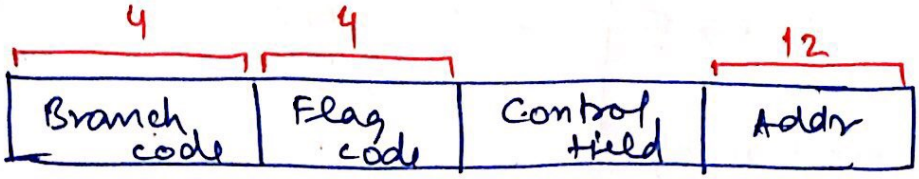
(a) speed (Vertical < Horizontal < Hardwired)

(b) Flexibility (Hardwired < Horizontal < Vertical)

Q. Consider a hypothetical CU which support 4K CW memory. Hardware contain 48 CS's, 16 Flag's and uses 16-branch condition. What is the size of a CW in bits, and control memory in bytes using -

(a) Horizontal μ-prog CU

(b) Vertical μ-prog CU



$\log_2 16$ = 4 bits

$\log_2 16$ = 4-bits

$\log_2 4K$ = 12-bits

Since size of control field depends upon type ie Horizontal or vertical config –

Horizontal

no. of bits (CF) = CS's in H/W

CF = 48-bits

so, CW size = 4+4+48+12

= 68-bits

code-mem size = 4K (CW)

= 4K × 68

= $\left(\frac{4K \times 68}{8}\right)$ bytes

= 34KB

Vertical

no. of bits (CF) = $\log_2$(CS's in H/W)

= $\log_2 48$

CF = 6-bits

CW size = 4+4+6+12

= 26-bits

code-mem size = 4K (CW)

= 4K × 26

= $\left(\frac{4K \times 26}{8}\right)$ Byts

= 13KB

NOTE:- In all the above cases we restricted the no. of cs per μ-inst to 1 i.e

1 cs / μ-inst for both Horizontal and vertical config

Q. A μ-prog cu support 400 cs's. μ-inst^n is designed with 5 cs's what is the size of μ-inst^n when the instruction is designed without branch logic.

Default μ-prog cu is vertical cu

so, size of function codes = $\log_2 400$

= 9 bit/cs

and size of μ-inst^n (CW) = 9 × 5

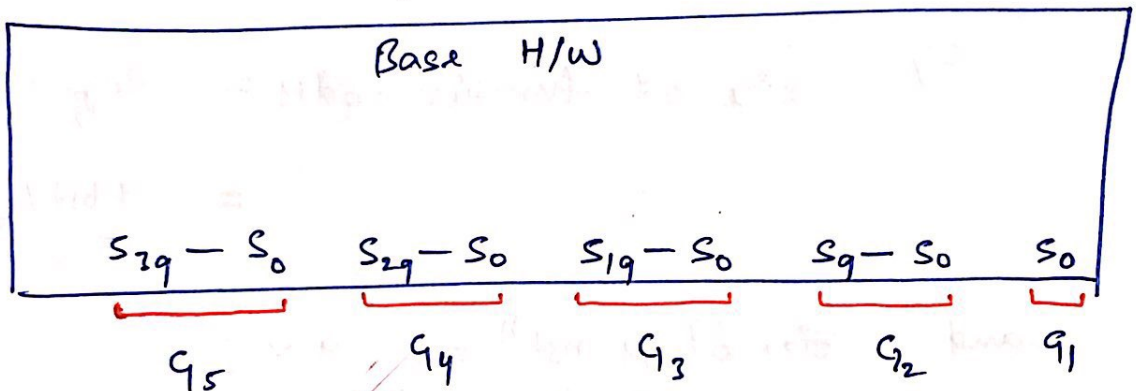
= 45-bits

as 5 cs' will be represented by 9-bits.

Q. Consider a hypothetical CU which supports 5-groups of a mutually exclusive CS's described below.

Group	G_1	G_2	G_3	G_4	G_5
CS	1	10	20	30	40

How many bits are saved using vertical over horizontal programming.



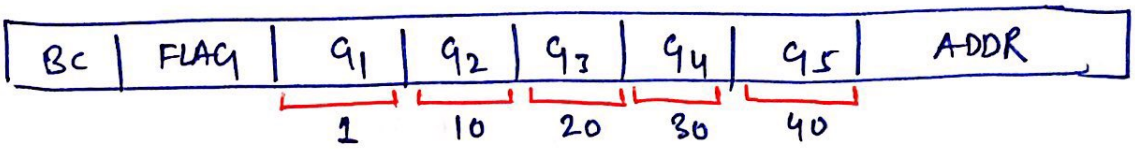
H : 40-bits 30-bits 20-bits 10-bits 1-bits

V : $\log_2 40$ $\log_2 30$ $\log_2 20$ $\log_2 10$ $\log_2 1$

↓ ↓ ↓ ↓ ↓

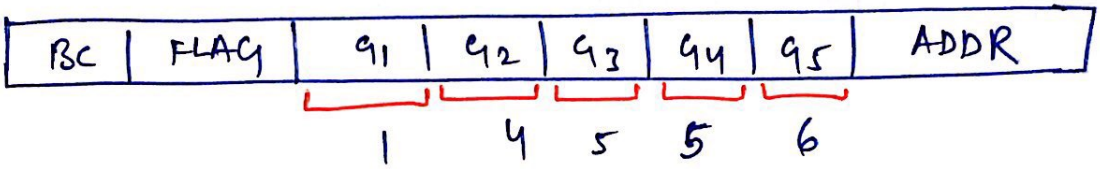
6 5 5 4 1-bit

Horizontal Design



(HCW = 101-bit)

Vertical Design



VCW = 21-bits

(bits saved = 101 - 21 = 80-bits)

Q. Consider a hypothetical computer which supports 120 instⁿ. Each instⁿ takes 10 cycles to complete. Hardware contains 200 CS's, 16-Flags, 32-branch-conditions. CPU uses Horizontal μ-programming to represent CS. What is the size of a CAR and CDR registers when the μ-instⁿ uses 1-addr Format to control the branch-logic.

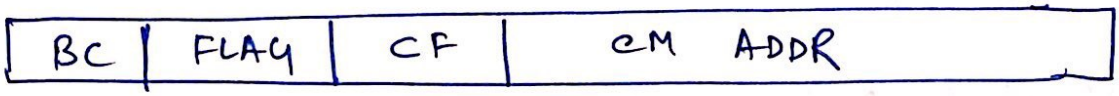
Horizontal CW

CS' in H/W = 200

decoded form of CS = 1 bit/CS

= 200-bits

M-Inst^n design with default 'ICS'



$\log_2 32$ $\log_2 16$ CS's 11-bits

5-bit 4-bit 200-bits

Inst^n in the CPU = 120

cycle / Inst = 10

i.e Mop^n/inst = 10

Total no. of Mop^n = 120 X 10

= 1200

↓

Programmed into ROM

So, Addr size = $\log_2 1200$

= (11-bits)